Remote Testing of Timed Specifications ⋆

Alexandre David[1], Kim G. Larsen[1], Marius Mikučionis[1], Omer L. Nguena Timo[2], Antoine Rollet[2]

[1] Department of Computer Science, Aalborg University, Denmark
{ adavid, kgl, marius } @cs.aau.dk
[2] LaBRI, University of Bordeaux - CNRS, France
{ nguena, rollet } @labri.fr

Abstract. We present a study and a testing framework on black box remote testing of real-time systems using Uppaal-TIGA. One of the essential challenges of remote testing is the communication latency between the *tester* and the *system under test* (SUT) that may lead to interleaving of inputs and outputs. This affects the generation of inputs for the SUT and the observation of outputs that may trigger a wrong test verdict. We model the overall test setup using Timed Input-Output Automata (TIOA) and present an adapted asynchronous semantics with explicit communication delays. We propose the Δ-testability criterion for the requirement model where Δ describes the communication latency. The test case generation problem is then reduced into a controller synthesis problem. We use Uppaal-TIGA for this purpose to solve a timed game with partial observability between the tester and the communication media together with the SUT. The objective of the game corresponds to a test purpose.

1 Introduction

This paper deals with black box conformance testing of remote real-time systems. Usually, conformance black-box testing is an activity where a tester executes selected *test cases* on a system (implementation) under test (SUT) and emits a test verdict (**pass**, **fail**, etc.). This verdict indicates the conformance between SUTs and the specification. It is computed according to the specification and a conformance relation between SUTs and the specification. Usually, the assumption of zero delay and synchronous communication between the tester and the SUT is done, but this is not realistic in many situations, such as network applications, or systems when time matters. In some cases, it may provide an erroneous verdict, potentially implying catastrophic situations. Our goal is to study the impact of explicit propagation delays between the implementation and the tester on test case generation and execution, and to provide a general testing framework for remote testing of real-time systems modeled with timed automata.

ioco *based theory.* In the case of untimed systems, the most common approaches are based on the Labeled Transition Systems (LTS) model, which is used as a semantics for many standardized languages such as SDL [1] or LOTOS [2]. The ioco relation theory [3] proposes a complete testing approach for LTS with inputs and outputs, using the idea that any output of the implementation should be authorized by the specification.

⋆ This work has been partially supported by the French research project ANR VACSIM.

They also introduce the notion of *quiescence* permitting to consider blocking states as a special output which should be explicitly specified. A complete framework based on this theory has been proposed in [4], especially providing the possibility to use Test Purposes in order to lead the testing process.

Testing with time. The ioco theory has inspired many testing approaches. [5] proposes an extension of the Finite States Machines with Time (TEFSM) and defines adapted conformance relations. [6], [7] and [8] propose extensions of ioco relation with time (tioco) including delays in the set of observable actions, leading to infinite systems. They propose non deterministic test generation algorithms based on Timed Input/Output Automata (TIOA). [8] also shows how to use the Uppaal tool suite in order to generate offline test cases using coverage criteria for timed models. More recently, [9] proposes a formal framework permitting to use Test Purpose and non-deterministic Timed Automata thanks to a determinization algorithm.

Remote testing. These works introduce conformance relations and test selection algorithms based on synchronous test execution algorithms. However, in the untimed setting, [10] and [11] point out the fact that synchronous execution of tests cases is not realistic when there is a significant distance between testers and SUTs. Under this remoteness assumption, the adequate communication mode should be asynchronous. [10] considers that SUTs and testers communicate via input and output queues, and asynchronous points of control of observation (PCOs). [10] also shows that simply using logical stamps permits to obtain the "same power" of testing than in a synchronous environment. In a more general way, [12] proposes a study revisiting asynchronous testing and showing possibilities (or not) to synchronize asynchronous testing.

Usually, the works done on this topic consider this as a "distributed" testing, implying several entities in a "system" of components. In this case, most effort is done for solving the problem of relative order between events induced by the communication process without state space explosion. [13] proposes a test generation framework using several Input Output State Machines (IOSM) and perfect FIFO queues between them. Then the author uses the Prime Event Structure in order to fix the problems of interleaving in the test generation process. Still using queues, but with Input Output Transition Systems (IOTS), [11] proposes a method to generate sound test cases with test purposes. They apply a transformation of the test purpose allowing to consider all possible distortions induced by the queues. The problem of interleaving is also addressed in [14] where authors focus on testing of concurrent systems. They propose to use Labeled Event Structures and partial order semantics in order to handle lightly concurrency aspects in the conformance relation. [15] proposes to add local clocks in each component and timestamps directly included in the exchanged messages. They propose different strategies depending on assumptions regarding how the clocks relate and give adapted conformance relations.

Contributions. The related works described above provide testing techniques for untimed systems. To the best of our knowledge, there is no explicit work that considers remote testing of timed specifications. Our contribution is two-fold. Firstly and based on a real example, we show how remote testing can be performed by modelling the communication channels with processes that delay the actions and synchronize with

the SUT and its environment. We discuss the drawback of this general approach. Then, after considering timed asynchronous behaviours, we provide a Δ-testability criterion ensuring remote testing with the same detecting properties as local one.

The paper is organised as follows. Section 2 recalls well-known concepts of the model-based testing theory with TIOA. In Section 3, we address the challenges for the remote testing. We present the disadvantages of using asynchronous timed traces in general. Asynchronous semantics described the observations of a remote tester. Section 4 relates observed traces with the traces of the implementation. We define the Δ-testable criterion and present some interesting properties. The remote testing framework with Uppaal-TIGA is described in Section 5 and it is followed by a case study in Section 6.

2 The tioco-based Testing Theory

The tioco-testing theory is based on the representation of the specifications and the implementations with deterministic TIOA. Let us now present formal notations and concepts for the tioco-testing theory that we extend later for remote testing.

Timed word, timed sequence, and timed trace. In the sequel, $\mathbb{R}_{\geq 0}$ denotes the set of non negative real-numbers that we will often call *delays*. A *timed word* over an alphabet of actions Γ is an element $w = w_1 \cdot w_2 \cdot \ldots \cdot w_n$ of $(\mathbb{R}_{\geq 0} \cup \Gamma)^*$. We define $w[i] = w_i$ and $w[i..j] = w_i \cdot w_{i+1} \cdot \ldots \cdot w_{j-1} \cdot w_j$, and $|w| = n$ denotes the length of w. We consider the causal/dependency relation between the actions in w and we say that w_j *depends on* w_i when $i < j$ and we write $w_i \prec_\rho w_j$. The timed word w is a *timed sequence* if the projection of w over $\mathbb{R}_{\geq 0}$ is empty or an increasing sequence of real-numbers i.e $\forall 0 \leq i \leq j \leq |n|$ such that $w_i, w_j \in \mathbb{R}_{\geq 0}$, it holds that $w_i \leq w_j$. A timed sequence is called a *timed trace* if it is a sequence of timestamped actions followed with a delay i.e it belongs to $(\mathbb{R}_{\geq 0} \times \Gamma)^* \times \mathbb{R}_{\geq 0}$. We will consider that the behaviours of real-time systems can be described with timed traces. Each delay in a timed sequence refers to the time that has elapsed since the system started.

2.1 Input/Output Timed Transition Systems (IOTTS)

Definition 1 (IOTTS). *An* input/output timed transition system *(*IOTTS*) is a tuple* $\mathcal{S} = \langle S, s^0, I, O, \Lambda, M \rangle$ *where* S *is the set of* states*,* s^0 *is the* initial state*,* I *is a finite set of* input actions*,* O *is a finite set of* output actions*,* Λ *is a finite set of* silent actions*,* $M \subseteq S \times (I \cup O \cup \Lambda \cup \mathbb{R}_{\geq 0}) \times S$ *is the set of* moves*. We will write* $s \xrightarrow{\alpha} s'$ *with* $\alpha \in (I \cup O \cup \Lambda \cup \mathbb{R}_{\geq 0})$ *to represent a move* $(s, \alpha, s') \in M$.

Moreover, we require the following standard properties for IOTTS : *time-Determinism* (if $s \xrightarrow{d} s'$ and $s \xrightarrow{d} s''$ with $d \in \mathbb{R}_{\geq 0}$, then $s' = s''$) , 0-*Delay* ($s \xrightarrow{0} s$), *additivity* (if $s \xrightarrow{d} s'$ and $s' \xrightarrow{d'} s''$ with $d,\ d' \in \mathbb{R}_{\geq 0}$, then $s \xrightarrow{d+d'} s''$), *continuity* (if $s \xrightarrow{d} s'$, then for every d' and d'' in $\mathbb{R}_{\geq 0}$ such that $d = d' + d''$, there exists s'' such that $s \xrightarrow{d'} s'' \xrightarrow{d''} s'$).

We denote by IOTTS(I, O, Λ), the class of IOTTS of which the input actions, the output actions and the silent actions belong to I, O and Λ, respectively. For $\mathcal{S} \in$ IOTTS(I, O, Λ), we define $Act(\mathcal{S}) = I \cup O \cup \Lambda$.

Notations. In the sequel we write $s \xrightarrow{\alpha}$ with $\alpha \in Act(\mathcal{S}) \cup \mathbb{R}_{\geq 0}$ when there exists $s' \in Q$ such that $s \xrightarrow{\alpha} s'$. We write $s \xrightarrow{\alpha_1.\alpha_2....\alpha_n} s'$ with $\alpha_i \in Act(\mathcal{S}) \cup \mathbb{R}_{\geq 0}$ when there exists $s_1, s_2, \ldots s_{n-1} \in S$ such that $s \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \rightarrow \cdots \rightarrow s_{n-1} \xrightarrow{\alpha_n} s'$.

Executions and timed traces. A *run* of $\mathcal{S}$ starting at $s \in S$, is a finite sequence $\pi = s.(\alpha_i.s_i)_{i=1..n} \in S \times ((Act(S) \cup \mathbb{R}_{\geq 0}) \times S)^*$ such that $s \xrightarrow{\alpha_1} s_1$ and $s_i \xrightarrow{\alpha_i} s_{i+1}$. We denote $\mathsf{Runs}(s)$ the set of *runs* of $\mathcal{S}$ starting from s and $\mathsf{Runs}(\mathcal{S}) = \mathsf{Runs}(s^0)$. The execution sequence of π is the sequence $\mathsf{Seq}(\pi) = \alpha_1.\alpha_2.\ldots.\alpha_n \in ((Act(S) \cup \mathbb{R}_{\geq 0})^*$, and we naturally extend the notation with $\mathsf{Seq}(s) = \{\mathsf{Seq}(\pi) \mid \pi \in \mathsf{Runs}(s)\}$ and $\mathsf{Seq}(\mathcal{S}) = \mathsf{Seq}(s^0)$. As usual, a move $s \xrightarrow{\alpha} s'$ with $a \in Act(\mathcal{S})$ means that s' is reached when the action a is executed on s (discrete move). A move $s \xrightarrow{d} s'$ with $d \in \mathbb{R}_{\geq 0}$ means that the state s' is reached after d time units has elapsed from s; so d is interpreted as the time distance between s and s' (time elapse).

The environment cannot observe the executions of silent actions in Λ. Moreover delays in executions are time distances between states. A timed trace corresponding to an execution is a timed sequence consisting of time-stamps and visible actions and such that the time-stamps indicate the dates of occurrences of the actions. Given an execution sequence $\rho = (\alpha_i)_{i=1..n} \in (Act(\mathcal{S}) \cup \mathbb{R}_{\geq 0})^*$, the timed trace associated with ρ is denoted $\mathsf{ttrace}(\rho)$ and it is defined by $\mathsf{ttrace}(\rho) = \mathsf{obs}(0, \rho)$ where

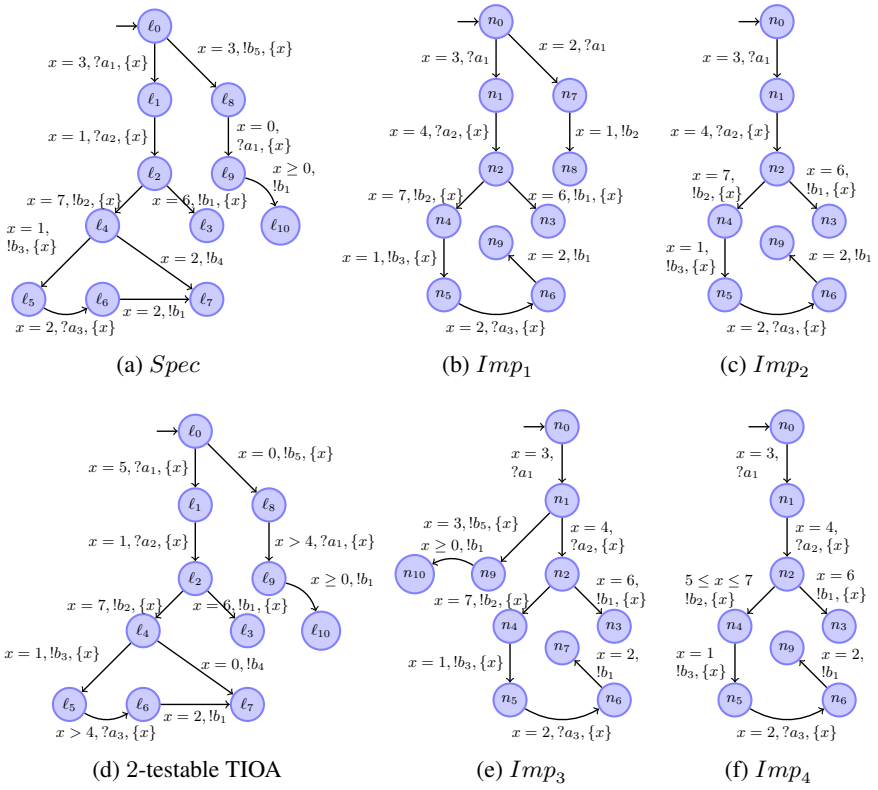
$$\mathsf{obs} : \mathbb{R}_{\geq 0} \times (I \cup O \cup \Lambda \cup \mathbb{R}_{\geq 0})^* \rightarrow (\mathbb{R}_{\geq 0} \times (I \cup O))^* \times \mathbb{R}_{\geq 0}$$

is a function that removes silent action from execution actions and that computes the date of the occurrence of the input and output actions. We propose the following recursive definition of obs: $obs(d, \varepsilon) = d$ with $d \in \mathbb{R}_{\geq 0}$; then $obs(d, \alpha.w)$ equals $obs(d + \alpha, w)$ if $\alpha \in \mathbb{R}_{\geq 0}$, otherwise it equals $obs(d, w)$ if $\alpha \in \Lambda$, otherwise it equals $d.\alpha.obs(d, w)$ if $\alpha \in (I \cup O)$.
In the sequel, $\mathsf{TTraces}(\mathcal{S}) = \{\mathsf{ttrace}(\rho) \mid \rho \in \mathsf{Seq}(\mathcal{S})\}$ denotes the set of *timed traces* of $\mathcal{S}$. Note that since $\sigma = (\delta_i \cdot a_i)_{i=1..m}.\delta_{m+1} \in \mathsf{TTraces}(\mathcal{S})$ is a timed sequence, it implies that $\delta_i \leq \delta_{i+1}$ for every $i \in [1..m]$. Given a timed trace $\sigma \in (\mathbb{R}_{\geq 0} \times (I \cup O))^* \times \mathbb{R}_{\geq 0}$, we consider as usual the after operator: $s \text{ after } \sigma = \{s' \in S \mid \exists \rho \in \mathsf{Seq}(s') \text{ s.t } s \xrightarrow{\rho} s' \wedge \sigma = \mathsf{ttrace}(\rho)\}$ represents the set of states that can be reached from s and after observing the behaviour σ. Then we define $\mathsf{elapse}(s) = \{\delta \in \mathbb{R}_{\geq 0} \mid s \xrightarrow{\delta}\}$ (Notice that $\mathsf{elapse}(s) = \mathbb{R}_{\geq 0}$ when there is no restriction on the elapse of the time in s), $\mathcal{S} \text{ after } \sigma = s^0 \text{ after } \sigma$; and $\mathsf{out}(s) = \{a \in O \mid \exists \rho \in \mathsf{Seq}(s) \text{ s.t } \mathsf{ttrace}(\rho) = (0 \cdot a) \cdot 0\} \cup \mathsf{elapse}(s)$ denotes the set of delays augmented with the set of outputs that can be observed from s without any delay, possibly preceded by the execution of silent actions. Illustrations of all these notations can be found in Example 1.

Complete and deterministic IOTTS. We say that $\mathcal{S}$ is *deterministic* if it has no silent transition and $s' = s''$ whenever there exists s, $\alpha \in Act(\mathcal{S})$, $s \xrightarrow{\alpha} s'$ and $s \xrightarrow{\alpha} s''$. IOTTS $\mathcal{S}$ is *input-complete* if all the inputs can be executed (observed) in each state.

2.2 Timed Input/Output Automata (TIOA)

A clock is a real-valued variable. Let X denote a set of clocks. A (clock) valuation over X is a function $v : X \rightarrow \mathbb{R}_{\geq 0}$ that assigns a non negative real value to each clock.



(a) $Spec$ (b) Imp_1 (c) Imp_2

(d) 2-testable TIOA (e) Imp_3 (f) Imp_4

Fig. 1: Models of one specification and four implementations

The set of valuations over X is denoted by $\mathbb{R}^X_{\geq 0}$. As usual, we consider two operations on valuations: the reset of the clocks and the elapse of the time. Given a valuation v, a real number $t \in \mathbb{R}_{\geq 0}$ and a subset of clocks $Y \subseteq X$, the valuation $v[Y := 0]$ is obtained from v by resetting every clock in Y and the valuation $v + t$ increases by t the value $v(x)$ of each clock $x \in X$. Formally, $v[Y := 0](x) = 0$ when $x \in Y$, otherwise $v[Y := 0](x) = v(x)$; and $(v + t)(x) = v(x) + t$. The valuation $\mathbf{0}$ assigns the zero value to every clock. A clock constraint over X is a boolean combination of equations of the form $n \preceq x$ where $n \in \mathbb{Q}$ is a rational number, $\preceq \in \{<, >, =, \leq, \geq\}$ and $x \in X$. We will denote by $\mathcal{C}(X)$ the set of clock constraints over X. The truth value of a clock constraint is computed w.r.t a valuation and this notion is standard. We say that $v \in \mathbb{R}^X_{\geq 0}$ satisfies $g \in \mathcal{C}(X)$ and we write $v \models g$ if g evaluates to true w.r.t to v.

Definition 2 (TIOA). *A timed input/output automaton (*TIOA*) is a tuple* $A = \langle L, \ell^0, I, O, \Lambda, X, E \rangle$ *where* L *is a finite set of* locations*,* ℓ^0 *is the* initial location*,* X *is a finite set of* clocks*,* I *is a finite set of* input actions*,* O *is a finite set of* output actions*,* Λ *is a finite set of* silent actions*,* $E \subseteq L \times (\mathcal{C}(X) \times (I \cup O) \times 2^X) \times L$ *is the set of edges.*

Definition 3 (Semantics of TIOA). *Let* $A = \langle L, \ell^0, I, O, \Lambda, X, E\rangle$ *be a* TIOA. *The semantics of* A *is the* IOTTS$(I, O, \emptyset)$ $[\![A]\!] = \langle S_A, s_A^0, I, O, \emptyset, M_A\rangle$ *where:* $S_A = L \times \mathbb{R}_{\geq 0}^X$ *is the set of* states of A, $s_A^0 = (\ell^0, \mathbf{0})$ *is the* initial state of A, I *is the set of inputs of* A, O *is the set of outputs of* A, $M_A \subseteq S_A \times (I \cup O \cup \mathbb{R}_{\geq 0}) \times S_A$ *is the set of moves of* A *defined such that:* $((\ell, v), d, (\ell, v+d))$ *for every* $d \in \mathbb{R}_{\geq 0}$, $(\ell, v) \in S_A$; *and* $((\ell, v), a, (\ell', v[Y := 0]))$ *whenever* $(\ell, g, a, Y, \ell') \in E$ *and* $v \models g$.

We define $\mathsf{TTraces}(A) = \mathsf{TTraces}([\![A]\!])$ and A after $\sigma = [\![A]\!]$ after σ with $\sigma \in \mathsf{TTraces}(A)$.

Deterministic TIOA. We say that A is deterministic if $[\![A]\!]$ is deterministic. Similarly, A is input-complete if $[\![A]\!]$ is input-complete.

Example 1. The TIOA $Spec$ in Fig. 1a is composed of a single clock x, the inputs are $?a_1, ?a_2, ?a_3$ and the outputs are $!b_1, !b_2, !b_3, !b_4, !b_5$. The edge $\ell_0 \xrightarrow{x=3, ?a_1, \{x\}} \ell_1$ is *passed* provided that x equals 3, the input $?a_1$ is received and x set to 0 just after the passing of the edge. An execution of $Spec$ is $\pi_1 = (\ell_0, 0) \xrightarrow{0.5} (\ell_0, 0.5) \xrightarrow{2.5} (\ell_0, 3) \xrightarrow{?a_1} (\ell_1, 0) \xrightarrow{0.7} (\ell_1, 0.7) \xrightarrow{0.3} (\ell_1, 1) \xrightarrow{?a_2} (\ell_2, 0) \xrightarrow{2} (\ell_2, 2) \xrightarrow{1} (\ell_2, 3) \xrightarrow{4} (\ell_2, 7) \xrightarrow{!b_2} (\ell_4, 0) \xrightarrow{1} (\ell_4, 1) \xrightarrow{!b_3} (\ell_5, 0) \xrightarrow{1} (\ell_5, 1) \xrightarrow{1} (\ell_5, 2) \xrightarrow{?a_3} (\ell_6, 0)$, and the execution sequence associated with π_1 is $\mathsf{Seq}(\pi_1) = 0.5 \cdot 2.5 \cdot ?a_1 \cdot 0.7 \cdot 0.3 \cdot ?a_2 \cdot 2 \cdot 1 \cdot 4 \cdot !b_2 \cdot 1 \cdot !b_3 \cdot 1 \cdot 1 \cdot ?a_3$ and the associated timed trace is $\mathsf{TTraces}(\mathsf{Seq}(\pi_1)) = (3 \cdot ?a_1) \cdot (4 \cdot ?a_2) \cdot (11 \cdot !b_2) \cdot (12 \cdot !b_3) \cdot (14 \cdot ?a_3) \cdot 0$. We have that $Spec$ after $(3 \cdot ?a_1) \cdot (4 \cdot ?a_2) \cdot (11 \cdot !b_2) \cdot 0 = \{(\ell_4, 0)\}$, $Spec$ after $(3 \cdot ?a_1) \cdot (4 \cdot ?a_2) \cdot (10 \cdot !b_1) \cdot 0 = \{(\ell_3, 0)\}$, $\mathsf{out}(Spec$ after $(3 \cdot ?a_1) \cdot (4 \cdot ?a_2) \cdot 9) = \mathbb{R}_{\geq 0}$, $\mathsf{out}(Spec$ after $(3 \cdot ?a_1) \cdot (4 \cdot ?a_2) \cdot 10) = \{!b_1\} \cup \mathbb{R}_{\geq 0}$ and $\mathsf{out}(Spec$ after $(3 \cdot ?a_1) \cdot (4 \cdot ?a_2) \cdot 11) = \{!b_2\} \cup \mathbb{R}_{\geq 0}$.

2.3 The Relation tioco and Synchronous Testing

As usual, the SUT is represented with an input-complete TIOA. We recall the tioco conformance relation definition, a common extension of **ioco**. In the following, we use this relation for conformance between the SUT and the specification.

Definition 4 (tioco). *Let* $\mathcal{S}$ *and* $\mathcal{I}$ *be in* TIOA(I, O) *where* $\mathcal{I}$ *is input-complete.*

$$\mathcal{I}\ \mathsf{tioco}\ \mathcal{S}\ \textit{iff} \forall \sigma \in \mathsf{TTraces}(\mathcal{S}), \mathsf{out}(\mathcal{I}\ \mathsf{after}\ \sigma) \subseteq \mathsf{out}(\mathcal{S}\ \mathsf{after}\ \sigma)$$

Example 2. Consider $Spec$ and the four implementations depicted in Figure 1. We can verify that Imp_1 conforms with $Spec$ even though Imp_1 can receive $?a_1$ when $x = 2$. The same, Imp_2 tioco $Spec$. But Imp_3 does not because $\mathsf{out}(Imp_3$ after $(3 \cdot ?a_1)) = \{!b_5\} \cup \mathbb{R}_{\geq 0}$ and $\mathsf{out}(Spec_1$ after $(3 \cdot ?a_1)) = \mathbb{R}_{\geq 0}$ and Imp_4 does not conform with $Spec$ because $\mathsf{out}(Imp_4$ after $(3 \cdot ?a_1) \cdot (4 \cdot ?a_2) \cdot 9) = \{!b_2\} \cup \mathbb{R}_{\geq 0}$ whereas $\mathsf{out}(Spec$ after $(3 \cdot ?a_1) \cdot (4 \cdot ?a_2) \cdot 9) = \mathbb{R}_{\geq 0}$.

An on-line tester for an SUT simulates the specification either by sending an input to the SUT or by letting the time elapses, while checking that the outputs emitted by the SUT are expected. Upon the reception of an unexpected output (outputs that are not specified or that arrive at bad instants), the tester emits the verdict **fail** indicating that

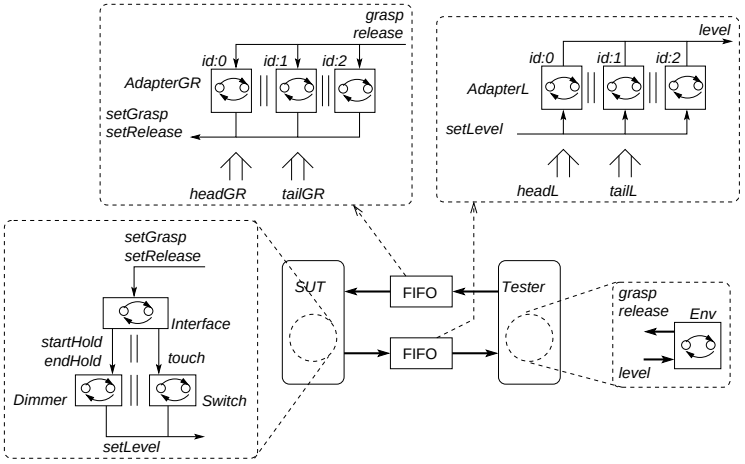


Fig. 2: The modeling pattern and how our example of the light controller is modeled.

the SUT does not conform to the specification. The communications between the tester and the SUT are synchronous meaning that the tester blocks upon transmitting an input to (receiving an output from) the implementation. There is no latency. Consequently, the time instant at which the tester sends available inputs or receives expected outputs should be exactly those described by the specification. Local and synchronous testers control the tests i.e each time a tester observes an output, that output depends on all the inputs it has sent before. The controllability of the test is an important property giving the possibility to lead the SUT into a particular situation.

3 Introduction to Remote Testing and Challenges

The main idea is that the SUT and the tester are not located at the same site and communications may be delayed. Fig. 2 illustrates our framework for remote testing. The model is centered around a $2FIFO(\bowtie, \Delta)$ architecture that consists of:

1. One FIFO for each direction of the communication between the SUT and the tester.
2. A communication latency bounded by Δ. The symbol $\bowtie$ stands for either $\leq$ or $=$.

3.1 Remote Testing Challenges

Remote test cases are different from the test cases designed for local testers. When the transmission of an input depends on the reception of an output, a remote tester should not wait to receive the output before sending the input since there is a latency. The experimentation with Uppaal-TIGA highlights this point. Let us now consider simple specification models to provide a theoretical point of view of remote testing.

Example 3. Consider the specification $Spec$ in Fig. 1a and assume that the latency is exactly 2 time units. If the tester wants that SUT receives a_1 at global time 3, it should send a_1 at time 1. When SUT sends b_2 at time 11, the tester receives it at time 13. etc. So

a tester shall observe the timed trace $\sigma'_1 = (1{\cdot}?a_1){\cdot}(2{\cdot}?a_2){\cdot}(12{\cdot}?a_3){\cdot}(13{\cdot}!b_2){\cdot}(14{\cdot}!b_3){\cdot}0$ and the SUT executes the timed trace $\sigma_1 = (3{\cdot}?a_1) \cdot (4{\cdot}?a_2) \cdot (11{\cdot}!b_2) \cdot (12{\cdot}!b_3) \cdot (14{\cdot}?a_3).0$. Note that the outputs $!b_2$ and $!b_3$ follow $?a_3$ in σ'_1 contrary to the trace $\sigma_1 \in \mathsf{TTraces}(Spec)$. This means that the tester does not wait for $!b_2$ and $!b_3$ before sending $?a_3$ despite the fact that the SUT sends $!b_2$ and $!b_3$ before receiving $?a_3$.

Remote testing introduces two news challenges: managing the **signal propagation delay** between the tester and the SUT; and managing the **input/output interleaving** caused by the asynchronous communication: the actions are not always received in the order they are transmitted and received.

In the next subsection we study the asynchronous traces and we will study the impact of the propagation delay and the interleaving on the tests cases.

3.2 Testing with Asynchronous Traces

We introduce the asynchronous semantics for TIOA and we present results on using asynchronous traces for testing. The asynchronous semantics for TIOA takes into account the queues and the latency; it describes the influence of the latency on the transmissions and the receptions of the actions.

Definition 5 (Asynchronous semantics for TIOA). *Let $A = \langle L, \ell^0, I, O, \emptyset, X, E\rangle$ be a* TIOA(I,O) *with no silent action. Let $\bowtie \in \{\leq, =\}$ and $\Delta \in \mathbb{N}$. The asynchronous semantics for A is an* IOTTS$(I, O, \Lambda_{I\cup O})$*,*
$(\!|A|\!)^{\bowtie\Delta} = \langle (L{\times}\mathbb{R}^X_{\geq 0}){\times}(\mathbb{R}_{\geq 0}{\times}(I{\cup}O))^*{\times}(\mathbb{R}_{\geq 0}{\times}(I{\cup}O))^*, (\ell_0, \mathbf{0}), I, O, \Lambda_{I\cup O}\}, M_{\bowtie\Delta}\rangle$ *where $\Lambda_{I\cup O} = \{\tau_a \mid a \in I \cup O\}$ is the set of silent actions. An asynchronous state is of the form $((\ell, v), p, q)$ where p and q are input and output queues respectively. The set of asynchronous moves, $M_{\bowtie\Delta}$ is defined by the following five rules:*

$$\frac{((\ell,v),p,q) \xrightarrow{?a} ((\ell,v),p.(0{\cdot}?a),q)}{}(\mathsf{r1}) \qquad \frac{((\ell,v),(\delta{\cdot}?a).p,q) \xrightarrow{\tau_a} ((\ell', v[Y:=0]),p,q)}{\ell \xrightarrow{g,?a,Y} \ell' \wedge v \models g \wedge \delta \bowtie \Delta}(\mathsf{r2})$$

$$\frac{((\ell,v),p,q) \xrightarrow{t} ((\ell,v+t),p+t,q+t)}{}(\mathsf{r3})$$

$$\frac{((\ell,v),p,(\delta{\cdot}!b).q) \xrightarrow{!b} ((\ell,v),p,q)}{\delta \bowtie \Delta}(\mathsf{r4}) \qquad \frac{((\ell,v),p,q) \xrightarrow{\tau_b} ((\ell', v[Y:=0]),p,q.(0{\cdot}!b))}{\ell \xrightarrow{g,!b,Y} \ell' \wedge v \models g}(\mathsf{r5})$$

The rules r1 and r2 (resp r5 and r4) are dual and they correspond to the transmission and the reception of an input (resp. output). The rule r3 corresponds to the time elapsing. The time elapsing operation on a queue is defined by $((\delta \cdot a).p) + t = (\delta + t, a).(p + t)$. Notice that $(\!|A|\!)^{\bowtie\Delta}$ is input-complete because the transmissions of the inputs do not require to check the clock constraints. The receptions of the pending inputs require to check for the validity of clock constraints. The length of each queue is unbounded. Based on $(\!|A|\!)^{\bowtie\Delta}$, we can define asynchronous runs, asynchronous execution sequences and asynchronous timed traces in $\mathsf{ATTraces}_{\bowtie\Delta}(A) = \mathsf{TTraces}((\!|A|\!)^{\bowtie\Delta})$.

On asynchronous tioco. Given an implementation $\mathcal{I}$ and a specification $\mathcal{S}$ modelled with a TIOA(I,O), one could try to adapt the relation tioco based on the asynchronous semantics. Such an adaptation has been studied in [10] for untimed systems. A quick adaptation of tioco that we call $\mathsf{atioco}_{\bowtie\Delta}$ can be defined as follows:

$$\mathcal{I} \ \mathsf{atioco}_{\bowtie\Delta} \ \mathcal{S}$$
iff
$$\forall \sigma \in \mathsf{ATTraces}_{\bowtie\Delta}(\mathcal{S}), \mathsf{out}((\langle\!|\mathcal{I}|\!\rangle^{\bowtie\Delta}) \ \mathsf{after} \ \sigma) \subseteq \mathsf{out}((\langle\!|\mathcal{S}|\!\rangle^{\bowtie\Delta}) \ \mathsf{after} \ \sigma).$$

Designing a remote testing algorithm as a simple adaptation of the local and synchronous testing algorithm with asynchronous traces can be the source of differences between local testing verdicts and remote testing verdicts. The following example highlights three relevant problems: the non preservation of conformance, the permissiveness and the lack of control during the test.

Permissiveness. The relation $\mathsf{atioco}_{\bowtie\Delta}$ is permissive in the way that there exists an implementation $\mathcal{I}$ of a specification $\mathcal{S}$, a delay Δ and $\bowtie \in \{<=, =\}$ such that $\neg(\mathcal{I} \ \mathsf{tioco} \ \mathcal{S})$ but $\mathcal{I} \ \mathsf{atioco}_{\bowtie\Delta} \ \mathcal{S}$.
For example, consider Fig. 1 and $2FIFO(=, 2)$. We check that $Imp_3 \ \mathsf{atioco}_{\bowtie\Delta} \ Spec$ because $\mathsf{out}(\langle\!|Imp_3|\!\rangle^{\bowtie\Delta} \ \mathsf{after} \ (1{\cdot}?a_1){\cdot}2) = \{!b_5\} \cup \mathbb{R}_{\geq 0}$ and $\mathsf{out}(\langle\!|Spec|\!\rangle^{\bowtie\Delta} \ \mathsf{after} \ (1{\cdot}?a_1){\cdot}2) = \{!b_5\} \cup \mathbb{R}_{\geq 0}$. But, as we discussed earlier, $\neg(Imp_3 \ \mathsf{tioco} \ Spec)$.

Non preservation of conformance. The relation $\mathsf{atioco}_{\bowtie\Delta}$ does not preserve the conformance in the way that there exists an implementation $\mathcal{I}$ of a specification $\mathcal{S}$, a delay Δ and $\bowtie \in \{<=, =\}$ such that $\neg(\mathcal{I} \ \mathsf{atioco}_{\bowtie\Delta} \ \mathcal{S})$ but $\mathcal{I} \ \mathsf{tioco} \ \mathcal{S}$.
For example, consider Fig. 1 and $2FIFO(=, 2)$. Because $\mathsf{out}(\langle\!|Imp_1|\!\rangle^{\bowtie\Delta} \ \mathsf{after} \ (0{\cdot}?a_1){\cdot}1) = \{!b_2\} \cup \mathbb{R}_{\geq 0}$ and $\mathsf{out}(\langle\!|Spec|\!\rangle^{\bowtie\Delta} \ \mathsf{after} \ (0{\cdot}?a_1) \cdot 1) = \mathbb{R}_{\geq 0}$ it comes that $\neg(Imp_1 \ \mathsf{atioco}_{\bowtie\Delta} \ Spec)$. But we can check that $Imp_1 \ \mathsf{tioco} \ Spec$.

Controllability of the test. We say that a specification is *controllable* if in the case a tester observes an input, this means that any output it has sent before has already been received by the implementation. When performing remote testing, signal propagating delay needs to be managed, especially when the tester sends the inputs early and receives the outputs lately. An output received after the transmission of an input it does not depend on forces the tester to change the test case it was executing. For example, consider the specification in Figure 1, and assume that the implementation is the same as the specification. Also assume that we want to test the path up to ℓ_7 through ℓ_6. For that purpose a tester may observe the asynchronous timed trace $\sigma'_1 = (1{\cdot}?a_1) \cdot (2{\cdot}?a_2) \cdot (12{\cdot}?a_3) \cdot (13{\cdot}!b_2) \cdot (14{\cdot}!b_3) \cdot 0$. This trace means that the tester sends $?a_3$ before it receives $!b_3$. But, the implementation can change the test purpose by sending $!b_4$ at the time 13. So a tester should remind that it already has sent $?a_3$ and it should pursue the test by following a new trajectory where $?a_3$ follows $!b_4$.

It is not reasonable that testing verdicts vary depending on the distance and the communication mode between tester and implementation. In order to return correct verdict whatever the distance and the communication mode, we can equip the implementations with an additional mechanisms, like logical stamping mechanism [10], that will help the testers to recover the causal order of the interleaved actions. But, that mechanism cannot allow the remote testers to control the test.

4 Input/Output Interleaving and Δ-Testability Criterion

Asynchronous timed traces are remote observations of local timed traces executed by the implementation. The execution order of actions may differ from the observation

order: this happens when inputs and outputs interleave in the communication channels. We intend to characterize remote observations that may lead to action interleaving. Thanks to the timing information, we introduce Δ-testable specifications of which asynchronous traces can be used for remote testing without using costly mechanisms.

4.1 Local Timed Traces and Action-interleaving in Asynchronous Timed Traces

We address the derivation of local timed traces from asynchronous traces. Let $\mathcal{S} \in \text{TIOA}(I, O)$ and let $\rho = (\alpha_i)_{i=1..n}$ in $Seq(\langle\!|\mathcal{S}|\!\rangle^{\bowtie\Delta})$ be an asynchronous execution sequence. Each occurrence of a silent action τ_a in an asynchronous execution sequences can be interpreted as the reception/transmission of input/output a. For $\rho[i] = \alpha_i \in I \cup O$, let us denote by $\zeta_{\rho[i]}$ the unique silent action associated with the visible action $\rho[i]$, when it exists. Notice that either $\rho[i]$ is an input action and $\rho[i] \prec_\rho \zeta_{\rho[i]}$ or $\rho[i]$ is an output action and $\zeta_{\rho[i]} \prec_\rho \rho[i]$: this is because the actions in a queue are delivered according to their positions in the queue. Moreover, $\zeta_{\rho[i]} \prec_\rho \zeta_{\rho[j]}$ whenever $\rho[i], \rho[j]$ are both either inputs or outputs and $\rho[i] \prec_\rho \rho[j]$. We say that ρ is *regular* if for every $\rho[i] \in I \cup O$, $\zeta_{\rho[i]}$ exists in ρ. *A regular asynchronous timed trace* is constructed from a regular asynchronous execution sequence. The local execution sequence associated with a regular asynchronous execution sequence $\rho = (\alpha_i)_{i=1..n}$, denoted by $\mathsf{apply}(\rho)$, is the timed word obtained from ρ by deleting the visible actions in $I \cup O$ and replacing each silent action $\zeta_{\rho[i]}$ $(1 \leq j \leq n)$ with the corresponding visible action $\rho[i]$.

Example 4. Let $\rho = 1{\cdot}?a_1 \cdot 1{\cdot}?a_2 \cdot 1 \cdot \tau_{a_1} \cdot 1 \cdot \tau_{a_2} \cdot 5 \cdot 2 \cdot \tau_{b_2} \cdot 1 \cdot \tau_{b_3} \cdot 0{\cdot}?a_3 \cdot 1{\cdot}!b_2 \cdot 0.6 \cdot 0.4{\cdot}!b_3 \cdot 0 \cdot \tau_{a_3}$ be in $\mathsf{ATTraces}_{\bowtie\Delta}(Spec)$. We have that $|\rho| = 22$. We have that $\zeta_{\rho[2]} = \rho[6] = \tau_{a_1}$ because τ_{a_1} corresponds to the remote execution of $?a_1$ that is transmitted at the time 1. $\zeta_{\rho[17]} = \rho[11] = \tau_b$ because this occurrence of τ_{b_2} corresponds to the transmission of $!b_2$ and $!b_2$ is observed later at the position 17. We can check that $\zeta_{\rho[15]} = \rho[22] = \tau_{a_3}$. Then $\mathsf{apply}(\rho) = 1{\cdot}1{\cdot}1?a_1 1{\cdot}?a_2{\cdot}5{\cdot}2{\cdot}!b_2{\cdot}1{\cdot}!b_3{\cdot}0{\cdot}1{\cdot}0.6{\cdot}0.4{\cdot}0{\cdot}?a_3$. We can also compute $\mathsf{ttrace}(\rho) = (1{\cdot}?a_1) \cdot (2{\cdot}?a_2) \cdot (12{\cdot}?a_3) \cdot (13{\cdot}!b_2) \cdot (14{\cdot}!b_3) \cdot 0$ and $\mathsf{ttrace}(\mathsf{apply}(\rho)) = (3{\cdot}?a_1) \cdot (4{\cdot}?a_2) \cdot (11{\cdot}!b_2) \cdot (12{\cdot}!b_3) \cdot (14{\cdot}?a_3).0$. We remark that $?a_3$ occurs before $!b_2$ in $\mathsf{ttrace}(\rho)$ but $!b_2$ occurs before $?a_3$ in $\mathsf{ttrace}(\mathsf{apply}(\rho))$ which is a timed traces of $Spec$.

The causal order of the action in an asynchronous timed trace may not be respected by a remote implementation. The order of execution of two actions may be inverted by the remote SUT. For example this situation happens in asynchronous sequence ρ that contains a pattern[1] of the form $\cdots \rho[i] \cdots \zeta_{\rho[j]} \cdots \zeta_{\rho[i]} \cdots \rho[j] \cdots$ where $\rho[i]$ and $\rho[j]$ are visible actions and $i < j$. When such a situation happens, we say that ρ is action-interleaving. For example, ρ presented in Example 4 is action-interleaving.

Definition 6. *A regular sequence $\rho \in Seq(\langle\!|\mathcal{S}|\!\rangle^{\bowtie\Delta})$ is* action interleaving *if there exists i, j s.t $\rho[i] \prec_\rho \rho[j]$ and $\zeta_{\rho[j]} \prec_\rho \zeta_{\rho[i]}$.*

Proposition 1 states that the causal order of the actions in a non interleaving asynchronous trace is preserved at the implementation site. Let us denote by $\mathsf{Proj}_{vis}(\rho)$ the projection of ρ over $I \cup O \cup \mathbb{R}_{\geq 0}$. Given a state $s = ((l, v), p, q)$ of $\langle\!|A|\!\rangle^{\bowtie\Delta}$, let us denote by $p(s) = p$ and $q(s) = q$ the content of the input and output queues at s.

[1] Note that there are three more patterns.

Proposition 1. *Let $\rho = (\alpha_i)_{i=1..n}$ in $Seq(\langle\!|\mathcal{S}|\!\rangle^{\bowtie\Delta})$ be a regular execution sequence. ρ is not action-interleaving iff $\mathsf{Proj}_{vis}(\mathsf{apply}(\rho)) = \mathsf{Proj}_{vis}(\rho)$.*

Proposition 2. *Let $\rho = (\alpha_i)_{i=1..n} \in Seq(\langle\!|\mathcal{S}|\!\rangle^{\bowtie\Delta})$ be a regular asynchronous execution sequence. ρ is action-interleaving iff $p(s)$ and $q(s)$ are non empty for some state s, some $k \leq n$ such that $s^0 \xrightarrow{\rho[1..k]} s$.*

4.2 Δ-Testable TIOA

We provide a Δ-testability criterion permitting to test remotely while preserving properties of local testing. Action-interleaving does not occur in Δ-testable specifications.

Definition 7 (Δ-testability). *Let $A \in \mathrm{TIOA}(I, O)$ and $\sigma \in \mathsf{TTraces}(A)$ such that $\sigma = (t_i \cdot a_i)_{i=1..n}.t_{n+1}$. The timed trace σ is Δ-testable if,*

- *either $n = 0$,*
- *or $(t_i \cdot a_i)_{i=1..n-1}$ is Δ-testable and $a_n \in O$,*
- *or $(t_i \cdot a_i)_{i=1..n-1}$ is Δ-testable and if $a_n \in I$, then for every $t_b \in \mathbb{R}_{\geq 0}$, every $b \in O$, and every $k \in [1..n-1]$ such that $!b \in \mathsf{out}(\llbracket A \rrbracket \text{ after } \sigma[1..k] \cdot t_b)$, it holds that $t_n - t_b > 2\Delta$.*

A is Δ-testable if every $\sigma \in \mathsf{TTraces}(A)$ is Δ-testable.

Example 5. $Spec$ is 1-testable. $Spec$ is not 2-testable. Indeed, one can consider the sub-specification rooted at ℓ_4. The delay between $!b_4$ and $?a_3$ equals 1 and it is not greater than $2 \times \Delta = 4$. The specification obtained in Fig. 1d that is obtained from $Spec$ by changing some constants is 2-testable.

The causal order of the observed actions is the same as the causal order of the actions executed by the remote implementation when the specification is Δ-testable.

Proposition 3. *Let A be a $\mathrm{TIOA}(I, O)$. If A is Δ-testable then $Seq(\langle\!|A|\!\rangle^{\bowtie\Delta})$ contains no action-interleaving sequence.*

Putting Proposition 2 and Proposition 3 together, we get Proposition 4.

Proposition 4. *Let A be a $\mathrm{TIOA}(I, O)$. Let s, $\rho \in Seq(\langle\!|A|\!\rangle^{\bowtie\Delta})$ such that $s^0 \xrightarrow{\rho} s$. A is Δ-testable iff $p(s)$ is non empty implies $q(s)$ is empty.*

According to Proposition 4, Δ-testability implies that at most one queue is non empty at every reachable state. However, Δ-testability does not guarantee that the sizes of the queues are bounded. A fast environment can increase the size of the input queue by sending repetitively the inputs faster than the latency.

We can show that Δ-testable specifications are controllable. Indeed, Δ-testable specifications have no action-interleaving sequences. Consequently a regular asynchronous timed trace ρ is such that $\rho[i] \prec_\rho \rho[j]$ iff $\zeta_{\rho[i]} \prec_\rho \zeta_\rho[j]$ for every $1 \leq i, j \leq |\rho|$. W.l.o.g, assume that $\rho[i] \in O$ and $\rho[j] \in I$. Then, $\zeta_{\rho[i]} \prec_\rho \rho[i]$, $\rho[j] \prec_\rho \zeta_{\rho[j]}$. Since the specification is Δ-testable, the delay between $\zeta_{\rho[i]}$ and $\zeta_{\rho[j]}$ is strictly greater than 2Δ. But since the delay between $\rho[k]$ and $\zeta_{\rho[k]}$ with $k \in \{i, j\}$ is bounded with Δ, we get the

delay between $\rho[i]$ and $\rho[j]$ is strictly positive. This implies that the output $\rho[i]$ is observed before the input $\rho[j]$. This means that the outputs transmitted earlier are received before the transmission of new inputs. Thus, each observed output depends on input transmitted earlier and the specification is controllable.

In brief, Δ-testability criterion takes advantage of the timing information that are not available in untimed models. We claim that if the specification is Δ-testable then, the asynchronous execution of the synthesized test cases is as simple as the synchronous execution, the tioco conformance is preserved and the tester can control the test.

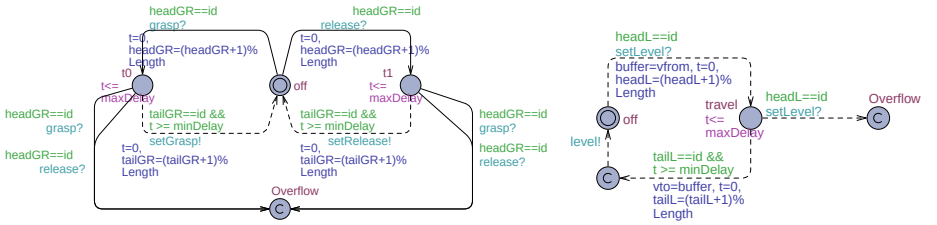
5 Remote testing Framework with Uppaal-TIGA

In this section we present our general framework using Uppaal-TIGA with partial observability [16]. We model the SUT, the communication channels, and the actual tester as a timed game with the twist that only some states or clocks are visible. The tester changes its states according to the output from the SUT (via the delayed FIFOs) and the goal is that given a test objective expressed as a formula (using an extra observer automaton or not), find a strategy using the actions of the tester and a fixed set of observations to reach that objective. This matches the situation that the tester can only observe the delayed output from the SUT and cannot see its state. The framework is an extension of [17].

Modelling Pattern. Fig. 2 presents our modelling pattern. The originality of the model is how the FIFOs are encoded. We want to transmit a message with optional data with a delay that may be non-deterministic. In general, this may change the order of the outputs of the "FIFO" if the delays overlap. Each *cell* of the FIFO buffer is modeled as an automaton with its own identifier (*id*) as shown in Fig. 2. Only the automaton with the right identifier that matches the head of the FIFO (a global variable) reacts to the communication. The head and tail of the FIFOs are simple counters managed by the automata. Each automaton has its own clock to delay the output of the incoming message. We rename the channels to do the delayed transmission. For our light controller example of Section 6, *grasp* becomes *setGrasp*, and *release setRelease*.

Then we compose the SUT in parallel with the FIFOs and the tester. The tester automaton is free to generate outputs with possibly some constraints. The next step is to solve the game to decide which outputs should be generated, and when.

Solving the Game. To generate the test, Uppaal-TIGA solves a two-player game between the tester and the implementation. The implementation (together with the FIFO) plays *uncontrollable* transitions and the tester plays *controllable* transitions. In addition, observations together with the test purpose are specified. To play the game an *action label* is associated with the transitions and the tester plays one given controllable action until its *observation changes*. In the meantime, the implementation can play its uncontrollable transitions. It is only when an observation changes that the tester can change its action. We refer to [16] and [17] for more details. The result is that Uppaal-TIGA will find a strategy for the tester to fulfil the test purpose under the specified observations *iff* there exists such a strategy.



(a) Automaton that delays *grasp* and *release*.

(b) Automaton that delays *level* and its value.

Fig. 3: Automata for the FIFOs.

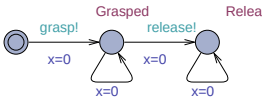
POCO Conformance. Uppaal Tiga with partial observability [17] assumes poco_P conformance relation constructed similarly to tioco, except that in addition to outputs the observations also contain a partial information about the system state defined by a set of predicates P. In theory the discrete changes in the partial state observation can be identified as special outputs and therefore emulated by tioco. In general, poco_P is most useful to relate to the SUT as a continuous observation of its partial state which might be difficult to achieve in practice. In this paper we assume that only the state of environment (the model of test assumptions or a tester) is observable and thus only observable I/O is communicated with the black-box SUT and media and therefore tioco is sufficient for our purposes.

6 Light Controller Example

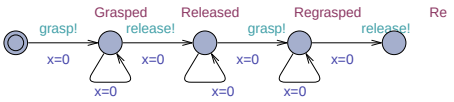
To apply our remote testing framework we consider the example of a light controller [18]. A user can grasp or release a trigger rod. Grasping and holding makes the intensity of the light vary. Grasping and releasing have the effect of switching off or on to the previous light level.

Encoding Delayed Communications. We specialize the FIFOs presented in Section 3 to send *grasp* and *release* to the SUT, and *level* together with a value to the tester. Fig. 3a shows the automaton used to delay *grasp* and *release*, and Fig. 3b the one to delay *level* with the value of the light level. The pattern for both automata is that upon synchronization on a given channel, a transition is taken to a state where the delay occurs and then a renamed output is produced. Data (Fig. 3b) may be stored and forwarded thanks to a local buffer.

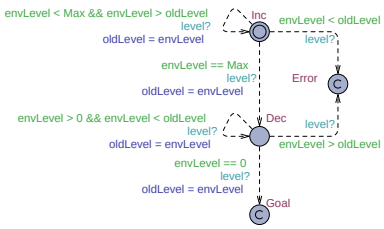
SUT, *Tester, and Test Purpose.* The light controller has an interface that receives the grasp and release commands. It controls two components to respectively dimmer or switch on or off the light. The actual details of the SUT are not important here since we are doing black-box testing. The internal communication is not visible to the tester or the FIFOs. The tester is an automaton that can generate *grasp* or *release* at any time. We can constrain the outputs and to illustrate this, we use two types of testers to generate test strategies. Our testers are shown in Fig. 4a and 4b. They restrict the tests to one or two grasp and release. The test purpose is a monitor automaton put in parallel together with the tester automaton to specify interesting sequences of outputs that we want to observe. Fig. 4c specifies that the light level should increase to its



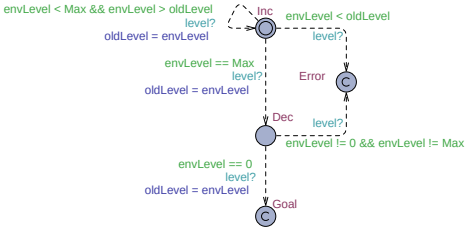
(a) Tester that can generate one grasp and release.



(b) Tester that can generate two grasps and releases.



(c) Test purpose to increase and then decrease the intensity of the light.



(d) Test purpose to increase the intensity of the light and then switch it off.

Fig. 4: Tester and test purpose automata.

maximum level and then decrease monotonically. Fig. 4d specifies that the light level should increase monotonically to its maximum level and then be switched off.

Playing the Game to Generate Tests. To generate the tests, Uppaal-TIGA solves a two-player game between the tester and the implementation. In the automata shown for the tester, purpose, and the FIFO, the *uncontrollable* transitions played by the implementation are *dashed*. The *controllable* transitions played by the tester are not dashed. In addition, we need to specify what is observable, which is done together with the formula giving the test purpose.

We specify the following test purposes:

1. `{user.x>=0 && user.x<1} control: A[ forall(s:slot_t) !adapterGR(s).Overflow U user.Released and envLevel==Max ]`
2. `{user.x>=0 && user.x<2, envLevel==Max, envLevel==0} control: A[ !purpose.Error && forall(s:slot_t) !adapterGR(s).Overflow U purpose.Goal ]`

Purpose 1 specifies to turn the light on to its maximum intensity level without having a buffer overflow in the FIFO[2]. In addition, the user must have released the trigger. We do not need an extra automaton for this purpose. To achieve this, the user has a clock x that can be reset (Fig. 4a or 4b) and can observe if $x \in [0, 1[$ or not. In addition, overflow and the released state together with the maximum light intensity are observable[3].

Purpose 2 specifies that the goal state of our monitor automaton should be reached while avoiding overflow or the error state in the monitor. To do so the user can observe if his clock $x \in [0, 2[$ or not, if the light is at its maximum level (or not), or if it is switched off (or not). This can be checked for both our purpose automata, though we need the user of Fig. 4b to fulfil the goal of the purpose of Fig. 4d.

[2] We model-checked that the 2^{nd} FIFO cannot overflow.

[3] The winning and losing conditions are always implicitly observable.

It is important to notice that the observations that are given are only from the tester's side and we do not see the internal state of the SUT, thus respecting the black-box testing principle. We show one strategy generated in a few second[4] for purpose 2 with a deterministic communication delay of 4 time units. We sanitized and minimized it (the raw output has 16 states).

State 0: GRASP until $x \notin [0, 2[$. Goto state 1.
State 1: delay until $envLevel \neq 0$. Goto state 2.
State 2: RESET until $x \in [0, 2[$. Goto state 3.
State 3: RELEASE until $x \notin [0, 2[$. Goto state 4.
State 4: GRASP until $x \in [0, 2[$. Goto state 5.
State 5: delay until $x \notin [0, 2[$. Goto state 6.
State 6: RESET until either $x \in [0, 2[$ and then goto state 7
or $envLevel = Max$ and then goto state 12.
State 7: RESET until $envLevel = Max$. Goto state 8.
State 8: RELEASE until $x \notin [0, 2[$. Goto state 9.
State 9: delay until $envLevel = 0$ and $envLevel \neq Max$. Goto state 10.
State 10: delay until $purpose.Goal$. Goto state 11.
State 11: $envLevel = 0$ and $purpose.Goal$, goal reached.
State 12: RESET until $x \in [0, 2[$. Goto state 8.

Δ-Testability The model is general and does not enforce minimal delays between inputs and outputs. We can constrain the environment model or add another purpose automaton to constrain the strategy. For example, if the delay between *grasp* and *release* exceeds the longest duration for registering a touch, then there is no strategy to satisfy purpose 2. This delay is the Δ of our example.

7 Conclusion

We addressed conformance testing of remote SUTs specified with timed input/output automata. Our testing architecture is composed of two queues with a communication latency threshold. Testers and SUTs communicate in an asynchronous way. We introduced the Δ-testability criterion allowing remote testing to be as powerful as local testing without any additional mechanism. The Δ-testability criterion ensures that input/output interleaving never occurs, controllability of the test and a remote verdict similar to local one. Then we presented a test selection approach with the partial observability timed game solver Uppaal-TIGA. The method has consisted in modelling the queues with new TIOA that receive and delay the actions. Then the test generation was reduced to synthesis of winning strategies in the game provided that the sizes of the queues are bounded. However the limitation of the size of the queue restricts the number of consecutive inputs/outputs the tester/SUT may send within the period of the latency threshold. Moreover, using one clock per cell leads to exponential blow up during the generation of the test cases whether the latency is deterministic or not.

We believe that testing Δ-testable criterion can be performed in a more efficient way and with less constraints on the size of the queues. Promising results hold in case of deterministic latencies. Further works include the design of dedicated testing algorithms for Δ-testable specifications and the automatic verification of the Δ-testability criterion.

[4] Using the pre-release version 0.17.

References

1. Mammeri, Z.: Introduction au langage de description et de spécification (sdl). Technical report, Université Paul Sabatier - Toulouse (2001)
2. ISO: Information processing systems – open systems interconnection – LOTOS – a formal description technique based on the temporal ordering of observational behaviour ISO/TC97/SC21/N DIS8807. (1987)
3. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software - Concepts and Tools **17** (1996) 103–120
4. Jard, C., Jéron, T.: TGV: theory, principles and algorithms. International Journal on Software Tools for Technology Transfer **7** (2005) 297–315
5. Núñez, M., Rodríguez, I.: Conformance testing relations for timed systems. In: 5th International Workshop on Formal Approaches to Software Testing, Revised Selected Papers. Volume 3997 of Lecture Notes in Computer Science., Springer (2005) 103–117
6. Krichen, M., Tripakis, S.: Black-box conformance testing for real-time systems. In: 11th International SPIN Workshop on Model Checking Software. Volume 2989 of Lecture Notes in Computer Science., Springer (2004) 109–126
7. Mikučionis, M., Larsen, K.G., Nielsen, B.: T-uppaal: Online model-based testing of real-time systems. In: 19th IEEE International Conference on Automated Software Engineering, IEEE Computer Society (2004) 396–397
8. Hessel, A., Larsen, K.G., Mikučionis, M., Nielsen, B., Pettersson, P., Skou, A.: Testing real-time systems using uppaal. In: Formal methods and testing. Springer (2008) 77–117
9. Bertrand, N., Jéron, T., Stainer, A., Krichen, M.: Off-line test selection with test purposes for non-deterministic timed automata. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer (2011) 96–111
10. Jard, C., Jéron, T., Tanguy, L., Viho, C.: Remote testing can be as powerful as local testing. Formal Desciption Techniques and Protocol Specification, Testing and Verification FORTE XI/PSTV XVIII **99** (1999)
11. Simao, A., Petrenko, A.: Generating asynchronous test cases from test purposes. Information and Software Technology **53** (2011) 1252–1262
12. Noroozi, N., Khosravi, R., Mousavi, M.R., Willemse, T.A.: Synchronizing asynchronous conformance testing. In: Software Engineering and Formal Methods. Springer (2011) 334–349
13. Henniger, O.: On test case generation from asynchronously communicating state machines. In Kim, M., Kang, S., Hong, K., eds.: Testing of Communicating Systems. IFIP — The International Federation for Information Processing. Springer (1997) 255–271
14. De León, H.P., Haar, S., Longuet, D.: Conformance relations for labeled event structures. In: Tests and Proofs. Springer (2012) 83–98
15. Hierons, R.M., Merayo, M.G., Núñez, M.: Using time to add order to distributed testing. In: FM 2012: Formal Methods. Springer (2012) 232–246
16. Cassez, F., David, A., Larsen, K.G., Lime, D., Raskin, J.F.: Timed control with observation based and stuttering invariant strategies. In: Proceedings of the 5th International Symposium on Automated Technology for Verification and Analysis. Volume 4762 of LNCS., Springer (2007) 192–206
17. David, A., Larsen, K.G., Li, S., Nielsen, B.: Timed testing under partial observability. In: 2nd IEEE International Conference on Software Testing, Verification, and Validation, IEEE Computer Society (2009) 61–70
18. Kim G. Larsen, Marius Mikučionis, B.N.: Uppaal TRON User Manual. CISS, BRICS, Aalborg University, Aalborg, Denmark. (2009)