Discrete Time Stochastic Petri Nets for Modeling and Evaluation of Real-Time Systems

Armin Zimmermann, Jörn Freiheit, and Günter Hommel

Real-Time Systems and Robotics Group
Technische Universität Berlin
Franklinstraße 28/29, D-10587 Berlin, Germany
E-mail: azi | freiheit | hommel @cs.tu-berlin.de

Abstract

The design of real-time systems requires modeling and analysis techniques, to ensure their correct and timely operation. In many cases a realistic model should be able to cover both fixed and stochastic times. Stochastic Petri nets are a promising description technique in this field, but mixing deterministic and randomly distributed times in one model makes the analysis often impossible. This paper shows that Petri net models with an underlying discrete time can be advantageous for the modeling and analysis of real-time systems. For a demonstration a simple application example is modeled and its behavior computed using the software tool TimeNET.

1. Introduction

One of the challenges in real-time systems today is the increasing complexity of upcoming applications and the associated goals for their design. Standard methods are often not capable of handling these new problems anymore. Appropriate methods for the system specification and analysis of quantitative and qualitative requirements are therefore needed. These approaches should be applicable to real-time systems which cannot be characterized accurately by a priori non-stochastic models.

One way to advance the analysis of complex real-time systems is the development of mathematical modeling and analysis frameworks for this application domain. Using appropriate analysis techniques allows to prove logical and temporal properties based on this model. Example measures could be the liveness and performance of the system, or the timeliness of task execution [10]. All these measures should take into account the effect of possible failures and repairs of the system. An overview of the wide range of existing specification techniques of real-time systems can be found in [2].

Delays in typical real-time computing models are usually assumed to be fixed worst-case times. Several authors have shown the drawbacks of this approach for certain systems. Recent paradigms therefore aim at generalizing the execution time model in different ways. On the other hand, in the field of modeling and analysis of non-Markovian stochastic systems techniques have emerged, that might be successfully applied to real-time systems.

The authors claim that Petri nets [14] and their stochastic timed extensions are a useful formalism for real-time systems. They are generally used for the modeling and analysis of discrete event systems because of their ability to describe them in a concise and appropriate way. On the other hand, there are a lot of different analysis and simulation techniques as well as software tools available. Petri nets have already been proposed in the context of real-time systems, see e.g. [3, 9, 13, 15].

The remainder of the paper is organized as follows: After a brief overview of the applicability of different Petri net classes for real-time systems in Section 2, discrete time deterministic and stochastic Petri nets (DDSPNs) and the application example are introduced in Section 3. The following three sections describe the reachability graph generation of a DDSPN and the further numerical transient and steady-state analysis, each using the example. Section 7 briefly describes the software tool TimeNET, in which the described algorithms have been implemented.

2. Petri Nets for Real-Time Systems Modeling

The time behavior of many existing real-time applications can only be described using both deterministic and stochastic times. Markovian stochastic Petri nets, like generalized stochastic Petri nets (GSPNs [4]), are not suitable to model this behavior. GSPNs only allow transition firing times to be either immediate or exponentially distributed. In contrast to this, non-Markovian SPNs offer deterministic or even more general distributions in addition to that [8]. This makes them suitable for real-time systems from the point of view of necessary firing time distributions. One example is the model class called deterministic and stochastic Petri nets (DSPNs [1]).

However, it is well known that the numerical analysis of non-Markovian SPNs is only efficiently possible if in every system state there is at most one transition enabled with non-exponentially distributed firing time [5, 8]. Different approaches aim at relaxing this constraint, like

- approximation of deterministic transitions by substituting them with Erlang or generalized Cox distributions [6]
- utilizing special properties as e.g. in cascaded deterministic and stochastic Petri nets [7]
- mapping DSPNs to general state space Markov chains [11].

The reason for this problem is that for every non-exponential transition the remaining firing time has to be memorized during the state space analysis. The restriction of exponentially timed transitions as in GSPNs leads to simple analysis algorithms, because well-known Markov chain techniques can directly be applied based on the generation of the reduced reachability graph.

A quite different approach is to interprete the Petri net model as having a discrete underlying time scale (as opposed to continuous time in GSPNs and in DSPNs). This class called discrete time DSPNs [16, 17]. Instead of the continuous exponential distribution, the discrete geometrical distribution is used preserving the memoryless property. Because in every time step the firing probabilities are given by the geometrical distribution, and the remaining firing times (having only some discrete values) can be stored together with the marking, the performance analysis does not pose mathematical problems. Moreover, as immediate and deterministic transitions are special cases of the geometric distribution, there is no problem in having any number of them enabled concurrently in a marking. This overcomes the main principal problem of analyzing real-time systems with continuous time stochastic Petri nets. Figure 1 shows a short overview of some relationships between models with underlying continuous and discrete time scale.

The analysis is then naturally based on a mapping of the (reduced) reachability graph onto a discrete time Markov chain. Unfortunately the reachability graph is larger with respect to the continuous time case, because for all enabled transitions the associated remaining firing times are part of

	Continuous time	Discrete time
Firing time distribution:		
	exponential	geometric
Timed transitions:		
	Firing rates $\mathbf{Q}$	Probabilities $\mathbf{P}$
State equation:		
	$\frac{d}{dt}\pi(t) = \pi(t)\,\mathbf{Q}$	$\pi(t+\Delta t) = \pi(t)\,\mathbf{P}$
Steady-state solution:		
	$0 = \pi\,\mathbf{Q}, \sum \pi_i = 1$	$\pi = \pi\,\mathbf{P}, \sum \pi_i = 1$
Transient solution:		
	$\pi(t) = \pi(0)\,e^{\mathbf{Q}t}$	$\pi(t) = \pi(0)\,\mathbf{P}^{\lfloor \frac{t}{\Delta t} \rfloor}$

Figure 1. Time scale relation

the unique state description. Care has to be taken with transitions trying to fire at the same instant of time. Proper association of priorities avoids the problem of confusions in this case [16, 17].

3. Discrete Time Deterministic and Stochastic Petri Nets

As the underlying time scale of the DDSPN model is discrete, the system is only observed at equidistant times. Enabled transitions fire with certain probabilities solely at these points in time. The following transition firing time distributions are allowed: Geometric distribution is the discrete memoryless distribution, while deterministic transitions fire after a fixed delay, with immediate transitions as a special case. In general, any discrete time phase distribution that can be described by a finite absorbing discrete time Markov chain can be used. Please refer to [16, 17] for a thorough description of the modeling formalism of DDSPNs.

Figure 2 shows a simple totally deterministic model of three periodic processes with the following parameters:

process #	period	comp. time	RMA priority
1	4	1	1
2	5	2	2
3	10	3	3

Priorities are assigned using RMA [12]. The use of inhibitor arcs (with a small circle at the inhibited transition) implements the priority of process 1 over 2 and 2 over 3. The ready time of all processes is assumed to be zero, thus in the initial marking (shown in figure 2) all places contain one token. The marking of place P1 inhibits the execution of process 2 and 3. More than one token in a place would indicate that the deadline of the corresponding process has

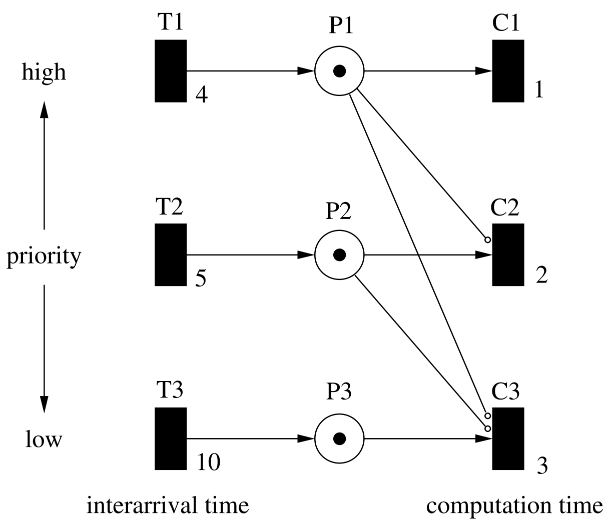

Figure 2. DDSPN model of three processes

not been met. One benefit of the shown model is that it is trivial to modify the model such that it captures aperiodic processes with arbitrary arrival time distributions and/or arbitrary computation time distributions. The overall utilization with the three processes is 95 percent.

The selection of the underlying time step Δt is important. If the real behavior of the modeled system only exhibits discrete times, they can be directly used as the transition firing delays, and the time step Δt should be chosen as the GCD of the firing times. If there are actions in the system with a continuous firing time distribution (e.g. exponential) in reality, they are modeled using a discrete distribution (e.g. geometrical), introducing a discretization error. This error decreases for smaller Δt. However, a smaller time step leads to a bigger state space and longer execution time.

4. Reachability graph generation

The analysis of the stochastic process described by a DDSPN model requires the generation of the reachability graph first. This graph contains one node for each state of the model that is reachable from its initial marking. Only states in which the process spends some time (*tangible*) are interesting for the later analysis. The remaining so-called *vanishing* states are eliminated during the generation, resulting in the reduced reachability graph. Figure 3 shows a part of the reachability graph including vanishing markings (denoted with round boxes).

It should be noted that in contrast to other Petri net classes, a DDSPN state contains – in addition to the standard vector of tokens in places – a vector that holds the remaining firing times of enabled timed transitions. The

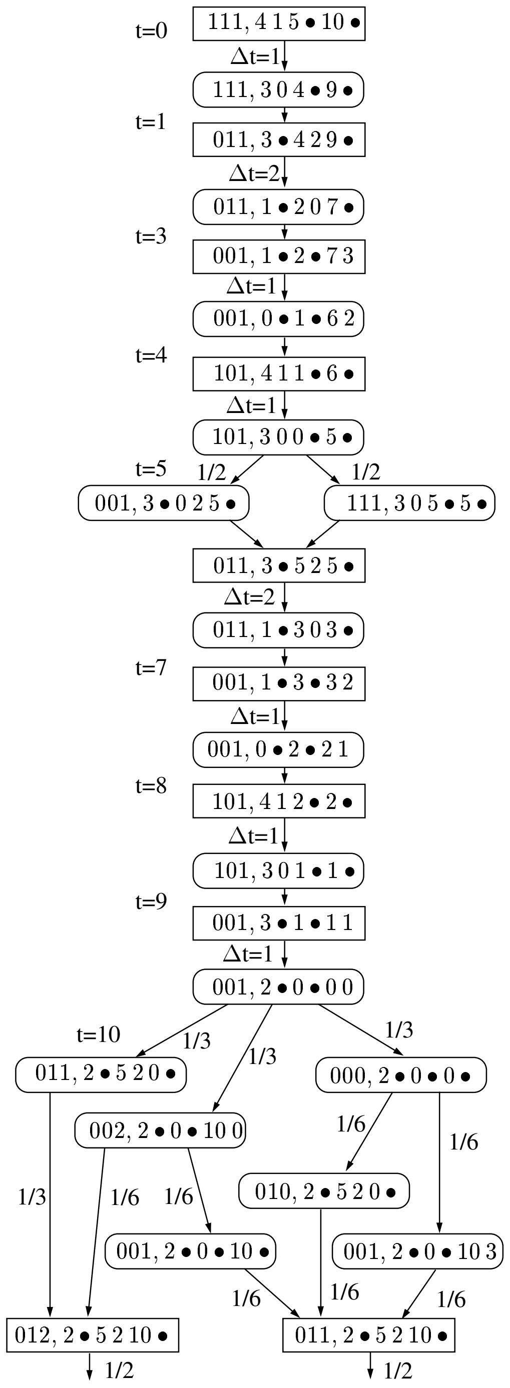

Figure 3. Partial reachability graph

first three digits show the number of tokens in places P1, P2 and P3. The remaining numbers correspond to the remaining firing times of the transitions in the sequence T1, C1, T2, C2, T3 and C3. A • denotes a disabled transition. The graph starts with the initial marking at time zero. Arcs are inscribed with either the time step Δt for state changes from tangible states, or the probabilities for immediate state changes.

Simplified, the algorithm works as follows: As long as there are states that have not been examined, one of them is taken and the firing times of all enabled timed transitions are decreased by Δt. If this yields zero for the remaining firing time equal to zero for at least one of these transitions, the new marking is vanishing and all possible firing sequences of transitions during the same time instant, leading to tangible states, are explored. Those states are stored in the reachability graph, together with the computed overall probability for traversing through any path from the previous tangible state to this one. Repeating the described steps until all states have been examined, results in the matrix $\mathbf{P}$ that contains the state change probabilities for each tangible state.

If no remaining firing time reaches zero after subtracting Δt, the new state is also tangible and the state change probability is one. An example for this case is shown in figure 3, where at time point 1 the remaining firing times of all enabled timed transitions is greater or equal 2. Time can then be advanced by $\Delta t = 2$ or the minimum of the remaining firing times of all enabled timed transitions in the general case. This technique is called *embedding* and avoids the generation of unnecessary intermediate states. The time spent in this state is then different from 1 and has to be stored for a later conversion step.

In the continuous time domain the probability of firing several timed transitions at the same instant of time is zero. This is not the case for DDSPNs due to their underlying discrete time scale. Moreover there is a (model dependent) very high probability of having to fire two timed transitions at the same time instant. If there are conflicting transitions trying to fire at the same time, the further evolution of the stochastic process depends on which one is fired first. This contradicts the notion of firing at the same time. This problem is well-known for the firing of immediate transitions in continuous-time nets as *confusion* [4].

The used analysis techniques [16, 17] detect confusions during the generation of the reduced reachability graph. The modeler can change the firing priorities of transitions to solve conflicts, thus avoiding confusions. Figure 3 shows a first part of the reachability graph without different priority associations to transitions. At time $t = 5$ there are transitions C1 and T2 fireable, and both possible firing sequences are traversed with probability $1/2$. As both lead to the same tangible state, no problem occurs. The lower part of figure 3 serves as an example for a confusion that is related to incorrect modeling. At time $t = 10$, there are three transitions fireable, namely T2, T3 and C3. After all possible firing sequences have been executed, two different tangible states are reached. Some of the state changes and the left state are wrong in the sense of the model. They occur because if there is a process termination and invocation at the same time, in the model there is no specification which happens first. In a correct model the termination should happen first. This is achieved by increasing the priority of all C transitions over all T transitions. After this change, only the right one of the two possible tangible states is reached. Giving different priorities to transitions as much as possible decreases the number of firing paths, and can therefore speed up the analysis process.

The reachability graph can already be analyzed to check for certain model properties like deadlocks. For the example the check of the reachability graphs shows that after the priority adjustment there is no state in which there is more than one token in each of the places. This means that the three processes are schedulable.

5. Transient analysis

A transient analysis shows the behavior of the model from a starting point in time (usually zero) to a predefined time. The basic iterative approach [16] to the transient analysis is to start with the initial probability distribution over all reachable states $\pi(0)$. To analyze the behavior until time t, $\lfloor \frac{t}{\Delta t} \rfloor$ matrix-vector multiplications have to be performed iteratively using $\pi(t+\Delta t) = \pi(t)\,\mathbf{P}$. From each probability distribution vector the particular performance measures can be computed and plotted over time. If only the probability vector at a time t is needed, the power method can be used alternatively: $\pi(t) = \pi(0)\,\mathbf{P}^{\lfloor \frac{t}{\Delta t} \rfloor}$

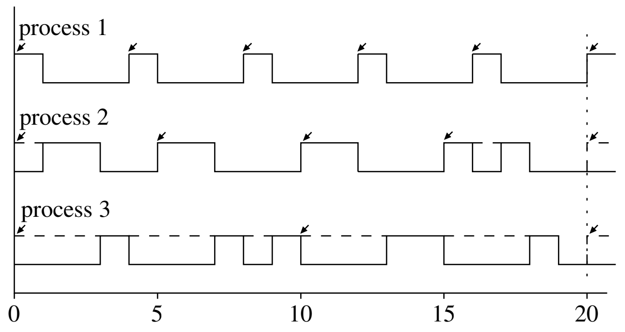

Figure 4. Transient behavior of the example

Figure 4 shows a transient plot of the number of tokens in places P1, P2 and P3 until time 20 (the superperiod). Arrows depict starting times of processes, and dashed lines mean that the process is waiting for the processor due to a higher prioritized process running. This analysis technique can be used for a graphical overview of the system behavior like a Gantt chart.

6. Steady-state analysis

Analyzing the quantitative steady-state behavior of a model answers questions about performance and dependability measures in equilibrium. Based on the **P** matrix that has been computed during the reachability graph generation, the stationary probability distribution vector π can be obtained by solving the following system of linear equations [16] with a standard solver:

$$\pi = \pi\,\mathbf{P}, \sum \pi_i = 1$$

In case of embedding, the result has to be rescaled by a multiplication with the holding times, and finally again normalized to ensure a total sum of probabilities equal to one. From the vector π the values of performance measures can be calculated.

The application model in figure 2 is purely deterministic, whose steady-state behavior is not very interesting due to its periodic evolution. In the case of aperiodic processes this is relevant, however.

In the following example it is assumed that the second process is aperiodic and the interarrival time is geometrically distributed and varied from 5 to 50. In the model this requires only a change of transition T2 from type deterministic to exponential. Figure 5 shows the probability that a process misses its deadline versus the overall system utilization, which simply can be determined by the probability that there are tokens in P1, P2 or P3. A utilization of 95% is reached with a mean interarrival time of 5. This is not surprisingly the same result as in the previous example where the period of this process was 5. The line named "P2" shows the probability of missing the deadline for the second process, and the line named "P3 det" the same for process 3. It is interesting to see that below an overall utilization of 85 percent the probability of P3 is less than that of P2, although the latter has priority over P3. This can easily be explained as the load due to process 2 equals the load due to process 3 at a total utilization of 85%.

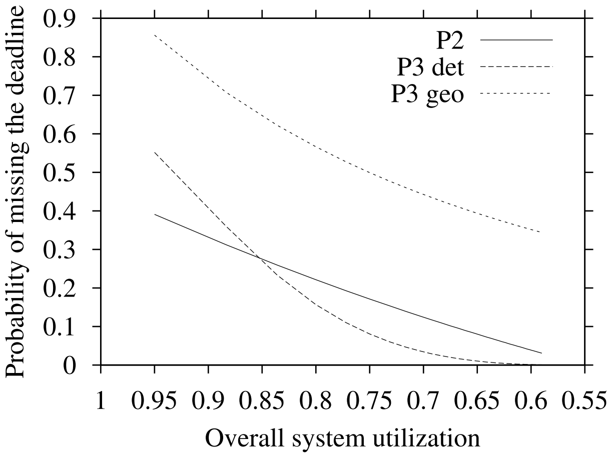

Figure 5. Analysis of real-time behavior

A second experiment was conducted to further explore the effect of stochastics to the real-time behavior. Now both interarrival times of processes 2 and 3 are set to stochastic, resulting in the plot named "P3 geo" showing the probability of process 3 missing its deadline. The curve for P2 is the same as before, because the lower priority process 3 does not change its behavior. Although the overall system utilization is the same as before, the real-time behavior of process 3 is much worse than in the first deterministic case.

7. Software tool support

Modeling and evaluation of complex systems is only feasible with the support of appropriate software tools. Since the modeling framework of stochastic Petri nets has been proposed, many algorithms and their implementations as software tools have been developed. A powerful and easy to use graphical interface is important in addition. The techniques described in this paper have been implemented in the tool TimeNET (**Time**d **N**et **E**valuation **T**ool, [18]). It offers non-Markovian uncolored and colored Petri net modeling and numerical analysis as well as simulation algorithms. For DDSPN models modules for the steady-state and transient numerical analysis as well as efficient parallel simulation are available [16].

For the current version 3.0 of TimeNET a new generic graphical user interface has been developed. Figure 6 shows a sample screen shot of the interface during a modeling session. The interface can be used for graph-like models with different types of nodes and arcs. Nodes can be hierarchically refined by corresponding submodels. It is implemented in C++ and uses the Motif toolkit.

TimeNET is available free of charge for non-commercial use under Solaris and Linux. For further information please refer to the web information at the URL `http://pdv.cs.tu-berlin.de/~timenet`.

8. Conclusion

Model based analysis is helpful in the design of real-time systems. Stochastic Petri nets with discrete timing (namely DDSPNs) can be used to describe and analyze real-time systems. It is possible to mix deterministic and stochastic times, which is a major advantage over standard analysis techniques for continuous-time stochastic Petri nets. In the paper the background and use of DDSPNs for a simple application example has been explained briefly. The described algorithms are implemented in a freely available software tool. An open problem is – like for other Petri net analysis techniques – the state space explosion problem.

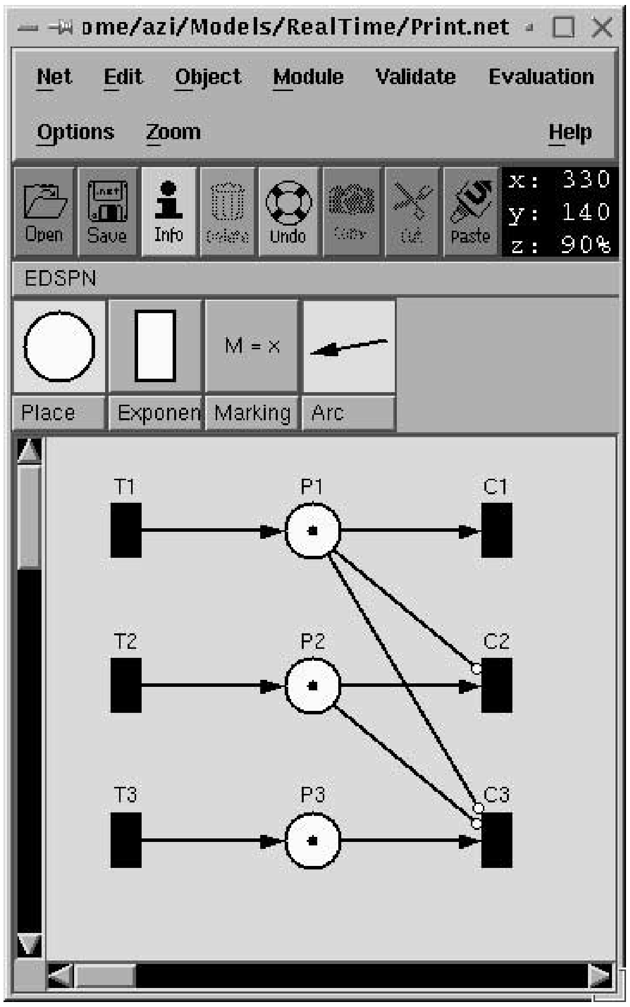

Figure 6. Screen shot of TimeNET

The authors would like to thank the participants of the Dagstuhl Seminar 271 on "Stochastic and Dynamic Real-Time Systems", held in July 2000, for the stimulating talks and discussions. Moreover, we would like to acknowledge the work of our former colleague Robert Zijal, who developed and implemented the DDSPN formalism and algorithms in his Ph. D. thesis.

References

[1] M. Ajmone Marsan and G. Chiola. On petri nets with deterministic and exponentially distributed firing times. In G. Rozenberg, editor, *Advances in Petri Nets 1987*, volume 266 of *Lecture Notes in Computer Science*, pages 132–145. Springer Verlag, 1987.

[2] G. Bucci, M. Campanai, and P. Nesi. Tools for specifying real-time systems. *Real-Time Systems*, 8:117–172, 1995.

[3] G. Bucci and E. Vicario. Compositional validation of time-critical systems using communicating time petri nets. *IEEE Transactions on Software Engineering*, 21(12):969–992, 1995.

[4] G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. Generalized stochastic petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, 19(2):89–107, 1993.

[5] G. Ciardo, R. German, and C. Lindemann. A characterization of the stochastic process underlying a stochastic petri net. *IEEE Transactions on Software Engineering*, 20:506–515, 1994.

[6] R. German. *Analysis of Stochastic Petri Nets with Non-Exponentially Distributed Firing Times*. Dissertation, Technische Universität Berlin, 1994.

[7] R. German. Cascaded deterministic and stochastic petri nets. In *Proc. 3rd Int. Meeting on the Numerical Solution of Markov Chains*, pages 111–130, Zaragoza, Sept. 1999.

[8] R. German. *Performance Analysis of Communication Systems, Modeling with Non-Markovian Stochastic Petri Nets*. John Wiley and Sons, 2000.

[9] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze. A unified high-level petri net formalism for time-critical systems. *IEEE Transactions on Software Engineering*, 17(2):160–172, Feb. 1991.

[10] W. Halang, R. Gumzej, M. Colnaric, and M. Druzovez. Measuring the performance of real-time systems. *Int. Journal of Time-Critical Computing Systems*, 18:59–68, 2000.

[11] C. Lindemann and G. Shedler. Numerical analysis of deterministic and stochastic petri nets with concurrent deterministic transitions. *Performance Evaluation, Special Issue Proc. of PERFORMANCE '96*, pages 565–582, 1996.

[12] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[13] A. Mazzeo, N. Mazzocca, S. Russo, and V. Vittorini. A systematic approach to the petri net based specification of concurrent systems. *Real-Time Systems*, 13:219–236, 1997.

[14] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[15] B. Ravindran, L. Welch, and C. Kelling. Building distributed, scalable, dependable real-time systems. In *Proceedings of the 10th IEEE Int. Conf. on Engineering of Computer Based Systems*, pages 452–459, Mar. 1997.

[16] R. Zijal. *Analysis of Discrete Time Deterministic and Stochastic Petri Nets*. Dissertation, Technische Universität Berlin, Oct. 1997.

[17] R. Zijal, G. Ciardo, and G. Hommel. Discrete deterministic and stochastic petri nets. In *9. ITG/GI-Fachtagung: Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen (MMB'97)*, pages 103–117, Freiberg, Germany, 1997. VDE-Verlag.

[18] A. Zimmermann, J. Freiheit, R. German, and G. Hommel. Petri net modelling and performability evaluation with TimeNET 3.0. In *11th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 188–202, Schaumburg, Illinois, USA, 2000. LNCS 1786.