Real-Time UML State Machines: An Analysis Approach

J. Trowitzsch*, A. Zimmermann

Technical University Berlin
Real-Time Systems and Robotics
Performance Evaluation Group
[joni,azi]@cs.tu-berlin.de

Abstract. Since real-time systems have special characteristics the development of such systems requires the observation of quantitative system aspects. Quantitative predictions are needed already during the modeling phase of the system development process. Recently the Unified Modeling Language (UML) including its Profile for Schedulability, Performance, and Time (SPT) gained increasing acceptance as a specification language for modeling real-time systems. These Real-Time UML models themself are not directly analyzable. This paper presents steps towards the derivation of analyzable Stochastic Petri Nets (SPN) from Real-Time UML state machines. The transformation of UML elements like for example pseudostates into corresponding SPN representations is covered in addition to prior work.

1 Introduction

Today's systems tend to be very complex, distributed and often with special performance requirements. Hence the design process of these complex systems mostly involves a modeling phase. The system model is used for the analysis of qualitative and quantitative properties. In the case of real-time systems special quantitative requirements like for example a certain timeliness or dependability have to be considered. Therefore it is especially important to ensure these aspects and thus an appropriate analysis method for these models is needed.

The *Unified Modeling Language* (UML) [1] in combination with its accompanying *UML Profile for Schedulability, Performance, and Time* (SPT) [2] are considered as a suitable specification language for the design of real-time systems. This combination is called Real-Time UML (RT UML) and allows the detailed specification of quantitative system properties. RT UML can be used for the consistent system design from the requirement specification towards implementation details. It provides several diagrams for the modeling of structural system properties as well as dynamic ones.

* The author's research work is supported by a PhD scholarship from the German Research Council (DFG) under grant GrK 621-2.

Since RT UML does not include a new analysis method the problem remains: how to make quantitative predictions for the system design? However, performance measures can not be obtained directly from UML models. For retrieving performance measures from RT UML two different fundamental strategies exist. The first strategy (*direct*) is the development and application of analysis methods that operates directly on the RT UML specification. The second strategy (*indirect*) is based on a transformation of RT UML specifications into an established performance model such as *Stochastic Petri Nets* [3] or *Queuing Network Models* [4]. By this, quantitative measures can be obtained by applying known analysis methods and tools for the chosen performance model. We consider the indirect strategy in this paper as the prefered one, because in this case a reuse of established knowledge for the analysis of the model is possible. Another aspect is that there also exist quite powerful tools that support quantitative analysis of established performance models, for example TimeNET (**Time**d **N**et **E**valuation **T**ool) [5] in the case of Stochastic Petri Nets.

RT UML comprises several diagram types. We think that it is recommended to focus on certain diagram types. Behavioral diagrams are the interesting ones when dealing with real-time systems. The focus within our work is on the RT UML state machine diagrams because we consider these diagrams as the appropriate basis for modeling real-time systems. In this paper we explain the transformation of RT UML state machine elements into corresponding Stochastic Petri Net fragments.

The transformation of RT UML state machines into Stochastic Petri Nets requires the preservation of the models semantics. The quantitative aspects like timing annotations must be interpreted and included into the resulting Stochastic Petri Net in such a way that the timing behavior is consistent.

Merseguer et al. present a similar indirect approach in [6]. The approach aims at software performance evaluation. It is a systematic and compositional approach that uses labeled *Generalized Stochastic Petri Nets* (GSPN) for analysis. Only exponentially distributed times are considered. Deterministic timing is not taken into account, although this is compulsory when dealing with real-time systems that include for example hard deadline requirements. Pooley and King also worked on the integration of performance evaluation techniques into the software design process using UML [7, 8]. An intuitive transformation from UML into GSPNs is proposed. Lindemann et al. present a direct approach for the quantitative analysis of UML in [9]. From extended state machine or activity diagrams a particular stochastic process is generated, the generalized semi-markov process (GSMP). Both exponentially distributed and deterministic times are covered by the approach.

Since the SPT profile has been adopted lately, it is preferable to use this standard to specify dynamic system aspects within UML. Existing approaches mostly use more or less their own extensions and annotations. We strictly follow the SPT profile standard in our work. Extensions to the SPT profile are proposed whenever necessary.

The remainder of the paper is organized as follows: In Section 2 we recall basic features of RT UML and Stochastic Petri Nets. Our transformation approach is explained in detail in Section 3. Section 4 finally gives a conclusion including open issues.

2 Background

This section recalls fundamental features of Real-Time UML and Stochastic Petri Nets. The term Real-Time UML (RT UML) refers to the UML standard [1] in combination with the SPT Profile [2].

2.1 Real-Time UML

The *Unified Modeling Language* (UML) [1] is a semi-formal language that was adopted by the *Object Management Group* (OMG) in 1997. It is a modeling language for specifying, visualizing, constructing, and documenting models of discrete event systems and models of software systems. It provides various diagram types and notations allowing the description of different system viewpoints. UML can be used for describing problems as well as their solutions. It especially achieved a wide acceptance in the field of object-oriented software development. Static and behavioral system aspects, interactions among system components and implementation details are captured. UML is quite flexible and customizable because of its extension mechanism .

UML defines several different structural and behavioral diagram types [1, Appendix A]. For modeling real-time systems especially the behavioral diagrams are important because they include the dynamic system properties and timing information. We consider the RT UML state machine diagram as the appropriate basis for modeling real-time systems and their behavior.

RT UML State Machines The RT UML state machine diagrams are a variant of Harel statecharts [10]. They can be used for modeling discrete behavior through finite state-transitions systems [1, Sec 15.1]. UML makes a distinction between *Behavioral State Machines* and *Protocol State Machines*. In the following we concentrate on *Behavioral State Machines* and refer to them when

speaking of state machines. These *Behavioral State Machines* are used to specify possible sequences of states which an individual entity may proceed through its lifetime. *Protocol State Machines* are used to express usage protocols, the legal transitions a classifier can trigger.

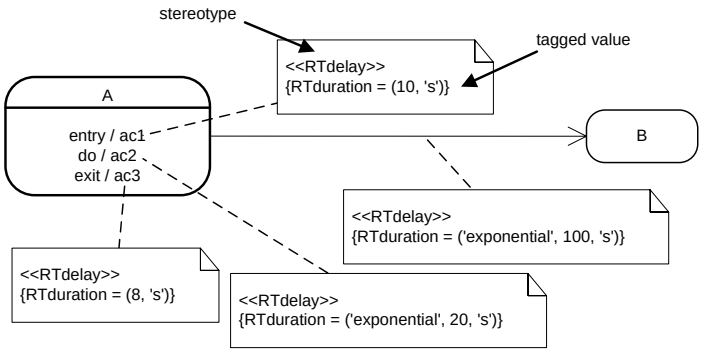


Fig. 1. Example of an annotated UML State Machine

RT UML state machine diagrams contain different elements for modeling behavior through finite state-transition systems. In detail they comprise one or more *regions* which include *vertices* (states) and *transitions*. An example of an RT UML state machine is shown in Figure 1. The two states `A` and `B` are connected via a state-to-state transition from `A` to `B`. Annotations from the SPT profile are used to add timing information to the state machine which will be explained later on.

Region A *region* is an orthogonal part of either a composite state or a state machine. It contains vertices and transitions [1, Sec 15.3].

A vertex is an abstraction of a node in a state machine graph. It can be both the source and the destination of any number of transitions [1, Sec 15.3]. Subclasses of vertices are states and pseudostates which both are introduced in the following.

State A *state* models a situation during which some usually implicit invariant condition holds. The invariant may represent a static situation such as an object waiting for some external event to occur. It can also model dynamic conditions such as the process of performing some activity [1, Sec 15.3]. When a state is entered as a result of a transition it becomes active. It becomes inactive if it is

exited as a result of a transition. Every state may optionally have one of each so-called *entry*, *exit*, and *do* activities (see state `A` in figure 1). Whenever a state is entered, it executes its *entry* activity before any other action is executed. A *do* activity represents an activity that occurs while the state machine is in the corresponding state. Before the state is exited because of an outgoing transition, the *exit* activity is executed [1]. Three kinds of states are distinguished:

- *Simple State* → is a state that does not have any substates [1, Sec 15.3].
- *Composite State* → either contains one region or is decomposed into two or more orthogonal regions. Each region has a set of mutually exclusive disjoint subvertices and a set of transitions [1, Sec 15.3].
- *Submachine State* → specifies the insertion of the specification of a submachine state machine [1, Sec 15.3].

Pseudostates A *pseudostate* is an abstraction that encompasses different types of transient vertices in the SM graph. It is used to connect multiple transitions into more complex state transitions paths and can be one of the following types:

- *inital* → represents a default vertex that is the source for a single transition to the default state of a composite state. Only one initial vertex can be in a region.
- *deepHistory* → represents the most recent active configuration of the composite state that directly contains this pseudostate.
- *shallowHistory* → represents the most recent active substate of its containing state.
- *join* → merges several transitions originating from source vertices in different orthogonal regions. Transitions entering a join vertex cannot have guards or triggers.
- *fork* → splits an incoming transition into two or more transitions terminating on orthogonal target vertices.
- *junction* → are semantic-free vertices that are used to chain together multiple transitions.
- *choice* → results in the dynamic evaluation of the guards of the triggers of its outgoing transitions.
- *entry point* → specifies an entry point of a state machine.
- *exit point* → specifies an exit point of a state machine.
- *terminate* → implies, when reached, that the execution of this state machine by means of its context object is terminated.

Transitions A *transition* is a directed relationship between a source vertex and a target vertex. It may be part of a compound transition, which takes the state

machine from one state configuration to another, representing the complete response of the state machine to a particular event [1, Sec 15.3]. In order to avoid confusion with the transitions from the Petri Net domain we refer to the transitions from the RT UML state machines as *SM-transitions* later on.

UML Profile for Schedulability, Performance, and Time The OMG adopted the *UML profile for schedulability, performance, and time* (SPT) [2] in order to eliminate UMLs lack of performance annotations and among other things to enable the advanced modeling of real-time systems. It extends UML by providing stereotypes and tagged values to represent resources used by the system, performance requirements and quantitative parameters including timing information. For example in Figure 1 the stereotype `RTdelay` with its tag `RTduration` is used for the state-to-state SM-transition and the optional internal activities of state `A`. The existence of this standard improves the interoperability between different existing UML tools. A better foundation for understanding between people is enabled. Misinterpretations are less likely to happen.

2.2 Stochastic Petri Nets

Petri Nets are based on the doctoral thesis of Carl Adam Petri [11]. They are a special kind of directed graph and have an underlying mathematical model which makes them analyzable. Petri Nets represent a model for describing the aspects of concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic systems [3]. Thus they are applicable to many systems. Two types of nodes can be found in Petri Nets: places and transitions. Arcs connect either a place to a transition or a transition to a place. Places are drawn as circles and transitions are drawn as rectangles (see example in Figure 2). Formally, following [3]:

Definition: *A Petri Net is 5-tuple,* $N = (P, T, F, W, M_0)$ *where:*

- $P = \{p_1, p_2, \ldots, p_m\}$ *is a finite set of places.*
- $T = \{t_1, t_2, \ldots, t_n\}$ *is a finite set of transitions, with* $P \cap T = \emptyset$ *and* $P \cup T \neq \emptyset$*.*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation).*
- $W : F \to \mathbb{N}^+$ *is a weight function.*
- $M_0 : P \to \mathbb{N}$ *is the initial marking.*

The basic concepts of Stochastic Petri Nets (SPNs) are reviewed in [12, 13]. In the following we assume that they are known to the reader. Transitions in the SPNs are associated with firing times. Based on their firing times transitions can be distinguished into *immediate*, *deterministic*, and *exponential* transitions. If a

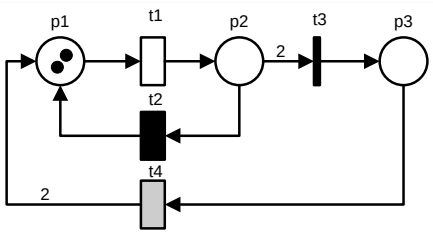


Fig. 2. Example for a Stochastic Petri Net

transition does not belong to any of these three types it is a so called *general* transitions. For a detailed description of properties of Petri Nets we refer to Murata [3]. In the case of real-time systems especially Deterministic and Stochastic Petri Nets (DSPNs) are of interest. DSPNs have been introduced in [14] and allow continuous-time modeling. Both constant timing and exponentially distributed timing are included.

Figure 2 shows an example of a SPN. It describes a two-component redundant system. Each component may fail (see `t1`) and be repaired (see `t2`). If both components fail, `t3` fires immediately, and a complete system repair is done (see `t4`). Immediate transitions are drawn as small rectangles (see `t3`). A big black rectangle represents a deterministic transition (see `t2`). A big empty rectangle shows an exponential transition (see `t1`) and a big gray rectangle represents a general transition (see `t4`). In the following we refer to the transitions from the Petri Net domain as *PN-transitions*, in order to avoid confusion with *SM-transitions*.

3 Transformation

In the following we explain our approach for transforming RT UML state machines into Stochastic Petri Nets aimed at quantitative analysis. In this context we presented first results and transformation rules in [15]. We recall the basic rules and present improved and extended ones for the basic state transformation as well as for the transformation of several pseudostates and the annotations from the SPT profile.

The approach is based on the decomposition of RT UML state machines into basic elements, like states, pseudostates, and SM-transitions. Transformation rules from RT UML to SPN fragments are specified for each element. These rules take into account, that certain annotations from the SPT profile might be associated to the RT UML elements. We focus in this context on the timing

annotations like the <<RTdelay>> stereotype. The resulting SPN fragments are finally composed following the decomposition. This is ensured because a fixed naming convention as explained in [15] is used.

3.1 Basic State Transformation

The basic state transformation as we propose it is shown in Figure 3. Each state may have optional internal *entry*, *do*, and *exit* activities. In the corresponding SPN fragment they are represented by general transitions, like for the *entry* action the PN-transition t_ent_A.

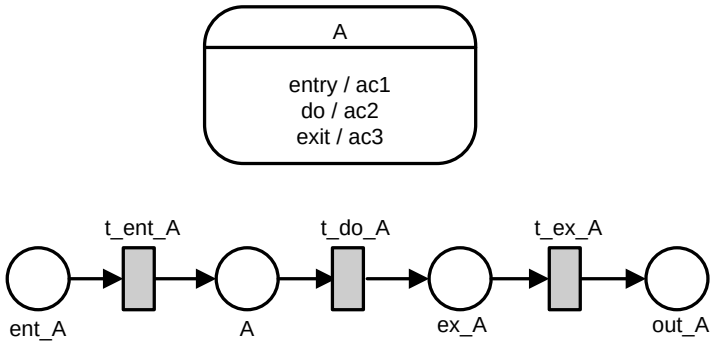


Fig. 3. Basic state transformation

Depending on the annotated timing information like in the state machine example in Figure 1 the general timing of the PN-transitions is refined. Constant delays result in deterministic PN-transitions. Exponentially distributed timing results in exponential PN-transitions. If no timing information is given or if an internal action is not specified, the resulting PN-transition is an immediate PN-transition. An example is shown in Figure 4. The missing *do* action in state A results in the immediate PN-transition t_do_A. The fixed delays for the *entry* and *exit* activities result in the deterministic PN-transition t_ent_A and t_ex_A, respectively. The SM-transition with an exponentially distributed delay with the mean value of 100 seconds results in the PN-transition t_trans_AB with the rate $\lambda = 1/100$.

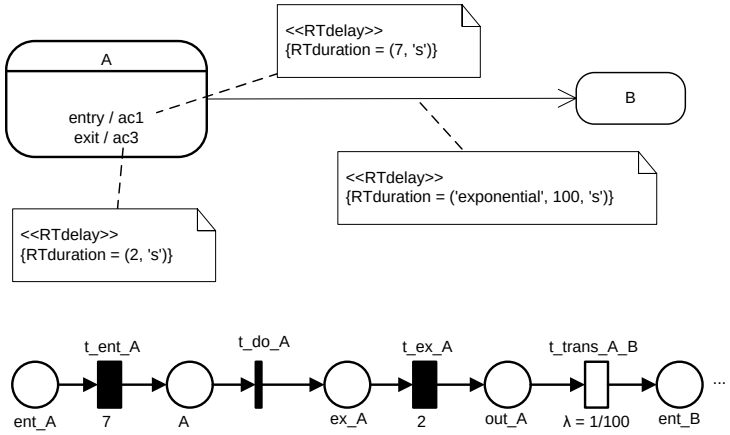
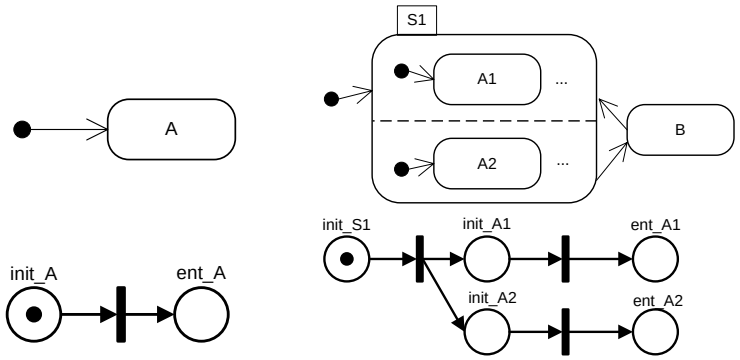


Fig. 4. Transformation of a simple state machine

3.2 Pseudostates Transformation

Pseudostates are abstractions of transient vertices in the RT UML state machines. They have special semantics that has to be considered during transformation.



(a) Simple initial (b) Advanced initial

Fig. 5. Initial pseudostate transformation

Initial The simple initial pseudostate is transformed like shown in Figure 5(a). The state `init_A` gets the initial marking of one token. If the initial pseudostate leads to two different orthogonal regions of a state machine each with its own

initial pseudostate, then the initial marking is split via immediate PN-transitions to the corresponding init places in the SPN. This can be seen for the composite state `S1` in Figure 5(b).

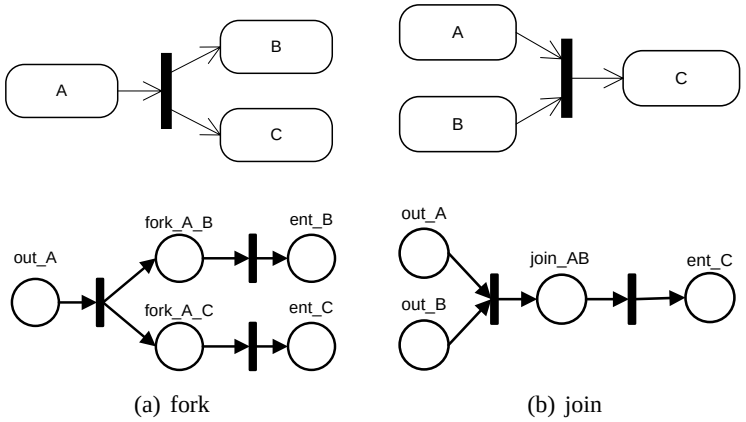


Fig. 6. Fork and Join pseudostate transformations

Fork and Join The fork and join pseudostates are used to split a transition into several orthogonal regions or to merge transitions from several orthogonal regions. Figure 6(a) shows the transformation of the fork pseudostate. The outgoing SM-transition of state `A` is split into two SM-transitions leading to the orthogonal states `B` and `C` respectively. This results in a branching of the corresponding SPN fragment. From the `out_A` place we end up in the places `fork_A_B` and `fork_A_C` as the starting points for the branches. The transformation of the join pseudostate is shown in Figure 6(b). State `C` can only be entered if both state `A` and state `B` are left and thus the related *exit* activities are completed. In the SPN domain this results in an immediate PN-transition that is activated if both places `out_A` and `out_B` contain a token. The place `join_AB` marks the point when `A` and `B` are joined and `C` is going to be entered.

Choice The choice pseudostate is a special kind of junction. It can be used to express for example probabilistic path decisions in RT UML state machines. This is for example shown in Figure 7. In difference to the presented transformation in [15] we introduced an additional state `choice_B` in order to represent the pseudostate semantic more precisely. The outgoing SM-transitions of a choice pseudostate may include a `PAprob` and a `RTduration` tag at the same time. In the SPN fragment this means that the probabilistic branching is done before

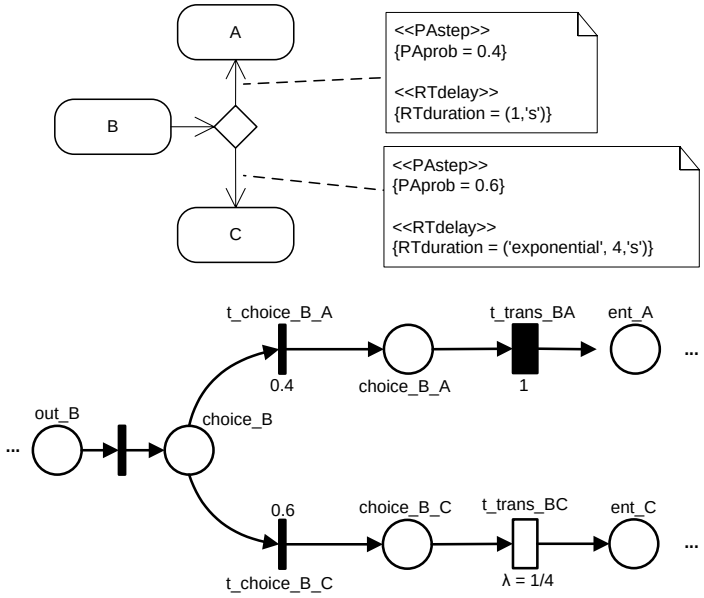


Fig. 7. Choice pseudostate transformation

the timed PN-transitions are enabled. For example in Figure 7 the immediate transition `t_choice_B_A` with the weight 0.4 appears before the deterministic PN-transition `t_trans_BA`.

Junction The junction pseudostate is a semantic free pseudostate. Figure 8 shows an example for the usage and the transformation of a junction. The SM-transitions from the states `A` and `B` end in a junction. Depending on the result of the evaluation of the guards `g1` and `g2` the junction leads either to state `C` or to state `D`. For this the corresponding guards must evaluate to *true*. If both guards evaluate to *false* no state-to-state transition is taken. For the case that both guards evaluate to *true*, no clear semantics is given by the UML specification. Therefore the guarded immediate PN-transitions have the same weight, letting the junction end either in state `C` or in state `D` depending on which PN-transition fires first.

3.3 Timing Annotations

The SPT profile provides several stereotypes for the specification of timing aspects within RT UML state machines. The `<<RTdelay>>` stereotype can be used to add durations to actions and SM-transitions. These timing information

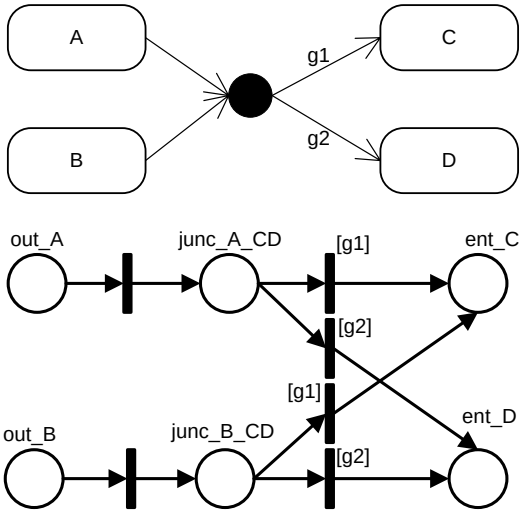


Fig. 8. Junction pseudostate transformation

is transformed into corresponding PN-transitions with an equivalent timing behavior. The proposed transformation of the currently considered `RTduration` values into resulting PN-transitions is summarized in Table 1.

Tagged Value	**PN-transition**
(8,'s')	deterministic - delay 8 sec
('exponential', 32,'s')	exponential - rate $\lambda = 1/\text{mean}$
('percentile', 80, (5, 's'), 'exponential')	exponential - rate via $F(x) = 1 - e^{-\lambda x}$

Table 1. Stereotype <<RTdelay>> - tagged value *RTduration* transformation

For the usage of the `percentile` construct we propose (in addition to the approach in [15]) that the type of the timing distribution must be specified in RT UML. This leads to a more precise specification and avoids confusions. An example can be found in Table 1: `('percentile', 80, (5, 's'), 'exponential')`. This means that for at least 80% of all cases the duration is less than 5 seconds while the time is exponentially distributed. By using the distribution function of the exponential distribution $F(x) = 1 - e^{-\lambda x}$ it is possible to calculate the rate λ. In the example this means $\lambda = -\frac{ln0.20}{5} \approx 0.3219 \left(F(5) = 1 - e^{-5\lambda} = 0.80\right)$.

For a detailed list and description of allowed distributions we refer to the SPT specification [2, Sec 5.2]. The non-exponential distributions currently all lead

to *general* PN-transitions with the corresponding firing time distributions. In these cases the resulting SPN is not numerically analyzable but simulation is still possible using for example TimeNET.

3.4 Timed Events

Events may trigger state-to-state transitions in the RT UML state machines. These events can be associated with timing information. The transformation of such construct is shown in Figure 9. Event `ev1` triggers the SM-transition from state `A` to state `B`. The event occurs after two seconds.

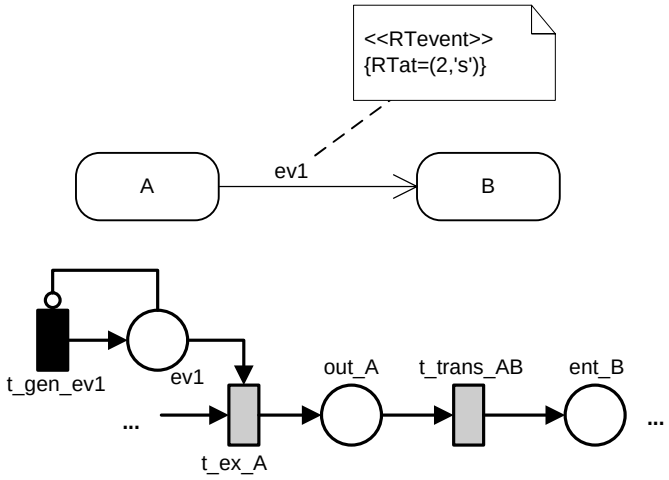


Fig. 9. Timed events transformation

4 Conclusion

The presented paper explains an approach that allows the quantitative analysis of RT UML state machine models by means of Stochastic Petri Nets. The presented approach is a compositional one. A transformation of RT UML state machine elements like states and different pseudostates into corresponding Stochastic Petri Net fragments is proposed. Quantitative annotations from the SPT profile are taken into account and included into PN-transition with equivalent timing behavior. Thus performance predictions are enabled by applying the known analysis techniques for Stochastic Petri Nets.

Not all elements from the RT UML state machines are covered in this paper. For example the *deepHistory* and *shallowHistory* pseudostates have not been studied yet. They have a complex semantic and will be investigated in the next steps.

Since the SPT profile includes more annotations than are presented here a further evaluation of our approach considering these stereotypes is needed. Another aspect is that RT UML offers different behavioral diagrams that should be part of future investigations.

References

1. Object Management Group: *Unified Modeling Language Specification v.2.0*. www.uml.org (2003)
2. Object Management Group: *UML profile for schedulability, performance, and time*. www.uml.org (2002)
3. Murata, T.: Petri Nets: Properties, Analysis and Applications. In: Proceedings of the IEEE. Volume 77(4). (1989) 541–580
4. Gross, D., Harris, C.: Fundamentals of Queueing Theory. 3rd edn. Wiley (1998)
5. Zimmermann, A., Freiheit, J., German, R., Hommel, G.: Petri net modeling and performability evaluation with TimeNET 3.0. In: Proceedings of the 11th Int. Conf. on Tools and Techniques for Computer Performance Evaluation, Schaumburg, Illinois, USA (2000) 188–202
6. Merseguer, J., Bernardi, S., Campos, J., Donatelli, S.: A Compositional Semantics for UML State Machines Aimed at Performance Evaluation. In: Proceedings of the 6th International Workshop on Discrete Event Systems (WODES), IEEE Computer Society Press (2002) 295–302
7. Pooley, R., King, P.: The Unified Modeling Language and Performance Engineering. In: IEE Proceedings - Software. Volume 146(2). (1999)
8. King, P., Pooley, R.: Using UML to derive stochastic Petri net models. In: Proceedings of the 15th UK Performance Engineering Workshop, Bristol, UK (1999) 45–56
9. Lindemann, C., Thümmler, A., Klemm, A., Lohmann, M., Waldhorst, O.: Performance Analysis of Time-enhanced UML Diagrams Based on Stochastic Processes. In: Proc. of the 3rd Workshop on Software and Performance (WOSP), Rome, Italy (2002) 25–34
10. Harel, D.: Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming **8** (1987) 231–274
11. Petri, C.A.: Kommunikation mit Automaten. PhD thesis, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2 (1962) Second Edition:, New York: Griffiss Air Force Base, Technical Report RADC-TR-65–377, Vol.1, 1966, Pages: Suppl. 1, English translation.
12. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Series in parallel computing. John Wiley and Sons (1995)
13. German, R.: Performance Analysis of Communication Systems, Modeling with Non-Markovian Stochastic Petri Nets. John Wiley and Sons (2000)
14. Ajmone Marsan, M., Chiola, G.: On Petri Nets with Deterministic and Exponentially Distributed Firing Times. LNCS **266** (1987) 132–145
15. Trowitzsch, J., Zimmermann, A., Hommel, G.: Towards Quantitative Analysis of Real-Time UML Using Stochastic Petri Nets. In: 13th Int. Workshop on Parallel and Distributed Real-Time Systems. (2005)