Towards Quantitative Analysis of Real-Time UML Using Stochastic Petri Nets

J. Trowitzsch*, A. Zimmermann, and G. Hommel
Technical University Berlin
Real-Time Systems and Robotics
Performance Evaluation Group
{joni, azi, hommel}@cs.tu-berlin.de

Abstract

In recent years the Unified Modeling Language (UML) including its profiles gained increasing acceptance as a specification language for modeling real-time systems. It is crucial to enable early quantitative predictions during the modeling phase of real-time systems development processes. UML itself is not directly analyzable. Performance evaluation techniques are thus necessary for these UML models. The challenge within our research work is the derivation of Stochastic Petri nets (SPNs) from UML models aimed at performance evaluation of real-time systems.

1. Introduction

Developing complex systems typically involves a modeling phase. The resulting model of the system can be used for qualitative analysis as well as for a quantitative analysis in the early stages of the system development process. Quantitative analysis is especially important when modeling real-time systems, because a certain timeliness and dependability must be ensured. Because of its growing acceptance in industry we consider the Unified Modeling Language [15] in combination with its *UML Profile for Schedulability, Performance, and Time* (SPT) [14] as specification language for the design of real-time systems.

However, performance measures can not be obtained directly from UML models. For retrieving performance measures from UML two different strategies exist. The first strategy (*direct*) is to develop and apply an analysis method that operates directly on the UML specification. The second strategy (*indirect*) is to map the UML specifications into an established performance model such as Stochastic Petri Nets or Queuing Network Models [3]. By this, quantitative measures can be obtained by applying the known analysis methods and tools for the chosen performance model. We consider the indirect strategy as the preferring one, because in this case a reuse of established knowledge for the analysis of the model is possible. 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) [17] in the case of Stochastic Petri Nets.

Both strategies have in common, that quantitative system aspects such as frequency, delay or service execution time have to be specified in the UML model. The mapping of UML into a performance model requires rules that specify how certain UML fragments have to be interpreted in the performance model context. In the resulting performance model the semantics of the model has to be preserved. It has to be ensured, that the timing behavior from the UML is transfered in an equivalent timing behavior in the performance model.

There are several approaches dealing with quantitative analysis of annotated UML diagrams. Mainly they are aimed at software performance evaluation. Merseguer et al. present in this context a systematic and compositional approach [12, 11, 10]. This evaluation process includes the translation of extended UML diagrams into labeled Generalized Stochastic Petri Net (GSPN) modules and finally the composition of the modules into a single model representing the whole system behavior [10]. Only exponentially distributed times are taken into account and the resulting Petri Net is thus a labeled GSPN. King and Pooley [7, 8, 16] are also working on the integration of performance evaluation into the software design process based on UML. Again GSPNs are used for the performance evaluation. An intuitive way of mapping UML State Machines (SM) into GSPNs is introduced. A state in the SM is represented as a place in the GSPN and state transitions in the SM are represented as transitions in the GSPN. The resulting GSPNs are composed, based on the UML Collaboration Diagrams (CD).

Lindemann et al. present in [9] an approach for the direct generation of an particular stochastic process, the general-

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

ized semi-markov process (GSMP), from enhanced UML state diagram or activity diagrams. The diagrams are enhanced by specifying deterministic and stochastic delays. No intermediate model like Stochastic Petri Nets is used.

Due to the lack of standards researchers have been using more or less their own UML annotation extensions to express quantitative system aspects. With the adoption of the UPSPT in 2002 it is now desirable to make use of this standard. By this a better understanding and interoperability between people and tools is enabled. Deterministic times such as constant action execution times are covered by Lindemann et al. but not by the presented GSPN-approaches.

However, when modeling real-time systems it is required to deal with deterministic and even more general behavior. Therefore we develop a transformation method to retrieve analyzable Stochastic Petri Net models from Real-Time UML. This paper shows first results from the ongoing research aimed on performance evaluation and prediction for real-time systems. It seems to be a promising approach to decompose single UML diagrams into basic scenarios and to compose the obtained performance model fragments corresponding to the decomposition. We present our approach for developing SPNs from annotated Real-Time UML models. Our focus is on the UML State Machine diagrams and by giving an example we show the application of our approach.

The remainder of the paper is organized as follows: Section 2 recalls basic aspects of UML and Stochastic Petri Nets. Our approach is introduced subsequently and in Section 4 a study for the *European Train Control System* communication link is shown. A summary and an outlook is finally given in Section 5.

2. Modeling Methods

In the following we recall some basic features of Real-Time UML and Stochastic Petri Nets. The term Real-Time UML (RT UML) refers to the upcoming UML standard 2.0 [15] in combination with the SPT Profile [14].

2.1. Real-Time UML

UML is a semi-formal modeling language for specifying, visualizing, constructing, and documenting models of discrete event systems and of software systems. It provides various diagram types allowing the description of different system viewpoints. Static and behavioral aspects, interactions among system components and implementation details are captured. UML is very flexible and customizable, because of its extension mechanism.

UML defines thirteen types of diagrams, that are divided into structural diagrams and behavioral diagrams [15, Appendix A]. Structural diagrams are used to model the logical and architectural structure of the system. Behavioral diagrams are used to describe system dynamics and thus include timing information. Therefore they are the important diagrams when dealing with quantitative analysis of RT UML models.

Among the behavioral UML diagrams we consider the UML State Machine (SM) diagram as the appropriate basis for modeling real-time systems and their behavior. Therefore we focus on this diagram type. UML SMs can be used for specifying possible sequences of states which an individual entity may proceed through its lifetime. This type of UML SMs is called *Behavioral State Machines* [15, Sec 15.1] and they are a variant of Harels statecharts [4].

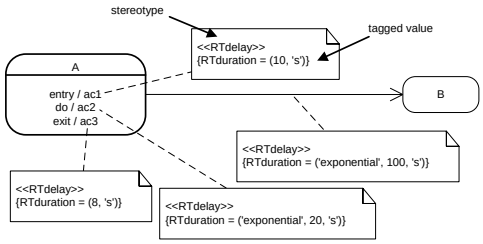

Figure 1. An UML State Machine example

UML SM include *states*, different *pseudo-states*, and *transitions*. A *state* models a situation during which some invariant (usually implicit) condition holds. 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, like for example 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 SM is in the corresponding state. Before the state is exited because of an outgoing transition, the *exit* activity is executed [15].

Pseudo-states are annotations that are used to indicate special semantics. Examples are *join*, *fork*, and *history* pseudo-states. We abstain from introducing them at this point in detail. A detailed description can be found in [15, Sec 15.3].

A *transition* causes exiting of a source state and entering of a target state. In Figure 1 one can see a transition leading from state `A` to state `B` . In the following we refer to the transitions from the UML SMs as *SM-transitions*.

The extension mechanism of UML allows the definition of so called *profiles*. A profile for a special application domain maps aspects from the domain to elements of the UML metamodel. The *UML Profile for Schedulability, Performance, and Time* is an example for such a profile. It enables advanced annotation of timing and performance informa-

tion within the behavioral UML diagrams. It provides a set of *stereotypes* and *tagged values* specializing UML without violating its existing semantics. In Figure 1 `<<RTdelay>>` is an example for a *stereotype*. This `<<RTdelay>>` *stereotype* is for example mapped to an *activity* or a SM-transition. Its *tagged value* `RTduration` can be used to specify the duration of such activity or SM-transition. A detailed description of the provided stereotypes and the associated tagged values can be found in [14].

2.2. Stochastic Petri Nets

Petri Nets [13] are a special kind of directed graph and with the underlying mathematical model they are applicable to many systems. They represent a model for describing the aspects of concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic systems. A Petri Net contains two types of nodes: 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. Formally, following [13]:

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).*
- $W : F \rightarrow \{1, 2, 3, \ldots\}$ *is a weight function.*
- $M_0 : P \rightarrow \{0, 1, 2, 3, \ldots\}$ *is the initial marking.*

We assume that basic concepts of Stochastic Petri Nets (SPNs) as defined in [1] are known to the reader. Within SPNs transitions can be associated with firing times. Transitions can be distinguished because of their firing times: *immediate*, *deterministic*, and *exponential* transitions. If a transition does not belong to any of these three types it is a so called *general* transitions. For a detailed description of more properties of Petri Nets we refer to Murata [13]. In the case of soft real-time systems especially Deterministic and Stochastic Petri Nets (DSPNs) are of interest. DSPNs have been introduced in [1] and allow continuous-time modeling. Both constant timing and exponentially distributed timing are included.

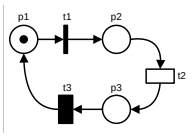

Figure 2. Example for a stochastic Petri net

An example for a SPN can be seen in Figure 2. Immediate transitions are drawn as small rectangles (see `t1`). A big black rectangle represents a deterministic transition (see `t3`). A big empty rectangle shows an exponential transition (see `t2`) and a big gray rectangle represents a general transition. 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. From RT UML SM to SPN

In the following we present the first results of how to derive SPNs from RT UML State Machines using our approach. In this paper we focus on the formalisms and annotations needed to understand the following case study. We are aware that UML SM may include different other constructs, like *composite states* and *history* pseudo-states. They successively will be subject of our future work.

We propose the decomposition of an UML SM into basic elements, like states and SM-transitions. These elements are translated into the corresponding SPN representations. Thereby we follow a certain naming convention for the places and PN-transitions in the resulting SPN fragments. By this a later composition of the fragments to a SPN representing the whole UML SM is enabled.

Our approach basically works as follows:

Ensure: use naming conventions in the resulting SPN
  identify elements in SM
  for all states $S : S \in SM$ **do**
    for all optional activities A **do**
      translate A
    end for
    translate S, by combining activities
  end for
  for all pseuo-states $P : P \in SM$ **do**
    translate P
  end for
  for all transitions $T : T \in SM$ **do**
    translate T
    connect T to SPNs of input and output state(s)
  end for
  compose stand-alone SPN fragments

In the following the translation of states, transitions, and the *choice* pseudo-state is explained in detail.

States. Within each state time may be consumed because of the optional *entry*, *do*, and *exit* activities. The execution of the activities may consume time. An example of how to translate a state can be seen in Figure 3 for state `A`. The deterministic PN-transition `t_ent_A` represents the *entry* activity `ac1` with the constant delay of 10 seconds. It connects the place `ent_A` (entering state `A`) and place `A` (entered state `A`). The *do* activity `ac2` is represented by the exponential PN-transition with the prefix `t_do_`: `t_do_A`. The

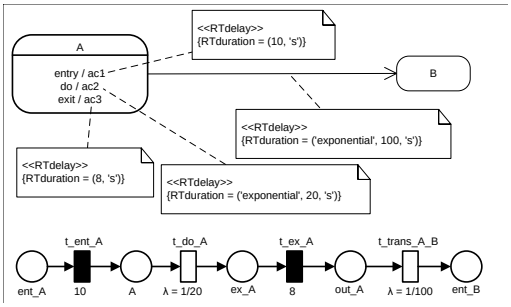

Figure 3. Basic state translation example

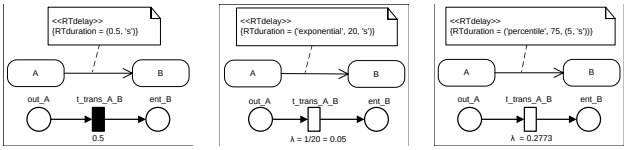

Figure 4. Translation of SM-transitions into PN-transitions

deterministic PN-transition `t_ex_A` represents the *exit* activity `ac3` with the constant delay of 8 seconds. The place `out_A` in the SPN represents the point, when state `A` has been left, thus the *exit* activity has been finished. The destination state `B` has not been entered at this point via the exponentially delayed SM-transition. In the SPN the PN-transition `t_trans_A_B` represents this SM-transition from state `A` to state `B`.

Regardless if there are no optional activities specified for a state, we always follow the logical and temporal order of the optional activities. If an optional activity is not specified for a state, the corresponding PN-transition is an immediate transition.

Thus the translation of a state *X* always results in a SPN fragment containing places and PN-transition in the following order: place `ent_X` → PN-transition `t_ent_X` (*entry* activity) → place *X* → PN-transition `t_do_X` (*do* activity) → place `ex_X` → PN-transition `t_ex_X` (*exit* activity) → place `out_X`, as shown for state `A` in Figure 3.

SM-transitions. The SM-transitions within UML SMs may consume time and are translated into corresponding PN-transitions. The naming convention for the resulting PN-transitions is as follows: For a SM-transition from state *A* to state *B* the resulting PN-transition is named `t_trans_A_B`. It connects the places `out_A` and `ent_B` (see state translation). SM-transitions without any timing annotation are considered not to consume any time. They are translated into immediate PN-transitions in the SPN model. For expressing pure delays UPSPT provides the `<<RTdelay>>` stereotype with its tag `RTduration` [14, Sec 5]. The tag is of type `RTtimeValue`. A SM-transition with a constant delay such as in Figure 4(a) is mapped into a deterministic PN-transition with an equal delay.

SM-transition can have an exponential delay like in Figure 4(b). We translate such a SM-transition into an exponential PN-transition in the SPN. We can calculate the rate λ, with the mean value being equal to $1/\lambda$. In the example we get: $\lambda = 1/20 = 0.05$.

In order to be able to express percentiles for delay functions we propose an extension of the `RTtimeValue` syntax. Similar to the `PAperfValue` [14, Sec 8] a percentile construct is introduced (`'percentile', <Real>"," <timeValStr>`). This enables for example the specification of a SM-transition with a delay of 5 seconds at most in 75% of all cases: `{RTduration = ('percentile', 75, (5, 's'))}`, see Figure 4(c). We assume that in these cases the time is exponentialy distributed. By this it is possible to calculate the rates for the corresponding exponential PN-transitions in the SPN via the density and distribution function of the exponential distribution: $f(x) = \lambda e^{-\lambda x}$ and $F(x) = 1 - e^{-\lambda x}$. For the example in Figure 4(c) this results in an exponential PN-transition with rate $\lambda = -\frac{ln 0.25}{5} \approx 0.2773 \left(F(5) = 1 - e^{-5\lambda} = 0.75\right)$.

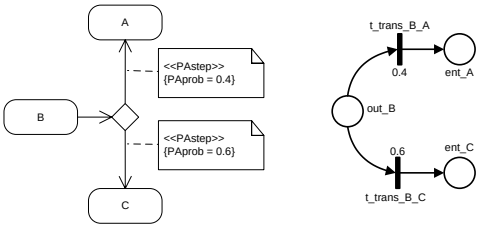

Figure 5. Usage of choice pseudo-state

Probabilistic choice. For real-time systems it is important to express probabilistic choice in the model. In contrast to the probabilistic UML-statecharts as proposed by Jansen et al. [6] we propose the usage of the *choice* pseudostate for describing probabilistic path decisions within UML SMs. In this context we follow the proposal by Merseguer to consider a SM-transition as special kind of `<<PAstep>>` [11]. By using the tag `PAprob` it is possible to express probabilistic choice. In this context we consider an UML SM as well-formed, if the sum of the probabilities attached to all outgoing SM-transitions of every *choice* pseudostate is 1. Otherwise the model might be inconsistent. An usage example and its translation is shown in Figure 5. The weight of

the resulting conflicting immediate PN-transitions is set according to the specified probabilities, which can be seen in the translation of the example.

4. Case Study

In this section we show the application of our approach to the future *European Train Control System* (ETCS) as an example. Similar models for ETCS have been considered in [5] and [18].

ETCS will be based on radio communication without using fixed track blocks. It is meant to enable fast, dense, and cross-border train traffic across Europe. One of the crucial factors for the safe and efficient operation of ETCS is the radio communication link between the on-board equipment and the radio block centers (RBC).

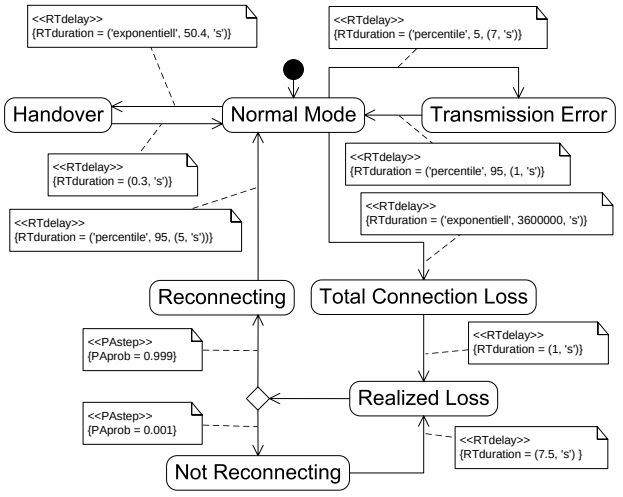

Figure 6. UML SM for ETCS radio link operational mode

The communication link between trains and RBCs is usually connected in normal operation mode. At this point three types of failures may occur: transmission errors, handovers, and connection losses. Transmission errors occur from time to time, possibly because of bad radio signal conditions. Handovers take place every time the train crosses the border between two neighboring base transceiver station (BTS) areas. Connecting to the next BTS happens automatically, but takes some time. If there are radio signal problems for a longer period of time a total connection loss may occur. Such a loss is detected by the train hardware after a certain timeout and an immediate attempt to reestablish the connection is started. This reconnection attempt may fail and in this case the reconnection procedure starts over again after a certain timeout.

Figure 6 shows the UML SM describing the ETCS radio communication link operation mode. We use the same values from the ETCS specifications as were used in [18] for modeling. Initially the radio link operates in `Normal Mode`. In this case it takes at least 7 seconds for a new transmission error to occur in 95% of all cases. This is modeled by the SM-transition from state `Normal Mode` to state `Transmission Error` with an `<<RTdelay>>` of: {`RTduration = ('percentile', 5, (7, 's'))`} (less than 7 seconds in 5% of all cases). It takes the radio link less than one second in 95% of all cases to operate in `Normal Mode` again, which is modeled by the SM-transition with an `<<RTdelay>>` of: {`RTduration = ('percentile', 95, (1,'s'))`}.

The mean distance between two neighboring BTS is 7 km. ETCS is required to work for speeds up to 500 km per hour (139 meter per second). Due to the speed of the train handovers occur quite often. The resulting worst-case mean time between two handovers is 50.4 seconds. So the transition duration is exponentially distributed with a mean value of 50.4 seconds, modeled in the UML SM with the SM-transition from state `Normal Mode` to state `Handover` with an `<<RTdelay>>` of: {`RTduration = ('exponential',50.4, 's')`}. Following the specification, the connection to the next BTS is required to take at most 300 msec. This is modeled by a SM-transition with a fixed delay of 0.3 seconds: `<<RTdelay>>` {`RTduration = (0.3,'s')`}.

A total connection loss takes place only rarely, namely 10^{-4} times per hour, every $3.6 * 10^6$ seconds once. This results in a SM-transition from state `Normal Mode` to state `Total Connection Loss` with an exponentially distributed delay with a mean value of $3.6 * 10^6$ seconds, which is modelled by an `<<RTdelay>>` of: {`RTduration = ('exponential', 3600000, 's')`}. The time needed to detect the connection loss is required to be one second at most. This is modeled by a SM-transition with a fixed delay of one second: `<<RTdelay>>` {`RTduration = (1,'s')`}. The reconnection attempt is required to be successful with a probability of 99.9%. In the remaining cases the attempt is canceled after 7.5 seconds and started over again. This is modeled in the SM using a *choice* state with two outgoing SM-transitions. One with a probability of 99.9% (`<<PAstep>>` {`PAprob = 0.999`}) and the other with a probability of 0.1% (`<<PAstep>>` {`PAprob = 0.001`}). The cancelation after 7.5 seconds is represented by a SM-transition with the fixed delay of 7.5 seconds: `<<RTdelay>>` {`RTduration = (7.5,'s')`}. In the case of a successful immediate reconnection it takes not more than 5 seconds in 95% of all

cases until the radio link operates in `Normal Mode` again. This is modeled by the SM-transition from state `Reconnecting` to state `Nomal Mode` with the following `<<RTdelay>>` annotation: {`RTduration = ('percentile', 95,( 5, 's'))`}.

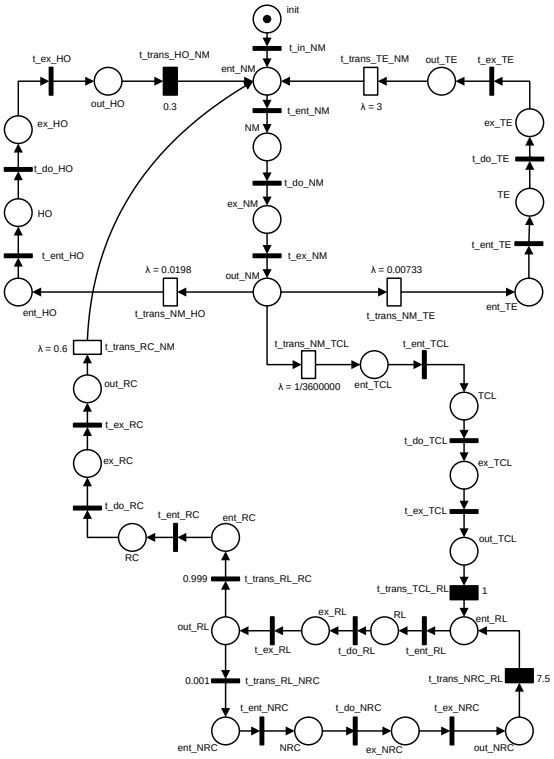

Figure 7. The resulting DSPN

The resulting SPN according to our translation approach is shown in Figure 7. For clarity reasons the following abbreviations are used: NM for Normal Mode, HO for Handover, TE for Transmission Error, TCL for Total Connection Loss, RL for Realized Loss, RC for Reconnecting, and NRC for Not Reconnecting. The SPN is a DSPN which is *strongly-connected* and *safe* (1-bounded). One can see unnecessary sequences of immediate PN-transitions, like in the handover part of the net. This is due to the fact, that in the states of the ETCS SM no *entry* or *exit* activities are specified, while the translation considers the order of all possible internal activities. In this case a trivial aggregation of these structures is feasible like shown in Figure 8.

The ETCS specification [2] requires an availability for the communication link of 99.95%. In our SM this means, that the probability of being in state `Normal Mode` has to be at least 99.95%. The resulting SPN (see Figure 7) is analyzable. The result of the numerical analysis of the SPN using TimeNET shows that the communication link is working in `Normal Mode` with a probability of 99.166% only. This result shows that the required availability for the communication link is not met.

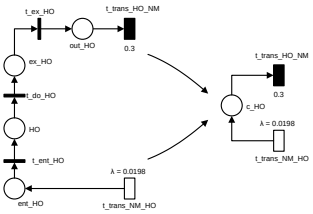

Figure 8. Trivial net size minimization

5. Conclusion

Because Real-Time UML becomes more and more established for modeling real-time systems it is useful to apply performance analysis methods on the resulting models. Our focus is on modeling using UML State Machine diagrams and the *UML Profile for Schedulability, Performance, and Time* (SPT). A model translation from Real-Time UML to Stochastic Petri Nets is proposed. Established methods from the Petri Net domain can be used for quantitative analysis. With the *European Train Control System* example the applicability of our model translation approach is shown.

Deterministic and exponentialy distributed timing are already covered by our SPN approach. Because the SPT Profile allows the use of several different non-exponential distributions translation into general PN-transitions might be necessary. Also the different pseudo-states and aspects like triggered transitions have to be studied in detail. Currently the resulting Stochastic Petri Net contains unnecessary sequences of immediate transitions. Thus a refinement of our approach considering a special treatment of states without internal activities is intended.

The model translation is currently performed by hand. An implementation of our approach in the framework of TimeNET [17] has already been started.

References

[1] M. Ajmone Marsan and G. Chiola. On Petri Nets with Deterministic and Exponentially Distributed Firing Times. *LNCS*, 266:132–145, 1987.

[2] EEIG ERTMS User Group. *Euroradio FFFIS*. UIC, Brussels, 2000.

[3] D. Gross and C. Harris. *Fundamentals of Queueing Theory*. Wiley, 3rd edition, 1998.

[4] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[5] D. Jansen and H. Hermanns. Dependability Checking with StoCharts: Is Train Radio Reliable Enough for Trains? In

Proc. of the 1st Int. Conf. on the Quantitative Evaluation of Systems (QEST), pages 250–259, Enschede, Netherlands, 2004.

[6] D. Jansen, H. Hermanns, and J.-P. Katoen. A Probabilistic Extension of UML Statecharts: Specification and Verification. *LNCS*, 2469:355–374, 2002.

[7] P. King and R. Pooley. Using UML to derive stochastic Petri net models. In *Proceedings of the 15th UK Performance Engineering Workshop*, pages 45–56, Bristol, UK, July 1999.

[8] P. King and R. Pooley. Derivation of Petri Net Performance Models from UML Specifications of Communications Software. In *Proceedings of the 11th Int. Conf. on Tools and Techniques for Computer Performance Evaluation*, pages 262–276, Schaumburg, Illinois, USA, 2000.

[9] C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O. Waldhorst. Performance Analysis of Time-enhanced UML Diagrams Based on Stochastic Processes. In *Proc. of the 3rd Workshop on Software and Performance (WOSP)*, pages 25–34, Rome, Italy, 2002.

[10] J. López-Grao, J. Merseguer, and J. Campos. Performance Engineering Based on UML and SPNs: A Software Performance Tool. In *Proceedings of the Seventeenth International Symposium On Computer and Information Sciences (ISCIS XVII)*, pages 405–409, Orlando, Florida, USA, October 2002. CRC Press.

[11] J. Merseguer. On the use of UML State Machines for Software Performance Evaluation. In *Proc. of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2004.

[12] J. Merseguer, S. Bernardi, J. Campos, and S. Donatelli. A Compositional Semantics for UML State Machines Aimed at Performance Evaluation. In *Proceedings of the 6th International Workshop on Discrete Event Systems (WODES)*, pages 295–302. IEEE Computer Society Press, October 2002.

[13] T. Murata. Petri Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, volume 77(4), pages 541–580, April 1989.

[14] Object Management Group. *UML profile for schedulability, performance, and time*. www.uml.org, March 2002.

[15] Object Management Group. *Unified Modeling Language Specification v.2.0*. www.uml.org, September 2003.

[16] R. Pooley and P. King. The Unified Modeling Language and Performance Engineering. In *IEE Proceedings - Software*, volume 146(2), March 1999.

[17] A. Zimmermann, J. Freiheit, R. German, and G. Hommel. 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*, pages 188–202, Schaumburg, Illinois, USA, 2000.

[18] A. Zimmermann and G. Hommel. Towards Modelling and Evaluation of ETCS Real-Time Communication and Operation. *Journal of Systems and Software*, 2004 (in press).