

Behavioral Specification of Real-time Requirements

Seyed Morteza Babamir¹ and Faezeh Sadat Babamir²

¹University of Kashan, Department of Computer Engineering, Kashan, Iran

²Shahid Bahonar University of Kerman, Department of Math and Computer Science, Kerman, Iran

¹babamir@kashanu.ac.ir ²babamir@gmail.com

Abstract

This paper aims to present a systematic method to: (1) specify high-level and event based real-time requirements and (2) map the specified requirements to low-level and state-based one. The former indicates the external system behavior while the latter indicates the internal one, which the external behavior are specified in environment events and the internal behavior is specified in software entities and operations such as variables and method calls. The mapping can be used in software development process and software monitoring against safety requirements. Lastly, we apply our method to requirements of a real-time safety critical system called Railroad Crossing Control (RCC).

1. Introduction

Reconciling behavior of system software with high-level users' requirements is a concern in software engineering fields. The software development process and the software monitoring are two subjects involved in the concern, for example. Since for many systems such as reactive, real-time, embedded ones, users' real-time requirements indicate real-time constraints posing on the system environment, the concern is reconciling behavior of system software with the real-time constraints. Real-time constraints are classified into *event-driven* (or event-based) and *time-driven* (or time-based). An event-driven constraint poses that the system should timely response on *observing* an environment event; however, a time-driven constraint poses that the system should *periodically* gather the information of its environment circumstances and response to its environment if some event has already happened.

A Railroad Crossing Control (RCC) system, for example, is a real-time system consisting of event-driven constraints such as the "moving down the gate

timely on the train-arrival event"; however, a Diabetes system, for example, is a real-time system consisting of time-driven constraints such as "sampling diabetic blood sugar once every ten minutes and delivering insulin if there is a sharp rise in the blood sugar already". In the Diabetes system, having sampled the diabetic blood and found a "sharp rise" in the blood sugar, the system software transfers to the new state and compute some insulin dose. Then, the system delivers the dose to the diabetic to control it. So, for the sugar-rising *event* and the dose-delivering *action*, which forms a real-time *interaction* sequence, there is a corresponding change in the system software state.

Some software engineering fields need mapping between interaction-based requirements of their users and corresponding states changing of their software. For run-time software monitoring, for example, the mapping helps that we able to monitor the system software behavior and determine whether the software behavior is in accord with high-level users' requirements or not.

The aim of this paper is to present a method to map interaction-based real-time users' requirements to corresponding system software behaviors. In the first step, we consider user's concerns and elicit events and reactions. For example, in the RCC system, "train" is a concern and the arrival activity is an event and in the Diabetic system, the "diabetic's blood" is a concern and the "sugar-raising" activity is an event. An action is used to control the system environment, such as "insulin (delivery)" in the Diabetic system or "gate (closing)" in the RCC system.

Having elicited the events and actions in the first step, in the second step, we present an interaction-based specification of users' requirements. Each interaction is a predicate with an event-variable in its premise and an action-variable in its conclusion. Therefore, each predicate premise indicates an environment event and each predicate conclusion indicates a required action. Lastly, the interaction-

based requirements are mapped to *mode-based* ones, which a mode indicates a system software operation and corresponds with a system action. The mode-based specification indicating the system software behavior shown in the *Petri Nets* automaton [1]. The automaton, in fact, is a typical automaton of the system software behavior based on the system environment events.

The paper, indeed, in three steps systematically maps each concerned event happening in the system environment to some required state changing in the system software. By the mapping, one can monitor required software behavior in line with high-level users' requirements, for example [2]. An Instance is monitoring behavior of the diabetes system software.

The paper continues with: stating related works in section 2, describing general principles of the approach in section 3 and applying the approach to the RCC real-time and safety-critical system in section 4. Finally in section 5, some conclusions are drawn out.

2. Related works

Mapping user-level (high-level) requirements to operational-level (low-level) ones is a matter of concern, which has already proposed by others. The common method to state high-level requirements is a narrative style of requirements i.e., scenarios, which specified in *Message Sequence Charts* (MSC) [3] and the common method to specify low-level requirements i.e., behavioral model is an *automata-based* one. For instance, [4,5,6] specify high-level requirements in MSC and then generate a behavioral model in Labeled Transition System (LTS). [7] states high-level requirements in MSC and then generates a behavioral model in UML Statecharts. [8] states user-level requirements in MSC and generates requirements specification in Linear Temporal Logic (LTL) formula and then generates a state-based specification in Buchi automata. In a similar manner, some [9] state scenarios in UML Sequence Diagrams and then generate behavioral model in UML Statecharts. The derivation of the SCR tabular specification from *goal-oriented* specification of requirements is a translation from special high-level requirements to behavioral model used by [10]. Inspired by the SCR, [11] specifies high level security policies and then map them into Event Calculus (EC) [12] Formulae.

However in this paper, two kinds of real-time requirements are considered by the scenarios are stated in a sequence of real-time event-action variables called real-time interactions and is formally specified based on *Event Calculus* formulae. Then a behavioral model in *Petri-Nets* is generated from the formula. The EC is capable of stating interrelationship between event happenings and states. This feature assists us in the

bridging gap between the event-based specification and the state-base one. The presented work by this paper is compared and contrasted with the other related works in the conclusion section.

3. The approach outline

In a number of software engineering fields, such as software monitoring, software development, model-driven elaboration and validation of requirements, operationalization of user-level (high-level) requirements, i.e. synthesizing a behavioral model (operational-level) of the requirements is a matter of concern. Such a model needs to satisfy the required intentional, structural, and behavioral dimensions of users' requirements, which may be an elaborated process. Some efforts (see section 2) were recently made to systematize this process by deriving a behavior model from scenarios of interactions between the system environment and the system software.

To contribute to resolve the above mentioned concern, this paper aims to present a method to map interaction-based real-time users' requirements to corresponding system software behaviors. The mapping is accomplished in three steps. In the first step, system environment events and system reactions to the events are elicited from user's and expert's vocabulary, which is consists of their concerns. On observing an event by the system, it should timely and properly takes some action to react to the event. The sequence event-reaction indicates an interaction between the system and its environment. Since each interaction is a real-time one, its specification consisting of events and actions should be time aware. The elicitation of events and their related actions from user's and expert's concerns are described in section 3.1

In the second step, for each interaction an implication rule is generated in form of Rule $R_{1,1}$ or Rule $R_{1,2}$ with an event in its premise and an action in its conclusion. Implication Rule $R_{1,1}$ indicates a *mandatory* and Implication Rule $R_{1,2}$ indicates a *prohibitory* reaction. Rule $R_{1,1}$ states if the event e_i happens at time τ , the system is obliged to take the action a_i at most after $\Delta\tau$. Rule $R_{1,2}$ states if the event e_j happens at time τ , the system is prohibited to take the action a_j . The $\Delta\tau$ is an acceptable deadline for the mandatory reaction, which the system should meet it before the next event happening. The generation of Rule $R_{1,1}$ and Rule $R_{1,2}$ is described in section 3.2.

$(R_{1,1}) \text{ Happens}(e_i, \tau) \rightarrow \text{TakeAct}(a_i, \tau+\Delta\tau)$

$(R_{1,2}) \text{ Happens}(e_j, \tau) \rightarrow \text{TakeNotAct}(a_j)$

Rule $R_{1,1}$ and Rule $R_{1,2}$ indicates real-time interactions between the system and its environment whose premise part indicates an environment event and

the conclusion one indicates a required reaction. An instance of Rule $R_{1,1}$ and Rule $R_{1,2}$ for the Diabetic System have shown in Rule $R_{1,3}$ and Rule $R_{1,4}$ respectively.

($R_{1,3}$) Happens(sugar-raising, τ) \rightarrow
TakeAct(deliver-insulin, $\tau+\Delta\tau$) \wedge $\Delta\tau < \tau$

($R_{1,4}$) Happens(sugar-normalizing, τ) \rightarrow
TakeNotAct(deliver-insulin)

In the third step, the event-based specification is mapped to a *mode-based* one indicates a transition from the current state of the system software to a new state, which is specified by Petri-Nets automaton. A mode indicates a system software operation and corresponds with a system action. The mapping is described in section 3.3.

3.1. Eliciting events and actions

An event is an activity of a concerned entity of system environment and an action is a system response to the event. Since an event changes the current state of the system environment to the new one, we should designate each event and its states together and elicit them from user's and expert's vocabulary. User's concerns are those whose state should be observed by the controlling system. Dealing with the events in event-driven constraints and time-driven ones differs. While in the former events are first class entities observed by the system, in the latter the impacts of the events are considered after a time period. For example, in the RCC system, the system takes some action (moves the gate down) on observing arrival event by its input sensor; however, in the Diabetic system, the system takes some action (delivers some insulin) when finds a sharp rise in the blood sugar after the time period.

Considering above-mentioned difference, we think of Table 1 for event-driven constraints and Table 2 for time-driven ones. First, we consider an event variable for each event connected to a concern. If a mandatory action should be taken in response to the event, we will

Table 1. event-action constraints in the event-driven system

seq	concern	event	current state	new state	action	acceptable delay
i	c_i	e_{i1}	s_{i1}	s_{i2}	a_{i1}	$\Delta\tau_{i1}$
	
		e_{in-1}	s_{in-1}	s_{in}	a_{in-1}	$\Delta\tau_{in-1}$

Table 2. state-action constraints in the time-driven system

seq	concern	current state	new state	action	acceptable delay
i	c_i	s_{i1}	s_{i2}	a_{i1}	$\Delta\tau$
	
		s_{in-1}	s_{in}	a_{in-1}	$\Delta\tau$

consider two state-variables and an action-variable for the event; however, if it is prohibited some action is taken, we will assign a null value to the action variable. Second, in Table 1, for each action taking (i.e., the event answering), we determine an acceptable delay. In Table 2, there is the same acceptable delay time between each state changing and the corresponding *action* because the system always acts (if there is a state changing) $\Delta\tau$ time units after the time period.

For the event-driven RCC system, an instance of a concern in Table 1 is as follows: the $C_1=train$ is a concern whose states are $S_{11}=distant$ (far from the rail crossing), $S_{12}=approaching$ (near the rail crossing), $S_{13}=inside$ (within the rail crossing) and $S_{14}=passed$ (departure). The $E_{11}=arrival$ event leaves the distant state and raises the approaching one, the $E_{12}=entrance$ event leaves the approaching state and raises the inside one and the $E_{13}=departure$ event leaves the inside state and raises the passed one. State-variables are determined based on the user's concerns values. The *distant* state-variable in the RCC system, for example, indicated by value $n<0$ showing the position of the train at n meters before the arrival point of the railroad crossing.

When the system observes an event (in the event-driven system) or when a time period run out (in the time driven system) the system should consider the raised state, determine value of its action and exercise the value over its environment timely and properly. This is a reaction to the environment event or environment state changing.

The *space complexity* of the generated event-action constraints table (Table 1) for an environment consisting n ($n \geq 1$) concerns is $O(n \times m)$ which m stands for average number of states of a concern. Since the table requires 5 columns to show the environment states and the system actions, $n \times m \times 5$ cells are totally necessary for the table. The same complexity is true for the *time complexity*.

3.2. Specifying real-time interactions

The sequence event-action constraint in Table 1 and the sequence state-action in Table 2 indicate a real-time interaction between the system and its environment; so, the constraints should be time aware. This shows need to time-based specification of the interactions, which originally implied by obligatory Rule $R_{1,1}$ and prohibitory Rule $R_{1,2}$ in section 3.

Now to specify real-time interactions in the event-action constraints, we formalize Table 1. There are two relations in Table 1: (1) the relation between the environment event (the event column) and the environment states (the "current state" column and the

"new state" one), (2) the relation between the environment event and the system action. *To formalize the first relation*, we use the central axiom S in the Simplified Event Calculus (SEC)[12]. The axiom has shown by Formula $F_{1.1}$ in which β is a *fluent* and α_1 and α_2 are events. A fluent is a variable or a predicate changes its truth value during time; therefore, it is analogous with a state variable. So, if we replace the fluent by the state variable, Formula $F_{1.2}$ will show the relation between events and new states in which the system is in state s_{ij} at time τ_0 but not at time τ ($\tau_0 < \tau$); so, the $\text{HoldsAt}(s_{ij+1}, \tau)$ predicate indicates the $\text{HoldsAt}(s_{ij}, \tau_0)$ or the $\sim\text{HoldsAt}(s_{ij}, \tau)$ one. This can be stated by Formula $F_{1.3}$. In fact, since Formula $F_{1.2}$ and Formula $F_{1.3}$ can be thought of as state formulae, we can use them in state-based (behavioral) specification of requirements in section 3.3.

$$\begin{aligned}
(F_{1.1}) \text{HoldsAt}(\beta, \tau) &\leftarrow \\
&\text{Happens}(\alpha_0, \tau_0) \wedge \text{Initiates}(\alpha_0, \beta) \wedge \sim\text{Clipped}(\tau_0, \beta, \tau) \\
\text{Clipped}(\tau_0, \beta, \tau) &\equiv \\
&\text{Happens}(\alpha_1, \tau_1) \wedge \text{Terminates}(\alpha_1, \beta) \wedge \tau_0 < \tau_1 < \tau \\
(F_{1.2}) \text{HoldsAt}(s_{ij+1}, \tau) &\leftarrow \\
&\text{Happens}(e_{ij}, \tau_0) \wedge \text{Initiates}(e_{ij}, s_{ij+1}) \wedge \\
&\sim\text{Clipped}(\tau_0, s_{ij+1}, \tau) \quad 1 \leq j \leq n-1 \\
(F_{1.3}) \sim\text{HoldsAt}(s_{ij}, \tau) &\leftarrow \text{Clipped}(\tau_0, s_{ij}, \tau)
\end{aligned}$$

To formalize the second relation in Table 1, we consider obligatory Rule $R_{1.1}$ and prohibitory Rule $R_{1.2}$ and show them as obligatory Rule $R_{2.1}$ and prohibitory Rule $R_{2.2}$ respectively. Consider that: (1) for each "new state" in Table 1 there is an event (i.e. we have an Initiate predicate), (2) we assume that no environment event happens during the $[\tau_0, \tau]$ interval (i.e. we have $\sim\text{Clipped}$ predicate in the interval) and (3) Formula $F_{1.2}$ indicates that the Happens, Initiate and $\sim\text{Clipped}$ predicates raise the HoldsAt predicate at time τ and then the system should be response after $\tau + \Delta\tau_{ij}$ at most; so, obligatory Rule $R_{2.3}$ and prohibitory Rule $R_{2.4}$ and obligatory Rules $R_{2.5}$ and prohibitory Rule $R_{2.6}$ are obtained by replacing the premise of Rule $R_{2.1}$ /Rule $R_{2.2}$ by the HoldsAt predicate. Formulae $F_{1.2}$ and $F_{1.3}$ and Rules $R_{2.1}$ and $R_{2.2}$ formally together specify relations in real-time event-driven interaction.

$$\begin{aligned}
(R_{2.1}) \text{Happens}(e_{ij}, \tau_0) &\rightarrow \text{TakeAct}(a_{ij}, \tau + \Delta\tau_{ij}) \quad 1 \leq j \leq n-1 \\
(R_{2.2}) \text{Happens}(e_{ij}, \tau_0) &\rightarrow \text{TakeNotAct}(a_{ij}) \quad 1 \leq k \leq n-1 \\
&\tau_0 < \tau \\
(R_{2.3}) \text{HoldsAt}(s_{ij+1}, \tau) &\rightarrow \text{TakeAct}(a_{ij}, \tau + \Delta\tau_{ij}) \\
(R_{2.4}) \text{HoldsAt}(s_{ij+1}, \tau) &\rightarrow \text{TakeNotAct}(a_{ij}) \\
(R_{2.5}) \sim\text{HoldsAt}(s_{ij}, \tau) &\rightarrow \text{TakeAct}(a_{ij}, \tau + \Delta\tau_{ij}) \\
(R_{2.6}) \sim\text{HoldsAt}(s_{ij}, \tau) &\rightarrow \text{TakeNotAct}(a_{ij})
\end{aligned}$$

An instance of Formula $F_{1.2}$ is Formula $F_{1.4}$ and an instance of Rule $R_{2.1}$ is Rule $R_{3.1}$ for the event-driven RCC system. While Rules $R_{2.3}$ and $R_{2.4}$ show linkage between states of the environment system and its

related system action, Rules $R_{2.1}$ and $R_{2.2}$ show linkage between environment events and its related system actions; in fact, these two set of rules together formally specify an event-driven real-time interaction and show two views of it.

$$\begin{aligned}
(F_{1.4}) \text{HoldsAt}(\text{approaching}, \tau) &\leftarrow \\
&\text{Happens}(\text{arrival}, \tau_0) \wedge \text{initiates}(\text{arrival}, \text{approaching}) \\
&\wedge \sim\text{Clipped}(\tau_0, \text{approaching}, \tau), \tau_0 < \tau \\
(R_{3.1}) \text{Happens}(\text{arrival}, \tau_0) &\rightarrow \\
&\text{TakeAct}(\text{MoveDown}, \tau + \Delta\tau), \tau_0 < \tau
\end{aligned}$$

Now to specify real-time interactions in the state-action constraints, we formalize Table 2. For the time-driven constraint, we can consider obligatory Rule $R_{2.3}$ as obligatory Rule $R_{4.1}$ and prohibitory Rule $R_{2.4}$ as prohibitory Rule $R_{4.2}$ when the period time runs out. For the Diabetic system, an instance of Rule $R_{4.1}$ is Rule $F_{4.3}$ and an instance of Rule $R_{4.2}$ is Rule $R_{4.4}$.

$$\begin{aligned}
(R_{4.1}) \text{Run-out}(\text{period}, \tau) \wedge \text{HoldsAt}(s_{ij+1}) &\rightarrow \\
&\text{TakenAct}(a_{ij}, \tau + \Delta\tau) \\
(R_{4.2}) \text{Run-out}(\text{period}, \tau) \wedge \text{HoldsAt}(s_{ij+1}) &\rightarrow \\
&\text{TakenNotAct}(a_{ij}) \\
(R_{4.3}) \text{Run-out}(\text{ten-minute interval}, \tau) \wedge & \\
&\text{HoldsAt}(\text{SugarHigh}) \rightarrow \text{TakeAct}(\text{deliver-insulin}, \tau + \Delta\tau) \\
(R_{4.4}) \text{Run-out}(\text{ten-minute interval}, \tau) \wedge & \\
&\text{HoldsAt}(\text{SugarNormal}) \rightarrow \\
&\text{TakeNotAct}(\text{deliver-insulin})
\end{aligned}$$

The *time complexity* to generate Formulae $F_{1.2}$ and $F_{1.3}$ and Rules $R_{2.3}$ to $R_{2.6}$ is $O(n \times m)$ which corresponds with time complexity of generating Table 1. For Formulae $F_{1.2}$ and $F_{1.3}$, $[(4+2) \times n \times m]$ predicates to be generated which n , m , 4 and 2 stand for the number of concerns, formulae, predicates of Formula $F_{1.2}$ and those of Formula $F_{1.3}$ respectively. For Rules $R_{2.3}/R_{2.4}$ and $R_{2.5}/R_{2.6}$, at the most, $[2 \times n \times m]$ predicates to be generated if Rules $R_{2.3}$ and $R_{2.5}$ always hold and at the least, one to be generated if Rules $R_{2.4}$ and $R_{2.6}$ always do (see Relation Re_1 in section 3.3.2); so, totally $8 \times n \times m$ time units is necessary to generate the formulae and the rules.

3.3. Mapping interactions to system behaviors

In this section, we aim to represent a behavioral specification of real-time requirements specified in section 3.2. For this purpose, we use an automata based specification called Petri-Nets. To represent behavioral specification by Petri-Nets, we should map predicates of real-time interactions in section 3.2 to elements of a Petri-Net.

A Petri-Net consists of *places*, *arcs* and *transitions* which places are connected to transitions by arcs. Places constitute inputs/outputs to/from transitions. Each place may own some token(s) and associated

with each transition there are an event and some condition(s). A transition is *enabled* when its input(s) place own some tokens. When the associated event with an enabled transition happens and its condition(s) hold, the transition will *fire*. On firing a transition, the token(s) of input place(s) of the transition will be moved to its output place(s).

Transitions of a Petri-Net can be time aware which called timed transition Petri-Net (TTPN) [1]. In a TTPN, firing an enabled transition can be delayed or can be set by a deadline. Now, we deal with behavioral specification of event-driven constraints. As stated in section 3.2, there are two kinds of relation between the constraints in Table 1. The former specified by Formulae $F_{1.2}$ and $F_{1.3}$ shows relation between system environment events and system environment states and the latter specified by Rules $R_{2.3}$ and $R_{2.4}$ and Rules $R_{2.5}$ and $R_{2.6}$ shows relation between system environment states and system actions. By using Petri-Nets we show behavioral specification of both of relations concurrently. For this purpose, we designate two Petri-Nets (one for the system environment and one for the system) for each concern.

3.3.1. Behavioral specifying the System environment. In this section, we designate a Petri-Net for each concern and map Formulae $F_{1.2}$ and $F_{1.3}$ to it as follows. For each j in Formulae $F_{1.2}$ and $F_{1.3}$, we designate a transition whose input and output places are s_{ij} and s_{ij+1} respectively and its associated event and deadline are e_{ij} and $\delta\tau_{ij}$ respectively.

The Happens(e_{ij}, τ_0) predicate indicates firing the transition and the Initiates(e_{ij}, s_{ij+1}) predicate i.e., moving token from the input place of the transition to its output place. After firing the transition, the Petri-Net implies both of the HoldsAt(s_{ij+1}, τ) and the \sim HoldsAt(s_{ij}, τ) predicates (the first level of Fig. 1). Fig. 1 shows evolution of the TTPN and ρ_1 shows Reachability graph of the TTPN in which mv_0, mv_1 and ev_0, ev_1 are *marking* and *enabling* vectors respectively. Each number in the marking vector indicates the number of place tokens and each number in enabling vector indicates a transition firing deadline.

In time-driven constraint, since we deal not with Formula $F_{1.2}$, we have no discussion on the constraint; however, connected with Rules $R_{4.1}$ and $R_{4.2}$ we propose the constraint in the next section.

3.3.2. Behavioral specifying the system. In this section, we aim to represent the system behavior by Petri-Net (the second level of Fig. 1). For this purpose, we consider obligatory/prohibitory Rules $R_{2.3}$ and $R_{2.4}$ and obligatory/prohibitory Rules $R_{2.5}$ and $R_{2.6}$. These rules show relation between states of the system

environment and the system actions. If we think of a system action as a system *mode of operation* including the idle mode, the TakeAct predicate in conclusion part of Rules $R_{2.3}$ and $R_{2.5}$ will represent happening a new mode of system operation; therefore, each TakeAct predicate will represent a transition from current system mode of operation (o_{ij}) to new one (o_{ij+1}) which has shown in Relation Re_1 . We designate a pair of two places representing current and new modes of operation with a transition between them in the Petri-Net. For the TakeNoAct predicate, however, no places are considered.

$$(Re_1) \text{TakeAct}(a_{ij}, \tau + \Delta\tau_{ij}) \equiv (\text{HoldsAt}(o_{ij}, \tau) \wedge \text{HoldsAt}(o_{ij+1}, \tau + \Delta\tau_{ij}))$$

Now, we consider premise of Rule $R_{2.3}$ and the TTPN in Fig. 1. Modes o_{ij} and o_{ij+1} indicate the system operation modes before and after taking the action respectively. The behavior of the TTPN in Fig. 1 is as follows. Because the environment transition T_{ij} and the system transition a_{ij} are enabled, on the happening of the event e_{ij} , T_{ij} will fire before the deadline $\delta\tau_{ij}$. Then during $\Delta\tau_{ij}$, if system takes the action a_{ij} , tokens will remove from places s_{ij} and o_{ij} and places s_{ij+1} and o_{ij+1} will take some token (this corresponds with obligatory Rules $R_{2.3}$ and $R_{2.5}$); however, if the system takes no action, the token will be removed from place s_{ij} and place s_{ij+1} will take the token (this corresponds with prohibitory Rules $R_{2.4}$ and $R_{2.6}$).

Graph ρ_1 shows Reachability graph of the TTPN in Fig. 1. The enabled vector mv_0 has $2n$ elements in which elements 1 to n represent state s_{i1} to state s_{in} and elements $n+1$ to n represent state o_{i1} to state o_{in} . In Graph ρ_2 , evolving the TTPN from (1-a) into (1-b) represents firing both of the transition (i.e., both the event e_{ij} happens and the action a_{ij} is obligated); while, evolving the TTPN from (2-a) into (2-b) represents firing only the environment transition (i.e., event e_{ij} happens, but any action is prohibited by the system). Vector ev_1 indicates that transition T_{ij} is no longer enabled after firing.

Now, we deal with the time-driven constraints presented by Rules $R_{4.1}$ and $R_{4.2}$. For each concern c_i , we designate a TTPN and implement behavioral

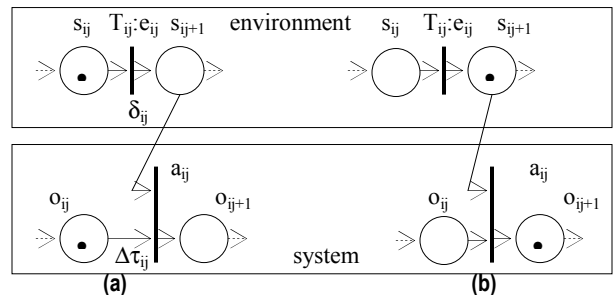
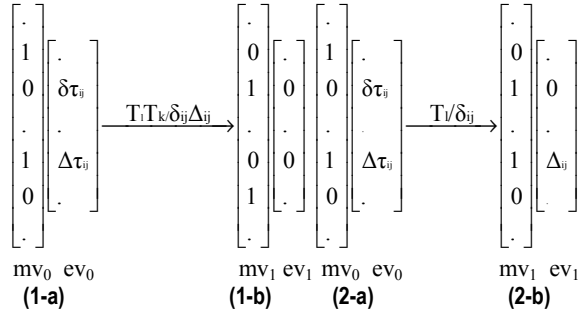


Fig. 1. Behavioral specification of an event-driven constraint



Reachability graph ρ_1 . Representing evolution of TTPN in Fig. 1

specification of obligatory/prohibitory Rules $R_{4.1}$ and $R_{4.2}$ by the TTPN as follows: for each s_{ij} ($1 \leq j \leq n-1$) in the rules, we consider a transition whose associated event and condition are the *run out* and s_{ij} respectively. The input place of the transition including a token indicates the system wait state. For obligatory Rule $R_{4.1}$, the output place of the transition indicates the system mode of operation; however, for prohibitory Rule $R_{4.2}$, the output place of the transition is its input place. Fig. 2 shows TTPN for time-driven constraint in which (a) implies obligatory Rule $R_{4.1}$ and (b) indicates prohibitory Rule $R_{4.2}$.

Now, to deal with *time complexity* of the generation of reachability graph shown in graph ρ_1 , we consider the required time both to generate the initial marking and enabling vectors and to evolve the vectors. To determine the time complexity for the marking vector initiation, we consider the HoldsAt predicates in Formulae $F_{1.2}$ and $F_{1.3}$ and Rules $R_{2.3}/R_{2.4}$ and $R_{2.5}/R_{2.6}$. There are two HoldsAt predicates in Formulae $F_{1.2}$ and $F_{1.3}$ plus at the most, two HoldsAt predicates for the TakeAct/TakeNoAct predicate (see Relation Re_1 in this section); so, the marking vector initiation will take at the most $(2+2) \times m$ time units which m stand for the number of states of a concern. The time complexity for the enabling vector initiation depends on the number of the environment events plus that of the system actions. If each concern generates m event on average and the system needs to react to half of them, there will $m+m/2$ the enabling vector entries (i.e., transitions in the Petri-Net) to be initialized. So, the total average time complexity for the vectors initialization is $5.5m$ for each concern.

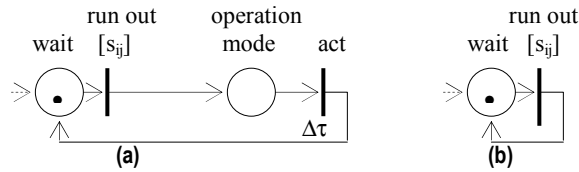


Fig. 2. The TTPN for time-driven constraint: (a) obligatory rule and (b) prohibitory rule

To determine time complexity for the Petri-Net evolution, we consider changes of the vectors values in every evolution. Since there are at most two changes for each of the vector in every evolution and the number of the evolutions is m , the evolution totally takes $4m$ time units. Therefore, time complexity for mapping the formulae and the rules to Petri-Net, on average, is $n(5.5m+4m)=9.5mn$ which m and n stand for environment events and concerns respectively.

4. The RCC system

Since we have no enough space, we only deal with an event-based real-time system called Railroad Crossing Control (RCC) system. Then based on Formulae $F_{1.2}$ and $F_{1.3}$ and Rules $R_{2.3}$ and $R_{2.4}$ and Rules $R_{2.5}$ and $R_{2.6}$, we deal with the interaction-based specification of the RCC system. Lastly, we present behavioral specification of the system.

The RCC system is intended to prevent from the collision between the train and some car at the junction of rail and road. The system comprised of an input sensor to monitor an approaching train to the cross, an output sensor to monitor a train exit from the cross, a timer to monitor the passing of time, a gate to close and open the road and a control unit. The unit controls the gate by the system application. On sensing the train, the input sensor (or the output one) notifies the control unit to move down (or move up) the gate. Therefore, the system contains three monitoring and one controlling components.

4.1. Eliciting events and actions

The system environment consists of the train and the road which the train needs to monitor; therefore, the system environment consists of the set of *concern* [$c_1=train$]. The concern states are: $s_{train}=[s_{11}=distant$ (far from the rail crossing), $s_{12}=approaching$ (near the rail crossing), $s_{13}=inside$ (within the rail crossing) and $s_{14}=passed$ (departure from the crossing)] which the default value is the "distant" value. The relevant concern *events* are: $E_{train}=[e_{11}=arrival$, which raises the approaching state, the $e_{12}=entrance$, which raises the inside one and $e_{13}=departure$, which raises the passed one]. In response to the events, the system actions are: $A_{system}=[a_{11}="gate move down"$ for the arrival event, $a_{12}="no action"$ for the entrance event and $a_{13}="gate move up"$ for the passed event].

Moreover, there is a timer to monitor the passage of the time, which the system Application will set it to zero when the application receives an E_{train} event and will increments it by one when it receives an Interrupt event; so the timer value always shows an elapsed

Table 3. event-action constraints for the RCC system

seq	concern	event	current state	new state	action	acceptable delay
1	c ₁	e ₁₁	s ₁₁	s ₁₂	a ₁₁	$\Delta\tau_{11} < t$
		e ₁₂	s ₁₂	s ₁₃	a ₁₂	-
		e ₁₃	s ₁₃	s ₁₄	a ₁₃	$\Delta\tau_{13} < t/3$

time of an event (i.e., $\delta\tau$). Same as Table 1, we show the event-actions for the RCC system in Table 3.

4.2. Specifying real-time interactions

There are two relations: (1) the relation between E_{train} and S_{train} , (2) the relation between E_{train} and A_{system} . We consider Formulae $F_{1,2}$ and $F_{1,3}$ and formalize the *first relation* as Formulae $F_{2,1}$ and $F_{2,2}$.

(F_{2,0}) Initially_T(s₁₁)

(F_{2,1}) HoldsAt(s_{1j+1}, τ) \leftarrow

Happens(e_{1j}, τ_0) \wedge Initiates(e_{1j}, s₁₂) \wedge

\sim Clipped(τ_0 , s_{1j+1}, τ)

(F_{2,2}) \sim HoldsAt(s_{1j}, τ) \leftarrow Clipped(τ_0 , s_{1j}, τ) (j=1,2,3)

To formalize the *second relation*, we designate some rules as follows: (1) we consider obligatory Rules R_{2,3} and R_{2,5} and show them as Rules R_{5,1} and R_{5,2}. The same holds for Rules R_{5,5}, R_{5,6}, (2) we consider prohibitory Rules R_{2,4} and R_{2,6} and show them as Rules R_{5,3} and R_{5,4}.

(R_{5,1}) HoldsAt(s₁₂, τ) \rightarrow TakeAct(a₁₁, $\tau + \Delta\tau_{11}$)

(R_{5,2}) \sim HoldsAt(s₁₁, τ) \rightarrow TakeAct(a₁₁, $\tau + \Delta\tau_{11}$)

(R_{5,3}) HoldsAt(s₁₃, τ) \rightarrow TakeNotAct(a₁₃)

(R_{5,4}) \sim HoldsAt(s₁₂, τ) \rightarrow TakeNotAct(a₁₃)

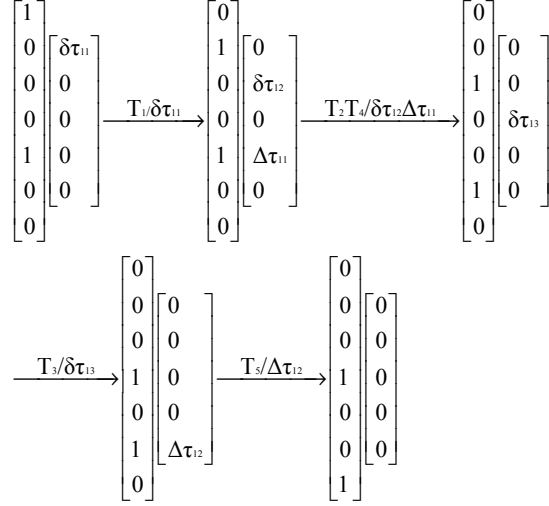
(R_{5,5}) HoldsAt(s₁₄, τ) \rightarrow TakeAct(a₁₃, $\tau + \Delta\tau_{13}$)

(R_{5,6}) \sim HoldsAt(s₁₃, τ) \rightarrow TakeAct(a₁₃, $\tau + \Delta\tau_{13}$)

4.3. Mapping interactions to system behaviors

As stated in section 3.3, to represent behavioral specification we use Petri-Nets. For the RCC system, we should map predicates of formulae and rules proposed in section 4.2 to elements of a TTPN.

Now, to specify the train behavior, we represent a Petri-net by using Formulae F_{2,0} to F_{2,2}. As stated in section 3.3.1, for each formula, we designate a transition whose input and output places are s_{1j} and s_{1j+1} (j=1,2,3) respectively and its associated event and deadline are e_{1j} and $\delta\tau_{1j}$ respectively (Fig. 3). The Happens(e_{1j}, τ_0) predicate (j=1,2,3) indicates firing the transition and the Initiates(e_{1j}, s_{1j+1}) predicate (j=1,2,3) i.e., moving token from the input place of the transition to its output place. After firing the transition, the Petri-Net implies both of the HoldsAt(s_{1j+1}, τ) and the \sim HoldsAt(s_{1j}, τ) predicates (j=1,2,3).



Reachability graph ρ_2 . Representing evolution of TTPN in Fig. 3

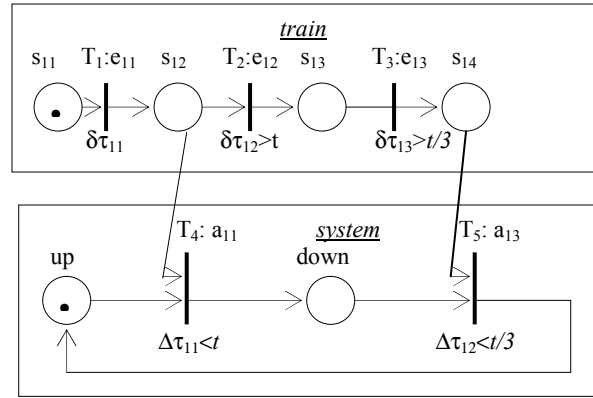


Fig. 3. The TTPN for the RCC system behavior

Now, to specify the system behavior, we use Rules R_{5,1} to R_{5,6}. For this purpose, we consider obligatory/prohibitory Rules R_{5,1} and R_{5,2}, R_{5,3} and R_{5,4} and R_{5,5} and R_{5,6}. The rules show relation between states of the train and the system actions. As stated in section 3.3.2, each TakeAct predicate can be represent by Relation Re₁(see section 3.3.2); so, we designate a pair of two places representing the HoldsAt(up,0) and HoldsAt(down, $\Delta\tau_{11}$) predicates with a transition between them in the Petri-Net. Each premise of Rules R_{5,1}, R_{5,3} and R_{5,5} show the conclusion part of Formula F_{2,1} and each premise of Rules R_{5,2}, R_{5,4} and R_{5,6} show the conclusion part of Formula F_{2,2}; so in Fig. 3, *up* and *down* places indicate the system operation modes before and after taking the action respectively. The behavior of TTPN in Fig. 3 is as follows: on happening e₁₁ Transition T₁ will fire before $\delta\tau_{11}$. Then during Δ_{11} , if system takes a₁₁, tokens will remove from s₁₁ and *up* places and then s₁₂ and *down* places will take the token (this corresponds with obligatory Rules R_{5,1} and R_{5,2}).

For TTPN in Fig.3, there is the reachability graph ρ_2 in which each marking vector consists of seven numerical values (four values for the train states and three values for the system states). Each numerical value indicates the number of tokens of a corresponding place in Fig. 3. Each enabling vector consists of four numerical values (three values for the train events and two values for the system actions) in which each positive numerical value indicates an enabled transition/action deadline and each zero value indicates a disabled transition/action.

5. Conclusion

In this paper, we proposed a method to map event and interaction based specification of real-time requirements to the behavioral one in which the former was specified based on Event Calculus Formulae and the latter was specified in Petri-Nets and its corresponding reachability graph. The time complexity of the method is $O(nm) = \#23nm$ which n stands for the environment concerns and m stands for the environment events in event-driven or the environment states in time-driven constraints.

In compare with the other related works stated in section 2, we considered some issues not proposed by them: (1) we proposed a systematic method started from users' requirements elicitation and concluded with behavioral specification of them. In our opinion, before formalizing users' requirements, they should be elicited in a proper manner. This helps requirements both to be taken comprehensively and to be ready to formalize. This is why we use a tabular method to elicit users' requirements. Using tabular method to state users' requirements is an appropriate method has already used by others [10].

(2) While others have used MSCs to state scenarios and considered untimed requirements, we considered both event-driven real-time requirements and time-driven one and stated them by scenarios in a sequence of real-time interactions. The scenarios were formalized in time-aware formulae and rules which the formulae were stated based on Event-Calculus predicates. Because the calculus is capable of stating interrelationship between event happenings and states, we could bridge gap between the interaction-based specification and the behavioral one.

(3) The used automaton we presented to specify behavior was Petri-Net. Since the net supports both concurrency and time-aware constraints, it is capable of behavioral specifying complex and real-time requirements; while the used automata by others, such as LTS one has not the capability to the requirements. However, since the UML Statecharts automaton

supports hierarchical and nested states, the detailed and in-depth requirements can be specified more clear than Petri-Nets.

Dealing with the *goal-oriented* requirements is an interesting issue used by others [10] which we didn't consider them in this paper. They have derived the event-based specification of requirements from *goal-oriented* ones in a tabular method.

6. References

- [1] David,R., Alla,H., "Discrete,Continuous and Hybrid Petri Nets", Springer-Verlag, 2005.
- [2] Feather,M.S., Fickas,S., Lamsweerde A.V. and Ponsard C, "Reconciling System Requirements and Runtime Behavior", Proceedings of Software Specification and Design, 1998, pp. 50–59, 1999.
- [3] Maum,S., "The Formalization of Message Sequence Charts", Computer Networks and ISDN system, 28, pp. 1643-1657, 1996.
- [4] Damas,C., Lambeau,B. and Lamsweerde,A.V., "Scenarios, Goals and State Machines: a Win-Win Partnership for Model Synthesis", Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 197-207, 2006.
- [5] Damas,C., Lambeau,B., Dupont,P. and Lamsweerde,A.V. , "Generating Annotated Behavior Models from End-User Scenarios", IEEE Transactions on Software Engineering, 31(12), pp. 1056-1073, 2005.
- [6] Uchitel,J., Kramer,J. and Magee,J., "Synthesis of Behavioral Models from Scenarios", IEEE Transactions on Software Engineering, 29(2), pp. 99-115, 2003.
- [7] Kruger,L., Grosu,R., Scholz,P. and Broy.M., "From MSCs to Statecharts", Proceedings of IFIP International Workshop on Distributed and Parallel Embedded Systems, Kluwer Academic Publishers, pp. 61-71, 1998.
- [8] Lamsweerde,A.V. and Willemet,L., "Inferring Declarative Requirements Specifications from Operational Scenarios". IEEE Transaction on Software Engineering, 24(12), pp. 1089-1114, 1998.
- [9] Makinen,E. and Systs,T. , "MAS-An Interactive Synthesizer to Support Behavioral Modeling in UML", Proceedings of 23rd International Conference on Software Engineering, pp. 15-24, 2001.
- [10] Landtsheer,R.D., Letier,E. and Lamsweerde,A.V., "Deriving Tabular Event-Based Specifications from Goal-Oriented Requirements Models", Proceedings of the 11th IEEE Joint International Conference on Requirements Engineering, pp. 200-210, 2003.
- [11] Babamir,S.M. and Jalili,S., "A Logical Based Approach to Detection of Intrusions against Programs", 2nd International Conference on Global E-Security, pp. 72-79, 2006.
- [12] Mueller, E.T., "Event Calculus", in Hermelen,F.V., Lifschitz,V. and Porter,B.(Eds.), Handbook of Knowledge Representation, Elsevier, pp. 1-38., 2006.