Behavior Analysis of Real-Time Systems Using PLA Method

Dalius Makackas, Regina Miseviciene, and Henrikas Pranevicius

Kaunas University of Technology, Faculty of Informatics, Studentu 56-443, Kaunas, Lithuania
{dalius.makackas,regina.miseviciene,
henrikas.pranevicius}@ktu.lt

Abstract. The paper deals with a behavior analysis task of real-time system specified by the PLA method. An algorithm for creating a reachable state graph is used while solving for the task. The algorithm evaluates intervals of time when the defined system events occur. An approach based on the algorithm for the reachable state graph generation is presented within this paper. The suggested approach is illustrated by an example.

Keywords: real -time system, analysis, trajectory modeling, reachable state graph.

1 Introduction

A real-time system's accuracy depends not only on the logical result of computations, but also on the time at which the results are produced [2]. For a design of this type of system, high security, reliability and performance requirements are raised. Various formal methods for describing such systems functioning are applied. The most commonly used are following formal notations: Time Petri Nets [6-7, 19], Discrete Event System Specification (DEVS) [2, 5, 20], Timed Automata [1, 4, 8], Piece-linear Aggregate (PLA) [16, 18], Finite State Machine [21] and others. Such formal specifications of real-time systems can be analyzed by functionality or behavior. A functional analysis is performed by creating a system simulation model. A behavioral analysis examines all the possible trajectories of the system, while checking whether the specification is made correctly. As the real-time systems interact with their environment in real-time, time properties are very important. Thus, more recently, considerable research efforts have been devoted to verification of the time properties. There are various verification techniques [9, 10, 12, 15]. However, conventional verification methods do not perform a full inspection of real-time systems. Their main drawback is that the traditional verification methods underestimate the system performance over time, or analyzes only system whose operation time is deterministic. Functioning of such systems is described only by one trajectory. However, many operations of real-time systems depend on a certain interval. Thus, describing such systems by a single trajectory is not possible, infinitely many endings of the operation in time.

T. Skersys, R. Butleris, and R. Butkiene (Eds.): ICIST 2013, CCIS 403, pp. 392–402, 2013.

The operations may result in any precisely specified time interval. Thus, real time systems can have a number of operating trajectories. Verification of these trajectories is problematic, because of the need to generate and verify all the possible modes.

A goal of this article is to present a novel approach for creation of reachable state graph of operating trajectories for behavior analysis. The approach is based on the algorithm for the reachable state graph generation. The algorithm permits precise evaluation of specified time intervals for operations. The algorithm is designed for real-time systems specified by Piece-linear aggregate method [16].

The remainder of this paper is organized with the following approach. The next section provides a formal definition of Piece-linear aggregate; Section 2 describes real-time system functioning trajectories; Section 3 provides a reachable state graph creation algorithm; and an illustrative example is proposed in Section 4. Conclusions are presented in the last section.

2 Piece-Linear Aggregate Specification Method

The paper analyzes real-time systems, specified for by Piece-linear aggregate method [16].

A system specified by the Piece-linear aggregate method is understood as a set of interacting piece-linear aggregates. Each aggregate is defined by a set of states $Z = \{z_1, z_2 ...\}$, a set of input signals $X = \{x_1, x_2 ...\}$, a set of output signals $Y = \{y_1, y_2 ...\}$, a set of internal E'' and external E' events, a set of transition $H : E \times Z \to Z$ and output $G : E \times Z \to Y$ operators.

The aggregate method generates time-point sequences $T = \{t_0, t_1 ...\}$ and state $\{z(t_0), z(t_1) ...\}$ transitions in these time points. The state $z(t) = (v(t), z_v(t))$ consists of two components: discrete $v(t)$ and continuous $z_v(t)$. Each element $w_i(t)$ of a continuous component $z_v(t) = (w_1(t), w_2(t) ...)$ indicates a time when an event e_i occurs. The event changes j elements of discrete and continuous component of state according to the law: $h_j^v(t) = h_j^v(t, z(t))$, $h_j^w(t) = h_j^w(t, z(t))$.

In the aggregate model it is also defined the concept of the operation. This function takes the following values:

$$O_e = O_e(t) = O(e,t) = \begin{cases} 1, & \textit{it is active at time t;} \\ 0, & \textit{it ended at time t;} \\ -1, & \textit{it is pasive at time t.} \end{cases}$$

Each operation is linked with continuous component. If the operation O_e is active then value of continuous component is $w_e(t) > t$; if the operation is passive, it is not known when the next event will occur and continuous component is $w_e(t) < t$; if the operation is ended at time t then $w_e(t) = t$.

The Piece-linear aggregate specification method can be used for real-time system specification. This method is described in detail by Russian scientists N. Buslenko

and I. Kovalenko [3]. Professor H.Pranevicius proposed a modification by adding to the method control sequences, which built in comfortable assumptions of these models in computer systems realization. The Piece-linear aggregate specification is used for two purposes: to create simulation models and to validate and to verify the system. Validation and verification is based on creation of a reachable state graph. The essence of the reachable state method consists in the fact that, with the aggregate system specifications, the system generates a set of all possible trajectories. Then, the trajectories are analyzed in respect of properties under investigation.

3 Real-Time System Functioning Trajectories

Real-time systems are defined as follows: "It is an environment that responds to random external events. The respond to a particular event is a set of actions; each of them must be carried out in certain time constraints" [11, 13, 14].

Based on the definition, real-time system has the strict, fixed temporary conditions. The actions must be carried out under the defined conditions. Real-time systems are divided into two categories: real-time systems with strict requirements and real-time systems with probabilistic requirements. This article explores the systems with strict requirements. They must ensure that the appropriate actions will be carried out strictly within the prescribed time interval.

The system is investigating by analyzing functioning trajectories $S_0, e_1(I_1), S_1, e_2(I_2), S_2, \ldots$, where I_i is time interval.

For example (Fig. 1), if the system contains two active operations O_1 and O_2 they can be ended by the relevant events e_1 and e_2. The event e_1 can occur in the interval $I_1 = (t_i + \alpha_1; t_i + \beta_1)$ and the event e_2 - in the interval $I_2 = (t_i + \alpha_2; t_i + \beta_2)$.

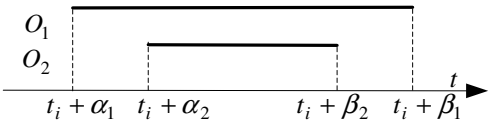

Fig. 1. Time intervals of active operations

In this case there are three time intervals:

1. If an event e_1 will occur at time $t_m \in (t_i + \alpha_1; t_i + \alpha_2)$, then the second event e_2 will occur at time $t_{m+1} \in (t_i + \alpha_2; t_i + \beta_2)$.
2. If an event e_1 will occur at time $t_m \in (t_i + \alpha_2; t_i + \beta_2)$, then the second event e_2 cannot occur before the first. The second event will occur at time $t_{m+1} \in (t_m; t_i + \beta_2)$.
3. If an event e_2 will occur at time $t_m \in (t_i + \alpha_2; t_i + \beta_2)$, then the first event e_1 will occur at the time $t_{m+1} \in (t_m; t_i + \beta_1)$.

There are three possible trajectories of system functioning:

$S_0,\quad e_1(t_i+\alpha_1;t_i+\alpha_2),\quad S_1,\quad e_2(t_i+\alpha_2;t_i+\beta_2),\quad S_2;$

$S_0,\quad e_1(t_i+\alpha_2;t_i+\beta_2),\quad S_1,\quad e_2(t_m;t_i+\beta_2),\quad S_2,$ where $t_i+\alpha_2<t_m<t_i+\beta_2;$

$S_0,\quad e_2(t_i+\alpha_2;t_i+\beta_2),\quad S_1^*,\quad e_1(t_m;t_i+\beta_1),S_2^*,$ where $t_i+\alpha_2<t_m<t_i+\beta_1;$

Graphically this is illustrated in a tree-like structure (Fig. 2).

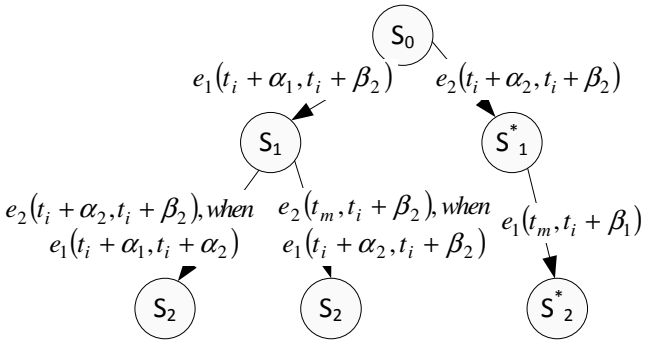

Fig. 2. Tree-like structure of example

4 Reachable State Graph Creation Algorithm

All functioning trajectories must satisfy the following statements. All the statements are proven in [17].

Statement 1. If $w_e(t)$ can take any value in the interval (α,β), then an event e can occur at any time $t_m \in (\alpha,\beta)$.

Statement 2. If the system is at the state s, then the next event e_i will occur at time $t \in (\min\limits_i \alpha_i, \min\limits_i \beta_i)$. According to this definition (Fig. 3) $\alpha = \min\limits_{1\le i\le n} \alpha_i$ and $\beta = \min\limits_{1\le i\le n} \beta_i$.

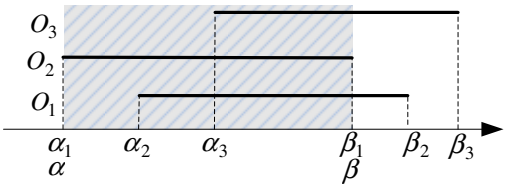

Fig. 3. Graphic depiction of operation ending intervals

Fig. 3 shows that in the interval (α_1,α_2) may finish only operation O_2; in the interval (α_2,α_3) - operations O_1 and O_2; in the interval (α_3,β_1) - operations O_1, O_2 and O_3. In this case $\alpha=\alpha_1$ and $\beta=\beta_1$.

Statement 3. Suppose that in a state s at time t' the operation O_j was active. If at the end of the operation O_i ($i \neq j$) at time t_m operator $H(e_i)$ did not change the continuous component $w_j(t)$, then the system will move to a state where the continuous component $w_j(t)$ satisfies the condition: $\max\{t_m, \alpha_j\} < w_j(t_m) < \beta_j$.

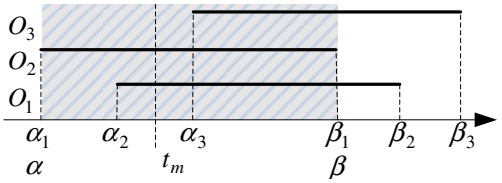

Fig. 4. Active operations range adjustment

Statement 4. The newly generated operation may fall either outside or inside of the relevant range (α, β) (Fig. 5). The earlier mentioned definitions should be evaluated in the both intervals.

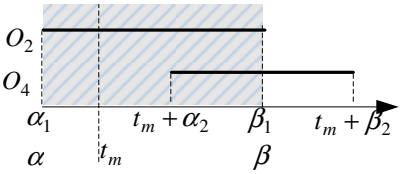

Fig. 5. The newly generated operation falls inside of the relevant interval

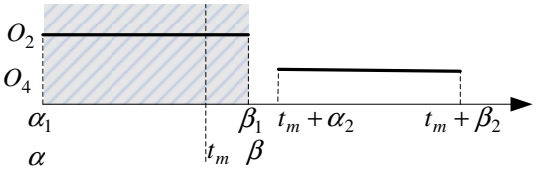

Fig. 6. The newly generated operation falls outside of the relevant interval

According to these definitions, a state graph is formed according to algorithm presented in Fig. 7.

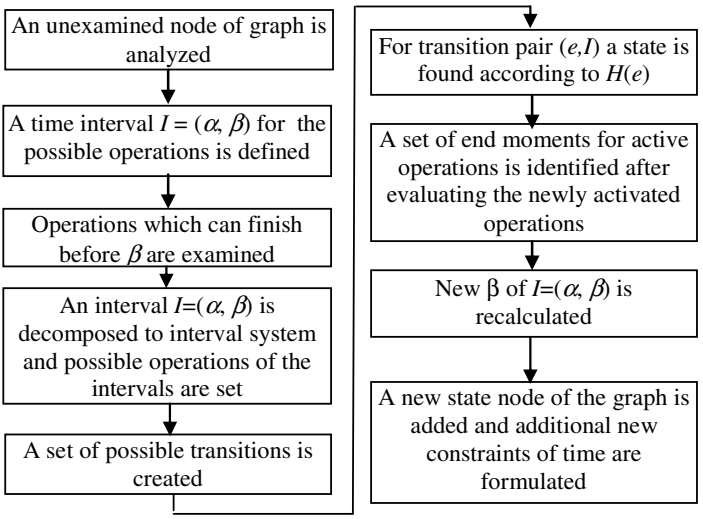

Fig. 7. A flowchart of the state graph analysis algorithm

5 Reachable State Graph Creation Example

A service system consists of one input and two service devices (Fig. 8). Service application messages, arriving to the system, are placed in a queue. When one of the devices becomes available for the service the message is passed to him. If both devices are available, the message is transmitted to the first device.

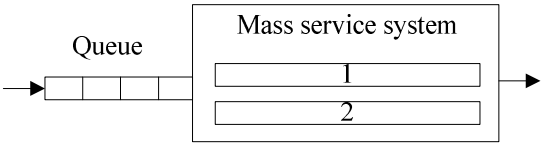

Fig. 8. Two-channel mass service system

The system specification consists of the components:

- a set of inputs $X = \varnothing$ and a set of outputs $Y = \varnothing$;
- a set of events $E = E' \cup E''$, where $E' = \varnothing$; $E'' = \{e_1, e_2, e_3\}$, e_1 - a new message arrived, e_2 - a first channel service is completed, e_3 - a second channel service is finished;
- controlling sequences $e_1 \mapsto \alpha_0, \alpha_1, \alpha_2 \ldots$, $e_2 \mapsto \beta_0, \beta_1, \beta_2 \ldots$, $e_3 \mapsto \gamma_0, \gamma_1, \gamma_2 \ldots$;
- a discrete component $v(t) = (n(t))$, where $(n(t))$ is a number of messages in a queue.
- a continuous component $z_v(t) = (w(e_1,t), w(e_2,t), w(e_3,t))$;
- a parameter s - is a maximum length of the queue.

- time limitations on the duration of operations are these: $4 < \alpha_i < 6$, $3 < \beta_i < 5$, $2 < \varphi_i < 4$, $\forall i = 1,2,\ldots$.

Transition operators are as follows:

$H(e_1)$:

$$n(t_m) = \begin{cases} n(t_m - 0) + 1, & t_m < w(e_2, t_m - 0) \wedge t_m < w(e_3, t_m - 0) \wedge n(t_m - 0) < s; \\ 0, & otherwise; \end{cases}$$

$$w(e_1, t_m) = t_m + \alpha_m;$$

$$w(e_2, t_m) = \begin{cases} t_m + \beta_m, & t_m > w(e_2, t_m - 0); \\ w(e_2, t_m - 0), & otherwise; \end{cases}$$

$$w(e_3, t_m) = \begin{cases} t_m + \gamma_m, & t_m < w(e_2, t_m - 0) \wedge t_m > w(e_3, t_m - 0); \\ w(e_3, t_m - 0), & otherwise; \end{cases}$$

$H(e_2)$:

$$n(t_m) = \begin{cases} n(t_m - 0) - 1, & n(t_m - 0) > 0; \\ 0, & otherwise; \end{cases}$$

$$w(e_2, t_m) = \begin{cases} t_m + \beta_m, & n(t_m - 0) > 0; \\ w(e_2, t_m - 0), & otherwise; \end{cases}$$

$H(e_3)$:

$$n(t_m) = \begin{cases} n(t_m - 0) - 1, & n(t_m - 0) > 0; \\ 0, & otherwise; \end{cases}$$

$$w(e_3, t_m) = \begin{cases} t_m + \gamma_m, & n(t_m - 0) > 0; \\ w(e_3, t_m - 0), & otherwise; \end{cases}$$

A generation of a reachable state graph is carried out in accordance to the algorithm presented in Fig. 7.

Step 1. The generation of the reachable state graph starts from the initial state. The state S consists of three components: a discrete component $\nu(t)$, a continuous component $z_\nu(t)$ and a set of time constraints R:

$:1:(0;(t_0+4, t_0+6), \varnothing, \varnothing; R_0)$, where $R_0 = \varnothing$.

The first interval $I = (\alpha, \beta)$ is defined according to formulas $\alpha = \min\{t_0 + 4\} = t_0 + 4$ and $\beta = \min\{t_0 + 6\} = t_0 + 6$ (Fig. 9). Operations, which may finish in the interval first of all, are found. According to PLA specification only one operation O_1 is active. Since there is only one operation, using a transition operator $H(e_1)$ we find the next state: $(0;(t_1+4, t_1+6), (t_1+3, t_1+5), \varnothing)$.

Check if the new activated operation will not end earlier than $\beta = t_0 + 6$. Since the condition is satisfied $\min\{t_0 + 6, t_1 + 3\} = t_0 + 6 = \beta$, a new activated operation can not finish before the examined interval. The next state is as follows:

$2:(0;(t_1+4, t_1+6), (t_1+3, t_1+5), \varnothing; R_{11})$, where $R_{11} = R_0 \cup \{t_0 + 4 < t_1 < t_0 + 6\}$.

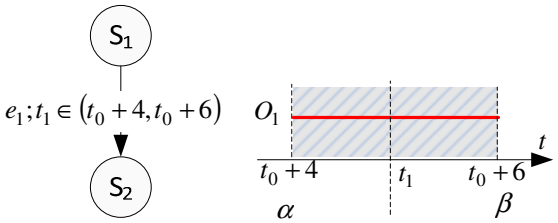

Fig. 9. Reachable state graph fragment $S_1 \to S_2$ and a transition e_1, t_1

Step 2. The next interval $I = (\alpha, \beta)$ is defined by formulas $\alpha = \min\{t_1+3, t_1+4\} = t_1+3$ and $\beta = \min\{t_1+5, t_1+6\} = t_1+5$.

Operations which may finish in the interval first of all, are found. They are two operations (O_1 and O_2). The interval $I = (\alpha, \beta)$ is separated (Fig. 10) into two intervals $\{t_1+3, t_1+4\}$ and $\{t_1+4, t_1+5\}$.

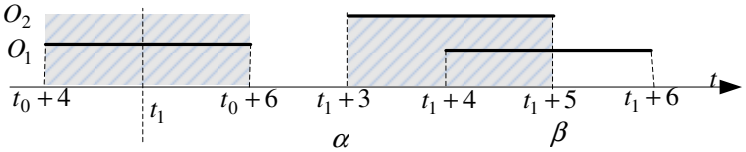

Fig. 10. Separated two intervals $\{t_1+3, t_1+4\}$ and $\{t_1+4, t_1+5\}$

The possible transitions there are three:
$(e_2, t_2 \in (t_1+3; t_1+4))$, $(e_2, t_2 \in (t_1+4; t_1+5))$, $(e_1, t_2 \in (t_1+4; t_1+5))$

Step 2.1. A transition $(e_2, t_2 \in (t_1+3; t_1+4))$ is analyzed first of all (Fig. 11).

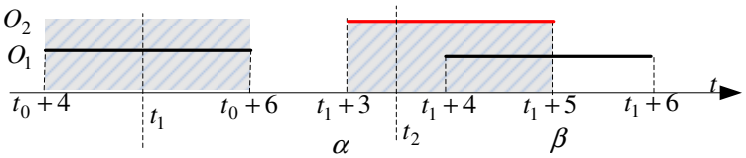

Fig. 11. An event $(e_2, t_2 \in (t_1+3; t_1+4))$

Using a transition operator $H(e_2)$ the next state is defined: $(0; (t_1+4, t_1+6), \varnothing, \varnothing)$. Since the operation O_1 after the event remained active, we have to recalculate the end of the interval in such a way: $(\max\{t_2, t_1+4\}, t_1+6) = (t_1+4, t_1+6)$.

The third state is as follows (Fig. 12):
$3: (0; (t_1+4, t_1+6), \varnothing, \varnothing; R_{21})$, were $R_{21} = R_{11} \cup \{t_1+3 < t_2 < t_1+4\}$

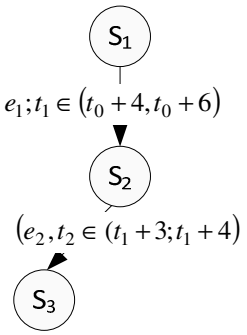

Fig. 12. A fragment of the reachable state graph (a transition $S_2 \to S_3$)

Step 2.2. The second transition $(e_2, t_2 \in (t_1 + 4; t_1 + 5)$ is analyzed next (Fig. 13).

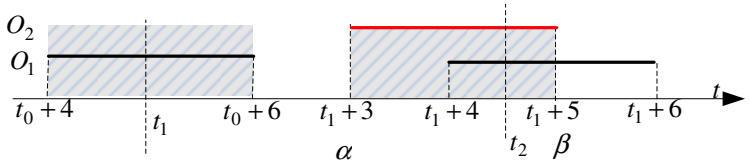

Fig. 13. An event $(e_2, t_2 \in (t_1 + 4; t_1 + 5)$

Using transition a transition operator $H(e_2)$ the next state is defined: $(0; (t_1 + 4, t_1 + 6), \varnothing, \varnothing)$. Since an operation O_1 after the event remained active, we have to recalculate the end of the interval in such a way: $(\max\{t_2, t_1 + 4\}, t_1 + 6) = (t_2, t_1 + 6)$.

The forth state is as follows (Fig. 14):

$4: (0; (t_2, t_1 + 6), \varnothing, \varnothing; R_{22})$, were $R_{22} = R_{11} \cup \{t_1 + 4 < t_2 < t_1 + 5\}$

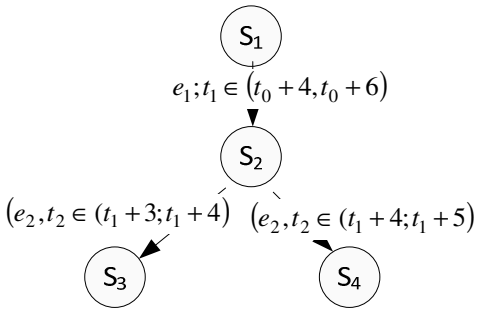

Fig. 14. A fragment of the reachable state graph (a transition $S_2 \to S_4$)

Step 2.3. The third transition $(e_1, t_2 \in (t_1 + 4; t_1 + 5)$ is analyzed next (Fig. 15).

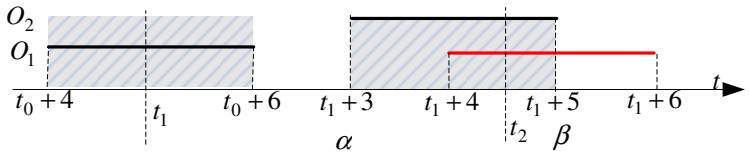

Fig. 15. An event $(e_1, t_2 \in (t_1+4; t_1+5)$

Using the transition operator $H(e_1)$, the next state is defined: $(0; (t_2+4, t_2+6), (t_1+3, t_1+5), (t_2+2, t_2+4))$.

Since an operation O_2 after the event remained active, we have to recalculate the end of the interval in such a way: $(\max\{t_2, t_1+3\}, t_1+5) = (t_2, t_1+5)$. The forth state is as follows (Fig. 16):

$5: (0; (t_2+4, t_2+6), (t_2, t_1+5), (t_2+2, t_2+4); R_{23})$, where $R_{23} = R_{11} \cup \{t_1+4 < t_2 < t_1+5\}$.

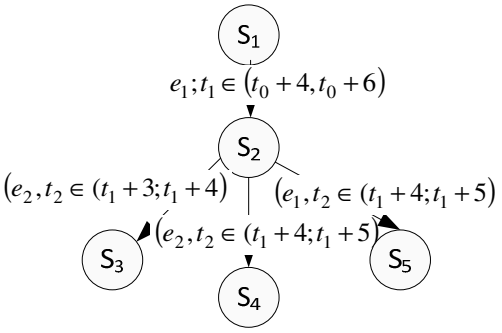

Fig. 16. A fragment of the reachable state graph (a transition $S_2 \rightarrow S_5$)

6 Conclusions

Conventional verification methods do not perform a full analysis of real-time systems as the traditional verification methods underestimate the system performance over time, or analyze only system whose operation time is deterministic. However, many operations of real-time systems depend to a certain interval and may result in any precisely specified time interval.

This paper presents a novel approach for creation of a reachable state graph. While creating the reachable state graph an algorithm is used. The algorithm permits to evaluate intervals of time when the defined system events occur. When the reachable state graph is made then various properties can be verified: dead ends, inefficient cycles, reachability and so on.

Acknowledgements. The work described in this paper has been carried out within the framework the Operational Programme for the Development of Human Resources

2007-2013 of Lithuania „Strengthening of capacities of researchers and scientists“ project VP1-3.1-ŠMM-08-K-01-018 „Research and development of Internet technologies and their infrastructure for smart environments of things and services“ (2012- 2015), funded by the European Social Fund (ESF).

References

1. Aceto, L., Bouyer, P., Burgueno, A., Larsen, K.G.: The power of reachability testing for timed automata. Theoretical Computer Science 300, 411–475 (2003)
2. Bonhomme, P.: Scheduling and control of real-time systems based on a token player approach. Discrete Event Dynamic Systems 23, 197–209 (2013)
3. Buslenko, N.P., Kalashnikov, V.V., Kovalenko, I.N.: Lectures on the Theory of Complex Systems. Sov. Radio, Moscow (1973) (in Russian)
4. Dang, Z., Ibarra, O.H., Kemmerer, R.A.: Generalized discrete timed automata: decidable approximations for safety verification. Theoretical Computer Science 296, 59–74 (2003)
5. David, R., Alla, H.: On hybrid Petri nets. Discrete Event Dynamic Systems 11, 9–40 (2001)
6. Ding, Z., Jiang, C., Zhou, M.: Design, Analysis and Verification of Real-Time Systems Based on Time Petri Net Refinement. ACM Transactions on Embedded Computing Systems (TECS) 12 (2013)
7. Ghomri, L., Alla, H.: Modeling and analysis using hybrid Petri nets. Nonlinear Analysis: Hybrid Systems 1, 141–153 (2007)
8. Gómez, R.: Model-checking timed automata with deadlines with Uppaal. Formal Aspects of Computing 25, 289–318 (2013)
9. Halbwachs, N., Proy, Y.E., Roumanoff, P.: Verification of real-time systems using linear relation analysis. Formal Methods in System Design 11, 157–185 (1997)
10. Knorreck, D., Apvrille, L., Pacalet, R.: Formal system-level design space exploration. Concurrency and Computation: Practice and Experience 25, 250–264 (2013)
11. Kopetz, H.: Real-time systems: design principles for distributed embedded applications. Springer Science+ Business Media (2011)
12. Krena, B., Vojnar, T.: Automated formal analysis and verification: an overview. International Journal of General Systems 42, 335–365 (2013)
13. Laplante, P.A.: Real-Time Systems Design and Analysis. Wiley-IEEE Press (2004)
14. Lorin, H., Deitel, H.M.: Operating Systems. Longman Higher Education (2009)
15. Mekki, A., Ghazel, M., Toguyeni, A.: Validation of a New Functional Design of Automatic Protection Systems at Level Crossings with Model-Checking Techniques. IEEE Transactions Intelligent Transportation Systems 13, 714–723 (2012)
16. Pranevicius, H.: Complex systems formalization and analysis (in Lithuanian). Technologija, Kaunas (2008)
17. Pranevicius, H., Raudys, S., Rudzionis, A., Ratkevicius, K., Sakalauskaite, J., Makackas, D.: Agent system models. Mokslo aidai, Vilnius (2008)
18. Pranevicius, H., Miseviciene, R.: Verification of piece-linear aggregate specifications. Kaunas, Technologija (2006)
19. Renganathan, K., Bhaskar, V.: Performance evaluation and model checking in systems modeled as Hybrid Petri nets. Applied Mathematical Modelling 36, 3941–3947 (2012)
20. Saadawi, H., Wainer, G.: Principles of Discrete Event System Specification model verification. Simulation 89, 41–67 (2013)
21. Yin, Y., Liu, B., Ni, H.: Real-time embedded software testing method based on extended finite state machine. Systems Engineering and Electronics 23, 276–285 (2012)