A Framework for Off-Line Conformance Testing of Timed Connectors

Shaodong Li[*], Xiaohong Chen[†], Yiwu Wang[*] and Meng Sun[*]
[*]LMAM & DI, School of Mathematical Sciences,
Peking University, Beijing, China
lisdpku@gmail.com, yiwuwang@126.com, sunmeng@math.pku.edu.cn
[†]Center for Software Engineering, Faculty of Computing, Engineering and The Built Environment,
Birmingham City University, United Kingtom
xiaohong.chen@mail.bcu.ac.uk

***Abstract*—Coordination is playing a key role in complex cyber-physical systems (CPSs). The complexity and importance of coordination models and languages for CPSs necessarily lead to a higher relevance of testing during development of CPSs. Model-based testing is a promising technology to test the conformance or non-conformance relation between the implementation-under-test (IUT) and its specification. In this paper, we present an approach to test the conformance relation tioco$_c$(Timed Input-Output Conformance) between the implementation of a timed Reo connector and its specification given by a timed constraint automaton (TCA). An algorithm to generate test cases from a TCA is proposed and the testing approach is implemented in UPPAAL.**

I. Introduction

Coordination is playing a key role in development of complex cyber-physical systems that integrate computing and communication with monitoring and controlling of physical entities, and is extremely important for their success. Currently, the coordination mechanisms in CPSs are usually ad hoc, error prone and inefficient. This is unacceptable in many CPSs since it may result in expensive costs or even fatal side-effects due to the erroneous or delayed behavior. Compositional coordination models and languages provide a formalization of the “glue code” that interconnects different components and organizes their communication and interaction in a distributed processing environment. They support development of large-scale CPSs by allowing construction of complex connectors for coordination out of simpler ones. As an example, Reo [5], [8] offers an attractive and powerful glue language that is able to express interaction protocols as connectors. Primitive connectors called *channels* in Reo, such as synchronous channels and FIFO channels, can be composed to build circuit-like connectors which serve as the glue code to exogenously coordinate the behavior of components in complex systems.

The complexity and importance of coordination models and languages for CPSs necessarily leads to a higher relevance of testing issues for connectors during development of CPSs. Model-based testing is a promising technology for automatic and systematic test case generation to show the conformance or non-conformance between the implementation-under-test (IUT) and its model that specifies its expected behavior. The model of the IUT serves to generate the input stimuli and as a test oracle for its expected behavior. Test cases are generated from the model and executed on the IUT. If all possible tests pass, then we have the conformance between the IUT and its model. One popular approach to check the conformance relation between the IUT and its model is the **ioco** testing theory (Input-Output Conformance), which uses labeled transition systems as models, and a formal implementation relation called **ioco** is used to show when an IUT is correct with respect to its model [22].

In this paper, we present a model-based testing approach to automatically derive tests from timed Reo connector specifications, which are given as timed constraint automata extended with inputs and outputs. Testing of connectors in Reo was first investigated in [2] where both connector specifications and their implementations are represented as designs, and implemented in Maude. Later another approach of testing Reo connectors based on the classical **ioco** testing theory was introduced in [15]. Both of the two approaches only involve testing the untimed connectors while timing properties are not considered. However, for cyber-physical systems which involve both discrete and continuous dynamics, the correct system behavior usually depends on real-time properties, for which a timed implementation relation is required.

Timed input-output conformance (**tioco**) theory has been proposed in [16], [17] for conformance testing of real-time systems. The approach proposed in this paper is based on the **tioco** theory. There are two typical models in **tioco** theory: *analog-clock* and *digital-clock*. Although analog-clock tests can observe the delays between two events precisely, they are difficult (if not impossible) to implement in most existing tools, and the semantics of both specification and implementation are more complex. In digital-clock tests, a short passage of time is sampled as a $tick$ action, which is regarded as part of the behavior of the whole system, in the same way as an input or output action does. Therefore the digital-clock test approach is easier to be implemented in a state-based model checking toolset, such as mCRL2.

The testing approach in this paper has been implemented in UPPAAL, which is an integrated tool environment for modeling, validation and verification of real-time systems modeled

978-1-4673-7612-9/15 $31.00 © 2015 IEEE
DOI 10.1109/TASE.2015.9

as networks of timed automata [9]. The toolset can be used for modeling, validation and verification of concurrent timed systems and protocols. As in Reo, systems in UPPAAL are constructed in a compositional way. Given a connector in Reo, an equivalent automata model can be automatically generated for verification with the UPPAAL toolset. Depending on the expressiveness need for the verification of specific properties, the data and time parameters could be omitted.

The remainder of this paper is structured as follows: In Section II we briefly summarize the coordination language Reo and its semantical model of timed constraint automata (TCA). Then in Section III, we present our definition of the timed input-output conformance relation $\mathbf{tioco}_c$ for testing connectors in Reo. In Section IV, we present the approach for testing the $\mathbf{tioco}_c$ conformance relation between a Reo connector and a specification given by a TCA. In Section V we discuss the implementation in UPPAAL for testing timed connectors. In Section VI, we present related works and compare them with our approach. Finally, Section VII concludes with some further research directions.

II. The Reo Coordination Language

In this section we provide a brief introduction to the coordination language Reo, together with its semantical model and composition operators.

A. Basic concepts

Reo [5] is a channel-based exogenous coordination language wherein complex coordinators, called *connectors*, are compositionally constructed from simpler ones. We summarize only the main concepts in Reo here. Further details about Reo can be found in [5], [8].

Components can dynamically compose, connect to, and perform I/O operation through connectors in Reo. Primitive connectors are *channels*. A channel is a primitive communication medium with two *ends*. A channel end can be either a *source* end through which data enter the channel or a *sink* end through which data leave the channel. Although channels can be defined by users, Reo provides a set of basic channels with predefined behaviors, which suffices to implement rather complex interaction protocols. Among these channels (Fig. 1) are:

- the *sync channel*, which is a directed channel that accepts a datum through its source end if and only if it can instantly dispense the datum through its sink end;
- the *lossysync channel*, which accepts a datum through its source end and tries to instantly dispense it though the sink end and if this is not possible, the data item is lost;
- the *syncdrain channel*, which is a channel with two source ends that accept data simultaneously and loses them subsequently;
- the *FIFO1 channel*, which is an asynchronous channel with a buffer of capacity one;
- the *t-timer channel*, which accepts any input value d via its source end and returns a "timeout" signal through its sink end after a delay of exactly t time units.

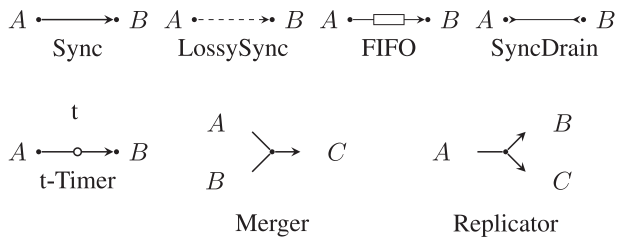

Fig. 1. Basic Channels and Nodes of Reo

Channels are joined together in nodes. The set of channel ends coincident on a node is disjointly partitioned into the sets of source and sink channel ends that coincide on the node, respectively. Nodes are categorized into source, sink and mixed nodes, depending on whether all channel ends that coincide on a node are source ends, sink ends or a combination of the two. A source node merges data from multiple channels that are connected to it (Merger in Fig. 1), while a sink node copies the inputs it receives and send them through the source ends that are connected to it simultaneously (Replicator in Fig. 1). The hiding operation is used to hide the internal topology of a component connector. Mixed nodes are hidden and can no longer be accessed or observed from outside.

Example 1 (FIFO with a lower time bound)**.** Figure 2 shows a FIFO buffer with a lower time bound of t. This Reo connector consists of four nodes A, B, C, D and four channels:

- two FIFO1 channels between nodes A and B, and nodes C and D;
- one t-timer channel between node A and C;
- and one syncdrain channel between node B and D.

The connector coordinates two components: a *Writer* that is linked to node A and a *Reader* that is linked to node B. Each datum that is received by the connector from the Writer via node A will stay in the upside buffer for at least t time units before being dispensed through node B.

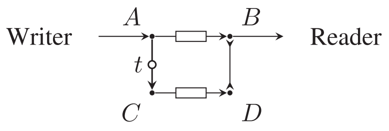

Fig. 2. A FIFO buffer with a lower time bound t

B. Timed constraint automata

Timed constraint automata (TCA) are a combination of constraint automata [8] and timed automata (TA) [3], [4] with local invariants. They serve as an operational semantic model for timed Reo connectors.

Let $Data$ be a finite and non-empty set of data items that can be transferred through channels and observed on nodes. Let $\mathcal{N}$ be a set of node names. A data assignment δ over a non-empty subset $N \subseteq \mathcal{N}$ is a mapping $\delta : N \to Data$

specifying the fact that the datum $\delta(A)$ is observed on node A for each $A \in N$, and no other I/O operations happen on a node outside N. A data constraint dc is a propositional formula built from the atoms of the form "$d_A \in P$" and Boolean operators $\wedge, \vee, \neg$, etc., where $A \in \mathcal{N}$, $P \subseteq Data$ and d_A is interpreted as $\delta(A)$. $\delta_\emptyset \models dc$ only when $dc = \text{true}$.

We write $DA(N)$ and $DC(N)$ to denote the set of all data assignments and data constraints on node-set $N \in \mathcal{N}$ respectively. We write DA for $\bigcup_{\emptyset \neq N \subseteq \mathcal{N}} DA(N)$ and DC for $DC(\mathcal{N})$. The projection of a data assignment δ on a node-set N is denoted as $\delta|_N \in DA(N)$ where $\delta|_N(A) = \delta(A)$ for any $A \in N$. And the projection of a data constraint dc on N is denoted as $dc|_N \in DC(N)$, which is obtained by keeping the constraints which are related to the nodes in N only.

For a set of clocks $\mathcal{C}$, a clock assignment v is a function $v : \mathcal{C} \to \mathbb{R}_{\geq 0}$ that assigns a value $v(C) \in \mathbb{R}_{\geq 0}$ to each clock $C \in \mathcal{C}$. A clock constraint cc on $\mathcal{C}$ is a conjunction of atomic constraints with form $x \bowtie n$ where $x \in \mathcal{C}$, $\bowtie \in \{<, \leq, >, \geq, =\}$ and $n \in \mathbb{N}$. The set of clock constraints is denoted as CC.

Definition 1. *(Timed constraint automata)* A TCA $\mathcal{A}$ is a tuple $(\mathcal{S}, \mathcal{C}, \mathcal{N}, \to, s_0, ic)$ consists of

- a set of states (also called locations) $\mathcal{S}$,
- a finite set of clocks $\mathcal{C}$,
- a set of node names $\mathcal{N}$,
- an initial state $s_0 \in \mathcal{S}$,
- an invariant function $ic : \mathcal{S} \to \mathcal{CC}$ that assigns each state $s \in \mathcal{S}$ a clock constraint $ic(s)$,
- a transition relation $\to \subseteq \mathcal{S} \times 2^{\mathcal{N}} \times DC \times CC \times 2^{\mathcal{C}} \times \mathcal{S}$.

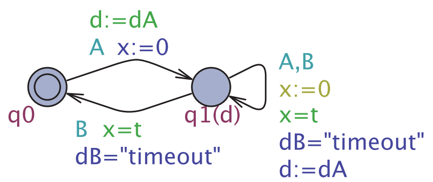

Fig. 3. Timed constraint automaton for a t-timer channel

Figure 3 shows the TCA for a t-timer channel. At the beginning, the TCA stays in the initial state represented by a double-circle node. Once it receives a datum from node A, it jumps following the above transition edge and resetting the clock x to 0. After exactly t time units when the condition $\mathsf{x = t}$ is satisfied, the TCA may jump back to the initial state if it dispenses the datum from node B at the time point. The self-loop indicates another possibility. After t time units when the channel receives a datum from node A, it may dispenses the datum from node B and at the same time, receives a new datum from node A. Under this circumstances the two I/O operations simultaneously happen on both node A and B.

In Fig. 3 we use a simpler notation for TCA. We use a parameter d to denote the datum that is received on node A, and prevent listing all the conditions. An edge labeled with $d := d_A$ of a *parametric* TCA represents a set of edges labeled with data constraints $d_A = d$ where d could be any possible value in $Data$.

C. Compositionality

Complex connectors in Reo are composed from simpler ones by the *join* operator, which is compositional, i.e., when composing several connectors, the order does not matter.

Definition 2 (The *join* operator of TCA)**.** Let $\mathcal{A}_i = (Q_i, C_i, \mathcal{N}_i, \to_i, q_{0,i}, ic_i)$ for $i \in \{1, 2\}$ be two TCA with disjoint clock sets, i.e., $C_1 \cap C_2 = \emptyset$. The join of $\mathcal{A}_1$ and $\mathcal{A}_2$ is a product TCA $\mathcal{A}_1 \bowtie \mathcal{A}_2 = (Q_1 \times Q_2, C_1 \cup C_2, \mathcal{N}_1 \cup \mathcal{N}_2, \to, ic)$ where ic is given by $ic(\langle q_1, q_2 \rangle) = ic_1(q_1) \wedge ic_2(q_2)$, and the edge relation $\to$ is obtained through the following rules:

(1) If there exists a pair of edges from $\mathcal{A}_1$ and $\mathcal{A}_2$ with the same common nodes, then an edge is introduced in $\mathcal{A}_1 \bowtie \mathcal{A}_2$:

$$\frac{\begin{array}{c}(q_1, N_1, dc_1, cc_1, C_1, q_1') \in \to_1, \\ (q_2, N_2, dc_2, cc_2, C_2, q_2') \in \to_2, \\ N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1, dc_1 \wedge dc_2 \not\equiv \text{false}\end{array}}{(\langle q_1, q_2 \rangle, N_1 \cup N_2, dc_1 \wedge dc_2, cc_1 \wedge cc_2, C_1 \cup C_2, \langle q_1', q_2' \rangle) \in \to'}.$$

(2) If there exists an edge in $\mathcal{A}_1$ on which the node set contains only local nodes (instead of common nodes with $\mathcal{A}_2$), then an edge is introduced in $\mathcal{A}_1 \bowtie \mathcal{A}_2$:

$$\frac{(q_1, N_1, dc_1, cc_1, C_1, q_1') \in \to_1, N_1 \cap \mathcal{N}_2 = \emptyset, q_2 \in Q_2}{(\langle q_1, q_2 \rangle, N_1, dc_1, cc_1, C_1, \langle q_1', q_2 \rangle \in \to')}.$$

Symmetrically, we have a similar rule for $\mathcal{A}_2$.

Notice the latter two symmetric rules also work on transitions with an empty node set.

We refer the compositionality of *join* operator to [6], [8].

III. Timed Input-Output Conformance Relation

The **tioco** relation for classic timed automata is proposed in [16], [17], which specifies a model-based conformance theory for checking *timed input-output conformance* relation between implementations and specifications. However, it is not suitable for testing timed Reo connectors. This is mainly because inputs and outputs operations may happen simultaneously on a timed Reo connector, which makes it difficult to decide the expected, unexpected and irrelevant behaviors of an implementation against a specification. Therefore we modify the **tioco** relation for classic TA models and propose a timed input-output conformance relation (**tioco**$_c$) for timed constraint automata (c represents that it is an extension in constraint automata).

In the following, we will use $\mathcal{N}_{in}$ to denote the set of all source nodes of a connector, and $\mathcal{N}_{out}$ the set of all sink nodes. All internal mixed nodes are hidden using *hide* operator. In the reset of this paper, for a node-set N and a data constraint $dc \in DC(N)$, we denote $N_{in} = N \cap \mathcal{N}_{in}$ as the input part of node set N, and $dc_{in} = dc|_{N_{in}}$ for the projection of dc on N_{in}. Similarly we have the output part of node set N_{out} and dc_{out}.

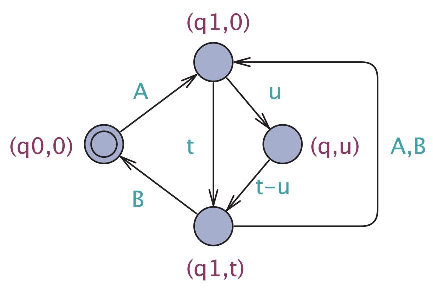

Fig. 4. Timed labeled transition system for FIFO with a lower time bound

A. *Timed labeled transition systems*

Timed constraint automata is a high-level model for Reo connectors. To formalize and analyze the behavior of Reo connectors, we turn to *timed labeled transition systems* (TLTS).

Definition 3. *(Timed labeled transition system)* A TCA $\mathcal{A} = (Q, C, \mathcal{N}, \rightarrow, q_0, ic)$ derives a TLTS $\mathcal{L}_{\mathcal{A}} = (S, s_0, \mathcal{N}, T_d, T_t)$ where

- the state set S consists of pairs (q, v) of a location $q \in Q$ and a clock assignment v;
- an initial state $s_0 = (q_0, \vec{0})$ where $\vec{0}$ is the clock assignment that set all clocks to 0;
- the node set N is the same with TCA $\mathcal{A}$;
- a discrete transition relation T_d and a timed transition relation T_t.

The transition $(q, v) \xrightarrow{(N,\delta)} (q', v')$ is a discrete transition in T_d, if there is an edge $(q, N, dc, cc, X, q') \in \rightarrow$, such that $\delta \vDash dc$, $v \in cc$ and $v' \in ic(q')$ where $v' = v[X := 0]$ is obtained by restting all clocks in X to zero and leaving the others unchanged; The transition $(q, v) \xrightarrow{t} (q, v + t) \in T_t$ is a timed transition in T_t, if $v \in ic(q)$ and $v + \Delta t \in ic(q)$ for any Δt with $0 < \Delta t \le t$.

Figure 4 shows the TLTS for a FIFO channel with a lower time bound t. Notice there are infinitely many timed transitions that lie between state $(q_1, 0)$ and (q_1, t). In fact, for any $u, v \in (0, 1)$ with $u < v$, transition $(q_1, u) \rightarrow (q_1, v)$ is a time transition.

An *event* is either an I/O operation (N, δ) or a time elapse $t > 0$. The set of all events of a TLTS is $\mathsf{Event} = (\mathcal{N} \times DA(\mathcal{N})) \cup \mathbb{R}_{>0}$. In particular, there is an empty event $(\emptyset, \emptyset)$ in Event. For any state s, $s' \in S$ and any event $\mu \in \mathsf{Event}$, we introduce the following simpler notations:

- $s \xrightarrow{\mu} s'$ iff $(s, \mu, s') \in T_d \cup T_t$;
- $s \xrightarrow{\mu}$ iff $\exists s' \in S$ such that $s \xrightarrow{\mu} s'$;
- $s \xrightarrow{\mu_1 \cdots \mu_n} s'$ iff $\exists s_1, \cdots, s_n \in S$ such that $s = s_1 \xrightarrow{\mu_1} s_2 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_n} s_n = s'$;
- $s \xrightarrow{\mu_1 \cdots \mu_n}$ iff $\exists s' \in S$ such that $s \xrightarrow{\mu_1 \cdots \mu_n} s'$.

Definition 4. *(RT-traces of TCA)* An RT-trace $\rho = \mu_1 \dots \mu_n$ is a finite sequence of events. The set of all RT-traces over action set $\mathcal{N}$ is denoted as $\mathsf{RT}(\mathcal{N})$. For a TCA $\mathcal{A}$, its RT-traces are defined as

$$\mathsf{TTraces}(\mathcal{A}) = \{\rho \in \mathsf{RT}(\mathcal{N}) \mid s_0 \xrightarrow{\rho}\}$$

and the observable RT-traces are defined as

$$\mathsf{ObsTTraces}(\mathcal{A}) = \{Obs(\rho) \mid \rho \in \mathsf{TTrace}(\mathcal{A})\},$$

where $Obs(\rho)$ is the sequence obtained by deleting all $(\emptyset, \emptyset)$, i.e., the empty node-set and the empty data assignment in the trace ρ.

The elapsed time $\mathsf{time}(\rho)$ of a trace ρ is the sum of all delays in ρ:

$$\mathsf{time}(\rho) = \begin{cases} \sum_{\mu \text{ is a time event in } \rho} \mu & \text{if } \rho \neq \epsilon \\ 0 & \text{if } \rho = \epsilon \end{cases}$$

.

The following lemma and theorem capture an interesting property of the RT-traces of two TCA and their product.

Lemma 1. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two TCA with the same node-set $\mathcal{N}$, and $\mathcal{L}_{\mathcal{A}_1}$ and $\mathcal{L}_{\mathcal{A}_2}$ be their TLTS respectively. If there is a transition $\langle s_1, s_2 \rangle \xrightarrow{\mu} \langle s_1', s_2' \rangle$ in $\mathcal{A}_1 \bowtie \mathcal{A}_2$, then

- $s_1 \xrightarrow{\mu} s_1'$ and $s_2 \xrightarrow{\mu} s_2'$ if $\mu \neq (\emptyset, \emptyset)$;
- $s_1 = s_1'$ and $s_2 \xrightarrow{\mu} s_2'$ (or symmetrically) if $\mu = (\emptyset, \emptyset)$, and vice versa.

Proof: We prove this according to the type of μ.

- If $\mu = t$ where t is a real number, suppose $s_i = (q_i, v_i)$, $i = 1, 2$. Notice that the invariant of $\langle q_1, q_2 \rangle$ is $ic(q_1) \wedge ic(q_2)$. Then t is an outgoing transition for $\langle s_1, s_2 \rangle$ if and only if $v_1 + t \in ic(q_1)$ and $v_2 + t \in ic(q_2)$, which equals to that t is an outgoing transition for both s_1 and s_2.
- If $\mu = (N, \delta)$, $N \neq \emptyset$, we can also prove the proposition based on Definition 3. Notice that this transition is generated by the edge $(\langle q_1, q_2 \rangle, N, dc_1 \wedge dc_2, cc_1 \wedge cc_2, C_1 \cup C_2, \langle q_1', q_2' \rangle) \in \rightarrow'$.
- If $\mu = (\emptyset, \emptyset)$, the result can be easily proved according to the second rule in Definition 2.

■

Theorem 1. Let $\mathcal{A}_i = (Q_i, C_i, \mathcal{N}_i, \rightarrow_i, q_{0,i}, ic_i)$ be two TCA with the same node-set, and states $s_{0,i}$ and s_i be the initial states and arbitrary states of TLTS $\mathcal{L}_{\mathcal{A}_i}$, for $i \in 1, 2$. Then

- for any RT-trace ρ of $\mathcal{A}_1 \bowtie \mathcal{A}_2$, $(s_{0,1}, s_{0,2}) \xrightarrow{\rho} (s_1, s_2)$, there exists an RT-trace ρ_i of $\mathcal{A}_i$, such that $s_{0,i} \xrightarrow{\rho_i} s_i$, and $Obs(\rho_i) = Obs(\rho)$;
- for any RT-trace ρ_i of $\mathcal{A}_i$, $s_{0,i} \xrightarrow{\rho_i} s_i$ with the same observations: $Obs(\rho_1) = Obs(\rho_2)$, there exists an RT-trace ρ of $\mathcal{A}_1 \bowtie \mathcal{A}_2$ such that $(s_{0,1}, s_{0,2}) \xrightarrow{\rho} (s_1, s_2)$ and $Obs(\rho) = Obs(\rho_1) = Obs(\rho_2)$.

Proof: Suppose $(s_{0,1}, s_{0,2}) \xrightarrow{\mu_1} (s_{1,1}, s_{1,2}) \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_n} (s_1, s_2)$. Then it can be easily proved by using Lemma 1.

■

B. ***tioco***$_c$ *relation*

A conformance relation answers whether an implementation respects its specification. Roughly speaking, an implementation conforms to a specification when the implementation is regarded as a particular subclass of the specification. For interactive systems, however, we need to distinguish between input and output events. The **tioco**$_c$ relation we propose takes both input and output events and time passages into account. A timed Reo connector $\mathcal{I}$ is said to conform to a specification (given by a TCA) $\mathcal{S}$, if for any observable trace σ of S, and any observable output μ of $\mathcal{I}$ after any trace that "matches" the observable trace σ, the observable output μ conforms to any observable outputs of S.

In order to formalize the above intuitive description, we need to answer: 1) what an observable output (of a TCA at a state) is; 2) how a TCA *matches* an observable trace; and 3) when an observable output *conforms* to other observable outputs.

The following example offers us an intuition to answer the third question.

Example 2. Suppose $\mathcal{N}_{in} = \{A, B\}$, $\mathcal{N}_{out} = \{C, D\}$, the observable events of state $s_{\mathcal{S}}$ in $\mathcal{S}$ are $\{A, C\}$, $\{C\}$ and $(0, 3]$. Then for the corresponding state $s_{\mathcal{I}}$ in $\mathcal{I}$, the node-sets $\{A, D\}$,$\{D\}$ or the real number 4 will be unexpected events, and $\{B\}$, $\{B, D\}$, $\emptyset$ should be ignored. Both the events of $s_{\mathcal{S}}$ and the ignored events will be considered conform with specification.

To make it formal, we give the following definition for *ignorance* of an event.

Definition 5. *(Ignore)* An event μ ignores a set of events E, denoted by μ **ignore** E, if μ is an empty event, or $\mu = (N, \delta)$ with

$$N_{in} \neq \emptyset$$

and

$$\nexists \mu' = (N', \delta') \in E : N'_{in} = N_{in} \wedge \delta|_N = \delta'|_N.$$

Definition 6. *(Conf)* An event μ *conforms* with a set of events E, denoted by μ **conf** E, if and only if $\mu \in E$ or μ **ignore** E.

For the second question, a TCA $\mathcal{A}$ may match an observable trace σ by executing an RT-trace ρ whose observable part is the same as σ.

$$\mathcal{A} \textbf{ after } \sigma = \{s \in S \mid \exists \rho \in \mathsf{RT}(\mathcal{N}) : s_0 \xrightarrow{\rho} s \wedge Obs(\rho) = \sigma\}.$$

For the first question, an observable event of a TCA at state s is either an invoked I/O operation, or a passage of time

$$\mathsf{elapse}(s) = \{t > 0 \mid \exists \rho \in \mathsf{RT}(\emptyset) : \mathsf{time}(\rho) = t \wedge s \xrightarrow{\rho}\}.$$

Suppose s is a state in TLTS, we denote all observable events of s as

$$\mathsf{out}(s) = \{(N, \delta) \mid s \xrightarrow{(N,\delta)}, N \neq \emptyset\} \cup \mathsf{elapse}(s)$$

and the point-wise extension

$$\mathsf{out}(S) = \bigcup_{s \in S} \mathsf{out}(s).$$

Now we are able to give a formal definition of the **tioco**$_c$ relation.

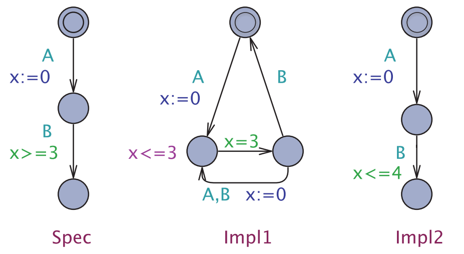

Fig. 5. Specification and Implementation of Example 1

Definition 7 (**tioco**$_c$ relation)**.** For two TCA $\mathcal{I}$ and $\mathcal{S}$, we say $\mathcal{I}$ conforms to $\mathcal{S}$ under the **tioco**$_c$ relation, denoted as $\mathcal{I}$ **tioco**$_c$ $\mathcal{S}$, if

$$\mu \textbf{ conf } \mathsf{out}(\mathcal{S} \textbf{ after } \sigma),$$

for any observable trace $\sigma \in \mathsf{ObsTTraces}(\mathcal{S})$ and any observable output $\mu \in \mathsf{out}(\mathcal{I} \textbf{ after } \sigma)$.

Figure 5 shows a specification and two implementations of Example 1. The specification requires that B must wait at least 3 time units after A happens. It also requires that the data transported through B is exactly the data receive from A. Impl1 is exactly the TCA of the *FIFO with a lower time bound*. We can state that Impl1 **tioco**$_c$ Spec. On the contrary, Impl2 is an example of a wrong implementation because the possible time gap between two discrete events A and B in Impl2 is $[0, 4]$, while the specification requires it to be greater than 3.

IV. Test Case Generation

In this section we propose an automatic workflow for testing the **tioco**$_c$ conformance relation between a Reo connector and a specification (given by a TCA $\mathcal{S}$). Firstly, we generate the TCA $\mathcal{I}$ of the Reo connector by composing TCA of each channels in the connector by the join operator. Then we introduce the test case $\mathcal{T}_{\mathcal{S}}$ of a specification $\mathcal{S}$ which helps checking the conformance relation. Finally, we show that the conformance relation checking problem is converted to a reachability checking problem of the product automata of $\mathcal{I}$ and $\mathcal{T}_{\mathcal{S}}$, and therefore can be automatically done (by model checking tools such as UPPAAL).

A TCA $\mathcal{A}$ is *deterministic* if for all $\sigma \in \mathsf{ObsTTraces}(\mathcal{A})$, $|\mathcal{A} \textbf{ after } \sigma| = 1$. We require that a specification $\mathcal{S}$ is a deterministic TCA without empty node-set on edges. This will help proving properties of test cases without losing expressiveness in practice.

Intuitively, in a test case, the event (which could be an I/O operation or a passage of time) that not **conf** with all expected events in $\mathcal{S}$ should lead to a *fail* location. The consistency of test case generation and **tioco**$_c$ relation will also be proved.

Our algorithm for test case generation can also deal with *quiescence*. A quiescent state is a state where the system is unable to produce an output now or in the future, without receiving an input first. The test case will also lead to *fail* location in a quiescent state. Figure 6 shows the test case of Spec in Fig. 5.

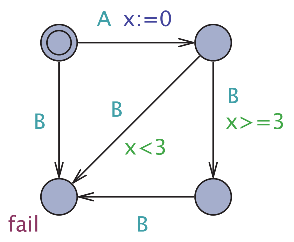

Fig. 6. Test case of Spec in Fig. 5

Definition 8 (Test case)**.** A test case $\mathcal{T}_{\mathcal{S}}$ of a TCA $\mathcal{S}$ is a TCA extended from $\mathcal{S}$. In the following, we always suppose $N_{in} \subseteq \mathcal{N}_{in}$ and $N_{out} \subseteq \mathcal{N}_{out}$. We say (N_{in}, dc_{in}) is a possible input edge from location q if there exists an edge $(q, N_{in} \cup N_{out}, dc_{in} \wedge dc_{out}, cc, X, q) \in \rightarrow$. In addition, we always see $(\emptyset, \emptyset)$ as a possible input edge.

Formally, given a specification $\mathcal{S} = (Q, q_0, \mathcal{N}_{in} \cup \mathcal{N}_{out}, \rightarrow, ic)$, then test case of $\mathcal{S}$ is $\mathcal{T}_{\mathcal{S}} = (Q \cup \{fail\}, q_0, \mathcal{N}_{in} \cup \mathcal{N}_{out}, \rightarrow', ic')$, where $ic'(q) =$ true for any $q \in Q \cup \{fail\}$ and $\rightarrow'$ is obtained by the following rules:

1) $\rightarrow \subseteq \rightarrow'$,
2) $(q, \emptyset, \text{true}, \neg ic(q), C, fail) \in \rightarrow'$ for each $q \in Q$.

For any (N_{in}, dc_{in}) which is a possible input edge from q and any node-set N' such that $N'_{in} = N_{in}$,

3) $(q, N', dc', \text{true}, C, fail) \in \rightarrow'$ where dc' is the weakest data constraint such that $dc'_{in} = dc_{in}$ and for each edge $(q, N'', dc'', cc, X, q') \in \rightarrow$ where $N''_{in} = N_{in}$ and $dc''_{in} = dc_{in}$ too, we have $N \neq N'$ or $dc' \wedge dc'' \equiv \text{false}$.

Under same assumptions of (N_{in}, dc_{in}) and N,

4) $(q, N, dc', cc', C, fail) \in \rightarrow'$ where $dc'_{in} = dc_{in}$ and cc' is the weakest clock constraint such that for each edge $(q, N, dc', cc, X, q') \in \rightarrow$, $\forall v \in cc' : v \notin cc \vee v[X := 0] \notin ic(q')$.

Intuitively, the following conditions will lead a test case T_S to the $fail$ location from its current location s:

- It does not leave location s before breaking the invariant.
- There is an edge such that the input part of the edge's node-set and data constraint have been defined in the specification, but the output part is not defined.
- The action performs with right data assignment but at a wrong time.

Here we prove some properties of $\mathcal{T}_{\mathcal{S}}$. Suppose $s = (q, v)$ is a state in the TLTS of $\mathcal{S}$, which is also a state in the TLTS $\mathcal{T}_{\mathcal{S}}$. We denote $E_S = \mathsf{out}(s)$ in $\mathcal{S}$ and $E_T = \mathsf{out}(s)$ in $\mathcal{T}_{\mathcal{S}}$. Apparently $E_S \subseteq E_T$, but there are more relations between these two sets of events.

Lemma 2. For any $\mu \in E_T$ then μ must satisfy exactly one of the following conditions:

- $\mu \in E_S$,
- $\mu = (N, \delta), N \neq \emptyset, s \xrightarrow{\mu} (fail, \vec{0})$,
- $\mu \in \mathbb{R}_{\geq 0}, s \xrightarrow{\mu(\emptyset,\emptyset)} (fail, \vec{0})$.

Proof: In the case of $\mu \in \mathbb{R}_{\geq 0}$, if $\mu \in ic(q)$, then $\mu \in E_S$, otherwise μ satisfies the third condition. In the case of $\mu = (N, \delta)$, it is obvious that either $\mu \in E_S$ or μ leads q to a $fail$ loction. ■

Lemma 3. For any $\mu \in \mathsf{Event}$, either μ **ignore** E_S or $\mu \in E_T$.

Proof: If $\mu \in \mathbb{R}$, since $ic'(q) =$ true, then $\mu \in E_T$. If $\mu = (\emptyset, \emptyset)$, μ **ignore** E_S. Now we consider the case of $\mu = (N, \delta)$,

1) if (N, δ) **ignore** E_S does not hold, then there must exist $(N', \delta') \in E_S$ such that $N'_{in} = N_{in}$ and $\delta'_{in} = \delta_{in}$;
2) then there must exist an edge $(q, N', dc', cc, X, q') \in \rightarrow$ such that $\delta' \vDash dc'$;
3) if $N \neq N'$ or $\delta \neq \delta'$, then based on step 3 in the definition of test case, μ lead q to a $fail$ location, so $\mu \in E_T$;
4) or $N = N'$ and $\delta = \delta'$, $\mu \in E_S$ if $v \in cc$, and $\mu \in E_T$ if $v \notin cc$ based on step 4.

■

The following lemma is obtained directly by Lemma 1 and Lemma 2.

Lemma 4. For any $\mu \in \mathsf{Event}$, exactly one of the following conditions holds:

- μ **conf** E_S,
- $\mu = (N, \delta), N \neq \emptyset, s \xrightarrow{\mu} (fail, \vec{0})$,
- $\mu \geq 0$ and $s \xrightarrow{\mu(\emptyset,\emptyset)} (fail, \vec{0})$.

The following lemma shows that $\mathcal{S}$ and $\mathcal{T}_{\mathcal{S}}$ will reach the same state after executing a sequence of observable events of $\mathcal{S}$.

Lemma 5. For any $\sigma \in \mathsf{ObsTTraces}(\mathcal{S}) : \mathcal{S}$ **after** $\sigma = \mathcal{T}_{\mathcal{S}}$ **after** σ.

Proof: Suppose σ_0 is the sequence of events with minimum length in $\mathsf{ObsTTraces}(\mathcal{S})$ and $\sigma_0 = \sigma'\mu$. For $\mathcal{S}$ is deterministic, we denote $\{s\} = (\mathcal{S}$ **after** $\sigma') = (\mathcal{T}_{\mathcal{S}}$ **after** $\sigma')$. Based on Lemma 2, since $\mu \in E_S$ ($E_S = \mathsf{out}(s)$ in $\mathcal{S}$), then μ cannot lead $\mathcal{T}_{\mathcal{S}}$ to a $fail$ location. Finally, $(\mathcal{S}$ **after** $\sigma) = (\mathcal{T}_{\mathcal{S}}$ **after** $\sigma)$. ■

Intuitively, if $\mathcal{I}$ **tioco**$_c$ $\mathcal{S}$, the failing state $fail$ should be unreachable in $\mathcal{I} \bowtie \mathcal{T}_{\mathcal{S}}$, and vice versa. The following theorem states this property formally. Figure 7 shows the test case of FIFO with lower time bounds.

Theorem 2. Let $\mathcal{I}$ and $\mathcal{S}$ be two TCA, $\mathcal{T}_{\mathcal{S}}$ be the test case of $\mathcal{S}$, and $(q_0, \vec{0})$ be the initial state of the TLTS of $\mathcal{I} \bowtie \mathcal{T}_{\mathcal{S}}$. Then

$$\mathcal{I} \textbf{ tioco}_c\ \mathcal{S} \iff \nexists \rho \in \mathsf{RT}(\mathcal{N}) : (q_0, \vec{0}) \xrightarrow{\rho} (\langle q', fail \rangle, v)$$

where q' is a loction of $\mathcal{I}$ and v is a clock assignment.

Proof: If there exists a trace $\rho : (q_0, \vec{0}) \xrightarrow{\rho} (\langle q', fail \rangle, v)$ with an observable part $\sigma = Obs(\rho) = \mu_1 \cdots \mu_n$. Based on Theorem 1, $\sigma \in \mathsf{ObsTTraces}(\mathcal{I})$ and $(fail, v') \in (\mathcal{T}_{\mathcal{S}}$ **after** $\sigma)$.

Let i be the maximal integer such that $\sigma' = \mu_1 \cdots \mu_i \in \mathsf{ObsTTraces}(\mathcal{S})$. Apparently, $i < n$. Based on Lemma 5, we denote $\{s\} = (\mathcal{S}$ **after** $\sigma') = (\mathcal{T}_{\mathcal{S}}$ **after** $\sigma')$. Since $\mu_{i+1} \in \mathsf{out}(s)$ in $\mathcal{T}_{\mathcal{S}}$ but $\mu_{i+1} \notin \mathsf{out}(s)$ in $\mathcal{S}$, we can infer $\mu\ \cancel{\textbf{conf}}\ \mathsf{out}(\mathcal{S}$ **after** $\sigma')$ from Lemma 1 and Lemma 3.

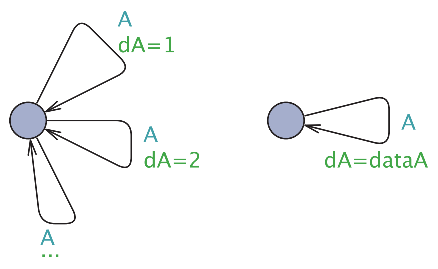

Fig. 7. Two ways to represent data

We also know $\mu \in \mathsf{out}(\mathcal{I}$ **after** $\sigma')$, so $\mathcal{I}$ **tioco**$_c$ $\mathcal{S}$ does not holds.

The proof of the opposite direction is similar. ∎

V. Testing Timed Reo Connectors in UPPAAL

The above discussion converts the problem of conformance testing to the problem of reachability checking in a TCA, which is similar to the work in [1]. We developed a toolset to accomplish this procedure automatically. This toolset implements the *join* and *hide* operators of TCA, as well as the algorithm of test case generation.

However, as far as we know, no proper model checking tools can be used to directly check TCA models. Therefore, we modify the syntax of TCA to fit in other model checkers. We choose UPPAAL [9] which is an integrated tool environment for modeling, validation and verification of real-time systems as our model checker.

In this section, we propose a framework to test timed Reo connectors in UPPAAL. Firstly UPPAAL works as an automata editor. The TCA of each channel as well as the specification are drawn in UPPAAL. Then we export the whole system as an XML file, product the TCA for different channels using *join* operator into one TCA as an implementation $\mathcal{I}$. The specification $\mathcal{S}$ is also exported and test case $\mathcal{T}_\mathcal{S}$ is generated, then we transform the syntax of $\mathcal{I} \bowtie \mathcal{T}_\mathcal{S}$ to a TSA to fit in UPPAAL. Then the reachability of *fail* location is checked in UPPAAL.

The automata we obtain in this approach may be very complex when data set $Data$ is quite large. Depending on different situations, we can make data abstraction in different levels. Sometimes the actual value of data is important, e.g., behavior of an alternating bit protocol depends on the value of data. In other situations, however, we are only concerned about where the I/O operations happen and do not need to know the actual data value. These two situations will lead to different complexities of the models.

If we want $Data$ to contain all the possible values, we may let $Data = \{1, 2, \cdots, n\}$ for example. Or we can label each data with the name of input ports. In this way, we define $Data = \{data_A, data_B, \cdots, data_N\}$, where $data_X$ represent the data transported from node X. Then the TCA models will be much simpler, as well as the finally generated product TCA. Fig. 7 shows a complete TCA and its abstract version.

When generating test cases, the idea is to formulate the problem as a reachability problem that can be solved with an existing model-checking tool. In this paper, we use the UPPAAL tool for reachability analysis. UPPAAL takes *Timed safety automata* (TSA) which was first introduced in [4] as input. The syntax of TSA is quite similar with TCA, we still assume CC is the set of clock constraints.

Definition 9. *(Timed safety automata)* A TSA is a tuple $(Q, q_0, A, C, \rightarrow, ic)$ where

- Q is a finite set of locations,
- q_0 is the initial location,
- A is a set of actions,
- C is a set of clocks,
- $\rightarrow \subseteq Q \times A \times CC \times 2^C \times Q$ is the set of edges,
- ic is an invariant function that assign each location q a clock constraint.

Since we only care if *fail* could be reached in T-CA, we do not need to keep the information of node-sets or data constraints when transforming to TSA. For a TCA $\mathcal{A} = (Q, C, \mathcal{N}, \rightarrow, q_0, ic)$, the corresponding TSA is $(Q, q_0, \emptyset, C, \rightarrow', ic)$ where $\rightarrow'$ is obtained by

$$\frac{(q, N, dc, cc, X, q') \in \rightarrow, dc \not\equiv \text{false}}{(q, \emptyset, cc, X, q')}.$$

We use the transformed TSA from $\mathcal{I} \bowtie \mathcal{T}_\mathcal{S}$ as the input of UPPAAL, then check whether $E <> fail$, which means there exists a path where *fail* is eventually reached. If the result is $false$, then there does not exist an RT-trace $\rho \in \mathsf{RT}$ such that $(\mathcal{N}) : (q_0, \vec{0}) \xrightarrow{\rho} (\langle q', fail\rangle, v)$. Therefore we know that $\mathcal{I}$ **tioco**$_c$ $\mathcal{S}$.

We have developed a prototype tool for testing Reo connectors in Java. The source code of the tool is available at http://github.com/lishaodong/TCAChecker. In the current version, the tool implements the *join* and *hide* operators of timed constraint automata, and it is also able to generate test cases and automatically convert TCA to TSA. There are also some examples on the website to demonstrate the possibility to automate the connector testing approach.

VI. Related Work

There are some works focused on testing of coordination protocols in Reo. A fault-based testing approach for test case generation from connector specifications was proposed in [2], [21]. Another approach to test connector designs given specifications of their expected behavior in the form of constraint automata extended with inputs and outputs was proposed in [15], in which the automata-based behavioral semantics for Reo is encoded in process algebra mCRL2 [13], which provides a powerful toolset for checking different kinds of properties of mCRL2 processes. However, timed connectors are not considered in [2], [15] and thus the approaches can only apply to untimed connectors. Timed connectors have been investigated in [6], [7], where timed constraint automata is proposed as the semantic model for timed Reo. But the testing issues are not considered in [6], [7].

Input-output conformance testing was first formally defined by Tretmans[22]. Based on the work of Tretmans, Brandán-Briones and Brinksma, Krichen and Tripakis and Larsen et al. have defined input-output conformance relation and test generation for timed systems in [11], [16], [18] separately. Krichen and Tripakis defined a conformance relation without quiescence where time is viewed as output of the implementation. Brandán-Briones and Brinksma defined a conformance relation with a quiescence action. Since then, many conformance relations with different kinds of extension have been proposed. For example, *Relativized Timed Conformance* ($\mathbf{rtioco}$) was proposed in [14], which suppose there is an environment and both implementation and specification can communicate with the environment. [20] gives a through discussion about all kinds of timed conformance relation, and also proved the equivalence for some of them under certain condition.

Test generation and execution based on the notion of test by Brandán-Briones and Brinksma has been implemented in TorX. A timed version of TorX was implemented by Bohnenkamp and Belinfante [10]. They also redefined timed input-output conformance with quiescence for timed automata instead of timed transition system. Test generation and execution based on the notion of test by Krichen and Tripakis has been implemented in the prototype tool TTG [19]. Independently the same notion of test has been implemented in UPPAAL-TRON by Larsen et al [23].

VII. Conclusion and Future Work

In this paper, we define a conformance relation $\mathbf{tioco}_c$ for timed connectors in the coordination language Reo. This relation defines when an implementation conforms to a specification given by a timed constraint automata. To our knowledge, this is the first implementation for testing timed connectors. We modeled a connector by a TCA, which is obtained by the product of TCA for its channels, and convert it to a TSA as the implementation under test imported into UPPAAL. Then we present an approach changing conformance relation checking problem to fail location reachability problem to check the test case automatically.

The approach to generate test case in this paper is quite intuitive, and the size of test case may be very large. Suppose m, n are the numbers of input and output actions respectively, then the number of edges from each location in the test case will be $O(m \times 2^n)$ at the worst case. We may make some optimization and simplify the test cases in the future. Off-line testing has its own merits and demerits. It is reusable and could save time during the testing, but its space complexity is also a problem when the size of specification is quite large. We may extend our work and integrate with online test case generation together. In our future work, we will also investigate the conformance relation for hybrid Reo connectors [12].

Acknowledgement: The work was partially supported by the NSFC under grant no. 61202069 and 61272160, and Research Fund for the Doctoral Program of Higher Education of China under grant no. 20120001120103. The authors are indebted to Dr. Jun Sun for helpful discussions.

References

[1] L. Aceto, P. Bouyer, A. Burgueño, and K. G. Larsen. The power of reachability testing for timed automata. *Theoretical Computer Science*, 300(1-3):411–475, 2003.

[2] B. K. Aichernig, F. Arbab, L. Astefanoaei, F. S. de Boer, M. Sun, and J. J. M. M. Rutten. Fault-based test case generation for component connectors. In *Proceedings of TASE 2009*, pages 147–154. IEEE Computer Society, 2009.

[3] R. Alur and D. L. Dill. Automata For Modeling Real-Time Systems. In *Proceedings of ICALP'90*, volume 443 of *LNCS*, pages 322–335. Springer, 1990.

[4] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[5] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[6] F. Arbab, C. Baier, F. de Boer, and J. Rutten. Models and Temporal Logics for Timed Component Connectors. In *Proceedings of SEFM2004*, pages 198–207. IEEE Computer Society, 2004.

[7] F. Arbab, C. Baier, F. de Boer, and J. Rutten. Models and Temporal Logical Specifications for Timed Component Connectors. *Software and System Modeling*, 6(1):59–82, 2007.

[8] C. Baier, M. Sirjani, F. Arbab, and J. Rutten. Modeling component connectors in Reo by constraint automata. *Science of Computer Programming*, 61:75–113, 2006.

[9] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. In *Hybrid Systems III: Verification and Control*, volume 1066 of *LNCS*, pages 232–243. Springer, 1996.

[10] H. C. Bohnenkamp and A. Belinfante. Timed Testing with TorX. In *Proceedings of FM 2005*, volume 3582 of *LNCS*, pages 173–188. Springer, 2005.

[11] L. B. Briones and E. Brinksma. A Test Generation Framework for *quiescent* Real-Time Systems. In *FATES 2004, Revised Selected Papers*, volume 3395 of *LNCS*, pages 64–78. Springer, 2005.

[12] X. Chen, J. Sun, and M. Sun. A Hybrid Model of Connectors in Cyber-Physical Systems. In *Proceedings of ICFEM 2014*, volume 8829 of *LNCS*, pages 59–74. Springer, 2014.

[13] J. F. Groote and M. R. Mousavi. *Modeling and Analysis of Communicating Systems*. The MIT Press, 2014.

[14] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, and A. Skou. Testing Real-Time Systems Using UPPAAL. In *Formal Methods and Testing*, volume 4949 of *LNCS*, pages 77–117. Springer, 2008.

[15] N. Kokash, F. Arbab, B. Changizi, and L. Makhnist. Input-output Conformance Testing for Channel-based Service Connectors. In *Proceedings of PACO 2011*, volume 60 of *EPTCS*, pages 19–35, 2011.

[16] M. Krichen and S. Tripakis. Black-Box Conformance Testing for Real-Time Systems. In *Proceedings of SPIN 2004*, volume 2989 of *LNCS*, pages 109–126. Springer, 2004.

[17] M. Krichen and S. Tripakis. Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3):238–304, 2009.

[18] K. G. Larsen, M. Mikucionis, B. Nielsen, and A. Skou. Testing real-time embedded software using UPPAAL-TRON: an industrial case study. In *Proceedings of EMSOFT 2005*, pages 299–306. ACM, 2005.

[19] M. Krichen. The timed test generator tool. http://www-verimag.imag.fr/∼krichen/ttg/index.html.

[20] J. Schmaltz and J. Tretmans. On Conformance Testing for Timed Systems. In *Proceedings of FORMATS 2008*, volume 5215 of *LNCS*, pages 250–264. Springer, 2008.

[21] M. Sun, F. Arbab, B. K. Aichernig, L. Astefanoaei, F. S. de Boer, and J. Rutten. Connectors as Designs: Modeling, Refinement and Test Case Generation. *Science of Computer Programming*, 77(7-8):799–822, 2012.

[22] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, 17(3):103–120, 1996.

[23] Uppsala Universiteit and Aalborg University. UPPAAL TRON. http://www.cs.aau.dk/∼marius/tron/.