Extensions to the Petri Net Library in Modelica

Stefan M.O. Fabricius
Swiss Federal Institute of Technology Zurich (ETHZ), Switzerland
Laboratory for Safety Analysis
fabricius@lsa.iet.mavt.ethz.ch

Abstract

The standard Modelica[1] library contains a sub-library for modeling of discrete phenomena with Petri net (PN) formalism. It is designed for, and implements, black deterministic priority PNs, which are well suited e.g., for control system specification, but can have rather limited expressiveness in other problem domains. Such may be in reliability engineering or in investigation of socio-technical aspects of complex technical systems. Therefore, the PN library already available is extended on the one hand with transitions allowing for deterministic or stochastic time delays before its firing and on the other hand with places capable of containing more than one token. Several modeling and simulation examples are given to demonstrate the usability of the enhancements–among them–queuing models and models to determine system availability.

1 Introduction

In 1962, C.A. Petri ([Pet62]) addressed in his dissertation the problem of representing co-operating, concurrent, or competing processes by a graphical modeling formalism, subsequently referred to as *Petri Nets*. Formally, PNs are bipartite digraphs and in their original form, no notion of time was associated.
Later, many extensions to the early concept were proposed, among other, time-modeling and modeling of random behavior with statistic functions. In the process, many different types of PNs have been defined, and for historical reasons–at times–somewhat confusing terms were used. E.g., *Stochastic Petri Nets (SPNs)* were introduced in the 1980's as a formalism for the description of systems whose dynamic behavior can be represented by means of continuous-time homogeneous Markov chains. This original SPN proposal assumed atomic firings, exponentially distributed firing times and a race execution policy, that is, when multiple transitions are simultaneously enabled, the one transition with the minimum delay is selected to fire, compare to [Sah96]. Other authors, as [Sch99], argument, that in their opinion, the term *Stochastic Petri Net* should encompass any randomness in a PN, and not be confined to exponential firing delays only. In this text, SPNs shall not be limited in their randomness to exponential distributions. [Rei85] gives a good introduction to PNs and their mathematical foundation. Here, not the mathematic formalism of PNs is of major interest thus, but rather, their usability for expressing and simulating discrete-event phenomena.
The current PN library in Modelica implements PN components, namely, places and transitions, as strictly local sets of Boolean equations allowing for unifying treatment of both continuous and discrete components (hybrid models) in an object-oriented modeling language as Modelica, see [Mo98] for a more detailed description. The authors of the library chose their PNs to be *normal*[2], *priority*[3] *Petri Nets* with *maximum firing*[4] semantics. They show how their approach allows to seamlessly integrate discrete and continuous behavior e.g., as is required for comprehensive modeling of embedded control systems. Their focus is on process control, for which, they explain, the chosen semantics are appropriate.
However, when modeling problems in other areas, the limitation to places that can hold one token only and the immediate firing policy of enabled transitions can be a severe constraint. This paper therefore describes extensions to the available PN library which give a modeler more flexibility in modeling discrete-event phenomena. The paradigms of object-oriented physical modeling of Modelica are not hurt by doing so. The tight integration of discrete and continuous equations in Modelica allows to generate efficient execution code for a hybrid model (synchronous equations

[1] Modelica is a trademark of the Modelica Association (http://www.modelica.org)

[2] Places of capacity 1

[3] To eliminate otherwise inherent non-determinism in PNs

[4] Enabled transitions must fire immediately

principle, see [Ot99] and [El00]).
Many sophisticated, computer-based, PN modeling tools exist in the marketplace and can be engaged for a wide variety of analysis tasks, using both analytical as well as simulative methods. One such a tool was used e.g., in [Fa01] to show how PNs can be used to investigate influence of component maintenance policy on system availability. But usually, the respective standalone PN tools show inherent trouble when having to deal with systems exhibiting hybrid behavior. The new extended Modelica PN library shall enable modeling in a fashion quite similar to respective distinguished PN tools, at the same time allowing to integrate discrete behavior with continuous one by profiting from the capabilites and other existing libraries in Modelica.
This text first illustrates the structure of the extended PN library, then explains the characteristics of the new PN components, before giving several modeling examples.

2 Library Structure

Fig. 1 depicts the extended PN library with three sublibraries "Extensions", "ExamplesExtendedPetriNets" and "Modules" added to the original library.

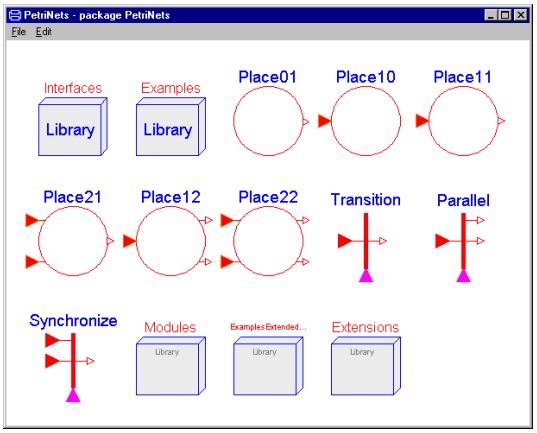

Figure 1: *PN library*

The contents of the first sub-library "Extensions" are shown in fig. 2 with a variety of new place and transition components, the second contains several models demonstrating the features of the new components and the third some compoposite models and utility blocks.

3 Transitions with Time Delay

The PN transitions as presented in [Mo98], fire, if all input places are marked and all output places are not. The new transitions presented here shall fire only after a certain time delay D has passed. In order to achieve

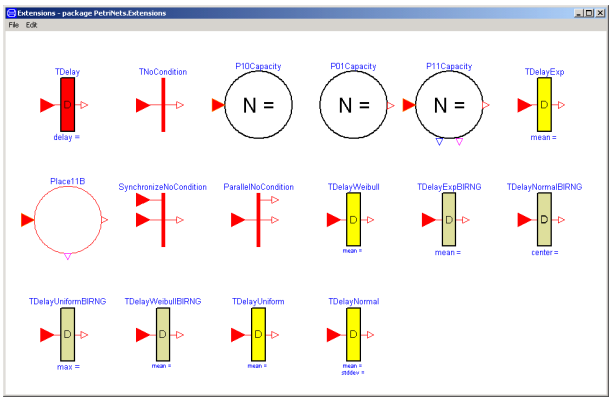

Figure 2: *PN library extensions*

such behavior, the firing process must be divided in two phases.

1. Activation: A transition is said to be *activated* (another term used is: *enabled*), if all of its input places are marked. At the time instant, the transition is activated, the delay process starts to run.

2. Actual firing: As soon as the time delay D–locally specified as a constant parameter or a variable of the transition–has passed, the transition can *fire*, if the transition is still activated and if all of its output places are free. If the output places are not free when the delay has passed, firing is either postponed until they are actually free at a later time, or firing will not occur because of loss of activation.

An important questions is, what happens, if one or all of the input places loose their marking while the delay process is running and regain it before the current delay time has passed. There is different ways in dealing with such situations. Here, a so-called *preemptive repeat different* (PRD) firing policy is used. This means, an interrupted job[5] will be repeated with a re-sampled delay time. The PRD policy will be illustrated with a simulation example later in this text.
Having clarified delayed firing semantics, attention is shifted to the implementation of the new delay transition in Modelica. Fig. 3 depicts an iconic representation of the new component with deterministic time delay (colored red).
Two new Boolean variables, *activated* and *delay_passed* are declared in the Modelica model of the transition. The second variable becomes "true", only if the respective transition was activated at run time for a time interval equal or longer than the delay time D. The

[5]Interrupted job signifiying a token removal from an in-place of a transition while it is activated and waits for its delayed firing

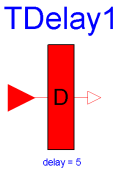

Figure 3: *Transition with deterministic time delay*

transition can fire if both new Boolean variables have a "true" value and if the output place is free. The respective Modelica code (somewhat abbreviated but containing all the important statements) is given below.

```
model TDelay
  parameter Real delay=5;

  Boolean activated;
  Boolean delay_passed;
  Boolean fire;
  Real last_activation_time;

equation
  //activation of transition
  activated = inTransition.state;

  //set activation time
  when activated then
    last_activation_time = time;
  end when;

  //activated and delay passed
  delay_passed = activated
         and ((time - delay) > last_activation_time);

  //fire command
  fire = activated and delay_passed
         and not outTransition.state;

  //propagate firing to in and output places
  inTransition.fire = fire;
  outTransition.set = fire;
end TDelay
```

For many problems to be modeled in the discrete-event domain, it is desirable to have access to time delays which have the characteristic of some probability distribution. Fig. 4 shows a transition which implements time delays realized as samples drawn from a random number generator (RNG) and transformed to behave as if they were exponentially distributed variables.

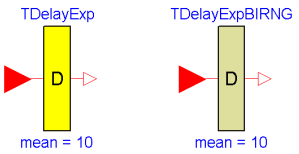

Figure 4: *Transitions with exponentially distributed delay*

Fig. 4 displays two variants, the transition on the left (yellow) implements a custom RNG, the transition on the right (greenish-yellow) makes use of the available, built-in RNG (acronym BIRNG) of the Modelica tool Dymola. In the yellow transition, a so-called linear congruential method (LCM) (see e.g., [Law00]), is used to generate pseudo random numbers x (eq. 1).

$$x_{k+1} = (ax_k) mod\, m, \quad k = 0, 1, 2 \ldots \tag{1}$$

The numbers produced are uniformly distributed. To obtain exponentially distributed numbers y with mean $1/\lambda$, the transformation

$$y = -\frac{1}{\lambda} ln(x) \tag{2}$$

can be carried out. The Modelica code for a transition with exponential distribution and customized RNG is shown below. As can be seen, each time the transition is becomes activated, a random number is generated and used as time delay D; seeds for the RNG can be set as parameter in each transition individually.

```
model TDelayExp
  parameter Real mean=10 "Exponential mean";
  parameter Integer seed=1973272912;

  Boolean activated;
  Boolean delay_passed;
  Boolean fire;
  Real last_activation_time;
  Real delay;
  Integer Xi(start=seed);
  Integer Xiplus;
  Integer m=(2^31) - 1;
  Integer a=7^5;

equation
  //activation of transition
  activated = inTransition.state;

  //activated and delay passed
  delay_passed = activated
        and ((time - delay) > last_activation_time);

  //fire command
  fire = activated and delay_passed
         and not outTransition.state;

  //propagate firing to in and output places
  inTransition.fire = fire;
  outTransition.set = fire;

  //set activation time
algorithm
  when activated then
    last_activation_time := time;
    Xiplus := mod(a*Xi, m);
    delay := -mean*ln(Xiplus/m);
    Xi := Xiplus;
  end when;
end TDelayExp
```

As mentioned, the tool Dymola, which is based on the Modelica modeling language specification, has a few built-in functions (e.g., "RandomUniform(time)") for random number generation. They are used in the transition variant on the right in Fig. 4, the respective code to call the RNG functions and the transition to exponential distribution is shown in the following code segment.

```
//set activation time
algorithm
```

```
when activated then
  last_activation_time := time;
  delay := -mean*ln(RandomUniform(time));
end when;
```

Test runs with models containing these Dymola BIRNG functions have miraculously produced, after a very large number of random samples were drawn, extremely large realizations for the random numbers ($x = 10^{300}$). The reason for this could not be determined since the source code of the random number generator could not be inspected.
That is one reason, why customized RNGs were implemented directly in the new library components. Of course, other methods than the LCM could be used for RNG as well, and a more sophisticated concept with special classes for RNG could be realized in Modelica. Currently, four probability distributions are implemented for use in transtions, namely, uniform, exponential, normal and Weibull.
The transformation to normal variates uses two independent uniform numbers x_1 and x_2 to yield one normal sample according to eq. 3 and eq. 4 (Box-Mueller method).

$$x_{normal,standard} = \sqrt{-2ln(x_1)cos(2\pi x_2)} \tag{3}$$

$$x_{normal} = mean + standardDev \cdot x_{normal,standard} \tag{4}$$

Given a uniform random number x, a Weibull variate x_W can be calculated with the transformation of eq. 5

$$x_W = -\frac{1}{\alpha} \cdot ln(1-x)^{1/\beta} \tag{5}$$

4 Multiple Tokens on Places

The original Modelica PN library implements places to hold only a single token, i.e., a place can either be marked or not (Boolean places). Such places are used in so-called C/E (condition/event) PNs.
If a PN was to serve as a model for e.g., flow and storage of numerous individual objects in a system, the need for places which can contain more than one token arises. The implementation of such places is discussed in this section. Fig. 5 displays the graphic representation of places which can contain N tokens at the maximum.

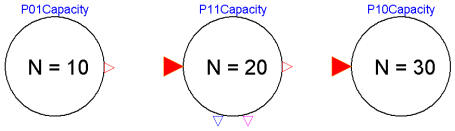

Figure 5: *Places capable of holding more than one token*

Naturally, the number of tokens present on a place at any time is of interest now. This number must be know, since it influences the firing of the transitions the respective place is connected to. A place which is *empty* (does not hold any tokens), can obviously not supply any tokens to transitions. In a similar fashion, a place which is *full* (the number of tokens has reached a certain locally defined limit) cannot receive any more tokens. The new place component therefore defines a capacity limit and several new variables e.g., for the number of tokens currently hold (Integer) and for empty and full states (Boolean). A possible implementation is given below (case of a place with one input and one output connector).

```
model P11Capacity
  parameter Integer num_tokens_start=0
            "Initial number of tokens";
  parameter Integer N=1 "Capacity limit for tokens";

  Integer num_tokens(start=num_tokens_start)
          "number of tokens present on the place";
  Integer new_num_tokens(start=num_tokens_start);
  Boolean full;
  Boolean empty;
  Boolean tokenin;
  Boolean tokenout;
  Boolean tokeninout;

equation
  // Set new token number for next iteration
  num_tokens = pre(new_num_tokens);

  tokenin    = inTransition.set and not full
               and not outTransition.fire;
  tokenout   = outTransition.fire and not empty
               and not inTransition.set;
  tokeninout = inTransition.set and outTransition.fire;

  when {tokenin,tokenout,tokeninout} then
    new_num_tokens = if edge(tokenin) then num_tokens + 1
      else if edge(tokenout) then num_tokens - 1 else num_tokens;
  end when;

  full = num_tokens == N;
  empty = num_tokens == 0;

  // Report state to input and output transitions
  inTransition.state  = full and not pre(tokenin)
                        and not pre(tokeninout);
  outTransition.state = not empty and not pre(tokenout)
                        and not pre(tokeninout );
end P11Capacity
```

The place in the middle of fig. 5 has two additional output ports

```
  outTokens.signal[1] = num_tokens;
  Bchange.signal[1] = not (num_tokens == new_num_tokens);
```

which propagate the current number of tokens hold in the place and a signal (Boolean) whenever the number changes. These signals are of value when collecting performance statistics during simulation runs as will be shown in the next sections.

5 Test Models

Some rather small PN models shall demonstrate the functionality implemented in the new PN components.

5.1 Deterministic Delay

The first test model consists of three places and two transitions and is depicted in fig. 6. It shall illustrate the delayed firing of a transition with time deterministic delay D. Place P1 is marked initially while P2 and P3 are not. The condition port of transition T1 obeys the relation "T1.condition = (time > 5.0)" and the delay of transition T2 is set to $D = 5.0$ time units (e.g., indicating seconds s).

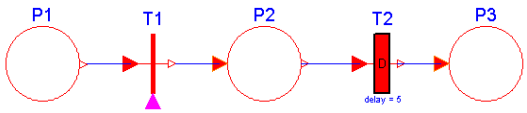

Figure 6: *Simple PN with deterministic delay*

The PN is simulated for a total of 15s. The marking of P1 is shown in fig. 7, its token is removed after 5.0s, when the external condition port is set to "true".

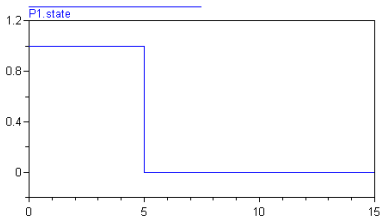

Figure 7: *Marking of place P1*

Place P2 receives a token when transition T1 fires at time $t = 5.0s$, at which point in time T2 is activated (see fig. 8. The delay process on T2 starts running and T2 can fire at time $t = 10.0s$ since it is still activated and its output place P3 is empty. Fig. 9 illustrates the Boolean variable "activated" of transition T2.

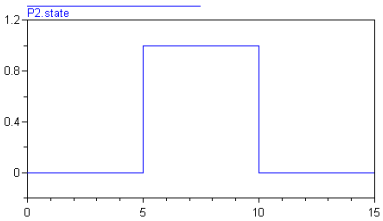

Figure 8: *Marking of place P2*

5.2 Preemptive Repeat Different Firing Policy

The model in fig. 10 shall demonstrate the PRD firing policy of transitions with time delay.
The following rules apply to the external condition ports of transitions T1, T3 and T4.

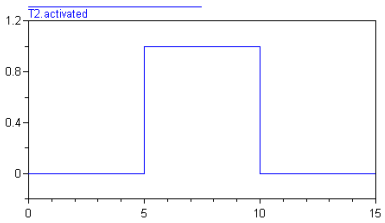

Figure 9: *Boolean variable "activated" of transition T2*

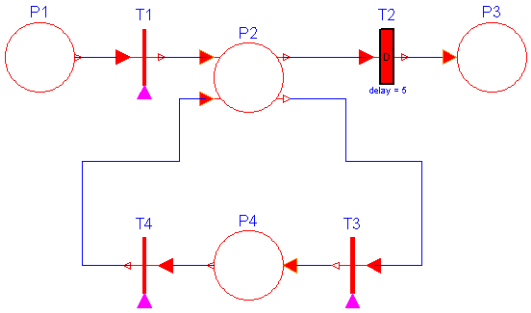

Figure 10: *Job interruption model*

```
T1.condition = (time > 5);
T3.condition = (time > 7.5) and (time < 8.0);
T4.condition = (time > 12.5);
```

Place P1 is marked initially. A simulation run of 20.0s is made. As can be seen in fig. 11 (marking of place P2), transition T1 fires at time 5.0s, at which point in time T2 and T3 become both activated. The delay on T2 starts running (the delay is D=5s), but T3 fires before the delay of T2 is over, at time 7.5s. As soon as the token leaves P2, transition T2 is no longer activated. After the firing of T3 and T4, the token is again present on place P2 and T2 is newly activated. The delay on T2 starts running once more and T2 fires at time 12.5s (this time, T3 cannot fire anymore because its external condition port is set to false after time $t \geq 8.0s$.

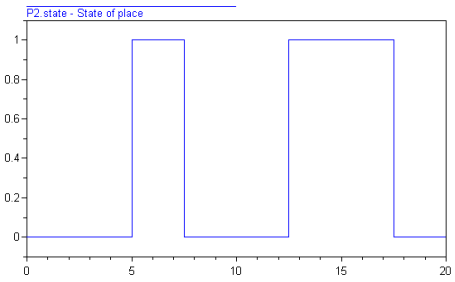

Figure 11: *Marking of place P2*

5.3 Capacity Limits On Places

Fig. 12 depicts a PN with capacity limits of 10, 100 and 10 tokens, respectively, on places P1, P2 and P3.

Initially, P1 carries 10, P2 5 and P3 2 tokens.

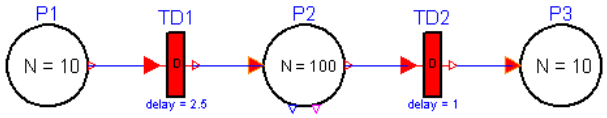

Figure 12: *Net with multiple tokens on places and limits*

Fig. 13 indicates the markings of the three places. It can be seen, how repeteatedly after a delay of 2.5s, a token is removed from P1 and added to P2. In parallel, transition T2 fires after delays of 1.0s until time $t = 7.0s$ when P2 reaches empty state. At $t = 7.5s$, P2 receives a token from T1, which is passed on to T2 when another delay of 1.0s has passed. At time $t = 8.5$, P3 becomes full and T2 cannot fire anymore, which in turn leads to an increase in the number of tokens on P2 in the following. As can be seen, the example net behaves as demanded by the stated firing semantics.

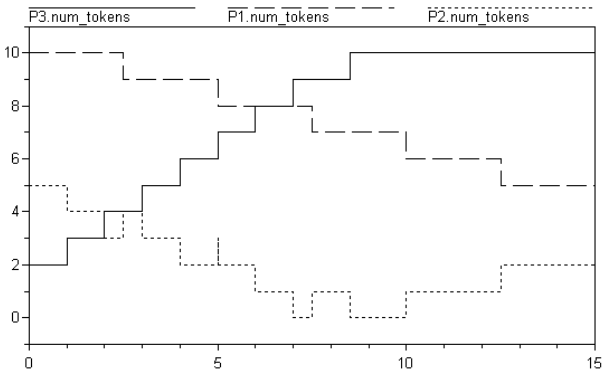

Figure 13: *Markings of places P1,P2 and P3*

6 Example Modeling Applications With the New PN Components

This section gives examples of modeling and simulations with the new PN components in two problem domains. The first deals with queuing systems, the second with availability analysis of systems of renewable components.

6.1 M/M/1 Queuing System

An *M/M/1*[6] queuing system is modeled as a PN, see fig. 14; in reality, it could be a ticket shop selling tickets for a bus ride, with customers arriving at certain points in time and waiting in line–if necessary–to be served at the desk.
In the PN model, the transition "Interarrival" represents the arrival process as a delay of exponential characteristic (delay signifies inter-arrival times). The place "Queue" denotes the line of waiting customers and transition "Service" models the duration of service again as an exponential stochastic process. The block in the lower half of the model collects the current number of tokens on the place representing the queue, and it recalculates the average queue length as a weighted time average while the simulation progresses at runtime.

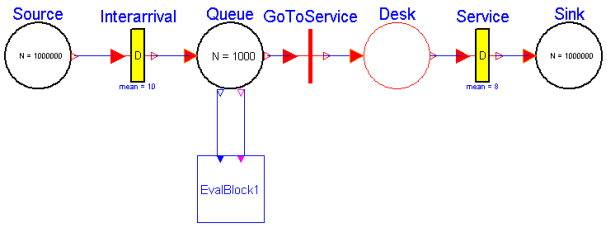

Figure 14: *Queuing model*

The behavior of such an M/M/1 queue can be investigated analytically. With arrival rate λ and service rate μ, the server utilization ρ is

$$\rho = \lambda/\mu. \tag{6}$$

According to the formula of Little, in the long run (steady-state performance), the average queue length for the M/M/1 system is

$$L_{queue} = \rho^2/(1-\rho). \tag{7}$$

In the above model, $\lambda = 1/10$ and $\mu = 1/8$, therefore, the expected value for queue length is $L_{queue} = 3.2$.
Fig. 15 displays the development of average queue length, it can be seen, the value stabilizes after about 2500 time units. The end value after 100'000 time units of simulation time is rather close to the analytically computed value. Note, the value depicted in fig. 15 is a so-called running average, i.e., it gives the current average queue length at a particular point in time during the simulation, accounting for the queue history from the beginning of the simulation until the current simulation time. Only the final value at the end of the

[6] "M" stands for the memoryless characteristic of the exponential distribution. The first "M" denotes the arrival process, the second "M" the service process and the "1" indicates the existence of one server only.

simulation serves as an estimate of the long-run performance of the system (standard simulation practice).

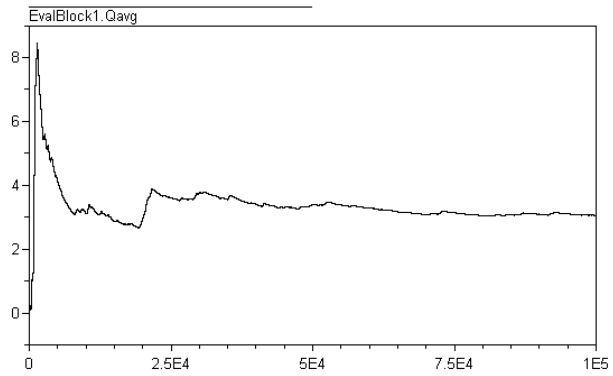

Figure 15: *Average queue length evolution during the simulation*

Of course, in order to investigate a system by means of simulation, it is usually not sufficient to carry out just one single simulation run. Often, several sequential batch runs and independent replications (with different random number streams) are carried out to produce multiple realizations of performance measures which are then manipulated statistically to obtain a point estimate with respective confidence interval, see textbooks on simulation topics, e.g., [Ban96], [Law00]. Simulation experiment parameters as simulation duration, initial transient phase (which is not used for performance measure collection), number of batches and replications must be chosen appropriately. However, in this text, the feasibility of the PN models for discrete-event simulation is of interest, not the very details of simulation experiments.
Fig. 16 is a zoom of the time window with the largest queue length that occurred during this particular simulation, maximum queue length is 26 customers in this case.

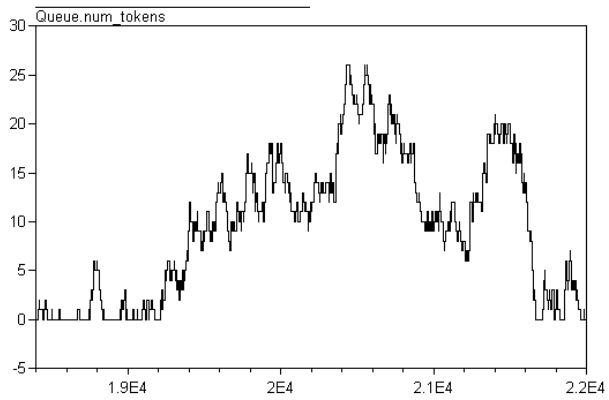

Figure 16: *Queue length*

6.2 Steady-State Availability Simulation

In this paragraph, system availability is investigated in a quantitative manner. If the failure behavior of the components installed in a system and their interaction is known and modeled, the overall system characteristics can be analyzed. Two examples of renewable systems are given and their availability is determined analytically and by carrying out steady-state[7] simulation experiments. The long run performance of the systems is of interest here, the so-called stationary availability[8].

6.2.1 System of Two Components Combined in Series

First, a combination of two components, arranged serially, is investigated, a practical configuration in reality may be two pumps installed serially in a pipe. The functional state of each of the two components can be modeled with a PN as depicted in fig. 17, with a functional and a nonfunctional state active when the respective place is marked. The transitions represent failure and repair processes modeled as exponential time delays (lifetime and repairtime).

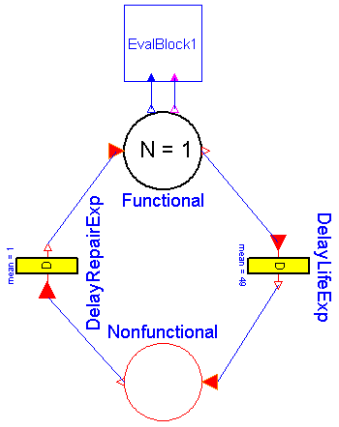

Figure 17: *Component with two states*

A Model of the serial system is given in fig. 18. The blocks “C1” and “C2”, at the bottom of the model representation, contain individual components as shown above in fig. 17, with an additional Boolean output connector propagating the component state (“true”=functional). The system state is given by markings of the two places in fig. 18, which depend on a combinatory logic of the two individual component states.

[7] As opposed to “terminating” simulation experiments, see [Ban96] and [Law00].

[8] E.g., the German industrial norm DIN 40041 also defines a transient availability as the probability that a system is in functional state at a certain point in time.

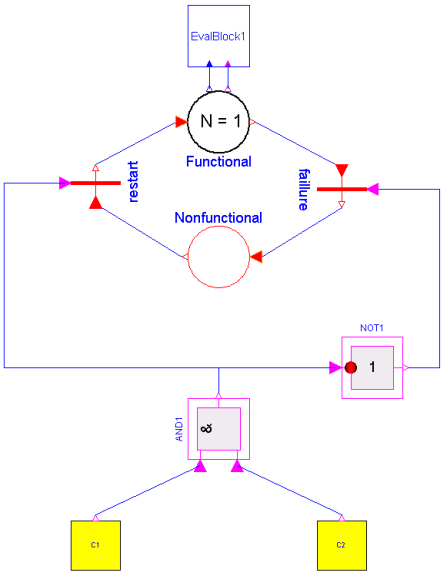

Figure 18: *Serial arrangement of two components*

The availability of such a system can be computed analytically with Markov theory. If the component failure rates are known (e.g., by collection and analysis of historic data in an industrial production environment) and the failure and repair characteristic is exponential with failure rate $\lambda = \frac{1}{49}$ and repair rate $\mu = 1$, then the system availability can be calculated to (see appendix A)

$$A_{2serial,renewable} = 0.9604.$$

In the PN model, the steady-state system availability A is calculated as running average during the simulation (usage of evaluation block from the "Modules" sub-library), in particular as the ratio between system up-time t_{up} and total time t_{total} available for system operation

$$A = \frac{t_{up}}{t_{total}} \tag{8}$$

with

$$t_{total} = t_{up} + t_{down}. \tag{9}$$

The trajectory of the availability performance measure is illustrated for a simulation run of 15'000 time units in fig. 19, the numerical value obtained at the end of the simulation run is one single estimate for expected system availability. As can be seen, the experimentally determined value is very close to the analytical result.

6.2.2 Two-out-of-Three System

The second availability simulation is carried out with a *two-out-of-three* system. Systems of this type are well

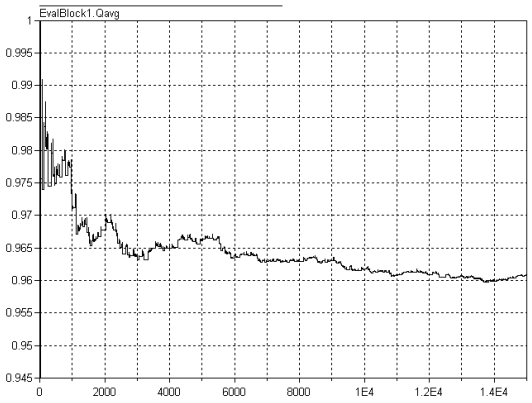

Figure 19: *Stabilizing availability measure*

know in reliability engineering, and are functional if at least two of the three components of the system are in working condition. A realization with components from the Modelica library is shown in fig. 20.

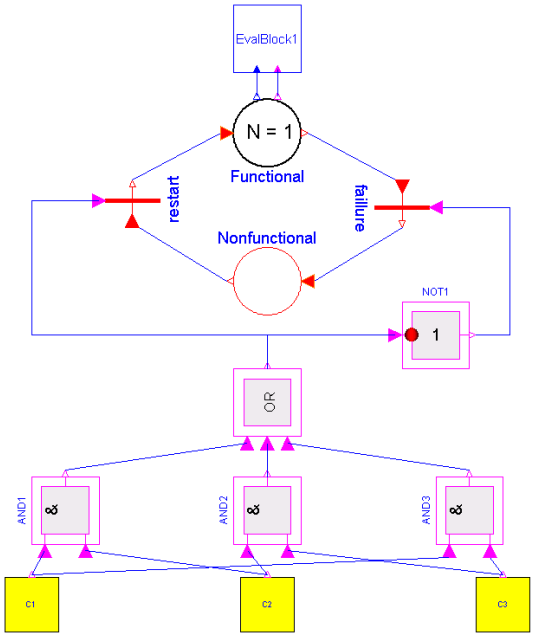

Figure 20: *Two-out-of-three system*

Analytical computation of the steady-state availability is given in appendix B. For $\lambda = 1/9$ and $\mu = 1$ (per definition, in an *m-out-of-n* system, the components have identical characteristic), system availability is

$$A_{203,renewable} = 0.972.$$

Fig. 21 depicts the running average of system availability for a run of 20'000 time units. Again, it can be seen that the simulation result is close to the analytically computed value.

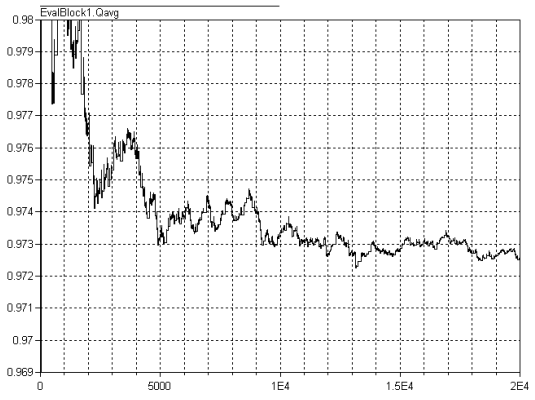

Figure 21: *Two-out-of-three system availability*

In [Fa01], it has been shown in more detail, how PN models can be used for simulative investigation of system availability. A stand-alone, computer-based, purely discrete-event PN simulator was used. With the PNs shown above, in elegant combination with the combinatory logic blocks available in the standard Modelica library, it is possible to seamlessly couple the PNs with dynamic, continuous-time physical models of any desired level of detail. This can greatly support the analysis of e.g., reliability, availability, safety, dependability or other performance measures of complex technical systems without having to neglect system dynamics, as is inherently the case with many established tools know in reliability engineering as e.g., fault or event trees.

7 Conclusion and Outlook

This paper has shown how the available, standard PN library in Modelica can be extended with semantics as known from timed, stochastic PNs. New PN components were introduced, allowing to generate PNs with delayed transition firing (deterministic and stochastic delay) and multiple tokens on places. As in the available library, the new components are defined strictly by local constraint equations, therefore living up to the principles of object-oriented system modeling in Modelica. Seamless integration with continuous-time model parts in one unique simulation environment is therefore possible and provides the basis for hybrid simulation.
It could be investigated in additional work, how token attributation could be included as well, hence supporting *colored* PNs. This would demand for appropriate data structures and management on places together with correct exchange of tokens via connectors, if feasible and consistent with the local equation paradigm. The idea hereby would not be to imitate all the features of stand alone, sophisticated PN modeling tools. It would be hard, if not impossible to include, e.g., analytical analysis capabilities or token game animation of such tools. Rather, the hybrid capabilities in the Modelica environment would be of main interest.

References

[Ban96] Banks, Jerry; Carson, John S.; Nelson, Barry L. (1996). *Discrete-Event System Simulation*. 2nd ed., Prentice Hall, New Jersey, U.S.A.

[El00] Elmqvist H., Mattsson S. E., Otter M. (2000) *Object-Oriented and Hybrid Modeling in Modelica*. ADPM 2000, Dortmund, Germany

[Fa01] Fabricius S.M.O., Badreddin E. (2001). *Stochastic Petri Net Modeling for Availability and Maintainability Analysis*. Proceedings of 14th International Congress and Exhibition on Condition Monitoring and Diagnostic Engineering Management (COMADEM), September 2001, Manchester, UK

[Law00] Law Averill M., Kelton W.David (2000). *Simulation Modeling and Analysis*. 3rd ed., McGraw-Hill, Boston, U.S.A.

[Ot99] Otter M., Elmqvist H., Mattsson S.E. (1999). *Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle*. 1999 IEEE Symposium on Computer-Aided Control System Design, CACSD'99, pp. 151-157, August 22-27, 1999, Hawaii, U.S.A.

[Mo98] Mosterman P.J., Otter M., Elmqvist H.: (1998). *Modeling Petri Nets as Local Constraint Equations for Hybrid Systems Using Modelica*. The Proceedings of the Summer Computer Simulation Conference, July 19-22, S. 314-319, 1998 Reno, Nevada, U.S.A.

[Pet62] Petri, C.A. (1962). *Kommunikation mit Automata*. PhD thesis University of Bonn, Bonn, West Germany

[Rei85] Reisig, W. (1985). *Petri Nets: An Introduction*. Springer-Verlag, Berlin, Germany

[Sah96] Sahner, Robin A.; Trivedi, Kishor S.; Puliafito, Antonio (1996). *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*. Kluwer Academic Publishers, Boston, U.S.A.

[Sch99] Schneeweiss, W.G. (1999). *Petri Nets for Reliability Modeling (in the Fields of Engineering Safety and Dependability)*. LiLoLe-Verlag GmbH, Hagen, Germany

A Analytical Computation of the Availability of a Serial System of Two Components

Two components A and B combined in series can be modeled by the Markov diagram in figure 22 (capital letters "A" and "B" indicate functional components, small letters "a" and "b" indicate non-functional components).

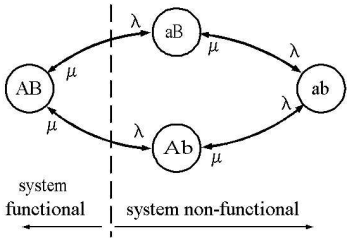

Figure 22: *Markov diagram, detailed (2 serial)*

For the case of identical components, the Markov model can be reduced to the diagram in figure 23.

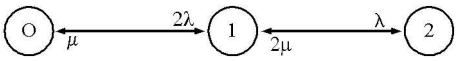

Figure 23: *Markov diagram, compact (2 serial)*

The equations for the renewable system state probabilities are

$$\frac{dP_0}{dt}(t) = -2\lambda P_0(t) + \mu P_1(t) \quad (10)$$
$$\frac{dP_1}{dt}(t) = 2\lambda P_0(t) - (\mu + \lambda) P_1(t) + 2\mu P_2(t) \quad (11)$$
$$\frac{dP_2}{dt}(t) = \lambda P_1(t) - \mu P_2(t) \quad (12)$$

and the availability is

$$A_{2serial,renewable}(t) = P_0(t) \quad (13)$$

The asymptotic availability or steady-state availability for time $t \to \infty$ can be calculated solving the following set of linear algebraic equations.

$$0 = -2\lambda P_0(\infty) + \mu P_1(\infty) \quad (14)$$
$$0 = 2\lambda P_0(\infty) - (\mu + 2\lambda) P_1(\infty) + 2\mu P_2(\infty) \quad (15)$$
$$0 = \lambda P_1(\infty) - \mu P_2(\infty) \quad (16)$$

and the availability is

$$A_{2serial,renewable}(t) = P_0(t) \quad (17)$$

The asymptotic availability or steady-state availability for time $t \to \infty$ can be calculated solving the following set of linear algebraic equations.

$$0 = -2\lambda P_0(\infty) + \mu P_1(\infty) \quad (18)$$
$$0 = 2\lambda P_0(\infty) - (\mu + 2\lambda) P_1(\infty) + 2\mu P_2(\infty) \quad (19)$$
$$0 = \lambda P_1(\infty) - \mu P_2(\infty) \quad (20)$$

to yield the expression

$$A_{2serial,renewable}(\infty) = \frac{\mu^2}{\mu^2 + 2\mu\lambda + \lambda^2} \quad (21)$$

For $\lambda = \frac{1}{49}$ and $\mu = 1$, the state probabilities are

$$P_0(\infty) = 0.9604, P_1(\infty) = 0.392, P_2(\infty) = 0.004$$

and the steady-state system availability is therefore

$$A_{2serial,renewable} = 0.9604.$$

B Analytical Computation of the Availability of a Two-out-of-Three System

The detailed as well as the reduced Markov diagrams for the 2-out-of-3 system are shown in figures 24 and 25.

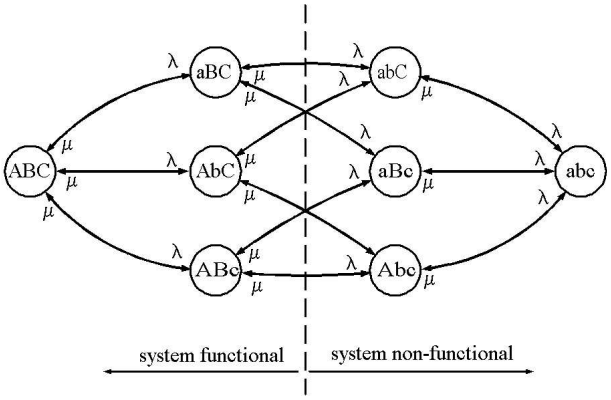

Figure 24: *Markov diagram, detailed*

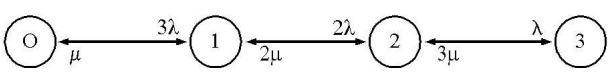

Figure 25: *Markov diagram, compact 2-out-of-3 system*

The equations for the state probabilities of the renewable system are

$$\frac{dP_0}{dt}(t) = -3\lambda P_0(t) + \mu P_1(t) \quad (22)$$
$$\frac{dP_1}{dt}(t) = 3\lambda P_0(t) - (\mu + 2\lambda)P_1(t) + 2\mu P_2(t) \quad (23)$$
$$\frac{dP_2}{dt}(t) = 2\lambda P_1(t) - (2\mu + \lambda)P_2(t) + 2\mu P_3(t) \quad (24)$$
$$\frac{dP_3}{dt}(t) = \lambda P_2(t) - 3\mu P_3(t) \quad (25)$$

The availability again is

$$A_{203,renewable}(t) = P_0(t) + P_1(t) \quad (26)$$

and the asymptotic availability for time $t \to \infty$ can be calculated solving the following set of linear algebraic equations.

$$0 = -3\lambda P_0(\infty) + \mu P_1(\infty) \quad (27)$$
$$0 = 3\lambda P_0(\infty) - (\mu + 2\lambda)P_1(\infty) + 2\mu P_2(\infty) \quad (28)$$
$$0 = 2\lambda P_1(\infty) - (2\mu + \lambda)P_2(\infty) + 2\mu P_3(\infty) \quad (29)$$
$$0 = \lambda P_2(\infty) - 3\mu P_3(\infty) \quad (30)$$

to yield

$$A_{203,renewable}(\infty) = \frac{\mu^2(3\lambda + \mu)}{3\mu^2\lambda + \mu^3 + 3\lambda^2\mu + \lambda^3} \quad (31)$$

For $\lambda = \frac{1}{9}$ and $\mu = 1$, the state probabilities are

$$P_0(\infty) = 0.729, P_1(\infty) = 0.243, P_2(\infty) = 0.027, P_3(\infty) = 0.001$$

and the steady-state system availability is therefore

$$A_{203,renewable} = 0.972.$$