From Scenarios to Timed Automata: Building Specifications from Users Requirements

Stéphane Somé, Rachida Dssouli, Jean Vaucher
Département d’Informatique et de Recherche Opérationnelle,
Université de Montréal.
C.P.6128, succ centre ville Montreal, P.Q. Canada H3C 3J7

Abstract

Scenarios as partial behavior descriptions, are used more and more to represent users requirements, and to conduct software engineering. This paper examines automatic generation of specifications from requirements. This is a crucial step when accuracy is desired in the requirement engineering process. Automatic construction of specifications from scenarios reduces to the merging of partial behaviors into global specifications, such that these specifications can reproduce them. This paper presents an incremental algorithm that synthesizes timed automata from scenarios with timing constraints. The algorithm is based on a formalism developed for scenarios. Its uses operations semantics, and a mapping between concepts of scenarios, and those of the theory of timed automata.

1 Introduction

A scenario is a partial behavior description of the interaction between a system and its environment in a restricted situation [3], composed of a succession of operations constrained by timing requirements.

Scenarios are an interesting way to express requirements as they describe how users want a system to behave. Their partial nature allows them to represent parts of a system behavior, making it possible for several users with different views or uses of a same system, to provide different but possibly overlapping scenarios. In fact, several proposed and actual system development methods use scenarios, or related concepts such as uses cases, to describe users requirements [8, 2, 10, 7]. We presented in [11], a requirement engineering method where scenarios are used for requirements description, as a preliminary step towards producing complete and valid specifications.

As scenarios are only partial descriptions, producing specifications from them therefore requires a way to merge these partial behaviors to obtain global ones. The merging process must produce a specification that includes all desired behaviors, with respect to temporal aspects and conditions. This paper describes an algorithm, that generates a specification from scenarios. We use timed automata [1] as a target specification language.

The paper is organized as follow. We present the scenario concept in Section 2, and the theory of timed automata in Section 3. The specification generation algorithm is described in Section 4, while Section 5 shows a specification construction example. Section 6 examines some related works and Section 7 concludes this paper.

2 Scenarios

A scenario is a partial description of a system and environment interaction, arising in a restricted situation. This section goes beyond this definition. It presents an operational view of scenarios, considers how they are composed, and explains the formal representation that is required for the specification generation algorithm.

2.1 Operational view of scenarios

A scenario can be represented as a sequence of operations and time of occurrence, that may depends on conditions in the system and environment. A scenario restricted situation, its pre-condition, is a set of conditions that must hold in the system and environment prior to the scenario execution. Possible times of occurrence of operations may also depend on temporal constraints.

Figure 1 shows a scenario that describes an interaction between a CUSTOMER and an automated
teller machine (ATM). We represent conditions as pairs 
<entity, value> describing the fact that an entity (a
system or environment component or attribute) has a
certain value. A condition can also reflect the fact that
an operation has been executed. As an example condition
<card, inserted> is asserted by default after oper-
ation insert card. The scenario in Figure 1, can be ex-
ecuted in situation <display, card.insert.prompt >. One of
the operation sequence permitted is insert card by
CUSTOMER, display pin.enter.prompt by ATM, en-
ter pin by CUSTOMER, check id by ATM, and if af-
after this operation the situation <id, invalid > and
<number.attempts, X> with X > 3 prevails, retain card
and reinit by ATM. Temporal constraints are such that
operation enter pin can be done only between 5 seconds
and 60 seconds after operation display pin.enter.prompt,
and operation check id must have occurred 60 seconds
after it. Alternative scenarios, that apply when these
temporal constraints are not respected, can be supplied
elsewhere.

2.2 Composing scenarios

Scenarios being descriptions of partial behavior,
global behavior is obtained by composing several of
them. There are three ways to combine scenarios: se-
quential composition, alternative composition and
parallel composition.

### 2.2.1 Sequential composition

Sequential composition produces a behavior where sce-
narios follow each other. Such a composition occurs
between scenarios that overlap. Figure 2 shows two

<table>
<thead>
<tr>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Resulting behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>op1</td>
<td>op2</td>
<td>op1</td>
</tr>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>op2</td>
<td>op3</td>
<td>op2</td>
</tr>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>op3</td>
<td>op4</td>
<td>op3</td>
</tr>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
</tbody>
</table>

Figure 2: scenarios sequential composition

scenarios that compose sequentially if we suppose that
op2 in scenario 2 has an applicability condition (condi-
tion that must hold prior to its execution) included
in the condition in the system and environment after
op1 in scenario 1. Two scenarios may also compose se-
quently if after executing one, the conditions in the
system and environment are included in the second pre-
condition.

### 2.2.2 Alternative composition

Alternative composition produces a behavior where
there are choices between scenarios. This kind of com-
position is obtained between scenarios that have a
common part (situation, operations) before differing.
Figure 3 shows an alternative composition of two sce-

<table>
<thead>
<tr>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Resulting behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>op1</td>
<td>op2</td>
<td>op1</td>
</tr>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>op2</td>
<td>op3</td>
<td>op2</td>
</tr>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>op3</td>
<td>op5</td>
<td>op3</td>
</tr>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>op4</td>
<td>op6</td>
<td>op4</td>
</tr>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
</tbody>
</table>

Figure 3: scenarios alternative composition

narios. In this example, the composition supposes that
conditions after executing op1 in scenario 1 are included in scenario 2's pre-condition.

A special kind of alternate composition is obtained between scenarios that have operations with complementary temporal constraints. Figure 4 shows a such alternate scenario of the scenario in Figure 1, and Figure 5, an overall behavior obtained when composing them.

Figure 4: Alternate scenario of scenario in Figure 1.

Figure 5: Overall behavior.

2.2.3 Parallel composition

Parallel composition produces a specification where each partial behavior may be taken in parallel with others. This kind of composition may be obtained from scenarios describing behaviors that occurs in separated sub-systems, or that are unrelated.

2.3 Scenarios representation

From a user point of view, a scenario is a series of interactions made of stimuli and system reactions to them. Stimuli include operations and conditions that trigger system reactions. As an example, the scenario in Figure 1 includes the following interactions:

- CUSTOMER insert card/ATM display pin.enter.prompt,
- CUSTOMER enter pin/ATM check id and
- the occurring of new situation \(< id, invalid >\) and \(< number.attempts, X >\) with X greater than 3\)/ATM retain card, ATM reinit.

The time of occurrence of operations can be constrained by interaction initial delays and timeouts, and scenario timeouts. An interaction initial delay specifies a minimal, a maximal or an exact amount of time that must pass between the interaction first operation, and the last operation of the interaction preceding it. Interactions and scenarios timeouts specifies a maximal delays for their completion. Expiry operations may be associated with timeouts in order to be executed when delays are not respected. Initial delays may be use to bound events occurrence time at a system interface, while timeouts can serve to state performance requirements.

We formally represent a scenario as a quadruple \(< R_{num}, R_P, R_I, R_D >\) where:

- \(R_{num}\) is a scenario number,
- \(R_P\) is the scenario pre-condition. \(R_P\) is a set of conditions \(<E, V>\) where: \(E\) is an entity and \(V\) a possible value of \(E\),
- \(R_I\) is a sequence of interactions \([I_1, \ldots, I_n]\). Each \(I_i = <\text{ind}_i, D_i, R_i, ID_i >\) with:
  - \(\text{ind}_i\) an initial delay,
  - \(D_i = [d_{i1}, \ldots, d_{in}]\) a set of stimuli (operations or conditions),
  - \(R_i = [r_{i1}, \ldots, r_{in}]\) operations which are system reactions,
  - \(ID_i = <d_{iv}, IDR_i >\) an interaction timeout, where \(d_{iv}\) is a delay and \(IDR_i\) a sequence of timeout expiry operations.
- \(R_D = <rv_i, D_R >\) a scenario timeout where \(rv_i\) is a scenario delay and \(D_R\) a sequence of timeout expiry operations.

As an example, assuming that the scenario described in Figure 1 number is Sc1, it can be formally represented as the quadruple \(< R_{num}, R_P, R_I, R_D >\) with:

\(R_{num} = \text{Sc1},\)
\(R_P = \{< \text{display}, \text{card.insert.prompt} >\}\)
\(R_I = [I_1, I_2, I_3]\) where:
\(I_1 = \text{nil}, [\text{insert.card}],\)
\([\text{display.pin.enter.prompt}], \text{nil} >\)
3 The theory of Timed Automata

A timed automaton is defined as a timed transition table $< \Sigma, S, S_0, C, E >$ where:

- $\Sigma$ is a finite alphabet, each symbol of the alphabet can be considered as the occurrence of an event,
- $S$ is a finite set of states,
- $S_0 \subseteq S$ is a set of start states,
- $C$ is a finite set of clock variables and
- $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ is a set of transitions.

A transition from state $s$ to $s'$ on the input symbol $a$ is represented as a 5-tuple $< s, s', a, \lambda, \gamma >$. $\lambda \subseteq C$ is a set of clock variables reseted with the transitions and $\gamma$, a set of clock constraints expressed using clock variables in $C$, that must be satisfied for the transition firing. Clock variables values are set according to a global abstract clock, and hold at each moment, the elapsed time since their resetting. The theory of Timed Automata uses a dense-time model in which time domain is a set of positive real values.

A word $(\sigma, \tau)$ recognised by a timed automaton $A$ consists on an event sequence $\sigma = \sigma_1, \cdots, \sigma_n$ and a temporal sequence $\tau = \tau_1, \cdots, \tau_m$, such that $\sigma_i$ is consumed at the moment $\tau_i$. A run $r(\delta, v)$ of a timed transition table over a timed word $(\sigma, \tau)$ is defined as an infinite sequence $r : < s_0, v_0 >^{(\sigma_1, \tau_1)} < s_1, v_1 >^{(\sigma_2, \tau_2)} < s_2, v_2 >^{(\sigma_3, \tau_3)} \cdots$, with $s_i \in S$ and $v_i \in [C \rightarrow R]$, for all $i \geq 0$, satisfying the following requirements:

- $\delta_0 \in S_0$ and $v_0(x) = 0$ for all $x \in C$ and
- for all $i > 0$, there is an edge in $E$ of the form $< s_{i-1}, s_i, \sigma_i, \lambda_i, \gamma_i >$ such that $(v_{i-1} + \tau_i - \tau_{i-1})$ satisfies $\gamma_i$ and $v_i$ equals $[\lambda_i \rightarrow 0] (v_{i-1} + \tau_i - \tau_{i-1})$.

A partial run $\tilde{r}$ of a run $r$, is a finite sequence $\tilde{r} : < s_1, v_1 >^{(\sigma_1, \tau_1)} \cdots < s_{i+n}, v_{i+n} >$ included in $r$.

A timed transition table $< \Sigma, S, S_0, C, E >$ is deterministic if and only if there is a single start state and for each pair of transitions $< s, a, \gamma_1 >$ and $< s, a, \gamma_2 >$, $\gamma_1$ and $\gamma_2$ are mutually exclusive.

4 Timed Automaton generation

This section presents the specification generation algorithm from scenarios. We first provide some definitions and principles of the algorithm, then a detailed description of specification construction. The presentation is concluded with some remarks about automata generated.

4.1 Principles of the algorithm

The generation algorithm aims at producing a timed automaton $A = (S, S_0, C, E)$ from a set of scenarios $R = (R_{scen}, R_P, R_T, R_D)$. It is an incremental algorithm, that enriches an empty specification as each scenario is added.

The algorithm is based on the expectation that there exists a partial run in the resulting automaton over each scenario. When a scenario is considered, a such partial run is sought, and if it does not exists, the automaton is augmented to include it. A correspondence is thus made between each scenario and parts of an automaton. More precisely, this correspondence exists between conditions (in scenarios) and states (in the automaton), and between interactions and transitions. Clock variables and constraints are added to transitions according to delays and timeouts in scenarios.

4.1.1 States determination

The following definitions are used to determine automaton states from scenarios. These definitions are similar
to modelling concepts introduced by state-based planning systems such as STRIPS [4].

**Proposition 1** Each state is defined by characteristic conditions which hold in this state.

**Proposition 2** A state is redundant if it has the same characteristic conditions as another state.

**Proposition 3** A state $s_0$ is a sub-state of a state $s_a$ (its sup-state), if its characteristic conditions include that of $s_a$.

Proposition 1 will be used to determine states. Proposition 2 and 3 will be used to link new scenarios to existing specification. States corresponding to a scenario are determined so that the first state in each partial run over a scenario characteristic conditions include the scenario pre-conditions, and the other states characteristic conditions are obtained by using operations semantics defined by their added-conditions and withdrawn-conditions.

**Proposition 4** An operation added-conditions is a set of conditions that becomes true after its execution, while its withdrawn-conditions is a set of conditions that are no longer true after its execution.

By default, each operation added-conditions include the fact that its has been executed. An operation can also withdraw all conditions previously asserted in the system before its execution.

![Diagram](image)

**Figure 6:** Example of conditions derivation

As an example of condition derivation, let $a$, $b$, $c$, $d$, $e$ and $f$ be conditions, let us suppose pre-conditions to be $a$ and $b$, that the operation sequence considered is $o1$, $o2$ and $o3$, and that:

- $o1$ added-conditions is $\{c\}$ and withdrawn-conditions $\{b\}$,
- $o2$ added-conditions is $\{d\}$ or $\{c\}$ and withdrawn-conditions $\{c\}$ and
- $o3$ added-conditions is $f$ and withdrawn-conditions the empty set.

The first states of partials runs over the scenario characteristic conditions include the set $\{a, b\}$, and Figure 6 shows the determination process of conditions that hold after the scenario operations. The two sets of conditions $\{a,c,f\}$ and $\{a,d,f\}$ are obtained as a result in this example, and may be used to determine other states of the partial run.

If, for a state derived from a scenario, there already exists an automaton state with the same characteristic conditions, then this existing state will be used to find the partial run. A new state is inserted in an automaton when there is no such state. State insertion is done according to the following criterions.

**Criterion 1** All transitions possible from a state must be possible from all its sub-states.

**Criterion 2** A non empty sequence of transitions must exist between any state and each of its sub-states.

Criterion 1 is motivated by the fact that operation execution depends on conditions, and whenever the initial conditions of an operation are verified, its must be possible to execute it. As all conditions of a state are verified in its sub-states, transitions possibles from these states must also be possible from their sub-states. Criterion 2 motivation is shown in figure 7, where a

![Diagram](image)

**Figure 7:** Synthetic transition addition

state $S_b$ characterized by conditions $(a, b$ and $c$) have a sup-state $S_a$ characterized by conditions $(a$ and $b$). Behavior described is such that operation $x$ can be done in $S_b$, but not in $S_a$. However in $S_a$, it should be possible to do $x$, if there is a mean to make condition $c$ becomes true. Such possibility is materialized by including a synthetic transition between $S_a$ and $S_b$, that aims at verifying condition $c$.  

52
4.1.2 Transition determination

We produce a single automaton event from each interaction. A state change may thus be provoked by a sequence of operations corresponding to stimulus and their reactions. The motivation for this kind of correspondence is that we build an abstract specification which shows the external view of a system behavior, and there may be no external visible state within interactions. The abstract description built from scenarios should however be further refined by decomposing interactions or operations.

Interactions initial delays and timeouts, and scenarios timeouts cause the addition of temporal constraints in automata transitions. Three cases should be considered:

- For an interaction with an initial delay, a clock variable is initialized in transitions arriving to states from which they can be initiated (a new clock variable may be added to the automaton), and used to express a clock constraint in transitions corresponding to the interaction.

- For an interaction with a timeout, a clock variable is initialized as for interactions initial delays. Then, a clock constraint \( c < d \) (where \( c \) is the clock variable used and \( d \) the timeout delay), is added to transitions corresponding to the interaction. When a timeout includes expiry operations, another transition with a clock constraint \( c = d \) corresponding to the execution of these operations is added.

- For scenarios with a timeout, a clock variable is initialized in transitions arriving to first states of their partial runs and a clock constraint \( c < d \) (where \( c \) is the clock variable used and \( d \) the timeout delay), is added to all transitions corresponding to the scenario interactions. When a scenario timeout includes expiry operations, transitions with a clock constraint \( c = d \) corresponding to the execution of these operations is added from all states in partials runs to the scenario, are added.

4.2 Generation algorithm

The algorithm takes as input a scenario \((R_{num}, R_P, R_T, R_D)\), and enriches an automaton \((S, S_0, \Sigma, C, E)\). It executes in two steps: first states determination and interactions insertion. The first step is a determination of states satisfying scenario pre-conditions. In the second step we determine states and transitions making the partial run in the automaton over the scenario. These two steps cause new states to be inserted to the automaton. Below we show how new states insertion is done, and describe the two step of the generation algorithm.

4.2.1 States insertion

<table>
<thead>
<tr>
<th>Input: a state ( s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A-1 ) let LCI set of sup-states of ( s ). If LCI ( \neq \emptyset )</td>
</tr>
<tr>
<td>( A-1.1 ) let LMG set of less general states of LCI in relation with ( s )</td>
</tr>
<tr>
<td>( A-1.2 ) For each ( a ) in LMG</td>
</tr>
<tr>
<td>( A-1.2.1 ) For each transition starting at ( a ), add a similar transition starting at ( s )</td>
</tr>
<tr>
<td>( A-1.2.2 ) add a synthetic transition between ( a ) and ( s )</td>
</tr>
<tr>
<td>( A-2 ) let LIC set of sub-states of ( s ). If LIC ( \neq \emptyset )</td>
</tr>
<tr>
<td>( A-2.1 ) let LPG set of more general states of LIC in relation with ( s )</td>
</tr>
<tr>
<td>( A-2.2 ) For each ( a ) in LPG add a synthetic transition between ( s ) and ( a ).</td>
</tr>
</tbody>
</table>

Figure 8: New state insertion algorithm

The addition of a new state to an automaton is done in accordance with criterion 2, that is, after insertion, a non empty sequence of transitions must exist between any state and all its sub-states in an automaton. Figure 8 shows new state insertion. A new state is inserted by adding synthetic transitions starting from its sup-states to it and other synthetic transitions starting from it to its sub-states.

As shown in line A-1.1 of the state insertion algorithm, we do not create synthetic transitions between every state and its sup-states, but we rather restrain to the set of less general states among them, in relation with \( s \). A state \( s_1 \) is less general than \( s_2 \) in relation with a state \( s \) if \( s_1 \) and \( s_2 \) are sup-states of \( s \), and the difference between characteristic conditions of \( s \) and those of \( s_1 \) does not include the difference between characteristic conditions of \( s \) and those of \( s_2 \). This restriction explains itself because within the new state sup-states, less general states are the only states that are not sup-states of any existing states, and according to the insertion criterion, there are already non empty sequences of transitions between each state and its sup-states in the automaton.

Similarly in line A-2.1 of the state insertion algorithm, a connection between a new state and all its sub-states is made by restraining to the set of more general states among them in relation with the new state. A
state \( s1 \) is more general than \( s2 \) in relation with a state \( s \) if \( s1 \) and \( s2 \) are sub-states of \( s \), and the difference between characteristic conditions of \( s \) and those of \( s1 \) does not include the difference between characteristic conditions of \( s \) and those of \( s2 \).

In line A–2.1, we repeat in the inserted state transitions starting from its sup-states as a state and its sup-states include the same conditions, and must therefore allow same interactions execution.

### 4.2.2 First states determination

First state determination (Figure 9) begins by seeking a state with characteristic conditions identical to the scenario pre-conditions. If a such state is founded, we return it with all its sub-states, and the scenario first states determination ends.

When no existing state in the automaton characteristic conditions match the pre-condition, we create a new state having this characteristic conditions, and inserts it in the automaton. The new state becomes a new initial state when it does not have any sup-state. The algorithm returns the new state with all its sub-states as the scenario first states.

#### 4.2.3 Interactions insertion

Figure 10 shows clock variables determination and initialization, in interaction insertion described in 11. Clock variables are used to state transitions timing constraints, in order to satisfy scenarios delays and timeouts. Clock variables must be set to zero in transitions preceding their use. When for a given clock, such transitions does not already exist in the automaton, we create a synthetic state and a synthetic transition going from it to the transition departing state, which initialize the clock variable (C–3).

Interaction addition is repeated for each of the scenario first states. When a scenario have a scenario delay, a clock variable is determined and initialized and a constraint is constructed from it, to be added to all transitions generated from the scenario.

Transitions realizing a scenario behavior, are generated by successive levels of interaction addition. A scenario first interaction is added from states determined using its pre-condition (Figure 9). Transitions arrival states, are founded using operations added-conditions and withdrawn-conditions, as shown in Figure 6. The set of arrival states, constitutes departing states of the scenario second interaction, but when an interaction stimulus includes conditions, only the sub-set of states in this set, in which these conditions hold is used. The states obtained are then used as the next interaction departure states, and this process is repeated until all interactions have been considered. Each transition may include temporal constraints that corresponds to initial delays, interaction timeouts and scenario timeouts. Timeout expiry operations treatment may also result on additional transitions creation as in D–1.3.2 and D–1.3.3. Synthetic states are removed from the automaton in D–2, when they are no longer needed D–2.

### 4.2.4 Non determinism in the automaton

Automata generated by our algorithm can be non deterministic: there can be several initial states, and it is possible to have the same transition leading to different states from a given one.

The existence of several initial states can be due to composition of parallel sets of scenarios, that describe behaviors in independent sub-systems. Distinct automata of parallel sub-systems, having no common transitions, may be obtained in this way.

Non deterministic transitions may be wanted or unwanted, according to systems design objectives because
specification generated are higher abstractions of their behaviors. Unwanted determinism may be created by contradictory scenarios, where same stimuli applied when same conditions hold, produce different reactions. When such transitions are obtained, that may imply a need for changes in scenarios.

Input: Flist a set of first states
D-1 For each state $s_i$ in Flist
   D-1.1 If $RD \neq \emptyset$, $RD = \langle Rdval, Rdexp \rangle$ determine and initialize clock variable $c$, using $s_i$, and construct Cont$SD$ from $Rdval$, using $c$
   D-1.2 let LS be the set with the single element $s_i$, and Icur the first interaction of the scenario
   D-1.3 For each $s_j$ in LS, let Icur $= \langle te, D, Re, tD \rangle$
      D-1.3.1 If $te \neq \emptyset$ determine and initialize a clock variable $c$ using $s_j$, and construct Cont$te$ from $te$, using $c$
      D-1.3.2 If $tD \neq \emptyset$, $tD = \langle td, Tdexp \rangle$, determine and initialize a clock variable $c$ using $s_j$, and construct Cont$td$ from $td$, using $c$
      If $Tdexp \neq \emptyset$ add expiry transitions corresponding to $Tdexp$
      D-1.3.3 If $Rdexp \neq \emptyset$ add expiry transitions corresponding to $Rdexp$
      D-1.3.4 let LSucc set of states successors of $s_j$, obtained by executing operations in $D$ and $Re$
      D-1.3.5 add in $E$ transitions between $s_j$, and elements of LSucc with event $D$ and $Re$, and time constraint Cont$SD$, Cont$te$ and Cont$td$
      D-1.3.6 remove Icur from $Rt$
      D-1.3.7 If $Rt \neq \emptyset$, $Icur$ becomes the next interaction, LS becomes LSucc, and Goto D-1.3
D-2 For each synthetic states $sbs$, let $s$ a state such that there is a transition between $sbs$ and $s$. If there are more than one transition arriving at $s$ remove $sbs$ and transition between $sbs$ and $s$

Figure 11: Interactions addition.

Synthetic states and transitions are due to misses in original scenarios. A synthetic transition between a state and one its sub-state outlines missing operations or conditions in scenarios. This situation may thus be corrected by modifying the set of scenarios.

5 A generation example

This section shows a composition of scenarios which describe an ATM behavior in interaction with a CUS-TOMER. An empty specification is incrementally enriched by two scenarios, in this example.

Scenario Sc1 shown in Figure 1 and formally described in Section 2.3, is the first scenario used. The automaton in Figure 12 is the specification obtained after adding it.

![Automaton generated from scenario Sc1](image)

Figure 12: Automaton generated from scenario Sc1

$S_0$ is the first state generated from the scenario. It is characterized by the pre-condition of Sc1, \langle display, card_insert_prompt \rangle. The second state generated, $S_1$ is obtained by computing the situation that hold after executing all operations in the scenario first interaction. This is done using operation added-condition and withdrawn-condition description. In this example, No explicit description is needed for operation insert card, but operation display pin_enter_prompt, obviously makes the display to become pin_enter_prompt, and withdraws previous condition on this device. State $S_1$ obtained after the first interaction is therefore characterized by condition \langle display, pin_enter_prompt \rangle.

The second interaction of scenario Sc1, produces four transitions, because each of operations enter PIN and check id may have two different added-conditions. After entering PIN, the total numbers of attempts is increased, and according to the focus chosen in the scenario, becomes greater than three. Operation check id can results on a valid id or an invalid one. Executing these two operations may therefore produce four possible situations, that characterize states $S_2$, $S_3$, $S_4$ and $S_5$. All transitions between $S_1$ and these states are constrained by a clock constraint $\mathbf{z_0} < 60$, as the interaction have a corresponding timeout. The expiry of this timeout, makes the ATM to reinit, an operation that cause the display to become pin_enter_prompt. A transition is thus added from $S_1$ to $S_0$ with the clock constraint $\mathbf{z_0} = 60$. The clock variable used, $\mathbf{z_0}$, is set
to zero in the single transition arriving at state $S_1$, the transition between $S_0$, and it.

Now consider the addition of a second scenario $Sc2$ whose operational view is shown in Figure 13.

![Diagram](image_url)

Figure 13: Operational view of Scenario $Sc2$

Figure 14 shows the automaton obtained after adding scenario $Sc2$. The two scenarios compose alternately as their two first interactions are identical, and they differ on the third one. States, and transitions determined by scenario $Sc2$, first and second interactions are therefore already in the automaton. As scenario $Sc2$ third interaction can be executed, only when condition $<id, valid>$ holds, transitions corresponding to it start from states $S_2$ and $S_4$, and end to state $S_8$ assuming that operation display service_menu withdraws all previous conditions and added-condition is $<display, service_menu>$. Remaining transitions are added from this state.

### 6 Related work

Our research combines two areas: scenario formalization, and partial behavior composition.

Formalization of scenarios is presented in [5] where scenario trees describe a user view of a system. A scenario tree include nodes, to represent states, and events to represent specific stimuli that may change the system state or trigger other events.

![Diagram](image_url)

Figure 14: Automaton obtained after scenario $Sc2$

Scenario trees are defined by analysts during requirement elicitation, and each of them is converted into a regular grammar which is used to construct a conceptual state machine. This abstract machine can then be used to verify inconsistencies, redundancies and incompleteness in scenarios. Abstract machines can also be used to generate other scenarios and prototypes. This work differs from ours in several ways. Our scenarios do not rely on the use of unique state names, but we use rather conditions and infers states from them. Conditions give us more flexibility when comparing and merging scenarios than state names, as there is no for-
nal mean to compare them. Another difference is that we consider timing constraints.

A kind of partial behavior merging similar to that described in our work, is described in [9]. The approach presented there uses trace diagrams, which represents scenarios as ordered events sent between objects. Finite state machines are synthesized from these trace diagrams, by an inductive inference mechanism. This approach differs from ours as trace diagrams must be first derived from scenarios, and also as we consider timing requirements. Another difference is that our algorithm is based on operation semantics (post-condition and withdrawn conditions) rather than inference inducting. Others approaches which incrementally construct systems global behavior from trace diagrams, are [12] and [6] that deals with telecommunication systems.

7 Conclusion

Scenarios describe users requirements in a natural way and shown to be useful for requirements engineering. We are concerned about accuracy in this process, and believe that automation is one way to achieve it. Requirements represented by scenarios may be used to automatically generate specifications, but the scenario concept must be formalized. The work presented here formalizes scenarios and uses the formalism developed to build an algorithm that generates timed specifications. Our method leans on operation semantics, providing us an accurate way to considers users requirements. The algorithm requires description of operations, but this step is often included in many software engineering methods.

We aim at building a requirement engineering aid tool, and automatic generation of specifications is a step toward that. We are pursuing this research to include specification completion, simulation and deal with changing requirements [11].

References


