A Practical Method for the Reachability Analysis of Real-Time Systems Modelled as Timed Automata

Abdeslam En-Nouaary and Rachida Dssouli
ECE Department, Concordia University
Montreal, Canada
{*ennouaar,dssouli*}*@ece.concordia.ca*

***Abstract*—Real-time systems (RTSs) interact with their environment under time constraints. Such constraints are so critical because any deviation from the specified deadlines might have severe consequences on both the human lives and the environment. To develop reliable RTSs, formal methods should be used along the development life cycle. Verification is one of these formal methods, which aims at ensuring that the system is correct before its deployment. This paper presents a new verification method for the reachability analysis of real-time systems modelled as timed automata (TA) [1]. The paper basically addresses two main issues: are all the transitions of the system executable? Are all the locations reachable from the initial location of the system? In order to answer these questions, our method uses a metric that gives the minimum delay between any state and all the transitions leaving that state.**

***Keywords*-Real-Time systems, Formal Methods, Timed Automata, Verification, Reachability Analysis.**

I. INTRODUCTION

Over the past two decades, many researchers have been investigating the verification and validation of real-time systems with different backgrounds. As a result, several verification methods have been devised to make sure that the system functions properly before its deployment. These verification techniques attempt to check if the specification of the system satisfies some desirable functional and performance properties. All the verification and validation techniques rely on the use of formal models to describe the behaviour of the systems being investigated (see for instance [2], [3], [4], [5], [6], [7], [8], [9]). In the case of real-time systems, timed automata model [1] is intensively used by researchers to develop verification and testing techniques. Although, existing verification methods and tools (see for instance [2], [3], [4], [5]) provide successful results for RTSs, most of them suffer from the state explosion problem and are a bit complicated to use. This is mainly due to the fact that most of the proposed techniques are based on either the region graph [1] or the zone graph [10] as semantics for timed automata. So, the need for practical verification and validation methods still exists.

In this paper, we present a new method for the reachability analysis of RTSs modelled as timed automata. We are basically addressing two main issues: are all the transitions of the system executable? Are all the locations reachable from the initial location of the system? In order to answer these questions, our method uses a metric that gives the minimum delay between any state and all the transitions leaving that state. Our method presents two advantages. On the one hand, it automatically calculates on the fly the paths that ensure the reachability of the transitions and locations. On the other hand, it avoids the costly operation of constructing the region graph of the timed automata.

The remainder of this paper is organized as follows. Section 2 presents the timed automata model and its related concepts. Section 3 introduces our contributions. Section 4 concludes the paper and presents future work.

II. BACKGROUND

This section presents the definitions and concepts required for introducing our method. We basically present the timed automata model and the related theoretical results illustrated with simple examples.

Definition 1: Timed Automata (TA)
A TA A is a $5-$tuple (Σ, L, l_0, C, T), where :

- Σ is a finite set of inputs and output messages. In this paper, inputs begin with "?" while outputs start with "!".
- 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 the TA.
- $l_0 \in L$ is the initial location where the execution of the TA starts.
- 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 (re-)initialized to zero.
- $T \subseteq L \times \Sigma \times \Phi(C) \times \mathcal{P}(C) \times L$ is the set of transitions, where $\Phi(C)$ and $\mathcal{P}(C)$ denote the set of clock guards and the power set of C, respectively.

A transition in a TA, 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 ($R \subseteq C$) 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.

A sequence of consecutive transitions that starts at a location l and ends at a location l' is called a path from l to l'; we write $path(l, l') = t_1.t_2...t_n$, where $t_i \in T$ for $1 \leq i \leq n$, and $source(t_1) = l$, $destination(t_n) = l'$ and $source(t_i) = destination(t_{i-1})$ for $2 \leq i \leq n$. Since paths in TA are made of transitions with clock guards that could be conflicting (i.e., they cannot be satisfied by the same values of clocks) one can easily see that a path might not be executable. Hence, finding an executable path from one location to another requires, as explained later on in this paper, a systematic approach and a deeper investigation as to how messages and clock values should be chosen to fire the transitions of the system.

We assume that the transitions in a TA are instantaneous (i.e., they don't take time to execute). Also, the clock guards of the transitions are supposed to be conjunctions of atomic formulas of the form $(b_1\ op_1\ x\ op_2\ b_2)$, where $x \in C$, $(op_1, op_2) \in \{<, \leq, =\}$, and b_1 and b_2 are natural numbers. Multiple clocks are used in the TA to express time constraints between more than two transitions. Each clock, $x \in C$, in a TA takes real number values and has a bounded domain $[0, B_x] \cup \{\infty\}$, as stated by Springintveld et. al. [11], where B_x is the largest integer constant appearing in the time constraints over clock 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, we write : $\forall \epsilon > 0, B_x + \epsilon = \infty$ and $\infty + \epsilon = \infty$.

For a clock x and a clock guard G of a transition in a TA, we define the projection of G over x, written $Proj(G, x)$, by the condition $(b_1\ op_1\ x\ op_2\ b_2)$ in G, obtained by removing the conditions over all the clocks except x; if clock x is not involved in G then $Proj(G, x) = true$.

Example 1: Figure 1 shows a TA for a specification of a simple telephone system. The system waits for the user to hang up, get the dial tone and dial two digits $Digit_1$ and $Digit_2$; then the system issues the output $Connect$, to indicate that the connection has been established and the user can start talking. At the end, the user lifts the phone to allow the system to go back to its initial location. The behaviour of the system is subject to several time constraints. On the one hand, the user should type the first digit 1 to 3 time-

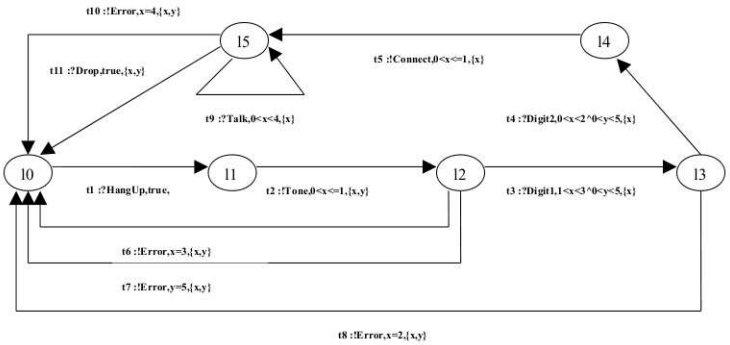

Figure 1. An Example of TA.

units after getting the tone and the second digit no more than 2 time-units after the first digit; the dialling operation (from getting the tone until dialing the last digit) should not exceed 5 time-units. On the other hand, the system must respond with the signal $Tone$ within 1 time-unit after the user hangs up, and with $Connect$ within 1 time-unit after the last digit has been typed. Whenever the time constraint of an input is not respected, the system times out, issues an error message and goes back to its initial location.

The TA model introduced thus far is an abstract model because it does not explain the execution of the system it describes. The executions, also called the operational semantics or the region graph of the TA, can be informally stated as follows. The TA 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 (re-)initialization. At any time, the TA can execute a transition $l \overset{m,G,R}{\longrightarrow} l'$ 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 TA changes its location to l'. To formalize the operational semantics of the TA, we need to define the concepts of clock valuations and states for the TA.

Definition 2: Clock valuations
Let $A = (\Sigma, L, l_0, C, T)$ be a $n-$clocks TA (i.e., an TA with n clocks), $\mathbf{R}^{\geq \mathbf{0}}$ be the set of non-negative real numbers.

- A clock valuation of A (or over C) is a function $v : C \rightarrow [\mathbf{R}^{\geq \mathbf{0}} \cup \{\infty\}]^n$, which assigns a positive value to each clock $x \in C$. A clock valuation is simply the binding of clocks to their actual values. In this paper, a clock valuation is represented by a vector $(v_{x_1}, v_{x_2}, .., v_{x_n})$, where $v(x_i) = v_{x_i}$ is the value of clock $x_i, 1 \leq i \leq n$. The set of all clock valuations of 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 (i.e., $y \in C$ and $y \notin R$. $[R := 0]v$ is the clock valuation obtained from v by resetting the clocks in R when a transition $l \stackrel{m,G,R}{\longrightarrow} l'$ is executed.
- A clock valuation $v \in V(C)$ satisfies the clock guard G of a transition $l \stackrel{m,G,R}{\longrightarrow} l'$, denoted by $v \models G$, if and only if G holds under v.

Informally speaking, a clock valuation is an interpretation of clocks, which allows us to know at any time the value of each clock used in the TA. In other words, a clock valuation can be used to determine how much time has elapsed since the execution of each transition that has last reinitialized a clock. The combination of a clock valuation and a location defines a state of the TA. The formal definition of such states follows.

Definition 3: States of the TA
Let $A = (\Sigma, L, l_0, C, T)$ be a TA.

- 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 $S(A)$.

Example 2: To illustrate the concepts of clock valuations and states, let us consider again the TA of Figure 1. The number of clocks in this TA is 2, namely clocks x and y. So, a clock valuation, here, consists of assigning a non-negative real number or ∞ to each of the clocks x and y. Examples of such clock valuations are $v_0 = (0, 0)$, $v_1 = (\frac{1}{4}, \frac{1}{4})$ and $v_2 = (\frac{1}{2}, \frac{3}{2})$. The set of the states of this TA is the set of all the pairs obtained by combining the locations and the clock valuations of the TA. Examples of such states are $s_0 = (l_0, v_0)$, $s_1 = (l_1, v_1)$ and $s_2 = (l_3, v_2)$, where l_0, l_1, and l_3 are the locations of the TA, and v_0, v_1, and v_2 are the clock valuations explained in this example.

Formally, the operational semantics of a TA is described by a state machine $M = (S, s_0, A, T)$, where S is the set of states of the TA, 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 the TA as well as the time delays (i.e., $A = \Sigma \cup [\mathbf{R}^{\geq \mathbf{0}} \cup \{\infty\}]^n$). Hence, there are two categories of transitions in M: The explicit transitions on input and output messages, and the implicit transitions on time delays. The explicit transitions are obtained from the transitions of the TA 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 the TA.
The operational semantics of timed automata helps us define the concepts of traces for real-time systems as follows. A trace for a real-time system is a sequence of input and output messages as well as time delays that starts at the initial state of the system and ends at one of its reachable state. It basically reflects an execution of the system on some input and output messages when the clock guards of the corresponding transitions are satisfied by the values of the clocks upon the occurrence of the messages. For instance, the trace $?m_1.\frac{1}{2}.?m_2.3.!m_3$ means that when the system starts its execution it immediately accepts the input message m_1, waits $\frac{1}{2}$ time-unit before accepting the input m_2, and then waits 3 time-units before responding with an output message m_3.

III. Our Method for the Reachability Analysis of Timed Automata

This section introduces our method for the reachability analysis of real-time systems modelled as timed automata. The two main issues dealt with in this paper are: the reachability of the locations from the initial location of the system and the executability of the transitions of the system. The objective of the former issue is to check if every location of the system is reachable from its initial location. However, the objective of the latter issue is to check if every transition of the system is executable. To address these issues, we propose a metric that determines the minimum delay between each state and each of its outgoing transitions. We also present an algorithm to implement the metric in order to decide the reachability issues automatically. The minimum delay between a state and a transition represents the minimum waiting time required at the state in order to execute the transition. It basically reflects the point of time right after the execution of the transition was impossible (i.e., as soon as the transition becomes executable). Formally, the minimum delay between a state $s = (l, v)$ and a transition $t = l \stackrel{m,G,R}{\longrightarrow} l'$, written $delay_{min}(s, t)$, is calculated as follows:

$delay_{min}(s, t) = Max_{x \in C}\{0, delay_{min}(v(x), \phi(x))\}$, where:

- $\phi(x) = Proj(G, x)$ is the projection of the transition's guard over the clock x, as explained in Section II,
- $v(x)$ is the value of the clock x at state s, and
- $delay_{min}(v(x), \phi(x))$ is the minimum waiting time at state s for clock x to satisfy its time constraint $\phi(x)$:

$$delay_{min}(v(x), \phi(x)) = \begin{cases} m_1 - v(x) + \epsilon \; if \; \phi(x) \; is \\ (m_1 < x \leq m_2) \\ m_1 - v(x) \; if \; \phi(x) \; is \\ (x = m_1) \; or \; (m_1 \leq x \leq m_2) \\ 0 \; if \; \phi(x) \; is \; true \end{cases}$$

ϵ is a small positive real value chosen by the designer. It is a parameter, which helps him/her specify how far from the lower bound of the open clock guard the transition should be executed.

Example 3: Let us consider again the telephone system of Figure 1 and suppose that the designer chooses $epsilon = \frac{1}{4}$. For the states $s_0 = (l_0, (0, 0))$ and $s_1 = (l_2, (\frac{1}{4}, \frac{1}{4}))$, and the transitions t_1 and t_3, we have:

- $delay_{min}(s_0, t_1) = Max\{0, delay_{min}(0, true), delay_{min}(0, true)\} = Max\{0, 0, 0\} = 0$.
- $delay_{min}(s_1, t_3) = Max\{0, delay_{min}(\frac{1}{4}, 1 < x < 3), delay_{min}(\frac{1}{4}, 0 < y < 5)\} = Max\{0, \frac{1}{2}, 0\} = \frac{1}{2}$.

Now, we explain how we address the reachability issues aforementioned. To deal with the reachability of the locations of the system, we proceed as follows. Let $A = (\Sigma, L, l_0, C, T)$ be a TA and l be a location of A (i.e., $l \in L$). l is said to be reachable from the initial location of the TA if and only if there exists an executable path, $path(l_0, l)$, from l_0 to l in A. There are at least three different ways to find an executable path from the initial location l_0 to another location l in the TA:

- The first method consists of first extracting all the paths from l_0 to l and then choosing the shortest one (i.e., the path with the least number of transitions).
- The second method consists of, as the first method, extracting all the paths from l_0 to l and then choosing randomly one of them.
- The third method consists of extracting on the fly only one path from l_0 to l according to some metrics that minimize either the number of the transitions in the path or the time it takes to execute the path.

The first two methods have the disadvantage of being costly because they have to extract all the paths from l_0 to l. Moreover, the chosen path (either the shortest one or the randomly selected one) might not be executable because of the conflicting clock guards of its transitions and hence the resulting path could be useless. The third method is less costly than the two others because it does not rely on the extraction of all the paths from l_0 to l. However, if the minimization adopted is with regard to the number of the transitions in the selected path then we will have no guarantee about the executability of the path, all as for the first and the second methods. Hence, the best way to choose and ensure an executable path from l_0 to l is to extract it on the fly by minimizing the time it takes to execute the path. This can be done by using the metric $delay_{min}()$ introduced so far. More precisely, to get an executable $path(l_0, l)$, we have to start at the initial state (l_0, v_0) and calculate the minimum time delay to execute each transition leaving l_0 and decide which one should be added to the path. Then, we compute the resulting states and repeat the process on the new states until we reach l.

Similarly, to address the executability of a transition we have to find an executable path from the initial location to the source location of the transition, and use the minimum time delay in order to calculate at least one time point that makes the transition executable from the last reached state from the path. In order to ensure an executable path from l_0 to the source location of the transition, we follow the same process described previously when dealing with the reachability of the locations of the TA. Regarding the time point that makes the transition executable, we calculate it using the minimum time delay between the last reached state in the path for the transition, and the transition being checked. Formally speaking, let $A = (\Sigma, L, l_0, C, T)$ be a TA and $t = l \overset{m,G,R}{\longrightarrow} l'$ be a transition of A. t is said to be executable if and only if:

- $source(t)$ is reachable from the initial location l_0 and the resulting state is $s = (source(t), v)$, and
- $(v + delay_{min}(s, t))$ satisfies the clock guard of t (i.e., $(v + delay_{min}(s, t)) \models G$).

It should also be noted that a location l is reachable if and only if there exists $t \in T$ such that $destination(t) = l$ and t is executable. Likewise, if a location l is not reachable then all the transitions leaving l are non-executable (i.e., for all $t \in T$ such that $source(t) = l$ the transition t is non-executable).

The algorithm used to check the reachability of the locations and transitions of the TA is shown below. The algorithm takes as input the TA and returns a Boolean value for each location and each transition that says whether or not the location (respectively the transition) is reachable (respectively executable). The algorithm starts by calculating the initial state of the TA and initializing all the variables to be used, namely RS (the set of reachable states) and HS (the set of the handled states among RS). Then, it goes through all the states in RS and handles all the outgoing transitions from each of these states. Indeed, for each reachable state, the algorithm checks all of the outgoing transitions from the location of the state and verifies if they are executable by calculating the minimum delay between the state and each of the transitions. If the clock guard of a transition is satisfied by the clock valuation of the current state plus the minimum delay calculated previously then the transition (respectively its destination

location) is marked as being executable (respectively reachable), and the resulting state is calculated and added to RS if it is not already there. When the algorithm terminates the handling of all the reachable states (i.e., all the states in RS), it goes through all the locations and transitions to check if they have been marked so far. If a location has not been marked then the location is declared unreachable. Likewise, if a transition has not been marked then the transition is declared non-executable. It should be noted that the algorithm does not construct the region graph [1] of the TA but calculates on the fly only one state for each transition in the TA. That state is obtained using the metric $delay_{min}(s,t)$ introduced in the beginning of this section. Hence, the proposed approach has the advantage of being scalable and rapid compared to existing methods that are based on the construction of either the region graph [1] or the zone graph [10] of the TA. We implemented the algorithm in Java and promising results are obtained for specifications with different sizes. The presentation of the tool and the analysis of the experimentation results are left for a future publication.

Algorithm 1: Our Algorithm for the Reachability Analysis of a TA.

ReachabilityAnalysis(INPUT: TA)
$s_0 \leftarrow (l_A^0, v_0)$ // ($v_0(x) = 0$ for every clock x in the TA).
$RS \leftarrow s_0$. // RS is the set of reachable states of the TA.
$HS \leftarrow \emptyset$. // HS is the set of handled states of the TA.
while *($RS \neq HS$)* **do**
  Pick one state $(s : (l, v) \in RS)$ not yet processed.
  Add s to HS.
  foreach *(transition $t : l \xrightarrow{m,G,R} l'$ in the TA)* **do**
    Calculate $\delta = delay_{min}(s,t)$.
    if *($(v+\delta) \models G$)* **then**
      Add the state $(l', [R := 0](v+\delta))$ to RS if not yet there.
      Mark the location l' and the transition t in the TA as they are reached.
foreach *(location $l \in L$)* **do**
  if *(l is not marked)* **then**
    l is not reachable
foreach *(transition $t \in T$)* **do**
  if *(t is not marked)* **then**
    t is not executable

The complexity of the algorithm is $\Theta(|L| \times |T|)$, where $|L|$ is the number of locations and $|T|$ is the number of transitions. Indeed, the algorithm goes through all the reachable states whose number is at most equal to $|L| \times |T|$. Each reachable state is handled only once and requires the processing of only the transitions leaving the location of the state. By adding up the number of these iterations we get an order of $\Theta(|L| \times |T|)$.

Example 4: Let us consider again the telephone system of Figure 1. By applying our algorithm, with $\epsilon = \frac{1}{4}$, we get the results shown in Figure 2. The first table gives for each location if it is reachable or not while the second table determines for each transition if it is executable or not. When a location (respectively, a transition) is reachable (respectively executable) the tables show one of the traces that make it possible. By examining the results in Figure 2, one can easily see that all the locations (respectively all the transitions) of the system modelled in Figure 1 are reachable (respectively executable). For each reachable location, the corresponding trace is obtained by extracting an executable path from the initial location to the location. Similarly, for each executable transition the corresponding trace is obtained by extracting an executable path from the initial location to the source location of the transition plus the time delay and the message to execute the transition. The executability of any path is ensured based on the metric $delay_{min}()$, introduced so far. For instance, the location l_4 is reachable and the transition $t_4 : l_3 \xrightarrow{?Digit_3, 0<x<2 \wedge 0<y<5, \{x\}} l_4$ is executable because the trace $?HangUp.1.!Tone.\frac{5}{4}.?Digit_1.\frac{1}{4}.?Digit_2$ makes it possible for the system to move from its initial state $(l_0, (0,0))$ to $(l_4, (0, \frac{3}{2}))$; the corresponding executable path then is $t_1.t_2.t_3.t_4$.

Example 5: Let us now change the specification of Figure 1 by changing the clock guards of the transitions t_3 t_4 and t_8 to $((2 < x < 3) \wedge (0 < y < 3))$, $((2 < x < 3) \wedge (0 < y < 3))$ and $(x = 3)$, respectively. Is it really easy to guess the reachability analysis of the new system? It is not that simple! By applying our algorithm, with $\epsilon = \frac{1}{4}$, we can see that the transitions t_4, t_5, t_9, t_{10} and t_{11} are non-executable, and the locations l_4 and l_5 are non-reachable. Let us say why. The minimum executable path to reach the location l_3 (the source location of the transition t_4) is $t_1.t_2.t_3$ and the corresponding trace is $?HangUp.1.!Tone.\frac{9}{4}.?Digit_1$. Hence, the state that should be considered to execute the transition t_4 is $s_4 = (l_3, (0, \frac{9}{4}))$, which gives $delay_{min}(s_4, t_4) = \frac{9}{4}$. But, by adding $\frac{9}{4}$ to $(0, \frac{9}{4})$ (i.e., the clock valuation of s_4) we obtain the clock valuation $(\frac{9}{4}, \frac{18}{4})$ that does not satisfy the clock guard $((2 < x < 3) \wedge (0 < y < 3))$ (i.e., the clock guard of t_4). Hence, the transition t_4 is non-executable and since t_4 is the unique transition between l_3 and l_4 then the location l_4 is not reachable. Consequently, all the transitions leaving l_4 are non-executable, namely the transition t_5. Since t_5 is the unique transition between l_4 and l_5 then the location l_5 is not reachable and all the transitions leaving l_5 become non-executable, namely t_9, t_{10} and t_{11}.

 ISBN: 978-1-61208-165-6

Loc.	Reach. (Y/N)	Corresponding Trace
l_0	Y	ϵ (the empty sequence)
l_1	Y	$?HangUp$
l_2	Y	$?HangUp.1.!Tone$
l_3	Y	$?HangUp.1.!Tone.\frac{5}{4}.?Digit_1$
l_4	Y	$?HangUp.1.!Tone.\frac{5}{4}.?Digit_1.\frac{1}{4}.?Digit_2$
l_5	Y	$?HangUp.1.!Tone.\frac{5}{4}.?Digit_1.\frac{1}{4}.?Digit_2.1$ $.!Connect$

Trans.	Exec. (Y/N)	Corresponding Trace
t_1	Y	$?HangUp$
t_2	Y	$?HangUp.1.!Tone$
t_3	Y	$?HangUp.1.!Tone.\frac{5}{4}.?Digit_1$
t_4	Y	$?HangUp.1.!Tone.\frac{5}{4}.?Digit_1.\frac{1}{4}.?Digit_2$
t_5	Y	$?HangUp.1.!Tone.\frac{5}{4}.?Digit_1.\frac{1}{4}.?Digit_2.1$ $.!Connect$
t_6	Y	$?HangUp.1.!Tone.3.!Error$
t_7	Y	$?HangUp.1.!Tone.5.!Error$
t_8	Y	$?HangUp.1.!Tone.\frac{5}{4}.?Digit_1.2.!Error$
t_9	Y	$?HangUp.1.!Tone.\frac{5}{4}.?Digit_1.\frac{1}{4}.?Digit_2.1$ $.!Connect.\frac{1}{4}.?Talk$
t_{10}	Y	$?HangUp.1.!Tone.\frac{5}{4}.?Digit_1.\frac{1}{4}.?Digit_2.1$ $.!Connect.4.!Error$
t_{11}	Y	$?HangUp.1.!Tone.\frac{5}{4}.?Digit_1.\frac{1}{4}.?Digit_2.1$ $.!Connect.?Drop$

Figure 2. The Reachability Results for the TA in Figure 1.

IV. CONCLUSION

We presented in this paper a new verification method for the reachability analysis of real-time systems modelled as timed automata. Our method addresses the reachability of the locations and transitions of the system by calculating a trace that allows the system to go from its initial state to the location or transition being investigated. To this end, the method uses a metric that gives the minimum delay between any state and all the transitions leaving that state. Our method has at least two advantages. On the one hand, it automatically calculates on the fly the paths that ensure the reachability of the transitions and locations. On the other hand, it avoids the costly operation of constructing the region graph of the TA, which makes the method more scalable than the others. To help us quantify precisely the gain of the method with respect to existing methods, we implemented the method to conduct more experimentation on TA specifications with different sizes. The tool and the analysis of the experimentation results will be discussed in a future paper.

We are currently working on two extensions of the proposed method. On the one hand, we would like to make it incremental to adjust to successive evolutions of the specification either when designing the system the first time or later when maintaining the system. On the other hand, we are investigating the possibility of adopting the incremental method to the area of testing real-time systems modelled as TA.

REFERENCES

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

[2] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages –. Springer-Verlag, 1995.

[3] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In *4th. DIMACS Workshop on Verification and Control of Hybrid Systems*, New Brunswick, New Jersey, October 1995.

[4] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.

[5] K.G. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *Proceedings of Fundamentals of Computation Theory*, pages –, Dresden, Germany, August 1995.

[6] Osmane Koné, Patrice Laurencot, and Richard Castanet. On the Fly Test Generation for Real-Time Protocols. In *International Conference on Computer Communications and Networks, Lafayette, Louisiana, USA*, pages 378–387, 1998.

[7] Brian Nielsen and Arne Skou. Automated Test Generation from Timed Automata. In *5th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems FTRTFT'98, Lyngby, Denmark*, pages 59–77, September 1998.

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

[9] A. En-Nouaary. A Scalable Method for Testing Real-Time Systems. *Software Quality Journal, Springer*, 16(1):3–22, March 2008.

[10] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of Timed Transition Systems. pages 340–354, 1992.

[11] J. Springintveld and F. Vaandrager. Minimizable Timed Automata. In B. Jonsson and J. Parrow, editors, *Proceedings of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems*, Uppsala, Sweden, volume 1135 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

 ISBN: 978-1-61208-165-6