INFORMATICA, 2004, Vol. 15, No. 1, 45–62

The Evaluation and Design Methodology for Real Time Systems

Egidijus KAZANAVIČIUS
Computer Department Kaunas University of Technology
Studentu 50–214c, LT-51368 Kaunas, Lithuania
e-mail: ekaza@ifko.ktu.lt

Received: August 2003

Abstract. Petri net variants are widely used as a real time systems modeling technique. Recently, UML activity diagrams have been used for the same purpose, even though the syntax and semantics of activity diagrams has not been yet fully worked out. Nevertheless, activity diagrams seem very similar to Petri net semantics. UML, being the industry standard as a common object oriented modeling language needs a well-defined semantic base for its notation. Formalization of the graphical notation enables automated processing and analysis tasks. Petri nets can provide a formal semantic framework for the UML notations plus the behavioral modeling/analysis strength needed to system designers. This paper describes the methodology for creating the model of the RT application that would allow testing the correctness of the algorithm and the fulfillment of the time constraints at the design stage using UML and Petri Nets.

Key words: real time system, unified modelling language, Petri net behaviour, verification.

1. Introduction

Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced. Real-time systems span a broad spectrum of complexity from very simple microcontrollers (such as a microprocessor controlling an automobile cruise control system) to highly sophisticated, complex and distributed systems (such as air traffic control for the Europe Union) (Stankovic and Ramamritham, 1998). Typically, a real-time system consists of a controlling system and a controlled system. For example, in an automobile cruise control system, the controlled system is the automobile with all its parts, while the controlling system is the controller and human interfaces that manage and coordinate the running of the automobile. Thus, the controlled system can be viewed as the environment with which the computer interacts.

The controlling system interacts with its environment based on the information available about the environment from various sensors. It is imperative that the state of the environment, as perceived by the controlling system, be consistent with the actual state of the environment. Otherwise, the effects of the controlling systems activities may be disastrous. Hence, periodic monitoring of the environment as well as timely processing of the sensed information is necessary.

Timing correctness requirements in a real-time system arise because of the physical impact of the controlling systems, activities upon its environment. In many real-time systems even more severe consequences will result if timing as well as logical correctness properties of the system are not satisfied, e.g., consider nuclear power plants or air traffic control systems failing or ultrasonic measurement systems (Kazanavicius *et al.*, 2004; Kazanavicius *et al.*, 2004a). Timing constraints for tasks can be arbitrarily complicated but the most common timing constraints for tasks are either periodic or aperiodic.

Model-based development is the best way to make an efficient application in today's high-complexity, short-development-cycle business environment. It is important to focus on the abstractions of the problem rather than on the low-level details of its implementation. The use of Unified Modelling Language (UML) for system designing is rapidly gaining attention from the industry. It allows capturing user requirement and system view of the application using single unified notation. However the early stages of the real-time development process involve some very specific tasks, beside the usual analysis and design stages. Examples include scheduling analysis, performance evaluation and formal verification of critical timing properties of the system. Due to public request, UML now is expanding in such a way that it allows to do specific real-time development tasks such as: timelines requirements capturing and scheduling analysis, behaviour an performance analysis. This expanded notation sometimes is called Real-Time UML (RT-UML). However it still does not address formal verification and performance evaluation problems.

On the other hand formal Petri Nets (PNs) methods are well suited for modelling and analysis of systems. They are also similar to UML in that they have a convenient and effective graphical language for their visualization. However, most real-time systems are sometimes subsystems of complex systems, including large computer systems.

The UML is user-friendly, easy to use and is useful for describing system effectively. On the other hand Petri Nets are formal methods, which allow strict analysis, verification, evaluation and simulation of the system.

So here we combine advantages of the UML and PNs and propose a UML-PNs integrated modelling method for real-time systems.

2. UML and Real-Time Systems

A. Problem Statement and UML

A typical real time system design project proceeds using the ROPES or the similar, process. In such a process, work begins with the identification of the use cases, actors and scenarios. The project proceeds with the definition of classes, objects and their relations. Some objects are reactive so their behaviour can be defined using activity diagrams. The formal method is needed that would allow to test the correctness of the timing constraints in the design stage.

The Unified Modelling Language (UML) is a language for expressing the constructs and relationships of complex systems. It was begun as a response to the Object Management Group's (OMG, 1999) request for proposal for a standard object-oriented methodology.

The UML is more complex than other methods in its support for modelling complex systems. It is particularly suited for modelling real-time, embedded systems (Selic and Rumbaugh, 1998). Major features include object model, use cases and scenarios, behavioural modelling with state charts, packaging of various kinds of entities, representation of tasking and task synchronization, models of physical topology, models of source code organization, support for object-oriented patterns.

The early stages of the real-time development process involve some very specific tasks, beside the usual analysis and design stages. Examples include: scheduling analysis, performance evaluation and formal verification of critical timing properties of the system. It is quit common to model real-time systems using *state diagrams*. A state diagram provide a static view of the entire state space of a system, however it doesn't provide means for specifying timing constraints. The most common way to capture timelines requirements is to use *sequence diagrams*. Sequence diagrams allow showing typical system behaviour paths, called *scenarios*.

Scenarios may not visit all states in the system nor activate all transitions, but they provide an order-dependent view of how the system is expected to behave when actually used. So, sequence diagrams almost always show the combined behaviour of a collaboration of objects working to achieve a common purpose, such as the realization of a use case. But it is not suited to capture overall system behaviour.

For this purpose *activity diagrams* are better choice. UML considers activity diagrams to be a kind of state machine in witch most or all the transitions are taken when the activity completes rather than wait for an external asynchronous event. Activity diagrams excellent at showing procedural flow and, in fact, show concurrency in a more natural and obvious way than state charts (Douglas, 1999). This is why activity diagrams are well suited to capture algorithmic flow. In addition activity diagrams allow specifying timing constraints as transitions guards. Asynchronous control can be specified using signal reception and signal sending states. Multitasking is also an integral part of activity diagrams. It is controlled using forks and joins. All this allow to specify real-time systems in details using activity diagrams.

Performance and dependability evaluation is an important step in the development of most real-time systems. Traditionally, performance and dependability evaluation has used a separate set of models, disregarding functional models that serve as basis for other steps of system development. Traditional models for performance analysis are queuing networks and stochastic processes. Recently, academic studies have concentrated on model-based performance evaluation, which proposes the use of the same models as in functional description, possibly with stochastic extensions. Existing results on model-based performance evaluation concern formalisms like: Petri nets (stochastic Petri nets), process algebra, but also more high-level functional specification formalisms such as SDL (Bozga *et al.*, 2001).

In particular, for formally verifying the timing properties of an UML specification, a parallel specification in formalism such as timed automata, timed Petri nets or SDL, may be necessary (Bozga *et al.*, 2001; Pranevicius and Budnikas, 2003).

The problem is, that these formal models have different concept than UML models, so system analysts have to put much effort and additional work to support UML model

and some formal model in parallel. This is why we try to define method how to get formal Petri net model from already defined UML model.

B. Real-Time System Modelling Tools

Nowadays modelling automation tools is one of the prerequisites for modelling notations to gain popularity. Some tools just provide possibility to draw diagrams, some also allow checking the model correctness and others allow testing the model behaviour. There are quite a number of powerful tools that allow drawing UML diagrams. Regarding the modelling of timed Petri nets this number is lower, but this is understandable, because UML has more common use and timed Petri nets are primarily used for modelling real time systems. *No Magic MagicDraw UML* (MagicDraw Tool, 2003) is a visual UML modelling tool. It provides the code engineering mechanism (with full round-trip support for Java, C#, C++, and CORBA IDL programming languages), as well as database schema modelling and reverse engineering facilities. *Rational Rose* (Rose Case Tool, 2003) is a big development tools family, which includes tools for project management, UML modelling, real-time systems modelling and others. *I-Logix Rhapsody* (ILogix Tool, 2003) is a UML-compliant design automation tool used in the development of real-time and embedded systems. *ArgoUML* (Argo UML Tool, 2003) is Java based, open source project running under BSD License. It allows designing fully featured UML diagrams and (what is quite uncommon for UML designing tool) has a module for designing Petri nets. It also saves data in XML format, so it can be easily processed or/and exported to other tools. It also has support for plug-in modules, which makes it easy to add additional functionality to this tool.

For our tests we have used ArgoUML tool extended with our modules for mapping activity diagrams to PN model and it's modelling tool. The main advantages of this tool are that it is open source, free and have support for Petri Nets. It also allows to plug-in additional modules, so was added additional functionality needed for our approach and tests.

3. Methodology

A. Petri-Net Formal Method for RTS Modelling

Effective modelling of complex concurrent RTS requires a formalism that can capture essential properties such nondeterminism, synchronization and parallelism. Petri nets provide for a mathematical approach to behaviour and analysis of systems, with a convenient, effective and highly intuitive graphical language for their visualization.

Petri nets cover a range of diverse applications, including communication protocols, computer networks, manufacturing systems, industrial process control and data flow computing. Similarly, they have been widely used in the study of behavioural properties as well as in areas such as simulations, performance evaluation and fault tolerance (Nisanake, 1997).

The Petri nets model: captures explicitly timing information; it is more expressive as tokens might carry information; systems may be represented at different levels of granularity; both control and data information may be captured by a unified design representation.

A RTS model is 7-tuple $N = (P, T, I, O, M_0, t, t2)$ where P is a set of places, T is a set of transitions, I is a set of input (place-transition) arcs, O is a set of output (transition-place), and M_0 is the initial marking of the net, t: $T \Rightarrow t \in R_0^+$ is mean time associated to transition, $t2$: $T \Rightarrow t \in R_0^+$ second time parameter (variance). A marking is an assignment of tokens to the places of the net. A token is a pair $k = \langle v, r \rangle$ where v is the token value (may be of any type) and r is the token time. Thus tokens carry time information attached to them stamps. The token type associated to a place p, denoted $\tau(p)$, is the type of value that a token may bear in p. A transition $t \in T$ may have a guard G, a condition that must be satisfied in order to enable the transition when all its input places hold tokens. For every transition $t \in T$, there exist a minimum transition delay d and maximum transition delay D. The non-negative real numbers $d \leqslant D$ represent the lower and upper bounds for the execution time of the activity function. Transition delays give the limits in time for the firing of the a transition since it becomes enabled, uless it is disabled by firing another transition.

B. Abstraction of RTS and Timing Analysis

The simplest form of abstraction of real time reduces it to an ordering, possibly a partial ordering, of events in real time according to their temporal precedence. The precedence relation may sometimes be implicit, as in the case of inputs and outputs, where it is taken for granted that that outputs follow inputs. Systems that are based on the notions of input and output are referred to as transformational systems. Their specification may be given as an input-output relation. Conformity of the actual behaviour with the specified input-output relationship is understood as logically correct behaviour. One implication of this view is that the computations concerned terminate; otherwise the correctness has to be treated as contributing only to a partial understanding.

In situations where the real time events may not be seen simply as inputs and outputs, but as more general complex patterns of interactions of the system with its environment, the temporal precedence relation is explicitly recognized, if not explicitly stated. Systems founded on events and interactions correspond to what are referred to as reactive systems. Since the interactions are typically non-terminating, the correctness has to be investigated by using more elaborate models than those employed in transformational systems.

In our methodology we will analyse reactive systems – the systems that are founded on events. For such systems many timing questions can be phrased in a particular framework, namely as an analysis of the extreme case separation in time between two system events. Then the problem of determining the extreme case separation in time between two events e_a and e_b is to determine the tightest bounds δ and Δ such that for all executions $\delta \leqslant \tau(e_b) - \tau(e_a) \leqslant \Delta$, where $\tau(e_{\mathrm{from}})$ and $\tau(e_{\mathrm{to}})$ represent the time of events e_a and e_b, respectively. That is δ and Δ are the smallest and largest separation between e_a and e_b, respectively.

Timing analysis of a real-time system answers questions like “How long may an event have to wait have to wait before it will be enabled?” or “How much time may pass between two system events?” Stochastic analysis, where delays are specified using probability distributions answers such questions with average case numbers in steady state execution. This information is important for performance evaluation for example to determine average utilization and average response time of the components of the system. However stochastic analysis methods are inappropriate for verifying correct operation. Instead delays are modelled using ranges rather than probability distributions and the timing analysis we consider provides extreme case rather than average case numbers.

C. Translating UML Activity Diagrams into PN

The global scheme of the UML – PNs integrated modelling method can be described as follows: the UML is utilized for requirement description, model specification and design. Then, the UML model is mapped to the PN model for model analysis and simulation, results of which can be fed back to the UML model design. Such process can be executed in an iterative way to achieve continued model improving. Lastly, model implementation is carried out according to the improved UML model (Fig. 1).

The development process using UML is discussed in details by many authors (Hurby, 1999), so we will concentrate on UML to PN mapping here.

We propose the methodology for RT UML mapping into Petri net.

This methodology is considered to assist RTS designers in supporting UML specification and formalism in Petri net in parallel. In the first step designer must design the system in UML. Following step is to describe the behaviour of the system using UML activity diagram. And last step is to map activity diagram to Petri net.

Then mapping from activity diagram to Petri net can be done as shown in Table 1.

Timing constraints in activity diagram can be specified in states using our proposed specific notation.

Such activity diagram already is essentially Petri net with the constraint that each activity state has a maximal capacity of a single token. The basic idea of mapping is to change every activity diagram state into Petri net place and every transition into Petri net transition. The initial marking of the net is determined by initial state of the activity diagram (Agerwala, 1979).

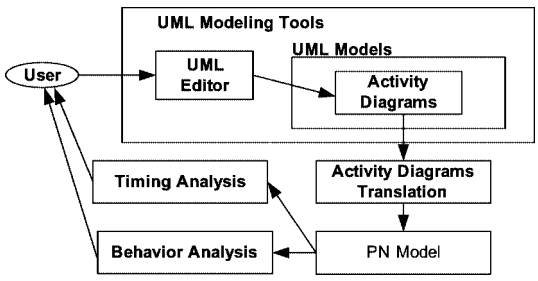
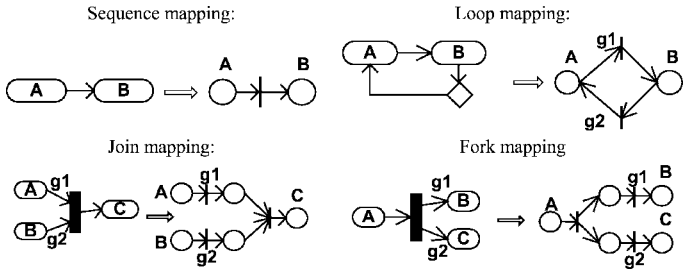


Fig. 1. Block diagram of the proposed methodology.

Table 1

Activity diagram to Petri net mapping scenarios



As mapping result we get *safe* timed Petri Net. A safe Petri net $\Sigma = (S, T, F, M_0)$ is such that in every execution has at most one token on each place (Murata, 1989):

$$\forall s \in S, \quad \forall M \in [M_0] : M(s) \leqslant 1.$$

The similar constraint was applied to initial activity diagram, so we can state that activity diagram in our case is essentially *safe* Petri net.

Safe Petri nets have a number of useful properties and restricting our model to safe nets greatly simplifies the execution semantics based on processes. The main reason for this is that a safe net has a finite number of reachable markings, and only one token can be assigned for a single place.

4. Experimental Results

This section presents a case study of real time temperature measurement system which we use in order to illustrate our proposed methodology and the improvement techniques. As mentioned earlier, our goal is to define a process for generation of Petri nets from UML specifications. More specifically, we focus here on two key experiments: 1) generation of Petri net models from UML activity diagrams, 2) description of algorithm and timing correctness verification (Experiment 1) and 3) our approach for the behavioural modelling and analysis (Experiment 2).

A. Temperature Measurement System: a Case Study

As an example of Real-Time system we take simple temperature measurement system.

Temperature sensor device measures the temperature. Measured and sampled value goes through ADC, where analogue signal is converted to digital. After this measured value put through communication Txd port to FIFO buffer, which length for the simplicity we assume is one. Data from buffer goes to data processor, where it is checked for suitability. In non-critical temperature case measured value is written to the database; while in critical temperature case in addition to storing the measured value to the database some actions are taken to cope with the situation and an error message is send to the operator display.

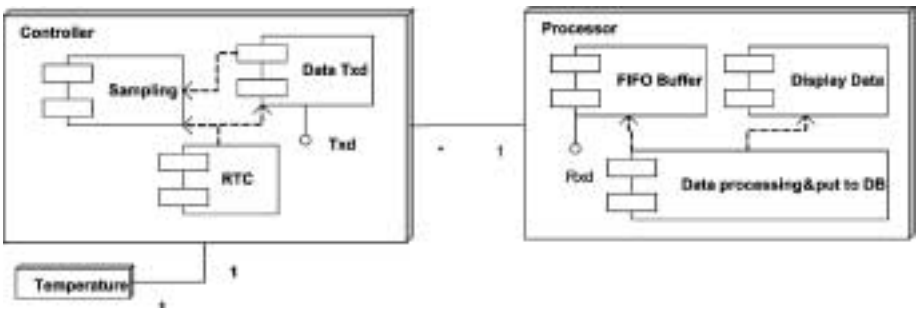


Fig. 2. Temperature measurement system – implementation diagram.

In there is an example of initial activity diagram, which models simple temperature measurement system. In $s2$ state the temperature value is read, then in $s3$ it is pushed to buffer. $s4$ state takes the value from buffer. The depending on the temperature value $s6$ (for non critical) or $s7$ (for critical) processes the value. Lastly $s5$ state does some finishing tasks.

Time ranges are presented for every state, except initial state, which is always [0, 0]. The data flow state shows what values should be taken during the automatic evaluation.

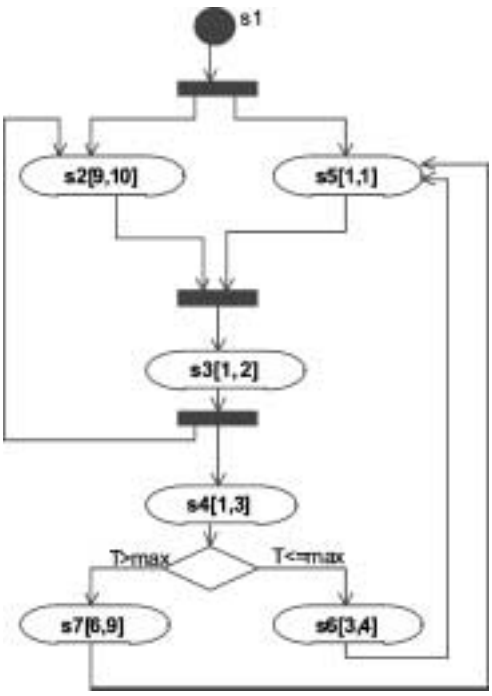


Fig. 3. Temperature measurement system – activity diagram.

B. Verification of RTS Using Timed Petri Net

A timed Petri net is a formal model of Real Time Systems that has a simple representation of concurrency and synchronization. It includes cycles and conditions that makes the task of execution time verification complicated. Therefore beside Timed Petri nets are defined process graphs of the net. A process graph is an acyclic and condition free net that can be thought of as an unfolding of a Petri net representing a particular execution. Processes are used to give semantics to Petri nets. Their key advantage is that concurrency is represented explicitly rather than implicitly. The possible execution of a Petri net is defined as a set of process graphs. The set of executions is represented using a finite directed graph called the process automation whose edges are annotated with processes and whose nodes represent states of the system.

Fig. 4 contains example of a Petri net with cyclic (places $s2$ and $s5$) and conditional (transitions $t4$ and $t6$) behaviour. The possible given Petri net unfolding is shown in Fig. 4, which contains two cycles of the Petri net execution and behaviour with $t6$ conditional transition enabled.

We assume that the set of all possible processes graphs of the Timed Petri net can be obtained and the timing analysis can be done using the set of process graphs that unfold the Petri net. After the UML diagram is converted to a Petri net, we have a safe, timed Petri net, where delay ranges d and D are associated with places. Delay ranges d

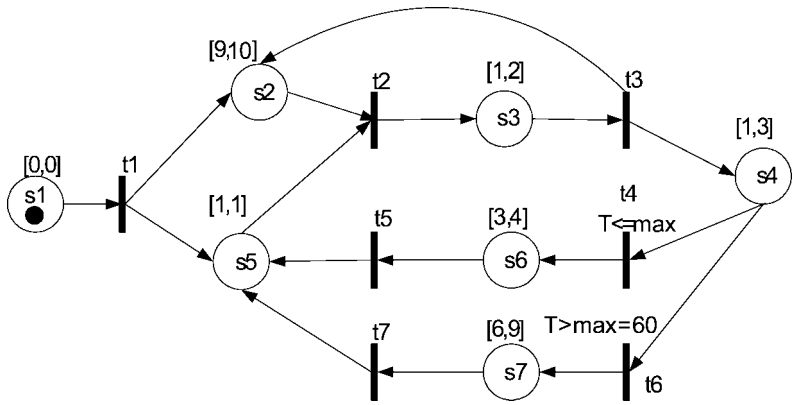


Fig. 4. Petri net, obtained from activity diagram in Fig. 2.

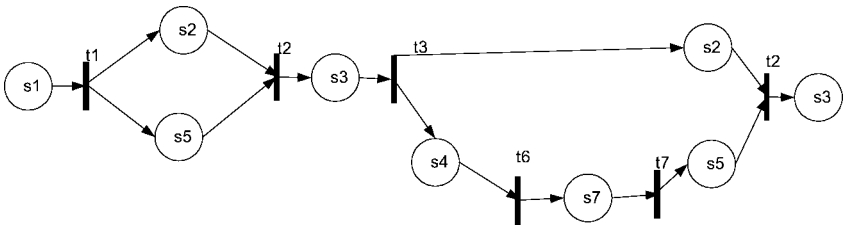


Fig. 5. Process graph, obtained from Petri net in Fig. 3.

and D represent a delay range for each place. Graphically, a place s is annotated with the interval $[d(s), D(s)]$ or just $d(s)$ when $d(s) = D(s)$. The delay bounds restrict the possible executions of the Petri net. During a timed execution of the net, when a token is added to a place s, the earliest it becomes available for a transition after s is $d(s)$ time units later and the latest is $D(s)$ time units later. A transition t must fire when there are available tokens at all places before t unless the firing of the transition disabled by firing another transition. The firing of t is instantaneous.

For all process graphs p from P and for all consistent timing assignments τ for p, determine the largest δ and smallest Δ such that:

$$\delta \leqslant \tau(s_{\mathrm{to}}) - \tau(s_{\mathrm{from}}) \leqslant \Delta.$$

The two numbers δ and Δ are the minimum and maximum separation between the transitions t_{from} ans t_{to} with the relationship specified by Timed Petri net. The most important is the maximum separation analysis Δ, because the minimum separation δ can by found from a maximum separation analysis of

$$\tau(s_{\mathrm{from}}) - \tau(s_{\mathrm{to}}) \leqslant -\delta.$$

That is, to determine the minimum separation between t_{from} and t_{to}, the maximum separation between t_{to} and t_{from} is determined (which may be negative) and the result is negated. The timing analysis analyzes two infinite sets: the set of element p in P, for each p, the set of consistent timing assignment for p. This leads to a natural decomposition of the maximum separation problem. If $\Delta(p)$ is the maximum separation for a particular p in P, i.e., Δ(p) is the maximum separation between s_{from} and s_{to} in the process p:

$$\Delta(p) = \max\{\tau(s_{\mathrm{to}}) - \tau(s_{\mathrm{from}}) \mid \tau \text{ is a consistent timing assignment for } p\}.$$

If a process graph p has no consistent timing assignments the $\Delta(p)$ will be defined $-\infty$. The maximum separation is determined by:

$$\Delta = \max\{\Delta(p) \mid p \in P\}.$$

Maximum and Minimum Timing Separation of Events (TSE) Algorithm

The process graph p is particular element from overall set of process graphs P for a Timed Petri net. The maximum separation problem for p is to determine the smallest number $\Delta(p)$ such that:

$$\tau(s_{\mathrm{to}}) - \tau(s_{\mathrm{from}}) \leqslant \Delta(p).$$

The TSE algorithm computes two values for each place s in the process graph, denoted $m(s)$ and $M(s)$. The algorithm consists of two phases, computing $m(s)$ in the first phase and $M(s)$ in the second. For convenience is used notation $\max_{\bullet s}\{\cdot\}$to denote

a maximization over all incoming events to event s in a process (the dot $\cdot$ denotes an arbitrary expression referring to s, s', and b):

$$\max_{\bullet s}\{\cdot\} = \begin{cases} \max\{\cdot \mid b \in \bullet s, \{e'\} = \bullet b\}, & \text{if } \bullet e \neq 0 \text{ and } \bullet b \neq 0, \\ -\infty, & \text{otherwise.} \end{cases}$$

From the definition of a process $\mid \bullet b \mid \leqslant 1$, and thus there is no ambiguity with respect to what s' event to use.

The value of $m(s)$ is the length of the longest path from an event s to the event e_{from} using the lower delay of bounds of the places:

$$m(s) = \max\{d(\rho) \mid \text{all paths } s \rightarrow s_{\text{from}},$$

where $d(\rho)$ denotes the sum of the lower delay bounds of the conditions of path ρ :

$$d(\rho) = \Sigma\{d(l(b)) \mid b \text{ is a condition on path } \rho\}.$$

The intuitive meaning of m-value for an event s is that it determines the minimum difference between the timing assignment of s and s_{from}.

If there is no path from an event s to s_{from}, denoted by $s \neq s_{\text{from}}$, the arbitrary constant to $m(s)$ can be assigned. Normally, the $m(s) = 0$ can be used, although it sometimes id advantageous to choose different constants for the various event e. The $m(s)$ can be computed by reversing topological traversal (p is acyclic) starting from s_{from}.

The value $M(s)$ is computed in the second phase for all events. The value for $M(e)$ determines how much an event s can be moved forward in time without changing the timing assignment for the event $s_{\text{from}}.M(s_{\text{root}}) = 0$ and then for all other events in (normal) topological order:

$$M(s) = \max_{\bullet s}\{X(s', b, s)\},$$

where:

$$X(s', b, s) = \begin{cases} \min(0, M(s') + D(l(b)) - m(e') + m(s)), & \text{if } s \rightarrow s_{\text{from}} \\ M(s' + D(l(b)) - m(s') + m(s)), & \text{if } s \mapsto s_{\text{from}}. \end{cases}$$

The maximum separation between s_{from} and s_{to} is determined from:

$$\Delta(p) = M(s_{\text{to}}) - m(s_{\text{to}}).$$

The TSE algorithm is shown in Fig. 6. This algorithm consists of two parts. In the first part (line 1 to 7), processes are constructed of increasing length and Δk is computed for each of them. The iteration stops either when a lower or upper bounds on Δ converge or when the (cyclic) system enters a repetitive pattern. In the second part (line 11–12), the matrices $R,\ S,\ T$ are constructed and the matrix closure of S is used to analyze the infinite number of further unfolding. By minimizing this value with the Δ-value obtained from the previous unfolding, the exact minimum separation is obtained.

1. $k \leftarrow 0$
2. $\Delta \leftarrow \infty$
3. **do**
4. $\Delta k \leftarrow \Delta(\pi_R(\pi_S)^k \pi_\tau)$
5. $\Delta \leftarrow \max(\Delta, \Delta_k)$
6. $k \leftarrow k + 1$
7. **until** termination-condition or repeating
8. **if** terminated **then**
9. **return** Δ
10. **else**
11. $\Delta_{\geqslant k} \leftarrow RS * T$
12. **return** $\max(\Delta, \Delta_{\geqslant k})$

Fig. 6. TSE algorithm.

C. The Correctness Verification

In stochastic analysis, where delays are specified by probability distributions, algorithms define occasions with average case numbers in steady state execution. This information is important for performance evaluation for example to determine average utilization and average response time of the components of the system. However, stochastic analysis is not appropriate to determine correctness of the system. In such a case it is better to determine the extreme case separation in time between two events. For the Timed Petri Net, generated from UML (see Fig. 4) we can calculate the lower and upper separation between two transitions or between recurrences of the same transition:

$$\delta \leqslant \tau(t_{\mathrm{to}}) - \tau(t_{\mathrm{from}}) \leqslant \Delta.$$

DEFINITION 1. Petri Net is well-formed, when it is safe, has only one root transition, root transition has a single place and this place is marked, all repeating transitions newer looses the possibility of firing.

DEFINITION 2. We will restrict the system verification method only to timed and well-formed Petri Nets, where timing constraints will be associated with transitions.

DEFINITION 3. If Petri net, converted form UML diagram is not recognized as well-formed, it will be converted to well-formed by adding additional place and setting initial marking to this place.

An algorithm for timing analysis of systems with conditional behaviour (Ramchandani, 1974) can analyze systems without conditional behaviour. In such a case process graph is directly obtained from Petri net. For most of Real Time Systems behaviour modelling, Petri net without conditions is not sufficient. The possibility to describe all kind of choices is needed. In addition to uni-process graph Petri net simulation our proposed

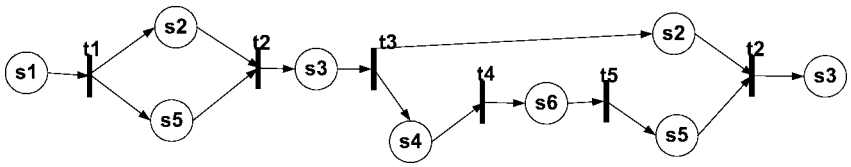


Fig. 7. One cycle process graph of Petri net, containing t4 transition and succeeding places after the branching.

algorithm, that can handle conditioned Petri net, making several process graphs for determination of minimum/maximum separation in time between two places. It generates all possible processes graphs of the system, described by the Timed Petri net with conditions. The input for the algorithm is Petri Net and two transitions ($t_{\rm from}$ and $t_{\rm to}$), or one transition repeated twice (if recurrence time of the same place is needed), for witch the lower and the upper separation bound is calculated.

There we have Petri net generated from UML activity diagram. It is cyclic conditional Petri net witch has conditional branching from place $s4$ (see Fig. 4). The task is to determine maximum separation in time between recurrences of transition $t2$.

Initially one empty process graph for Petri net is generated. The initial net place is added to this process graph, with the marking. The succeeding transition is added to the process. In the case of transition with no conditions (no branching) Petri net places and transitions step by step are added to the process graph. In the case of conditional transition – when branching is reached, initial process is duplicated to as many processes graphs as branching had ramifications. All ramification transitions are added separately to distinct processes graphs. The process graph building is finished when if contains $t_{\rm from}$ and $t_{\rm to}$ transitions or it pushes the Petri net into the state in with it already was according to that process. Following these directions we have to finish all started processes graphs.

In such a way we end up with a finite number of processes graphs. With such a partitioning of Petri net to processes, conditions are eliminated and timing constraints can be obtained directly. So, in our proposed methodology Petri net with conditions can be simulated instead of simple Petri net without condition analysis, when Petri net is equivalent to the process graph.

In Fig. 4 the Petri Net presented we get two process graphs presented in Fig. 7 and Fig. 5. Then for every process that contains transitions $t_{\rm from}$ and $t_{\rm to}$ the minimum and maximum separation in time between these places is found according to the algorithm of finding exact bounds on the time separation of events in concurrent systems. After the transition marked as second for timing analysis is reached, we have a set of processes, that hold used condition values of ramifications (choices, used for all executed transitions) and minimum, maximum times $(t_{\min}, t_{\max})$ of process execution. The whole system behaviour minimum and maximum time (δ, Δ) can be calculated as following:

$$\delta = \min(\delta_i); \quad \Delta = \max(\Delta_i),\ i \in (1, N),$$

where N – number of processes graphs; δ_i – minimum separation in time $t_{\rm from}$ and $t_{\rm to}$ in process π_i and Δ_i – maximums separation in time $t_{\rm from}$ and $t_{\rm to}$ in process π_i, for $\forall i$, such that $t_{\rm from} \in \pi_i$ and $t_{\rm to} \in \pi_i$.

5. Results and Discussion

Experiment 1. We have analyzed programs, used for designing UML diagrams and Petri Nets and decided that for UML designing, Petri Net modelling and implantations of plug-ins the most appropriate is Argo UML. So we implemented an ArgoUML plug-in that does mapping of UML diagram to a Petri Net diagram and evaluates the timing constraint of that Petri net.

We modelled temperature measurement system in UML activity diagram (Fig. 3) using Argo UML tool and converted it to Petri net (Fig. 4) according to mapping methodology. The mapping is done automatically in two passes. In first pass we map all activity diagram states, forks and joins into Petri net places and in second pass we map activity diagram transitions into Petri net transitions. So the complexity of mapping is linear for any activity diagram.

Now we can to evaluate the system. The main concern is to find out if data handling process is able to finish its job before the temperature reading process obtains new value.

In order to get timing constraints after temperature reading place ($s2$) before the synchronization on $t2$ have added addition transition ($t2'$). The adjusted Petri net is presented in Fig. 8.

In order to evaluate the system performance we evaluated this Petri net using Time Separation of Events (TSE) Fig. 6 algorithm. Was evaluated Petri net when $t_{\text{from}} = t_1$, using two simulation iterations. The simulations results are presented in Table 2.

From the results we can clearly see that in some cases data reading process has to wait for data handling process ($t2'_{\Delta} = 22$ and $t2_{\Delta} = 25$). Then by checking the first iterations results we find out that this is a case for process presented in Table 2. From calculated results it is also clear that for not critical temperature the system is able to process data before the new value arrives.

Experiment 2. There we show a particular kind of PN suitable for the modelling of RT temperature measurement system behaviour which is periodic, and then demonstrate

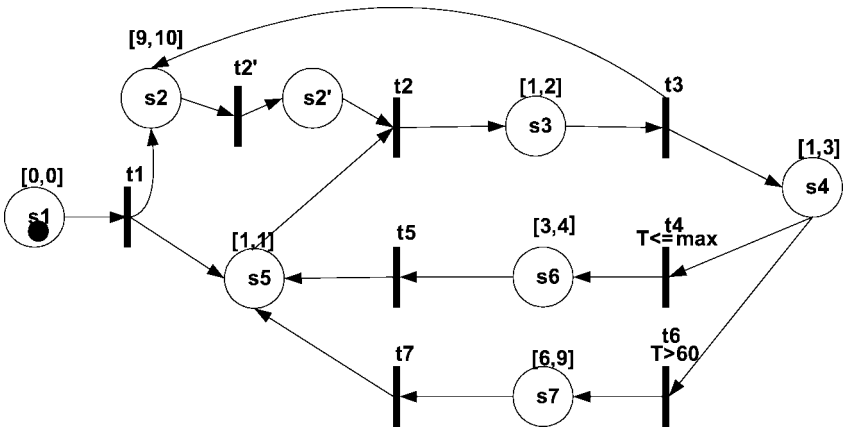


Fig. 8. Petri net with additional transition $t2'$.

Table 2

Petri net evaluation results

Transition	Time Constraint (Iteration 1)	Time Constraint (Iteration 2)
t1	**[0, 0]**	
t2'	**[9, 10]**	**[19, 22]**
t2	**[9, 10]**	**[19, 25]**
t3	**[10, 12]**	**[20, 27]**
t4	**[11, 15]**	**[21. 27]**
t5	**[14, 19]**	**[24, 31]**
t6	**[11, 15]**	**[21, 30]**
t7	**[17, 24]**	**[27, 39]**

how the PN model of a RT composed of communicating Controller and Processor may be analyzed for the presence of data transfer and processing with data transfer lost. Our presentation throughout will be rather informal and uses existing results taken from discussion in this paper.

As shown in Fig. 9, the sequential behaviour of the simplest periodic RTC in Fig. 2 (sampling, reading data from ADC, writing to transmitter Txd port, reading data from input port and putting them to the cyclic FIFO buffer, performing calculation and data visualization) may be modelled by a condition/event generated from RTC. The RTC event is related with every one transition on PN.

In Fig. 10 is shown PN model generated from previous activity diagram. Controller and Processor communication is presented as Producer/Consumer model (Nisanke, 1997), where place P4 define the size of buffer FIFO (in this case length equal 4), P5 – amount of total transferred measurements, P6 – lost transfers, P7 – total amount of received measured data and ready to display data. Next measurement calculation is defined by transfer T4, which firing condition depends on actual value of random variable rep-

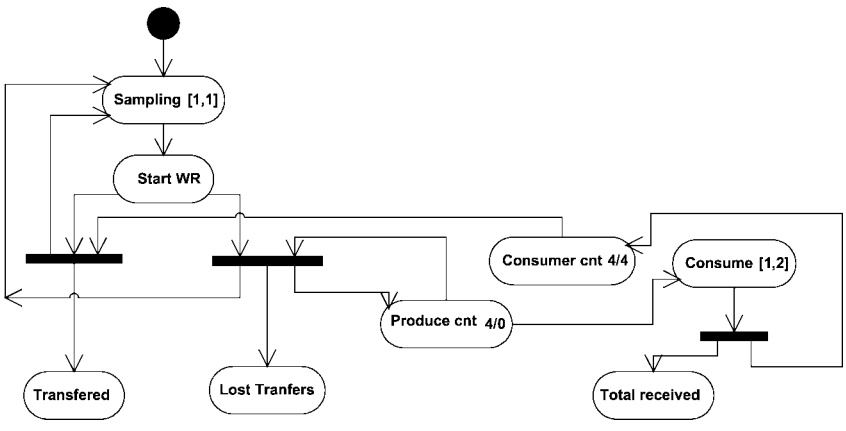


Fig. 9. Activity diagram for sequential behaviour of RTS.

resenting interval – arrival time between transmission and displaying data. Transition T3 defines measurement value transmission operation. If the FIFO buffer is not full the measurement value is put, otherwise a collision occurs – the first sample in buffer is discard.

In Fig. 11 simulated results of the Petri net model (see Fig. 10) are shown. The curve M(P5) represents amount of data sent from Producer to Consumer process. This amount is modelled as the count of tokens at place P5. Similarly curve M(P7) represents the amount of data received at the Consumer side and M(P6) is amount of data lost due to FIFO buffer overflow.

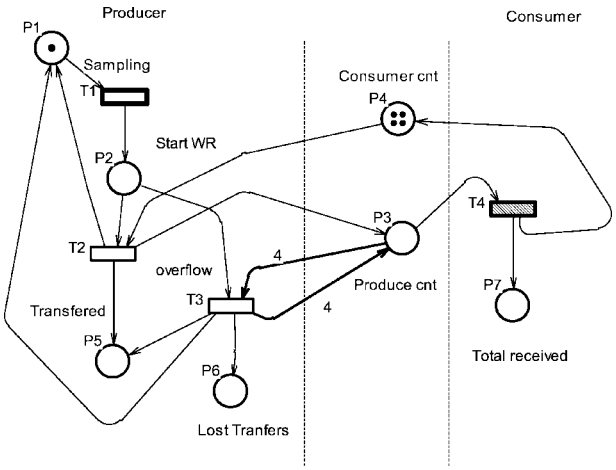


Fig. 10. Petri Net RTS model, generated from activity diagram.

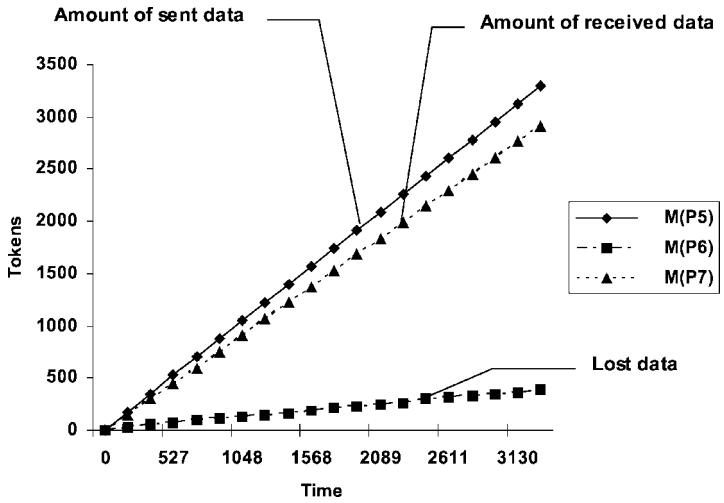


Fig. 11. Simulation results.

6. Conclusions

In this paper, we overviewed and analyzed the UML and PNs and proposed a UML-PN integrated modelling method for real-time systems development. The methodology provided an integrative framework supporting requirements description, model specification and design, model analysis using simulation and verification tools.

The TSE algorithm is very efficient for calculating timing constraints for Petri nets that do not include conditional behaviour. However, for systems where the conditional behaviour comes from the fact that the system has several distinct modes of operation we suggest to analyze each mode separately by splitting Petri net into particular processes.

The UML is well suited for capturing requirements, system view and implementation details, and Petri nets provide formalism for checking real time system's behaviour correctness.

Modeling specific applications with UML would be easier using a more specialized (domain-specific) notation representing the basic elements and patterns of the target domain.

Formally defined domain-dependent use semantics are required to avoid multiple interpretations of the same Petri net models and to support analysis and simulation tools.

Multiple diagrams can be used to capture related aspects of a design. The possibility of viewing and describing the same object from different perspectives makes the system specification phase easier, especially in embedded real time software development phase.

Experimental results have demonstrated the worthiness of such improvement techniques, and by combining the UML specification and strategy and the transformational approach the efficiency of design RTS is improved considerably.

References

Agerwala, T. (1979). Putting Petri nets to work. *IEEE Computer*, **12**(12), 85–94.

Argo UML Tool `http://argouml.tigris.org`

Bozga, M., S.G.L. Mounier, L.O.J.-L. Roux and D. Vincent (2001). *Timed Extensions for SDL*.

Coolahan, J.E., Jr. and N. Roussopoulos (1983). Timing requirements for time-driven systems using augmented Petri nets. *IEEE Transactions on Software Engineering*, **9**(5), 602–616.

Douglass, B.P. (1999). *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*. Addison–Wesley Longman Publishing Co., Inc., Boston, MA.

Gehrke, T., U. Goltz and H. Wehrheim (1998). *The Dynamic Models of UML: towards a Semantics and its Application in the Development Process*. Hildesheimer Informatik-Bericht 11/98,

Hruby, P. (1999). Designing UML based development processes, UML'99 – The Unified Modeling Language. Beyond the Standard. In *Proceedings of Second International Conference, LNCS*, Vol. 1723. Springer. pp. 308–323.

Hulgaard, H., S.M. Burns, T. Amon and G. Borriello (1995). An algorithm for exact bounds on the time separation of events in concurrent systems. *IEEE Transactions on Computers*, **44**(11), 1306–1317.

ILogix Tool (2003). `http://www.ilogix.com`

Kazanavicius, E., A. Mikuckas, I. Mikuckiene (2004). Nondestructive testing method in artificial intelligence teal time systems. *Solid State Phenomena*, **97–98**, 71–76.

Kazanavicius, E., A. Mikuckas, I. Mikuckiene, V. Kazanavicius (2004a). Noisy signal processing in real time DSP systems. *The e-Journal of Nondestructive Testing & Ultrasonics*.

MagicDraw Tool (2003). `http://www.magicdraw.com`

Murata, T. (1989). Petri nets: properties, analysis and applications. In *Proceedings of the IEEE*, **77**(4).
Nisanake, N. (1997). *Real Time Systems*. Prentice Hall.
Object Management Group (1999). *OMG Unified Modeling Language Specification*, Version 1.3.
Pranevičius, H., G. Budnikas (2003). Creation of ESTELLE/Ag specifications using knowledge bases. *Informatica*, **14**(1), 63–74.
Ramchandani, C. (1974). Analysis of asynchronous concurrent systems by Petri nets. *Technical Report*. M.I.T., Cambridge, MA, Project MAC, TR 120.
Rose Case Tool (2003). `http://www.rational.com/rose`
Selic, B., and J. Rumbaugh (1998). *Using UML for Modeling Complex Real-Time Systems*. White Paper, Rational Software Corporation. `http://www.rational.com/media/whitepapers/umlrt.pdf`
Stankovic, J. (1998). Misconceptions about real-time computing: a serious problem for next generation systems. *IEEE Computer*, **21**(10).
Stankovic, J., and K. Ramamritham (1998). Hard real-time systems. *Tutorial Text*, IEEE Computer Society Press, Wash. D.C.
Xiaoqun Du, C.R. Ramakrishnan, S.A. Smolka (2000). Real-time verification techniques for untimed systems. *Electr. Notes Theor. Comput. Sci.*, **39**(3).

E. Kazanavičius is an associate professor in Computer Department at Kaunas University of Technology. His research interests focused on design, modeling and simulation of embedded, real-time, digital signal processing and computational systems. He is a head of Computer Department and a head of Digital Signal Processing Laboratory an Research group leader. His diploma engineer degree is from Kaunas Polytechnical Institute (1979), his dr. and associate professor from Kaunas University of Technology (1982).

Realaus laiko sistemų projektavimo ir įvertinimo metodologija

Egidijus KAZANAVIČIUS

Straipsnyje pateikiama realaus laiko sistemų projektavimo metodologija naudojant UML ir Petri tinklų modeliavimą. Nagrinėjama UML veiklos diagramų aprašymo transformavimo būdas į formalų Petri tinklo modelį, skirtą realaus laiko sistemų elgsenos ir jos funkcijų vykdymo laiko apribojimų įvertinimui. Pasiūlytas sistemos funkcionavimo kritiniu laiko intervalu įvertinimo algoritmas panaudojant Petri tinklo modelį. Šios metodologijos pagrindu realizuoti taikomųjų sistemų intarpai ARGO UML aplinkoje, pateikti realaus laiko sistemos realizavimo pavyzdys ir eksperimentų gauti rezultatai.