Real-time simulation of concurrent components

OUSMANE KONÉ
University of Bordeaux & Aerospace Valley F-33405 Bordeaux
University of Toulouse & Aerospace Valley F-31062 Toulouse
FRANCE

Abstract: This paper deals with the simulation of interworking components, constrained with hard time requirements. Such components are usually involved in embedded applications and process control systems. They can be interconnected to form powerful, but complex applications. We propose an incremental composition approach of simulation scenarios and show that the proposed approach provides good results against the complexity involved in real-time distributed applications model.

Key–Words: Model Simulation, Real-time, Components

1 Introduction

In order to analyse and then show the features of an application, one can explore a model of the application as if it was really running. Simulation scenarios are execution patterns useful to demonstrate the correctness and the expected requirements of the system [6, 4, 8]. They also permit to exhibit the design errors of the system descriptive model. Moreover, the scenarios may be replayed against some prototypes or concrete implementations in order to experiment their dependability. As the simulation results are used for assessing system reliability, the design of the simulation approach is itself a important issue. The computed patterns must reflect the soundness of the considered descriptive model, in terms of interpretation (semantics), while guaranteeing a good coverage of the model.

For the analysis of composite (local, distributed) systems, many approaches are based on the exploration of the *product* or *global* model of the components descriptive models [2, 9, 3, 5]. This is an intuitive approach which works well when the analysed behaviour is of reasonable size. However, with formalisms such as state and clock based models (e.g. timed automata), the underlying timed behaviour model is generally large with a complexity which is an exponential function depending on the number of states n_s and clocks n_c of the descriptive model (We denote this by $Exp_{sem}(n_s, n_c)$). This is a limitation of the previous approaches against real-time.

Our paper brings a contribution to the previous problem by avoiding the product computation approach. We propose a new simulation approach based on (1) grid automata semantics and (2) composition of previously computed patterns. This enables the design of simulation scenarios against concurrent real-time applications, with low cost. Moreover, it allows reuse of existing simulation scenarios already computed by other means or tools [10]. Our approach is inspired by works like [7, 11] (except that those works did not deal with real-time), but augmented in our paper for taking into account timing requirements as well as the inherent time semantics problem, which is addressed here through grid computation. We exploit the concurrent analysis of grid automata based simulation and the proposed Timed-C method is based on the composition of individual scenarios computed by separate means. We argue that such a method is practical in the sense that it generates a reasonable number of simulation cases while ensuring a good coverage, powerful enough to detect potential errors. To illustrate the results of the method, we also present an example with a real-time connection system implemented with interconnected real-time components.

We introduce some preliminary definitions on the descriptive formalism used. Then we present the different steps and algorithms of our method which results against the case study example are presented in section 4.

2 Models

In this section we present the syntax and semantic of Communicating Timed Automata (CTA). A CTA is essentially a channel system modelelling concurrent interworking timed automata. We use the basic model of automata by Alur and Dill[1]. For the reader not familiar with the notion of Timed Automata we give in the following a short description.

Let $\mathbf{R}^+$ be a set of non-negative real numbers and let X be a set of non negative real-time valued variables called *clocks*. The set of *Guards* $G(X)$ is defined by the grammar $g := x \sim c \mid x - y \sim c \mid g \wedge g \mid true$, where x and $y \in X$, $c \in \mathbf{R}^+$ and $\sim \in \{<, \leq, >, \geq\}$. We denote by $\mathbf{T}$ the finite sequences of elements in $\mathbf{R}^+$ called *time domain*, and by Σ the finite set of *actions*. A *time sequence* over $\mathbf{T}$ is a finite no decreasing sequence $\rho = t_1, t_2, \ldots, t_n$ and a *timed word* $w = (a_1, t_1,), (a_2, t_2) \ldots (a_n, t_n)$ is an element of $(\Sigma \times \mathbf{R}^+)^*$. A *clock valuation* is a function $\nu : X \rightarrow \mathbf{R}^+$, if $\delta \in \mathbf{T}$ the $\nu + \delta$ denotes the valuation such that for each clock $x \in X$, $(\nu + \delta)(x) = \nu(x) + \delta$. If $r \subseteq X$ then $\nu[r := 0]$ denotes the valuation such that for each clock $x \in X \setminus r, \nu[r := 0](x) = \nu(x)$ and for each clock $x \in r$, $\nu[r := 0](x) = 0$. $[r := \infty]\nu$ denotes the valuation such that for each clock $x \in X \setminus r, [r := \infty]\nu(x) = \nu(x)$ and for each clock $x \in r$, $[r := \infty]\nu(x) = \infty$.

Definition 1. A *Timed Automaton* (TA) is a tuple $A = (L, L_0, L_f, X, \Sigma, E, I)$, where L is a finite set of locations, $L_0 (L_f) \subset L$ is a subset of initial (final) locations, X is a finite set of clocks. Σ is a finite set of events. If the set of events (actions) is partitioned in two disjoint subsets $\Sigma^?$ and $\Sigma^!$, where $\Sigma^?$ is the set of *input* actions and $\Sigma^!$ is the set of *output* actions, the TA A is called *Timed Input Output Automaton* (TIOA). $E \subseteq L \times G(X) \times \Sigma \times R(X) \times L$ is a set of edges. We write $l \xrightarrow{a,g,r} l'$ iff $(l, a, g, r, l') \in E$, where $l, l' \in L$ are the source and destination locations, $g \in G(X)$ is a conjunction of constraints in $G(X)$, $a \in \Sigma$ is the action (or event), $r \in R(X)$ is the set of clocks to be *reset*. $I : L \rightarrow G(X)$ assigns invariants to locations.

We use the notation such as $l \xrightarrow{a}$ (resp. $l \not\xrightarrow{a}$) to denote that there exists l' such that $l \xrightarrow{a} l'$ (resp. there is no such l'). This notation naturally extends to time sequences. We write $l \xrightarrow{(a,t)}$ if from location l, a can be executed at time t. A TIOA A is said to be *complete*, if it accepts every action in Σ at every time. It is said to be input-complete if it accepts every input action in $\Sigma_!$ at every time. A TIOA is called deterministic if $\forall l, l', l'' \in L \cdot \forall a \in \Sigma \cdot \forall t \in \mathbf{R}^+ \cdot l \xrightarrow{(a,t)} l' \wedge l \xrightarrow{(a,t)} l'' \Rightarrow l' = l''$. It is called non-blocking if $\forall l \in L, \forall a \in \Sigma^! \cup \mathbf{R}^+ \cdot l \xrightarrow{a}$.

A *Path* P in TA A is a finite sequence of consecutive transitions $l_0 \xrightarrow{g_1,a_1,r_1} l_1 \xrightarrow{g_2,a_2,r_2} l_2 \ldots$. It is said to be *Accepting* if it starts in an initial location ($l_0 \in L_0$) and ends in a final location ($l_f \in L_f$). A *Run* of the automaton along the path P is a sequence of the form $(l_0, \nu_0) \xrightarrow[t_1]{g_1,a_1,r_1} (l_1, \nu_1) \xrightarrow[t_2]{g_2,a_2,r_2} (l_2, \nu_2) \ldots$, where $\sigma = t_1, t_2 \ldots$ is a time sequence in $\mathbf{T}$, and $\nu_i (i = 1, 2 \ldots)$ is a clock valuation such that: $\nu_0(x) = 0, \forall x \in X$; $\nu_{i-1} + (t_i - t_{i-1}) \models g_i$; $\nu_i = [r_i := 0](\nu_{i-1} + (t_i - t_{i-1}))$. The label of the run is the timed word $\omega = (a_1, t_1), (a_2, t_2), \ldots (a_n, tn)$. The set of all timed words in A is denoted $\mathsf{Traces}(A)$. If the path P is accepting the timed word ω then it is said to be accepted by the TA A.

Definition 2. A network of *Communicating Timed Automata* (CTA) is a tuple $(A_1, A_2, \ldots, A_n, c_{1,j}, c_{2,j}, \ldots, c_{n,j})$ where each $A_i = (L_i, L_{i_0}, L_{i_f}, X_i, \Sigma_i, E_i, I_i)$ is a timed automaton and each $c_{i,j}$, $i, j \in 1, 2, \ldots, n$ is an unbounded undirectional channel containing messages sent from A_i to A_j. The set of transitions of CTA is $E \subseteq L \times (1 \ldots n \times !, ? \times \Sigma) \times G(X) \times \Sigma \times R(X) \times L$. Transitions in CTA are labeled by not only inputs or outputs but also information about channels. For example the transition $(l_1, (4!a), \emptyset, \emptyset, l_2)$ of A_3 means that A_3 can move from the location l_1 to the location l_2 sending the action a into the channel $c_{3,4}$ (The transition has no guard, it does not reset any clocks). When such a transition is taken the action a is put into the channel $c_{3,4}$. A transition $(l_3, (2?b), \emptyset, \emptyset, l_4)$ of A_1 means that A_1 can move from l_3 to l_4 and reads b from $c_{2,1}$.

Note that there can be pairs of TA which are not connected by channel (if there is no $c_{1,2}$ between A_1, and A_2), and that there can be channels from the automaton to itself (System $(A_1, c_{1,1})$). Such a system can serve as a model of two timed automata with shared states connected by channel.

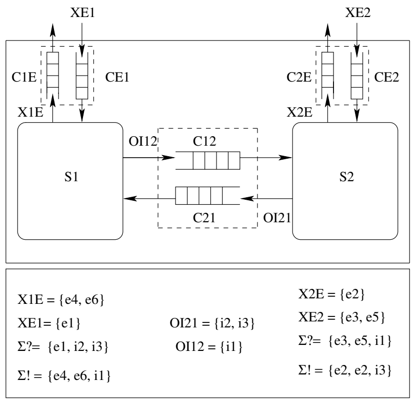

Figure 1: The CTA $(A_1, A_2, c_{1,E}, c_{E,1}, c_{1,2}, c_{2,1}, c_{2,E}, c_{E,2})$

Case study Example. Figure 1 shows an example of communicating timed components. These components communicate by means of the events (i_1, i_2, i_3). They interact with their environment through the events $(e_1, e_2, e_3, e_4, e_5, e_6)$. Events are exchanged trough six unbounded unidirectional channels $(c_{1,E}, c_{E,1}, c_{1,2}, c_{2,1}, c_{2,E}, c_{E,2})$. The behavior of

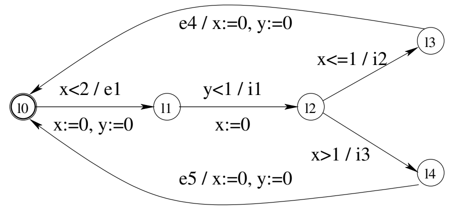

Figure 2: Automaton of the component 1

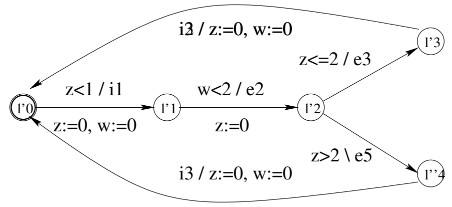

Figure 3: Automaton of the component 2

each component can be modeled by the timed automata of the figure 2 and 3. For space (and readability) reasons, we do not detail the role of the interactions defined in our example. The reader may easily check that these interactions follow a standard connection service, where the service operation is constrained here with real-time requirements. Labels e_i are external service level interactions while i_k are internal interactions exchanged between the concurrent components.

Definition 3. Let $A_1 = (L_1, L_{1_0}, L_{1_f}, X_1, \Sigma_1, E_1, I_1)$ and $A_2 = (L_2, L_{2_0}, L_{2_f}, X_2, \Sigma_2, E_2, I_2)$ be CTA. Let $OI_{1,2} = \Sigma_1^! \cap \Sigma_2^?(OI_{2,1} = \Sigma_2^! \cap \Sigma_1^?)$ The set of actions exchanged between A_1 and A_2 (A_2 and A_1) called *Internal actions*. The *concurrent composition* of A_1 and A_2, denoted $A_1 \parallel A_2$ is defined by the derived Timed automaton $A = (L, L_0, L_f, X, \Sigma, E, I)$ where:

- $L = L_1 \times L_2$ is the set of Global locations. A location is a tuple $(l_1, \omega_1, l_2, \omega_2)$.
- $L_0 = L_1^0, \omega_1^0, L_2^0 \omega_2^0$ is the initial global location where $\omega_1^0 = \omega_2^0 = \emptyset$
- $L_f = L_1^f, \omega_1^f, L_2^f, \omega_2^f$ is the final global location.
- $\Sigma = \Sigma^? \cup \Sigma^!$ where $\Sigma^? = (\Sigma_1^? - OI_{2,1}) \cup (\Sigma_2^? - OI_{1,2})$ and $\Sigma^! = (\Sigma_1^! - OI_{1,2}) \cup (\Sigma_2^! - OI_{2,1})$ is the set of Inputs and outputs respectively.
- $X = X_1 \cup X_2$ is the set of clocks.
- E is the set of global transitions. A global transition is a transition between two global locations caused by a transition in a component of the system. E_i is defined by the following rules:

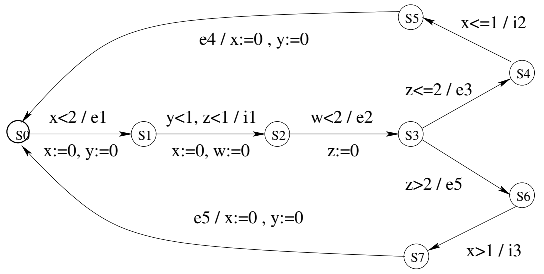

Figure 4: The global automaton

1. $$\frac{(l_1 \xrightarrow{a,g_1,r_1} l_1') \wedge (l_2 \xrightarrow{a,g_2,r_2} l_2')}{(l_1,l_2) \xrightarrow{a,g_1 \wedge g_2, r_1 \cup r_2} (l_1',l_2')}$$
2. $$\frac{(l_1 \xrightarrow{a,g_1,r_1} l_1') \wedge (l_2 \not\xrightarrow{a,g_2,r_2} l_2')}{(l_1,l_2) \xrightarrow{a,g_1,r_1} (l_1',l_2)}$$
3. $$\frac{(l_1 \not\xrightarrow{a,g_1,r_1} l_1') \wedge (l_2 \xrightarrow{a,g_2,r_2} l_2')}{(l_1,l_2) \xrightarrow{a,g_1,r_1} (l_1,l_2')}$$

- $I((l_1, l_2)) = I(l_1) \wedge I(l_2)$ define the invariants to locations in A.

As an example, the concurrent composition of the CTA of the figure1 (model in 2 and 3) can be represented by the figure4. It represent the resulting behavior of the system $A = A_1 \parallel A_2$.

The semantic of CTA is an *infinite labeled transition system* where each state is a tuple $(l_1, \nu_1, \omega_1, \ldots, l_n, \nu_n, \omega_n)$, where l_i is a location of A_i, ν_i is the clock valuation and $\omega_k \in \Sigma$ is the content of channel $c_{i,j}$. For analysing such systems *finite abstraction* of CTA is required. Grid Automata GA is a graph in which the infinite behavior of CTA is abstracted by a finite representation. The grid automaton is constructed from the region automaton[1] by representing each region with a *finite* and *representatives* set of clock valuations. The continous domain of CTA is *sampled* with the granularity (gr). To cover all clock regions, the granularity of sampling should be less or equal to $(\frac{1}{n+1})$ where n is the nomber of clocks. A *tour* in grid automata is a path that starts and ends at the same state. It is called *minimal*, if no edge is contained more than once in the tour. An *initial tour* is a tour that starts and ends at the initial state.

Example. Finite abstraction of TA, Grid Automata. As the number of clocks in each automaton is 2, the granularity of sampling is equal to 0.33. Figures 5 represents the grid automata of component 1 in our example. As an example of initial tours we have $(l_0, (0, 0), \emptyset) \xrightarrow{0.66} (l_0, (0.66, 0.66), e_1) \xrightarrow{e1} (l_1, (0, 0), i_1) \xrightarrow{0.33} (l_1, (0.33, 0.33), i_1) \xrightarrow{i_1} (l_2, (0, 0.33), i_2) \xrightarrow{i_2} (l_3, (0, 0.33), e_4) \xrightarrow{0.33}$

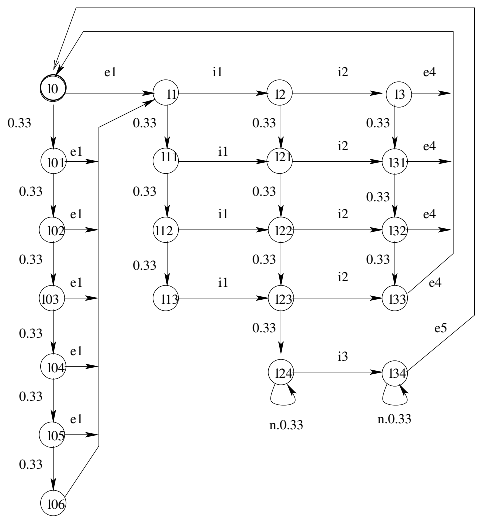

Figure 5: an example of finite abstraction of the component 1

$(l_3,(0.33,0.66),e_4) \xrightarrow{e_4} (l_0,(0,0),\emptyset)$, and $(l'_0,(0,0),\emptyset) \xrightarrow{0.99} (l'_0,(0.99,0.99),\emptyset) \xrightarrow{i1} (l'_1,(0,0),i_1) \xrightarrow{1.99} (l'_1,(1.99,1.99),i_1) \xrightarrow{e2} (l'_2,(0,1.99),\emptyset) \xrightarrow{0.33} (l'_2,(0.33,1.33),e_5) \xrightarrow{e_5} (l_4,(0.33,1.99),\emptyset) \xrightarrow{e_4} (l'_0,(0,0),\emptyset)$.

3 Timed C-Method

3.1 Rationale and prerequisite

In general, Communication timed components are simulated from the global timed automaton constructed from the product of all automata in the whole system. However this approach has some limits; the combinatory explosion problem where it leads to a huge number of simulation cases; The indeterminism due to the concurrency of the composite system; and the repetition of all simulation cases already executed at the component level. The Timed C-Method is proposed to avoid this limits. It is applicable when the implementations of the design components remains unchanged. The Timed C-Method satisfies the following properties:

- It is not necessary to compute the product of components (global Automaton);
- Only simulation cases checking the correctness of the implementation of the composition operator are generated;
- Simulations already performed at the component level (only one component) are not repeated.

Definitions: In the following we give some definitions and notations we use to derive scenarios for CTA.
An *augmented timed scenario case* (ttc) is an non empty timed word $(a_1,t_1)(a_2,t_2),\dots,(a_n,t_n)$.
An augmented timed scenario suite tts is a non empty set of augmented timed scenario cases $\{ttc_1, ttc_2, \dots, ttc_n\}$.
The *concurrent composition* of augmented scenario cases ttc_1 and ttc_2, where ttc_1 is a timed scenario case of A_1 and ttc_2 is the timed scenario case of A_2, is a path $cttc_{1,2}$ obtained by sequencing the elements in ttc_1 and ttc_2 according to the following constraints:

- Any internal input action should be consumed only after its corresponding internal output action is produced $(t_{!a} \geq t_{?a})$ where $a \in \Sigma$.
- The order of timed input actions of ttc_1 and ttc_2 is preserved ;
- The order of timed output actions of ttc_1 and ttc_2 is preserved ;
- the result of concurrent composition can be represented by the timed word $(a_1,t_1)(a_2,t_2),\dots,(a_n,t_n)$ where $t_1 \leq t_2 \leq,\dots,\leq t_n$.

Examples:
from grid automata of A_1 and A_2 we can generate the following augmented timed scenario cases:
- $ttc_{1,1} = (?e_1,0.66)(!i_1,0.99)(?i_2,1.66)(!e_4,1.99)$
- $ttc_{2,1} = (?i_1,0.33)(!e_2,1.33)(?e_3,1.66)(!i_2,1.99)$
- $ttc_{2,2} = (?i_1,0.99)(!e_2,1.33)(?e_3,1.33)(!i_2,1.66)$
- $ttc_{2,3} = (?i_1,0.99)(!e_2,1.33)$

The concurrent composition of $ttc_{1,1}$ and $ttc_{2,1}$ is unfeasible because the internal action i_1 in ttc_2 can be consumed only before 0.33 *tu* whereas it is produced by ttc_1 only after 0.99 *tu*. The concurrent timed scenario case $cttc_{1,3} = ttc_1 \parallel ttc_3 = (?e_1,0.66)(!i_1,0.99)(?i_1,0.99)(!e_2,1.33)(?e_3,1.33)(!i_2,1.66)(?i_2,1.66)(!e_4,1.99)$ is a feasible and meaningful composition of ttc_1 and ttc_3.
Definition. The concurrent composition of two augmented timed scenario cases is called *complete* if all their internal actions are included, and the input queues of the corresponding TA will be empty after their execution. Otherwise, it is called *incomplete*.
For example, the scenario $cttc_{1,3}$ is complete but the scenario $cttc_{1,4} = ttc_1 \parallel ttc_2 = (?e_1,0.66)(!i_1,0.99)(?i_1,0.99)(!e_2,1.33)$ is incomplete.

3.2 Algorithm based on coverage criterion

To obtain a meaningful composition of CTA, some criteria should be satisfied; the internal actions of

either automaton should be consumed by the other automaton, real time constraints should be respected when internal inputs (outputs) are consumed (produced), and the composed system should be free of internal deadlocks.
To check whether two CTA meet these criteria, we assume that they are always capable to return (resynchronize) to their initial states. If this assumption is satisfied, it suffices to consider the initial tours of both automata, and to check whether for each initial tour, there exist its corresponding initial tour in the other automaton such that their concurrent composition is complete. The initial tour coverage tree is a tree containing all minimal initial tours such that every edge is covered at least once and no tour is contained as a prefix or suffix of another tour. The algorithms 1 and 2 show how to construct the initial tour coverage.

Algorithm 1

Input: A_G : Grid Automaton of A
Output: $H(s_i)$: Minimal tree covers all cycle free paths of A_G leading from each state s_i to the initial state s_0

```
1. $H(s_i)$: Minimal tree covers all cycle free paths leading from the state $s_i$ to the initial state $s_0$
2. $M(s_i)$: The set of states already visited.
3. $R(s_i)$: The set of successor states of $s_i$
4. For each state $s_i$.
5. ...$H(s_i) = s_i$
6. ...$M(s_i) = \{s_0, s_i\}$.
7. ...Repeat
8. ......$R(S_i) = Succ(s_i)$
9. ......For each $s_j \in R(s_i)$
10. .........if $\{s_j\} \cap M(s_i) \neq \emptyset$
11. ............$H(s_i) \leftarrow H(s_i) \cup ((s_i, a, s_j) \wedge s_j)$, where $a \in \Sigma \cup tick$ (i.e, added the transition ............$(s_i, a, s_j)$ and the state $s_j$ to the tree $H(s_i)$)
12. ............continue
13. .........else
14. ............$M(s_i) \leftarrow M(s_i) \cup \{s_j\}$
15. ............$H(s_i) \leftarrow H(s_i) \cup ((s_i, a, s_j) \wedge s_j)$
16. ............$i \leftarrow j$
17. ......FFor
18. remove all leafs $\neq s_0$ in $H(s_i)$
19. return $H(s_i)$
20. FFor
```

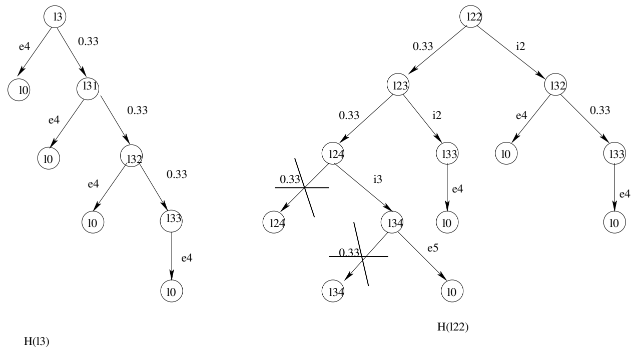

Figure 6: Minimal trees of states $l3$ and $l_{2,2}$

Example: the minimal trees cover all cycle free paths leading from the states l_3 and $l_{2,2}$ to the initial state l_0 in the grid automaton of the figure 5 is shown in Figure 6.

Algorithm 2

Input: A_G : Grid Automaton of A and $H(s_i)$
Output: $ITCT$ Initial tour coverage tree

```
1. For each state $s_i$.
2. ...$ITCT = s_0$
3. ...$M = \{s_0\}$.
4. ...Repeat
5. ......$R(S_i) = Succ(s_i)$
6. ......For each $s_j \in R(s_i)$
7. .........if $\{s_j\} \cap M \neq \emptyset$
8. ............$ITCT \leftarrow ITCT + ((s_i, a, s_j) \wedge s_j)$, where $a \in \Sigma \cup tick$
9. ............continue
10. .........else
11. ............$M \leftarrow M \cup \{s_j\}$
12. ............$ITCT \leftarrow ITCT + ((s_i, a, s_j) \wedge s_j)$
13. ............$i \leftarrow j$
14. ......FFor
15. For each leaf $s_k$ in $ITCT$
16. ...$s_k \leftarrow H(s_k)$
17. return ITCT
18. FFor
```

Algorithm 1 construct a minimal tree that can cover all cycle free paths of A_G leading from each state s_i to the initial state s_0. This tree is used to construct Initial tour coverage tree (Algorithm 2). Both algorithms 1 and 2 are used to select scenario cases of the compositional system (Algorithm 3)

Algorithm 3 :

Input: GA_1 and GA_2 : Grid automata of A_1 and A_2
Output: $ctts_{1,2}$: Augmented timed concurrent scenario suite of the CTA $A_1 \parallel A_2$

1. Initialization : $ctts_{1,2} = \emptyset$
2. Construct $ITCT_1$ (resp. $ITCT_2$) from GA_1 (resp. GA_2); The initial tour coverage tree of A_1 (A_2) using Algorithms 1 and 2.
3. Extract tts_1 (resp. tts_2) from $ITCT_1$ (resp.$ITCT_2$) : All accepting paths in $ITCT_1$
4. From $tts_1(tts_2)$ remove all augmented scenarios containing only external actions
   (Concern only one component but not the composition)
5. Remove the maximum suffix containing only external actions.
   (Concern only one component but not the composition)
6. For each $ttc_{1,i}$ from the resulting tts_1, find the set of corresponding scenario cases (denoted $tts_{2,(1,i)}$) from tts_2, such that For each $ttc_{2,(1,i),j} \in tts_{2,1,i}$, $ttc_{1,i} \parallel ttc_{2,1,i,j}$ is complete.
7. From $tts_{2,(1,i)}$, select only scenario cases such that the composition $cttc_{1,2,i,k} = ttc_{1,i} \parallel ttc_{2,(1,i),k}$ forms a timed word (i.e, $t_1 \leq t_2 \leq , \ldots, \leq t_n$).
8. $ctts_{1,2} \leftarrow ctts_{1,2} \cup cttc_{1,2,i,k}$
9. Return $ctts_{1,2}$

4 Results analysis

From the initial tour coverage tree constructed from grid automaton of the first component (Figure 5), we can derive the following simulation scenario:

$- ttc_{1,i} = (?e_1, 0.66)(!i_1, 0.66)(?i_2, 1.66)(!e_4, 1.99)$

The set of its corresponding scenario cases denoted $tts_{2,(1,i)}$ in tts_2 is:

$- tts_{2,(1,i)} =\{$

$(?i_1, 0.00)(!e_2, 0.00)(?e_3, 0.00)(!i_2, 0.00)$

$(?i_1, 0.33)(!e_2, 0.33)(?e_3, 0.33)(!i_2, 0.33)$,

$(?i_1, 0.66)(!e_2, 0.66)(?e_3, 0.66)(!i_2, 0.66)$,

$(?i_1, 0.99)(!e_2, 0.99)(?e_3, 0.99)(!i_2, 0.99)$,

$(?i_1, 0.00)(!e_2, 0.33)(?e_3, 0.33)(!i_2, 0.33)$,

$(?i_1, 0.00)(!e_2, 0.66)(?e_3, 0.66)(!i_2, 0.66)$,

$(?i_1, 0.00)(!e_2, 0.99)(?e_3, 0.99)(!i_2, 0.99)$,

$(?i_1, 0.33)(!e_2, 2.33)(?e_3, 2.33)(!i_2, 2.33)$,

$(?i_1, 0.66)(!e_2, 0.2.66)(?e_3, 2.66)(!i_2, 2.66)$,

$(?i_1, 0.66)(!e_2, 0.1.66)(?e_3, 1.66)(!i_2, 1.66)$,

$(?i_1, 0.66)(!e_2, 0.1.99)(?e_3, 1.99)(!i_2, 1.99)$,

$(?i_1, 0.99)(!e_2, 0.1.99)(?e_3, 1.99)(!i_2, 1.99)$,

$\ldots\}$

From this subset we can derive only three concurrent simulation cases that meet the conditions of the section 3. They are displayed in figure 7.

The reader may easily check that the resulting patterns are sound in the sense that they actually correspond to execution schemes of the concurrent systems presented before in section 2. The result shows that we did not require to compute the product model of the sub-components. The grid computation of such product model would be as complex as $Exp_{Grid}(25, 4)$: Indeed, each of the components has 5 states and 2 clocks (Cf section 2, figure 2 and figure 3), which involves a composite model with $(2+2 = 4)$ clocks, a state space of $(5 \times 5 = 25)$, and 8 accessible global states (figure 4). Our method produces a full coverage simulation patterns with grid automata construction restricted to a complexity of $Exp_{Grid}(5, 2)$ for each single component, where the component has 5 states and 2 clocks (Cf section 2).

$$cttc_{1,2,1} = (?e_1, 0.66)(!i_1, 0.66)(?i_1, 0.66)(!e_2, 0.66)(?e_3, 0.66)(!i_2, 0.66)(?i_2, 1.66)(!e_4, 1.99)$$

$$cttc_{1,2,2} = (?e_1, 0.66)(!i_1, 0.66)(?i_1, 0.99)(!e_2, 0.99)(?e_3, 0.99)(!i_2, 0.99)(?i_2, 1.66)(!e_4, 1.99)$$

$$cttc_{1,2,3} = (?e_1, 0.66)(!i_1, 0.66)(?i_1, 0.66)(!e_2, 1.66)(?e_3, 1.66)(!i_2, 1.66)(?i_2, 1.66)(!e_4, 1.99)$$

Figure 7: Simulation cases results

5 Conclusion

For the analysis of concurrent complex applications, one must investigate design strategies to either reduce the complexity or to avoid it. The approach presented in this paper attempts to avoid the complexity related to the construction of the global product of the communicating components. Instead of constructing such product, we compute simulation scenarios for each single subsystem. And after that, we compose some specific patterns selected from each separate suite that involve actual interaction between the subcomponents. Another advantage of this approach is also the possibility of reusing existing scenario (pre-computed by other means) which may be furtherly composed. The results of the presented method are sound execution schemes based on grid semantics, produced with low complexity. The method has been illustrated with real-time concurrent component and the compositional simulation approach shows a promising direction to the simulation of large behaviour of interconnected real-time components.

References:

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

[2] Anagnos *Introducing a UML model for faster-than-real-time simulation.* Proc. Winter Simulation Conference, Orlando, FL, 2005

[3] R. Cardell-Oliver, *Conformance Tests for Real Time Systems with Timed Automata Specification*. Formal Aspect of Computing Journal, 350-371. 2000.

[4] J.S.Carson II. *Introduction to modelling and simulation.* Proc. Winter Simulation Conference, Orlando, Florida, 2005

[5] A. En-Nouaary and G. Liu : Timed Test Cases Generation Based on MSC-2000 Test Purposes, in Workshop on Integrated-reliability with Telecommunications and UML Languages (WITUL'04), part of the 15th IEEE International Symposium on Software Reliability Engineering (ISSRE), Rennes, France, November 2004.

[6] R. M. Fujimoto. *Parallel discrete event simulation.* Communications of the ACM, 33 (10) 1990.

[7] R. Gotzhein and F. Khendek, *Compositional Testing of Communication Systems*, Proceedings of IFIP Testcom 2006, Lecture Notes in Computer Science 3964, Springer, New York, NY USA, May 2006.

[8] O.Koné, *Controlled simulation of real-time systems.* The Eighth IASTED International Conference on Control and Applications, Canada, May 2006.

[9] R. Castanet, O. Kone and P. Laurencot, *On-the-Fly Test Generation for Real-Time Protocols*. IEEE International Conference on Computer Communication and Networks. Lafayatte, 1998.

[10] J. Kovács, I. Benyó, and G. Lipovszki *RST Simulator – An Advanced Tool for Control Education.* IASTED Conference on Applied Simulation and Modelling - Greece, ASM 2004.

[11] F. Moradi, P. Nordvaller, and R. Ayani Simulation Model Composition Using BOMs Proc. 10-th DS-RT 2006: Torremolinos, Spain