COMPILING REAL-TIME SCENARIOS INTO A TIMED AUTOMATON*

Aziz Salah, Rachida Dssouli, Guy Lapalme
Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, H3T 1J4 Montreal, PQ, Canada
{salah,dssouli,lapalme}@iro.umontreal.ca

Abstract In this paper, we aim at synthesizing an executable specification for a real-time system by integrating real-time scenarios into a timed automaton. A scenario represents a partial description of a system behavior. A formal semantics is given for the model of a scenario and is used to compile a scenario into a timed automaton. The compilation algorithm is generalized to integrate several scenarios into a single timed automaton which simulates the behaviors specified by the scenarios. The results of the compilation algorithm are independent of the order in which the scenarios are added.

Keywords: real-time system, timed automata, formal specification, scenarios integration

1. INTRODUCTION

A real-time reactive system interacts with its environment under strict timing constraints. The formal specification of such systems is a difficult task that often designers prefer to avoid to move directly from informal specifications to implementation. The detection of design errors and thus the reliability of the system become difficult to evaluate increasing the development costs.

A scenario is a natural means to specify interactions of a system with its environment. The use of scenarios reduces the complexity of systems' specification since a scenario specifies a partial behavior of the system. Our main motivation is to ease the specification of real-time reactive systems by automating some activities in the specification process. Our approach consists of synthesizing an executable specification from a set

*Partially supported by France Telecom

of provided scenarios. The resulting system specification represents a prototype in the form of a timed automaton [AD94].

Most approaches based on scenarios [HSG+94, Gli95, SDLV00] concentrate on the requirements acquisition phase at the very beginning of the system development process. Scenario based approaches may be compared based on many criteria like, for instance, the input scenarios formalism, the method of integrating scenarios and its target formalism. The method of integration is the most important criterion since any approach is based on it. The integration of scenarios consists of merging many scenarios into a global model expressed in the target formalism. The main challenge is how to characterize the steps of a scenario so that occurrences of the same steps can be identified in other scenarios. Some approaches use a manual labeling to identify scenarios' steps while others use regular grammar[HSG+94]. Our approach uses the values of variables, which define the state of the system, to characterize scenarios' steps. We consider two kinds of variables, discrete variables describing system properties and continuous variables (clocks) measuring time.

A scenario is defined by sequences of interactions between the system and its environment. This work is limited to sequential systems in which only one interaction is allowed to be performed at a time. We consider two basic types of interactions namely the reception and the transmission of a message which are both viewed as observable actions. We define a model of scenarios and then we define its formal semantics which is used for compilation. To compile a single scenario into a timed automaton, the scenario is first transformed into an intermediate flat format which we will refer to by the set of *rule-actions* of the scenario. When several scenarios are integrated, the set of all rule-actions resulting from all scenarios is compiled into a single timed automaton. Our method of scenario integration is insensitive to the order of adding scenarios so the specification of a system can be built incrementally by adding scenarios to the current prototype.

The rest of the paper is organized as follows. In section 2, we present preliminaries and notations related to the definition of the timed automata model and discrete variables of a system. In section 3, we describe the syntax of a system specification, the formal semantics of a scenario and define the timed automaton of a scenario. Section 4 is dedicated to the construction algorithm of the scenario timed automaton. We illustrate, in section 5, the use of our algorithm to synthesize a timed automaton of a scenario. Section 6 generalizes this algorithm to compile several scenarios into one timed automaton and discusses the approach.

2. PRELIMINARIES

This section is dedicated to definitions and related notations. We first recall briefly the timed automata model and semantics, and then we present system discrete variables syntax.

2.1. Timed Automata

Timed Automata (TA) model uses dense model of time for modeling timed control. TA [AD94] is an extended labeled transition system with a finite number of real variables called clocks. Clocks values increase at the same rate. Let H be the set of clock variables and $\Phi(H)$ the set of clock constraints. Each $\phi \in \Phi(H)$ is defined by the grammar $\phi ::= x\#c \mid x - y\#c \mid \phi \wedge \phi \mid True$, where x, y are clocks in H and $\# \in \{\leq, <, =, \geq, >\}$ and c a constant in $\mathbb{N}$. A clock interpretation θ is a mapping from H to the set of non-negative reals. $\Theta(H)$ denotes the set of clock interpretations. For each $\theta \in \Theta(H)$ and a clock constraint $\phi \in \Phi(H)$, $\phi(\theta)$ is a boolean value describing whether θ satisfies ϕ or not. Given a non negative real d, $\theta + d$ denotes the clock interpretation which maps each clock x to the value $\theta(x) + d$.

For $\lambda \subset H$, $\theta[\lambda]$ is the clock interpretation which assigns the value 0 to each $x \in \lambda$ and agrees with θ over the other clocks. λ represents a clock assignment. Given a clock constraint $\phi \in \Phi(H)$, $\phi[\lambda]$ denotes the clock constraint satisfied by all interpretations $\theta[\lambda]$ such that θ satisfies ϕ.

Definition 1. *Each Timed Automaton (TA) is a tuple* $A = (L_A, L^o{}_A, M_A, T_A, H_A)$ *where,*

- L_A *is a finite set of locations*
- $L^o{}_A \subset L_A$ *is a set of initial locations*
- M_A *is a finite set of labels*
- H_A *is a finite set of clocks*
- $T_A \subset L_A \times M_A \times \Phi(H_A) \times 2^{H_A} \times L_A$ *is the set of transitions*
- *Inv is a mapping that assigns to each location in* L_A *a clock constraint from* $\Phi(H_A)$. *This constraint is called the invariant of the location.*

Definition 2. *A labeled transition system is a tuple* $\Sigma = (Q, Q_o, \rightarrow)$ *where* Q *is a set of states,* Q_o *is the set of initial states and the relation* $\rightarrow \subset Q \times Act \times Q$ *is the set of transitions, where Act denotes the set of labels. We write* $q \xrightarrow{a} q'$ *iff the transition* $(q, a, q') \in \rightarrow$.

To define the semantics of a TA A, a labeled transition system Σ_A is associated with it. Each state of Σ_A is a pair $(s, \theta) \in L_A \times \Theta(H_A)$ such that θ satisfies $Inv(s)$. Σ_A has two types of transitions:

- transitions modeling a time elapsing of a duration d: $(s, \theta) \xrightarrow{d} (s, \theta + d)$, such that for all $d' \leq d$, $\theta + d'$ satisfies $Inv(s)$, and
- transitions $(s, \theta) \xrightarrow{m} (s', \theta[\lambda])$ such that $(s, m, \phi, \lambda, s') \in T_A$, θ satisfies $\phi \wedge Inv(s)$ and $\theta[\lambda]$ satisfies $Inv(s')$.

In a discrete transition system, traces describe system runs. For a TA A, the notion of next state has no meaning, in fact, a transition $e \xrightarrow{d} e'$ of Σ_A models that the system passes through all states $e + d'$ such that $0 \leq d' \leq d$. We introduce now the notion of step as in [HNSY94, ACD93] to define a TA run.

Definition 3. *Let A be a TA, Σ_A its associated transition system and e, e' two states of Σ_A. e' is reachable in one step from e, denoted $e \overset{d,m}{\triangleright} e'$, if Σ_A allows the transitions $e \xrightarrow{d} e + d$ and $e + d \xrightarrow{m} e'$ for $d \geq 0$ and $m \in M_A \cup \{0\}$. A run σ is a (finite or infinite) sequence of steps $\sigma = e_0 \overset{d_1,m_1}{\triangleright} e_1 \overset{d_2,m_2}{\triangleright} e_2 \overset{d_3,m_3}{\triangleright} \ldots$*

Definition 4. *let $\sigma = e_0 \overset{d_1,m_1}{\triangleright} e_1 \overset{d_2,m_2}{\triangleright} e_2 \overset{d_3,m_3}{\triangleright} e_3 \ldots$ be a run. The timed trace of σ is defined as the sequence $((\theta_1, l_1) \ldots (\theta_j, l_j) \ldots)$ such that if $e \overset{d,m}{\triangleright} e'$ the jth element of σ for which $e = (s, \theta)$ and $m \in M_A$ then $\theta_j \overset{def}{=} \theta + d$ and $l_j \overset{def}{=} m$.*

2.2. Discrete Variable Constraints and Assignments

Let v be a system discrete variable in V. Let $Dom(v)$ denote a finite and discrete domain of v values. Let $\Psi(V)$ denote the set of variable constraints ψ defined by the grammar $\psi ::= v \# c \mid \psi \wedge \psi \mid \neg\psi \mid True$, where $v \in V$, $c \in Dom(v)$ and $\#$ is a binary relation of $Dom(v) \times Dom(v)$. Let $\Omega(V)$ be the set of variable interpretations. Each variable interpretation $\omega \in \Omega(V)$ maps a value in $dom(v)$ to each variable in V. $\psi(\omega)$ denotes a boolean value describing whether ω satisfies ψ or not.

Variable assignments allow variable modifications as a transformation functions. For example, given $v_1, v_2 \in V$ and an assignment $\delta = \{v_1 := v_2 + 3, v_2 := 1\}$ and a variable interpretation $\omega \in \Omega(V)$, $\omega[\delta]$ denotes a new variable interpretation defined by $\omega[\delta](v_1) = \omega(v_2) + 3$, $\omega[\delta](v_2) = 1$ and $\omega[\delta](v) = \omega(v)$ for $v \in V - \{v_1, v_2\}$. There is no limitation on assignment statements expressions but the resulting value of each variable

```
V={A_status, A_signal, B_status, B_signal}
Dom(A_status)  = {BUSY, IDLE}
Dom(A_signal)  = {NONE,TONE,BUSY_TONE,DIALING(B),ECHO_RING(B),TALKING(B)}
Dom(B_status)  = {BUSY , IDLE} ;
Dom(B_signal)  = {NONE,TONE,BUSY_TONE,RING(A),TALKING(A)};
H={h}
```

Figure 1. An example of discrete variables set V and clocks set H

must remain in its domain. We write $\Delta(V)$ to denote the set of variable assignments. Given a constraint variable $\psi \in \Psi(V)$ and an assignment $\delta \in \Delta(V)$, the constraint $\psi[\delta] \in \Psi(V)$ designates the constraint satisfied by $\omega[\delta]$ such that the variable interpretation $\omega \in \Omega(V)$ satisfies ψ. Figure 1 shows the example of a description of a set V and a set H for a telephone switch controller system.

In the remaining part of this document we'll adopt the following notations: θ denotes a clock interpretation in $\Theta(H)$, ω represents a variable interpretation in $\Omega(V)$, φ and ϕ denote clock constraints in $\Phi(H)$, Ψ represents a variable constraint in $\Psi(V)$, $\delta \in \Delta(V)$ denotes a variable assignment and $\lambda \subset H$ represents a clock assignment.

3. SPECIFYING A REAL-TIME REACTIVE SYSTEM BEHAVIOR

A real-time reactive system is a process characterized by a continuous interaction with its environment under strict timing constraints. This work is restricted to sequential processes. Process behaviors are specified by a set of scenarios. Each scenario is represented by sequences of actions. An action in a scenario describes the states of the process before and after the execution of the action.

Our main goal is to synthesize a TA from the provided scenarios. This TA represents a prototype of the desired system. Let P be the system process. P is a tuple (S, H, V, Act) where S is a set of scenarios, V a set of discrete system variables, H a set of clocks and Act a set of observable labels used during interaction with the environment. The syntax for clocks in H and discrete variable in V have been already defined in subsections 2.1 and 2.2 respectively. In the remainder of this section, we define scenarios syntax and their semantics.

3.1. Scenario Formalization

The scenarios in S describe possible behaviors of a process P. Each scenario action describes the changes in the process's state after the

execution of the action or after a deadline. Each state of the process is a pair $e = (\omega,\theta) \in \Omega(V) \times \Theta(H)$ composed of a variable interpretation ω and a clock interpretation θ. Let us define some notations: $\psi(e) = \psi(\omega)$, $\varphi(e) = \varphi(\theta)$, $e[\delta] = \omega[\delta]$ and $e[\lambda] = \theta[\lambda]$.

Definition 5. *A scenario sc is a tree* $sc = (N, \mapsto, A)$ *where:*

- $N = N_p \cup N_s$ *such that* N_p *and* N_s *are respectively the sets of primary and secondary vertices of sc. We write* $N_p = [N_1, N_2, \ldots, N_n, N_{n+1}]$, $n + 1 = |N_p|$. *Secondary vertices are leaves of the tree.*
- *A is the set of actions that label the tree edges. Each* $a \in A$ *is a tuple* $(\varphi_a, \psi_a, \phi_a, lab_a, \delta_a, \lambda_a)$ *where:*
  - $\varphi_a \in \Phi(H)$ *and* $\psi_a \in \Psi(V)$ *are constraints on the state of process.*
  - $lab_a \in Act$ *is a label of an observable action*
  - $\phi_a \in \Phi(H)$ *is an enabling constraint of the action a*
  - $\delta_a \in \Delta(V)$ *is an assignment updating the state of the process after the execution of the action a.*
  - λ_a *is a clock assignment that occurs after the execution of the action a.*
- $\mapsto \subset N \times A \times N$ *is the set of edges.*
- *Each primary vertex* N_i *for* $1 \le i \le n$: $N_i = [a_{i1}, \ldots, a_{ik_i}]$ *such that* $(N_i, a_{i1}, N_{i+1}) \in \mapsto$. a_{i1} *is called the primary action of the vertex* N_i.

For $1 < j \le k_i$, the actions a_{ij} of the vertex N_i are alternatives of the primary action a_{i1}. These alternatives model timers expirations or any other interruption.

The semantics of a scenario action is formalized by associating with each primary vertex a pair of constraints $(\psi N_i, \varphi N_i) \in \Psi(V) \times \varphi N_i \in \Phi(H)$ defining the context of the vertex. *To execute an action, the state of the process has to satisfy the context constraints of its vertex.* The context constraints of a vertex are defined by the following sequence:

$$(\psi_{N_1}, \varphi_{N_1}) = (\psi_{a_{11}}, \bigwedge_{1 \le k \le k_1} \varphi_{a_{1k}}), \text{ and for } 2 \le i \le n$$

$$(\psi_{N_i}, \varphi_{N_i}) = (\psi_{N_{i-1}}[\delta_{a_{(i-1)1}}] \wedge \psi_{a_{i1}}, \bigwedge_{1 \le k \le k_i} \varphi_{a_{ik}} \wedge (\varphi_{N_{i-1}} \wedge \phi_{a_{(i-1)1}})[\lambda_{a_{(i-1)1}}])$$

$$\text{and } (\psi_{N_{n+1}}, \varphi_{N_{n+1}}) = (\psi_{N_n}[\delta_{a_{n1}}], (\varphi_{N_n} \wedge \phi_{a_{n1}})[\lambda_{a_{n1}}])$$

Let a_{ij} be an action of N_i. While the state e of the process P satisfies the condition $\psi N_i(e) \wedge \psi_{a_{ij}}(e) \wedge \varphi N_i(e)$, the process P may either execute

the action a_{ij} provided that $\phi_{a_{ij}}(e)$ is true, or let the time passes while the state of the process satisfies φ_{N_i}. When P executes a_{ij} from a state e, it moves to a new state $e' = (e[\delta_{a_{ij}}], e[\lambda_{a_{ij}}])$.

3.2. Timed Automaton of a Scenario

According to the expression of a context constraints of a primary vertex, the execution of an action depends on the context created by the preceding actions in the scenario. By merging scenario action and the context constraints of its primary vertex, we obtain a self contained rule-action which is independent from the execution context. A rule-action contains all the information about the states of the process and its condition of activity before and after the execution of the rule-action. Let $R(sc)$ designate the set of rule-actions resulting from the scenario sc. Given an action a_{ij} of the vertex N_i, the resulting rule-action is a tuple $r = (\psi_r, \varphi_r, lab_r, \phi_r, \delta_r, \lambda_r, \varphi'_r)$ where:

- $\psi_r = \psi_{N_i} \wedge \psi_{a_{ij}}$
- $\phi_r = \phi_{a_{ij}}$
- $\varphi_r = \varphi_{N_i}$
- $\delta_r = \delta_{a_{ij}}$
- $lab_r = lab_{a_{ij}}$
- $\lambda_r = \lambda_{a_{ij}}$
- For φ'_r, two cases are to be distinguished:

$$\varphi'_r = \begin{cases} \varphi_{N_{i+1}} & \text{if } 1 \le i \le n \text{ and } j = 1 \\ (\varphi_{N_i} \wedge \phi_{a_{ij}})[\lambda_{aij}] & \text{if } 1 \le i \le n \text{ and } 2 \le j \le k_i \end{cases}$$

To define the semantics of a scenario sc and its set of rule-actions $R(sc)$ we will associate to the scenario sc a labeled transition system Σ_{sc} which we refer to by the labeled transition system of the scenario sc, and is defined as follows:

Definition 6. *Each rule-action $r \in R(sc)$, implies that Σ_{sc} will contain the two following sets of transitions:*

- *immediate transitions $e \xrightarrow{lab_r} e'$ such that $e = (\omega, \theta)$ and $e' = (\omega[\delta_r], \theta[\lambda_r])$ provided that $\psi_r(\omega) \wedge \varphi_r(\theta) \wedge \phi_r(\theta) \wedge \varphi'_r(\theta[\lambda_r])$ is true*
- *elapsing time transitions $e \xrightarrow{d} e'$ such that $e = (\omega, \theta)$ and $e' = (\omega, \theta + d)$ provided that $\psi_r(\omega) \wedge (\forall 0 \le d' \le d \,.\, \varphi_r(\theta + d'))$ is true*

Definition 7. *Let Σ_A be the labeled transition system of the TA A. An observational timed trace is a timed trace in Σ_A from which all invisible labels ε are removed. $OTT(e)$ denotes the set of observational timed traces that start from the state e of Σ_A.*

Definition 8. *Let Σ_A and $\Sigma_{A'}$ be the labeled transition system of the TA A and A′ respectively. We assume that A and A′ have the same*

observational labels. Σ_A *and* $\Sigma_{A'}$ *are observational timed trace equivalent iff for each state* e *in* Σ_A, *there exists a state* e' *in* $\Sigma_{A'}$ *such that* $OTT(e) = OTT(e')$ *and for each state* e' *in* $\Sigma_{A'}$, *there exists a state* e *in* Σ_A *such that* $OTT(e) = OTT(e')$.

Theorem 1. *Given a scenario* $sc \in S$, *there exists a TA called TA of the scenario sc, noted* A_{sc}, *such that its labeled transition system* $\Sigma_{A_{sc}}$ *is observational timed trace equivalent to* Σ_{sc}.

The proof of this theorem will be given in the next section. A location of the TA A_{sc} is characterized by a pair (ω, φ) where φ represents the invariant of the location (ω, φ). Each location (ω, φ) includes all of the states (ω, θ) such that θ satisfies φ. The rule-actions of a scenario are not necessarily pairwise-disjoint (i.e for a given scenario *sc,* there may exist two rule-actions $r_1, r_2 \in \mathcal{R}(sc), \omega$ and θ such that $\psi_{r_1}(\omega) \wedge \psi_{r_2}(\omega) = True$ and $\varphi_{r_1}(\theta) \wedge \varphi_{r_2}(\theta) = True$). Let us consider the set composed of locations (ω, φ_r) and $(\omega[\delta_r], \varphi'_r)$ for all $r \in R(sc)$. Each element of this set must be split into new elements so that in the resulting set $L_{A_{sc}}$ no two elements will have a common state. The set $L_{A_{sc}}$ is the set of locations for the TA of the scenario. Let us now define its set of transitions $T_{A_{sc}}$ which is composed of two types of transitions:

- transitions resulting from a rule-action $r \in R(sc)$ like $((\omega, \varphi), lab_r, \phi_r, \lambda_r, (\omega[\delta_r], \varphi'))$ provided that:
  - $(\omega, \varphi) \in L_{A_{sc}}$, $(\omega[\delta_r], \varphi') \in L_{A_{sc}}$ such that $\psi_r(\omega)$ is true, and
  - $\varphi \wedge \varphi_r = \varphi$ and $\varphi' \wedge \varphi'_r = \varphi'$
- transitions labeled by ε which we considere as the invisible label or the empty word. These transitions guarantee the time progress.

4. CONSTRUCTION OF THE TIMED AUTOMATON OF A SCENARIO

Given a scenario *sc,* the construction of its TA A_{sc} takes two phases. First, the locations resulting from rule-actions of $R(sc)$ are split to produce $L_{A_{sc}}$, and then transitions are added according to the rule-actions of $R(sc)$ to build the final $T_{A_{sc}}$.

4.1. Phase 1: Construction of the Locations Set of A_{sc}

To split the locations resulting from $R(sc)$, we'll use an efficient algorithm based on the generic algorithm proposed by Bouajjani and al. [BFH+92] to obtain a minimal split. The generic algorithm generates a minimal transition system directly from an implicit description (like a

program). This algorithm is useful in our context since our rule-actions are implicit description of the labeled transition system of a scenario. Let us briefly describe this algorithm and adapt it in order to construct the locations set $L_{A_{sc}}$.

Let $(Q, Q_o, \rightarrow)$ be a labeled transition system. For a partition ρ of Q and a state $q \in Q$, $post_\rho(q)$ denotes the set of ρ elements that are immediately reachable from q. The function $post_\rho$ is extended to the subsets of Q by $post_\rho(C) = \bigcup_{q \in C} post_\rho(q)$ where $C \subset Q$.

Given a class $C \in \rho$, C is said to be stable with respect to the partition ρ of Q iff $\forall C' \in \rho$:

$$((\exists q_1 \in C, \exists q_1' \in C' \ . \ q_1 \xrightarrow{a} q_1') \text{ implies } (\forall q_2 \in C \ \exists q_2' \in C' \ . \ q_2 \xrightarrow{a} q_2')$$

The minimization algorithm uses $split(C, \rho)$ operator which splits the class C into stable classes with respect to ρ such that:

$$split(C,\rho) = \{C_1, \ldots, C_k \mid (\forall 1 \leq i,j \leq k \ . \ C_i \cap C_j = \emptyset) \ and \ (C_1 \cup \cdots \cup C_k = C) \ and \ (\forall i \ . \ C_i \ stable) \ and \ (\forall i,j \ . \ C_i \cup C_j non \ stable)\}$$

The minimization algorithm (figure 2)progressively refines the partition ρ starting from the initial partition ρ_o. *Acc* denotes the set of reachable states from the initial states. *Stb* contains the stable classes with respect to ρ. The algorithm stops when all of the reachable classes are stable with respect to ρ.

```
procedure Minimization
  ρ := ρ_o ; Acc := {C ∈ ρ | Q_0 ∩ C ≠ ∅}; Stb := ∅;
  while Acc ≠ Stb do
    choose C ∈ Acc − Stb
    D := split(C, ρ)
    if D = {C} then
      Stb := Stb ∪ {C} ; Acc := Acc ∪ post_ρ(C)
    else
      Acc := (Acc − {C}) ∪ {C' ∈ D | C' ∩ Q_o ≠ ∅}
      Stb := Stb − {C' ∈ Stb | C ∈ post_ρ(C')}
      ρ := (ρ − {C}) ∪ D
```

Figure 2. Minimization algorithm

To apply the minimization algorithm let us consider the TA A defined by:

- $H_A = H$
- $M_A = \{lab_r \ . \ r \in \mathcal{R}(sc)\}$
- $L_A = \{(\omega, \varphi_r) \in \Omega(V) \times \Phi(H) \mid r \in \mathcal{R}(sc) \text{ and } \psi_r(\omega)\}$
  $\cup \{(\omega[\delta_r], \varphi_r') \in \Omega(V) \times \Phi(H) \mid r \in \mathcal{R}(sc) \text{ and } \psi_r(\omega)\}$

- $L_A^o = \{(\omega, \varphi_{N_1}) \in \Omega(V) \times \Phi(H) \mid \psi_{N_1}(\omega)\}$
- $T_A = \{((\omega, \varphi_r), \phi_r, lab_r, \lambda_r, (\omega[\delta_r], \varphi'_r)) \mid r \in \mathcal{R}(sc) \text{ and } \psi_r(\omega)\}$
- $\forall (\omega, \varphi) \in L_A,\ Inv(\omega, \varphi) = \varphi$

The locations of L_A are the resulting locations from the rule-actions of $R(sc)$. The TA A contains all the immediate transitions of A_{sc} but not all transitions modeling time elapsing. The TA A is not minimal because the rule-actions of $R(sc)$ are not necessarily pairwise-disjoint. The locations of L_A will be split by the minimization algorithm in order to build a minimal TA. A minimal TA means that for a given pair (ω, θ), if there exists a location (ω, φ) in this TA such that $\psi(\omega) \wedge \varphi(\theta)$ is true, then this location is unique.

The minimization algorithm is applied to the labeled transition system of the TA A and with the following initial ρ_o:

$$\rho_o = \{(\psi_r, \varphi_r).r \in \mathcal{R}(sc)\} \cup \{(\psi_r[\delta_r], \varphi'_r).r \in \mathcal{R}(sc)\} \qquad (1)$$

Elements of ρ_o are classes. $[\varphi]$ and $[\psi]$ denote the set of clock interpretations and variable interpretations respectively which satisfy φ and ψ respectively. For a class $C \in \rho$ denoted (ψ_C, φ_C), C is the set $[\psi_C] \times [\varphi_C]$.

For a constraint $\varphi \in \Phi(H)$, $[\varphi]$ is a convex polyhedron of $\mathbb{R}^{|H|}$. Let us define the operators '$\cap$' and '$-$' on classes of ρ:

$$C \cap C' = (\psi_C \wedge \psi_{C'}, \varphi_C \wedge \varphi_{C'}) \qquad C - C' = (\psi_C \wedge \neg\psi_{C'}, \varphi_C \wedge \neg\varphi_{C'})$$

The intersection of convex polyhedrons is convex but their union is not necessarily convex. So $[\varphi_C \wedge \neg\varphi_{C'}]$ may not be convex. For a class $C = (\psi_C, \varphi_C)$, $[\varphi_C]$ must be convex because φ_C will be the invariant of a location of the TA under construction. Thus, let us define the operator $convex(\varphi)$ which transforms $[\varphi]$ into a partition of convex polyhedrons. The *convex* operator is extended to classes by the expression $convex(\psi, \varphi) \stackrel{def}{=} \{(\psi, \varphi') | \varphi' \in convex(\varphi)\}$.

The algorithm of $split(\rho, C)$ (figure 3) uses the function $pre_r(C)$ denoting the class of elements from which C can be reached by transitions resulting from r.

The minimization algorithm assumes that ρ_o is a partition but in our case, initial ρ_o defined by the formula (1), is not necessarily a partition since the rule-actions of $R(sc)$ may be not disjoint. The classes of ρ_o are reachable because they are resulting from $R(sc)$, and therefore there exists an iteration in the minimization procedure where ρ becomes a partition and remains so until the end.

In the remainder of the paper, ρ designates the resulting partition from the minimization procedure (ie all of ρ's classes are stable). The

```
function split(ρ, C)
  Z := {C}
  for each C' ∈ ρ
    for each r ∈ R
      P := ∅
      U := pre_r(C')
      for each X ∈ Z do
        W := U ∩ X
        if (W = ∅ or W = X) then P = P ∪ {X}
        else P := P ∪ {W} ∪ convex(X − W)
      Z := P
  return (Z)
```

Figure 3. Split function

set of locations $L_{A_{sc}}$ may now be explicitly described by the following expression:

$$L_{A_{sc}} = \{(\omega, \varphi) \in \Omega(V) \times \Phi(H) \mid \exists C \in \rho \,.\, \psi_C(\omega) \; and \; \varphi = \varphi_C\}$$

4.2. Phase 2: Construction of the Transitions Set of A_{sc}

As mentioned earlier, the set of transitions $T_{A_{sc}}$ contains two types of transitions:

- transitions deducted from $R(sc)$ like $((\omega, \varphi), lab_r, \phi_r, \lambda_r, (\omega[\delta_r], \varphi'))$ such that $r \in \mathcal{R}(sc)$, $\psi_r(\omega)$, $[\varphi] \subset [\varphi_r]$ and $[\varphi'] \subset [\varphi'_r]$ and
- ε -transitions modeling time elapsing and labeled by the invisible action ε. Timing constraints of ε-transitions will be defined further in the paper.

For example, given two locations (ω, $h < 3$) and (ω, $h \geq 3$) of $L_{A_{sc}}$ where h is a clock of H. To model continuous time elapsing, we need to add the ε-transition t = ((ω, $h < 3$), ε, $h = 3$, ∅, (ω, $h \geq 3$)). Since the invariant of the location (ω, $h < 3$) is ($h < 3$), the ε-transition t will never be fired as the constraint (($h < 3$) ∧ ($h = 3$)) can't never be satisfied. We also know that as soon as time leaves the zone [$h < 3$], it enters to the zone [$h \geq 3$]. To avoid this topological problem, the invariant of the location (ω, $h < 3$) is relaxed to (ω, $h \leq 3$). For preserving the same transition system, the previous constraint $h < 3$ is added to the guarding constraint of the ingoing and the outgoing transitions of the relaxed location. The complete description of adding ε-transitions and

function *add_ε-transition*
  let $T_{A_{sc}} := \{((\omega,\varphi), lab_r, \phi_r, \lambda_r, (\omega[\delta_r], \varphi')) \mid$
    $r \in \mathcal{R}(sc),\ \psi_r(\omega),\ [\varphi] \subset [\varphi_r]$ **and** $[\varphi'] \subset [\varphi'_r]\}$
  for each $(\omega,\varphi_1),(\omega,\varphi_2) \in L_{A_{sc}}$ **such that** $\varphi_1 \prec \varphi_2$
    if $\overline{\varphi_1} = \varphi_1$
    then $s := (\omega,\varphi_2)$; $t_\varepsilon := ((\omega,\varphi_1), \varepsilon, \overline{\varphi_1 \wedge \varphi_2}, \emptyset, (\omega,\overline{\varphi_2}))$
    else $s := (\omega,\varphi_1)$; $t_\varepsilon := ((\omega,\overline{\varphi_1}), \varepsilon, \overline{\varphi_1 \wedge \varphi_2}, \emptyset, (\omega,\varphi_1))$
    $T_{A_{sc}} := Relax\_Inv(T_{A_{sc}}, s)$; $T_{A_{sc}} := T_{A_{sc}} \cup \{t_\varepsilon\}$
  return $T_{A_{sc}}$

function $Relax\_Inv(\mathcal{T}, (\omega,\varphi))$
  for each $t \in \mathcal{T}$ **such that** $t = ((\omega,\varphi), lab, \phi, \lambda, s')$ **and** $lab \neq \varepsilon$
    $\mathcal{T} := \mathcal{T} - \{t\} \cup \{((\omega,\overline{\varphi}), lab, \phi \wedge \varphi, \lambda, s')\}$
  for each $t \in \mathcal{T}$ **such that** $t = (s, lab, \phi, \lambda, (\omega,\varphi))$ **and** $lab \neq \varepsilon$
    $\mathcal{T} := \mathcal{T} - \{t\} \cup \{(s, lab, \phi \wedge \varphi[\lambda], \lambda, (\omega,\overline{\varphi}))\}$
  $L_{A_{sc}} := L_{A_{sc}} - \{(\omega,\varphi)\} \cup \{(\omega,\overline{\varphi})\}$
  return $\mathcal{T}$

Figure 4. Adding ε-transitions and locations relaxing functions

relaxing locations is given by functions *add_ε-transition* and *Relax_Inv* (figure 4). These algorithms use the following definitions:

Definition 9. *Given* $\varphi \in \Phi(H)$, *the relaxed constraint of* φ *denoted* $\overline{\varphi}$, *is the clock constraint in which the relation* $<$ *and* $>$ *are respectively replaced by* $\leq$ *and* $\geq$ *in the expression of* φ.

Definition 10. *We write* $\varphi_1 \prec \varphi_2$ *iff* $[\varphi_1] \cap [\varphi_2] = \emptyset$ *and* $([\varphi_1] \cap [\overline{\varphi_2}] \neq \emptyset$ *or* $[\overline{\varphi_1}] \cap [\varphi_2] \neq \emptyset)$

The resulting sets $T_{A_{sc}}$ and $L_{A_{sc}}$ after the *add_ε-transition* function call represent respectively the set of transitions and the set of locations of the TA A_{sc}. The set of labels is $M_{A_{sc}} = M_A \cup \{\varepsilon\}$.

4.3. Observational Timed Trace Equivalence

Σ_{sc} and $\Sigma_{A_{sc}}$ are the labeled transition systems of the scenario *sc* and its TA A_{sc} respectively. A_{sc} may contain some ε-transitions while Σ_{sc} does not. $\Sigma_{A_{sc}}$ and Σ_{sc} must be equivalent according to theorem 1. The objective of this equivalence is to allow us to ignore ε labels in the timed traces of A_{sc} to be able to compare them with the timed traces of Σ_{sc} since ε-transitions are not observable by the environment. Let us first define what we mean by runs and timed traces for Σ_{sc}. If we consider that Σ_{sc} is a labeled transition system of a fictive TA, the runs and timed traces of Σ_{sc} may be defined exactly as we did previously in definitions

3 and 4 for a TA. The next proposition defines an equivalence between $\sum_{sc}$ and $\Sigma_{A_{sc}}$:

Proposition 1. *$\sum_{A_{sc}}$ and $\sum_{sc}$ are observational timed trace equivalent.*

Proof. (sketch) Given a state e of Σ_{sc} there exists $r \in R(sc)$, ω and θ such that $e = (\omega,\theta)$ and at least one of the two following cases is true : 1) $\psi_r(\omega)$ and $\varphi_r(\theta)$, or 2) $\omega = \omega'[\delta_r]$ and $\psi_r(\omega')$ and $\varphi'_r(\theta)$. The states of $\sum_{A_{sc}}$ are in the form of $((\omega, \varphi), \theta)$ since (ω, φ) may represent a location in A_{sc}. There exists a state $e' = ((\omega,\varphi),\theta)$ of $\Sigma_{A_{sc}}$ resulting from the split of either the class (ψ_r, φ_r) or $(\psi_r[\delta_r], \varphi'_r)$. One can then proof $OTT(e) = OTT(e')$. □

According to the previous proposition, $\sum_{sc}$ et $\sum_{A_{sc}}$ are equivalent from the point of view of observational timed trace equivalence. The construction of A_{sc} and the proposition 1 provide a proof of theorem 1.

The resulting partition ρ from the construction of A_{sc} allows to reduce the combinatorial explosion of the locations in A_{sc}. ρ abstracts A_{sc} into a compact TA A_{sc}/ρ such that:

- $L_{A_{sc}/\rho} = \rho$
- $L^o_{A_{sc}/\rho} = \{C \in \rho | C \cap L^o_A \neq \emptyset\}$
- $T_{A_{sc}/\rho} = \{(C, m, \phi, \lambda, C') |\ (s, m, \phi, \lambda, s') \in T_{A_{sc}},\ s \in C \text{ and } s' \in C'\}$

A location of the TA A_{sc}/ρ groups together all the location in A_{sc} which are imperceptible if we do not take into account the values of the discrete variable. Such representation is more suitable as an input to verification tools since it is more compact than the initial TA A_{sc}.

5. APPLICATION: TELEPHONE SWITCH CONTROLLER

This section illustrates the construction of a TA from a simple scenario. The process P models a telephone switch controller. Discrete variables set V and clocks set H are those described in figure 1. In this example, we will construct the TA of the following call establishment scenario:

> "When the user A is idle, if the controller receives the $pickup(A)$ message, it sends the tone to user A. If user A calls user B before 30 time units then the controller sends the ring to B provided that he's idle. But if user A does nothing during 30 time units, the controller sends him the $busy\_tone(A)$ message. While the user B terminal is ringing, if user B picks up before 60 time units, the controller establishes the call, else the later is canceled and the controller sends $busy\_tone(A)$".

The representation of the tree of this scenario was removed from this document, for lack of space, but it can be restored from the first column of table 1 according to definition 5. Table 1 describes the set of rule-actions resulting from the call establishment scenario. The construction of the TA scenario uses as input the rule-actions of table 1 and its initial distribution classes ρ_o. The abstraction of the scenario TA defined by the resulting partition ρ is drawn in figure 5.

a_{ij}	Rule-action r							ρ_o
	φ_r	Lab_r	ψ_r	ϕ_r	δ_r	λ_r	φ'_r	
a_{11}	$True$	$pickup(A)$	$A\_stat{=}IDLE$ $\wedge A\_sig{=}NONE$	$True$	$A\_stat{:=}BUSY$	$\{h\}$	$h{=}0$	C_1 C_2
a_{21}	$h{=}0$	$send\_tone(A)$	$A\_stat{=}BUSY$ $\wedge A\_sig{=}NONE$	$h{=}0$	$A\_sig{:=}TONE$	$\{h\}$	$h{\leq}30$	C_2 C_3
a_{31}	$h{\leq}30$	$dialing(B)$	$A\_stat{=}BUSY$ $\wedge A\_sig{=}TONE$	$h{<}30$	$A\_sig{:=}$ $DIALING(B)$	$\{h\}$	$h{=}0$	C_3 C_4
a_{32}	$h{\leq}30$	$busy\_tone(A)$	$A\_stat{=}BUSY$ $\wedge A\_sig{=}TONE$	$h{=}30$	$A\_sig{:=}$ $BUSY\_TONE$	$\{h\}$	$True$	C_3 C_8
a_{41}	$h{=}0$	$ring(A,B)$	$A\_sig{=}$ $DIALING(B)$ $\wedge A\_stat{=}BUSY$ $\wedge B\_stat{=}IDLE$ $\wedge B\_sig{=}NONE$	$h{=}0$	$A\_sig{:=}$ $ECHO\_RING(B)$, $B\_sig{:=}RING(A)$, $B\_stat{:=}BUSY$	$\{h\}$	$h{\leq}60$	C_5 C_6
a_{51}	$h{\leq}60$	$pickup(B)$	$A\_sig{=}$ $ECHO\_RING(B)$ $\wedge B\_sig{=}RING(A)$ $\wedge A\_stat{=}BUSY$ $\wedge B\_stat{=}BUSY$	$h{<}60$	$A\_sig{:=}$ $TALKING$, $B\_sig{:=}$ $TALKING$	$\emptyset$	$True$	C_6 C_7
a_{52}	$h{\leq}60$	$busy\_tone(A)$	$A\_sig{=}$ $ECHO\_RING(B)$ $\wedge B\_sig{=}RING(A)$ $\wedge A\_stat{=}BUSY$ $\wedge B\_stat{=}BUSY$	$h{=}60$	$A\_sig{:=}$ $BUSY\_TONE$, $B\_sig{:=}NONE$, $B\_stat{:=}IDLE$	$\{h\}$	$True$	C_6 C_9

Table 1. Description of the rule-action set $R(sc)$ for the call establishment scenario. Each row in the table, contains the original scenario action, its corresponding rule-action and its two resulting classes in the initial ρ_o

6. METHOD OF SCENARIOS INTEGRATION

The objective of the integration of scenarios is to merge many of them into a single TA. The construction of the scenario TA algorithm may be applied on any rule-actions set. Let A_R be the TA synthesized from the set of rule-actions R. Based on this notation, the TA of a scenario sc may also be denoted $A_{R(sc)}$. Particularly, for the scenarios sc_1 and sc_2, their respective TA A_{sc_1} and A_{sc_2} are merged into one TA by the operator $\oplus$ defined as follows:

$$A_{sc_1} \oplus A_{sc_2} = A_{\mathcal{R}(sc_1)} \oplus A_{\mathcal{R}(sc_2)} \stackrel{def}{=} A_{\mathcal{R}(sc_1) \cup \mathcal{R}(sc_2)}$$

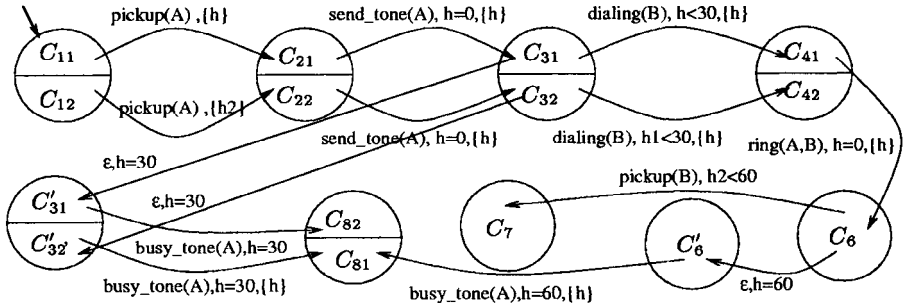

Figure 5. Abstract Timed automata of the call establishment scenario. The classes, for example, C_{11} and C_{12} classes result from the split of the class C_1 and so on.

Proposition 2. *For each state e in* Σ_{sc} $(sc \in \{sc_1, sc_2\})$ *there exists a state e′ in* $\Sigma_{A_{sc_1} \oplus A_{sc_2}}$ *such that* $OTT(e) \subset OTT(e')$. *The reverse is false because of the possible overlapping of the scenarios.*

The proposition 2 allows to construct one TA A which models the process $P = (S, H, V, Act)$ and to integrate all of its scenarios. The TA A results from the compilation of the rule-actions set $\bigcup_{sc \in S} R(sc)$. Given $S = \{sc_1, \ldots, sc_n\}$, the following formula describes the above stated compilation:

$$\mathcal{A} = A_{sc_1} \oplus A_{sc_2} \oplus \cdots \oplus A_{sc_n} = A_{\mathcal{R}(sc_1) \cup \mathcal{R}(sc_2) \cup \cdots \cup \mathcal{R}(sc_n)} \qquad (2)$$

According to proposition 2, A includes all the behaviors allowed by each integrated scenario. A also includes some extra behaviors which may result from some overlapping scenarios. It is possible to identify these extra behaviors for an eventual validation process. The salient features of the integration algorithm may be inferred from the formula (2) as follows :

- The specification of a system is incremental. Assuming that the current specification of an existing system results from the compilation of a set of scenarios S, the system extension to support new services consists of adding new scenarios to the current specification. Assume S' is the set of those new scenarios. The whole system prototype is now the TA $A_R \oplus A_{R'}$ where $R = \bigcup_{sc \in S} R(sc)$ and $R' = \bigcup_{sc \in S'} R(sc)$. Each intermediate prototype of the system may be checked for the detection of possible features interaction.

- The operator $\oplus$ is commutative and associative as the union $\cup$. So, the order in which scenarios are added to construct the specification doesn't matter.

- Previous compilation results are reused when a new scenario sc is added. In the formula $A_{\bigcup_{sc \in S} R(sc)} \oplus A_{sc'}$, the first operand is not recomputed but just reused to get the new prototype of the system.

7. CONCLUSION

In this paper, our main contribution is the development of an algorithm for compiling many real-time scenarios into a single timed automaton which represents an executable prototype of the system. This algorithm is insensitive to the order in which scenarios are integrated.

First we have proposed a syntax and a formal semantics for scenarios. Then we defined a flat format of a scenario as a set of self contained rule-actions. The compilation algorithm synthesizes a timed automaton from a set of rule-actions. The set of rule-actions may come from more than one scenarios and therefore the resulting timed automaton represents the integration of these scenarios. The compact representation format of the timed automaton produced by our integration algorithm is compatible to use with model checking tools like KRONOS and UPPAAL.

As future work, we wish to further define operators for explicit integration of the scenarios. These operators are used as directives of compilation.

REFERENCES

[ACD93] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation,* 104(1):2–34, may 1993.

[AD94] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science,* 126: 183–235, 1994.

[BFH+92] A. Bouajjani, J.-C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Science of Computer Programming,* 18, 1992.

[Gli95] M. Glinz. An Integrated Formal Model of Scenarios Based on Statecharts. In *Software Engineering - ESEC'95,* pages 254–271. Springler LNCS 989, 1995.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation,* 111(2), jun 1994.

[HSG+94] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima, and C. Chen. Formal approach to scenario analysis. *IEEE Software,* 11:33–41, march 1994.

[SDLV00] A. Salah, R. Dssouli, G. Lapalme, and D. Vincent. Vers un environnement de création de services fondé sur les scénarios enrichis. In *Proceedings of CFIP,* 2000.