Journal of Computer Science 4 (7): 530-537, 2008
ISSN 1549-3636

Synthesizing Behavioral Model of Event-Based Requirements

Seyyed Morteza Babamir
Department of Computer Engineering, University of Kashan, Kashan, Iran

Abstract: Problem Statement: in the software engineering field, satisfaction of user's requirements by software has been a matter of concern. Therefore, monitoring software behavior against user's high-level requirements has already received a considerable and significant attention. However, the gap between low-level software behavior and high-level requirements has put an obstacle in the way of monitoring. **Approach:** to overcome the obstacle, we presented a method to synthesize a behavioral model of the event-based requirements in three steps: (1) eliciting event-based requirements; (2) specifying the requirements in event-based formulae and (3) mapping the formulae into a behavioral model. **Results**: to show effectiveness of the method, it was applied to requirements of a safety critical system, called Railroad Crossing Control (RCC) one and a behavioral model was synthesized. The model was used to synthesize monitor of the RCC system. The monitor is responsible for surveillance of software behavior for preventing the collision between the train and some car at the junction of rail and road.**Conclusions**: we proposed a systematic method started from users' requirements elicitation and concluded with its behavioral specification. Focus of the method was on event-based real-time requirements which were stated by scenarios in a sequence of real-time interactions.

Key words: Requirements Specification, Event-Based, Behavior Model

INTRODUCTION

A concern in some software engineering fields such as software monitoring, software development process, for example, has been reconciling behavior of system software with high-level users' requirements. For run-time software monitoring, for example, the reconciliation helps us to able to monitor the system software behavior and determine whether the software behavior is in accord with high-level users' requirements or not.

Since in real-time systems, users' requirements indicate real-time constraints posing on the system environment, the concern has been reconciling behavior of system software with the real-time constraints. A class of real-time systems is the event-based one in which the system is responsible for adequate response to the system environment events. The adequate response is a timely response to the environment on *observing* an environment event.

A Railroad Crossing Control (RCC) system, for example, is a real-time system that a "train arrival" is a system environment event, "moving down the gate" is a system reaction and "moving down the gate timely on observing the train-arrival event" is a user requirement.

The aim of this study was to present a method to synthesize a behavioral model from event-based users' requirements. In the first step, we considered system environment consisting of some concerns and then elicited event-based user's requirements in environment events and its related reactions. For example, in the RCC system, "train" is an environment concern and the arrival activity is an event and moving down the gate is its related reaction, which it is used to control the system environment.

Having elicited the requirements, in the second step, we presented a formal specification of the requirements in predicates having an event-variable in its premise and an action-variable in its conclusion. So, each predicate premise indicates an environment event and each predicate conclusion indicates a required action. Lastly, the requirements were mapped to a formal mode-based specification, which a mode indicates a system software operation and corresponds with a system action. The mode-based specification indicating behavior model of the system software was shown in the Petri-Nets automaton[1].

To synthesize the behavioral model, we systematically mapped a sequence of event happening and reaction to a required mode changing of the system software. The need for the mapping, which has stipulated by[2], has shown by Fig. 1[3].

Corresponding Author: Seyyed Morteza Babamir, Department of Computer Engineering, University of Kashan, Kashan, Iran
Tel: +98-361-5555333 Fax: +98-5559930

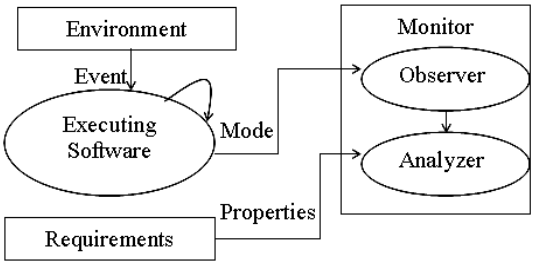


Fig. 1: Reconciling event-based requirements with software behavior[3]

Problem Statement: In the software engineering field, satisfaction of user's requirements by software has been a matter of concern. To get this satisfaction, software behavior should be monitored against user's high-level requirements. However, the gap between low-level software behavior and high-level requirements has put an obstacle in the way of monitoring. To bridge on the gap, we reconciled high-level (user-level) requirements with low-level (operational-level) ones in three steps: we (1) elicited event-based requirements; (2) specified the elicited requirements in event-based formulae and (3) mapped the formulae to a behavioral model.

MATERIAL AND METHODS

Reconciling high-level (user-level) requirements to low-level (operational-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 specified in Message Sequence Charts (MSC)[4,5] and the common method to specify low-level requirements i.e., behavioral model is an automata-based one. For instance[6-9], specify high-level requirements in MSC and then generate a behavioral model in Labeled Transition System (LTS). Kruger *et al*.[10] shows high-level requirements in MSC and then generates a behavioral model in UML Statecharts. Lamsweerd and Willemet[11] shows user-level requirements in MSC and generates requirements specification in Linear Temporal Logic (LTL) formula and then generates a state-based (mode-based) model in Buchi automata. In a similar manner, some[12,13] show scenarios in UML Sequence Diagrams and then generate behavioral model in UML Statecharts. The derivation of the SCR tabular model from goal-oriented specification of requirements is a translation from special high-level requirements to behavioral model used by[14].

However in this research, we consider real-time requirements via the scenarios stated in a sequence of real-time event- action variables called real-time interactions and is formally specified based on Event Calculus (EC)[15] formulae. Then a behavioral model in Petri-Nets is generated from the formula. The EC is capable of stating interrelationship between occurrences of environment events and initiating environment states. This feature assists us in the bridging gap between the event-based specification and the state-based one. While each EC formula formally shows an interrelationship between an environment event and an environment state changing, the Petri-Net shows the system state changing corresponding with the formula. In[16], in a reverse manner, we stated the expected behavior (states) of a program in a tabular method and then extracted the program security policies in the EC formulae from the table.

The approach principles: Our approach deals with operationalization of user-level requirements by synthesizing a behavioral model of the requirements. Some efforts stated were recently made to systematize this process by deriving a behavior model from scenarios of interactions between the system environment and the system software; however, we focus on event-based real-time systems and therefore it is necessary to use a event-aware formal method to specify event-based requirements and use a mode-based formal method supporting clearly events to specify behavioral model.

To contribute to resolve the above mentioned concern, this study aims to present a method to map event-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 an interaction is a real-time one, its specification should be time aware. The elicitation of events and their related actions from user's and expert's concerns constituting user's requirements are described in the research.

In the second step, we generate an implication rule for each interaction in form of Rule $R_{1.1}$ or Rule $R_{1.2}$ with an event in its premise and an action in its conclusion. The rules indicate a mandatory and a prohibitory reaction respectively. The first rule states if Event e_i happens at time τ, the system is obliged to take Action a_i at most after $\Delta\tau$. Rule $R_{1.2}$ states if Event e_j

happens at time τ, the system is prohibited to take Action a_j. Time $\Delta\tau$ is an allowable deadline for the mandatory reaction, which the system should meet it before the next event happening. Generation of the rules is described in the study. An instance of the rules for the RCC system is shown by Rule $R_{1.3}$ and Rule $R_{1.4}$ respectively.

($R_{1.1}$): Happens(e_i, τ) $\rightarrow$ TakeAct(a_i, $\tau+\Delta\tau$)
($R_{1.2}$): Happens(e_j, τ) $\rightarrow$ TakeNotAct(a_j)
($R_{1.3}$): Happens(train-arrival, τ) $\rightarrow$ TakeAct(gate moving down, $\tau+\Delta\tau$) $\wedge$ $\Delta\tau < \tau$
($R_{1.4}$): Happens(enter-to-cross, τ) $\rightarrow$ TakeNotAct (gate moving up)

In the third step, the event-based specification is mapped to a mode-based one indicating a transition from the current mode of the system software to a new one, 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 this study.

Obtaining Event-Based Requirements, Step One: An event is an environment activity and an action is a system response to the activity, which we obtain them from user's and expert's vocabulary (Table 1). User's concerns are the environment concerns should be observed by the system. Table 1 shows event-based requirements in which we: (1) considered user's concerns and took an event-variable for each event of a concern entity. If a mandatory action should be taken in response to an event, we will assign an action to action variable; however, if some prohibited action to be taken, we will assign a Null value to the action variable, (2) determine a maximum allowable delay for each action taking.

For the RCC system, an instance of a concern of Table 1 is as follows: the C_1 = train is a concern whose events are E_{11} = Arrival (to arrive at input line), E_{12} = Entrance (to enter to cross) and E_{13} = Departure (to pass from output line).

Formalizing event-based requirements, step two: The sequences of event-action requirement in Table 1 show a real-time interaction between the system and its environment; so, the requirements should be time-aware. This shows need to time-based specification of the interactions, which originally implied by the $R_{1.1}$ obligatory Rule and the $R_{1.2}$ prohibitory Rule in the research.

Table 1: Event-based requirements

Seq	Concern	Event	Action	Max delay
I	c_i	e_{i1}	a_{i1}	$\Delta\tau_{i1}$
		…	…	…
		e_{in}	a_{in}	$\Delta\tau_{in}$

Now, we deal with formalizing Table 1 to formally specify the requirements. For each row of Table 1, we take Rule R1.1 or Rule R1.2 and consider an environment state initiated by the event. In the RCC system, for example, we consider the approaching state initiated by the arrival event. Then, we make Rules R1.1 and R1.2 state-aware in Rules R1.3 and R1.4, which s_{ij} indicates the environment state. The new rules consist of three variable, event, state and action.

($R_{1.3}$): Happens(e_{ij}, τ) $\wedge$ Initiates(e_{ij}, s_{ij}) $\rightarrow$ TakeAct(a_{ij}, $\tau+\Delta\tau$)
($R_{1.4}$): Happens(e_{ij}, τ) $\wedge$ Initiates(e_{ij}, s_i) $\rightarrow$

Now, we use the central axiom S in the Simplified Event Calculus (SEC)[17]. The axiom has shown by Formula $F_{1.1}$ in which β is a fluent and α_0 and α_1 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 and the premise of Rules $R_{1.3}$/$R_{1.4}$ by the right-hand side of Formula $F_{1.2}$, we will have Rules 1.5 and 1.6 by which we able to show relation between environment states and related system actions. For the RCC system, an instance of Rule $R_{1.5}$ is Rule $R_{1.7}$

($F_{1.1}$): HoldsAt(β, τ) $\leftarrow$ Happens (α_0, τ_0) $\wedge$ Initiates (α_0, β) $\wedge$ ~ Clipped (τ_0, β, τ) Clipped (τ_0, β, τ) $\equiv$ Happens (α_1 , τ_1) $\wedge$ Terminates (α_1, β) $\wedge$ $\tau_0 < \tau_1 < \tau$
($R_{1.5}$): HoldsAt(s_{ij}, τ) $\rightarrow$ TakeAct(a_{ij}, $\tau+\Delta\tau$)
($R_{1.6}$): HoldsAt(s_{ij}, τ) $\rightarrow$ TakeNotAct (a_{ij})
($R_{1.7}$): Happens(arrival, τ) $\rightarrow$ TakeAct(MoveDown, $\tau+\Delta\tau$)

Specifying mode-based requirements, step three: In this study, we aim to represent a behavioral specification of event-based requirements stated by Rules $R_{1.1}$ and $R_{1.2}$ and mapped to the rules to Rules $R_{1.5}$ and $R_{1.6}$. For this purpose, we use an automata-based specification called Petri-Nets. To represent behavioral specification by Petri-Nets, we should map premise and conclusion parts of the rules predicates 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.

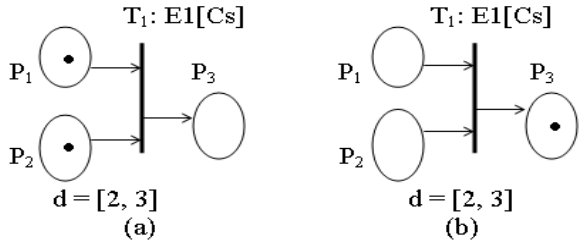


Fig. 2: A TTPN (a) before and (b) after firing

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 on 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 removed and its output place(s) will take token(s).

Transitions of a Petri-Net can be time-aware which called timed transition Petri-Net (TTPN)[18]. In a TTPN, firing an enabled transition can be delayed or can be set by a deadline. Figure 2a shows a TTPN before firing its transition in which transition T_1 has two input places (P_1 and P_2), an output place (P_3), 2 time units delay and 3 time units deadline. Since all input places of T_1 own tokens, it is enabled and will fire not before 2 time units and not after 3 time units when event E_1 happens and conditions Cs hold. Figure 2b shows the TTPN after firing its transition in which a token has removed from the input place and the output place has taken a token.

By using Petri-Nets, we can show behavioral specification of the system and its environment concurrently. For this purpose, we first designate a Petri-Net for each environment concern in study and complete it in research for the system behavior.

Specifying the environment behavior: Considering the premise of Rule $R_{1.5}$, we designated a Petri-Net for each concern as follows. For each j (state of a concern), we designated a transition whose input and output places are s_{ij-1} and s_{ij} respectively and its event and decline deadline are e_{ij} and $\delta\tau_{ij}$ respectively (Fig. 3).

Before firing the transition, token of the input place indicates the environment is in the s_{ij-1} state. After firing, the token is removed from the input place and the output place takes a token indicating the environment is in the s_{ij} state. So, the Petri-Net implies both of the HoldsAt (s_{ij}, τ) and the ~HoldsAt (s_{ij-1},τ) predicates.

We show the evolution of a TTPN of Fig. 3 by Reachability Graph ρ_1 in which mv_0 and ev_0 are marking and enabling vectors respectively.

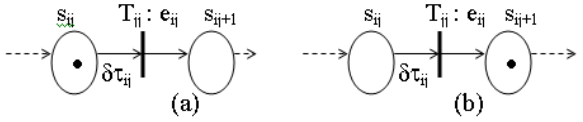


Fig. 3: The TTPN representing the environment behavior

$$\rho_1 : \begin{bmatrix} . \\ 1 \\ 0 \\ . \end{bmatrix} \begin{bmatrix} . \\ \delta\tau_{ij} \\ . \end{bmatrix} \xrightarrow{Tj/\delta\tau ij} \begin{bmatrix} . \\ 0 \\ 1 \\ . \end{bmatrix} \begin{bmatrix} . \\ 0 \\ . \end{bmatrix}$$

(a) m_0 ev_0 (b) mv_1 ev_1

Reachabilty graph ρ_1 representing Fig. 3

Each number in the marking vector indicates the number of tokens of a place and each number in enabling vector indicates deadline of firing a transition. Reachabilty graph ρ_1 states that: (1) state s_{ij-1} owns one token and state s_{ij} owns no token before firing transition T_{ij}, (2) state s_{ij-1} owns no token and state s_{ij} owns one token after firing the transition and (3) transition T_{ij} is enabled before firing and disabled after firing. Graph ρ_1, in fact, represents behavioral specification of the system environment for a concern.

Specifying the system behavior: In this research by representing the system behavior, we complete the Petri-Net designated in the study. For this purpose, we consider the conclusion part of obligatory/prohibitory Rules $R_{1.5}$/$R_{1.6}$. The 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 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-1}) to new one (o_{ij}) which has shown in Relation Re_1. We designate 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): TakeAct(a_{ij}, $\tau+\Delta\tau_{ij}$) ≡ (HoldsAt(o_{ij-1}, τ) ∧ HoldsAt(o_{ij}, $\tau+\Delta\tau_{ij}$)

We complete Fig. 3 as a TTPN in Fig. 4 in which o_{ij-1} and o_{ij} indicate the system operation modes before and after taking the action respectively. The behavior of the TTPN is as follows: because environment transition T_{ij} and the system transition are enabled, on happening event e_{ij} transition T_{ij} will fire before deadline $\delta\tau_{ij}$.

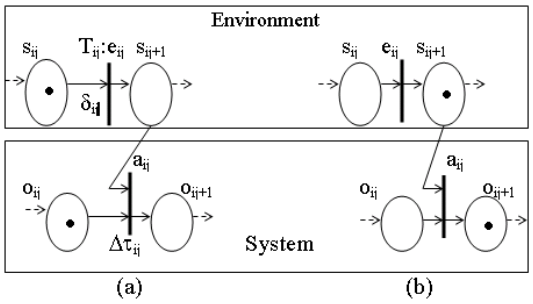
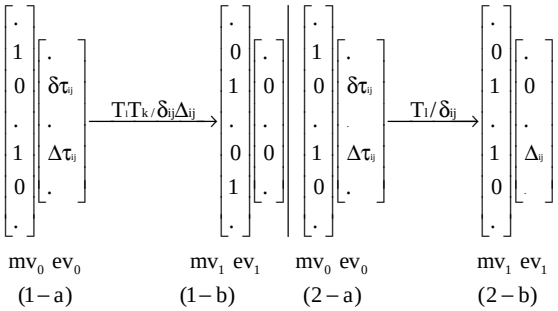


Fig. 4: Behavioral specification of an event-based system



Reachabilty Graph ρ_2 representing evolution of Fig. 4

Then during $\Delta\tau_{ij}$, if system takes action a_{ij}, tokens will remove from places s_{ij-1} and o_{ij-1} and places s_{ij} and o_{ij} will take the token; however, if the system takes no action, token will only be removed from Place s_{ij-1} and Place s_{ij} will take the token.

For the TTPN in Fig. 4, Reachability Graph ρ_1 is completed as Reachability Graph ρ_2. 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 2n represent state o_{i1} to sate o_{in}. In Graph ρ_2, evolving TTPN from (1-a) into (1-b) represents firing both of the transition (i.e., both event e_{ij} happens and action a_{ij} is obligated); while, evolving 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.

RESULTS

We applied our method to an event-based real-time system called Railroad Crossing Control (RCC) one: (1) we dealt with specification of the RCC system user's requirements and presented behavioral specification of the requirements. The RCC system has been intended to prevent from the collision between the train and some car at the junction of rail and road.

Table 2: Deadlines of the RCC real-time system

Deadline	Description
Approach to cross	The distance between the input sensor and the crossing point is given (after the train detected by the input sensor, at least it takes t time units until the train arrives at the crossing point.
Exit to detect	The interval time between two successive trains is given (there is at least t/3 time units between a train departure from the crossing point and the next train arrival at the input sensor)
Pass	Maximum speed of train is given (at least it will take a time unit until the train passes the crossing point)

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. The system responses deadlines to the events are shown in Table 2 in which the first and the second row indicate the acceptable maximum time to close and to open the road respectively.

DISCUSSION

Obtaining event-based requirements: The system environment consists of the train and the road which the train needs to monitor; therefore, the set of concerns is [c_1 = train] and the set of the concern events is E_{train} = [e_{11} = arrival, e_{12} = entrance, ane$_{13}$ = departure]. 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]. Corresponding with Table 1, Table 3 shows train events and the corresponding system actions.

Present study declared, to formalize event-base requirements stated in Table 2, we should determine the environment (train) states. Event e_{11} raises the approaching state, Event e_{12} raises the inside state and the Event e_{13} raises the passed event; so the concern states consist of: 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.

Moreover, there is a timer to monitor the passage of time, which the system application sets it to zero when the application receives an E_{train} event and increments it by one when it receives an Interrupt event; so the timer value always shows the elapsed time of an event (i.e., $\delta\tau$).

Table 3: Event-action constraints of the RCC system

S	Concern	Event	Action	Max delay
		e_{11}	a_{11}	$\Delta\tau_{11}<t$
1	c_1	e_{12}	$a_{12} = \sim a_{13}$	-
		e_{13}	a_{13}	$\Delta\tau_{13}<t/3$

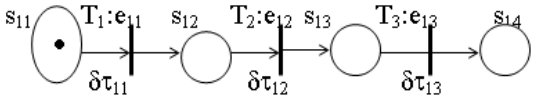
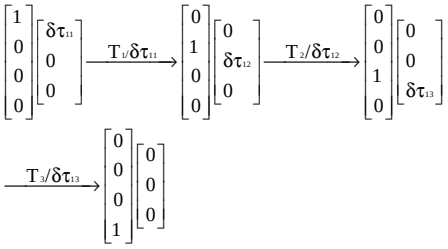


Fig. 5: The TTPN of the train behavior in the RCC system



Reachabilty graph ρ_3, representing evolution of Fig. 5

Formalizing event-based requirements: To formalize event-based requirements, we use Rules $R_{1.3}$ and $R_{1.4}$ and generate Rules $R_{2.1}$ and $R_{2.2}$ for the train. Now considering Rules $R_{2.1}$ to $R_{2.3}$, we use Rules $R_{1.5}$ and $R_{1.6}$ and generate Rules $R_{3.1}$ to $R_{3.4}$.

($R_{2.1}$): Happens(e_{11},τ) ∧ Initiates(e_{11},s_{12}) → TakeAct (a_{11}, τ+Δτ)
($R_{2.2}$): Happens(e_{12}, τ) ∧ Initiates(e_{12}, s_{13}) → TakeNotAct(a_{12})
($R_{2.3}$): Happens(e_{13},τ) ∧ Initiates(e_{13},s_{14}) → TakeAct (a_{13}, τ+Δτ)
($R_{3.1}$): Initially$_T$(s_{11})
($R_{3.2}$): HoldsAt(s_{12}, τ) → TakeAct(a_{11}, τΔτ) ∧ Δτ < τ
($R_{3.3}$): HoldsAt(s_{13}, τ) → TakeNotAct (a_{13})
($R_{3.4}$): HoldsAt(s_{14}, τ) → TakeAct(a_{13}, τΔτ) ∧ Δτ < τ

Specifying the environment behavior: To specify the train (environment) behavior, we consider: (1) Rule $R_{3.1}$, (2) the premise part of the Rules $R_{2.1}$ and $R_{3.2}$, (3) the premise part of the Rules $R_{2.2}$ and $R_{3.3}$ and (4) the premise of Rules $R_{2.3}$ and $R_{3.4}$ and synthesize TTPN of the train (Fig. 5). For each state s_{1j} ($1\leq j\leq 4$), we take a place and for each Happens, we take a transition including the e_{1j} event which the s_{11} state initially has a token. Having taken the places and the transitions, above-mentioned cases (2), (3) and (4) constitutes the TTPN.

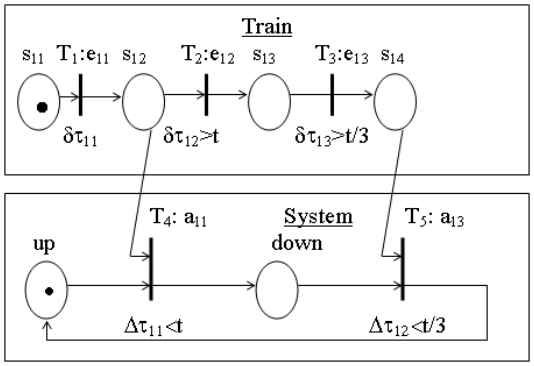
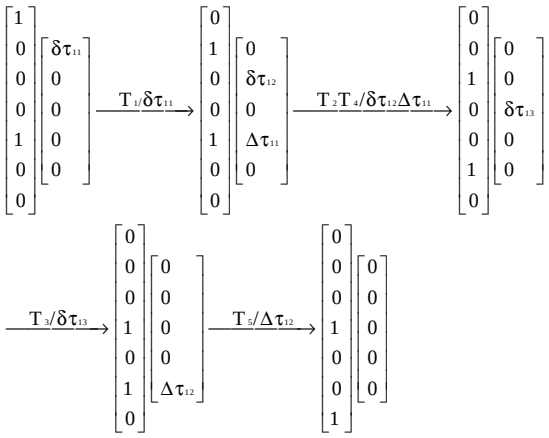


Fig. 6: The TTPN of the RCC system behavior



Reachabilty graph ρ_4, representing evolution of the TTPN in Fig. 6

The Happens (e_{1j}, τ) predicate (j = 1, 2, 3) indicates firing the transition and the Initiates(e_{1j}, s_{1j}) predicate (j = 1, 2, 3) indicates moving token from the input place of the transition to its output place. After firing the transition, the Petri-Net implies the HoldsAt(s_{1j+1}, τ) predicates (j = 1, 2, 3). We have shown the evolution of the TTPN of Fig. 5 by Reachability Graph ρ_3.

Specifying the system behavior: Now, to specify the system behavior, we use obligatory/prohibitory Rules $R_{2.1}$ to $R_{2.3}$ and $R_{3.1}$ to $R_{3.4}$ to complete the Petri-Net designated in Fig. 5. The rules show relation between states of the train and the system actions. Present study declared, each TakeAct predicate can be represent by Relation Re_1; so, we designate a pair of two places representing the HoldsAt(up, τ) and HoldsAt(down, $\tau+\Delta\tau_{11}$) predicates with a transition between them in the Petri-Net.

We completed Fig. 5 as the TTPN in Fig. 6 in which the up and down places indicate the system

operation modes before and after taking the action respectively. The behavior of the TTPN in Fig. 6 is as follows: on happening e_{11} Transition T_1 will fire before $\delta\tau_{11}$. Then during Δ_{11}, if system takes a_{11}, tokens will remove from s_{11} and up places and then s_{12} and down places will take the token.

For the TTPN in Fig. 6, Reachability Graph ρ_3 was completed as Reachability Graph ρ_4. Each element of the graph has a marking vector and an enabling one.

Each marking vector consists of seven numerical values (four values for the train states and three values for the system states) in which each numerical value indicates the number of tokens of a corresponding place of Fig. 6. 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 an disabled transition/action.

CONCLUSION

In this study, 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.

In compare with the other related research, we considered some issues not proposed by them:

- 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[14].
- While others have used MSCs to state scenarios and considered un-timed requirements, we considered event-based real-time requirements 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.
- 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 the 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 detailed than Petri-Nets.

Dealing with the goal-oriented requirements is an interesting issue used by[14] which we didn't consider them in this research. They have derived the event-based specification of requirements from goal-oriented ones in a tabular method.

REFERENCES

1. David, R. and H. Alla, 2005. Discrete, Continuous and Hybrid Petri Nets. 2nd Edn., Springer-Verlag, ISBN: 978-3-540-22480-8, pp: 524.
2. Feather, M.S., S. Fickas, A.V. Lamsweerde and C. Ponsard, 1998. Reconciling system requirements and runtime behavior. Proceedings of the 9th International Workshop on Software Specification and Design, Apr. 16-18, IEEE Computer Society, Washington, DC., USA., pp: 50-55. http://portal.acm.org/citation.cfm?id=857205.858297.
3. Delgado, N., A.Q. Gates and S.A. Roach, 2004. Taxonomy and catalog of runtime software-fault monitoring tools. IEEE Trans. Software Eng., 30: 859-872. DOI: 10.1109/TSE.2004.91.
4. ITU-TS, 2000. Application of formal description techniques Z.110-Z.119, Message sequence chart Z.120-Z.129. http://www.itu.int/ITU-T/2001-2004/com17/languages/Z.120AnnB-0498.pdf.
5. Maum, S., 1996. The formalization of message sequence charts. Comput. Networks ISDN Syst., 28: 1643-1657. DOI: 10.1016/0169-7552(95)00123-9.
6. Damas, C., B. Lambeau and A.V. Lamsweerde, 2006. Scenarios, goals and state machines: A win-win partnership for model synthesis. Proceedings of 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), Nov. 5-11, Portland, Oregon, USA., pp: 197-207. DOI: 10.1145/1181775.1181800.

7. Damas, C., B. Lambeau, P. Dupont and A.V. Lamsweerds, 2005. Generating annotated behavior models from end-user scenarios. IEEE Trans. Software Eng., 31: 1056-1073. DOI: 10.1109/TSE.2005.138.
8. Uchitel, J., J. Kramer and J. Magee, 2003. Synthesis of Behavioral Models from Scenarios. IEEE Trans. Software Eng., 29: 99-115. DOI: 10.1109/TSE.2003.1178048.
9. Letier, E., J. Kramer, J. Magee and S. Uchitel, 2005. Monitoring and control in scenario-based requirements analysis. Proceedings of 27th International Conference on Software Engineering (ICSE), May 15-21, St. Louis, MO., USA., pp: 382-391. http://portal.acm.org/citation.cfm?id=1062527.
10. Kruger, I., R. Grosu, P. Scholz and M. Broy, 1998. From MSCs to statecharts. Proceedings of IFIP International Workshop on Distributed and Parallel Embedded Systems, Kluwer Academic Publishers, 1998, Norwell, MA., USA., pp: 61-71. http://portal.acm.org/citation.cfm?id=328655.
11. Lamsweerd, A.V. and L. Willemet, 1998. Inferring declarative requirements specifications from operational scenarios. IEEE Trans. Software Eng., 24: 1089-1114. DOI: 10.1109/32.738341.
12. Whittle, J. and J. Schumann, 2000. Generating statechart designs from scenarios. Proceedings of 22nd International Conference on Software Engineering (ICSE), June 4-11, IEEE Xplore Press, USA., pp: 314-323. DOI: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=870422.
13. Makinen, E. and T. Systa, 2001. MAS-an interactive synthesizer to support behavioral modeling in UML. Proceedings of 23rd International Conference on Software Engineering (ICSE), May 12-19, pp: 15-24, IEEE Xplore Press, USA., DOI: 10.1109/ICSE. 2001.919077.
14. Landtsheer, R.D., E. Letier and A.V. Lamsweerde, 2003. Deriving tabular event-based specifications from goal-oriented requirements models. Proceedings of 11th IEEE Joint International Conference on Requirements Engineering, Sept. 8-12, IEEE Xplore Press, USA., pp: 200-210. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1232751.
15. Hermelen, F.V., V. Lifschitz and B. Porter, 2007. Handbook of Knowledge Representation. 1st Edn., Elsevier Science, USA., ISBN-10: 0444522115, pp: 1034.
16. Babamir, S.M. and S. Jalili, 2006. A logical based approach to detection of intrusions against programs. Proceedings of the 2nd Conference on Global E-Security, (ICGeS-06), London, pp: 72-79.
17. Sadri, F. And R. Kowalski, 1995. Variants of the event calculus. Proceedings of the 12th International Conference on Logic Programming (ICLP), June 13-16, MIT Press, USA., pp: 67-82. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.296.
18. Ghezzi, C., M. Jazayeri and D. Mandrioli, 2002. Fundamentals of Software Engineering. 2nd Edn., Prentice-Hall, USA., ISBN: 10: 0133056996, pp: 624.