We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



6

Construction of Real-Time Oracle Using Timed Automata

Seyed Morteza Babamir[1] and Mehdi Borhani Dehkordi[2]
[1]*University of Kashan, Kashan,*
[2]*University of Science and Applied Shar-e-Kord, Shahr-e-Kord, Iran*

1. Introduction

Verification of real time software is facing two problems: (1) how we should manage to produce verification rules and (2) how we should apply the rules to specify the problem. In this chapter we provide a method by which we get down to these two problems. In the first step, we specify real time software using *Timed Automata* and then we state it in RTL, real time logic, propositions. Timed Automata address modeling systems in time (Alure & Dill, 1996). In the second step, we obtain the safety constraints from *reachability graph* of Timed Automata of the problem specification and after that we state it in real time logic propositions. These propositions showing safety constraints are used for verification of the propositions, i.e. the results of the specification obtained in the first step. To show the effectiveness of our method, we set forth it for the RCC (Rail Road Crossing Control) real time system.

Software verification is an important process in constructing software and a main factor for obtaining safety from the quality of software. For verification of software we should deal with its verification against the expected behavior. The expected behavior is safe characteristics or prepositions that the software must always agree with. For this purpose, there are three main approaches: (1) static verification, (2) software testing and (3) run-time verification. Meanwhile one of the important and key approaches, essentially used for mission and safety critical systems, is the run-time verification; the existence of the problems in the first and second approaches is the reason of using run-time verification. For example, we can indicate the disability of the first method in proving the complicated and large specifications. Another instance is rapidly increasing the number of states (called state explosion) in the *model checking* method, which is a static verification one.

Because Timed Automata are methods based on time and event, they are suitable for describing the behavior of real time systems and because Timed Automata are visual methods, their understanding is easy. But they have limitations for specification of some statuses and also they cannot specify some conditions very well. Therefore, we need especial methods for analysis and verification of Timed Automata behavior that increase safety in these systems and decrease the amount of the faults.

There are different ways to verification of Timed Automata behavior. One of these ways is using *reachability graph*; however an especial method should be chosen to cover the

weakness of this method. We can use methods that are based on logic for reasoning and timing the Timed Automata. In this chapter, we suggest a framework for setting and executing Timed Automata using *Real Time Logic* (RTL) providing a reasoning framework based on *First Order Logic* (FOL). RTL reflects a different method for timed systems (Paneka et al, 2006).

In this chapter, we show that Timed Automata per se is not able to show constraints of the systems, but if it is used with formal textual language such as RTL, expressing the constraints of the system would by more.

This chapter includes four sections: (1) in the second section, we express a brief explanation about Timed Automata, (2) in the third one we address an explanation about Real Time Logic and (3) in fourth section we express our approach to simulate Timed Automata by means of Real Time logic. In this section, we propose a case study and specify it using Timed Automata and Real Time logic. Then, we discuss the system constraints to supervise unsafe states.

2. Timed automata

Timed Automata (Alur & Dill, 1994) was proposed by Alur and improved by Dill. Timed Automata are finite machines equipped with sorts of clocks. Clocks: (1) are real functions with continuous time that record the times between the events separately and (2) are increased equally. Timed Automata are introduced as a formal specification for modeling the behavior of real time systems. Timed Automata are: (1) general methods for exhibition of timed transition state diagrams that use a number of time variables having real amounts, (2) finite clocked automata to specify the timed systems and (3) suitable for verification of distributed systems, optimization, verification of multi-tasking programs, network analysis, planning and scheduling (Paneka et al, 1998).

In Timed Automat, a safe path from the first state to a final state is a set of states in which actions are performed and timed requirements are satisfied. If such path is found, it is a solution for the problem. Timed Automata are stated in tuple M=($\sum$, S, S_0, X, E) in which $\sum$ is a finite set of actions, S is a finite set of states, S_0 is a finite set of initial states ($S_0 \subset S$), X is a finite set of clocks and E is a set of transitions. Every transition consists of <L, a, g, λ, L'>. A transition from present state L to the next state L' is made when the action a is performed and clock g(x) having true amount is passed. Notation λ is a subset of X during which the transition will be reset. Relation 1 states that clock values or difference of two clock values are real numbers.

$$g::=x\leq c \mid c\leq x \mid x-y\leq c \mid x<c \mid c<x \mid x-y<c \mid g\wedge g \mid \text{true that } x,y \in X,\ c\in R \qquad (1)$$

2.1 Networks of timed automata

In this section, we define networks of Timed Automata, consisting of several Timed Automata running in parallel and communicating with each other.

Definition. A timed automaton (TA, for short) is a six-element tuple, $\vartheta = (A, L, l^0, E, X, I)$

Where

- A is a finite set of actions, where $A \cap IR_{0+} = \varphi$,
- L is a finite set of locations,
- $l^0 \in L$ is an initial location,
- X is a finite set of clocks,
- $E \subseteq L * A * C_X^\theta * 2^X * L$ is a transition relation,
- $I : L \to C_X^\theta$ is a (location) invariant.

Each element *e* of *E* is denoted by $l \xrightarrow{a,cc,X} l'$ represents a transition from the location l to the location l', executes the action a, with the set $X \subseteq \chi$ of the clocks to be reset, and with the clock constraint cc defining an enabling condition. The function I assigns each location $l \in L$ a clock constraint defining the conditions under which ϑ can stay in l.

If the enabling conditions and the values of the location invariant are in the set C_X only, then the automaton is called *diagonal-free*. Given the transition $e : l \xrightarrow{a,cc,X} l'$, we write source(s), action(s), target(s), guard(s) and reset(s) for l, l', a, cc and X, respectively. The clocks in Timed Automata allow expressing the time properties. An enabling condition constrains the execution of a transition. An invariant condition permits an automaton to stay at the location l as long as the clock constraint I(l) is satisfied.

Real-time systems are usually represented by networks (sets) of Timed Automata. A typical example widely considered in the literature, is modeling an automated railroad crossing (known as the Train–Gate–Controller).

A set of timed automata can be composed into a global (product) timed automaton as follows: the transitions of the timed automata that do not correspond to a shared action are interleaved whereas the transitions labeled with a shared action are synchronized. There are many different definitions for a parallel composition. One definition is determining the multi-way synchronization, i.e., each component that contains a communication transition (labeled with a shared action) has to perform this action (Penczek & Polrola, 2006).

Definition. Let $\lambda = \{i_1, ..., i_{n\lambda}\}$ be a finite ordered set of indices, and $\zeta = \{\vartheta_i \mid i \in \lambda\}$ where $\vartheta_i = (A_i, L_i, l_i^0, E_i, X_i, I_i)$, is a set (network) of Timed Automata indexed with λ. The automata in ζ are called components. Let $A(a) = \{i \in \lambda \mid a \in A_i\}$ be a set of the indices of the components containing the action $a \in \bigcup_{i \in \lambda} A_i$. A composition (product) of the Timed Automata $\vartheta_{i_1} || ... || \vartheta_{i_{n\lambda}}$ is a timed automaton specified as: $\vartheta = (A, L, l^0, E, X, I)$, where

- $A = \bigcup_{i \in \lambda} A_i$,
- $L = \prod_{i \in \lambda} L_i$,
- $l^0 = (l_{i_1}^0, ..., l_{i_n}^0)$,
- $X = \bigcup_{i \in \lambda} X_i$,
- $I((l_{i_1}, ..., l_{i_{n\lambda}})) = \wedge_{i \in \lambda} I_i(l_i)$,

And the *transition relation* is given as Relation 2.

$$\begin{aligned} &(l_{i_1}, ..., l_{i_{n\lambda}}), a, \wedge_{i \in A(a)} cc_i, \bigcup_{i \in A(a)} X_i, (l'_{i_1}, ..., l'_{i_{n\lambda}}) \in E \Leftrightarrow \\ &(\forall i \in A(a))(l_i, a, cc_i, X_i, l'_i) \in E_i \text{and} (\forall i \in \lambda \setminus A(a)) l'_i = l_i. \end{aligned} \tag{2}$$

2.2 Semantics of timed automata

Let $\vartheta = (A, L, l^0, E, X, I)$ be a timed automaton. A concrete state of A is defined as ordered pair (l, v), where $l \in L$ and $v \in IR_{0+}^{n_x}$ is a valuation. The *concrete (dense) state space* ϑ is a *transition system*, $C_c(\vartheta) = (Q, q^0, \xrightarrow[c]{})$ where

- $Q = L * IR_{0+}^{n_x}$ is the set of all the concrete states,
- $q^0 = (l^0, v^0)$ with $v^0(x) = 0$ for all $x \in X$ is the initial state, and
- $\xrightarrow[c]{} \subseteq Q * (E \cup IR_{0+}) * Q$ is the transition relation, defined by an action

A time successor is defined as Relation 3.

$$\text{for } \delta \in IR_{0+}, (l, v) \xrightarrow{\delta}_c (l, v + \delta) \text{iff } v, v + \delta \in \|I(l)\| \text{ (Time successor)}, \tag{3}$$

- For $a \in A, (l, v) \xrightarrow{a}_c (l', v') \text{iff} (\exists cc \in C_x)(\exists X \subseteq x)$ such that
- $l \xrightarrow{a, cc, X} l' \in E$, $v \in [cc]$, $v' = v[X := 0]$, and $v' \in \|I(l')\|$ (action successor).

Intuitively, a time successor does not change the location l of a concrete state, but it increases the clocks, provided that their values still satisfy the invariant l. Since the invariants are zones, if v and v' satisfy $I(l)$, then all the clocks between v and v' satisfy $I(l)$. An action successor corresponding to the action a is executed when the guard cc holds for v and v' obtained after resetting the clocks in X, satisfies the invariant l'.

For $(l, v) \in Q$ and $\delta \in IR_{0+}$, let $(l, v) + \delta$ denotes $(l, v + \delta)$. Concatenation of two time steps $q \xrightarrow{\delta}_c q + \delta$ and $q + \delta \xrightarrow{\delta'}_c q + \delta + \delta'$ is the time step $q \xrightarrow{\delta + \delta'}_c q + \delta + \delta'$. Similarly, if $q \xrightarrow{\delta}_c q + \delta$, then for any δ there exist:

$$\delta_1, ..., \delta_k \in IR_{0+} \text{ where, } \delta_1 + ... + \delta_k = \delta \ \& \ q \xrightarrow{\delta_1}_c q + \delta_1 \xrightarrow{\delta_2}_c ... \xrightarrow{\delta_k}_c q + \delta .$$

The second relation denotes that each time step can be split into an arbitrary number of consecutive times. A (dense) q_0-run ρ of ϑ is a maximal (i.e., non-extendable) sequence,

$$\rho = q_0 \xrightarrow{\delta_0}_c q_0 + \delta_0 \xrightarrow{a_0}_c q_1 \xrightarrow{\delta_1}_c q_1 + \delta_1 \xrightarrow{a1}_c q_2 \xrightarrow{\delta_2}_c ...$$

where $a_i \in A$ and $\delta_i \in IR_{0+}$, for each $i \geq N$ (notice that due to the fact that δ can be equal to zero, two consecutive transitions can be executed without any time passing in between, and that consecutive time passages are concatenated). Such runs are called weakly monotonic (Penczek & Polrola, 2006).

3. Real Time Logic

Real Time Logic (RTL) introduced by Jahanian and Mok in 1986, is an FOL having a set of elements for specification of requirements of real time systems (Bellini et al, 2000). The logic is a formal language used for reasoning about time characteristics of the real time systems. RTL deals with ordering of the events (Jahanian-a, 1998). There is a significant difference between event and action in RTL (Jahanian-b et al, 1998).

There are four types of events:

1. External event like pushing a button by the user,
2. The starting event denoting the beginning of an action.
3. The ending event denoting the completion of an action,

Performing an action is shown by starting and ending events [4]. Events have unique name and the capital letters are used for showing them. To show the external events, we use notation Ω before the name of an event. Also notation ↑ is used before the name of an event to show a starting event (denoted by ↑A) and notation ↓ to show an ending event (denoted by ↓A). Propositions of RTL are the relations between occurrences of the events using orderings < , = , >, ≥ ,≤ (Jahanian-b et al, 1998). The RTL formulas are made of equal/unequal, existential forms, quantifiers ∀ and ∃ and logical relations, $\wedge,\vee,\rightarrow,\neg$ (Jahanian, 1994). A function is used to show the time of the occurrence of an event and allocation of an amount of time to the occurrence of the event. It is shown by notation "@". Relation "@(e,i)" denotes the i^{th} occurrence of event *e* where *e* can be a starting or external event (Jahanian-a et al, 1998).

Example:

Consider a system specified in a natural language; the system samples and displays data on demand by external stimuli. Upon pressing button #1, action SAMPLE is executed within 30 time units. During each execution of this action, data are sampled and subsequently appeared in the display panel. The computation time of action SAMPLE is 20 time units. The following set of formulas is a partial description of the system in RTL:

$$\forall i \forall t R(\Omega \text{BUTTON1}, i, t) \rightarrow$$
$$[\exists x, y R(\uparrow \text{SAMPLE}, i, x) \wedge R(\downarrow \text{SAMPLE}, i, y) \wedge t \le x \wedge y \le t + 30]$$
$$\forall i \forall x, y[R(\uparrow \text{SAMPLE}, i, x) \wedge R(\downarrow \text{SAMPLE}, i, y)] \rightarrow x + 20 \le y$$

3.1 The RTL Language

In this section, we introduce the RTL language; the following notations are used in RTL formulae:

- true and false
- *A*, set of time variables
- *B*, set of occurrence variables
- C, constants including the natural numerals
- D, set of events
- Function +
- Predicates $\le,\ge,<,>,=$
- Occurrence relation *R*
- Logical connections: $\wedge$, $\vee$, $\neg$ and $\rightarrow$
- Existential and universal quantifiers: $\forall$, $\exists$.

The *time terms* in RTL consist of constants, variables and if t_1 and t_2 are time terms, then $t_1 + t_2$ is a time term. The *occurrence terms* in RTL are expressions consisting of constants, variables and if *i* and *j* are occurrence terms, then i + j is an occurrence term.

The propositions in RTL consist of truth symbols, *true* and *false* and:

- if t_1 and t_2 are time terms and ρ is an inequality/equality predicate, then t_1 ρ t_2 is a proposition
- if *i* and *j* are occurrence terms and ρ is an inequality/equality predicate symbol, then *i* ρ *j* is a proposition
- if *i* is an occurrence term, t is a time term and e is an event constant, then R (e, i, t) is a proposition.

The formulas of RTL are constructed from the propositions, logical connections and quantifiers.

Definition: An occurrence relation is a relation on the set E × Z+ × N where E is a set of events, **Z+** is the set of positive integers, and **N** is the set of natural numbers, such that the following axioms hold:

Monotonicity Axioms: For each event *e* in the set D,

$$\forall i \forall t \forall t' [R(e,i,t) \wedge R(e,i,t') \rightarrow t = t']$$
$$\forall i \forall t [R(e,i,t) \wedge i > 1] \rightarrow [\exists t' R(e,i-1,t') \wedge t' < t]$$

The first axiom states that at most onetime value can be associated with each occurrence *i* of an event ***e***, i.e., two same occurrences of an event cannot happen at two distinct times. The second axiom expresses that if the i^{th} occurrence of event *e* happens, then the previous occurrences of *e* would happened before. This axiom also states that two distinct occurrences of the same event must happen at different times.

Start/Stop Event Axioms: For each pair of start/stop events in the set **D**, we have:

$$\forall i \forall t R(\downarrow A, i, t) \rightarrow [\exists t' R(\uparrow A, i, t') \wedge t' < t]$$

where ↑A and ↓A denote the events: start and stop of action A, respectively. The axiom states that every occurrence of a stop event would be preceded by a corresponding start event.

Transition Event Axioms: For the transition events in the set *D* corresponding to a state variable *S*, we have the following relations. Proposition 4(a) states that the first occurrence of event (S:=T) is at time zero. This means that initially the system is in state *S*. Proposition 4(b) states that i^{th} occurrence of event (S:=F) is at time *t*. This means that i^{th} time that the system is not in state *S* is at time *t*. The transition from 4(a) to 4(b) shows a *transition event* where the system is initially in state *S* and at future time the system is not in state *S*. The transition from 4(b) to 4(c) shows the entrance of the system to state *S* and the transition from 4(c) to 4(d) shows the exit of the system from state *S*. Transitions in 5(a) to 5(d) is similar to transitions in 4(a) to 4(d) but the system is not initially in state *S*. In fact, Propositions 4(a) to 4(d) and Propositions 5(a) to 5(d) define the order in which two complementary transition events can occur depending on whether S is initially true or false (Jahanian-b et al, 1998).

$$R((S := T), 1, 0) \rightarrow \quad (4a)$$

$$(\forall i > 1, \forall t > 0, R((S := F), i, t) \rightarrow \quad (4b)$$

$$[\exists t'R((S := T), i, t') \wedge t' < t] \wedge \forall i \forall t R((S := T), i+1, t) \rightarrow \quad (4c)$$

$$[\exists t'R((S := F), i, t') \wedge t' < t] \quad (4d)$$

$$R((S := F), 1, 0) \rightarrow \quad (5a)$$

$$(\forall i \forall t R((S := T), i, t) \rightarrow \quad (5b)$$

$$[\exists t'R((S := F), i, t') \wedge t' < t] \wedge \forall i \forall t R((S := F), i+1, t) \rightarrow \quad (5c)$$

$$[\exists t'R((S := T), i, t') \wedge t' < t] \quad (5d)$$

3.2 State predicates

RTL provides nine different *state predicates* to evaluate the value of S over an interval as follows:

S[x,y], S(x,y), S<x,y>, S[x,y), S[x,y>, S(x,y], S(x,y>, S<x,y], S<x,y)

Each state predicate qualifies the timing of two events, one denotes the transition event that changes the value of the state variable to true and the other denotes the transition event that changes the value of S to false. The arguments, x and y, in the state predicates are used in conjunction with the symbols "[", "]", "(", ")", "<" and ">" to denote an interval over which the state variable remains true (Jahanian, 1994). Suppose E_t and E_f denote the transition events making S true and false, respectively. Informally:

- "[x " denotes that E_t occurs at time x,
- "(x " denotes that E_t occurs before or at time x,
- "<x" denotes that E_t occurs before time x,
- " y]" denotes that E_f occurs at time y,
- " y)" denotes that E_f does not occur before time y,
- " y>" denotes that E_f does not occur before or at time y.

Definition: If state variable S is initially true, i.e., $R((S := T), 1, 0)$, then

$$S[x,y] \equiv \exists i, R((S := T), i, x) \wedge R((S := F), i, y)$$
$$S[x,y) \equiv \exists i, R((S := T), i, x) \wedge [\forall t R((S := F), i, t) \rightarrow y \leq t]$$
$$S[x,y> \equiv \exists i, R((S := T), i, x) \wedge [\forall t R((S := F), i, t) \rightarrow y < t]$$

The other intervals are defined similarly. If state variable S is initially false, i.e., $R((S := F), 1, 0)$, then $S[x,y] \equiv \exists i R((S := T), i, x) \wedge R((S := F), i+1, y)$. The other intervals are defined similarly (Jahanian-b , 1998).

4. The proposed approach

In this section, we propose a method to produce verification rules from Timed Automata. These rules are used for verification of the software behavior that is obtained from the problem specification. Our method has five steps (Fig.1). In the first step, we specify the

system behavior in the Timed Automata and then verify the different states of the system. In the second step, we explain specification of the system in RTL. In the third step, based on the model of the Timed Automata, we work out the reachability graph of the system, and show the behavior of the system in safety and risky states. In the fourth step, we use reachability graph obtained in the third step to extract constraints and express them in RTL by which we verify entering the system to unsafe states.

After specifying the system constraints in RTL and specification of the system behavior by the model in the second step, we verify the problem specification (in fact, we show a visual model of the system in Timed Automata and its behavior in RTL. By the rules extracted from the system behavior, we address verification of the system in entering to unsafe states

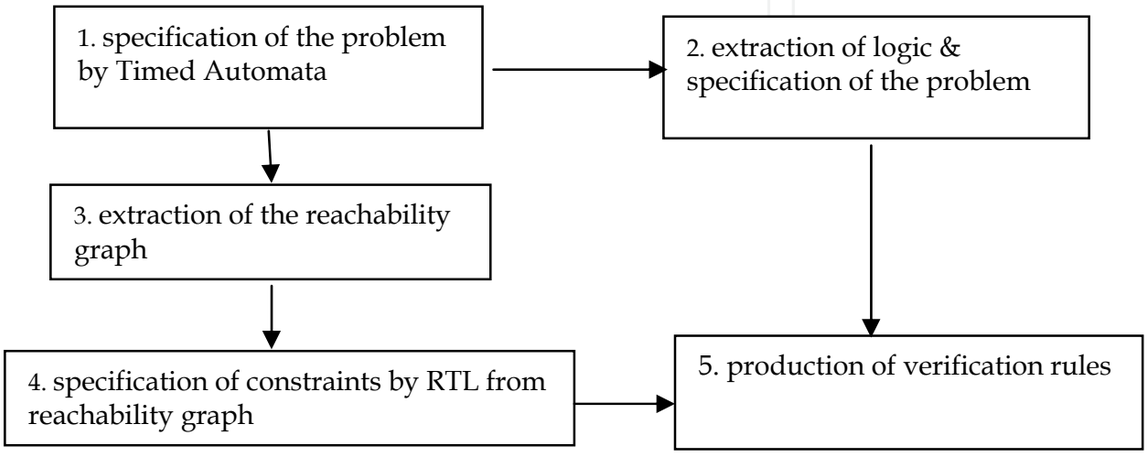


Fig. 1. The proposed approach

4.1 Specifying the problem in timed automata

In this section, we show states of a real time system in the visual and logic forms.

Structure 1: Change of a state due to occurrence of an event. The change of a state in real time systems might be taken place due to occurrence of an event.

Visual definition. When the system is in state *p* and a guard event takes place, the system makes a transition from state *p* to state *q* (Fig. 2).

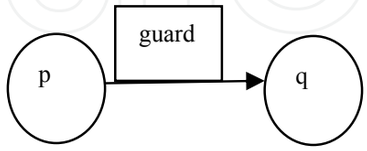


Fig. 2. Change of a state by occurrence of a guard event

Logical definition of structure 1. When the system is in state *p* at time point *t*, then by i^{th} occurrence of guard event at time *t*, the system goes to state *q* (Relation 6). Considering state predicates in Section 3.2, we define a time point as p(t,t).

$$\forall i\ \forall t[p(t,t)\ \wedge @(GUARD,i)=t\ ] \rightarrow \exists t',d[q(t',t') \wedge t' \geq t+d \wedge d\geq 0] \quad (6)$$

Structure 2. Change of a state by occurrence of an event and the action execution. After occurring the event and the action execution, the state of the system might change.

Visual definition. When the system (Timed Automata) is in state p and a guard event occurs, the action is accomplished and the system holds in state q (Fig. 3).

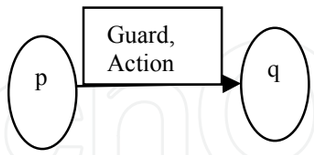


Fig. 3. Change of a state by occurrence of an event and execution of an action

Logical definition of structure 2. If the system is in the state *p* at time *t* and a guard event takes place, we have an occurrence of the starting event of the action denoted by ↑ACTION. This causes the start of the action and ending event of action denoted by ↓ACTION. The action is accomplished during the time ε>0 and the state changes (Relation 7).

$$\forall i \ \forall t[p(t,t) \wedge @(GUARD,i)=t \ ] \rightarrow \exists j,t',d[@(\uparrow ACTION,j)=t' \wedge t' \geq t+d \wedge d \geq 0] \quad (7)$$

$$\forall i \ \forall t[@(\uparrow ACTION,i)=t] \rightarrow \exists j,t',d[@(\downarrow ACTION,j)=t' \wedge t' \geq t+d \wedge d \geq \varepsilon]$$

$$\forall i \ \forall t[@(\downarrow ACTION,i)=t] \rightarrow \exists t',d[q(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Structure 3. Being in one state. In real time systems, a state indicates the system state at a time instant/interval .

Visual definition. According to Fig. 2, when the system is in state *p*, the state indicates the current state of the system at that specific moment.

Logical definition of structure 3. This structure shows that state *p* is true at time t. In other words, the system is in state *p* at time t (Relation 8).

$$\forall i \ \forall t[p(t,t)] \quad (8)$$

Structure 4. Remaining in a state and change of the state. The system should stay in a state for a while and then the state should change.

Visual definition. When the system is in state L_1, it should stay in this state at least for time unit d. After passing of the time and satisfaction of the condition, it would enter to state L_2 (Fig. 4).

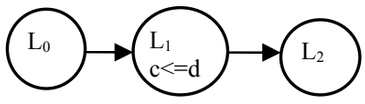


Fig. 4. Change of a state (C indicates clock)

Logical definition of structure 4. If the system is in state L_1 after satisfaction of condition (c<=d) at time t, it would go to a new state (Relation 9).

$$\forall i \ \forall t[L_1(t,t) \wedge @(COND \ (c<=d) \ , \ i) = t] \rightarrow \exists \ t' \ [L_1(t',t') \wedge t' \geq t+d \wedge d \geq \varepsilon] \quad (9)$$

Structure 5. Transition from one state to several states by different events.

Visual definition. In Timed Automata, some state changes may depend on the condition holds on the transition, i.e., if the condition holds, the state change would happen. Some possible target states may be accessible from one source state when different conditions hold (Fig. 5).

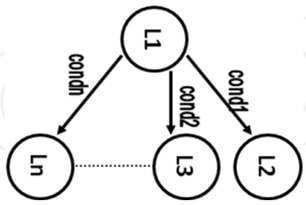


Fig. 5. Change of states by occurrence of event

Logical definition of structure 5. Every time one of the events takes places and according to the occurred event, the system state changes (Relation 10).

$$\forall i, t\ [\ L_0(t,t) \wedge @(COND_1\ , i) = t] \rightarrow \exists\ t',d\ [L_1(t',t') \wedge t' \geq t+d \wedge d \geq 0] \vee \quad (10)$$

$$\forall i, t\ [\ L_0(t,t) \wedge @(COND_2\ , i) = t] \rightarrow \exists\ t',d\ [L_2(t',t') \wedge t' \geq t+d \wedge d \geq 0] \vee$$

$$\forall i, t\ [\ L_0(t,t) \wedge @(COND_n\ , i) = t] \rightarrow \exists\ t',d\ [L_n(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Structure 6. To reach states of a system periodically. When we verify the system behavior, some states may be reached periodically.

Visual definition. In Timed Automata some sets of states and actions might be executed for several times. In such a situation, there will be iterations of some states for several times (Fig. 6).

Logical definition of structure 6. While the system is in state L_n and event $COND_n$ occurs, the system will go to state L_1; this leads to an iteration of states L_1 and L_n. However, if the system is in state L_n and event *condi* take places, the system will go to state L_{n+1} (Relation 11).

$$\forall i\ \forall t\ [L_n(t,t) \wedge @(CONDn\ , i) = t] \rightarrow \exists\ t_1,d\ [L_{n+1}(t_1,t_1) \wedge t_1 \geq t+d \wedge d \geq 0] \vee \quad (11)$$

$$\forall i\ \forall t\ [L_n(t,t) \wedge @(CONDi\ , i) = t] \rightarrow \exists\ t_2,d\ [L_1(t_2,t_2) \wedge t_2 \geq t+d \wedge d \geq 0]$$

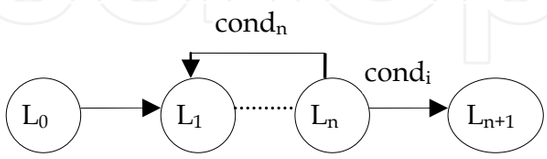


Fig.6. Periodic change of states

4.2 Case study

In this section we propose a critical system and apply the first and second steps (specifying problem in Timed Automata and extracting the logical specification of the problem) of our

method. The case study is an example of an automatic controller in which the *railway* gate opens and closes an intersection. The system contains three parts: *train*, *controller*, and *gate* (Fig.7). The connection between the train and controller is created by two ***exit*** and ***approach*** methods where the former denotes the train exits from the intersection and the latter denotes the train approaches the intersection. It is necessary that the train sends the approach signal less than two minutes before entering the intersection. This has been shown by a *protecting condition* X>2 along with event ***in***. Table 1 shows explanation of states of train, gate and controller.

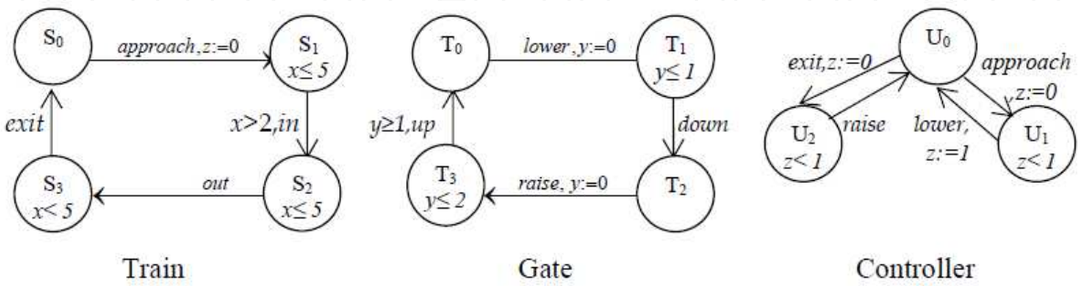


Fig. 7. Train, Gate and Controller states

Train	S_0: the train is far from the intersection S_1:the train is approaching to the intersection S_2: The train is in intersection S_3: The train is leaving the intersection	Controller U_0: the controller is in standby state U_1:The controller is in the lower signal state U_2:The controller is in the raise signal state
Gate	T_0: The gate is in upward state T_1: The gate is coming down T_2: The gate is in down state T_3: The gate is going up	

Table 1. Explanation of Train, Controller and Gate states

In addition, we know that the most delay between *exit* and *approach* signals is five minutes. This has been shown by the conditions X<=5 in transitions between states S_1, S_2 and S3. The gate is opened at T_0 and closed at T_2. The gate answers to ***lower signal*** by the open action in one to two minutes. The system clock is used to show the *constraints*. Initially the controller is in state U_0 (*Idle)*. The controller responses to the gate by sending the *lower signal* when it receives the *approach signal* from the train and the controller responses to the gate by sending the raise signal when it receives the *exit signal* from the train. The response time of the controller for approach and exit signals are one minute and more than one minute respectively; these limitations have been shown by clock *z*.

The system is in the safe state when the train is in the intersection, and the gate is low. In other words, if the train is in state S_1, the gate should be in state T_2. For example, consider paths originate from nodes S_0, T_0 and U_0 and end with nodes S_1, T_0 and U_1 and paths originate from nodes S_1, T_0 and U_1 and end with nodes S_2, T_0 and U_1 representing the event *approach* followed by event ***in*** immediately. However, this sequence cannot be true if we consider clocks. Now based on the former rules we resemble the previous model with real time logic.

Specification 1 [based on Logical definition 3]. Initially, the train, the gate and the controller are in states S_0, T_0 and U_0 respectively.

$$\forall t\ [S_0(t,t)\ \wedge U_0(t,t)\ \wedge T_0(t,t)]$$

Specification 2 [based on Logical definitions 2 and 3]. When the train is in state S_0 and signal (event) *approach* occurs, after a while event ↑Reset(x) of clock *x* occurs. When event ↓Reset(x) occurs and clock *x* is reset by time *d*, the train is in the *approaching* state.

$$\forall i\ \forall t\ [\ S_0(t,t) \wedge @(\text{SIGNAL(approach)}, i) = t\ ] \rightarrow \exists\ j, t', d\ [@(\uparrow \text{RESET}(x), j) = t' \wedge t' \geq t + d \wedge d \geq 0]$$

$$\forall i\ \forall t\ [@(\uparrow \text{RESET}(x)\ , i) = t] \rightarrow \exists j, t', d\ [@(\downarrow \text{RESET}(x), j) = t' \wedge t' \geq t + d \wedge d \geq \varepsilon]$$

$$\forall i\ \forall t\ [@(\downarrow \text{RESET}(x), i) = t] \rightarrow \exists\ t', d\ [S_1(t', t') \wedge t' \geq t + d \wedge d \geq 0]$$

Specification 3 [based on Logical definitions 2 and 3]. When the train is approaching to the intersection, it is in state S_1.

$$\forall t\ [\ S_1(t,t)\ ] \rightarrow \exists t', d\ [S_1\ (t', t') \wedge t' \geq t + d \wedge d \geq 0 \wedge d \leq 5]$$

Specification 4 [based on Logical definitions 2 and 3]. When the controller is in state *standby* (denoted by state U_0) and signal (event) approach occurs, the clock *z* is reset for making a transition from state U_0 to state U_1.

$$\forall i\ \forall t\ [U_0(t,t) \wedge @(\text{SIGNAL(APPROACH)}, i) = t] \rightarrow$$

$$\exists j, t', d\ [@(\uparrow \text{RESET}(z), j) = t' \wedge t' \geq t + d \wedge d \geq 0]$$

$$\forall i\ \forall t\ [@(\uparrow \text{RESET}(z), i) = t] \rightarrow \exists j, t', d\ [@(\downarrow \text{RESET}(z), j) = t' \wedge t' > t + d \wedge d \geq \varepsilon]$$

$$\forall i\ \forall t\ [@(\downarrow \text{RESET}(z), i) = t] \rightarrow \exists\ t'\ [U_1(t', t') \wedge t' \geq t + d \wedge d \geq 0]$$

Specification 5 [based on Logical definitions 1 and 3]. As long as condition *z<=1* holds the controller stays in state U_1.

$$\forall i, t_1, t_2\ [\ U_1(t_1, t_1) \wedge @(\text{COND}(z <= 1), i = t_2)\ ] \rightarrow \exists t', d\ [U_1\ (t', t') \wedge t' \geq t + d \wedge d \geq 0 \wedge d \leq 1]$$

Specification 6 [based on Logical definitions 2 and 3]. Assume that: (1) the controller and the gate are in states U_1 and T_0 at time *t* respectively, (2) at this time signal *lower* is generated and (3) event ↑Reset(y) occurs at time *t'*. Accordingly, the gate goes to state T_1 and clock *y* is reset. If condition *z=1* holds at time *t*, the controller would enter the state at time *t'*.

$$\forall i, j\ \forall t\ [U_1(t,\ t \wedge T_0(t,\ t) \wedge @(\text{SIGNAL(lower)}, i) = t \wedge (\uparrow \text{RESET}(y), j) = t] \rightarrow$$

$$\exists t', q, d\ [T1(t', t') \wedge @(\downarrow \text{RESET}(y), q) = t' \wedge t' \geq t + d \wedge \geq \varepsilon]$$

$$\forall i, j\ \forall t\ [\ U_1(t,\ t) \wedge T_0(t,t) \wedge @(\ \text{SIGNAL(lower)}, i) = t \wedge @(\uparrow \text{RESET}(y)\ , j) = t \wedge$$

$$@(\text{COND}\ (z=1), i) = t\ ] \rightarrow$$

$$\exists t', q, d\ [T_1(t', t') \wedge U_0(t', t') \wedge @(\downarrow \text{RESET}(y), q) = t' \wedge t' \geq t + d \wedge d \geq \varepsilon].$$

Specification 7 [based on Logical definitions 1, 3 and 4]. Gate stays in state T_1 as long as condition *y<=1* holds. In other words, if the gate is in state T_1 at time *t*, it stays in the same state at time t′ while condition *y<=1* holds.

$$\forall i\ \forall t\ [T_1(t,t) \wedge @(\text{COND } (y<=1)\ ,\ i) = t] \rightarrow \exists t',d\ [\ T_1(t',t') \wedge t' \geq t+d \wedge d \leq 1]$$

Specification 8 [based on Logical definitions 1 and 3]. Gate will be in state T_2 at time *t′* if event DOWN occurs at time *t* and after time unit *d* the gate enters state T_2.

$$\forall i\ \forall t\ [T_1(t,t) \wedge @(\text{DOWN}\ ,\ i) = t] \rightarrow \exists t',d\ [\ T_2(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Specification 9 [based on Logical definitions 1 and 3]. Train will be in state S_2 at time *t* if at time *t* event *in* occurs and condition *X>2* holds. After that the train goes state S_2 after time unit *d* if condition *X<=5* holds at the same time.

$$\forall i,j\ \forall t,t',d\ S_1(t,t) \wedge @(\ \text{IN}\ ,\ i) = t \wedge @(\text{COND } (x>2)\ ,\ i) = t \wedge @(\text{COND } (x<=5)\ ,\ i) = t] \rightarrow \exists d,t'[S_2(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Specification 10 [based on Logical definitions 1 and 3]. As long as condition *X<=5* holds and train is in state S_2, train stays in this state.

$$\forall i\ \forall t\ [\ S_2(t,t) \wedge @(\text{COND } (x<=5)\ ,\ i) = t] \rightarrow \exists t',d\ [S_2(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Specification 11 [based on Logical definitions 1 and 3]. Train will be in state S_3 at time *t′* if event OUT have occurred at time *t.* After that, train enters state S_3 after time unit *d* if condition *X<=5* holds.

$$\forall i\ \forall t\ [\ S_2(t,t) \wedge @(\text{COND } (x<=5)\ ,\ i) = t \wedge @(\text{OUT}\ ,\ i)=t\ ] \rightarrow \exists t',d\ [S_3(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Specification 12 [based on Logical definitions 1 and 3]. Train will be in state S_0 at time *t′* if event *exit* occurs at time *t*. After time unit *d*, train will be in state S_0.

$$\forall i\ \forall t\ [S_3(t,t) \wedge @(\text{SIGNAL } (\text{exit})\ ,\ i) = t] \rightarrow \exists t',d\ [S_0(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Specification 13 [based on Logical definitions 2 and 3]. Controller will be in state U_2 at time *t′* if it receives event *exit*. The event occurs due to exiting train from intersection at time *t*. Also if at the same time event *reset* of clock *z* occurs, train will enter the state U_2 after time unit *d* and clock *z* is reset to zero at the same time.

$$\forall i,\ j\ \forall t\ [U_0(t,t) \wedge @(\ \text{SIGNAL } (\text{exit})\ ,\ i) = t \wedge @(\uparrow \text{RESET}(z),\ j)= t\ ] \rightarrow$$

$$\exists t',\ q,\ d\ [U_2(t',t') \wedge @(\downarrow \text{RESET}(z)\ ,\ q)= t' \wedge t' \geq t+d \wedge d \geq \varepsilon]$$

Specification 14 [based on Logical definitions 1 and 3]. Controller stays in state U_2 as long as condition *Z<=1* holds.

$$\forall i \forall t\ [U_2(t,t) \wedge @(\text{COND } (z<=1)\ ,\ i) = t] \rightarrow \exists t',d\ [\ U_2(t',t') \wedge t' \geq t+d \wedge d \leq 1]$$

Specification 15 [based on Logical definitions 1 and 3]. Controller will be in state U_0 after time unit *d* when it receives signal *raise* at time *t*.

$$\forall i\ \forall t\ [\ U_2(t,t) \wedge @(\text{SIGNAL } (\text{raise})\ ,\ i) = t] \rightarrow \exists t',d\ [U_0(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Specification 16 [based on Logical definitions 2 and 3]. Gate goes upward by receiving signal *raise* at time *t* and event ↑RESET(y) resets clock *y* at the same time. After time unit *d* if condition y<=2 holds, the gate enters state T_3 at time *t′* and at the same time event ↓RESET(y) resets clock *y* to zero.

$$\forall i,j,q\ \forall t,t',d\ [T_2(t,t)) \wedge @(\text{SIGNAL(raise)}, i) = t \wedge @(\uparrow\text{RESET(y)}, j) = t \wedge$$

$$@(\text{COND (y<=2)}, q) = t\ ] \rightarrow$$

$$\exists t', m, d[T_3(t',t') \wedge @(\downarrow\text{RESET(y)}, m) = t' \wedge t' \geq t+d \wedge d \geq \varepsilon]$$

Specification 17 [based on Logical definitions 1 and 3]. Gate will be in its first state, T_0, at time *t′* if event *up* occurs at time *t* and at this time, events COND(y<=2) and COND (y>=1) occur. After time unit *d*, the gate enters state T_0.

$$\forall i, j, q\ \forall t\ [\ T_3(t,t) \wedge @(\text{UP}, i) = t \wedge @(\text{COND (y>=1)}, j) = t \wedge @(\text{COND (y<=2)}, q) = t] \rightarrow$$

$$\exists t', d\ [T_0(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

4.3 Extracting reachability graph

In the previous section, we stated textual specification of the Timed Automata of road and railway intersection in RTL. Having extracted reachability graph from Timed Automata, we use RTL to verify those situations in which the system enters *unsafe* states. Fig. 8 shows the reachability graph of road and railway intersection system. The *gray* parts of the graph are *unsafe* states that the system should never enter. To verify the system behavior against the unsafe states, we aim to construct verification rules (constraints). These rules are intended to monitor change of system states against the unsafe states. These rules are based on the structures specified in Section 4.

Verification rule 1. When the train is approaching the intersection, the gate should get lower (state T_1).

$$\forall t\ [S_1(t,t)] \rightarrow \exists t',d\ [T_1(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Verification rule 2. When the train is approaching the intersection, the gate should not be up or should not move up (states T_0 and T_1); otherwise it is an unsafe situation (state H_3 in Fig. 8).

$$\forall t\ [S_1(t,t)\ ] \rightarrow \exists t',d\ [\neg T_1(t',t') \wedge \neg T_3(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Verification rule 3. When train is in intersection (state S_2), the gate should be down (state T_2); otherwise it is an unsafe situation (state H_4 in Fig. 8).

$$\forall t\ [S_2(t,t)] \rightarrow \exists t',d\ [T_2(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Verification rule 4. When train is in intersection (state S_2), the gate should not be up or should not move down; otherwise it is an unsafe situation (states H_4 and H_5 in Fig. 8).

$$\forall t\ [S_2(t,t)\ ] \rightarrow \exists t',d\ [(\neg T_0(t',t') \vee \neg T_1(t',t')) \wedge t' \geq t+d \wedge d \geq 0]$$

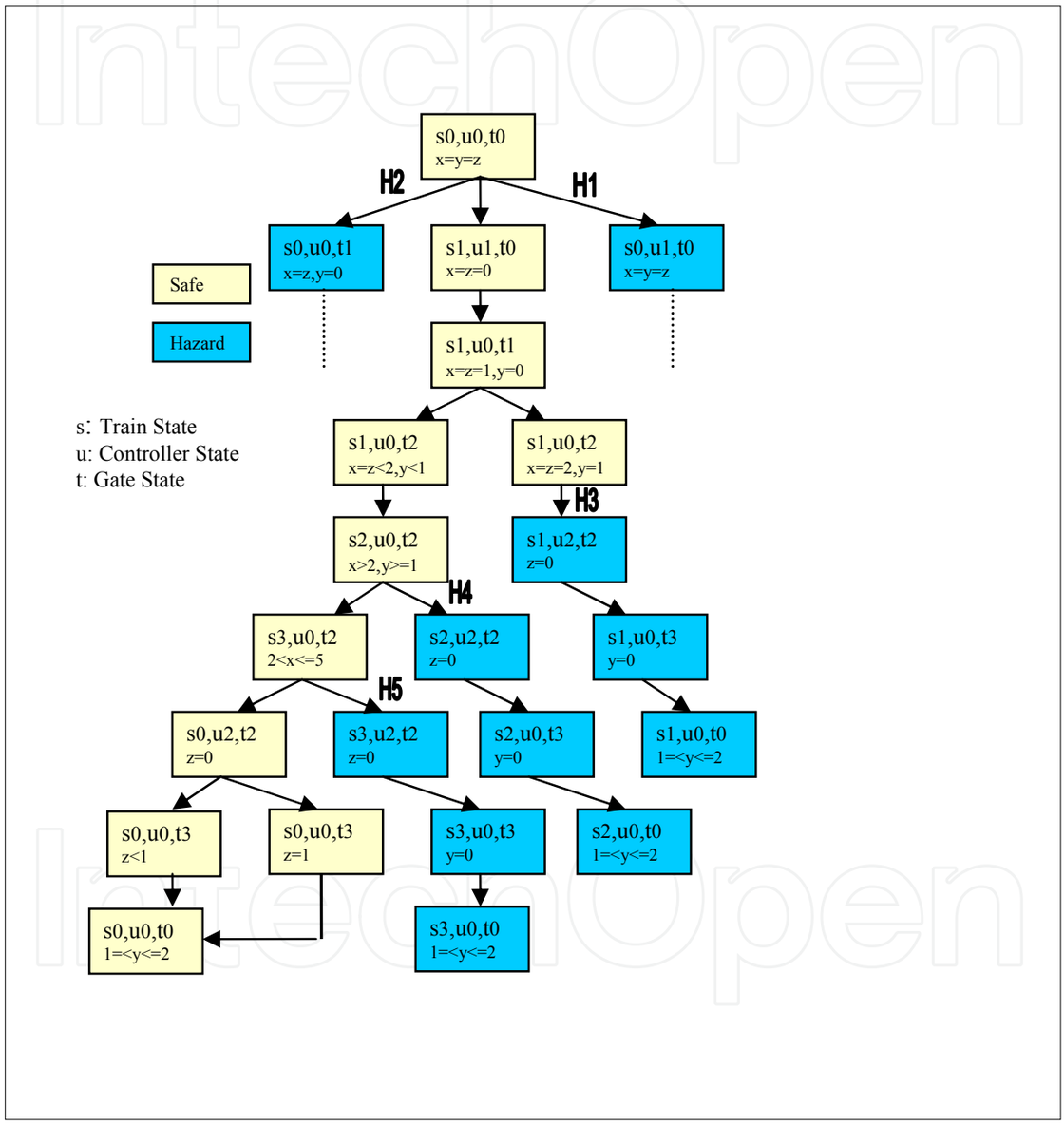


Fig. 8. Reachability graph

Verification rule 5. When train has not left the intersection (state S_3), the gate should be down; otherwise it is an unsafe situation (state H_5 in Fig. 8).

$$\forall t\ [S_3(t,t)] \rightarrow \exists t',d\ [T2(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Verification rule 6. When train has not left the intersection (state S_3), the gate should not up or should not move up; otherwise it is an unsafe situation (state H_5 in Fig. 8).

$$\forall t\ [S_3(t,t)\ ] \rightarrow \exists t',d\ [(\neg T_0(t',t') \vee \neg T_1(t',t') \vee \neg T_3(t',t')\ ) \wedge t' \geq t+d \wedge d \geq 0]$$

Verification rule 7. When train is leaving the intersection, the gate should go up; otherwise it is an unsafe situation (state H_2 in Fig. 8).

$$\forall t\ [S_0(t,t) \wedge U_0(t,t)\ ] \rightarrow \exists t',d\ [T_3(t',t') \wedge t' \geq t+d \wedge d \geq 0]$$

Verification rule 8. When train is not in intersection, the gate should not be down or should not move down; otherwise it is an unsafe situation (states H_1 and H_2 in Fig. 8).

$$\forall t\ [S_0(t,t)\ ] \rightarrow \exists t',d[(\neg T_1(t',t') \vee \neg T_2(t',t')) \wedge t' \geq t+d \wedge d \geq 0]$$

5. Conclusions and related work

In this paper, we dealt with producing verification rules for verification of real-time systems systematically. As a matter of fact, our contribution to the rule production had three aspects. The first one was determination and representation of basic constructions of Timed Automata that were susceptible to making basic verification rules. The second one was producing rules systematically, where it was achieved through reachability graph. The reachability graph as a bridge helped us to extract safety rules from visual specifications. The third aspect was mapping the safety properties obtained from the reachability graph into RTL propositions forming the verification rules. Finally, to show effectiveness of our approach we applied it to a real-time system. In the following, we state the related work.

Jahanian and Mok specified high-level requirements by the Modechart visual and specific language and mapped them to a constraint graph (Chen & Rosu, 2005) where the graph was used by a monitor in order to verify software behavior.

Deriving monitors from requirements has been considered by some researchers. The *Eagle* method used the *Eagle logic* to specify requirements and synthesized a rule based monitor (Barringer et al, 2004). It extended Mu Calculus to support past and future time linear logic and real-time one. *Eagle* exploiting a state-based approach (d'Amiron & Havelund, 2005) was extended by the HAWK logic (d'Amiron & Havelund, 2005). To support event-based specifications, the HAWK language, used to specify both high-level specifications and low-level run-time states, has included low-level programming definitions to verify java programs.

(Mok & Liu, 1997, Chen & Rosu, 2005) used real-time logic (RTL) for the requirements specification of real-time system. Li and Dang presented algorithms to combine automata with *black-box testing* in order to verify safety properties (Li, & Dang, 2006). (Mok & Liu,

1997, Douglass, 1999, Jalili & MirzaAghaei, 2007) exploiting event-based approach for real-time systems, used real-time logic (RTL) for high-level specification of system requirements; so, they were capable of specifying real-time requirements, particularly ordering of events.

6. References

Alur R. & Dill D.L. (1994). A theory of timed automata, *Theoretical Computer Science, Elsevier*, Vol. 126, No. 2, pp. 183-235.

Alur R. & Dill D.L. (1996). Automata-theoretic verification of real-time systems, *Formal Methods for Real-Time Computing*, *Trends in Software Series*, Wiley & Sons, pp. 55-82.

Barringer H. et al. (2004). Rule-Based Runtime Verification, *Proceedings of the 5th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'04),* LNCS2937, Springer, pp. 44-57.

Bellini P. et al *(* 2000). Temporal logics for real-time system specification, *ACM Computing Surveys*, Vol. 32, No. 1, pp. 12-42.

Chen, F. & Rosu, G. (2005). Java-MOP: A Monitoring Oriented Programming Environment for Java, *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 546-550.

d'Amiron, M. & Havelund, K. (2005). Event-Based Runtime Verification of Java Programs, *Proceedings of the 5th Workshop of ICSE (International Conference on Software Engineering) on Dynamic Analysis*, pp. 1-7.

Douglass, B.P. (1999). Doing Hard Time, Developing Real-Time Systems with UML, Objects, Framework and Patterns, Addison-Wesley.

Jahanian-a F. et al (1988). A Graph-theoretic approach for timing analysis in real time logic , *Proceedings of IEEE Real-Time Systems Symposium*.

Jahanian-b F. et al (1988). Formal specification of real-time systems, Technical Report, *University of Texas at Austin Austin*, *USA*.

Jahanian F. (1994). Modechart: a specification language for real-time systems, *IEEE Trans. on Software Engineering*, Vol. 20 , No.12, pp. 933-947.

Jalili S. & MirzaAghaei, M. (2007). RVERL: Run-time Verification of Real-time and Reactive Programs using Event-based Real-Time Logic Approach, *Proceedings of the 5th International Conference on Software Engineering Research, Management & Applications (SERA 2007)*, IEEE Computer Society, pp. 550-557.

Li, C. & Dang, Z. (2006). Decompositional Algorithms for Safety Verification and Testing of Aspect-Oriented Systems, *Proceedings of Formal Approaches to Software Testing and Runtime Verification*, *FATES 2006 & RV 2006*, Springer, Vol. 4262.

Mok, A.K. & Liu, G. (1997). Efficient Run-time Monitoring of Timing Constraints, *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pp. 252-262.

Paneka S. et al (2006). Scheduling and planning with timed automata, *Proceedings of 16th European Symposium on Computer Aided Process Engineering and 9th International Symposium on Process Systems Engineering*, W. Marquardt, C. Pantelides (Editors), Elsevier.

Penczek W. & Polrola A. (2006). Advances in verification of timed petri-nets and timed automata, *Studies in Computational Intelligence*, Springer.

IntechOpen

IntechOpen



Real-Time Systems, Architecture, Scheduling, and Application

Edited by Dr. Seyed Morteza Babamir

ISBN 978-953-51-0510-7
Hard cover, 334 pages
Publisher InTech
Published online 11, April, 2012
Published in print edition April, 2012

This book is a rich text for introducing diverse aspects of real-time systems including architecture, specification and verification, scheduling and real world applications. It is useful for advanced graduate students and researchers in a wide range of disciplines impacted by embedded computing and software. Since the book covers the most recent advances in real-time systems and communications networks, it serves as a vehicle for technology transition within the real-time systems community of systems architects, designers, technologists, and system analysts. Real-time applications are used in daily operations, such as engine and break mechanisms in cars, traffic light and air-traffic control and heart beat and blood pressure monitoring. This book includes 15 chapters arranged in 4 sections, Architecture (chapters 1-4), Specification and Verification (chapters 5-6), Scheduling (chapters 7-9) and Real word applications (chapters 10-15).

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Seyed Morteza Babamir and Mehdi Borhani Dehkordi (2012). Construction of Real-Time Oracle Using Timed Automata, Real-Time Systems, Architecture, Scheduling, and Application, Dr. Seyed Morteza Babamir (Ed.), ISBN: 978-953-51-0510-7, InTech, Available from: http://www.intechopen.com/books/real-time-systems-architecture-scheduling-and-application/construction-of-rea-time-oracle-using-timed-automata-

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2012 The Author(s). Licensee IntechOpen. This is an open access article distributed under the terms of the Creative Commons Attribution 3.0 License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

IntechOpen

IntechOpen