Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT

Information and Software Technology 45 (2003) 715–725

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Implicit integration of scenarios into a reduced timed automaton ☆

A. Salah[a,*], R. Dssouli[b], G. Lapalme[c]

[a] *Département d'Informatique, Université du Québec à Montréal, C.P. 8888, succ. centre ville Montréal, Montréal, Qué., Canada H3C 3P8*
[b] *Electrical and Computer Engineering (ECE), Concordia University, 1455 de Maisonneuve Blvd. W., Montreal, Que., Canada H3G 1M8*
[c] *Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, C.P. 6128, succ. centre ville Montréal, Montréal, Qué., Canada H3C 3J7*

Abstract

We aim at synthesizing an executable specification for a real-time reactive system by integrating real-time scenarios into a reduced timed automaton (TA). A scenario is a part of the specification of a system behavior. The integration of scenarios into a single TA is based on its formal semantics. The TA, which results from the integration of a set of scenarios, is independent of the order in which the scenarios are added to. We also present an algorithm to reduce such resulting TA in order to prevent combinatorial explosion.

Keywords: Design and synthesis; Scenarios; Timed automata

1. Introduction

Requirements engineering is the first step in the software engineering life cycle. In the case of real-time reactive systems, requirements engineering consists of eliciting, discovering, understanding and representing the users requirements. Documents that arise from this step have informal representations and may induce a great number of software failures [1]. An important part of these failures come from undetected inconsistencies and incompleteness of user requirements. Errors are also introduced during the conversion of the requirements into a formal specification. Such errors are the most difficult to detect. Moreover, passing from informal requirements to formal specification is hard task.

This paper aims at facilitating the formal specification of a real-time reactive system. Our approach is based on a simple formal specification format called *scenario*. The designer converts the informal representations into a set of scenarios. Each scenario represents a part of the specification. The use of scenarios hides the formal method complexity and is similar to the divide and conquer strategy used in problem analysis. We propose an automatic method to integrate scenarios into a single *Timed Automaton* (TA) [2,3].

The specification of a system is an iterative process in which new scenarios can be added to the current prototype. Consequently, the resulting specification should not depend on the order in which scenarios are added.

2. Related work

A scenario is a natural means to specify interactions of a reactive system with its environment. The use of scenarios reduces the complexity of systems' specification since a scenario allows the description of partial behaviors of the system.

Most approaches based on scenarios [4–8] 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 such as their input formalism, their method of scenarios-integration and their target formalism which should have a formal semantics.

The method of integration is the most important criterion and consists in merging scenarios into one global specification expressed in the target formal model.

To the best of our knowledge, there are two methods for the integration of scenarios which we will refer to, respectively, by *implicit integration* method and *explicit integration* method. In the case of implicit integration, only a set of scenarios is provided and the main challenge is how to characterize the steps of a scenario so that occurrences of

☆ Partially supported by France Telecom.
* Corresponding author. Tel.: +1-514-987-3000; fax: +1-514-987-8477.
E-mail address: salah.aziz@uqam.ca (A. Salah).

0950-5849/03/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0950-5849(03)00068-5

the same steps may be identified in other scenarios. Some implicit approaches use a manual labeling while others are based on regular grammar [4] in order to characterize scenarios steps. In the case of explicit integration methods, designers also specify the order of the executions of scenarios using defined *operators*. Consequently, scenarios are considered as Lego blocks and those operators specify how to link them to each other. Such approaches may assume that scenarios are disjoint [5].

We propose an automatic method for implicit integration of scenarios in the case of real-time reactive systems. Such systems interact with the environment under strict timing constraints. We use variables to characterize scenarios' steps. We consider two kinds of variables, discrete variables describing system properties and continuous real variables (clocks) measuring time. Such features allow the description of real-time scenarios in which the behavior of the system depends on time.

A scenario describes possible sequences of interactions between the system and its environment. We are limited to sequential systems in which only one interaction is allowed to be performed at a time. In a scenario, we consider two basic types of interactions namely the reception and the transmission of a message, which are both viewed as observable actions. Our approach takes as input a set of scenarios and synthesizes an output in the form of a TA.

The rest of the paper is organized as follows. Section 3 presents a model of timed automata and conformance relations. Section 4 describes both static aspect and behavior aspect of the specification of a system. The static aspect regards the syntax of variables and labels of actions while behavior aspect describes an internal and basic form of the system behavior representation called rule-action. Section 5 transforms rule-actions into an equivalent TA.

Section 6 defines the form of scenarios and shows how to transform a scenario into its canonical form which is a set of rule-actions. Section 7 treats the integration of scenarios and defines its characteristic conformance relation. Section 8 describes an algorithm to reduce the TA of a set of rule-actions. Section 9 illustrates our approach of the integration of scenarios into a reduced TA.

3. Timed automata

A TA [2,3] is a labeled transition system (LTS) extended with a finite number of real variables called clocks and is based on a dense model of time. Clocks values increase at the same rate.

Let H be the set of clocks. $\Phi(H)$ designates the set of clock-constraints φ defined by the grammar $\varphi ::= h_1 \# c | h_1 - h_2 \# c | \varphi \wedge \varphi | \varphi \vee \varphi | True$, where h_1, h_2 are clocks in and H and $\# \in \{ \leq, <, =, \geq, > \}$ and c a constant in $\mathbb{N}$.

A clock-interpretation θ is a vector composed of $|H|$ non-negative real numbers. $\Theta(H)$ denotes the set of clock-interpretations. For each $\theta \in \Theta(H)$ and a clock-constraint $\varphi \in \Phi(H)$, $\varphi(\theta)$ is a Boolean value describing whether θ satisfies φ or not. Given a non-negative real number d, $\theta + d$ denotes the clock-interpretation in which d is added to each component of the vector θ.

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

Definition 1. A TA is a tuple $A = (L_A, L_A^0, M_A, T_A, H_A, Inv_A)$, where

- L_A is a finite set of locations
- $L_A^0 \subset L_A$ 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 timed transitions
- Inv_A 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 LTS is a tuple $\Sigma = (Q, Q_0, \rightarrow)$ where Q is a set of states, Q_0 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 $(q, a, q') \in \rightarrow$.

The semantics of a TA A is defined by its LTS Σ_A as follows. 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 positive real numbers $d' \leq d$, $\theta + d'$ satisfies $Inv_A(s)$, and
- immediate transitions in the form of $(s, \theta) \xrightarrow{m} (s', \theta[\lambda])$ such that $(s, m, \varphi, \lambda, s') \in T_A$, θ satisfies $\varphi \wedge Inv_A(s)$ and $\theta[\lambda]$ satisfies $Inv_A(s')$.

Comparison between two LTS is based on conformance relations. In this work, we use two well-known conformance relations namely bisimulation and simulation.

Definition 3. Let $\Sigma_i = (Q_i, Q_{0i}, \rightarrow_i)$ be an LTS for $i \in \{1, 2\}$. Let $\mathscr{B} \subset Q_1 \times Q_2$ be a binary relation. We write $q_1 \mathscr{B} q_2$ iff $(q_1, q_2) \in \mathscr{B}$. $\mathscr{B}$ is a bisimulation between Σ_1 and Σ_2 iff:

(ia) $\forall q_1 \in Q_1, \exists q_2 \in Q_2 \cdot q_1 \mathscr{B} q_2$
(ib) $\forall q_2 \in Q_2, \exists q_1 \in Q_1 \cdot q_1 \mathscr{B} q_2$
(iia) $\forall q_1 \in Q_1 \cdot (q_1 \xrightarrow{a}_1 q_1'$ and $q_1 \mathscr{B} q_2)$ implies $(\exists q_2' \in Q_2 \cdot q_2 \xrightarrow{a}_2 q_2'$ and $q_1' \mathscr{B} q_2')$
(iib) $\forall q_2 \in Q_2 \cdot (q_2 \xrightarrow{a}_2 q_2'$ and $q_1 \mathscr{B} q_2)$ implies $(\exists q_1' \in Q_1 \cdot q_1 \xrightarrow{a}_1 q_1'$ and $q_1' \mathscr{B} q_2')$

A bisimulation means that the two LTS have the same behavior in two bisimilar states and the corresponding arrival two states are bisimilar too. If a binary relation satisfies only the conditions (ia) and (ib), such a relation is called a simulation between Σ_1 and Σ_2. In a such case, any behavior in an state of Σ_1 is a behavior of a similar state in Σ_2. We can say that Σ_2 can simulate Σ_1.

A relation $\mathscr{B}_2$ refines a relation $\mathscr{B}_1$ iff $(q_1 \mathscr{B}_2 q_2)$ implies $(q_1 \mathscr{B}_1 q_2)$.

A bisimulation is a relation between two LTS. However, it is possible to have a bisimulation between an LTS and itself. Such bisimulation is called an auto-bisimulation and is used to reduce this LTS.

4. System specification

This section is dedicated to some definitions and its related notations. A formal specification of a system is usually composed of both a static and a dynamic descriptions. The static description of a system defines elements of the language to be used in the dynamic description, which represents the behavior of the system. In the remaining part of this paper, *the system* designates the real-time reactive system that is being specified.

4.1. Elements of the static description of the system

Elements of the static description of the system includes clocks, events and discrete variables. Clocks respect the syntax defined for the clocks of a TA and H represents the set of clocks of the system.

An event represents the execution of an action. We associate to each action a label which is observed by the environment when this action is executed and we designate by Ev the set of such labels.

4.2. Discrete variables, variable-constraints and variable-assignments

Discrete variables are used to model non-temporal properties of a system. When a system interacts with its environment, its state changes. Therefore, the discrete variables of the system are updated to capture the new state of the system.

Let $V = \{v_1, \ldots, v_{|V|}\}$ denote the set of a system discrete variables such that $|V|$ is the number of variables in V. $Dom(v_i)$ is a finite set of constants which represents the domain of values of the variable $v_i \in V$. Let $\Psi(V)$ denote the set of variable-constraints ψ defined by the grammar $\psi ::= v \# c | \psi \wedge \psi | \neg \psi | True$, where $v \in V$, $c \in Dom(v)$ and $\#$ is a binary relation of $Dom(v) \times Dom(v)$. Let $\Omega(V) = Dom(v_1) \times Dom(v_2) \times \cdots \times Dom(v_{|V|})$ be the set of variable-interpretations. For each variable-interpretation $\omega \in \Omega(V)$, we can write $\omega = (\omega_1, \omega_2, \ldots, \omega_{|V|})$ and $\omega(v_i) = \omega_i$.

$V = \{A\_sta,\ A\_sig,\ B\_sta,\ B\_sig\}$
$Dom(A\_sta) = \{BUSY,\ IDLE\}$
$Dom(A\_sig) = \{NONE,\ TONE,\ BUSY\_TONE,\ DIALING(B),\ ECHO\_RING(B),\ TALKING(B)\}$
$Dom(B\_sta) = \{BUSY\ , IDLE\};$
$Dom(B\_sig) = \{NONE\ , TONE\ , BUSY\_TONE\ , RING(A)\ , TALKING(A)\}$
$H = \{h1,\ h2\}$
$Ev = \{pickup(A),\ pickup(B),\ send\_tone(A),\ busy\_tone(A),\ dialing(B)\}$

Fig. 1. Discrete variables set V and clocks set H for a telephone switch.

The Boolean value $\psi(\omega)$ denotes whether ω satisfies ψ or not. We write $[\psi]$ the set of all ω such that $\psi(\omega)$ is true.

Variable-assignments allow modifications of variables to update the state of a system. For example, an assignment $\delta = \{v_1 := v_1 + 3, v_2 := 1\}$ and a variable-interpretation $\omega \in \Omega(V)$, $\omega[\delta]$ denotes a new variable-interpretation defined by $\omega[\delta](v_1) = \omega(v_1) + 3$, $\omega[\delta](v_2) = 1$ and $\omega[\delta] \times (v) = \omega(v)$ for $v \in V - \{v_1, v_2\}$. There is no limitation on assignment statements expressions but the resulting value of each variable must remain in its domain of values. We write $\Delta(V)$ to denote the set of variable-assignments. Given a variable-constraint $\psi \in \Psi(V)$ and a variable-assignment $\delta \in \Delta(V)$, the constraint $\psi[\delta] \in \Psi(V)$ designates the constraint satisfied by all $\omega[\delta]$ such that the variable-interpretation $\omega \in \Omega(V)$ satisfies ψ. Fig. 1 shows an example of the description of a domain of application in the case of a telephone switch controller system. The description of a domain of application represents the declaration the set of discrete variables V, the set of clocks H and the set of labels Ev.

In the remaining part of this document we will adopt the notations described in Table 1.

4.3. Basic element of a system behavior

We describe the behavior of the system using a self-contained basic element called *rule-action* which includes all of the needed information regarding the observation of an event (an action). A rule-action describes the possible evolution of the system state before and after the execution of its related action.

The behavior of the system depends only on the values of clocks and variables. Consequently, the state of the system is characterized by a pair $e = (\omega, \theta) \in \Omega(V) \times \Theta(H)$ composed of a variable-interpretation ω and a clock-interpretation θ. For simplicity, we adopt the following

Table 1
Table of notations

Notation	Description
$\theta \in \Theta(H)$	Clock-interpretation
$\omega \in \Omega(V)$	Variable-interpretation
$\varphi, \varphi' \in \Phi(H)$	Clock-constraint
$\psi \in \Psi(V)$	Variable-constraint
$\lambda \subset H$	Clocks-assignment
$\delta \in \Delta(V)$	Variable-assignment

writing rules: $\psi(e) = \psi(\omega)$, $\varphi(e) = \varphi(\theta)$, $e[\delta] = \omega[\delta]$ and $e[\lambda] = \theta[\lambda]$. We say, for example, that the system state e satisfies the variable-constraint ψ if the component θ of e satisfies ψ. For $d \geq 0$, $e + d = (\omega, \theta + d)$. Let now define the syntax and the semantics of a rule-action.

Definition 4. A rule-action r is tuple $r = (\psi_r, \varphi_r, l_r, \varphi'_r, \delta_r, \lambda_r, \varphi''_r)$ where:

- $\psi_r \in \Psi(V)$: a variable-constraint that represents the precondition of the rule-action.
- $\psi_r \in \Phi(H)$: a clock-constraint which represents the condition of activity of the state of the system when this state satisfies ψ_r.
- $l_r \in Ev$: a observable label used for synchronization with the environment.
- $\varphi'_r \in \Phi(H)$: a clock-constraint that enables the rule-action.
- $\delta_r \in \Delta(V)$: a variable-assignment describing the update of the state of the system after the execution of the rule-action r.
- $\lambda_r \subset H$: a set of clocks to be reset after the execution of the rule-action.
- $\varphi''_r \in \Phi(H)$: a clock-constraint representing the condition of activity of the state of the system after the execution of the rule rule-action, i.e. when the state of the system satisfies $\psi_r[\delta_r]$.

The semantics of a rule-action r is defined as follows. The system can let time pass while its state e satisfies the predicate $\varphi_r(e) \wedge \psi_r(e)$. The state of the system changes continuously by time elapsing. Moreover, the rule-action r cannot be executed unless the state of the system satisfies the constraint φ'_r as a pre-condition and φ''_r as a post-condition. After the execution of the rule-action r in the state e, the state of the system becomes $e' = (e[\delta_r], e[\lambda_r])$.

The behavior of the system is described by a set of rule-actions. Table 2 represents the behavior of the telephone switch which the description of the domain of application is given in Fig. 1. Section 5, we transform a set of rule-actions into TA.

5. Timed Automaton of a set of rule-actions

In this section, we assume that the specification of a system is given as a set of rule-actions R. Each rule-action in R provides a part of the system specification. Grouping together the rule-actions of R into an LTS, provides a whole single model of the specification of the system which includes all of the possible behaviors that are specified by the rule-actions. We write Σ_R the LTS of the set of rule-actions R.

Definition 5. The set of states of the LTS Σ_R is a subset of $\Omega \times \Theta$. The set of transitions of Σ_R is composed of two types of transitions:

- immediate transitions in the form of $e \xrightarrow{l_r} e'$ such that:
  $e = (\omega, \theta)$ and $e' = (\omega[\delta_r], \theta[\lambda_r])$
  and the predicate $\psi_r(\omega) \wedge \varphi_r(\theta) \wedge \varphi'_r(\theta) \wedge \varphi''_r(\theta[\lambda_r])$ is true
- elapsing time transitions in the form of $e \xrightarrow{d} e'$ such that $e = (\omega, \theta)$, $e' = (\omega, \theta + d)$ and either
  $\psi_r(e) \wedge (\forall 0 \leq d' \leq d \cdot \varphi_r(e + d'))$
  $\psi_r[\delta_r](e) \wedge (\forall 0 \leq d' \leq d \cdot \varphi'_r(e + d'))$

  and the set of transitions of Σ_R satisfies the additivity property which means that if $e \xrightarrow{d} e'$ and $e' \xrightarrow{d'} e''$ are two elapsing time transitions in Σ_R, it implies that $e \xrightarrow{d+d'} e''$ is also an elapsing time transition in Σ_R.

Table 2
A set of rule-actions describing the telephone switch system behavior which may be informally expressed as:

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 is 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)

ψ_r	φ_r	l_r	φ'_r	δ_r	λ_r	φ''_r	(*)
$A\_sta = IDLE, A\_sig = NONE$	True	*pickup*(A)	True	$A\_sta := BUSY$	$\{h2\}$	$h2 = 0$	$[\psi_1]\ [\psi_2]$
$A\_sta = BUSY, A\_sig = NONE$	$h2 = 0$	*send_tone*(A)	$h2 = 0$	$A\_sig := TONE$	$\{h1\}$	$h1 \leq 30$	$[\psi_2]\ [\psi_3]$
$A\_sta = BUSY, A\_sig = TONE$	$h1 \leq 30$	*dialing*(B)	$h1 < 30$	$A\_sig := DIALING(B)$	$\{h2\}$	$h2 = 0$	$[\psi_3]\ [\psi_4]$
$A\_sta = BUSY, A\_sig = TONE$	$h1 \leq 30$	*busy_tone*(A)	$h1 = 30$	$A\_sig = BUSY\_TONE$	$\{h1\}$	$h1 = 0$	$[\psi_3]\ [\psi_8]$
$A\_sig = DIALING(B), A\_sta = BUSY,$ $B\_sta = IDLE, B\_sig = NONE$	$h2 = 0$	*ring*(A, B)	$h2 = 0$	$A\_sig = ECHO\_RING(B),$ $B\_sig = RING(A), B\_sta := BUSY$	$\{h2\}$	$h2 \leq 60$	$[\psi_5]\ [\psi_6]$
$A\_sig = ECHO\_RING(B), B\_sig = RING(A),$ $A\_sta = BUSY, B\_sta = BUSY$	$h2 \leq 60$	*pickup*(B)	$h2 < 60$	$A\_sig := TALKING, B\_sig = TALKING$	$\{h2\}$	$h2 = 0$	$[\psi_6]\ [\psi_7]$
$A\_sig = ECHO\_RING(B), B\_sig = RING(A),$ $A\_sta = BUSY, B\_sta = BUSY$	$h2 \leq 60$	*busy_tone*(A)	$h2 = 60$	$A\_sig = BUSY\_TONE\ B\_sig = NONE,$ $B\_sta := IDLE$	$\{h2\}$	$h2 = 0$	$[\psi_6]\ [\psi_9]$

The sets of transitions and states of Σ_R are infinite. Consequently, Σ_R has the drawback to have an infinite representation. The LTS Σ_R has the same structure as the LTS of a TA. It may be possible to bring out a TA A_R which have Σ_R has its LTS. The TA A_R has a finite representation and models the behavior specified by R. A_R is called the *TA of the rule-actions set R* (TARAS). We define $A_R = (L_{A_R}, L^0_{A_R}, M_{A_R}, T_{A_R}, H, Inv_{A_R})$ as follows:

- $L_{A_R} = \{\omega \in \Omega(V) | \exists r \in R \cdot \omega \in [\psi_r] \cup [\psi_r[\delta_r]]\}$
- $L^0_{A_R} = L_{A_R}$
- $M_{A_R} = Ev$
- $T_{A_R} = \{(\omega, l_r, \varphi_r \wedge \varphi'_r, \lambda_r, \omega[\delta_r]) | \exists r \in R \cdot \omega \in [\psi_r]\}$
- H is the set of clocks defined in the description of the domain of application.

In order to define the invariant of a locations ω of A_R, let $Time_R(\omega)$ be a set of clock-constraints such that:

$$Time_R(\omega) = \{\varphi''_r | \exists r \in R \cdot \omega \in [\psi_r[\delta_r]]\} \cup \{\varphi_r | \exists r \in R \cdot \omega \in [\psi_r]\} \quad (1)$$

If φ is an element of $Time_R(\omega)$, we deduce that for each clock-interpretation $\theta \in [\varphi]$, the pair (ω, θ) is a state of the LTS Σ_R. The invariant of $\omega \in L_{A_R}$ is:

$$Inv_{A_R}(\omega) = \bigvee_{\varphi \in Time_R(\omega)} \varphi$$

6. Model and semantics of real-time scenarios

The designer does not have to provide the description of the system in the form of a set of rule-actions but in the form of a set of scenarios. The scenarios are more natural and easier to use. The rule-actions will be computed as the canonical form of the scenarios and represent their semantics. In this section, we start by presenting the formal model of real-time scenarios and then describe their canonical form.

6.1. Formalization of scenarios

Most models of scenarios are in the form of a linear sequence of *stimulus/reaction*. Such form of the specification is suitable for hardware component. In general, a real-time reactive system may have more complex behavior in which any combination of *stimulus* and *reaction* is allowed. Consequently, in our scenario model we abstract either a stimulus or a reaction into what we call an *action* which is labeled by an element in *Ev*. The description of an action includes other informations about the state of the system. We describe a scenario as set of sequences of actions which is organized in the form of a tree shown in Fig. 2. The edges of the tree of a scenario represent the actions of the scenario.

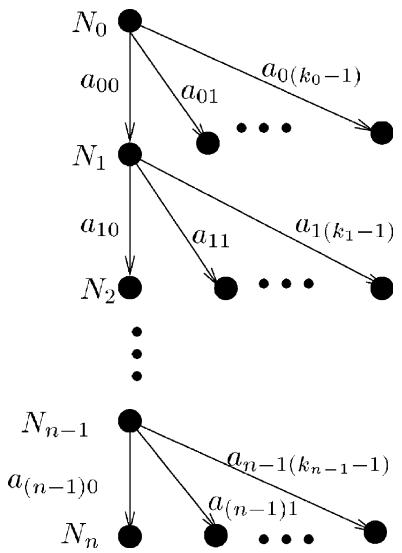

Fig. 2. Representation of the tree of a scenario.

The tree of a scenario shows the causality between its actions.

Definition 6. Each action a of a scenario is a tuple $a = \{\psi_a, \varphi_a, \varphi'_a, l_a, \delta_a, \lambda_a)$, where:

- $\varphi_a \in \Phi(H)$ and $\psi_a \in \Psi(V)$ are constraints on the system state
- $l_a \in Ev$ is an observable label
- $\varphi'_a \in \Phi(H)$ is an enabling constraint of the action a
- $\delta_a \in \Delta(V)$ is a variable-assignment updating the state of the system after the execution of a
- λ_a is a clock-assignment that occurs after the execution of the action a.

Definition 7. A scenario sc is a tree $sc = (\mathcal{N}, \mapsto)$ where:

- $\mathcal{N} = \mathcal{N}_p \cup \mathcal{N}_s$ such that $\mathcal{N}_p$ and $\mathcal{N}_s$ are, respectively, the sets primary and secondary vertices of the scenario sc. We write $\mathcal{N}_p = \{N_0, N_1, ..., N_{n-1}, N_n\}$. The secondary vertices and the primary vertex N_n are the leaves of the scenario tree.
- $\mapsto \subset \mathcal{N} \times \mathcal{N}$ is the relation predecessor–successor between the vertices the tree. For each pair $(N, N') \in \mapsto$ is identified as an action of the scenario. We write $N \xrightarrow{a} \times N'$.
- Each primary vertex N_i such that $0 \leq i \leq n-1$ is an indexed set of k_i actions: $N_i = \{a_{i0}, ..., a_{i(k_i-1)}\}$. The action a_{i0} is the primary action of the vertex N_i and connects two adjacent primary vertices in the form $N_i \xrightarrow{a_{i0}} N_{i+1}$.

The possible executions of the scenario starts from the root of its tree and ends by one of the scenario leaves. In a primary vertex, actions that are not the primary action, are alternatives which represent either a timer expiration, an

interruption or any problem preventing the scenario from continuing.

The execution of a primary action is a precondition for the next actions in the tree of the scenario. In order to consider this aspect, the scenario semantics is formalized by associating with each primary vertex N_i two constraints $\psi_{N_i} \in \Psi(V)$ and $\varphi_{N_i} \in \Phi(H)$ expressing its constraints of its context. *To execute an action, the state of the system has to satisfy the constraints of the context of the vertex of the action as well as the constraints of this action*. The constraints of the context of primary vertices are defined by the following sequence:

$$N_0 : \begin{cases} \psi_{N_0} = \psi_{a_{00}} \\ \varphi_{N_0} = \bigwedge_{0 \le k \le k_0 - 1} \varphi_{a_{0k}} \end{cases} \quad (2)$$

For $1 \le i \le n-1$,

$$N_i : \begin{cases} \psi_{N_i} = (\psi_{N_{i-1}}[\delta_{a_{(i-1)0}}] \wedge \psi_{a_{i0}}) \\ \varphi_{N_i} = \bigwedge_{0 \le k \le k_{i-1}} \varphi_{a_{ik}} \end{cases} \quad (3)$$

$$N_n : \begin{cases} \psi_{N_n} = \psi_{N_{n-1}}[\delta_{a_{(n-1)0}}] \\ \varphi_{N_n} = (\varphi_{N_{n-1}} \wedge \varphi'_{a_{(n-1)0}})[\lambda_{a_{(n-1)0}}] \end{cases} \quad (4)$$

The execution of an action of a scenario depends on the constraints of the context. Let a_{ij} be an action of N_i. While the state e of the system satisfies the condition $\psi_{N_i}(e) \wedge \psi_{a_{ij}}(e)$, the system may either let time pass or execute the action a_{ij} provided that $\varphi'_{a_{ij}}(e)$ is true. When the system executes a_{ij} from a state e, it moves to a new state $e' = (e[\delta_{a_{ij}}], e[\lambda_{a_{ij}}])$.

6.2. Timed automaton of a scenario

The execution of an action depends on the constraints of the context, which result from the execution the preceding action in the tree of the scenario. By grouping the description of an action and the constraints of the context of its primary vertex, we obtain a rule-action, which contains all of the conditions that the state of the system has to satisfy before and after the execution of this action. We define now the rule-action r that we obtain by adding the constraints of the context to the action a_{ij} of the vertex N_i :

- $\psi_r = \psi_{N_i} \wedge \psi_{a_{ij}}$
- $\varphi_r = \varphi_{N_i}$
- $l_r = l_{a_{ij}}$
- $\varphi'_r = \varphi'_{a_{ij}}$
- $\delta_r = \delta_{a_{ij}}$
- $\lambda_r = \lambda_{a_{ij}}$
- For φ''_r, two cases are to be distinguished according whether the target vertex after the execution of a_{ij}, in the scenario tree, it is not a leaf or it is a leaf:

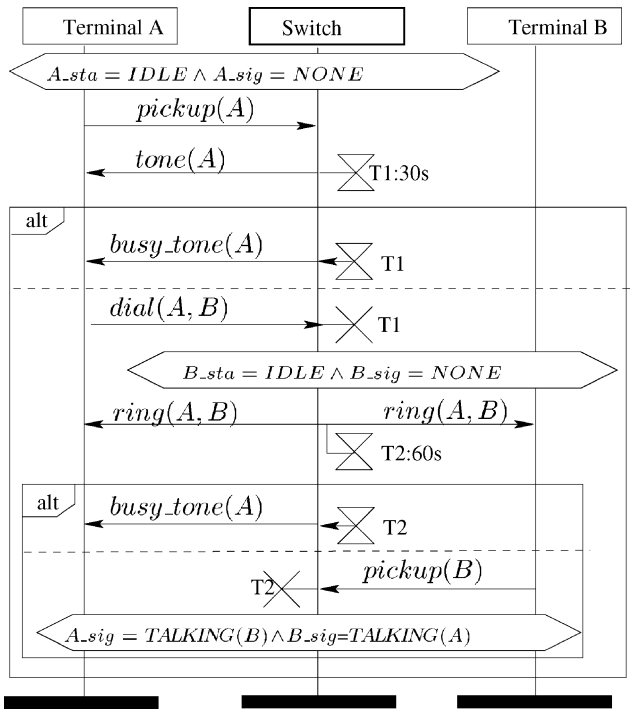

Fig. 3. An MSC describing the behavior of the telephone switch according to the rule-actions (Table 2).

(a) if $0 \le i < n-1$ and $(j = 0)$ then $\varphi''_r = \varphi_{N_{i+1}}$
(b) if $((0 \le i \le n-1$ and $0 < j \le k_i - 1)$ or $(i = n-1$ and $j = 0))$ then $\varphi''_r = (\varphi_{N_i} \wedge \varphi'_{a_{ij}})[\lambda_{a_{ij}}]$

The partial behavior of the system that a scenario describes is formally specified by the rule-actions corresponding to its actions. Let $\mathscr{R}(sc)$ designate the set of rule-actions resulting from the actions of the scenario and represents the canonical form of this scenario. The TARAS $A_{\mathscr{R}(sc)}$ associated to $\mathscr{R}(sc)$ represents an executable specification of the scenario sc. The TA $A_{\mathscr{R}(sc)}$ is used to practice intra-scenario verification which means verification of the scenario as an isolated entity.

The rule-actions of Table 2 results from a scenario which describes the establishment of a telephone call. This scenario was expressed in natural language in the caption of Table 2 and represented in Fig. 3 by a *message sequence chart* (MSC). The TARAS of this scenarios will be computed in Section 8.

7. Scenarios integration

The objective of the integration of scenarios is to merge many of them into a single TA, which represents the current prototype of the system. The resulting TA is defined as the TARAS corresponding to the rule-actions of these scenarios. The formalization of the integration of scenarios is based on an operator $\oplus$ which merges the corresponding TAs into a single TA. Particularly, let sc_1 and sc_2 be two

scenarios, the operator $\oplus$ is defined as follows:

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

How to insure that a TA, which results from the scenarios integration, allows all of the behaviors of those scenarios? Formally, what is the conformance relation between $A_{\mathscr{R}(sc_1)} \oplus A_{\mathscr{R}(sc_2)}$ and $A_{\mathscr{R}(sc_1)}$?

Theorem 1. *Let S be a set of scenarios, then $\forall sc' \in S$, the LTS of $\oplus_{sc \in S} A_{\mathscr{R}(sc)}$ simulates the LTS of $A_{\mathscr{R}(sc')}$.*

Theorem 1 means that if an action of the scenario sc' is executable in a state of the system according to $A_{\mathscr{R}(sc')}$, this action is also executable in the same state of the system according to the TA $\oplus_{sc \in S} A_{\mathscr{R}(sc)}$. The reverse implication is not true because the TA $\oplus_{sc \in S} A_{\mathscr{R}(sc)}$ may contain some extra behaviors which are in no scenario but results from the overlapping between scenarios. Fig. 4 shows the overlapping between sc_1 and sc_2. The behavior $l_1 l'_2$ is an extra that results from the overlapping between sc_1 and sc_2. It is possible to identify these extra behaviors for an eventual validation process.

The salient features of our integration method are:

- The specification of a system is incremental. Assuming that the current specification of an existing system results from the integration 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 prototype of the system is now the TA $A_R \oplus A_{R'}$ where $R = \bigcup_{sc \in S} \mathscr{R}(sc)$ and $R' = \bigcup_{sc \in S'} \mathscr{R}(sc)$. Each intermediate prototype of the system may be checked for the detection of possible features interaction.
- As $\cup$ is commutative and associative the operator $\oplus$ is also a commutative and associative operator. So, the order in which scenarios are added to construct the prototype does not matter.
- Previous integration results are reused when a new scenario sc is added. In the formula $A_{\bigcup_{sc \in S} \mathscr{R}(sc)} \oplus A_{sc'}$, the first operand is not recomputed but just reused to get the new prototype of the system.
- Our method of scenario integration facilitates both the discovery of the requirements and the automatic synthesis of the specification. Moreover, it allows feature interaction detection which may be treated as scenarios overlapping detection.

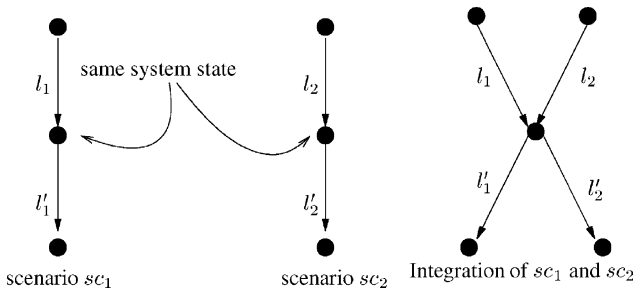

Fig. 4. An illustration of overlapping between scenarios.

8. Reduction of the TA of a rule-actions set

Since the locations of a TARAS are variable-interpretations, there is a risk of a combinatorial explosion of the number of these locations. To reduce the number of locations in a TARAS, we group together those locations from which the same rule-actions can be executed. The grouped locations satisfy a variable-constraint, which characterizes these locations. Each grouped locations represents an equivalence class. In order to construct these equivalence classes, we transform the reduction of the TARAS into the minimization of the LTS $\Sigma'_R = (Q, Q_0, \rightarrow)$:

- $Q = \{\omega \in \Omega(V) | \exists \psi \in \Psi_R \cup \Psi'_R \cdot \omega \in [\psi]\}$
- $\rightarrow = \{(\omega, l_r, \omega[\delta_r]) | \exists r \in R \cdot \omega \in [\psi_r]\}$
- $Q_0 = Q$ is the set of initial states of Σ'_R,

where Ψ_R and Ψ'_R represent two sets of sets of variable-interpretations defined by:

$$\Psi_R = \{[\psi_r] | r \in R\} \tag{5}$$

$$\Psi'_R = \{[\psi_r[\delta_r]] | r \in R\} \tag{6}$$

Each set in Ψ_R (respectively Ψ'_R) is composed of variable-interpretations that are component of a state which is a departure of a rule-action of R (respectively an arrival state after the execution a rule-action of R).

We have chosen $Q_0 = Q$ because the current specification of the system may not be complete and the initial state may be unknown yet. The minimization of Σ_R is based on an equivalence relation $\sim$. Let ω_1 and ω_2 be two equivalent variable-interpretations in Q. We write $\omega_1 \sim \omega_2$ iff:

(1a) $\forall r \in R$, $(\exists \omega'_1 \in Q \cdot \omega_1 \xrightarrow{r} \omega'_1)$ implies
$(\exists \omega'_2 \in Q \cdot \omega_2 \xrightarrow{r} \omega'_2$ such that $\omega'_1 \sim \omega'_2)$
(1b) $\forall r \in R$, $(\exists \omega'_2 \in Q \cdot \omega_2 \xrightarrow{r} \omega'_2)$ implies
$(\exists \omega'_1 \in Q \cdot \omega_1 \xrightarrow{r} \omega'_1$ such that $\omega'_1 \sim \omega'_2)$
(2a) $\forall r \in R$, $(\exists \omega'_1 \in Q \cdot \omega'_1 \xrightarrow{r} \omega_1)$ implies
$(\exists \omega'_2 \in Q \cdot \omega'_2 \xrightarrow{r} \omega_2)$
(2b) $\forall r \in R$, $(\exists \omega'_2 \in Q \cdot \omega'_2 \xrightarrow{r} \omega_2)$ implies
$(\exists \omega'_1 \in Q \cdot \omega'_1 \xrightarrow{r} \omega_1)$.

The implications (1a) and (1b) mean that the equivalence relation $\sim$ is an auto-bisimulation, so locations in which the same rule-actions can be executed are grouped together. The conditions (2a) and (2b) are needed to insure that if two locations ω_1 and ω_2 are in same equivalence class then we have $Time_R(\omega_1) = Time_R(\omega_2)$.

The minimization of Σ'_R consists in computing the equivalence classes of the relation $\sim$. In general, to minimize

an LTS, it suffices to construct the coarsest auto-bisimulation by refining a suitably chosen initial partition [9,10].

We chose as an initial partition, the coarsest partition of Q that satisfies the conditions (2a) and (2b) and then we refine it to get the coarsest auto-bisimulation.

8.1. Choosing the initial partition

Let P be the set of subset of Q defined by:

$$P = \Psi'_R \cup \left\{ Q \backslash \left(\bigcup_{[\psi] \in \Psi'_R} [\psi] \right) \right\} \tag{7}$$

P is not a partition of Q but the union of all of the elements of P is equal to Q. Let ρ_0 be the coarsest partition which is included in P. It may be proved that the partition ρ_0 is the coarsest partition of Q that satisfies the conditions (2a) and (2b) [11]. The calculus of ρ_0 will be illustrated later in the current section.

8.2. Refining the initial partition into the coarsest auto-bisimulation

Our approach is inspired by the minimization algorithm developed by Bouajjani et al. [10]. Their algorithm constructs the coarsest auto-bisimulation of the reachable part of an unlabeled transition system by refining the initial partition. An auto-bisimulation is the coarsest auto-bisimulation refining an initial partition iff all auto-bisimulations refining this initial partition, refine this coarsest auto-bisimulation. We extend the algorithm of Bouajjani et al. to treat LTS and adapt it to not perform reachability analysis because all of the states of Σ'_R are reachable.

For a partition ρ of Q and a state $\omega \in Q$, $post_\rho(\omega)$ denotes the set of elements of ρ that are immediately reachable from ω. The function $post_\rho$ is extended to the set of subsets of Q by $post_\rho([\psi]) = \bigcup_{\omega \in [\psi]} post_\rho(\omega)$ where $[\psi] \subset Q$.

Given a class $[\psi] \in \rho$, $[\psi]$ is said to be stable with respect to the partition ρ of Q iff:

$$\forall [\psi'] \in \rho \cdot ((\exists \omega_1 \in [\psi], \exists \omega'_1 \in [\psi'] \cdot \omega_1 \xrightarrow{r} \omega'_1) \text{ implies } (\forall \omega_2 \in [\psi] \exists \omega'_2 \in [\psi'] \cdot (\omega_2 \xrightarrow{r} \omega'_2)).$$

The minimization algorithm uses a $split([\psi], \rho)$ function which splits the class $[\psi]$ into stable classes with respect to ρ such that:

$$\begin{aligned} split([\psi], \rho) = \{ & [\psi_1], \ldots, [\psi_k] \mid \\ & (\forall 1 \le i, j \le k \cdot [\psi_i] \cap [\psi_j] = \emptyset) \\ & \text{and } ([\psi_1] \cup \cdots \cup [\psi_k] = [\psi]) \\ & \text{and } (\forall i \cdot [\psi_i] \text{ stable}) \\ & \text{and } (\forall i, j \cdot [\psi_i] \cup [\psi_j] \text{ non-stable}) \} \end{aligned} \tag{8}$$

The *ω-minimization* algorithm, described in Fig. 5, progressively refines the partition ρ starting from the initial partition ρ_0. *Stb* contains the stable classes with respect to ρ.

```
Algorithm ω-minimisation
Input : ρ_o,
Output: ρ partition stable
Local: Stb set of stable classes
  ρ := ρ_o ; Stb := ∅
  while (ρ ≠ Stb) do
    choose [ψ] ∈ (ρ − Stb)
    D := split([ψ], ρ)
    if D = {[ψ]} then
      Stb := Stb ∪ {[ψ]}
    else
      Stb := Stb − {[ψ'] ∈ Stb |
                [ψ] ∈ post_ρ([ψ'])} ;
      ρ := (ρ − {[ψ]}) ∪ D;
```

Fig. 5. *ω-minimization* algorithm.

The algorithm stops when all of the classes of ρ are stable with respect to ρ.

Let $\bar{\rho}$ be the resulting partition of the *ω-minimization* algorithm with ρ_0 as an initial partition. $\bar{\rho}$ is the coarsest auto-bisimulation refining ρ_0 [12,13] and consequently $\bar{\rho}$ represents the quotient set of the equivalence relation $\sim$.

8.3. Reducing a TARAS

We use the partition $\bar{\rho}$ of Q to reduce the TARAS with respect to an equivalence relation. When locations of the TARAS are grouped into a location of the quotient TARAS, they should have the same invariant which will be the invariant of the location of the quotient TARAS.

Proposition 1. *The locations of the TARAS which belong to the same $\sim$ equivalence class have the same invariant.*

This proposition is proved in Ref. [11] and allows to write $Time_R([\psi]) = Time_R(\omega)$ where $[\psi] \in \bar{\rho}$ and $\omega \in [\psi]$. We define the quotient TA of A_R as $A_R/\sim = (L_{A_R/\sim}, L^0_{A_R/\sim}, M_{A_R/\sim}, T_{A_R/\sim}, Inv_{A_R/\sim})$ where:

- $L_{A_R/\sim} = \bar{\rho}$
- $L^0_{A_R/\sim} = \bar{\rho}$
- $M_{A_R/\sim} = Ev$
- $T_{A_R/\sim} = \{([\psi], l_r, \varphi_r \wedge \varphi'_r, \lambda_r, [\psi']) \mid \exists r \in R \exists [\psi'] \in \bar{\rho} \cdot [\psi] \subset [\psi_r] \text{ and } [\psi] \in \bar{\rho} \text{ and } [\psi[\delta_r]] \subset [\psi'] \}$
- $\forall [\psi] \in L_{A_R/\sim},\ Inv_{A_R/\sim}([\psi]) = \bigvee_{\varphi \in Time([\psi])} \varphi$

Theorem 2. *The LTS of the TARAS of R and the LTS its quotient TARAS with respect to $\sim$ are bisimilar.*

This result is more general than the TA defined in Ref. [14] where the conformance relation was observational equivalence defined in Ref. [15]. The bisimulation relation is stronger than the observational equivalence relation.

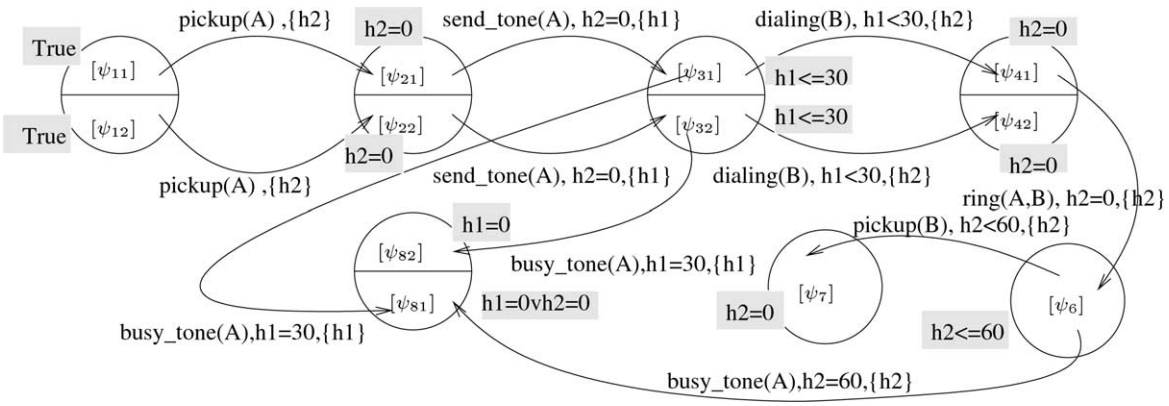

Fig. 6. Quotient TARAS of rule-actions of Table 2.

8.4. Example of application

This subsection illustrates the construction of the quotient TA of a TARAS. We consider the set of rule-actions specified in Table 2. This example aims only at illustrating the reduction of a TARAS.

In the remaining of this example, we compute the initial partition and then we apply the *ω-minimization* algorithm.

Each cell of the column (*) in Table 2 contains two elements, one in Ψ_R and the other in Ψ'_R corresponding to the rule-action described by the row of this cell.

$$\Psi_R = \{[\psi_1], [\psi_2], [\psi_3], [\psi_5], [\psi_6]\}$$

$$\Psi'_R = \{[\psi_2], [\psi_3], [\psi_4], [\psi_6], [\psi_7], [\psi_8], [\psi_9]\}$$

We compute P from Eq. (7), so $P = \{[\psi_2], [\psi_3], [\psi_4], [\psi_6], [\psi_7], [\psi_8], [\psi_9], [\psi_{10}]\}$ where

$$\psi_{10} = \underbrace{\psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4 \vee \psi_5 \vee \psi_6 \vee \psi_7 \vee \psi_8 \vee \psi_9}_{\varrho = \bigcup_{[\psi] \in \psi_R \cup \psi'_R} [\psi]} \wedge$$

$$\neg \underbrace{(\psi_2 \vee \psi_3 \vee \psi_4 \vee \psi_6 \vee \psi_7 \vee \psi_9)}_{\bigcup_{[\psi] \in \Psi'_R} [\psi]}$$

By simplifying ψ_{10} we get, $\psi_{10} = \psi_1$ and thus $P = \{[\psi_2], [\psi_3], [\psi_4], [\psi_6], [\psi_7], [\psi_8], [\psi_9], [\psi_1]\}$

The initial partition ρ_0 is obtained by partitioning $[\psi_4]$ and $[\psi_8]$. $[\psi_4]$ is partitioned with respect to $[\psi_5]$ into $[\psi_{41}]$ and $[\psi_{42}]$:

$\psi_{41} = \psi_4 \wedge \psi_5 = \psi_5$ and $\psi_{42} = \psi_4 \wedge \neg\, \psi_5$

and $[\psi_8]$ is partitioned with respect to $[\psi_9]$ into $[\psi_{81}]$ and $[\psi_{82}]$:

$\psi_{81} = \psi_8 \wedge \psi_9 = \psi_9$ and $\psi_{82} = \psi_8 \wedge \neg\, \psi_9$

So the initial partition is,

$$\rho_0 = \{[\psi_1], [\psi_2], [\psi_3], [\psi_{41}], [\psi_{42}], [\psi_6], [\psi_7], [\psi_{81}], [\psi_{82}]\}$$

We show the relevant steps of the application of *ω-minimization* algorithm (Fig. 5). After the initialization phase $\rho = \rho_0$ and $Stb = \varnothing$. The call $split([\psi_3, \rho])$ split $[\psi_3]$ into $[\psi_{31}]$ and $[\psi_{32}]$:

$$\psi_{31} = A\_sta = BUSY \wedge A\_sig = TONE \wedge B\_sta = IDLE \wedge B\_sig = NONE$$

$$\psi_{32} = A\_sta = BUSY \wedge A\_sig = TONE \wedge (B\_sta = BUSY \vee B\_sig \neq NONE)$$

The function call $split([\psi_2, \rho])$ splits $[\psi_2]$ into $[\psi_{21}]$ and $[\psi_{22}]$:

$$\psi_{21} = A\_sta = BUSY \wedge A\_sig = NONE \wedge B\_sta = IDLE \wedge B\_sig = NONE$$

$$\psi_{22} = A\_sta = BUSY \wedge A\_sig = NONE \wedge (B\_sta = BUSY \vee B\_sig \neq NONE)$$

The function call $split([\psi_1, \rho])$ splits $[\psi_1]$ into $[\psi_{11}]$ and $[\psi_{12}]$:

$$\psi_{11} = (A\_sta = IDLE) \wedge (A\_sig = NONE) \wedge (B\_sta = IDLE) \wedge (B\_sig = NONE)$$

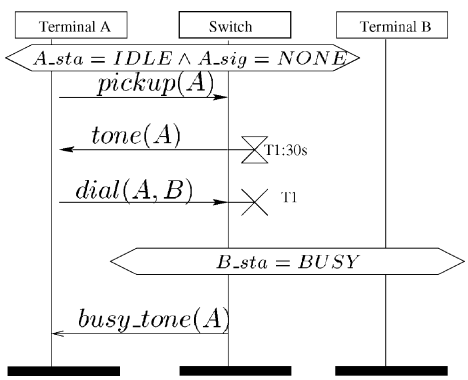

Fig. 7. This MSC is a representation of scenario sc_2.

Table 3
Specification of scenario sc_2

a	φ_a	l_a	ψ_a	φ'_a	δ_a	λ_a
a_{00}	True	$pickup(A)$	$(A\_sta = IDLE) \wedge (A\_sig = NONE)$	True	$A\_sta := BUSY$	$\{h2\}$
a_{10}	$h2 = 0$	$send\_tone(A)$	True	$h2 = 0$	$A\_sig := TONE$	$\{h1\}$
a_{20}	$h1 \leq 30$	$dial(A, B)$	True	$h1 < 30$	$A\_sig := DIALING(B)$	$\{h2\}$
a_{30}	$h2 = 0$	$busy\_tone(A)$	$(B\_sta = BUSY)$	$h2 = 0$	$A\_sig := BUSY\_TONE$	$\{h2\}$

$$\psi_{12} = (A\_sta = IDLE) \wedge (A\_sig = NONE) \wedge ((B\_sta = BUSY) \vee (B\_sig \neq NONE))$$

At this step all the classes are stable.

$$\bar{\rho} = \{[\psi_{11}], [\psi_{12}], [\psi_{21}], [\psi_{22}], [\psi_{31}], [\psi_{32}], [\psi_{41}], [\psi_{42}], [\psi_6], [\psi_7], [\psi_{41}], [\psi_{42}]\}$$

The computation of $Time(\psi)$ for all ψ in $\bar{\rho}$ allows to determine the invariants of the quotient TARAS locations. The quotient TARAS is represented by the graph of Fig. 6. The clock-constraints with a gray background are the invariant of the locations.

9. Example of application (continued)

We consider again the example of the telephone switch previously described. We adopt the same application domain description of Fig. 1. We assume now that the behavior of the system is specified by two scenarios sc_1 and sc_2. The first one sc_1 corresponds to the rule-actions of Table 2 and described by the MSC of Fig. 3.

The second scenario sc_2 specify how the system behaves if a terminal A calls a terminal B while the later is busy. We describe the scenario sc_2 by the MSC shown at Fig. 7. sc_2 consists of sending $busy\_tone(A)$ to terminal A if the called terminal B is busy. Table 3 shows the specification of scenario sc_2. All of the rule-actions in $R(sc_2)$ are already in $R(sc_1)$ except one which corresponds to action a_{30}. We synthesize a TA which represents a prototype of the telephone switch system by integrating scenarios sc_1 and sc_2.

We drew in Fig. 8 the quotient TARAS of the TA which results from the integration of scenarios sc_1 and sc_2.

By adding the rule-action of a_{30} to $R(sc_1)$ we get exactly $R(sc_1) \cup R(sc_2)$. During the computation of the initial partition ψ_{42} is partitioned into ψ_{421} and ψ_{422}. The application of *ω-minimization* causes splits of classes ψ_{32}, ψ_{22} and ψ_{12}.

10. Conclusion

Our main contributions are the formalization of the behavior of real-time reactive systems in the form of real-time scenarios and the synthesis of a formal specification by an automatic integration of scenarios. Our integration method is insensitive to the order in which scenarios are integrated. This property facilitates the practice of incremental specification.

Our method of scenarios integration was implemented as a formal specification support tool. The input to the tool is description of the domain application and a set of scenarios. The tool synthesizes a prototype of the system in the form of a reduced TA if no specification error or inconsistency are detected. The tool is now under test at France Telecom labs,

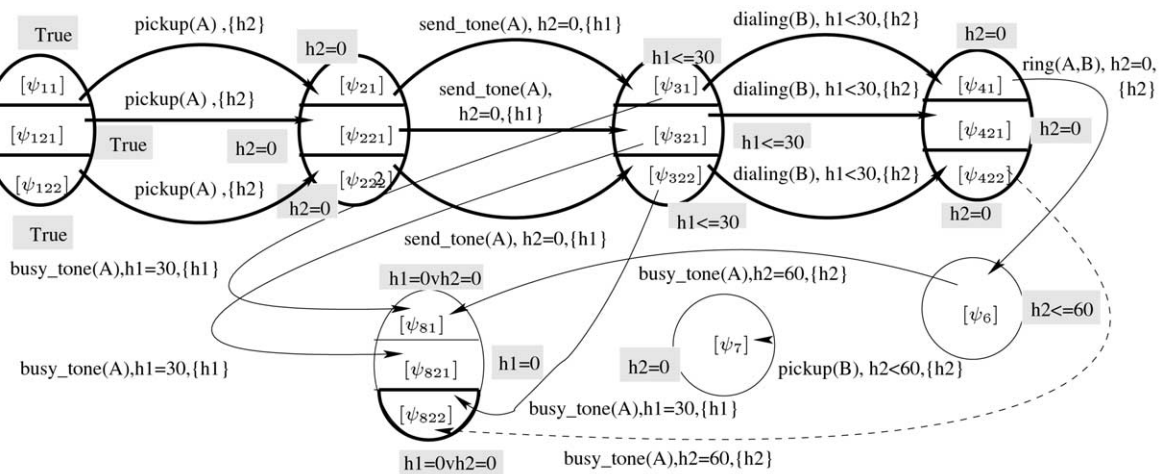

Fig. 8. The quotient TARAS of the TA resulting from integration of scenarios sc_1 and sc_2. Transitions and locations in thick line are shared by scenario sc_1 and sc_2. The transition in dashed line is proper to sc_2 and corresponds to a_{30}. Transitions and locations in thin line are proper to sc_1.

our industrial partner, to experiment its practical applicability.

References

[1] B.W. Boehm, Software engineering, IEEE Transaction on Computers C-25 (12) (1976) 1226–1241.

[2] R. Alur, D. Dill, A theory of timed automata, Theoretical Computer Science 126 (1994) 183–235.

[3] X. Nicollin, J. Sifakis, S. Yovine, Compiling real-time specifications into extended automata, IEEE Transactions on Software Engineering 18 (9) (1992) 794–804. Special Issue on the Specification and Analysis of Real-Time Systems.

[4] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima, C. Chen, Formal approach to scenario analysis, IEEE Software 11 (1994) 33–41.

[5] M. Glinz, An integrated formal model of scenarios based on statecharts, in: W. Schäfer, P. Botella (Eds.), Proceedings of the Fifth European Software Engineering Conference, Lecture Notes in Computer Science, vol. 989, Springer, Berlin, 1995, pp. 254–271.

[6] B. Dano, H. Briand, F. Barbier, A use case driven requirements engineering process, Requirements Engineering Journal 2 (2) (1997) 79–91. Springer.

[7] D. Amyot, L. Logrippo, R. Buhr, Spécification et conception de systèmes communicants: une approche rigoureuse basée sur des scénarios d'usage, CFIP97, Liège, Belgique, 1997, pp. 159–174.

[8] R. Dssouli, S. Some, J. Vaucher, A. Salah, Service creation environment based on scenarios, Information and Software Technology 41 (11/12) (1999) 697–713.

[9] J. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Language, and Computation, Addison-Wesley, Reading, MA, 1979.

[10] A. Bouajjani, J.-C. Fernandez, N. Halbwachs, P. Raymond, C. Ratel, Minimal state graph generation, Science of Computer Programming 18 (1992).

[11] A. Salah, Génération automatique d'une spécification formelle à partir de scénarios temps-réels, PhD Thesis, Université de Montréal, May 2002.

[12] J.-C. Fernandez, An implementation of an efficient algorithm for bisimulation equivalence, Science of Computer Programming 13 (2/3) (1990) 219–236.

[13] R. Paige, R.E. Tarjan, Three partition refinement algorithms, SIAM Journal on Computing 16 (6) (1987) 973–989.

[14] A. Salah, R. Dssouli, G. Lapalme, Compiling real-time scenarios into a timed automaton, in: M. Kim, B. Chin, S. Kang (Eds.), Formal Description Techniques and Protocol Specification, Testing and Verification FORTE XIV/PSTV XXI 2001, Kluwer Academic Publishers, Cheju Island, Korea, 2001, pp. 135–150.

[15] R. Milner, A Calculus for Communicating Processes, Lecture Notes in Computer Science, vol. 92, Springer, Berlin, 1980.