A Concept for the Modular Description of Stochastic Petri Nets

(Extended Abstract)

Reinhard German
Technische Universität Berlin, Franklinstr. 28/29, 10587 Berlin, Germany
e-mail: rge@cs.tu-berlin.de

1 Introduction

Petri nets represent a modeling paradigm which allows to build general models by using a small number of graphical primitives. The strong point of Petri nets lies in their ability to model system aspects such as concurrency and synchronization and to represent these aspects graphically. Due to this reason, *stochastic Petri nets* (SPNs) are frequently used for model-based performance and dependability evaluations. However, the main drawback is caused by the problem of complexity, both of model specification and of model evaluation. A number of approaches have been suggested in order to manage the complexity problem, e.g., *generalized stochastic Petri nets* [1], *stochastic reward nets* [4], *stochastic activity networks* [10], *coloured Petri nets* [7], and *stochastic well-formed nets* [3]. In PROTOB [2], Pascal-like language constructs have been added to Petri nets in order to describe models in a modular fashion. In the recent years there has been considerable interest in applying object-oriented technology to Petri nets [8, 9]. Another direction of research is to use operators from process algebras for the composition of SPNs, e.g., [5].

In this paper the language SPNL (SPN Language) is presented which is designed for the modular description of SPN performance and dependability models. The intention is to define a language which can be used throughout the entire process of modeling. Such a language should provide a high modeling power by means of few primitives with an easily understandable semantics. Concepts for the structured description of models should allow to apply methods known from software engineering (e.g., stepwise refinement of models or reuse of models). Furthermore, a unique description is desirable both for more abstract models subject to numerical analysis and for more detailed models subject to simulation.

In the proposed framework, an SPN is considered as a *process*. Such a process is composed of other processes and of ordinary SPN elements (places, transitions, arcs). Processes do mainly interact via *ports*. Ports are Petri net arcs crossing the boundary between the inside and outside of a process and can be interpreted as synchronous (unbuffered) communication links. Processes and ports are just syntactical constructs, they do not represent new Petri net primitives. A second possibility of process interaction is given by the *fusion* of places and of transitions. The public part of a process is given by its ports, and by externally visible variables, refered to as *rewards* and *measures*. Processes can either be declared as single entities or by means of process types and process instantiations. A module is given by a collection of declarations in a separate file. A compiler is used to translate an SPNL model description into a machine representation suitable for analysis and simulation.

SPNL provides additional linguistic constructs which are especially tailored to performance and dependability modeling. Detailed properties of transitions can be specified (e.g., general firing time distributions, preemption policies, degree of concurrency, weights for conflict resolution, …). The concept of rate and impulse rewards is used both for the definition and for the observation of model properties. Rewards are parameter- and marking-dependent expressions which allow a flexible specification of model properties (e.g., for marking-dependent firing rates, in guards, …). They can be observed from the outside of a process and allow to define stochastic result measures (e.g., expectation of a reward expression). The possibility of using reward variables of one process in another process represents a third mechanism of process interaction. SPNL also provides a formal framework for defining hierarchical and iterative models by passing measures as parameters between processes (in case of cyclic parameter dependencies this gives rise to fixed-point iteration).

SPNL is a mixed textual and visual modeling language. The basic structure is textual and given by modules and processes. The syntax is motivated by programming languages like Pascal, Modula-2, or Ada. The block concept and visibility rules are taken from these languages. Inside the processes their internal structure can be graphically represented (the places, transitions, process instances, arcs, ports). Other language elements are represented textually (e.g., parameters, rewards, distributions, …). This leads to a balance between graphical and textual language elements. Furthermore, each textual element has a well-defined location in the description, thus a confusing “overloading” of figures with textual inscriptions is avoided. Another important aspect of SPNL is the clear distinction between syntax and semantics of the language. The semantics is defined by an ordinary SPN without need of any additional primitives. Furthermore, the semantics is defined for the language and does not depend on any tool environment. Due to this aspect SPNL (or a variant of it) would be well suited to serve as an exchange format between different SPN tools.

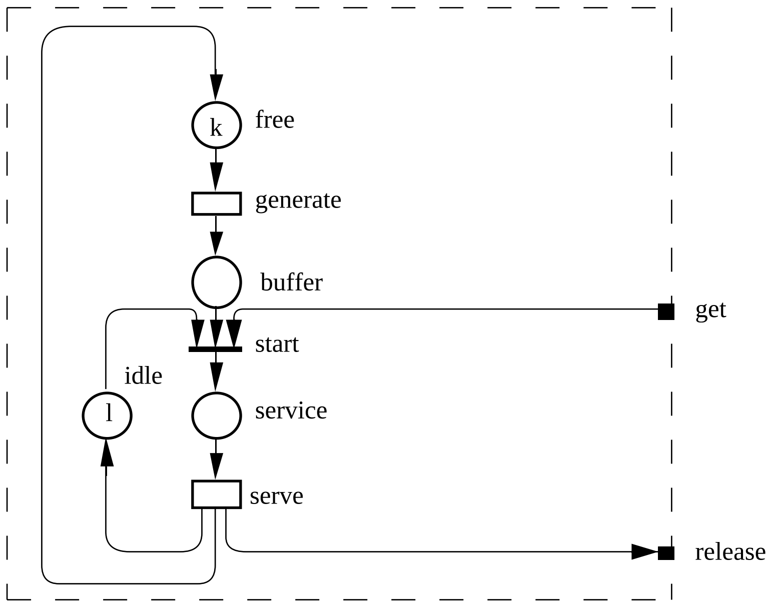
The main purpose of SPNL is to facilitate the specification of complex SPN models. The language also allows to exploit the structure of the model in the evaluation: lumping in case of model symmetries, structured generator matrix descriptions, approximation by fixed-point iteration, or the use of locality of model behavior in discrete-event simulations.

A tool for modeling with SPNL is currently under development. It will contain graphical and textual editors for the model specification and a compiler for the conversion of the description into an internal representation. Components will be provided for the animation and evaluation by numerical analysis and by discrete-event simulation. A further subject of research is the adaption of efficient evaluation techniques combined with automatic detection mechanisms whether these techniques can be applied to a given model.

2 Modeling examples

A queueing system consisting of *n* M/M/l/k queueing systems which share a pool of *m* servers is considered. In the following this system is described in SPNL in a modular fashion. The description is divided into two *modules*. Figure 1 shows the module "queueing" which contains a *process type* "mmlk". Figure 2 shows the main module "sharing" which describes the entire system. The process type in module "queueing" is a description of one M/M/l/k queueing system. The process type *declaration* contains a *formal parameter list*, a *public part*, and a *private part*. In the public part, externally visible objects are declared. Such public objects may be *reward* and *measure variables* as well as *ports*. Ports may be considered as cut arcs which have to be connected with suitable places or transitions in the external world. In each port declaration its type has to be given (e.g., "**t<-p**" indicates that an input arc to an internal transition is crossing the boundary).

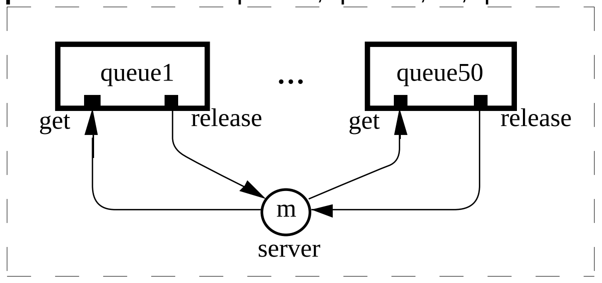
```
module queueing;
    (* module providing a process type for an M/M/l/k queueing system *)
    process type mmlk(l, k : integer; arr, ser : real);
        port        get : t<-p;
                    release : t->p;
        measure     S, N, W : real;
    private
```



```
        transprop   generate : dist = exp(arr);
                    serve : dist = exp(ser), concurrency = is;
        measure     S = E{#generate}; N = E{#buffer+#service}; W = N/S;
    end mmlk;
end queueing.
```

Figure 1. Module "queueing"

```
main module sharing;
    use queueing;
    parameter l = 10; k = 5; m = 20; a = 2.0; s = 1.0;
    process alltogether;
        measure W : real;
    private
        process instance queue1, queue2, …, queue50 : mmlk(l,k,a,s);
```



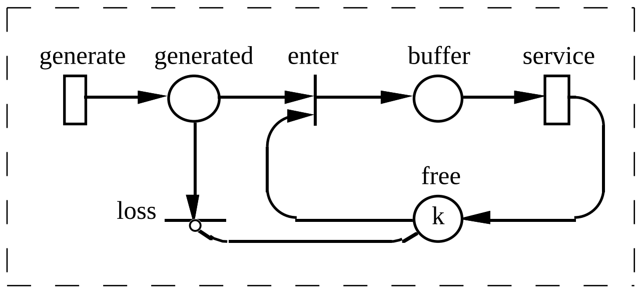
```
        measure W = (queue1.W+…+queue50.W)/50;
    end alltogether;
end sharing.
```

Figure 2. Main module "sharing"

The private part contains the combined graphical and textual representation of the internal structure of the process. In the example, the graphical part consists of customary Petri net elements. The ports which have been declared in the public part are visualized as small black squares. The graphical part can be augmented by textual statements: the firing times of transition "generate" are exponentially distributed, the rate is given by the formal parameter "arr". Transition "serve" also has an exponential firing time distribution with rate "ser", its degree of concurrency is infinite-server. Measure expressions are based on reward expressions. In the example, "S" is the throughput (expected value of an impulse reward), "N" is the mean queue length (expected value of a rate reward), and "W" is the mean waiting time.

Declarations in other modules can be made visible by *imports*. This leads to a hierarchy of modules with a main module on top of this hierachy. The process declaration in main module "sharing" of Fig. 2 is similar to a process type declaration and represents the top level of the model. The private part contains the declaration of process instances. An instance is shown in the graphical part as a rectangle with thick lines. The ports are depicted as small black squares and are connected by Petri net arcs. In the public part, again a measure "W" is declared. In the private part, "W" is defined as the arithmetic mean of the measures "W" of the process instances. Reference to those measures is given by instance name qualifiers. In the given example, "W" is the only variable which can be observed from outside (in a tool enevironment, the value of this variable could be shown as a curve during transient analysis, iterative stationary analysis, or simulation).

main module protocol_layers;
    parameter n = 10; arr2 = 0.5; buf2 = 10; ser1 = 1; buf1 = 20;
    process type mm1k(k : **integer**; arr, ser : **real**);
        measure S, W : **real**;
    private



        transprop   generate : **dist** = **exp**(arr);
                         serve : **dist** = **exp**(ser);
        measure   S = **E**{#enter};
                         W = **E**{#buffer}/S;
    end mm1k;
    (* two interdependent main processes: *)
    process instance layer2 : mm1k(buf2, arr2, 1/layer1.W);
    process instance layer1 : mm1k(buf1, n*layer2.S, ser1);
end protocol_layers.

Figure 3. Main module "protocol_layers"

Another possibility of structuring models is to decompose them into parts and to pass measures as parameters between them. This leads to hierachical or iterative models. To illustrate this, two layers of a communication protocol are considered. Packets arrive at several queues of layer 2 and are passed to one queue of layer 1. The service time of each queue at layer 2 depends on the waiting time in the queue of layer 1 and the arrival rate to the queue at layer 1 depends on the throughput of the queues at layer 2. A description is shown in Figure 3.

The model is described within one single main module "protocol_layers". A process type "mm1k" represents an M/M/1/K queueing system. Measures throughput "S" and mean waiting time "W" are public. Two processes "layer2" and "layer1" are declared as instances of the process type representing two SPN models. Measures of both processes are mutually used as actual parameters. Thus, an iterative model is represented which has to be solved by fixed-point iteration.

SPNL offers a variety of additional linguistic constructs: *composed* processes can be defined which share either *fusion places* or *fusion transitions*. An extension to coloured tokens is straightforward. More examples and a definition of the syntax and semantics of SPNL is provided in [6].

Many extensions of the language are possible. Potential extensions are for instance: object-orientation, process algebra operators, structured ports and arcs, mixed discrete and continuous state space.

References

[1] M. Ajmone Marsan, G. Balbo, G. Chiola, S. Donatelli, G. Franceschinis. *Modeling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

[2] M. Baldarassi, G. Bruno. PROTOB: An Object Oriented Methodology Based on Hierachical Colored Petri Nets. In K. Jensen, G. Rozenberg (Eds.): *High-level Petri Nets, Theory and Application.* Springer-Verlag, pp. 624–648, 1991.

[3] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad. Stochastic Well-Formed Coloured nets for Symmetric Modelling Applications. *IEEE Trans. on Comp.* **42** (1993) 1343–1360.

[4] G. Ciardo, A. Blakemore, P.F. Chimento, J.K. Muppala, and K.S. Trivedi. Automated Generation of Markov Reward Models using Stochastic Reward Nets. In: C. Meyer and R.J. Plemmons (Eds.), *Linear Algebra, Markov Chains, and Queueing Models*, Vol. 48 of IMA Volumes in Mathematics and its Applications, Springer Verlag, 1993.

[5] S. Donatelli. Superposed Generalized Stochastic Petri Nets: Definition and Efficient Solution. *Proc. 15th Int. Conf. on Application and Theory of Petri Nets,* pp. 258–277, Zaragoza, Spain, 1994, LNCS 815, Springer-Verlag.

[6] R. German. SPNL: A Modular Description Language for Stochastic Petri Nets. Internal Paper. Technische Universität Berlin, 1996.

[7] K. Jensen. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use.* Vol. 1, Springer-Verlag, 1992.

[8] C. Lakos. From Coloured Petri Nets to Object Petri Nets. *Proc. 16th Int. Conf. on Application and Theory of Petri Nets,* LNCS 935, pp. 278–297, Torino, Italy, Springer-Verlag.

[9] *Proc. 1st and 2nd Workshop on Object-Oriented Programming and Models of Concurrency*, Torino, Italy, 1995 and Osaka, Japan, 1996 (in conjunction with the Int. Confs. on Application and Theory of Petri Nets).

[10] W. H. Sanders, J. F. Meyer. A Unified Approach for Specifying Measures of Performance, Dependability, and Performability. *Dependable Computing for Critical Applications*, Vol. 4, pp. 215–238, Springer-Verlag, 1991.