A Boundary Checking Technique for Testing Real-Time Systems Modeled as Timed Input Output Automata

Abdeslam En-Nouaary[1] and Abdelwahab Hamou-Lhadj
Department of Electrical and Computer Engineering
Concordia University
1515 Sainte Catherine West
Montréal, Québec H3G 2W1, Canada
{ennouaar, abdelw}@ece.concordia.ca

Abstract

The behavior of real-time systems depends not only on their interaction with the environment but also on very rigid time constraints that puts restrictions on when these interactions take place. The timing aspect of such systems renders the testing process difficult without defining adequate test selection criteria that ensure good coverage of the system while keeping the number of needed test cases considerably low. In this paper, we propose a method for testing real-time systems, formally modeled as Timed Input Output Automata (TIOA), which aims at generating a set of test cases that would allow us to check every transition of the TIOA as soon as possible, as late as possible, and at the middle between these two executions. The execution times of every transition are determined based on the minimum and maximum delays between the source state of the transition and its clock guards.

1. Introduction

Modern society increasingly depends on complex real-time software systems for operating its critical infrastructures including telecommunication networks, health care devices, transportation, etc. Unlike other types of systems, the behavior of real-time systems depends not only on their interactions with the environment but also on the time at which these interactions take place, which renders testing such applications harder compared to their counterparts.

The normal testing process consists of generating test cases from the specification of the system, which are then applied to the implementation under test (IUT). If the reactions of the IUT match the expected outputs (i.e., those derived from the specification), the implementation is considered correct, otherwise the implementation is declared faulty and the diagnosis process should be started in order to locate and fix the fault. In the case of real-time systems, a test case consists of not only a sequence of inputs and outputs, but also the times of occurrence of these events. The goal is to ensure that the implementation of the IUT respects the timing constraints described in the problem specification at both input and output levels.

An important aspect in testing a software system is to carefully design test cases that would uncover as many faults as possible. The power of a test cases generation technique is referred to as *fault coverage*. In the case of real-time systems, the fault model consists of four types of faults: output faults, transfer faults, clock guard restriction faults, and clock guard widening faults.

In this paper, we present a new approach for black-box testing of real-time systems that achieves the aforementioned fault coverage while minimizing the number of test cases that are needed. Our approach is based on modeling a real-time system as a Timed Input Output Automata (TIOA), which is a variant of timed automaton in which clocks, real-valued variables, increase synchronously at the same speed, and measure the amount of time elapsed since last initialization [1]. In our approach, we generate test cases from the TIOA, which allow every transition to be tested at three different instants, namely: immediately when the transition becomes executable, immediately before the transition is complete, and between these two points of executions.

In our approach, time is represented using a dense time model [2]. Unlike discrete time models that require the time sequence to be a monotonically increasing sequence of integers (i.e., input and output messages take place at time instants, which are integer

[1] This author is on leave at Institut National des Postes et Télécommunication, Rabat, Morocco.

values), dense time models require input and output messages to take place at time instants, which are real number values [2].

The remainder of this paper is structured as follows. Section 2 introduces the TIOA model and related concepts. Section 3 presents the proposed approach for timed test cases, followed with related work. In Section 4, we conclude the paper and point out to key future directions.

2. Background

This section presents the definitions and the theoretical ingredients related to TIOA.

Definition 1: TIOA

A TIOA is a 6−tuple (*I, O, L, l_0, C, T*), where :

- *I* is a finite set of inputs. Inputs represent the messages received by the system from the environment. In this paper, inputs are preceded by *"?"*.
- *O* is a finite set of outputs. Outputs represent the messages sent by the system to the environment. In this paper, outputs are preceded by *"!"*.
- *L* is a finite set of locations. A location represents the "status" of the system after the execution of a transition. The term location is used instead of the term "state"' because the latter is used to define the operational semantics of TIOA (see def. 3).
- $l_0 \in L$ is the initial location.
- *C* is a finite set of clocks, all initialized to zero in l_0. A clock is a time variable that counts how much time has elapsed since the clock was initialized. The time domain of these clocks is dense, which means that the values of the clocks are non-negative real numbers.
- $T \subseteq L \times (I \cup O) \times \phi(C) \times P(C) \times L$ is the set of transitions.

A transition in TIOA, denoted by $t : l \xrightarrow{m,G,R} l'$, consists of a source location *l* (i.e., *source(t) = l*), an input or output message *m*, a clock guard (or time constraint) *G*, which should hold to execute the transition, a subset of clocks *R* to be reset when the transition is fired, and a destination location *l′* (i.e., *destination(t) = l′*). Each clock in *R* is used to record, when not reinitialized to zero, how much time has elapsed since the execution of the transition. Such clocks are mainly used to set clock guards between the transition where they are reset and future transitions. We assume that transitions in TIOA are instantaneous (i.e., they don't take time to execute). Also, transitions on inputs are supposed to be guarded with conjunctions of atomic formulas of the form *x op b*, where $x \in C, op \in \{<, \leq, =, >, \geq\}$ and *b* is a natural number, while transitions on outputs are supposed to be guarded with conjunctions of atomic formulas of the form *x op b*, where $x \in C, op \in \{\leq, =, \geq\}$ and *b* is a natural. The tester needs to know the exact time when outputs are expected because they are not controllable by him. In definition 1 above, *Φ(C)* and *P(C)* denote the set of clock guards and the power set of *C*, respectively. Multiple clocks are used in TIOA to express time constraints between more than two transitions. Moreover, we suppose that each clock $x \in C$ has a bounded domain $[0, B_x] \cup \{\infty\}$ [3], where B_x is the largest integer constant appearing in time constraints over *x* in the automaton. This means that each clock *x* is relevant only under the integer constant B_x, and all the values of *x* greater than B_x are represented by ∞. Hence, the following equations hold:

$$\forall \varepsilon > 0, B_x + \varepsilon = \infty \text{ and } \forall \varepsilon > 0, \infty + \varepsilon = \infty.$$

For a clock *x* and a clock guard *G* of a transition in TIOA, we define the projection of *G* over *x*, denoted by *Proj(G, x),* by the condition *(x op b)* in *G*, obtained by removing the conditions over all clocks except *x*; if clock *x* is not involved in *G* then *Proj(G, x) = true*.

In Figure 1, we show an example of a TIOA that describes the behavior of a simple multimedia system. The system receives an *image*, followed by its *sound* within two time units, sends an acknowledgment, *ackAll*, in no more than three time units after the reception of the *image* and no more than two time units after *sound*, and then accepts the message *reset* and goes back to its initial location to await another image. If the input *sound* does not follow the input *image* within two time units, the system times out, issues the message *error*, and goes back to its initial location. The TIOA that describes the system has four locations *(l_0, l_1, l_2, and l_3)*, five transitions and two clocks *x* and *y*. The initial location of the system is l_0. The transition $l_0 \xrightarrow{?image, 0 \leq x < 2, \{x,y\}} l_1$ is executed when the system receives the message image and the value of clock *x* is less than 2. When the transition is fired, clocks *x* and *y* are both set to 0. In this example, the time domains of clocks *x* and *y* are ($[0,2] \cup \{\infty\}$) and ($[0,3] \cup \{\infty\}$), respectively. The projection of the clock guard ($0 \leq x < 2$) over clock *x* is ($0 \leq x \leq 2$) while its projection over clock *y* is true (i.e., the clock *y* is not involved in the clock guard). However, the projection

of the clock guard (($0 \le x < 2$) && ($1 \le y \le 2$)) over clock x is ($0 \le x \le 2$), while its projection over clock y is ($1 \le y \le 2$) (here, both clocks x and y are involved in the clock guard).

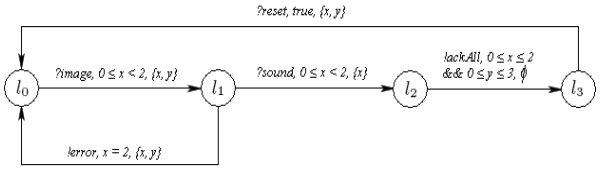

Figure 1. An Example of TIOA

The TIOA model introduced thus far is an abstract model because it does not explain the execution of the system it describes. The execution, also called the operational semantics of TIOA, can be informally stated as follows. The TIOA starts at its initial location with all clocks initialized to zero. Then, the values of the clocks increase at the same speed and measure the amount of time elapsed since the last initialization. At any time, the TIOA can execute a transition $l \xrightarrow{m,G,R} l'$ if the input/output message m takes place, its current location is l, and the values of its clocks satisfy the clock guard G. After this transition, all the clocks in R are reset and the TIOA changes its location to l'. To formalize the operational semantics of TIOA, we need to define the concepts of clock valuations and states for TIOA.

Definition 2: Clock valuations

Let $A = (I, O, L, l_0, C, T)$ be a n−clocks TIOA (i.e., an TIOA with n clocks), $R \ge 0$ be the set of non-negative real numbers.

- A clock valuation of A (or over C) is an application $v : C \to [R \ge 0 \cup \{\infty\}]^n$, which assigns a value (non-negative real number or ∞) to each clock $x \in C$. Clock valuation is simply the binding of clocks to their actual values. In this paper, a clock valuation is represented by a vector $(v_{x1}, v_{x2}, ..., v_{xn})$, where $v(x_i) = v_{xi}$ is the value of clock $x_i, 1 \le i \le n$. The set of all clock valuations for A is referred to by $V(C)$.
- For any clock valuation $v \in V(C)$ and any non-negative real number d, $v + d$ is a clock valuation that assigns the value $v(x) + d$ to each clock $x \in C$. $v + d$ is the clock valuation reached from v by letting time elapse by d time units.
- For any clock valuation $v \in V(C)$ and any subset of clocks $R \subseteq C$, $[R := 0]v$ is a clock valuation that assigns the value 0 to each clock $x \in R$ and $v(x)$ to any other clock y such that $y \in C$ and $y \notin R$. $[R := 0]v$ is the clock valuation obtained from v by resetting the clocks in R when a transition $l \xrightarrow{a,G,R} l'$ is executed.
- A clock valuation $v \in V(C)$ satisfies the clock guard G of a transition $l \xrightarrow{a,G,R} l'$, denoted by $v \models G$, if and only if G holds (i.e., G evaluated to true) under v.

Notice that a clock valuation is used to determine how much time has elapsed since the execution of each transition that has recently reinitialized a clock. The combination of a clock valuation and a location defines a state in TIOA. The formal definition of such states follows.

Definition 3: States of TIOA

Let $A = (I, O, L, l_0, C, T)$ be a TIOA.

- A state of A is a pair (l, v) consisting of a location $l \in L$ and a clock valuation $v \in V(C)$. Intuitively, a state of A is a configuration that indicates the current location of A and the current value of each clock used in A.
- The initial state of A is the pair (l_0, v_0), where $v_0(x) = 0$ for each clock $x \in C$. Intuitively, the initial state of A is the configuration of A in the beginning of its execution (i.e., the location is l_0 and all clocks are set to 0 as stated in definition 1).
- The set of states of A is denoted by $S(A)$.

Formally, the operational semantics of TIOA is described by a state machine $M = (S, s_0, A, T)$, where S is the set of states of TIOA, s_0 is the initial state, A is the set of actions, and T is the set of transitions. The actions of M are made up of the input and output messages of TIOA as well as time delays. There are two categories of transitions in M: explicit transitions on input and output messages, and implicit transitions on time delays. The explicit transitions are obtained from the transitions of TIOA and they describe the interactions of the system with its environment. The explicit transitions do not contain time constraints because the clock valuations of their source states do satisfy their clock guards. On the other hand, the delay transitions describe the progression of time but they do not appear in the transitions of TIOA. In the rest of this paper, we need the following notations.

- For a state *(l, v)* in *M* and a real-valued time delay *δ*, *M* contains the time delay transition on $\delta : (l, v) \xrightarrow{\delta} (l, v + \delta)$.
- For a state *(l, v)* in *M* and a transition $l \xrightarrow{m,G,R} l'$ in TIOA such that $v \models G$, *M* contains the explicit transition *(l, v)* $\xrightarrow{m}$ *(l', [R:=0]v)*.
- For two states *s* and *s′* in *M* and a sequence of input/output messages and time delays *σ*, we write $s \overset{\sigma}{\Rightarrow} s'$ if and only if *σ* brings *M* from *s* to *s′*.

3. Approach

This section introduces our approach for testing the TIOA, which ensures the checking of every transition of the TIOA at least once. To ensure good fault coverage, each transition is tested at three different points of time: at the boundaries of the transition (i.e., as soon as the transition becomes firable and right before the execution of the transition is no longer possible) as well as in the middle of the execution of the transition. Consequently, the number of test cases generated by the proposed method is at most equal to three times the number of transitions in the TIOA, which should make our approach scalable to large systems. To test a transition in TIOA, the method needs:

- A preamble that brings the TIOA from its initial location to the source location of the given transition. The preamble should be executable. That is, we should find instants in time that make it possible for each transition in the preamble to be fired.
- The input or output of the transition provided that the clock guard of the given transition is satisfied.
- A postamble that brings the TIOA from the destination location of the given transition back to its initial location. Similar to the preamble, the postamble should be executable.

Besides, we are making the following assumptions when generating test cases:

- The TIOA dealt with is deterministic; i.e, at any location of the TIOA we cannot find two or more transitions whose clock guards can simultaneously be satisfied.
- The implementation refuses any unexpected input by displaying an error message and going back to its initial state.
- The TIOA dealt with is connected; i.e., there is a path between any two locations in TIOA. If a location has no outgoing transition, then an implicit reset transition should be added to bring the TIOA back to its initial location.

To formalize the approach, let *t* be the transition to be tested. *Preambles(t)* denotes the set of preambles for *t*; *Preambles(t)* = *{(t_1, t_2, ..., t_n)* such that: *source(t_1)* = *l_0*, *destination(t_n)* = *source(t)*, and *destination(t_i−1)* = *source(t_i), 1 < i < n}*. Similarly, *Postambles(t)* denotes the set of postambles for *t*; *Postambles(t)* = *{(t_1, t_2, ..., t_m)* such that: *source(t_1)* = *destination(t)*, *destination(t_m)* = *l_0*, and *destination(t_i−1)* = *source(t_i), 1 < i < m}*.

So, to test t we should concatenate in this order the execution of a preamble from *Preambles(t)*, the input/output of t after satisfying its clock guard, and the execution of a postamble from *Postambles(t)*. To get the execution of a preamble *pr* (respectively a postamble *ps*) we put together the input/output messages of *pr* (respectively *ps*) and the time delays used to execute their transitions. The time delay used for a transition is the minimum waiting time required to execute the transition, which is denoted below by *$delay_{min}$(v,G)*. It is calculated based on the minimum delay between the value of a clock and a constraint on that clock, as follows:

$$delay_{\min}(v(x), \phi(x)) = \begin{cases} m_1 - v(x) + \varepsilon & \text{if } \phi(x) \text{ is } (m_1 < x < m_2) \text{ or } (m_1 < x \le m_2) \\ m_1 - v(x) & \text{if } \phi(x) \text{ is } (x = m_1) \text{ or } (m_1 \le x < m_2) \text{ or } (m_1 \le x \le m_2) \\ 0 & \text{if } \phi(x) \text{ is true} \end{cases}$$

Where *v(x)* is the value of clock x, $\phi(x)$ is the constraint on clock x, and ε is a small positive real value chosen by the tester, hence:

$delay_{min}(v,G) = Max_{x \in C}\{delay_{min}(v(x), Proj(G, x)) > 0\}$

The process of calculating the execution of a preamble is as follows. We start at the initial state *(l_0, v_0)*. Then, we determine the minimum time delay required to execute the first transition in the preamble. After that, we apply the message of this transition and update the state of the system.

Afterward, we determine the minimum delay for the second transition in the preamble, and so on until we reach the source state of *t* (which is also the destination state of the last transition in the preamble). Likewise, the execution of a postamble is computed as follows. We start at the destination state of the transition being tested. Then, we determine the minimum time delay required to execute the first transition in the postamble. After that, we apply the message of this transition and

update the state of the system. Afterward, we determine the minimum delay for the second transition in the postamble, and so on until we reach the initial state of the TIOA (which is also the destination state of the last transition in the postamble).

Next, to make it possible for a transition to execute, the clock guard of the transition should be satisfied by the current values of the clocks. This means that during test generation, the value of each clock should be chosen to satisfy the condition on the clock in the transition. In other words, we should find states in the state machine describing the operational semantics of the TIOA that make it possible for the transition to execute. As mentioned earlier, each transition is tested three different points of time:

- As soon as the transition becomes firable,
- Right before the execution of the transition is no longer possible, and
- In the middle of the execution of the transition.

This is done by using the minimum delay introduced above as well as the maximum delay defined below. The maximum delay between a clock valuation and a clock guard is written $delay_{max}(v,G)$, and is calculated based on the maximum delay between the value of a clock and a constraint on that clock:

$$delay_{\max}(v(x),\phi(x)) = \begin{cases} m_2 - v(x) - \varepsilon & if\ \phi(x)\ is\ (m_1 < x < m_2)\ or \\ & (m_1 \le x < m_2) \\ m_2 - v(x) & if\ \phi(x)\ is\ (x = m_2)\ or \\ & (m_1 < x \le m_2)\ or\ (m_1 \le x \le m_2) \\ \infty & if\ \phi(x)\ is\ true \end{cases}$$

Where $v(x)$ is the value of clock x, $\phi(x)$ is the constraint on clock x, and ε is a small positive real value chosen by the tester, hence:

$delay_{max}(v,G) = Min_{x \in C}\{delay_{max}(v(x), Proj(G, x)) > 0\}$

The minimum delay is used to execute the transition as soon as the transition is ready to execute while the maximum delay is used to test the transition at its latest convenience. The execution of the transition in the middle of its execution is determined by taking the average of the minimum and maximum delays between the clock valuation in the source state of the transition and its clock guard.

More formally, to test a transition $t : l \xrightarrow{m,G,R} l'$ in TIOA, we need three test sequences: $exec(pr).\delta_{min}.m.exec(ps)$, $exec(pr).\delta_{avg}.m.exec(ps)$, and $exec(pr).\delta_{max}.m.exec(ps)$ such that:

- $pr \in Preambles(t)$, $ps \in Postambles(t)$, $\delta_{min} > 0$, $\delta_{avg} > 0$ and $\delta_{max} > 0$,
- $(l_0, v_0) \overset{exec(pr)}{\Rightarrow} (l, v_{t_n})$,
- $(l, v_{t_n}) \xrightarrow{\delta \min} (l, v_{t_n} + \delta_{min})$,
- $(v_{t_n} + \delta_{min}) \models G$,
- $(l, v_{t_n}) \xrightarrow{\delta avg} (l, v_{t_n} + \delta_{avg})$,
- $(v_{t_n} + \delta_{avg}) \models G$,
- $(l, v_{t_n}) \xrightarrow{\delta \max} (l, v_{t_n} + \delta_{max})$,
- $(v_{t_n} + \delta_{max}) \models G$,
- $\delta_{min} = delay_{min}(v, G)$,
- $\delta_{max} = delay_{max}(v, G)$,
- $\delta_{avg} = \dfrac{\text{delay}_{\max}(v,G) + \text{delay}_{\min}(v,G)}{2}$
- $(l', v_{t_n} + \delta_{min})[R := 0]) \overset{exec(ps)}{\Rightarrow} (l_0, v_0)$,
- $(l', v_{t_n} + \delta_{avg})[R := 0]) \overset{exec(ps)}{\Rightarrow} (l_0, v_0)$,
- $(l', v_{t_n} + \delta_{max})[R := 0]) \overset{exec(ps)}{\Rightarrow} (l_0, v_0)$.

To illustrate the approach, let us consider again the TIOA in Figure 1. Every transition requires three test cases. The first one tests the transition as soon as it becomes executable. The second test case is for testing the transition in the middle, while the third one is for checking the transition at its latest point of execution. For instance, the transition $t2 : l_1 \xrightarrow{?sound, 0 \le x < 2, \{x\}} l_2$ is tested with the following test cases:

- $?image.?sound.2.!ackAll.?reset$
- $?image.\frac{3}{4}.?sound.2.!ackAll.?reset$
- $?image.\frac{3}{2}.?sound.\frac{3}{2}.!ackAll.?reset$

Here, the preamble used to reach the transition is t_1 while the postamble used to go back to the initial location of the TIOA is $t_3.t_4$. The three test cases are obtained by taking the execution of the preamble to reach the source state of the transition, the minimum/average/maximum delay between the state $(l_1, (0, 0))$ and the clock guard of the transition (i.e., $0 \le x < 2$), and the execution of the postamble to go from the destination state of the transition back to the initial state of the TIOA. For all of the three test cases, the execution of the preamble is the same, namely $?image$. The minimum delay and the execution of the postamble for the first test cases are 0 and $2.!ackAll.?reset$, respectively; the average delay and the execution of the postamble for the second test case

are $\frac{3}{4}$ and 2.!*ackAll.*?*reset*, respectively; The maximum delay and the execution of the postamble for the third test case are $\frac{3}{2}$ and $\frac{3}{2}$.!*ackAll.*?*reset*, respectively. It should be noted that the minimum and maximum delays are calculated here with $\varepsilon = \frac{1}{2}$.

4. Related Work

Over the last two decades, several algorithms have been developed for testing real-time systems. These methods differ from one another depending on the formal specification model used, the technique adopted for test generation, and the number of resulting test cases. In what follows, we give an overview of each of these methods and we compare them to the approach proposed in this paper.

In [5], Mandrioli et al. propose a tool for the generation of test cases from specifications given as temporal logic formulas extended with time measures. But, the generated test cases cover only integer values.

In [6], Clarke et al. introduce a framework for testing time constraints in real-time systems. They derive test cases from a specification described in the form of a constraint graph. These test cases satisfy some test criteria for real-time systems. However, the constraint graph is restricted to the description of the minimum and maximum allowable delays between the input/output events in the execution of a system, following Taylor's and Dasarathy's classification of timing requirements [13, 14].

In [9, 4], we presented a method for testing real-time systems modeled as timed automata. The proposed method (called the Timed Wp-Method) is an adaptation of the Wp-Method [17] for generating timed test cases from timed automata based on the state characterization technique. The test cases generated by this method have full fault coverage with regard to the fault model presented in [18]. However, the number of test cases generated by the timed Wp-Method is quite large for broader specifications. In [10, 11, 12], we presented another method for testing time input output automata with an acceptable number of test cases. The proposed method is based on sampling the TIOA of the specification to come up with a testable automaton, called Grid Automaton (GA). Although the approach is scalable, the construction of the GA and its traversal are quite time consuming.

In [7], Cardell et al. propose a technique to generate timed test cases from timed transition systems (TTSs). A TTS is a pair of values consisting of an initial state and a set of transition rules. To generate test cases, the authors first transform the TTS into a labeled transition system containing all the observable traces of the system, and then apply an adapted version of the W-method [15] to the resulting automaton. This approach has two drawbacks. First, the authors adopt a discrete time model that does not allow specification of a system with clocks assuming continuous values. Secondly, the number of generated test cases may be very large if the implementation has more states than the specification.

In [8], Khoumsi et al. present an approach and an architecture to test real-time protocols specified by timed input-output automata with discrete time domains (an extension to the continuous time domain is presented in [19]). Their approach consists of two steps. First, the timed specification is transformed into an equivalent untimed one using the established timer operations *Set(T, d)* and *Exp(T)*. Then, the Wp-method [17] is applied to the resulting automaton. The major drawback of this approach is that it may introduce undesirable non-determinism during testing and it may produce non-executable test cases.

5. Conclusion

In this paper, we presented an approach for testing real-time systems modeled as Timed Input Output Automata. Our approach ensures good fault coverage of the system by testing each transition of the system at least once. Indeed, for each transition, we generated test cases so as to test the transition at three different points of time: At each boundary of the transition (i.e., as soon as the transition is ready to be execute and right before its execution is completed) as well as in the middle of the execution of the transition. We believe that this approach does not only allow good coverage of the system but it will also help uncover faulty behaviors that occur at the boundaries of the transition and that go usually undetected.

As future directions, we intend to implement the approach proposed in this paper and experiment with it using real-world systems. The objective is to assess its effectiveness to detect faulty behaviors as well as its scalability when applied to large systems.

6. References

[1] R. Alur and D. Dill, "A Theory of Timed Automata", *Journal of Theoretical Computer Science*, 1994, pp. 183–235.

[2] D. Dill., "Timing Assumptions and Verification of Finite-State Concurrent Systems", *In Proc. of the*

International Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France, 1990, pp. 197–212.

[3] J. Springintveld and F. Vaandrager, "Minimizable Timed Automata", *In Proc. of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems*, Uppsala, Sweden, 1996, pp. 130–147.

[4] A. En-Nouaary, R. Dssouli, F. Khendek, and A. Elqortobi, "Timed Test Cases Generation Based on State Characterisation Technique" *In Proc. of the 19th Real-Time Systems Symposium*, Madrid, Spain, 1998, pp. 220–230.

[5] D. Mandrioli, S. Morasca, and A. Morzenti, "Generating Test Cases for Real-Time Systems from Logic Specifications", *ACM Transactions on Computer Systems*, 13(4), 1995, pp. 365–398.

[6] D. Clarke and I. Lee, "Automatic Generation of Tests for Timing Constraints from Requirements", *In Proc. of the 3th International Workshop on Object-Oriented Real-Time Dependable Systems*, Newport Beach, California, 1997, pp. 199-206.

[7] Rachel Cardell-Oliver and T. Glover, "A Practical and Complete Algorithm for Testing Real-Time Systems", *In Proc. of the International Conference on Formal Techniques for Real-Time Fault Tolerant Systems*, Lyngby, Denmark, 1998, pp. 251–261.

[8] A. Khoumsi, M. Akalay, R. Dssouli, A. En-Nouaary, and L. Granger, "An Approach for Testing Real-Time Protocols", *In Proc. of the the 12th IFIP International Conference on Testing of Communicating Systems*, Ottawa, Canada, 2000.

[9] A. En-Nouaary, R. Dssouli, and F. Khendek, "Timed Wp-Method: Testing Real-Time Systems", *IEEE Transactions on Software Engineering*, 28(11), 2002, pp. 1023–1038.

[10] A. En-Nouaary and R. Dssouli, "A Guided Method for Testing Timed Automata", *In Proc. of the the 15th IFIP International Conference on Testing of Communicating Systems,* Nice, France, 2003.

[11] A. En-Nouaary, "A Scalable Method For Testing Real-Time Systems", *Software Quality Journal*, 16(1), 2008, pp. 3-22.

[12] A. En-Nouaary, "A Test Purpose Based Approach for Testing Timed Input Output Automata", *Journal of Software Testing, Verification, and Reliability*, 2008.

[13] B. Taylor, "Introducing Real-time Constraints into Requirements and High Level Design of Operating Systems", *In Proc. of the National Telecommunications Conference*, Houston, TX, 1980, pp. 18.5.1–18.5.5.

[14] B. Dasarathy, "Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them", *IEEE transactions on Software Engineering*, 11(1), 1985, pp. 80–86.

[15] T. S. Chow, "Testing Software Design Modeled by Finite State Machine", *IEEE Transactions Software Engineering*, 4(3), 1978, pp. 178–187.

[16] K. Čerāns, "Decidability of Bisimulation Equivalences for Parallel Timer Processes", *In Proc. of the 4th International Workshop on Computer Aided Verification*, Montreal, Canada, 1992, pp. 302–315.

[17] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test Selection Based on Finite-State Models", *IEEE Transactions Software Engineering*, 17(6), 1991, pp. 591–603.

[18] A. En-Nouaary, F. Khendek, and R. Dssouli, "Fault Coverage in Testing Real-Time Systems", *In Proc. of the International Conference on Real-Time Systems Computing Systems and Applications*, Hong Kong, 1999, pp. 13-15.

[19] A. Khoumsi, A. En-Nouaary, R. Dssouli, and M. Akalay, "A New Method for Testing Real-Time Systems", *In Proc. of the Seventh International Conference on Real-Time Systems and Applications*, Cheju Island, South Korea, 2000, pp. 441–451.