Extending the Interface between the Modeling Languages MOSEL and CSPL by Adding Simulation Constructs

Semester Thesis in Computer Science

written by
Patrick Wüchner
born June 12th 1976 in Kronach

Department of Computer Science
(Distributed Systems and Operating Systems)
University of Erlangen-Nürnberg

Advisors: **Dipl.-Inf. Jörg Barner**
Prof. Dr. rer. nat. Fridolin Hofmann

Begin: October 1st 2002
Submission: June 30st 2003

Copyright © 2003 Patrick Wüchner.

Permission is granted to copy and distribute this document provided it is complete and unchanged.

Parts of this work may be cited provided the citation is marked and its source is referenced.

Contents

1 Introduction **1**

2 SPNP: Stochastic Petri Net Package **3**

2.1 Petri Nets . . . 3

2.1.1 Petri Nets (untimed) . . . 3

2.1.2 Stochastic Petri Nets . . . 4

2.1.3 Generalized Stochastic Petri Nets . . . 4

2.1.4 Deterministic Stochastic Petri Nets . . . 5

2.1.5 Extended Stochastic Petri Nets . . . 5

2.1.6 Fluid Stochastic Petri Nets . . . 5

2.1.7 High Level Petri Nets . . . 5

2.2 CSPL: C-based Petri Net Language . . . 5

2.2.1 options()-Part . . . 5

2.2.2 net()-Part . . . 6

2.2.3 assert()-Part . . . 7

2.2.4 ac_init()-Part . . . 7

2.2.5 ac_reach()-Part . . . 7

2.2.6 ac_final()-Part . . . 7

2.3 Evaluation Methods . . . 9

2.3.1 Numerical Analysis . . . 9

2.3.2 Discrete Event Simulation . . . 9

2.4 Distributions . . . 11

2.4.1 Beta Distribution . . . 11

2.4.2 Binomial Distribution . . . 11

2.4.3 Cauchy Distribution . . . 12

2.4.4 Deterministic Distribution . . . 12
2.4.5 Erlang Distribution . . . 13
2.4.6 Gamma Distribution . . . 13
2.4.7 Geometric Distribution . . . 14
2.4.8 Hyperexponential Distribution . . . 14
2.4.9 Hypoexponential Distribution . . . 15
2.4.10 Lognormal Distribution . . . 15
2.4.11 Normal Distribution . . . 16
2.4.12 Pareto Distribution . . . 16
2.4.13 Poisson Distribution . . . 17
2.5 Resampling . . . 17
2.5.1 Preemptive Repeat Different (PRD) . . . 18
2.5.2 Preemptive Repeat Identical (PRI) . . . 18
2.5.3 Preemptive ReSume (PRS) . . . 19

3 MOSEL: MOdeling, Specification and Evaluation Language 21
3.1 Introduction to MOSEL . . . 21
3.2 Modeling and Evaluating using MOSEL . . . 21
3.2.1 Building the Conceptual Model . . . 22
3.2.2 Describing the Conceptual Model as MOSEL Model . . . 22
3.2.3 Invoking the MOSEL-2 Modeling Environment . . . 24
3.2.4 MOSEL-2 working . . . 25
3.3 Restrictions of (old) MOSEL . . . 26

4 Changes to MOSEL-2 29
4.1 Command Line Parameters . . . 29
4.2 Result Parser . . . 29
4.3 Introducing new Distributions . . . 30
4.3.1 Incrementing Number of Parameters . . . 30
4.3.2 Enabling Solving of DSPNs by SPNP Simulation . . . 30
4.3.3 Introducing new ESPN distributions to MOSEL-2 . . . 30
4.4 Providing Resampling Policy PRI . . . 31
4.5 Blocking new Constructs from normal SPNP mode and other Tools . . . 31
4.6 Updating Model Dumping Function . . . 32

4.7 Handling of DIST 32
4.8 Setting Simulation Option Defaults 35

5 Testing of MOSEL-2 2.0 37

5.1 Testing all new Distributions and Results 37
5.2 Testing MOSEL-2 Performance and Memory Requirements 38
5.2.1 Testing Conditions 38
5.2.2 Test Results 39
5.2.3 Test Conclusion 40

6 Conclusion and Future Work 41

A MOSEL-2 Language Extensions 43

A.1 Distributions 43
A.2 Resampling Policies 45

B Non-working SPNP Distributions 47

B.1 Uniform Distribution 47
B.2 Weibull Distribution 47
B.3 Negative Binomial Distribution 47
B.4 Binomial Distribution 48
B.5 Erlang Distribution 48

C MOSEL-2 Code Adaptations 49

D Full Example Code and Output 71

D.1 trivialPN.c (using SPNP only) 71
D.2 trivialPN.out (using SPNP only) 72
D.3 trivialPN.mos 73
D.4 trivialPN.c 73
D.5 trivialPN.log 75
D.6 trivialPN.out 76
D.7 trivialPN.res 76
D.8 Plot of trivialPN.igl 77
D.9 trivialPN_exp.mos 77

Bibliography **79**

Index **84**

Chapter 1

Introduction

The modeling and performance evaluation of different systems like hardware and software products, work flows or industrial processes is a widespread research field. For this a variety of different modeling languages and evaluation tools were developed during the last decades. Each of them has its own strength and weaknesses.

One of the biggest obstacles a user has to overcome is to choose among those tools to find the one that is appropriate for his particular problem. A further disadvantage is that almost each tool has its own model description language that has to be learned. Therefore, the user is not able to change quickly between the usable tools to obtain the optimal evaluation method.

Remedial action is taken by “MOSEL” (→ Chap. 3, P. 21), a universal model specification and evaluation language developed by Helmut Herold at the Department of Computer Science (Chair of Distributed Systems and Operating Systems) University of Erlangen-Nürnberg, Germany.

The goal of the MOSEL program suite is to provide a simple model description language to let the user concentrate on the actual task: the modeling of the system that has to be evaluated. Nevertheless, an adequate evaluation is provided by the integration of many different tools but without having effect on the model creation.

A very powerful and flexible tool integrated in MOSEL is SPNP (→ Chap. 2, P. 3). SPNP is able to analyze and simulate Petri nets (→ Chap. 2.1, P. 3), i.e. ESPNs (→ Chap. 2.1.5, P. 5), modeled in CSPL (→ Chap. 2.2, P. 5). But not all capabilities of SPNP are used by MOSEL yet. Recently only so called GSPNs (→ Chap. 2.1.3, P. 4) could be directly processed by MOSEL. The integration of TimeNET [30] into the MOSEL system in the form of a new version completely revised by Björn Beutel called “MOSEL-2” [4] provides at least the possibility to analyze DSPNs (→ Chap. 2.1.4, P. 5) numerically and to simulate ESPNs with uniform distributed firing times (also called ,,eDSPNs”).

Of course a few distributions (e.g. hypoexponential, hyperexponential) could be modeled by using the exponential distribution, but with integration of the simulation component of SPNP[1] into the modeling environment of MOSEL-2[2] as presented in this thesis, it is now possible to simulate models containing these distributions and a lot of others (→ Chap. 2.4, P. 11) directly.

[1]SPNP version 6.1.2a

[2]Base: MOSEL-2 version 1.0. Result: MOSEL-2 version 2.0

Chapter 2

SPNP: Stochastic Petri Net Package

SPNP analyzes and simulates Petri nets that are described in CSPL. In the beginning of this chapter, Petri nets will be introduced in a few words. For a more formal and detailed introduction into Petri net theory see [1] or [9]. After that, SPNP's modeling language CSPL will be explained in brief (→ Chap. 2.2, P. 5). In section 2.3 the two main evaluation methods of SPNP, numerical analysis and simulation, will be illustrated. At the end of this chapter all distributions (→ Chap. 2.4, P. 11) and resampling policies (→ Chap. 2.5, P. 17) that are provided by SPNP are listed.

2.1 Petri Nets

Like queuing networks and Markov chains [20] *Petri nets* (PN) are a very universal and one of the best-known and most commonly used methods for performance and reliability model specification. Especially aspects of concurrency, resource contention, communication and synchronization can be expressed graphically but nevertheless with mathematical rigour [8].

They were designed 1962 by Carl Adam Petri and were used in his thesis [7] for the first time. Since then these (yet untimed) Petri nets were permanently improved and extended. By that more and more Petri net families were created:

2.1.1 Petri Nets (untimed)

Petri nets consist of several *places*, graphically represented by circles, *transitions*, represented by bars, several input and output *arcs*, *tokens*, represented by dots and a set of *firing rules*. The graphical representation of a simple Petri net is shown in Figure 2.1. Due to its structure of two serial service stations, the presented system is also called ,,Tandem Network" [10].

The state of a Petri net is described by its marking, i.e. by the distribution of tokens among the net's places. The state can change by *firing* of a transition which causes the migration of tokens from the *input* places to the *output* places of the transition. A transition may only fire if it is *enabled*, i.e. if all input places are occupied by at least one token.

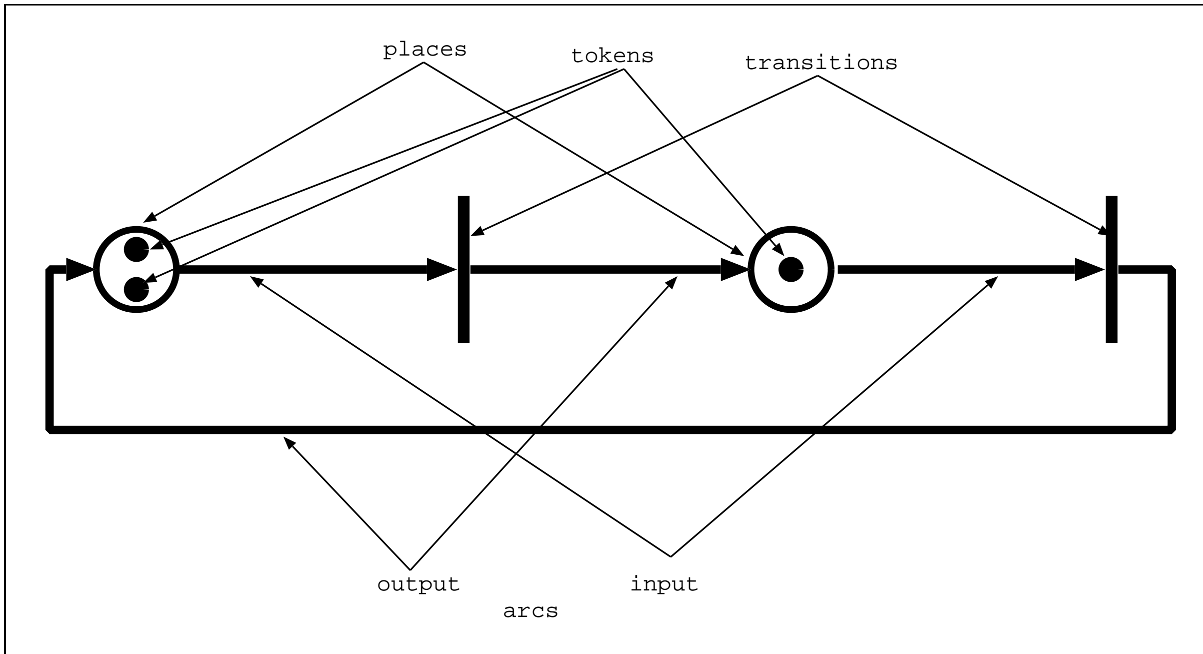


Figure 2.1: simple Petri net: tandem network

Structural extensions to that (untimed) Petri nets, like *arc multiplicity*, *inhibitor arcs*, *transition priorities*, etc., will not be discussed in this document. We refer the interested reader to [15] or [16] for a detailed description.

2.1.2 Stochastic Petri Nets

The modeling power of Petri nets was substantially increased by introducing the consideration of timing aspects of transitions. These new form of Petri nets is called *stochastic Petri nets* (SPN). The random delays associated with transitions are called *firing times* [1] and are sampled from a (negative) exponential distribution which is fully determined by its mean.

2.1.3 Generalized Stochastic Petri Nets

Generalized stochastic Petri nets (GSPN) do not only allow exponentially distributed firing times but also timeless transitions. These will fire immediately as soon as they are enabled. GSPNs fulfill the *Markov property* (→ Chap. 2.3.1, P. 9). Therefore, their reachability graph can be converted to a continuous-time Markov chain (CTMC) [20] and solved with different techniques described in [1].

Unfortunately GSPNs are not suitable for all practical applications. See section 2.4 for examples.

GSPNs are supported by MOSEL since the first version.

2.1.4 Deterministic Stochastic Petri Nets

Deterministic stochastic Petri nets (DSPN) additionally provide deterministic firing times. If the transitions may also be uniformly distributed, they are also called *extended DSPNs* (eDSPN) [30]. Both are non-Markovian Petri nets. Their analysis is much more complicated than that of GSPNs because of the loss of the *Markov property* (→ Chap. 2.3.1, P. 9).

DSPNs are supported by SPNP 6.1.2a [3], TimeNET 3.0 [30] and by MOSEL-2 [4] but not by the first MOSEL version [1].

2.1.5 Extended Stochastic Petri Nets

As well as deterministic stochastic Petri nets, *extended stochastic Petri nets* (ESPN) are non-Markovian. They support arbitrary distributions.

SPNP 6.1.2a supports all distributions described in section 2.4. As a result of this work, *MOSEL-2 version 2.0* is now able to evaluate ESPNs containing these distributions.

2.1.6 Fluid Stochastic Petri Nets

The class of *fluid stochastic Petri nets* (FSPN) was introduced by Trivedi, Kulkarni et al [17]. By replacing the traditional integer tokens with positive real tokens, stochastic fluid models can be analyzed [1].

The evaluation of FSPNs is not supported by MOSEL-2 yet.

2.1.7 High Level Petri Nets

Special techniques like *high level*[19] and *colored Petri nets*[18] are not in the scope of this thesis.

2.2 CSPL: C-based Petri Net Language

The CSPL language is used to describe the Petri net that has to be solved. Moreover, the desired results can be specified in CSPL.

A CSPL file should start with `#include "user.h"` (constants are defined there) and consists of six parts [3][15]. You can enhance the Petri net specification with arbitrary C-code [20].

2.2.1 options()-Part

Within the *options()*-part, option values can be set by the functions *iopt()* and *fopt()* and runtime inputs can be declared by *input()* and *finput()*.

As an example, you can start SPNP's simulation component by using the following *option()*-part:

```
void options() {
   iopt(IOP_SIMULATION,VAL_YES); /*activate SPNP simulation*/
   iopt(IOP_SIM_RUNS, 10000);    /*number of simulation runs*/
   fopt(FOP_SIM_LENGTH, 20.);    /*time to run for each iteration*/
}
```

2.2.2 net()-Part

The *net()*-part is used to describe the Petri net that has to be evaluated. Each place is defined by *place()*, its initial marking by *init()*. Immediate transitions are inserted by *imm()*, timed ones by *rateval()* for exponential distribution or by another function described in section 2.4 for other distributions. The places and transitions are linked by input and output arcs, defined by *iarc()* and *oarc()*.

Of particular interest for non-Markovian Petri nets (ESPNs) are the resampling policies for transitions, that where disabled by the firing of a competitive transition. In SPNP three different policies (→ Chap. 2.5, P. 17) are implemented that can be chosen by the function *policy()*.

Additional functions are implemented in SPNP to describe the Petri net architecture and behavior, like *priority()*, *affected()*, *halting_condition()*, *harc()*, *miarc()*, *moarc()*, *guard()*, *ratedep()*, *fplace()*, etc. Please refer to the SPNP user's manual [3] for their purpose and usage.

Assuming the simple Petri net shown in Figure 2.1 consists of one exponentially distributed (firing rate: 0.3) and one deterministic distributed (firing time: 3) transition, the net could be described this way:

```
void net() {
   place("place1");          /*defining first place*/
   place("place2");          /*defining second place*/
   init("place1",2);         /*initially mark place1 with two tokens*/
   init("place2",1);         /*initially mark place2 with one token*/
   rateval("trans1",0.3);    /*defining first transition with exponential
                               distribution and firing rate 0.3*/
   detval("trans2",3);       /*defining second transition with
                               deterministic firing time 3*/
   iarc("trans1","place1"); /*link first place with first transition*/
   oarc("trans1","place2"); /*link first transition with second place*/
   iarc("trans2","place2"); /*link second place with second transition*/
   oarc("trans2","place1"); /*link second transition with first place*/
}
```

2.2.3 assert()-Part

To exclude forbidden markings from calculation, they are declared illegal within the *assert()*-part. The functions *mark()* and *enabled()* are used to determine the state of the system. If the current state is not acceptable, *assert()* should return the constant RES_ERROR.

The following example ensures, that in a closed Petri net (like Figure 2.1) the number of tokens has to be constant:

```
int assert() {
  if (mark("place1") + mark("place2") != 3)
    return(RES_ERROR);
  return(RES_NOERR);
}
```

2.2.4 ac_init()-Part

In *ac_init()* the function *pr_net_info()* may be called. Before constructing the reachability graph it prints information about the actual model to the output file, in consideration of runtime parameters. The call of function *pr_net_info()* is optional, thus the *ac_init()*-part can be empty. Nevertheless it has to exist:

```
void ac_init() {
}
```

2.2.5 ac_reach()-Part

The *ac_reach()*-part provides two functions, *pr_rg_info()* and *pr_parms()*. The first one prints information about the reachability graph after its creation to the output file. Note, that the reachability graph is not built in simulation mode. The second one can be called to print the current parameter values of all parameterized transitions to the output file. Both functions are optional:

```
void ac_reach() {
}
```

2.2.6 ac_final()-Part

The *ac_final()*-part is used to specify the desired results of the calculation. The *ac_final()*-part is evaluated at the end of the computation and can contain the functions *solve()*, *expected()*, *pr_expected()*, *pr_value()* and a lot more. With these functions combined with the capabilities of the C programming language [20] the user can define the desired results very flexibly.

Therefore, the user can for example build his own reward functions [3][1] in C code which have to be defined before they can be called in the *ac_final()*-part (usually as global C function in front of the *ac_final()*-part):

```
/* declaring own C functions */

double tokens1() {
  return(mark("place1"));
}

double tokens2() {
  return(mark("place2"));
}

double busy1() {
  return((mark("place1")!=0) ?  1 :  0);
}

double empty2() {
  return((mark("place2")==0) ?  1 :  0);
}

/* ac_final */

void ac_final() {
  /* local variables */
  int busy2;
  int tokens1_2;
  /* use steady state metrics */
  solve(INFINITY);
  /* declare desired outputs */
  pr_expected("Average token number in 'place1':", tokens1);
  pr_expected("Average token number in 'place2':", tokens2);
  tokens1_2 = expected(tokens1) + expected(tokens2);
  pr_value("Average token number in system:", tokens1_2);
  pr_expected("Utilization of 'place1':", busy1);
  busy2 = 1 - expected(empty2);
  pr_value("Utilization of 'place2':", busy2);
}
```

For more information about *ac_final()* see section 4.7 or the SPNP user's manual [3].

See section D.1 for the full SPNP-code of this simple example of a Petri net. The SPNP output of this net is shown in section D.2.

2.3 Evaluation Methods

SPNP provides two different solution methods: numerical analysis and simulation. Each of them can be used for steady-state or transient analysis of the system.

2.3.1 Numerical Analysis

The numerical method is used for Petri nets that fulfill the *Markov property* [10][20][26]. In general only GSPNs can be solved this way because the firing times Y of all transitions are *memoryless* distributed [1]:

$$\boxed{P[Y \leq t + y] = P[Y \leq t + y | Y \geq y] = P[Y \leq y];} \tag{2.1}$$

Therefore, it is indistinguishable if work is lost or not if enabled transitions are preempted, e.g. by a competitive transition, because the future behaviour of the system is independent from the time Y the system spends in the current state (also called *State Sojourn Time*[20]).

The exponential distribution is the only continuous distribution with this *memoryless* property.

The *clocks* (→ Chap. 2.3.2, P. 9) of preempted transitions do not have to be considered, because it does not make a difference if the firing time is resampled when the preempted transition is re-enabled again.

There are different methods, the user can choose for steady-state (Successive Overrelaxation, Gauss-Seidel and Power method) and transient (standard, Fox-Glynn and stiff uniformization) analysis.

The solution algorithms are described in detail in [2].

A numerical solution is planned for FSPNs but is not supported by the current version of SPNP [3].

2.3.2 Discrete Event Simulation

Most Petri nets that do not fulfill the *Markov property* (→ Chap. 2.3.1, P. 9) cannot be solved analytically. The complete reachability graph is not built and cannot be mapped to a *CTMC* (→ Chap. 2.1.3, P. 4). Therefore, the behavior of the Petri net has to be simulated. This method is recommended as well for very large Markovian Petri nets, whose reachability graph grows huge. In SPNP the simulation method is also used for Markovian and non-Markovian FSPNs.

Due to the fact that we want to use general firing distributions (ESPNs) we have to describe the state of the Petri net not only by its marking, but as well by a continuous vector describing the remaining firing times, i.e. the *clock* of each transition. This extension increases MOSEL-2's modeling power to the power of *Generalized semi-Markov processes*(GSMP) [33] and cannot be solved analytically in general.

Therefore, in case of a GSMP we have to use (discrete event) simulation [14][5] and consider the *clocks* of all transitions.

The following (simplified) algorithm describes a single simulation run of SPNP[1]:

1. initialization:
   - set the global simulation clock to zero
   - set the start state/marking of the Petri net
   - define the termination event (e.g. *SIM_LENGTH* (→ Chap. 2.2.1, P. 5))
2. determine the set of enabled transitions (goto step 8 if set is empty)
3. determine the firing time of each enabled transition:
   - sample a firing time for each just enabled transition and for formerly preempted transitions with resampling policy PRD (→ Chap. 2.5.1, P. 18)
   - continue with clocks for re-enabled transitions with resampling policy PRS (→ Chap. 2.5.3, P. 19)
   - restart the clocks of re-enabled transitions with resampling policy PRI (→ Chap. 2.5.2, P. 18)
4. find the transition with the shortest remaining clock time (i.e. firing time) and increment the simulation clock and all clocks of concurrent PRS transitions by this time (The simulation algorithm has to ensure, that there is only a single transition that can fire at a certain time.)
5. goto step 8 if the termination event was reached (e.g. simulation clock reaches *SIM_LENGTH*)
6. compute the new marking obtained by the firing of the transition and the marking dependent results
7. continue with step 2
8. end of simulation, provide results

There are also different *speed-up methods* available for SPNP simulation like *importance splitting*[31] (RESTART and splitting) or *importance sampling*[32] (in implementation). These techniques are aiming on faster evaluation of models including interesting *rare events*, e.g. failure of components in a failure tolerant system, by making them occur more often. Without these speed-up techniques the simulation would require a huge simulation length and therefore a very long time.

The theoretical background behind SPNP simulation is described in [5].

[1]This algorithm does not pay attention to several efficiency improvements and other methods than SPNP's regular discrete event simulation method. Refer to [5] for details.

2.4 Distributions

The following distributions of transitions are (besides exponential and timeless transitions) supported by SPNP 6.1.2a and – as a result of this thesis – by MOSEL-2 2.0. (See section A.1 for the corresponding syntactical extensions to the MOSEL language.)

In CSPL syntax the first parameter of each distribution is the name of the transition it is assigned to. In the following descriptions, only the other parameters are mentioned that have effect on the distribution.

2.4.1 Beta Distribution

- type: continuous
- SPNP keyword: `betval`
- MOSEL-2 keyword: `BETA`
- 2 parameters: $\alpha > 0$, $\beta > 0$
- density function:

$$f_X(x) = \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \quad \forall\, 0 < x < 1; \tag{2.2}$$

  with beta function:

$$B(\alpha,\beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1} dx; \tag{2.3}$$

- possible applications: small damages and damage rates (insurance industry), Bayesian analysis
- information taken from: [3], [21], [22], [23]

2.4.2 Binomial Distribution

- type: discrete
- SPNP keyword: `binoval`
- MOSEL-2 keyword: `BINOM`
- 3 parameters: $t \in \mathbb{N}$ (number of trials), $0 \leq p \leq 1$ (probability), $s > 0$ (weighting factor)

- probability function:

$$P_X(s \cdot x) = \binom{t}{x} p^x (1-p)^{t-x}, \quad \forall\, x \in (0, 1, \dots t); \tag{2.4}$$

- possible applications: number of successes in t Bernoulli trials, number of defective items in a batch of size t, number of items demanded from a inventory
- information taken from: [3], [1]
- note: This distribution does not work properly in SPNP 6.1.2a. See appendix B for details.

2.4.3 Cauchy Distribution

- type: continuous
- SPNP keyword: `cauval`
- MOSEL-2 keyword: `CAUCHY`
- 2 parameters: $\alpha \in \mathbb{R}$ (location), $\beta > 0$
- density function:

$$f_X(x) = \frac{\beta}{\pi(\beta^2 + (x-\alpha)^2)}, \quad \forall\, x > 0; \tag{2.5}$$

- possible applications: pathological cases, good check for robust techniques that are designed to work well under a wide variety of distributional assumptions
- information taken from: [3], [24]

2.4.4 Deterministic Distribution

- type: continuous
- SPNP keyword: `detval`
- MOSEL-2 keyword: `AFTER`
- 1 parameter: $t > 0$ (delay time)
- density function:

$$f_X(x) = \delta(x - t), \quad \forall\, x > 0; \tag{2.6}$$

- possible applications: processing packets with constant length, processes with constant delay, timeouts in protocols
- information taken from: [3], [25]

2.4.5 Erlang Distribution

- type: continuous
- SPNP keyword: `erlval`
- MOSEL-2 keyword: `ERLANG`
- 2 parameters: $0 < \mu$ (rate), $k \in \mathbb{N}$ (number of phases)
- density function:

$$f_X(x) = \frac{k\mu(k\mu x)^{k-1}}{(k-1)!} e^{-k\mu x}, \quad \forall\, x > 0; \tag{2.7}$$

- possible applications: queuing theory, switching of telephone calls
- information taken from: [3], [10]
- note: This distribution does not work properly in SPNP 6.1.2a. See appendix B for details.

2.4.6 Gamma Distribution

- type: continuous
- SPNP keyword: `gamval`
- MOSEL-2 keyword: `GAMMA`
- 2 parameters: $\alpha > 0$ (shape), $\lambda > 0$ (scale)
- density function:

$$f_X(x) = \frac{\lambda^\alpha x^{\alpha-1} e^{-\lambda x}}{\Gamma(\alpha)}, \quad \forall\, x > 0; \tag{2.8}$$

  with gamma function:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt, \quad \forall\, z > 0; \tag{2.9}$$

- possible applications: completion time of a job, service time of customer or machine repair, generalized linear models for risk theory, examination of Poisson processes
- information taken from: [3], [1], [21], [23]

2.4.7 Geometric Distribution

- type: discrete
- SPNP keyword: `geomval`
- MOSEL-2 keyword: `GEOM`
- 2 parameters: $0 \leq p \leq 1$ (probability), $s > 0$ (weighting factor)
- probability function:

$$\boxed{P_X(s \cdot x) = p(1-p)^{n-1}, \quad \forall\, n \in \mathbb{N};} \tag{2.10}$$

- possible applications: number of failures before first success in a sequence of Bernoulli trials, number of items in a batch of random size, number of cells in a transmitted sequence on a ATM link, fulfills the *Markov property*
- information taken from: [3], [1], [23]

2.4.8 Hyperexponential Distribution

- type: continuous
- SPNP keyword: `hyperval`
- MOSEL-2 keyword: `HYPER`
- 3 parameters: $\mu_1 > 0$ (rate of first stage), $\mu_2 > 0$ (rate of second stage), $0 \leq p \leq 1$ (probability to choose first stage)
- density function:

$$\boxed{f_X(x) = p\mu_1 e^{-\mu_1 x} + (1-p)\mu_2 e^{-\mu_2 x}, \quad \forall\, x > 0;} \tag{2.11}$$

- possible applications: processes that compose several (here two) alternate or parallel service phases with different exponential distributed service time, CPU service time in computer systems, failure density of a product manufactured in two parallel assembly lines, two different classes of packets with different distributed length
- information taken from: [3], [1], [25]

2.4.9 Hypoexponential Distribution

- type: continuous
- SPNP keyword: `hypoval`
- MOSEL-2 keyword: `HYPO`
- 4 parameters: $\mu_1 > 0$ (rate of first stage), $\mu_2 > 0$ (rate of second stage), $\mu_3 > 0$ (rate of third stage), $n \in \{2,3\}$ (number of stages)
- density function ($n = 2$):

$$\boxed{f_X(x) = \frac{\mu_1\mu_2}{\mu_1 - \mu_2}(e^{-\mu_2 x} - e^{-\mu_1 x}), \quad \forall\, x > 0;} \tag{2.12}$$

- density function ($n = 3$):

$$\boxed{f_X(x) = \sum_{i=1}^{3} \mu_i e^{-\mu_i x} \left(\prod_{j \neq i} \frac{\mu_j}{\mu_j - \mu_i} \right), \quad \forall\, x > 0;} \tag{2.13}$$

- possible applications: processes that compose several (here two or three) sequential service stages with different exponential distributed service time, service time of I/O operations in computer systems, execution times of programs that are organized into a set of sequential phases
- information taken from: [3], [1], [10]
- note: There is an error in [3]. The order of parameters is not as described on page 27 of the manual.

2.4.10 Lognormal Distribution

- type: continuous
- SPNP keyword: `lognval`
- MOSEL-2 keyword: `LOGN`
- 2 parameters: $\mu > 0$ (location or mean), $\sigma^2 > 0$ (shape)
- density function:

$$\boxed{f_X(x) = \frac{1}{x\sqrt{2\pi\sigma^2}}\, e^{-\frac{(ln x - \mu)^2}{2\sigma^2}}, \quad \forall\, x > 0;} \tag{2.14}$$

- possible applications: modeling major losses (insurance industry), modeling failure times in reliability applications
- information taken from: [3], [21], [24]

2.4.11 Normal Distribution

- type: continuous
- SPNP keyword: `normval`
- MOSEL-2 keyword: `NORM`
- 2 parameters: $\mu > 0$ (location or mean), $\sigma^2 > 0$ (shape or variance)
- density function:

$$\boxed{f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad \forall\, x > 0;} \tag{2.15}$$

- possible applications: errors in measurements of various types, central limit theory, life time of a component in its wear out period, many classical statistical tests are based on the assumption that the data follows a normal distribution, used to find significance levels in many hypothesis tests and confidence intervals, in modeling applications (such as linear and non-linear regression) the error term is often assumed to follow a normal distribution, many phenomena are normal distributed (e.g. in financial industry, astronomy, econometrics, biology), Wiener processes
- information taken from: [3], [1], [24], [23]

2.4.12 Pareto Distribution

- type: continuous
- SPNP keyword: `parval`
- MOSEL-2 keyword: `PARETO`
- 2 parameters: $\alpha > 0$, $\beta > 0$
- density function:

$$\boxed{f_X(x) = \frac{\alpha\beta^\alpha}{(x+\beta)^{\alpha+1}}, \quad \forall\, x \geq 0;} \tag{2.16}$$

- possible applications: modeling major losses - in particular industrial fires (insurance industry), economic policy, modeling wide area traffic, distribution of file sizes in internet traffic
- information taken from: [3], [21], [27], [28]

2.4.13 Poisson Distribution

- type: discrete
- SPNP keyword: `poisval`
- MOSEL-2 keyword: `POIS`
- 2 parameters: *lambda* > 0 (mean and variance), $s > 0$ (weighting factor)
- density function:

$$\boxed{P_X(s \cdot x) = \frac{\lambda^x}{x!}\, e^{-\lambda}, \quad \forall\, n \in \mathbb{N}_0;} \tag{2.17}$$

- possible applications: number of events in a time interval when the events are occurring at constant rate, number of jobs arriving at a computer center in an interval, number of queries to a central database, number of new call requests arising in a telephone system in a given interval, analysis of frequency tables, analysis of count data, number of defective items in a big inventory, approximation of binomial distribution
- information taken from: [3], [1], [24], [23]

Other distributions, like the Weibull[2], uniform or negative binomial distribution, are mentioned in the SPNP user's manual [3], but a test showed, that these are unfortunately not implemented in SPNP version 6.1.2a. See appendix B for details.

2.5 Resampling

As a result of the loss of the *Markov property* (→ Chap. 2.3.1, P. 9) by using several of the distributions mentioned above, we now have to consider resampling (also called re-enabling) policies for former enabled transitions that were disabled by the firing of a competitive transition or for still enabled transitions whose firing time is affected by the firing of another transition [29][3]. These policies describe how the clocks (→ Chap. 2.3.2, P. 9) of preempted transitions are treated.

SPNP provides the function *policy()* which has to be located in the *net()*-part (→ Chap. 2.2.2, P. 6). This function requires two parameters: The first parameter chooses the transition, the policy ought to be assigned to. The second parameter specifies which policy should be used:

[2]Note that the Weibull distribution can be approximated by an Erlang distribution and modeled using the exponential distribution. See [1] for details.

2.5.1 Preemptive Repeat Different (PRD)

- keyword: PRD
- explanation: The interrupted job is repeated with a resampled random time, i.e. the transitions clock is set to a new time sampled from its distribution. By default transitions obey this policy.
- illustration[3]:

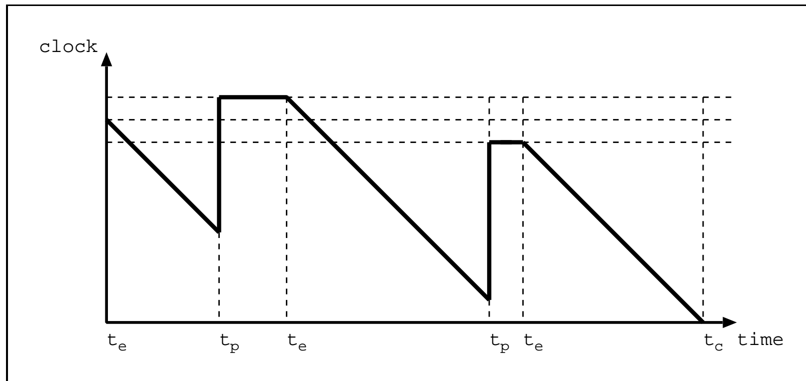


Figure 2.2: PRD

2.5.2 Preemptive Repeat Identical (PRI)

- keyword: PRI
- explanation: The interrupted job is repeated with an identical firing time, i.e. the transitions clock is reset to its last start value.
- illustration[3]:

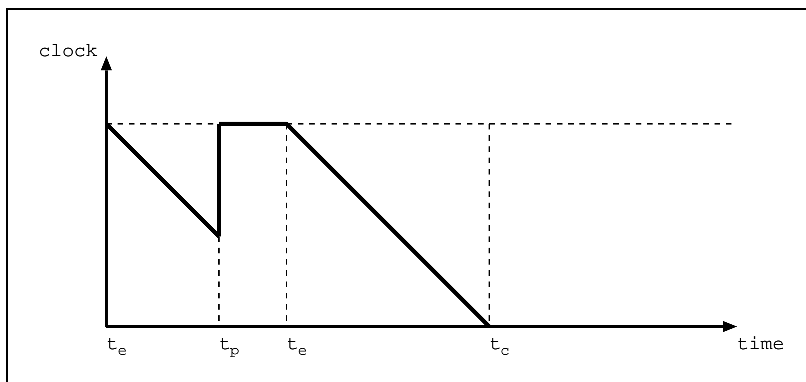


Figure 2.3: PRI

[3] t_e: enabling time; t_p: preemption time; t_c: completion time (firing point) [5]

2.5.3 Preemptive ReSume (PRS)

- keyword: PRS
- explanation: The interrupted job continues with the remaining firing time. The transitions clock is saved at the time of the transitions preemption and proceeded when the transition is re-enabled. This policy is also called *work-conserving*.
- illustration[3]:

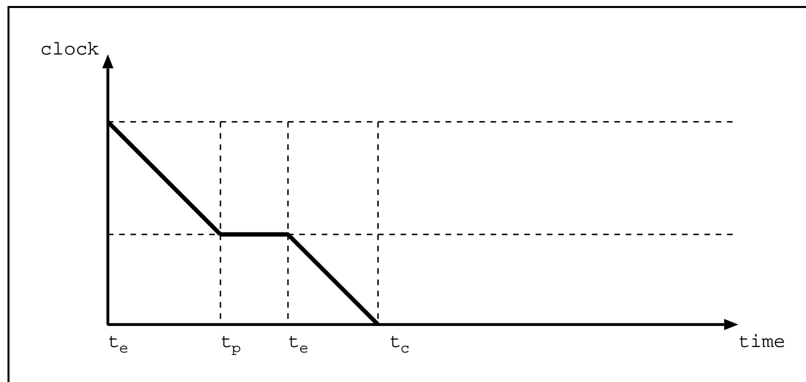


Figure 2.4: PRS

The MOSEL-2 syntax of these policies is described in [4] (PRD, PRS) and section A.2 (PRI).

Chapter 3

MOSEL: MOdeling, Specification and Evaluation Language

To explain which enhancements could be done to MOSEL, the MOSEL language will be described in brief at first.

3.1 Introduction to MOSEL

As it is written in [1], *MOSEL* (MOdeling, Specification and Evaluation Language) is a ,,powerful modeling language [...] developed at the University of Erlangen, which allows the modeling of complex systems in a very intuitive and simple way. The performance and reliability measures can be specified in a simple manner within the script describing the system. With some experience, it is possible to write down the MOSEL description of a system immediately only by knowing the behavior of the system under study." Furthermore a complete modeling environment has been developed. It is called *MOSEL* as well and is able to translate systems described in MOSEL language into the modeling languages of MOSES [29] and SPNP (→ Chap. 2, P. 3). The later version *MOSEL-2* [4] is a revised form of MOSEL that also provides translation into *TimeNET*'s [30] modeling language. Nevertheless, the modeling environment MOSEL ,,does not only translate MOSEL descriptions into the languages appropriate to the different tools, but it also starts the tools automatically, captures the computed results, and presents them in a graphical form." [1]

3.2 Modeling and Evaluating using MOSEL

System evaluation with MOSEL is done in four major steps. The first three steps have to be done manually. The last step is done rather automatically.

3.2.1 Building the Conceptual Model

When a real system has to be computed, the first step is to simplify and formalize it by abstracting and transforming it into a conceptual model, e.g. into a *queuing network* [20][10], *Markov chain* [20] or *Petri net* (→ Chap. 2.1, P. 3). Of course, this step can be left over for small and uncomplicated systems, the MOSEL model then can be built immediately.

3.2.2 Describing the Conceptual Model as MOSEL Model

As a next step the conceptual model has to be described in MOSEL language. To give an example, the simple Petri net in Figure 2.1 will be described below. For details of MOSEL's modeling language see [1]. Refer to [4] for the changes between MOSEL and MOSEL-2.

The source code containing the model description in MOSEL is stored in an ASCII-file. These files should be suffixed by *.msl* for (older) MOSEL files, *.mos* for MOSEL-2 files, respectively.

Due to the fact that this thesis (finally) extended the abilities of MOSEL-2 – the first changes were made to MOSEL and had to be discarded later – we will stick to MOSEL-2 syntax. You can find the complete MOSEL-2 code of this example in section D.3.

As a simple example the Petri net of Figure 2.1 will be described in MOSEL-2 language now. Note that the distributions will be chosen quite similar [1] to section 2.2.2.

A MOSEL-2 model description is divided in five parts. Some of them, for example the first (constant part) and the last (picture part), are optional.

Constant and Parameter Part

At the beginning the constants and parameters should be defined that will be used later. By doing this, the model can easily modified anytime. Note that in MOSEL-2 no variables can be declared as it could be done in MOSEL [2].

```
CONST rate := 4i     /* rate of first transition/rule */
CONST delay := 0.5;  /* delay of second transition/rule */
CONST tokens := 3;   /* number of tokens/jobs in closed system */
CONST init1 := 2;    /* initial number of tokens in first place/node */
CONST init2 := tokens-init1; /* initial number of tokens in 2nd node */
```

[1] Now, we will choose a firing rate equal ,,4" for the exponential distributed transition and a firing delay of ,,0.5" for the deterministic transition.

[2] See [4] for all differences between MOSEL and MOSEL-2.

Node Part

In the *Node Part* the static components of the net are described. MOSEL's basic modeling primitive is called *node*. These *nodes* can contain a integer number of *jobs* (tokens). The maximum number of jobs a node can contain is called *capacity* of this node. The *start state* of the system is defined by giving the initial number of jobs for each node. Using the *assert* directive, illegal system states can be specified:

```
NODE place1[tokens] := init1;    /* introduce 'place1' with max. 'tokens'
                                    tokens and 'init1' initial tokens */
NODE place2[tokens] := init2;    /* introduce 'place2' with max. 'tokens'
                                    tokens and 'init2' initial tokens */
ASSERT place1 + place2 = tokens; /* the number of tokens in the system
                                    always has to be 'tokens' */
```

Rule Part

The *Rule Part* describes the dynamic behavior of the system. In MOSEL/MOSEL-2 the transitions are called *rules*.

```
FROM place1 TO place2 RATE rate;   /* exp. distr. transition from 'place1'
                                      to 'place2' with parameter 'rate' */
FROM place2 TO place1 AFTER delay; /* determ. distr. transit. f. 'place2'
                                      to 'place1' w. parameter 'delay' */
```

However, a full MOSEL-2 rule may consist of seven different parts (or even multiple of these):

```
IF (cond)              /* condition part */
  FROM place1          /* from part */
  TO place2            /* to part */
  RATE rate            /* timing part */
  WEIGHT weight        /* branching probability */
  PRIO priority        /* priority */
  PRD                  /* resampling policy */
;
```

For a description of these rule elements see [4].

Result Part

The desired results of the evaluation can be specified in the *Result Part*.

```
PRINT avg_no_of_tokens_in_place1 := MEAN(place1);
PRINT avg_no_of_tokens_in_place2 := MEAN(place2);
PRINT usage_of_place1 := PROB(place1 != 0);
PRINT usage_of_place2 := PROB(place2 != 0);
PRINT DIST place1;
```

Picture Part

Within the *Picture Part* the optional IGL (*Intermediate Graphic Language* by Helmut Herold [1]) output can be justified. This textual output can be interpreted by the IGL interpreter, which is part of the MOSEL-2 modeling environment. Unfortunately the abilities of MOSEL-2 to specify IGL output is quite restricted compared to MOSEL.

```
PICTURE "token distribution of place1"
 CURVE DIST place1
```

3.2.3 Invoking the MOSEL-2 Modeling Environment

We will now evaluate the just described simple DSPN by using the *MOSEL-2 Modeling Environment*[4] with options appropriate to let MOSEL-2 start the SPNP tool after it has created the CSPL model. To obtain all possible command line options that are supported by mosel refer to [4] or call MOSEL with option „-h".

We can evaluate the MOSEL-2 model above by calling MOSEL-2 from a shell using this command line:

```
mosel2 -csk -o +SIMULATION -o SIM_LENGTH=100 -o SIM_RUNS=1000 trivialPN.mos
```

Explanation:

- `mosel2` : The first argument is always the program name itself, of course.
- `-csk` : Combination of three different options that can also be written separately:
  - `-c` : This option tells MOSEL-2 to translate the model into CSPL language.
  - `-s` : This option tells MOSEL-2 to start the appropriate tool (SPNP in this case because of the -c option) to evaluate the model after the model translation.
  - `-k` : Keep all files, that were produced by MOSEL-2 and the called tool. Normally all generated files – naturally except for the final result files – are deleted after the computation.

- `-o` : After this delimiter, you can pass options directly to the selected tool, here SPNP. Possible simulation specific options are shown in table 3.1 on page 27[3], all other supported SPNP options are listed in [4] and described in [3]. For this example we use:
  - `+SIMULATION` : Activates SPNP simulation mode.
  - `SIM_LENGTH=100` : Declaration of the simulation length, i.e. the time to run for each simulation iteration. We choose a length of 100 simulation time units. The default would be a length of 1000.
  - `SIM_RUNS=1000` : Number of simulation runs to obtain the desired accuracy. We choose 1000 runs. This would also be the default.
- `trivialPN.mos` : The name of the MOSEL-2 model description file that has to be evaluated. The output files that will be generated will have the same filename, but certainly with another suffix (→ Chap. 3.2.4, P. 25).

3.2.4 MOSEL-2 working

After MOSEL-2 is called properly, it automatically performs the following steps (assuming the command line above):

- **parsing command line options:** To know, which input file should be evaluated and with which tool, at first the command line is parsed.
- **parsing input file:** The input file (*trivialPN.mos*) is parsed and checked. All model data is collected in an internal data structure.
- **translating input file:** The just created data structure is used to build a model description in the appropriate language. In our case, the CSPL file *trivialPN.c* will be created (→ Chap. D.4, P. 73).
- **calling appropriate tool:** MOSEL-2 will then call *spnp trivialPN*. SPNP does all necessary computations and writes its result in a file called *trivialPN.out* (→ Chap. D.6, P. 76). Besides *trivialPN.out* SPNP creates the three output files *trivialPN.o* (binary produced by compiling *trivialPN.c*), *trivialPN.spn* (binary executable produced by linking *trivialPN.o* together with the SPNP object files) and *trivialPN.log* (contains all SPNP messages produced by SPNP during model solution) (→ Chap. D.5, P. 75)[4].
- **parsing tool output:** As long as no error occurs MOSEL-2 will now parse *trivialPN.out* and collect the results produced by the SPNP analysis in an internal, tool independent data structure.
- **writing output files:** Out of this new data structure MOSEL-2 will finally build the two output files *trivialPN.out* (readable textual result file) (→ Chap. D.7, P. 76) and

[3]Information is taken from [3]. Due to a fatal type setting error within that manual ([3], page 51, table 7.3), the default values had to be excerpted from the postscript version's source of the manual. However, some of the default values are changed by MOSEL-2 so that they differ from the SPNP defaults.

[4]Remember that all intermediate files are only kept, when option *-k* is set.

trivialPN.igl (intermediate graphic language file that can be interpreted by the IGL interpreter [1]) (→ Chap. D.8, P. 77).

3.3 Restrictions of (old) MOSEL

In the example above we used constructs, like SPNP simulation and deterministic firing distribution, that were not implemented in MOSEL yet. Deterministic (and uniform) firing distribution can be evaluated by MOSEL-2 version 1.0 [4], but not until this thesis, all distributions described in section 2.4 can be computed with MOSEL-2 by using SPNP simulation.

Option	**Values**	**Default**	**Brief Description**
SIMULATION	yes, no	no	Mandatory for using SPNP simulation.
SIM_LENGTH	non-negative double	1000	Specifies the time to run for each iteration (in regular discrete event simulation mode) or the length of the batches (Batch means). Option is ignored in regenerative simulation.
SIM_RUNS	non-negative int	1000	Upper bound for the number of simulation runs (in regular discrete event simulation mode) or number of batches (Batch means). If this option is set to zero, SPNP runs until SIM_ERROR is reached.
SIM_ERROR	non-negative double, between 0.0 and 1.0	0.1	SPNP will run until this error precision is reached, unless SIM_RUNS is specified.
SIM_CUMULATIVE	yes, no	yes	If set to *yes*, the data will be collected as time averages from zero to SIM_LENGTH. Otherwise point-of-time estimates will be computed.
SIM_CONFIDENCE	non-negative double, currently only: 0.90, 0.95 or 0.99	0.95	Specifies the desired confidence when computing the confidence intervals. (These are not shown in the MOSEL-2 output.)
SIM_SEED	non-negative int	52836	Allows to change the seed of the random generator.
SIM_STD_REPORT	yes, no	no	If set to *yes*, SPNP will print a standard report to its output file. This report contains the four standard measures: utilization of each place, average number of tokens in each place, probability that a transition is enabled, throughput of a transition. The report is not parsed by MOSEL-2.
SIM_RUNMETHOD	VAL_REPL, VAL_BATCH, VAL_RESTART, VAL_SPLIT	VAL_REPL	Default is using the standard discrete event simulation with independent replications. For more information about the other simulation methods and their specific options see [3].

Table 3.1: SPNP Simulation Options

Chapter 4

Changes to MOSEL-2

In this chapter the most important steps to obtain MOSEL-2 version 2.0 will be described in brief. You can find all changed code lines[1] in their context in the appendix (→ Chap. C, P. 49).

4.1 Command Line Parameters

The first problem occurs while trying to enhance MOSEL-2 to pass SPNP simulation options (→ Table 3.1, P. 27), that were specified on the command line, to the CSPL file. The options are denied by MOSEL-2 as „invalid CSPL options". After we introduce the CSPL simulation options into MOSEL-2 (→ Chap. C.2, P. 49) the options will be recognized properly and printed to the CSPL *option()*-part. MOSEL-2 is now able to use SPNP in simulation mode and SPNP provides correct results.

4.2 Result Parser

Unfortunately the results provided by SPNP cannot be parsed by MOSEL-2 yet. The problem is located in the result parser. The result parser does only then parse the results, if a variable called „shot" does not have the value „-1". This is always the case in simulation mode. So MOSEL-2 has to be told, that it is allowed to parse results that are obtained by SPNP simulation even if „shot" is negative (→ Chap. C.26, P. 69). After that we are able to simulate GSPNs with MOSEL-2 and SPNP.

At that time it emerges for the first time, that there will be a problem with simulating MOSEL-2 DIST constructs. Due to the fact that this seems not to be a trivial problem, we will put that problem on hold (→ Chap. 4.7, P. 32).

Thereafter further small changes to the result parser have to be committed to provide the MOSEL-2 output with a simulation specific header (→ Chap. C.25, P. 68).

[1] First work has been done to the (old) MOSEL version by Helmut Herold [1]. Most of this changes had to be discarded after the introduction of Björn Beutel's MOSEL-2 [4] and therefore are not described in this thesis.

4.3 Introducing new Distributions

This step is the most intricate one. It has to be done in several sub-steps:

4.3.1 Incrementing Number of Parameters

At first the number of distribution parameters per distribution has to be increased. MOSEL-2 version 1.0 supported three parameters maximal (discrete uniform distribution with its three parameters: *start*, *end* and *step* [4]). However the hypoexponential distribution will need four parameters (→ Chap. 2.4.9, P. 15).

Fortunately there are only a few places, where we have to change this:

- **Central Storage for Model Data:** The constant *DIST_PARAM_MAX* has to be incremented (→ Chap. C.20, P. 64).
- **Model Parser:** We have to introduce the new parameter for function *set_distrib()* and also copy that new parameter to the data structure *rule* (→ Chap. C.21, P. 64) within this function. Furthermore we have to introduce the fourth parameter with value *NULL* also to all *set_distrib()* calls of already existent distributions (→ Chap. C.23, P. 67).

4.3.2 Enabling Solving of DSPNs by SPNP Simulation

Still we are only able to simulate GSPNs. As a next step we will make the deterministic distribution that is already existent in MOSEL-2 available to the SPNP simulation component.

Due to the fact, that non-GSPN-distributions can only be evaluated by SPNP if it works in simulation mode, we have to differ between the SPNP modes within MOSEL. The two modes are introduced (→ Chap. C.5, P. 50) and selected (→ Chap. C.3, P. 49) by the command line parser. The default method is of course „not simulated" (→ Chap. C.4, P. 50).

Then we have to prevent MOSEL-2 from blocking the usage of DSPNs with SPNP simulation. This can be done in the main module (→ Chap. C.12, P. 59).

Finally we have to provide the MOSEL-2 CSPL generator with the ability to model deterministic distributed transitions in CSPL (→ Chap. C.8, P. 54) and their parameter functions if necessary (→ Chap. C.6, P. 50).

After that, we are able to evaluate DSPNs with MOSEL-2 using SPNP simulation.

4.3.3 Introducing new ESPN distributions to MOSEL-2

Now we have to introduce new keywords for the MOSEL-2 language with that a user will be able to specify the new distributions. This is done in the model parser (→ Chap. C.22, P. 67) and in the lexical scanner (→ Chap. C.27, P. 69).

Then the new distribution types have to be introduced to the central data storage (→ Chap. C.18, P. 64) as well as the property types (→ Chap. C.17, P. 63) that will be set, if a specific distribution is used (→ Chap. C.14, P. 61).

After that, each new distribution has to attend the following process:

- extending the model parser by introducing the code for recognizing the new distribution in the MOSEL-2 input file, parsing its parameters and sending them to function *set_distrib()* (→ Chap. C.23, P. 67)
- extending *set_distrib()* by adding appropriate parameter checks for the new distribution (→ Chap. C.21, P. 64)
- let the CSPL generator produce proper parameter functions (→ Chap. C.6, P. 50)
- provide the CSPL generator with the ability to produce transitions with the new distribution and the corresponding parameters (→ Chap. C.8, P. 54)
- testing the new distribution with a small MOSEL-2 test file (yet without job distribution measurements)

When this is done, MOSEL-2 will be able to evaluate different ESPNs, but there are still things to do.

4.4 Providing Resampling Policy PRI

To introduce the new[2] resampling policy PRI into MOSEL-2, at first the keyword has to be announced to the lexical scanner (→ Chap. C.27, P. 69) and to the parser (→ Chap. C.22, P. 67). The new policy type has to be defined in the central data storage (→ Chap. C.19, P. 64). The parser has to be prepared to recognize the keyword and set the appropriate policy type (→ Chap. C.23, P. 67). The CSPL generator finally has to be made capable of writing PRI-policies (→ Chap. C.9, P. 56).

4.5 Blocking new Constructs from normal SPNP mode and other Tools

All blocking is done in the main module. Concerning SPNP, DSPNs and ESPNs only have to be blocked, if SPNP is in ,,normal" (not simulating) mode (→ Chap. C.12, P. 59). We do not need to block re-enabling policies from SPNP.

Concerning MOSES we have to block all models with new distributions (→ Chap. C.11, P. 59). Re-enabling policies were blocked from MOSES since MOSEL-2 version 1.0. We have nothing to do here.

The new distributions can neither be used with TimeNET (→ Chap. C.13, P. 60). Nevertheless, the re-enabling policies can be used except the PRI policy. To enable the evaluation of models containing the policy PRI anyway without having to rewrite it, PRI is replaced by the default policy PRD. An adequate warning is displayed (→ Chap. C.28, P. 69).

[2]The resampling policies PRD and PRS are already defined in MOSEL-2

4.6 Updating Model Dumping Function

MOSEL-2 provides the option *-d* for dumping the just parsed model data to standard output. This is only used for debugging. Nevertheless, the code of the dumping function has to be suitably completed (→ Chap. C.15, P. 61).

4.7 Handling of DIST

A somewhat tricky problem is the handling of the MOSEL-2 job distributions (DIST). These are realized in MOSEL and MOSEL-2 version 1.0 by using global variables in CSPL to have influence on the reward functions while model evaluation:

Example:
A `PRINT DIST place` in MOSEL-2 is translated to the following CSPL-constructs:

```
[..]
051: char *dist_node_;
052: int dist_i_;
053:
054: double dist_func_ (void)
055: {
056:   return (mark (dist_node_) == dist_i_ ? 1.0 : 0.0);
057: }
058:
059: void print_dist_ (char *node, int capacity)
060: {
061:   char text[200];
062:
063:   dist_node_ = node;
064:   for (dist_i_ = 0; dist_i_ <= capacity; dist_i_++)
065:   {
066:     sprintf (text, "DIST_ %s %d", node, dist_i_);
067:     pr_value (text, expected (dist_func_));
068:   }
069: }
[..]
083: void ac_final (void)
084: {
[..]
089:  print_dist_ ("place", 3);
090: }
```

Explanation:
In ,,normal" SPNP mode, i.e. SPNP is using analytic-numeric methods, not discrete event simulation, **ac_final()** is called only once[3] after the creation of the Markov Reward Model (MRM). With the help of the MRM the model is fully determined and every desired result can be computed out of it. By the time **ac_final()** is called, **print_dist_()** will initialize the first global variable **char *dist_node_** with the nodes name ,,place" and by run-

[3]This can be easily proved by using a simple **fprintf()** within **ac_final()**.

ning into its loop (line 064) the second global variable **int dist_i_**[4] will be set to ,,0" first. Within this loop **expected (dist_func_)** is called (line 067) to obtain the desired results[5] from the marking dependent function **dist_func_()** (line 054) which will at this time return **(mark ("place") == 0 ? 1.0 : 0.0)**. The next iteration of the for-loop provides us with the mean value of the function **(mark ("place") == 1 ? 1.0 : 0.0)** and so on until the for-loop reaches the capacity of the current node, ,,3" in our case.

So this works quite fine and supplies the user with correct results[6]:

```
VALUE: DIST_ place 0 = 0.533333333333
VALUE: DIST_ place 1 = 0.266666666667
VALUE: DIST_ place 2 = 0.133333333333
VALUE: DIST_ place 3 = 0.0666666666667
```

Unfortunately this will not work in SPNP simulation mode, as the following ,,results" demonstrate[7]:

```
VALUE: DIST_ place 0 = 0
VALUE: DIST_ place 1 = 0
VALUE: DIST_ place 2 = 0
VALUE: DIST_ place 3 = 0
```

Providing **ac_final()** with a **fprintf()** debug output, it can be observed, that the debug output appears *twice* while simulating the CSPL model with SPNP. The first output appears shortly after the tool is started, independent from the choice of the parameters **IOP_SIM_RUNS** and **FOP_SIM_LENGTH**. The second output shows up shortly before SPNP is finished. So in simulation mode **ac_final()** is called twice: The first time before the start of the actual discrete event simulation to ,,collect" the functions, that have to be evaluated during simulation, the second time after the discrete event simulation has finished to present the results.

In other words: The simulation does neither construct the whole state space nor memorizes all states that where reached. The reachability graph is not built. So each computation has to be done for each single state when it is reached. Therefore, it must be known which results are desired, before the discrete event simulation starts. This is done by noting the marking dependent reward functions specified in each **expected()** or similar function.

The consequences: The global variables are useless, because for each reward function only the last combination of global variables is memorized after the first run of **ac_final()**. But this combination of global variables is not even a valid one. For example the global variable **dist_i_** introduced above, is during simulation not even ,,3" as you would expect but ,,4", because the break condition of the for-loop (,,> 3") has to be fulfilled before the simulation can start. So the simulation will not create a single useful result.

[4]**int dist_i_** is meant as the current number of jobs in the node.

[5]**expected()** computes the mean of the return values of the (mandatory parameterless) function that was given as parameter. The mean is calculated referring to the whole state space of the MRM.

[6]This is a snapshot of the SPNP output. Blank lines were deleted.

[7]The same CSPL file was used. Only the simulation options **iopt(IOP_SIMULATION, VAL_YES)**, **iopt(IOP_SIM_RUNS, 1000)** and **fopt(FOP_SIM_LENGTH, 100)** were set.

The workaround: Due to the fact, that global variables cannot be used in simulation mode to emulate multiple reward functions by one, we have to build one reward function (and a appropriate **expected()**-call to it) for each possible combination of global variables. This is done by the CSPL generator in MOSEL-2 2.0 (→ Chap. C.10, P. 56).

The simulation will now produce reasonable results:

```
VALUE: DIST_ place 0 = 0.512
VALUE: DIST_ place 1 = 0.284
VALUE: DIST_ place 2 = 0.127
VALUE: DIST_ place 3 = 0.077
```

The CSPL file shows this constructs now:

```
[..]
056: double dist_func_sim_place_0_ (void)
057: {
058:   return (mark ("place") == 0 ? 1.0 : 0.0);
059: }
060:
061: double dist_func_sim_place_1_ (void)
062: {
063:   return (mark ("place") == 1 ? 1.0 : 0.0);
064: }
065:
066: double dist_func_sim_place_2_ (void)
067: {
068:   return (mark ("place") == 2 ? 1.0 : 0.0);
069: }
070:
071: double dist_func_sim_place_3_ (void)
072: {
073:   return (mark ("place") == 3 ? 1.0 : 0.0);
074: }
075:
076: void print_dist_ (char *node, int capacity)
077: {
078:
079:   if ( strcmp (node, "place") == 0 )
080:   {
081:     pr_value ("DIST_ place 0", expected (dist_func_sim_place_0_));
082:     pr_value ("DIST_ place 1", expected (dist_func_sim_place_1_));
083:     pr_value ("DIST_ place 2", expected (dist_func_sim_place_2_));
084:     pr_value ("DIST_ place 3", expected (dist_func_sim_place_3_));
085:   }
086:
087: }
[..]
108: void ac_final (void)
109: {
[..]
114: print_dist_ ("place", 3);
115:}
```

Of course this method has a big disadvantage: The CSPL-file grows huge[8] when job distributions of many places with large capacity are computed. Therefore, this method is only used in simulation mode. Although the new method would produce correct results even in numerical analysis, the more compact method including global variables is used there.

4.8 Setting Simulation Option Defaults

To provide more user-friendliness we will set some SPNP simulation options by default if they were not given by the user at command line. The new defaults are shown in table 3.1 on page 27. They are set by the CSPL generator (→ Chap. C.7, P. 54).

[8] An estimation is given in section 5.2.

Chapter 5

Testing of MOSEL-2 2.0

In this chapter we will use the simple Petri net of Figure 2.1 to perform some MOSEL-2 functionality, correctness, performance and memory usage testing.

5.1 Testing all new Distributions and Results

At first we will test, if all distributions are working and if their simulation provides reasonable results. Therefore, the second transition of the simple tandem Petri net introduced above (Figure 2.1) will be substituted by all possible distributions (→ Chap. 2.4, P. 11) successively. See section D.3 for a example using exponential (rate=4) and deterministic firing distribution. The deterministic transition will be substituted with the particular distribution out of section 2.4.

The hence emerging versions of the ESPN system are then computed using the new MOSEL-2 modeling environment in SPNP simulation mode. Afterwards some results are compared with the results provided by other solution methods, e.g. MOSEL-2 using the numerical analysis of SPNP or TimeNET:

Distribution	Parameters	Results[1]				Comparison			
		$\bar{k_1}$	$\bar{k_2}$	ρ_1	ρ_2	$\bar{k_1}$	$\bar{k_2}$	ρ_1	ρ_2
Beta	BETA (1.2, 3.8)	1.513	1.487	0.774	0.759				
Binomial[2]	BINOM (10, 0.5, 1)	3	0	1	0				
Cauchy	CAUCHY (0.2, 0.01)	1.566	1.434	0.798	0.767				
Deterministic	AFTER 0.25	1.481	1.519	0.828	0.826	1.479	1.521	0.824	0.824[5]
Erlang[2]	ERLANG (10, 2)	3	0	1	0	1.79	1.21	0.858	0.683[6]
Exponential	RATE 4	1.55	1.45	0.761	0.736	1.5	1.5	0.75	0.75[4]
Gamma	GAMMA (2, 6)	1.503	1.497	0.752	0.753				
Geometric	GEOM (0.4, 0.1)	1.525	1.475	0.791	0.767				
Hyperexponential	HYPER (3.5, 4.8, 0.55)	1.493	1.507	0.741	0.753				
Hypoexponential[3]	HYPO (7.8, 7.8, 1, 2)	1.495	1.505	0.786	0.779				
Lognormal	LOGN (0.25, 1.6)	1.508	1.492	0.71	0.708				
Normal	NORM (0.25, 0.01)	1.483	1.517	0.832	0.835				
Pareto	PARETO (0.2, 5.5)	1.511	1.489	0.824	0.815				
Poisson	POIS (0.25, 1)	1.485	1.515	0.61	0.61				

Result symbols:

- **$\bar{k_1}$:** average number of jobs (i.e. tokens) in first node (i.e. place)
- **$\bar{k_2}$:** average number of jobs in second node
- **ρ_1:** utilization of first node
- **ρ_2:** utilization of second node

5.2 Testing MOSEL-2 Performance and Memory Requirements

5.2.1 Testing Conditions

All performance test were made using the i606 PC *faui06o.informatik.uni-erlangen.de* with i386-Linux (kernel 2.4.21), *Intel*® *Pentium*® 4 CPU (2392.326 MHz, 4771.02 BogoMips[7]) and 506004 kB RAM.

The timing tests were done with *time(1)*. The memory usage was estimated using *mempeak*[8] which was slightly adjusted to provide a faster sampling rate.

[1] MOSEL-2 using SPNP simulation (`SIM_RUNS=1000, SIM_LENGTH=1000`).
[2] This distribution does not work properly due to a bug in SPNP 6.1.2.a. See appendix B for details.
[3] Note that there is an error in [3]. The order of parameters is not as it is described in the manual (p.27).
[4] Results provided by SPNP using numerical analysis.
[5] Results provided by TimeNET using continuous time stationary analysis.
[6] Results provided by SPNP simulation using equivalent hypoexponential distribution.
[7] ,,the number of million times per second a processor can do absolutely nothing" (quoted from the internet, http://www.hobby.nl/~clifton/bogomips-2.html, 19.06.03, origin unknown)
[8] http://www.rninet.de/darkstar/mempeak.html, 12.06.03

The small MOSEL-2 model description described in appendix D.9 was tested.

5.2.2 Test Results

Eval. Method	**Model Modifications**	**Memory**		**Times**[1]		
		RAM[2]	harddisk[3]	real	user	system
SPNP analysis	*unchanged*[4]	1568k, 21574k	1815, 535539	0.897	0.108	0.044
SPNP simulation 1000/1000[5]	*unchanged*	1568k, 24416k	1946, 535098	8.864	8.156	0.046
SPNP simulation 2000/1000	*unchanged*	1568k, 23280k	1946, 535110	16.989	16.222	0.034
SPNP simulation 1000/2000	*unchanged*	1568k, 23140k	1946, 535098	17.601	16.198	0.052
TimeNET analysis	*unchanged*	1568k, 25272k	977, 768	1.863	0.228	0.092
SPNP analysis	3 jobs, one *DIST* measure	1568k, 20088k	2477, 540833	1.005	0.103	0.045
SPNP simulation 1000/1000	3 jobs, one *DIST* measure	1568k, 21812k	3237, 541042	9.618	8.348	0.046
SPNP analysis	3 jobs, two *DIST* measures	–[6]	2506, –	–	–	–
SPNP simulation 1000/1000	3 jobs, two *DIST* measures	–	4308, –	–	–	–
SPNP analysis	30 jobs, no *DIST* measures	–	1802, –	–	–	–
SPNP simulation 1000/1000	30 jobs, no *DIST* measures	–	1951, –	–	–	–
SPNP analysis	30 jobs, one *DIST* measure	–	2483, –	–	–	–
SPNP simulation 1000/1000	30 jobs, one *DIST* measure	–	9768, –	–	–	–
SPNP analysis	30 jobs, two *DIST* measures	1568k, 23964	2513, –	0.751	0.114	0.044
SPNP simulation 1000/1000	30 jobs, two *DIST* measures	1568k, 23816	17365, –	22,956	21.368	0.054

(The table shows the average results of five test runs.)

[1]in seconds

[2]memory usage of MOSEL in bytes, sum of memory usage in bytes of the tool called by MOSEL and of subprocesses called by this tool (note that this is only an upper bound for the maximal memory that is used at the same time but should stand for drawing comparisons)

[3]size of tool specific model description file (*.c, *.TN) in bytes, sum of size in bytes of all intermediate files produced by the tool called by MOSEL

[4]3 jobs in system, no *DIST* measures

[5]number of runs / simulation length

[6]data not collected (not of interest)

5.2.3 Test Conclusion

The following facts could be derived from the test results:

- The memory usage of MOSEL is independent from the system size.
- The number of jobs does not (yet) have a remarkable effect on the memory used by the evaluating tool, neither in analysis nor in simulation mode.
- A MOSEL model description (in this test: 570 bytes) is much more compact than a CSPL model description (in this test: 1815 to 17365 bytes).
- Using MOSEL with SPNP analysis, *DIST* measures do not result in a significant enlargement of the CSPL model which is the result of the ability to use global variables (→ Chap. 4.7, P. 32).
- Using *DIST* measures with MOSEL and SPNP simulation, the filesize S of the generated CSPL model description file is approximately:

$$\boxed{S \approx ((J+1) \cdot N \cdot 236 \text{ bytes}) + (N \cdot 124 \text{ bytes}) + 346 \text{ bytes} + A} \tag{5.1}$$

  with:

  N : number of nodes, whose job distributions have to be computed,

  J : maximal number of jobs in the system, i.e. capacity of each node,

  A : size (in bytes) of CSPL model description without *DIST* measures and simulation constructs (this example: 1820 bytes),

  236 bytes : marking dependent reward functions and *expected()* calls,

  124 bytes : *print_dist_()* calls and node specific *print_dist_()* parts,

  346 bytes : size of standard simulation options (131 bytes) plus *DIST* measures framework (215 bytes).

- Therefore, *DIST* measures should be used wisely when simulation large Petri nets with a lot of tokens/jobs.
- The duration of the simulation depends on the simulation length and on the number of simulation runs likewise.

Chapter 6

Conclusion and Future Work

As we have seen, MOSEL-2 is now able to evaluate not only (extended) deterministic stochastic Petri nets (eDSPNs) but finally a lot of extended stochastic Petri nets (ESPNs) by introducing twelve new distribution types. So MOSEL-2's modeling power is increased substantially from Markov chains to Generalized semi-Markov processes.

A lot of time and work was invested to obtain this goal, whereupon I must say that the pure implementation of the new constructs was not the biggest problem. I had to spend a lot of time with getting along with the SPNP user's manual [3] which is quite faulty and inconsistent and with bugs and restrictions of SPNP 6.1.2.a. A lot of work also has been done to old MOSEL. Most of this work had to be discarded after I was told to work on the new MOSEL-2 version introduced by Björn Beutel in January 2003. Unfortunately at that time MOSEL-2 was not a completely worked out, final version, so I had to merge my own changes to the constantly appearing new versions of MOSEL-2 several times, and sometimes I had to wonder, if bugs that occurred where my fault or where based on the incomplete MOSEL-2 source.

As a result of the final lack of time I unfortunately could test neither SPNP's special methods for the simulation of rare events, i.e. importance splitting / sampling nor its ability to evaluate FSPNs, nor check their suitability for the MOSEL language properly. So this will have to be done in future work.

Another useful thing, that can be done, is the introduction of distributions, that will (perhaps) be supported by oncoming SPNP versions. With the information provided by this thesis, this should be no huge problem. It might even make sense to step back to SPNP version 6.1 that seems to support more distributions and seems to be less buggy than version 6.1.2a.

Further on some limitations of MOSEL-2 attracted my attention that were not present in old MOSEL. Above all there are some restrictions to the *picture part* that are quite shortening its mightiness:

- **LIST :** *LIST* is no longer supported by MOSEL-2. So the user is not able to specify single values to be printed as bar graph.
- **DIST :** In MOSEL-2 only a single nodes distribution can be printed as one graph into one diagram.

- **RESULT DIST :** There is no *RESULT DIST* in MOSEL-2. So job distributions can only be calculated if they are printed.

Perhaps anyone can make this MOSEL features also accessible to MOSEL-2.

Nevertheless I finally have to say that Björn has done a great work, and so has Helmut Herold, the inventor of MOSEL.

Appendix A

MOSEL-2 Language Extensions

A.1 Distributions

As you can read in [4], the rules in MOSEL-2 consist of several rule parts. Following new rule parts were introduced to handle further distributions of firing times. All following rule parts may only occur once in a rule definition. For a description of these distributions see (→ Chap. 2.4, P. 11).

- **BETA part**

  rule-part ::= "BETA" "(" *alpha* "," *beta* ")" .
  alpha ::= *const-expr* .
  beta ::= *const-expr* .

  This part states that the rule has beta distributed firing time after enabling.

- **BINO part**

  rule-part ::= "BINO" "(" *number_of_trials* "," *probability* "," *weighting* ")" .
  number of trials ::= *const-expr* .
  probability ::= *const-expr* .
  weighting ::= *const-expr* .

  This part states that the rule has binomial distributed firing time after enabling.

- **CAU part**

  rule-part ::= "CAU" "(" *alpha* "," *beta* ")" .
  alpha ::= *const-expr* .
  beta ::= *const-expr* .

  This part states that the rule has Cauchy distributed firing time after enabling.

- **ERLANG part**

  rule-part ::= “ERLANG” “(” *rate* “,” *number_of_phases* “)”.
  rate ::= *const-expr* .
  number of phases ::= *const-expr* .

  This part states that the rule has Erlang distributed firing time after enabling.

- **GAM part**

  rule-part ::= “GAM” “(” *alpha* “,” *lambda* “)”.
  alpha ::= *const-expr* .
  lambda ::= *const-expr* .

  This part states that the rule has gamma distributed firing time after enabling.

- **GEOM part**

  rule-part ::= “GEOM” “(” *probability* “,” *weighting* “)”.
  probability ::= *const-expr* .
  weighting ::= *const-expr* .

  This part states that the rule has geometric distributed firing time after enabling.

- **HYPER part**

  rule-part ::= “HYPER” “(” *rate* “,” *rate* “,” *probability* “)”.
  rate ::= *const-expr* .
  probability ::= *const-expr* .

  This part states that the rule has hyperexponential distributed firing time after enabling.

- **HYPO part**

  rule-part ::= “HYPO” “(” *rate* “,” *rate* “,” *rate* “,” *number_of_stages* “)” .
  rate ::= *const-expr* .
  number of stages ::= *const-expr* .

  This part states that the rule has hypoexponential distributed firing time after enabling.

- **LOGN part**

  rule-part ::= “LOGN” “(” *location* “,” *shape* “)” .
  location ::= *const-expr* .
  shape ::= *const-expr* .

  This part states that the rule has lognormal distributed firing time after enabling.

- **NORM part**

  rule-part ::= "NORM" "(" *location* "," *shape* ")" .
  location ::= *const-expr* .
  shape ::= *const-expr* .

  This part states that the rule has normal distributed firing time after enabling.

- **PARETO part**

  rule-part ::= "PARETO" "(" *alpha* "," *beta* ")" .
  alpha ::= *const-expr* .
  beta ::= *const-expr* .

  This part states that the rule has Pareto distributed firing time after enabling.

- **POIS part**

  rule-part ::= "POIS" "(" *mean* "," *weighting* ")" .
  mean ::= *const-expr* .
  weighting ::= *const-expr* .

  This part states that the rule has Poisson distributed firing time after enabling.

A.2 Resampling Policies

Furthermore the policy keywords had to be extended. Besides "PRD" and "PRS" we are now able to use

rule-part ::= "PRI" .

under SPNP simulation.

Appendix B

Non-working SPNP Distributions

B.1 Uniform Distribution

Simulating a CSPL test file containing the line:

```
unifval("timedtrans_1_", 5, 15);
```

using SPNP 6.1.2a, the following error is produced:

```
ERROR: Distribution not implemented yet
EXIT: Internal error, please contact the developers
```

B.2 Weibull Distribution

Simulating a CSPL test file containing the line:

```
weibval("timedtrans_1_", 5, 15);
```

using SPNP 6.1.2a, the following error is produced:

```
ERROR: Distribution not implemented yet
EXIT: Internal error, please contact the developers
```

B.3 Negative Binomial Distribution

Simulating a CSPL test file containing the line:

```
negbval("timedtrans_1_", 5, 15, 20);
```

using SPNP 6.1.2a, the following error is produced:

```
undefined reference to ‘negbval’
```

B.4 Binomial Distribution

Binomial distributed transitions always behave like immediate transitions regardless of which parameters were given. This is a problem of SPNP 6.1.2a and is independent from MOSEL-2.

Using the GNU debugger *gdb* the problem can be traced back to a function called *get_random_sample()* in SPNP 6.1.2a (file sim.c, line 732). After this function is called with appropriate parameters for binomial distributions, e.g. *(type=DIS_BIN, val=10, val2=0.5, val3=1, val4=0)*, it calls function *binomial()* (file randfunction.c, line 463) with parameters *(n=0, p=0.5)*. Because of *n=0*, and not *n=10* as it should be, this function always returns the value *0*. Only if we set the parameter *n* to *10* manually by using *gdb*-features, we will get reasonable results from function *binomial()* (e.g. the values *5, 4, 5, 4, 3* while testing).

Unfortunately this bug could neither be verified by rummaging in the source code of SPNP nor be fixed there, because the code was not available.

B.5 Erlang Distribution

Like binomial distributed transitions also Erlang distributed transitions always behave like immediate transitions. This is a further problem of SPNP 6.1.2a.

Analog to the binomial distribution, it can be shown, that the function *get_random_sample()* is called correctly with the parameters *(type=DIS_ERL, val=10, val2=2, val3=0, val4=0)* but calls function *erlang()* with wrong parameters *(u=10, k=0)*. If we are setting *k=2* manually, we get reasonable results.

Fortunately HYPO(val1, val1, 1, 2) is equivalent to ERLANG(val1, 2), and HYPO(val1, val1, val1, 3) is equivalent to ERLANG(val1, 3). So at least Erlang distributions with two or three stages can be easily replaced by hyperexponential distributions. Other Erlang distributions can be modeled by using exponential distributions. See [1] for details.

Appendix C

MOSEL-2 Code Adaptations

C.1 basic.h - line 6 et sqq.

Lines 6 and 7 have been inserted to all touched files. They are not mentioned again in the following sections.

```
6:/* Adaptations from version 1.0 to 2.s (supporting SPNP simulation)
7: * 2003 by Patrick Wuechner (are marked with 'PW')                  */
[...]
20:#define VERSION_INFO "2.0 (2003-06-01)" /* PW */
```

C.2 cmdline.c - line 25 et sqq.

```
25:/* CSPL options (with possible values) */
26:static struct cspl_option
27:{ char *name; char *values; }
28:cspl_options[] =
29:{
[...]
59:  /* CSPL simulation options                    - PW **/
60:  {"SIMULATION",      "YES NO"},                    /**/
61:  {"SIM_RUNMETHOD",   "REPL BATCH"},                /**/
62:  {"SIM_LENGTH",      "double"},                    /**/
63:  {"SIM_STD_REPORT",  "YES NO"},                    /**/
64:  {"SIM_CUMULATIVE",  "YES NO"},                    /**/
65:  {"SIM_CONFIDENCE",  "double"},                    /**/
66:  {"SIM_RUNS",        "int"},                       /**/
67:  {"SIM_ERROR",       "double"},                    /**/
68:  {"SIM_SEED",        "int"},                       /**/
69:  /**************************************************/
70:};
```

C.3 cmdline.c - line 164 et sqq.

```
164:static void
165:process_cspl_option (char *arg)
166:{
[...]
178:  if (arg[0] == '+' || arg[0] == '-')
```

```
179:  {
180:    /* Process a +/- option. */
[...]
186:    /* if option '+SIMULATION' is set, change cspl_mode        - PW **/
187:    /* to CSPL_SIMULATION                                     */ /**/
188:    if (strcmp ("SIMULATION", name) == 0 && arg[0] == '+')         /**/
189:         cmdline.cspl_mode = CSPL_SIMULATION;                      /**/
190:    if (strcmp ("SIMULATION", name) == 0 && arg[0] == '-')         /**/
191:         cmdline.cspl_mode = CSPL_NORMAL;                          /**/
192:    /****************************************************************/
```

C.4 cmdline.c - line 342 et sqq.

```
342:process_command_line (int argc, char *argv[])
343:{
[...]
351:  /* Set command line defaults. */
[...]
354:  /* default method for CSPL: not simulated - PW **/
355:  cmdline.cspl_mode = CSPL_NORMAL;             /**/
356:  /***********************************************/
```

C.5 cmdline.h - line 39 et sqq.

```
39:  /* different SPNP modes will allow different distributions - PW **/
40:  enum { CSPL_NORMAL, CSPL_SIMULATION } cspl_mode;              /**/
41:  /****************************************************************/
```

C.6 cspl.c - line 299 et sqq.

```
302:static void
303:print_rate_functions (void)
304:{
[...]
311:    /* to insert other distributions provided by (SPNP Simulation)                  - PW **/
312:    /* this code: */                                                                     /**/
313:                                                                                         /**/
314:    /* ------------------------ %< ---------------------------------- */             /**/
315:    /* if (rule->distrib_type != DIST_EXP                             */             /**/
316:    /*  || expr_is_const (rule->distrib_param[0]))                    */             /**/
317:    /*  {                                                             */             /**/
318:    /*    continue;                                                   */             /**/
319:    /*  }                                                             */             /**/
320:    /*  fprintf (stream, "\ndouble rate_func_%d_ (void)\n{\n", i);      */             /**/
321:    /*  fprintf (stream, "  return ");                                */             /**/
322:    /*  print_cspl_expr (rule->distrib_param[0], 0);                  */             /**/
323:    /*  fprintf (stream, ";\n}\n");                                   */             /**/
324:    /* ------------------------ %< ---------------------------------- */             /**/
325:                                                                                         /**/
326:    /* had to be changed into this: */                                                   /**/
327:                                                                                         /**/
328:    if ( rule->distrib_type == DIST_IMMED )                                              /**/
329:    {                                                                                    /**/
330:       continue;                                                                         /**/
331:    }                                                                                    /**/
332:    else                                                                                 /**/
333:    {                                                                                    /**/
334:      switch (rule->distrib_type)                                                        /**/
```

```
335:     {                                                                    /**/
336:        case DIST_EXP :                                                   /**/
337:              if ( !expr_is_const(rule->distrib_param[0]) ) {             /**/
338:               fprintf (stream, "\ndouble rate_func_%d_ (void)\n{\n", i); /**/
339:               fprintf (stream, "  return ");                             /**/
340:               print_cspl_expr (rule->distrib_param[0], 0);               /**/
341:               fprintf (stream, ";\n}\n");                                /**/
342:              }                                                           /**/
343:               break;                                                     /**/
344:                                                                          /**/
345:        case DIST_DETERM :                                                /**/
346:              if ( !expr_is_const(rule->distrib_param[0]) ) {             /**/
347:               fprintf (stream, "\ndouble det_func_%d_ (void)\n{\n", i);  /**/
348:               fprintf (stream, "  return ");                             /**/
349:               print_cspl_expr (rule->distrib_param[0], 0);               /**/
350:               fprintf (stream, ";\n}\n");                                /**/
351:              }                                                           /**/
352:               break;                                                     /**/
353:                                                                          /**/
354:        case DIST_GEOM :                                                  /**/
355:              if ( !expr_is_const(rule->distrib_param[0]) ) {             /**/
356:               fprintf (stream, "\ndouble geom_func_%d_1 (void)\n{\n", i); /**/
357:               fprintf (stream, "  return ");                             /**/
358:               print_cspl_expr (rule->distrib_param[0], 0);               /**/
359:               fprintf (stream, ";\n}\n");                                /**/
360:              }                                                           /**/
361:              if ( !expr_is_const(rule->distrib_param[1]) ) {             /**/
362:               fprintf (stream, "\ndouble geom_func_%d_2 (void)\n{\n", i); /**/
363:               fprintf (stream, "  return ");                             /**/
364:               print_cspl_expr (rule->distrib_param[1], 0);               /**/
365:               fprintf (stream, ";\n}\n");                                /**/
366:              }                                                           /**/
367:               break;                                                     /**/
368:                                                                          /**/
369:        case DIST_NORM :                                                  /**/
370:              if ( !expr_is_const(rule->distrib_param[0]) ) {             /**/
371:               fprintf (stream, "\ndouble norm_func_%d_1 (void)\n{\n", i); /**/
372:               fprintf (stream, "  return ");                             /**/
373:               print_cspl_expr (rule->distrib_param[0], 0);               /**/
374:               fprintf (stream, ";\n}\n");                                /**/
375:              }                                                           /**/
376:              if ( !expr_is_const(rule->distrib_param[1]) ) {             /**/
377:               fprintf (stream, "\ndouble norm_func_%d_2 (void)\n{\n", i); /**/
378:               fprintf (stream, "  return ");                             /**/
379:               print_cspl_expr (rule->distrib_param[1], 0);               /**/
380:               fprintf (stream, ";\n}\n");                                /**/
381:              }                                                           /**/
382:               break;                                                     /**/
383:                                                                          /**/
384:        case DIST_LOGN :                                                  /**/
385:              if ( !expr_is_const(rule->distrib_param[0]) ) {             /**/
386:               fprintf (stream, "\ndouble logn_func_%d_1 (void)\n{\n", i); /**/
387:               fprintf (stream, "  return ");                             /**/
388:               print_cspl_expr (rule->distrib_param[0], 0);               /**/
389:               fprintf (stream, ";\n}\n");                                /**/
390:              }                                                           /**/
391:              if ( !expr_is_const(rule->distrib_param[1]) ) {             /**/
392:               fprintf (stream, "\ndouble logn_func_%d_2 (void)\n{\n", i); /**/
393:               fprintf (stream, "  return ");                             /**/
394:               print_cspl_expr (rule->distrib_param[1], 0);               /**/
395:               fprintf (stream, ";\n}\n");                                /**/
396:              }                                                           /**/
397:               break;                                                     /**/
398:                                                                          /**/
399:        case DIST_ERLANG :                                                /**/
400:              if ( !expr_is_const(rule->distrib_param[0]) ) {             /**/
401:               fprintf (stream, "\ndouble erl_func_%d_1 (void)\n{\n", i); /**/
```

```
402:                fprintf (stream, "  return ");                                        /**/
403:                print_cspl_expr (rule->distrib_param[0], 0);                          /**/
404:                fprintf (stream, ";\n}\n");                                            /**/
405:               }                                                                       /**/
406:               if ( !expr_is_const(rule->distrib_param[1]) ) {                         /**/
407:                fprintf (stream, "\ndouble erl_func_%d_2 (void)\n{\n", i);              /**/
408:                fprintf (stream, "  return ");                                        /**/
409:                print_cspl_expr (rule->distrib_param[1], 0);                          /**/
410:                fprintf (stream, ";\n}\n");                                            /**/
411:               }                                                                       /**/
412:                break;                                                                 /**/
413:                                                                                       /**/
414:         case DIST_GAMMA :                                                             /**/
415:               if ( !expr_is_const(rule->distrib_param[0]) ) {                         /**/
416:                fprintf (stream, "\ndouble gam_func_%d_1 (void)\n{\n", i);              /**/
417:                fprintf (stream, "  return ");                                        /**/
418:                print_cspl_expr (rule->distrib_param[0], 0);                          /**/
419:                fprintf (stream, ";\n}\n");                                            /**/
420:               }                                                                       /**/
421:               if ( !expr_is_const(rule->distrib_param[1]) ) {                         /**/
422:                fprintf (stream, "\ndouble gam_func_%d_2 (void)\n{\n", i);              /**/
423:                fprintf (stream, "  return ");                                        /**/
424:                print_cspl_expr (rule->distrib_param[1], 0);                          /**/
425:                fprintf (stream, ";\n}\n");                                            /**/
426:               }                                                                       /**/
427:                break;                                                                 /**/
428:                                                                                       /**/
429:         case DIST_BETA :                                                              /**/
430:               if ( !expr_is_const(rule->distrib_param[0]) ) {                         /**/
431:                fprintf (stream, "\ndouble bet_func_%d_1 (void)\n{\n", i);              /**/
432:                fprintf (stream, "  return ");                                        /**/
433:                print_cspl_expr (rule->distrib_param[0], 0);                          /**/
434:                fprintf (stream, ";\n}\n");                                            /**/
435:               }                                                                       /**/
436:               if ( !expr_is_const(rule->distrib_param[1]) ) {                         /**/
437:                fprintf (stream, "\ndouble bet_func_%d_2 (void)\n{\n", i);              /**/
438:                fprintf (stream, "  return ");                                        /**/
439:                print_cspl_expr (rule->distrib_param[1], 0);                          /**/
440:                fprintf (stream, ";\n}\n");                                            /**/
441:               }                                                                       /**/
442:                break;                                                                 /**/
443:                                                                                       /**/
444:         case DIST_CAUCHY :                                                            /**/
445:               if ( !expr_is_const(rule->distrib_param[0]) ) {                         /**/
446:                fprintf (stream, "\ndouble cau_func_%d_1 (void)\n{\n", i);              /**/
447:                fprintf (stream, "  return ");                                        /**/
448:                print_cspl_expr (rule->distrib_param[0], 0);                          /**/
449:                fprintf (stream, ";\n}\n");                                            /**/
450:               }                                                                       /**/
451:               if ( !expr_is_const(rule->distrib_param[1]) ) {                         /**/
452:                fprintf (stream, "\ndouble cau_func_%d_2 (void)\n{\n", i);              /**/
453:                fprintf (stream, "  return ");                                        /**/
454:                print_cspl_expr (rule->distrib_param[1], 0);                          /**/
455:                fprintf (stream, ";\n}\n");                                            /**/
456:               }                                                                       /**/
457:                break;                                                                 /**/
458:                                                                                       /**/
459:         case DIST_POIS :                                                              /**/
460:               if ( !expr_is_const(rule->distrib_param[0]) ) {                         /**/
461:                fprintf (stream, "\ndouble pois_func_%d_1 (void)\n{\n", i);             /**/
462:                fprintf (stream, "  return ");                                        /**/
463:                print_cspl_expr (rule->distrib_param[0], 0);                          /**/
464:                fprintf (stream, ";\n}\n");                                            /**/
465:               }                                                                       /**/
466:               if ( !expr_is_const(rule->distrib_param[1]) ) {                         /**/
467:                fprintf (stream, "\ndouble pois_func_%d_2 (void)\n{\n", i);             /**/
468:                fprintf (stream, "  return ");                                        /**/
```

```
469:               print_cspl_expr (rule->distrib_param[1], 0);                     /**/
470:               fprintf (stream, ";\n}\n");                                      /**/
471:              }                                                                 /**/
472:               break;                                                           /**/
473:                                                                                /**/
474:        case DIST_PARETO :                                                      /**/
475:              if ( !expr_is_const(rule->distrib_param[0]) ) {                   /**/
476:               fprintf (stream, "\ndouble par_func_%d_1 (void)\n{\n", i);       /**/
477:               fprintf (stream, "  return ");                                   /**/
478:               print_cspl_expr (rule->distrib_param[0], 0);                     /**/
479:               fprintf (stream, ";\n}\n");                                      /**/
480:              }                                                                 /**/
481:              if ( !expr_is_const(rule->distrib_param[1]) ) {                   /**/
482:               fprintf (stream, "\ndouble par_func_%d_2 (void)\n{\n", i);       /**/
483:               fprintf (stream, "  return ");                                   /**/
484:               print_cspl_expr (rule->distrib_param[1], 0);                     /**/
485:               fprintf (stream, ";\n}\n");                                      /**/
486:              }                                                                 /**/
487:               break;                                                           /**/
488:                                                                                /**/
489:        case DIST_BINOM :                                                       /**/
490:              if ( !expr_is_const(rule->distrib_param[0]) ) {                   /**/
491:               fprintf (stream, "\ndouble bino_func_%d_1 (void)\n{\n", i);      /**/
492:               fprintf (stream, "  return ");                                   /**/
493:               print_cspl_expr (rule->distrib_param[0], 0);                     /**/
494:               fprintf (stream, ";\n}\n");                                      /**/
495:              }                                                                 /**/
496:              if ( !expr_is_const(rule->distrib_param[1]) ) {                   /**/
497:               fprintf (stream, "\ndouble bino_func_%d_2 (void)\n{\n", i);      /**/
498:               fprintf (stream, "  return ");                                   /**/
499:               print_cspl_expr (rule->distrib_param[1], 0);                     /**/
500:               fprintf (stream, ";\n}\n");                                      /**/
501:              }                                                                 /**/
502:              if ( !expr_is_const(rule->distrib_param[2]) ) {                   /**/
503:               fprintf (stream, "\ndouble bino_func_%d_3 (void)\n{\n", i);      /**/
504:               fprintf (stream, "  return ");                                   /**/
505:               print_cspl_expr (rule->distrib_param[2], 0);                     /**/
506:               fprintf (stream, ";\n}\n");                                      /**/
507:              }                                                                 /**/
508:               break;                                                           /**/
509:                                                                                /**/
510:        case DIST_HYPER :                                                       /**/
511:              if ( !expr_is_const(rule->distrib_param[0]) ) {                   /**/
512:               fprintf (stream, "\ndouble hyper_func_%d_1 (void)\n{\n", i);     /**/
513:               fprintf (stream, "  return ");                                   /**/
514:               print_cspl_expr (rule->distrib_param[0], 0);                     /**/
515:               fprintf (stream, ";\n}\n");                                      /**/
516:              }                                                                 /**/
517:              if ( !expr_is_const(rule->distrib_param[1]) ) {                   /**/
518:               fprintf (stream, "\ndouble hyper_func_%d_2 (void)\n{\n", i);     /**/
519:               fprintf (stream, "  return ");                                   /**/
520:               print_cspl_expr (rule->distrib_param[1], 0);                     /**/
521:               fprintf (stream, ";\n}\n");                                      /**/
522:              }                                                                 /**/
523:              if ( !expr_is_const(rule->distrib_param[2]) ) {                   /**/
524:               fprintf (stream, "\ndouble hyper_func_%d_3 (void)\n{\n", i);     /**/
525:               fprintf (stream, "  return ");                                   /**/
526:               print_cspl_expr (rule->distrib_param[2], 0);                     /**/
527:               fprintf (stream, ";\n}\n");                                      /**/
528:              }                                                                 /**/
529:               break;                                                           /**/
530:                                                                                /**/
531:        case DIST_HYPO :                                                        /**/
532:              if ( !expr_is_const(rule->distrib_param[0]) ) {                   /**/
533:               fprintf (stream, "\ndouble hypo_func_%d_1 (void)\n{\n", i);      /**/
534:               fprintf (stream, "  return ");                                   /**/
535:               print_cspl_expr (rule->distrib_param[0], 0);                     /**/
```

```
536:                fprintf (stream, ";\n}\n");                                     /**/
537:               }                                                                 /**/
538:               if ( !expr_is_const(rule->distrib_param[1]) ) {                   /**/
539:                fprintf (stream, "\ndouble hypo_func_%d_2 (void)\n{\n", i);      /**/
540:                fprintf (stream, "  return ");                                   /**/
541:                print_cspl_expr (rule->distrib_param[1], 0);                     /**/
542:                fprintf (stream, ";\n}\n");                                      /**/
543:               }                                                                 /**/
544:               if ( !expr_is_const(rule->distrib_param[2]) ) {                   /**/
545:                fprintf (stream, "\ndouble hypo_func_%d_3 (void)\n{\n", i);      /**/
546:                fprintf (stream, "  return ");                                   /**/
547:                print_cspl_expr (rule->distrib_param[2], 0);                     /**/
548:                fprintf (stream, ";\n}\n");                                      /**/
549:               }                                                                 /**/
550:               if ( !expr_is_const(rule->distrib_param[3]) ) {                   /**/
551:                fprintf (stream, "\ndouble hypo_func_%d_3 (void)\n{\n", i);      /**/
552:                fprintf (stream, "  return ");                                   /**/
553:                print_cspl_expr (rule->distrib_param[2], 0);                     /**/
554:                fprintf (stream, ";\n}\n");                                      /**/
555:               }                                                                 /**/
556:                break;                                                           /**/
557:                                                                                 /**/
558:         default :                                                               /**/
559:                error_msg (FATAL, "Internal: Unknown distribution type in CSPL");  /**/
560:                                                                                 /**/
561:      }                                                                          /**/
562:    }                                                                            /**/
563:    /***************************************************************************************/
```

C.7 cspl.c - line 590 et sqq.

```
590: static void
591: print_options (void)
592: /* Print function "options" which sets options. */
593: {
[...]
596:  bool_t sim_runs_set = 0;       /* PW */
597:  bool_t sim_length_set = 0;     /* PW */
598:  bool_t sim_stdrep_set = 0;     /* PW */
[...]
617:  /* setting default simulation options if not yet set                     - PW  **/
618:  if (cmdline.cspl_mode == CSPL_SIMULATION) {                                    /**/
619:       for (i = 0; i < cmdline.cspl_option_count; i++)                          /**/
620:       {                                                                        /**/
621:         option = &cmdline.cspl_options[i];                                     /**/
622:         if ( strcmp (option->name, "SIM_RUNS") == 0 )  sim_runs_set = 1;       /**/
623:         if ( strcmp (option->name, "SIM_LENGTH") == 0 )  sim_length_set = 1;   /**/
624:         if ( strcmp (option->name, "SIM_STD_REPORT") == 0 ) sim_stdrep_set = 1; /**/
625:       }                                                                        /**/
626:       if (sim_runs_set != 1)                                                   /**/
627:          fprintf (stream, "  iopt (IOP_SIM_RUNS, 1000);\n");                   /**/
628:       if (sim_length_set != 1)                                                 /**/
629:          fprintf (stream, "  fopt (FOP_SIM_LENGTH, 1000);\n");                 /**/
630:       if (sim_stdrep_set != 1)                                                 /**/
631:          fprintf (stream, "  iopt (IOP_SIM_STD_REPORT, VAL_NO);\n");           /**/
632:  }                                                                             /**/
633:  /********************************************************************************/
```

C.8 cspl.c - line 640 et sqq.

```
640:static void
641:print_net (void)
642:{
[...]
666:  /* Define transitions. */
667:  for (i = 0; i < get_rule_count (); i++)
668:  {
[...]
677:    /* Create distribution. */
678:    if (rule->distrib_type == DIST_IMMED)
679:      fprintf (stream, "  imm (\"trans_%d_\");\n", i);
```

```
680:    else if (rule->distrib_type == DIST_EXP)
[...]
690:    /* ESPN distributions, only for SPNP Simulation                                    - PW **/
691:    else if (rule->distrib_type == DIST_DETERM) {                                       /**/
692:      if (expr_is_const (rule->distrib_param[0])) {                                     /**/
693:        fprintf (stream, "  detval (\"trans_%d_\", %g);\n",                             /**/
694:                 i, eval_const_expr (rule->distrib_param[0], current_setting));         /**/
695:      }                                                                                 /**/
696:      else                                                                              /**/
697:        fprintf (stream, "  detfun (\"trans_%d_\", det_func_%d_);\n", i, i);            /**/
698:    }                                                                                   /**/
699:    else if (rule->distrib_type == DIST_GEOM) {                                         /**/
700:       if  (expr_is_const (rule->distrib_param[0])) {                                   /**/
701:        fprintf (stream, "  geomval (\"trans_%d_\", %g, %g);\n", i,                     /**/
702:                 eval_const_expr (rule->distrib_param[0], current_setting),             /**/
703:                 eval_const_expr (rule->distrib_param[1], current_setting) );           /**/
704:       }                                                                                /**/
705:       else                                                                             /**/
706:        fprintf (stream, "  geomfun (\"trans_%d_\", geom_func_%d_1, geom_func_%d_2);\n", /**/
707:                 i, i, i);                                                              /**/
708:    }                                                                                   /**/
709:    else if (rule->distrib_type == DIST_NORM) {                                         /**/
710:       if  (expr_is_const (rule->distrib_param[0])) {                                   /**/
711:        fprintf (stream, "  normval (\"trans_%d_\", %g, %g);\n", i,                     /**/
712:                 eval_const_expr (rule->distrib_param[0], current_setting),             /**/
713:                 eval_const_expr (rule->distrib_param[1], current_setting) );           /**/
714:       }                                                                                /**/
715:       else                                                                             /**/
716:        fprintf (stream, "  normfun (\"trans_%d_\", norm_func_%d_1, norm_func_%d_2);\n", /**/
717:                 i, i, i);                                                              /**/
718:    }                                                                                   /**/
719:    else if (rule->distrib_type == DIST_LOGN) {                                         /**/
720:       if  (expr_is_const (rule->distrib_param[0])) {                                   /**/
721:          fprintf (stream, "  lognval (\"trans_%d_\", %g, %g);\n", i,                   /**/
722:                eval_const_expr (rule->distrib_param[0], current_setting),              /**/
723:                eval_const_expr (rule->distrib_param[1], current_setting) );            /**/
724:       }                                                                                /**/
725:       else                                                                             /**/
726:          fprintf (stream,                                                              /**/
727:                "  lognfun (\"trans_%d_\", logn_func_%d_1, logn_func_%d_2);\n",         /**/
728:                i, i, i);                                                               /**/
729:    }                                                                                   /**/
730:    else if (rule->distrib_type == DIST_ERLANG) {                                       /**/
731:       if  (expr_is_const (rule->distrib_param[0])) {                                   /**/
732:          fprintf (stream, "  erlval (\"trans_%d_\", %g, %g);\n", i,                    /**/
733:                eval_const_expr (rule->distrib_param[0], current_setting),              /**/
734:                eval_const_expr (rule->distrib_param[1], current_setting) );            /**/
735:       }                                                                                /**/
736:       else                                                                             /**/
737:          fprintf (stream,                                                              /**/
738:                "  erlfun (\"trans_%d_\", erl_func_%d_1, erl_func_%d_2);\n",            /**/
739:                i, i, i);                                                               /**/
740:    }                                                                                   /**/
741:    else if (rule->distrib_type == DIST_GAMMA) {                                        /**/
742:       if  (expr_is_const (rule->distrib_param[0])) {                                   /**/
743:          fprintf (stream, "  gamval (\"trans_%d_\", %g, %g);\n", i,                    /**/
744:                eval_const_expr (rule->distrib_param[0], current_setting),              /**/
745:                eval_const_expr (rule->distrib_param[1], current_setting) );            /**/
746:       }                                                                                /**/
747:       else                                                                             /**/
748:          fprintf (stream,                                                              /**/
749:                "  gamfun (\"trans_%d_\", gam_func_%d_1, gam_func_%d_2);\n",            /**/
750:                i, i, i);                                                               /**/
751:    }                                                                                   /**/
752:    else if (rule->distrib_type == DIST_BETA) {                                         /**/
753:       if  (expr_is_const (rule->distrib_param[0])) {                                   /**/
754:          fprintf (stream, "  betval (\"trans_%d_\", %g, %g);\n", i,                    /**/
755:                eval_const_expr (rule->distrib_param[0], current_setting),              /**/
756:                eval_const_expr (rule->distrib_param[1], current_setting) );            /**/
757:       }                                                                                /**/
758:       else                                                                             /**/
759:          fprintf (stream,                                                              /**/
760:                "  betfun (\"trans_%d_\", bet_func_%d_1, bet_func_%d_2);\n",            /**/
761:                i, i, i);                                                               /**/
762:    }                                                                                   /**/
763:    else if (rule->distrib_type == DIST_CAUCHY) {                                       /**/
764:       if  (expr_is_const (rule->distrib_param[0])) {                                   /**/
765:          fprintf (stream, "  cauval (\"trans_%d_\", %g, %g);\n", i,                    /**/
766:                eval_const_expr (rule->distrib_param[0], current_setting),              /**/
767:                eval_const_expr (rule->distrib_param[1], current_setting) );            /**/
768:       }                                                                                /**/
769:       else                                                                             /**/
770:          fprintf (stream,                                                              /**/
771:                "  caufun (\"trans_%d_\", cau_func_%d_1, cau_func_%d_2);\n",            /**/
772:                i, i, i);                                                               /**/
773:    }                                                                                   /**/
774:    else if (rule->distrib_type == DIST_POIS) {                                         /**/
775:       if  (expr_is_const (rule->distrib_param[0])) {                                   /**/
776:          fprintf (stream, "  poisval (\"trans_%d_\", %g, %g);\n", i,                   /**/
777:                eval_const_expr (rule->distrib_param[0], current_setting),              /**/
```

```
778:              eval_const_expr (rule->distrib_param[1], current_setting) );                                         /**/
779:       }                                                                                                            /**/
780:       else                                                                                                         /**/
781:          fprintf (stream,                                                                                          /**/
782:                "  poisfun (\"trans_%d_\", pois_func_%d_1, pois_func_%d_2);\n",                                     /**/
783:                i, i, i);                                                                                           /**/
784:    }                                                                                                               /**/
785:    else if (rule->distrib_type == DIST_PARETO) {                                                                   /**/
786:       if  (expr_is_const (rule->distrib_param[0])) {                                                               /**/
787:          fprintf (stream, "  parval (\"trans_%d_\", %g, %g);\n", i,                                                /**/
788:                eval_const_expr (rule->distrib_param[0], current_setting),                                          /**/
789:                eval_const_expr (rule->distrib_param[1], current_setting) );                                        /**/
790:       }                                                                                                            /**/
791:       else                                                                                                         /**/
792:          fprintf (stream,                                                                                          /**/
793:                "  parfun (\"trans_%d_\", par_func_%d_1, par_func_%d_2);\n",                                        /**/
794:                i, i, i);                                                                                           /**/
795:    }                                                                                                               /**/
796:    else if (rule->distrib_type == DIST_BINOM) {                                                                    /**/
797:       if  (expr_is_const (rule->distrib_param[0])) {                                                               /**/
798:          fprintf (stream, "  binoval (\"trans_%d_\", %g, %g, %g);\n", i,                                           /**/
799:                eval_const_expr (rule->distrib_param[0], current_setting),                                          /**/
800:                eval_const_expr (rule->distrib_param[1], current_setting),                                          /**/
801:                eval_const_expr (rule->distrib_param[2], current_setting));                                         /**/
802:       }                                                                                                            /**/
803:       else                                                                                                         /**/
804:          fprintf (stream,                                                                                          /**/
805:                "  binofun (\"trans_%d_\", bino_func_%d_1, bino_func_%d_2, bino_func_%d_3);\n",                     /**/
806:                i, i, i, i);                                                                                        /**/
807:    }                                                                                                               /**/
808:    else if (rule->distrib_type == DIST_HYPER) {                                                                    /**/
809:       if  (expr_is_const (rule->distrib_param[0])) {                                                               /**/
810:          fprintf (stream, "  hyperval (\"trans_%d_\", %g, %g, %g);\n", i,                                          /**/
811:                eval_const_expr (rule->distrib_param[0], current_setting),                                          /**/
812:                eval_const_expr (rule->distrib_param[1], current_setting),                                          /**/
813:                eval_const_expr (rule->distrib_param[2], current_setting));                                         /**/
814:       }                                                                                                            /**/
815:       else                                                                                                         /**/
816:          fprintf (stream,                                                                                          /**/
817:                "  hyperfun (\"trans_%d_\", hyper_func_%d_1, hyper_func_%d_2, hyper_func_%d_3);\n",                 /**/
818:                i, i, i, i);                                                                                        /**/
819:    }                                                                                                               /**/
820:    else if (rule->distrib_type == DIST_HYPO) {                                                                     /**/
821:       if  (expr_is_const (rule->distrib_param[0])) {                                                               /**/
822:          fprintf (stream, "  hypoval (\"trans_%d_\", %g, %g, %g, %g);\n", i,                                       /**/
823:                eval_const_expr (rule->distrib_param[0], current_setting),                                          /**/
824:                eval_const_expr (rule->distrib_param[1], current_setting),                                          /**/
825:                eval_const_expr (rule->distrib_param[2], current_setting),                                          /**/
826:                eval_const_expr (rule->distrib_param[3], current_setting));                                         /**/
827:       }                                                                                                            /**/
828:       else                                                                                                         /**/
829:          fprintf (stream,                                                                                          /**/
830:                "  hypofun (\"trans_%d_\", hypo_func_%d_1, hypo_func_%d_2, hypo_func_%d_3, hypo_func_%d_4);\n",     /**/
831:                i, i, i, i, i);                                                                                     /**/
832:    }                                                                                                               /**/
833:    /***************************************************************************************************************/
```

C.9 cspl.c - line 836 et sqq.

```
836:    switch (rule->policy)
[...]
843:    /* new policy provided by spnp simulation              - PW **/
844:    case POLICY_PRI:                                            /**/
845:      fprintf (stream, "  policy (\"trans_%d_\", PRI);\n", i);  /**/
846:      break;                                                    /**/
847:    /************************************************************/
```

C.10 cspl.c - line 1055 et sqq.

```
1055:static void
1056:print_result_functions (void)
1057:/* Print C functions that are necessary to compute the results. */
1058:{
1059:  int i;
1060:  int j;                  /* required for job-loop */               /* PW */
```

```
1061:  node_t *node;           /* required for ext. dist_func-creation */ /* PW */
1062:  bool_t job_dist_used=0; /* is dist_func-creation necessary? */     /* PW */
[...]
1093:  /* checking if job distribution output is desired           - PW **/
1094:  for (i = 0; i < get_node_count (); i++)                        /**/
1095:  {                                                              /**/
1096:    node = get_node (i);                                         /**/
1097:    if (node->print_distrib || node->print_avg_distrib)          /**/
1098:    {                                                            /**/
1099:       job_dist_used=1;                                          /**/
1100:       break;                                                    /**/
1101:    }                                                            /**/
1102:  }                                                              /**/
1103:  /* global variables not useful in simulation                    **/
1104:  if (job_dist_used==1 && cmdline.cspl_mode != CSPL_SIMULATION) { /**/
1105:     /* job distribution functions necessary and SPNP not in */   /**/
1106:     /* simulation mode, we can use the old method (global   */   /**/
1107:     /* variables but shorter)                               */   /**/
1108:  /***************************************************************/
[...]
1140:   /* if simulation is used, we have to provide expected()-calls            - PW **/
1141:   /* for each combination of node and its marking */                          /**/
1142:   }                                            /* end of non-simulated part */ /**/
1143:   else if (job_dist_used==1)                                                  /**/
1144:   {                                                          /* simulated */ /**/
1145:       /* printing all required dist_functs_ */                                /**/
1146:       fprintf (stream, "\n/* distribution handling for simulation */\n");     /**/
1147:       for (i = 0; i < get_node_count (); i++)                                 /**/
1148:       {                                                                       /**/
1149:          node = get_node (i);                                                 /**/
1150:          if (node->print_distrib || node->print_avg_distrib)                  /**/
1151:          {                                                                    /**/
1152:             for (j = 0;                                                       /**/
1153:                  j <= eval_int_const_expr (node->capacity, current_setting);   /**/
1154:                  j++)                                                         /**/
1155:             {                                                                 /**/
1156:                fprintf                                                        /**/
1157:                 (stream, "\n"                                                 /**/
1158:                  "double dist_func_sim_%s_%d_ (void)\n"                       /**/
1159:                  "{\n"                                                        /**/
1160:                  "  return (mark (\"%s\") == %d ? 1.0 : 0.0);\n"              /**/
1161:                  "}\n", node->name, j, node->name, j                          /**/
1162:                 );                                                            /**/
1163:             }                                                                 /**/
1164:          }                                                                    /**/
1165:       }                                                                       /**/
1166:       /* printing print_dist_ */                                              /**/
1167:       fprintf                                                                 /**/
1168:          ( stream,                                                            /**/
1169:            "\n"                                                               /**/
1170:            "void print_dist_ (char *node, int capacity)\n{\n"                 /**/
1171:            "  \n"                                                             /**/
1172:          );                                                                   /**/
1173:       for (i = 0; i < get_node_count (); i++)                                 /**/
1174:       {                                                                       /**/
1175:          node = get_node (i);                                                 /**/
1176:          if (node->print_distrib)                                             /**/
1177:          {                                                                    /**/
1178:             fprintf                                                           /**/
1179:                ( stream,                                                      /**/
1180:                  "  if ( strcmp (node, \"%s\") == 0 )\n"                      /**/
1181:                  "  {\n"                                                      /**/
1182:                  , node->name                                                 /**/
1183:                );                                                             /**/
1184:             for (j = 0;                                                       /**/
1185:                  j <= eval_int_const_expr (node->capacity, current_setting);   /**/
1186:                  j++)                                                         /**/
```

```
1187:             {                                                                       /**/
1188:                fprintf                                                              /**/
1189:                   ( stream,                                                         /**/
1190:                     "    pr_value (\"DIST_ %s %d\", "                                /**/
1191:                     "expected (dist_func_sim_%s_%d_));\n"                           /**/
1192:                     , node->name, j, node->name, j                                  /**/
1193:                   );                                                                /**/
1194:             }                                                                       /**/
1195:             fprintf                                                                 /**/
1196:                ( stream,                                                            /**/
1197:                  "  }\n\n"                                                          /**/
1198:                );                                                                   /**/
1199:          }                                                                          /**/
1200:       }                                                                             /**/
1201:       fprintf                                                                       /**/
1202:          ( stream,                                                                  /**/
1203:            "\n"                                                                     /**/
1204:            "}\n\n"                                                                  /**/
1205:          );                                                                         /**/
1206:       /* printing print_avg_dist_ */                                                /**/
1207:       fprintf                                                                       /**/
1208:          ( stream,                                                                  /**/
1209:            "\n"                                                                     /**/
1210:            "void print_avg_dist_ (char *node, int capacity, double time)\n{\n"  /**/
1211:            "  \n"                                                                   /**/
1212:          );                                                                         /**/
1213:       for (i = 0; i < get_node_count (); i++)                                       /**/
1214:       {                                                                             /**/
1215:          node = get_node (i);                                                       /**/
1216:          if (node->print_distrib)                                                   /**/
1217:          {                                                                          /**/
1218:             fprintf                                                                 /**/
1219:                ( stream,                                                            /**/
1220:                  "  if ( strcmp (node, \"%s\") == 0 )\n"                            /**/
1221:                  "  {\n"                                                            /**/
1222:                  , node->name                                                       /**/
1223:                );                                                                   /**/
1224:             for (j = 0;                                                             /**/
1225:                  j <= eval_int_const_expr (node->capacity, current_setting);        /**/
1226:                  j++)                                                               /**/
1227:             {                                                                       /**/
1228:                fprintf                                                              /**/
1229:                   ( stream,                                                         /**/
1230:                     "    pr_value (\"DIST_ %s %d\", "                                /**/
1231:                     "cum_expected (dist_func_sim_%s_%d_) / time);\n"                /**/
1232:                     , node->name, j, node->name, j                                  /**/
1233:                   );                                                                /**/
1234:             }                                                                       /**/
1235:             fprintf                                                                 /**/
1236:                ( stream,                                                            /**/
1237:                  "  }\n\n"                                                          /**/
1238:                );                                                                   /**/
1239:          }                                                                          /**/
1240:       }                                                                             /**/
1241:       fprintf                                                                       /**/
1242:          ( stream,                                                                  /**/
1243:            "\n"                                                                     /**/
1244:            "}\n\n"                                                                  /**/
1245:          );                                                                         /**/
1246:       fprintf (stream,"\n/* ending distribution handling for simulation */\n");/**/
1247:    }                                                                                /**/
1248:    /********************************************************************************/
```

C.11 main.c - line 86 et sqq.

```
86:static bool_t
87:gen_moslang (void)
88:/* Generate MOSLANG file(s), and optionally run MOSES. */
89:{
[...]
109:  /* distributions not allowed in MOSES                            - PW **/
110:  check_not_used ("MOSLANG", DIST_GEOM_USED,                           /**/
111:                 "geometric distribution not allowed in MOSES");       /**/
112:  check_not_used ("MOSLANG", DIST_NORM_USED,                           /**/
113:                 "normal distribution not allowed in MOSES");          /**/
114:  check_not_used ("MOSLANG", DIST_LOGN_USED,                           /**/
115:                 "lognormal distribution not allowed in MOSES");       /**/
116:  check_not_used ("MOSLANG", DIST_ERLANG_USED,                         /**/
117:                 "Erlang distribution not allowed in MOSES");          /**/
118:  check_not_used ("MOSLANG", DIST_GAMMA_USED,                          /**/
119:                 "gamma distribution not allowed in MOSES");           /**/
120:  check_not_used ("MOSLANG", DIST_BETA_USED,                           /**/
121:                 "beta distribution not allowed in MOSES");            /**/
122:  check_not_used ("MOSLANG", DIST_CAUCHY_USED,                         /**/
123:                 "Cauchy distribution not allowed in MOSES");          /**/
124:  check_not_used ("MOSLANG", DIST_BINOM_USED,                          /**/
125:                 "binomial distribution not allowed in MOSES");        /**/
126:  check_not_used ("MOSLANG", DIST_POIS_USED,                           /**/
127:                 "Poisson distribution not allowed in MOSES");         /**/
128:  check_not_used ("MOSLANG", DIST_PARETO_USED,                         /**/
129:                 "Pareto distribution not allowed in MOSES");          /**/
130:  check_not_used ("MOSLANG", DIST_HYPER_USED,                          /**/
131:                 "hyperexponential distribution not allowed in MOSES");  /**/
132:  check_not_used ("MOSLANG", DIST_HYPO_USED,                           /**/
133:                 "hypoexponential distribution not allowed in MOSES");   /**/
134:  /**********************************************************************/
```

C.12 main.c - line 150 et sqq.

```
150:static bool_t
151:gen_cspl (void)
152:/* Generate CSPL file(s), and optionally run SPNP. */
153:{
[...]
157   /* this is now possible                                                  - PW **/
158   /* check_not_used ("CSPL", DETERM_DIST_USED,              */               /**/
159   /*              "Deterministic firing distribution used");  */             /**/
160   /*********************************************************************************/
[...]
163:  /* distributions not allowed in normal analysis (but simulation)         - PW **/
164:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                     /**/
165:          check_not_used ("CSPL", DETERM_DIST_USED,                           /**/
166:                          "Deterministic firing distribution only allowed "   /**/
167:                          "if simulation is used.");                          /**/
168:  }                                                                           /**/
169:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                     /**/
170:          check_not_used ("CSPL", DIST_GEOM_USED,                             /**/
171:                          "Geometric firing distribution only allowed "       /**/
172:                          "if simulation is used.");                          /**/
173:  }                                                                           /**/
174:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                     /**/
175:          check_not_used ("CSPL", DIST_NORM_USED,                             /**/
176:                          "Normal firing distribution only allowed "          /**/
177:                          "if simulation is used.");                          /**/
178:  }                                                                           /**/
179:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                     /**/
```

```
180:          check_not_used ("CSPL", DIST_LOGN_USED,                                    /**/
181:                          "Lognormal firing distribution only allowed "         /**/
182:                          "if simulation is used.");                            /**/
183:  }                                                                            /**/
184:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                      /**/
185:          check_not_used ("CSPL", DIST_ERLANG_USED,                            /**/
186:                          "Erlang firing distribution only allowed "            /**/
187:                          "if simulation is used.");                            /**/
188:  }                                                                            /**/
189:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                      /**/
190:          check_not_used ("CSPL", DIST_GAMMA_USED,                             /**/
191:                          "Gamma firing distribution only allowed "             /**/
192:                          "if simulation is used.");                            /**/
193:  }                                                                            /**/
194:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                      /**/
195:          check_not_used ("CSPL", DIST_BETA_USED,                              /**/
196:                          "Beta firing distribution only allowed "              /**/
197:                          "if simulation is used.");                            /**/
198:  }                                                                            /**/
199:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                      /**/
200:          check_not_used ("CSPL", DIST_CAUCHY_USED,                            /**/
201:                          "Cauchy firing distribution only allowed "            /**/
202:                          "if simulation is used.");                            /**/
203:  }                                                                            /**/
204:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                      /**/
205:          check_not_used ("CSPL", DIST_BINOM_USED,                             /**/
206:                          "Binomial firing distribution only allowed "          /**/
207:                          "if simulation is used.");                            /**/
208:  }                                                                            /**/
209:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                      /**/
210:          check_not_used ("CSPL", DIST_POIS_USED,                              /**/
211:                          "Poisson firing distribution only allowed "           /**/
212:                          "if simulation is used.");                            /**/
213:  }                                                                            /**/
214:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                      /**/
215:          check_not_used ("CSPL", DIST_PARETO_USED,                            /**/
216:                          "Pareto firing distribution only allowed "            /**/
217:                          "if simulation is used.");                            /**/
218:  }                                                                            /**/
219:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                      /**/
220:          check_not_used ("CSPL", DIST_HYPER_USED,                             /**/
221:                          "Hyperexponential firing distribution only allowed "  /**/
222:                          "if simulation is used.");                            /**/
223:  }                                                                            /**/
224:  if (cmdline.cspl_mode == CSPL_NORMAL) {                                      /**/
225:          check_not_used ("CSPL", DIST_HYPO_USED,                              /**/
226:                          "Hypoexponential firing distribution only allowed "   /**/
227:                          "if simulation is used.");                            /**/
228:  }                                                                            /**/
229:  /*****************************************************************************/
```

C.13 main.c - line 245 et sqq.

```
245:static bool_t
246:gen_timenet (void)
247:/* Generate TimenNET file(s), and optionally run TimeNET. */
248:{
[...]
257:  /* distributions not allowed in TimeNET                          - PW **/
258:  check_not_used ("TimeNET", DIST_GEOM_USED,                           /**/
259:                "geometric distribution not allowed in TimeNET");      /**/
260:  check_not_used ("TimeNET", DIST_NORM_USED,                           /**/
261:                "normal distribution not allowed in TimeNET");         /**/
262:  check_not_used ("TimeNET", DIST_LOGN_USED,                           /**/
```

```
263:                "lognormal distribution not allowed in TimeNET");       /**/
264:  check_not_used ("TimeNET", DIST_ERLANG_USED,                           /**/
265:                "Erlang distribution not allowed in TimeNET");          /**/
266:  check_not_used ("TimeNET", DIST_GAMMA_USED,                            /**/
267:                "gamma distribution not allowed in TimeNET");           /**/
268:  check_not_used ("TimeNET", DIST_BETA_USED,                             /**/
269:                "beta distribution not allowed in TimeNET");            /**/
270:  check_not_used ("TimeNET", DIST_CAUCHY_USED,                           /**/
271:                "Cauchy distribution not allowed in TimeNET");          /**/
272:  check_not_used ("TimeNET", DIST_BINOM_USED,                            /**/
273:                "binomial distribution not allowed in TimeNET");        /**/
274:  check_not_used ("TimeNET", DIST_POIS_USED,                             /**/
275:                "Poisson distribution not allowed in TimeNET");         /**/
276:  check_not_used ("TimeNET", DIST_PARETO_USED,                           /**/
277:                "Pareto distribution not allowed in TimeNET");          /**/
278:  check_not_used ("TimeNET", DIST_HYPER_USED,                            /**/
279:                "hyperexponential distribution not allowed in TimeNET");/**/
280:  check_not_used ("TimeNET", DIST_HYPO_USED,                             /**/
281:                "hypoexponential distribution not allowed in TimeNET"); /**/
282:  /*************************************************************************/
```

C.14 model.c - line 797 et sqq.

```
797:void
798:check_model (void)
799:/* Check the MOSEL-2 description for consistency and check its properties. */
800:{
[...]
843:  /* Check rules. */
844:  for (i = 0; i < rule_count; i++)
845:  {
[...]
854:    /* spnp simulation distributions used           - PW **/
855:    if (rule->distrib_type == DIST_GEOM)                /**/
856:                set_property ( DIST_GEOM_USED);         /**/
857:    if (rule->distrib_type == DIST_NORM)                /**/
858:                set_property ( DIST_NORM_USED);         /**/
859:    if (rule->distrib_type == DIST_LOGN)                /**/
860:                set_property ( DIST_LOGN_USED);         /**/
861:    if (rule->distrib_type == DIST_ERLANG)              /**/
862:                set_property ( DIST_ERLANG_USED);       /**/
863:    if (rule->distrib_type == DIST_GAMMA)               /**/
864:                set_property ( DIST_GAMMA_USED);        /**/
865:    if (rule->distrib_type == DIST_BETA)                /**/
866:                set_property ( DIST_BETA_USED);         /**/
867:    if (rule->distrib_type == DIST_CAUCHY)              /**/
868:                set_property ( DIST_BINOM_USED);        /**/
869:    if (rule->distrib_type == DIST_POIS)                /**/
870:                set_property ( DIST_POIS_USED);         /**/
871:    if (rule->distrib_type == DIST_PARETO)              /**/
872:                set_property ( DIST_PARETO_USED);       /**/
873:    if (rule->distrib_type == DIST_HYPER)               /**/
874:                set_property ( DIST_HYPER_USED);        /**/
875:    if (rule->distrib_type == DIST_HYPO)                /**/
876:                set_property ( DIST_HYPO_USED);         /**/
877:    /****************************************************/
```

C.15 model.c - line 889 et sqq.

```
889:static void
890:dump_rule (int rule_count)
```

```
891:/* Dump the rule RULES[RULE_COUNT]. */
892:{
[...]
950:   /* Print firing distribution. */
951:  switch (rule->distrib_type)
952:  {
[...]
977:  /* further distributions (spnp simulation)                - PW **/
978:  case DIST_GEOM:                                               /**/
979:    printf (" GEOM ( ");                                        /**/
980:    print_expr (stdout, rule->distrib_param[0], 0);             /**/
981:    printf (", ");                                              /**/
982:    print_expr (stdout, rule->distrib_param[1], 0);             /**/
983:    printf (")");                                               /**/
984:    break;                                                      /**/
985:  case DIST_NORM:                                               /**/
986:    printf (" NORM ( ");                                        /**/
987:    print_expr (stdout, rule->distrib_param[0], 0);             /**/
988:    printf (", ");                                              /**/
989:    print_expr (stdout, rule->distrib_param[1], 0);             /**/
990:    printf (")");                                               /**/
991:    break;                                                      /**/
992:  case DIST_LOGN:                                               /**/
993:    printf (" LOGN ( ");                                        /**/
994:    print_expr (stdout, rule->distrib_param[0], 0);             /**/
995:    printf (", ");                                              /**/
996:    print_expr (stdout, rule->distrib_param[1], 0);             /**/
997:    printf (")");                                               /**/
998:    break;                                                      /**/
999:  case DIST_ERLANG:                                             /**/
1000:   printf (" ERLANG ( ");                                      /**/
1001:   print_expr (stdout, rule->distrib_param[0], 0);             /**/
1002:   printf (", ");                                              /**/
1003:   print_expr (stdout, rule->distrib_param[1], 0);             /**/
1004:   printf (")");                                               /**/
1005:   break;                                                      /**/
1006: case DIST_GAMMA:                                              /**/
1007:   printf (" GAMMA ( ");                                       /**/
1008:   print_expr (stdout, rule->distrib_param[0], 0);             /**/
1009:   printf (", ");                                              /**/
1010:   print_expr (stdout, rule->distrib_param[1], 0);             /**/
1011:   printf (")");                                               /**/
1012:   break;                                                      /**/
1013: case DIST_BETA:                                               /**/
1014:   printf (" BETA ( ");                                        /**/
1015:   print_expr (stdout, rule->distrib_param[0], 0);             /**/
1016:   printf (", ");                                              /**/
1017:   print_expr (stdout, rule->distrib_param[1], 0);             /**/
1018:   printf (")");                                               /**/
1019:   break;                                                      /**/
1020: case DIST_CAUCHY:                                             /**/
1021:   printf (" CAUCHY ( ");                                      /**/
1022:   print_expr (stdout, rule->distrib_param[0], 0);             /**/
1023:   printf (", ");                                              /**/
1024:   print_expr (stdout, rule->distrib_param[1], 0);             /**/
1025:   printf (")");                                               /**/
1026:   break;                                                      /**/
1027: case DIST_POIS:                                               /**/
1028:   printf (" POIS ( ");                                        /**/
1029:   print_expr (stdout, rule->distrib_param[0], 0);             /**/
1030:   printf (", ");                                              /**/
1031:   print_expr (stdout, rule->distrib_param[1], 0);             /**/
1032:   printf (")");                                               /**/
1033:   break;                                                      /**/
1034: case DIST_PARETO:                                             /**/
1035:   printf (" PARETO ( ");                                      /**/
1036:   print_expr (stdout, rule->distrib_param[0], 0);             /**/
```

```
1037:   printf (", ");                                             /**/
1038:   print_expr (stdout, rule->distrib_param[1], 0);            /**/
1039:   printf (")");                                              /**/
1040:   break;                                                     /**/
1041: case DIST_BINOM:                                             /**/
1042:   printf (" BINOM ( ");                                      /**/
1043:   print_expr (stdout, rule->distrib_param[0], 0);            /**/
1044:   printf (", ");                                             /**/
1045:   print_expr (stdout, rule->distrib_param[1], 0);            /**/
1046:   printf (", ");                                             /**/
1047:   print_expr (stdout, rule->distrib_param[2], 0);            /**/
1048:   printf (")");                                              /**/
1049:   break;                                                     /**/
1050: case DIST_HYPER:                                             /**/
1051:   printf (" HYPER ( ");                                      /**/
1052:   print_expr (stdout, rule->distrib_param[0], 0);            /**/
1053:   printf (", ");                                             /**/
1054:   print_expr (stdout, rule->distrib_param[1], 0);            /**/
1055:   printf (", ");                                             /**/
1056:   print_expr (stdout, rule->distrib_param[2], 0);            /**/
1057:   printf (")");                                              /**/
1058:   break;                                                     /**/
1059: case DIST_HYPO:                                              /**/
1060:   printf (" HYPO ( ");                                       /**/
1061:   print_expr (stdout, rule->distrib_param[0], 0);            /**/
1062:   printf (", ");                                             /**/
1063:   print_expr (stdout, rule->distrib_param[1], 0);            /**/
1064:   printf (", ");                                             /**/
1065:   print_expr (stdout, rule->distrib_param[2], 0);            /**/
1066:   printf (", ");                                             /**/
1067:   print_expr (stdout, rule->distrib_param[3], 0);            /**/
1068:   printf (")");                                              /**/
1069:   break;                                                     /**/
1070: default :                                                    /**/
1071:   error_msg (FATAL, "Internal: Unknown distribution type!");  /**/
1072: /*************************************************************/
```

C.16 model.c - line 1074 et sqq.

```
1074:  switch (rule->policy)
1075:  {
[...]
1079:  /* policy provided by SPNP simulation  - PW **/
1080:  case POLICY_PRI: printf (" PRI"); break;   /**/
1081:  /********************************************/
```

C.17 model.h - line 9 et sqq.

```
 9:typedef enum
10:{
[...]
24:  /* other distrib. used  - PW **/
25:  /* prov. by SPNP sim. */    /**/
26:  DIST_GEOM_USED,            /**/
27:  DIST_NORM_USED,            /**/
28:  DIST_LOGN_USED,            /**/
29:  DIST_ERLANG_USED,          /**/
30:  DIST_GAMMA_USED,           /**/
31:  DIST_BETA_USED,            /**/
32:  DIST_CAUCHY_USED,          /**/
33:  DIST_BINOM_USED,           /**/
```

```
34:  DIST_POIS_USED,             /**/
35:  DIST_PARETO_USED,           /**/
36:  DIST_HYPER_USED,            /**/
37:  DIST_HYPO_USED,             /**/
38:  /******************************/
[...]
46:} property_t;
```

C.18 model.h - line 114 et sqq.

```
114:typedef enum
115:{ DIST_IMMED, /* Fires immediately. */
[...]
118:  /* new distributions - PW **/
119:  /* prov. by SPNP sim. */ /**/
120:  DIST_GEOM,               /**/
121:  DIST_NORM,               /**/
122:  DIST_LOGN,               /**/
123:  DIST_ERLANG,             /**/
124:  DIST_GAMMA,              /**/
125:  DIST_BETA,               /**/
126:  DIST_CAUCHY,             /**/
127:  DIST_BINOM,              /**/
128:  DIST_POIS,               /**/
129:  DIST_PARETO,             /**/
130:  DIST_HYPER,              /**/
131:  DIST_HYPO,               /**/
132:  /***************************/
133:} distrib_type_t;
```

C.19 model.h - line 137 et sqq.

```
137:typedef enum
138:{ POLICY_DEFAULT, /* use the default policy */
139:  POLICY_PRD, /* use a new firing time */
140:  POLICY_PRS, /* leave the remaining firing time */
141:  /* policy provided by SPNP simulation   - PW **/
142:  POLICY_PRI, /* use identical firing time */ /**/
143:  /*********************************************/
144:} policy_t;
```

C.20 model.h - line 146 et sqq.

```
146:enum { DIST_PARAM_MAX = 4 }; /* maximum number of distribution parameters. */
147:/* Incremented by 1 to 4 because of SPNP's hyperexp. distribution    - PW  **/
```

C.21 parser.y - line 490 et sqq.

```
490:static void
491: set_distrib (distrib_type_t type,
492:              expr_t *param0, expr_t *param1, expr_t *param2, expr_t *param3)
493:/* added *param3 for hyperexp. distribution - PW */
494:/* Set firing distribution of current rule. */
495:{
[...]
```

```
501:  /* Do distribution-dependent checks. */
502:  switch (type)
503:  {
[...]
551:  /* new distributions for SPNP simulation                                  - PW **/
552:  case DIST_GEOM:                                                                /**/
553:    if ( ! expr_is_non_negative(param0) )                                        /**/
554:         error_msg (ERROR,                                                       /**/
555:         "First parameter of geometric distribution must be in [0..1] ");        /**/
556:    for (i = 0; i < setting_count; i++)                                          /**/
557:    {                                                                            /**/
558:      if (eval_const_expr (param0, i) > 1 )                                      /**/
559:      {                                                                          /**/
560:        error_msg (ERROR,                                                        /**/
561:        "First parameter of geometric distribution must be in [0..1] ");         /**/
562:        break;                                                                   /**/
563:      }                                                                          /**/
564:    }                                                                            /**/
565:    if ( ! expr_is_positive(param1) )                                            /**/
566:         error_msg (ERROR,                                                       /**/
567:         "Second parameter of geometric distribution"                            /**/
568:                                                  " must be positive ");         /**/
569:    break;                                                                       /**/
570:  case DIST_NORM:                                                                /**/
571:    if ( ! expr_is_positive(param0) )                                            /**/
572:         error_msg (ERROR,                                                       /**/
573:         "First parameter of normal distribution"                                /**/
574:                                                  " must be positive ");         /**/
575:    if ( ! expr_is_positive(param1) )                                            /**/
576:         error_msg (ERROR,                                                       /**/
577:         "Second parameter of normal distribution"                               /**/
578:                                                  " must be positive ");         /**/
579:    break;                                                                       /**/
580:  case DIST_LOGN:                                                                /**/
581:    if ( ! expr_is_positive(param0) )                                            /**/
582:         error_msg (ERROR,                                                       /**/
583:         "First parameter of lognormal distribution"                             /**/
584:                                                  " must be positive ");         /**/
585:    if ( ! expr_is_positive(param1) )                                            /**/
586:         error_msg (ERROR,                                                       /**/
587:         "Second parameter of lognormal distribution"                            /**/
588:                                                  " must be positive ");         /**/
589:    break;                                                                       /**/
590:  case DIST_ERLANG:                                                              /**/
591:    if (! expr_is_positive (param0))                                             /**/
592:         error_msg (ERROR,                                                       /**/
593:         "First parameter of Erlang distribution must be positive");             /**/
594:    if (! expr_is_positive (param1))                                             /**/
595:         error_msg (ERROR,                                                       /**/
596:         "Second parameter of Erlang distribution must be positive");            /**/
597:    break;                                                                       /**/
598:  case DIST_GAMMA:                                                               /**/
599:    if ( ! expr_is_positive(param0) )                                            /**/
600:         error_msg (ERROR,                                                       /**/
601:         "First parameter of gamma distribution"                                 /**/
602:                                                  " must be positive ");         /**/
603:    if ( ! expr_is_positive(param1) )                                            /**/
604:         error_msg (ERROR,                                                       /**/
605:         "Second parameter of gamma distribution"                                /**/
606:                                                  " must be positive ");         /**/
607:    break;                                                                       /**/
608:  case DIST_BETA:                                                                /**/
609:    if ( ! expr_is_positive(param0) )                                            /**/
610:         error_msg (ERROR,                                                       /**/
611:         "First parameter of beta distribution"                                  /**/
612:                                                  " must be positive ");         /**/
613:    if ( ! expr_is_positive(param1) )                                            /**/
```

```
614:         error_msg (ERROR,                                                    /**/
615:         "Second parameter of beta distribution"                              /**/
616:                                                 " must be positive ");       /**/
617:     break;                                                                   /**/
618:   case DIST_CAUCHY:                                                          /**/
619:     if (! expr_is_positive (param1))                                         /**/
620:         error_msg (ERROR,                                                    /**/
621:         "Second parameter of Cauchy distribution must be positive");         /**/
622:     break;                                                                   /**/
623:   case DIST_POIS:                                                            /**/
624:     if (! expr_is_positive (param0))                                         /**/
625:         error_msg (ERROR,                                                    /**/
626:         "First parameter of Poisson distribution must be positive");         /**/
627:     if ( ! expr_is_positive(param1) )                                        /**/
628:         error_msg (ERROR,                                                    /**/
629:         "Second parameter of Poisson distribution"                           /**/
630:                                                       " must be positive "); /**/
631:     break;                                                                   /**/
632:   case DIST_PARETO:                                                          /**/
633:     if (! expr_is_positive (param0))                                         /**/
634:         error_msg (ERROR,                                                    /**/
635:         "First parameter of Pareto distribution must be positive");          /**/
636:     if (! expr_is_positive (param1))                                         /**/
637:         error_msg (ERROR,                                                    /**/
638:         "Second parameter of Pareto distribution must be positive");         /**/
639:     break;                                                                   /**/
640:   case DIST_BINOM:                                                           /**/
641:     if ( ! expr_is_positive(param0) || ! expr_is_cardinal(param0) )          /**/
642:         error_msg (ERROR,                                                    /**/
643:         "First parameter of binomial distribution"                           /**/
644:                                         " must be a positive integer ");     /**/
645:     if ( ! expr_is_non_negative(param1) )                                    /**/
646:         error_msg (ERROR,                                                    /**/
647:         "Second parameter of binomial distribution "                         /**/
648:                                                       "must be in [0..1] "); /**/
649:     for (i = 0; i < setting_count; i++)                                      /**/
650:     {                                                                        /**/
651:       if (eval_const_expr (param1, i) > 1 )                                  /**/
652:       {                                                                      /**/
653:         error_msg (ERROR,                                                    /**/
654:         "Second parameter of binomial distribution "                         /**/
655:                                                       "must be in [0..1] "); /**/
656:         break;                                                               /**/
657:       }                                                                      /**/
658:     }                                                                        /**/
659:     if ( ! expr_is_positive(param2) )                                        /**/
660:         error_msg (ERROR,                                                    /**/
661:         "Third parameter of binomial distribution"                           /**/
662:                                                       " must be positive "); /**/
663:     break;                                                                   /**/
664:   case DIST_HYPER:                                                           /**/
665:     if (! expr_is_positive (param0))                                         /**/
666:         error_msg (ERROR,                                                    /**/
667:         "First parameter of hyperexponential distribution "                  /**/
668:                                                         "must be positive"); /**/
669:     if (! expr_is_positive (param1))                                         /**/
670:         error_msg (ERROR,                                                    /**/
671:         "Second parameter of hyperexponential distribution "                 /**/
672:                                                         "must be positive"); /**/
673:     if ( ! expr_is_non_negative(param2) )                                    /**/
674:         error_msg (ERROR,                                                    /**/
675:         "Third parameter of hyperexponential distribution "                 /**/
676:                                                       "must be in [0..1] "); /**/
677:     for (i = 0; i < setting_count; i++)                                      /**/
678:     {                                                                        /**/
679:       if (eval_const_expr (param2, i) > 1 )                                  /**/
680:       {                                                                      /**/
```

```
681:        error_msg (ERROR,                                                /**/
682:        "Third parameter of hyperexponential distribution "             /**/
683:                                                "must be in [0..1] "); /**/
684:        break;                                                         /**/
685:      }                                                                /**/
686:    }                                                                  /**/
687:    break;                                                             /**/
688:  case DIST_HYPO:                                                      /**/
689:    if (! expr_is_positive (param0))                                   /**/
690:         error_msg (ERROR,                                             /**/
691:         "First parameter of hyperexponential distribution "           /**/
692:                                                 "must be positive");  /**/
693:    if (! expr_is_positive (param1))                                   /**/
694:         error_msg (ERROR,                                             /**/
695:         "Second parameter of hyperexponential distribution "          /**/
696:                                                 "must be positive");  /**/
697:    if (! expr_is_positive (param2))                                   /**/
698:         error_msg (ERROR,                                             /**/
699:         "Third parameter of hyperexponential distribution "           /**/
700:                                                 "must be positive");  /**/
701:    for (i = 0; i < setting_count; i++)                                /**/
702:    {                                                                  /**/
703:      if (eval_const_expr(param3, i) != 2 && eval_const_expr(param3, i) != 3) /**/
704:      {                                                                /**/
705:        error_msg (ERROR, "Forth parameter of hyperexponential distribution " /**/
706:                                                 "must be 2 or 3");    /**/
707:        break;                                                         /**/
708:      }                                                                /**/
709:    }                                                                  /**/
710:    break;                                                             /**/
711:  default :                                                            /**/
712:    error_msg (FATAL, "Internal: Unknown distribution type!");         /**/
713:  /***************************************************************************/
[...]
720:  /* new Parameter needed for SPNP's hyperexp. distrib. - PW **/
721:  rule->distrib_param[3] = param3;                    /**/
722:  /*********************************************************/
723:}
```

C.22 parser.y - line 1287 et sqq.

```
1287:/* Reserved keywords */
[...]
1292:/* further tokens for SPNP simulation               - PW **/
1293:%token GEOM NORM LOGN ERLANG GAMMA BETA CAUCHY          /**/
1294:%token BINOM POIS PARETO HYPER HYPO PRI                 /**/
1295:/*********************************************************/
```

C.23 parser.y - line 1468 et sqq.

```
1468:rule_element: FROM
[...]
1480:          | RATE state_expr
1481:            { set_distrib (DIST_EXP, $2.expr, NULL, NULL, NULL); }                /* added 4th parameter 'NULL' - PW */
1482:          | AFTER const_expr
1483:            { set_distrib (DIST_DETERM, $2.expr, NULL, NULL, NULL); }             /* added 4th parameter 'NULL' - PW */
1484:          | AFTER const_expr ".." const_expr
1485:            { set_distrib (DIST_UNIFORM, $2.expr, $4.expr, NULL, NULL); }         /* added 4th parameter 'NULL' - PW */
1486:          | AFTER const_expr ".." const_expr STEP const_expr
1487:            { set_distrib (DIST_DISCRETE_UNIFORM, $2.expr,$4.expr,$6.expr, NULL); } /* added 4th parameter 'NULL' - PW */


1488:            /* new distributions and PRI for SPNP Simulation                    - PW **/
1489:            | GEOM '(' const_expr ',' const_expr ')'                              /**/
```

```
1490:              { set_distrib (DIST_GEOM, $3.expr, $5.expr, NULL, NULL); }          /**/
1491:            | NORM '(' const_expr ',' const_expr ')'                              /**/
1492:              { set_distrib (DIST_NORM, $3.expr, $5.expr, NULL, NULL); }          /**/
1493:            | LOGN '(' const_expr ',' const_expr ')'                              /**/
1494:              { set_distrib (DIST_LOGN, $3.expr, $5.expr, NULL, NULL); }          /**/
1495:            | ERLANG '(' const_expr ',' const_expr ')'                            /**/
1496:              { set_distrib (DIST_ERLANG, $3.expr, $5.expr, NULL, NULL); }        /**/
1497:            | GAMMA '(' const_expr ',' const_expr ')'                             /**/
1498:              { set_distrib (DIST_GAMMA, $3.expr, $5.expr, NULL, NULL); }         /**/
1499:            | BETA '(' const_expr ',' const_expr ')'                              /**/
1500:              { set_distrib (DIST_BETA, $3.expr, $5.expr, NULL, NULL); }          /**/
1501:            | CAUCHY '(' const_expr ',' const_expr ')'                            /**/
1502:              { set_distrib (DIST_CAUCHY, $3.expr, $5.expr, NULL, NULL); }        /**/
1503:            | POIS '(' const_expr ',' const_expr ')'                              /**/
1504:              { set_distrib (DIST_POIS, $3.expr, $5.expr, NULL, NULL); }          /**/
1505:            | PARETO '(' const_expr ',' const_expr ')'                            /**/
1506:              { set_distrib (DIST_PARETO, $3.expr, $5.expr, NULL, NULL); }        /**/
1507:            | BINOM '(' const_expr ',' const_expr ',' const_expr ')'              /**/
1508:              { set_distrib (DIST_BINOM, $3.expr, $5.expr, $7.expr, NULL); }      /**/
1509:            | HYPER '(' const_expr ',' const_expr ',' const_expr ')'              /**/
1510:              { set_distrib (DIST_HYPER, $3.expr, $5.expr, $7.expr, NULL); }      /**/
1511:            | HYPO '(' const_expr ',' const_expr ',' const_expr ',' const_expr ')' /**/
1512:              { set_distrib (DIST_HYPO, $3.expr, $5.expr, $7.expr, $9.expr); }    /**/
1513:            | PRI                                                                 /**/
1514:              { set_policy (POLICY_PRI); }                                        /**/
1515:            /*************************************************************************/
```

C.24 result.c - line 62 et sqq.

```
62:/* variables for simulation output                                      - PW **/
63:static float simulation_length = -1;  /* -1 means 'no simulation used' */ /**/
64:/****************************************************************************/
```

C.25 result.c - line 424 et sqq.

```
424:void
425:write_results (char *file)
426:/* Write the results into FILE. */
427:{
[...]
443:  /* Write header */
442:  /* own header for SPNP simulation                                      - PW **/
443:  if (simulation_length != -1) {                                             /**/
444:   /* SPNP simulation is used */                                             /**/
445:     fprintf (stream, "Simulation of \"%s\" by %s (simulation length: %.0f)\n",  /**/
446:           cmdline.input_file, tool_name (cmdline.tool), simulation_length);     /**/
447:     if (analysis.transient)                                                  /**/
448:     {                                                                        /**/
449:      if (analysis.start_time == analysis.end_time)                           /**/
450:       fprintf (stream, "(Evaluated transient for time %g)\n",                /**/
451:                analysis.start_time);                                         /**/
452:      else                                                                    /**/
453:       fprintf (stream, "(Evaluated transient for times %g..%g, step width %g)\n",/**/
454:                analysis.start_time, analysis.end_time, analysis.step_width);     /**/
455:     }                                                                        /**/
456:  }                                                                           /**/
457:  else { /* no SPNP simulation used */                                        /**/
458:  /******************************************************************************/
[...]
463:  } /* end bracket for 'no SPNP simulation used' - PW */
```

C.26 result.c - line 1176 et sqq.

```
1176:static void
1177:parse_spnp_results (char *file, int setting)
1178:/* Read results for SETTING from FILE in SPNP ".out" format. */
1179:{
[...]
1189:  /* Read lines from FILE until end is reached. */
1190:  while (! feof (stream))
1191:  {
1192:    line = read_line (stream);
1193:    word = read_word (&line);
1194:    if (strcmp (word, "TIME") == 0)
1195:    {
[...]
1204:      else if (is_float_number (word)) { /* "{" required  - PW */
1205:        shot = time_to_shot (atof (word));
1206:      /* if simulation is used, print results, even if shot was -1                - PW **/
1207:        if ( cmdline.cspl_mode == CSPL_SIMULATION )  {                /*simulated */ /**/
1208:           shot = 0;                                 /* valid because of simulation */ /**/
1209:           simulation_length = atof (word);       /* in spnp-output after 'TIME :' */ /**/
1210:        }                                                                            /**/
1211:      }                                                                              /**/
1212:      /*********************************************************************************/
```

C.27 scanner.l - line 62 et sqq.

```
108: /* tokens for ESPN Distributions - PW **/
109: /* and PRI-policy (SPNP simulation)   **/
110:<NORMAL>GEOM           return GEOM;  /**/
111:<NORMAL>NORM           return NORM;  /**/
112:<NORMAL>LOGN           return LOGN;  /**/
113:<NORMAL>ERLANG         return ERLANG;/**/
114:<NORMAL>GAMMA          return GAMMA; /**/
115:<NORMAL>BETA           return BETA;  /**/
116:<NORMAL>CAUCHY         return CAUCHY;/**/
117:<NORMAL>BINOM          return BINOM; /**/
118:<NORMAL>POIS           return POIS;  /**/
119:<NORMAL>PARETO         return PARETO;/**/
120:<NORMAL>HYPER          return HYPER; /**/
121:<NORMAL>HYPO           return HYPO;  /**/
122:<NORMAL>PRI            return PRI;   /**/
123: /**************************************/
```

C.28 timenet.c - line 62 et sqq.

```
259:static void
260:print_transition (char *name, distrib_type_t distrib, double param,
261:                  bool_t param_is_const, policy_t policy, int priority)
[...]
266:{
[...]
279:  switch (distrib)
280:  {
[...]
286:  default :                                                              /* PW */
287:    error_msg (FATAL, "Internal: Unknown distribution type in TimeNET.");   /* PW */
288:  }
289:  switch (policy)
290:  {
```

```
291:  /* policy PRI (SPNP simulation)           - PW **/
292:  case POLICY_PRI:                               /**/
293:    error_msg (WARNING, "Policy PRI unknown in" /**/
294:      "TimeNET. Using default (PRD) instead."); /**/
295:  /**********************************************/
296:  case POLICY_DEFAULT:
297:  case POLICY_PRD:
298:    fprintf (stream, "RE "); break;
```

Appendix D

Full Example Code and Output

D.1 trivialPN.c (using SPNP only)

```
#include "user.h"


void options() {
        iopt(IOP_SIMULATION, VAL_YES); /*activate SPNP simulation*/
        iopt(IOP_SIM_RUNS, 1000);      /*number of simulation runs*/
        fopt(FOP_SIM_LENGTH, 1000);    /*time to run for each iteration*/
}

void net() {
        place("place1");         /*defining first place*/
        place("place2");         /*defining second place*/
        init("place1",2);        /*initially mark place1 with two tokens*/
        init("place2",1);        /*initially mark place2 with one token*/
        rateval("trans1",0.3);   /*defining first transition with exponential
                                   distribution and firing rate 0.3*/
        detval("trans2",3);      /*defining second transition with
                                   deterministic firing time 3*/
        iarc("trans1","place1"); /*link first place with first transition*/
        oarc("trans1","place2"); /*link first transition with second place*/
        iarc("trans2","place2"); /*link first place with second transition*/
        oarc("trans2","place1"); /*link second transition with second place*/
}

int assert() {
        if (mark("place1") + mark("place2") != 2)
            return(RES_ERROR);
        return(RES_NOERR);
}

void ac_init() {
}

void ac_reach() {
}


/* declaring own C functions */

double tokens1() {
        return(mark("place1"));
}
```

```
double tokens2() {
        return(mark("place2"));
}

double busy1() {
        return((mark("place1")!=0) ? 1 : 0);
}

double empty2() {
        return((mark("place2")==0) ? 1 : 0);
}


/* ac_final */

void ac_final() {
        double busy2;
        double tokens1_2;
        double t1, t2;
        solve(INFINITY);
        pr_expected("Average token number in 'place1':", tokens1);
        pr_expected("Average token number in 'place2':", tokens2);
        tokens1_2 = expected(tokens1) + expected(tokens2);
        pr_value("Average token number in system:", tokens1_2);
        pr_expected("Utilization of 'place1':", busy1);
        busy2 = 1 - expected(empty2);
        pr_value("Utilization of 'place2':", busy2);
}
```

D.2 trivialPN.out (using SPNP only)

```
==========================================================================

                         TIME :   1.00E+03

==========================================================================

Simulation Analysis Result
Number of RUNS = 1000
Number of Events per run:
5.1721e+02[5.1588e+02,5.1855e+02]@95%
==========================================================================
AVERAGE:
 PLACE            Pr[nonempty]                   Av[tokens]
 place1           8.6168e-01[8.6015e-01,8.6321e-01]@95% 1.6398e+00[1.6338e+00,1.6457e+00]@95%
 place2           7.7497e-01[7.7297e-01,7.7697e-01]@95% 1.3602e+00[1.3543e+00,1.3662e+00]@95%

 TRANSITION       Pr[enabled]                    Av[throughput]
 trans1           8.6168e-01[8.6015e-01,8.6321e-01]@95% 2.5828e-01[2.5761e-01,2.5895e-01]@95%
 trans2           7.7497e-01[7.7297e-01,7.7697e-01]@95% 2.5793e-01[2.5726e-01,2.5860e-01]@95%
==========================================================================
EXPECTED: Average token number in 'place1':(1000)       = 1.6480e+00[1.5881e+00,1.7079e+00]  @95%
EXPECTED: Average token number in 'place2':(1000)       = 1.3520e+00[1.2921e+00,1.4119e+00]  @95%

VALUE: Average token number in system: = 3
EXPECTED: Utilization of 'place1':(1000)        = 8.6600e-01[8.4488e-01,8.8712e-01]  @95%

VALUE: Utilization of 'place2': = 0.782
```

D.3 trivialPN.mos

```
/* Constant and Parameter Part */
CONST rate := 4;    /* rate of first transition/rule */
CONST delay := 0.5; /* delay of second transition/rule */
CONST tokens := 3;  /* number of tokens/jobs in closed system */
CONST init1 := 2;   /* initial number of tokens in first place/node */
CONST init2 := tokens-init1; /* initial number of tokens in 2nd node */

/* State Part */
NODE place1[tokens] := init1;    /* introduce 'place1' with max. 'tokens' tokens
                                    and 'init1' initial tokens */
NODE place2[tokens] := init2;    /* introduce 'place2' with max. 'tokens' tokens
                                    and 'init2' initial tokens */
ASSERT place1 + place2 = tokens; /* the number of tokens in the system always
                                    has to be 'tokens' */

/* Rule Part */
FROM place1 TO place2 RATE rate;   /* exp. distr. transition from 'place1'
                                      to 'place2' with parameter 'rate' */
FROM place2 TO place1 AFTER delay; /* determ. distr. transit. f. 'place2'
                                      to 'place1' with parameter 'delay' */

/* Result Part */
PRINT avg_no_of_tokens_in_place1 := MEAN(place1);
PRINT avg_no_of_tokens_in_place2 := MEAN(place2);
PRINT usage_of_place1 := PROB(place1 != 0);
PRINT usage_of_place2 := PROB(place2 != 0);
PRINT DIST place1;

/* Picture Part */
PICTURE "token distribution of place1"
 CURVE DIST place1
```

D.4 trivialPN.c

```
* Generated by MOSEL-2 from "trivialPN.mos" */

#include <stdio.h>
#include <math.h>
#include "user.h"

/* Constants */
#define rate_ 4
#define delay_ 0.5
#define tokens_ 3
#define init1_ 2
#define init2_ 1

void options (void)
{
  iopt (IOP_SIMULATION, VAL_YES);
  fopt (FOP_SIM_LENGTH, 100);
  iopt (IOP_SIM_RUNS, 1000);
  iopt (IOP_SIM_STD_REPORT, VAL_NO);
}

void net (void)
{
  /* Places */
  place ("place1");
  init ("place1", 2);
  place ("place2");
```

```
  init ("place2", 1);

  /* MOSEL-2 Rule 1 */

  rateval ("trans_0_", 4);
  iarc ("trans_0_", "place1");
  mharc ("trans_0_", "place2", 3),
  oarc ("trans_0_", "place2");

  /* MOSEL-2 Rule 2 */

  detval ("trans_1_", 0.5);
  iarc ("trans_1_", "place2");
  mharc ("trans_1_", "place1", 3),
  oarc ("trans_1_", "place1");
}

int assert (void)
{
  if (! (mark ("place1") + mark ("place2") == tokens_))
    return RES_ERROR;
  return RES_NOERR;
}

void ac_init (void)
{
  pr_net_info ();
}

void ac_reach (void)
{
  pr_rg_info ();
}

double reward_func_0_ (void)
{
  return mark ("place1");
}

double reward_func_1_ (void)
{
  return mark ("place2");
}

double reward_func_2_ (void)
{
  return (mark ("place1") != 0 ? 1.0 : 0.0);
}

double reward_func_3_ (void)
{
  return (mark ("place2") != 0 ? 1.0 : 0.0);
}

/* distribution handling for simulation */

double dist_func_sim_place1_0_ (void)
{
  return (mark ("place1") == 0 ? 1.0 : 0.0);
}

double dist_func_sim_place1_1_ (void)
{
  return (mark ("place1") == 1 ? 1.0 : 0.0);
}

double dist_func_sim_place1_2_ (void)
```

```
{
  return (mark ("place1") == 2 ? 1.0 : 0.0);
}

double dist_func_sim_place1_3_ (void)
{
  return (mark ("place1") == 3 ? 1.0 : 0.0);
}

void print_dist_ (char *node, int capacity)
{

  if ( strcmp (node, "place1") == 0 )
  {
    pr_value ("DIST_ place1 0", expected (dist_func_sim_place1_0_));
    pr_value ("DIST_ place1 1", expected (dist_func_sim_place1_1_));
    pr_value ("DIST_ place1 2", expected (dist_func_sim_place1_2_));
    pr_value ("DIST_ place1 3", expected (dist_func_sim_place1_3_));
  }
}

void print_avg_dist_ (char *node, int capacity, double time)
{

  if ( strcmp (node, "place1") == 0 )
  {
    pr_value ("DIST_ place1 0", cum_expected (dist_func_sim_place1_0_) / time);
    pr_value ("DIST_ place1 1", cum_expected (dist_func_sim_place1_1_) / time);
    pr_value ("DIST_ place1 2", cum_expected (dist_func_sim_place1_2_) / time);
    pr_value ("DIST_ place1 3", cum_expected (dist_func_sim_place1_3_) / time);
  }
}

/* ending distribution handling for simulation */

void ac_final (void)
{
  solve (INFINITY);
  {
    double reward_0_ = expected (reward_func_0_);
    double reward_1_ = expected (reward_func_1_);
    double reward_2_ = expected (reward_func_2_);
    double reward_3_ = expected (reward_func_3_);
    double avg_no_of_tokens_in_place1_ = reward_0_;
    double avg_no_of_tokens_in_place2_ = reward_1_;
    double usage_of_place1_ = reward_2_;
    double usage_of_place2_ = reward_3_;

    pr_value ("avg_no_of_tokens_in_place1", avg_no_of_tokens_in_place1_);
    pr_value ("avg_no_of_tokens_in_place2", avg_no_of_tokens_in_place2_);
    pr_value ("usage_of_place1", usage_of_place1_);
    pr_value ("usage_of_place2", usage_of_place2_);
  }

  print_dist_ ("place1", 3);
}
```

D.5 trivialPN.log

```
Simulating SPN...
...Simulation finished
End of execution.
```

D.6 trivialPN.out

```
NET:
==========================================================================
        discrete places: 2
        immediate transitions: 0
        timed transitions: 2
        constant input arcs: 2
        constant output arcs: 2
        constant inhibitor arcs: 2
        variable input arcs: 0
        variable output arcs: 0
        variable inhibitor arcs: 0
==========================================================================
==========================================================================

                         TIME :   1.00E+02

==========================================================================

Simulation Analysis Result
Number of RUNS = 1000
==========================================================================

VALUE: avg_no_of_tokens_in_place1 = 0.582

VALUE: avg_no_of_tokens_in_place2 = 2.418

VALUE: usage_of_place1 = 0.483

VALUE: usage_of_place2 = 0.983

VALUE: DIST_ place1 0 = 0.517

VALUE: DIST_ place1 1 = 0.401

VALUE: DIST_ place1 2 = 0.065

VALUE: DIST_ place1 3 = 0.017
```

D.7 trivialPN.res

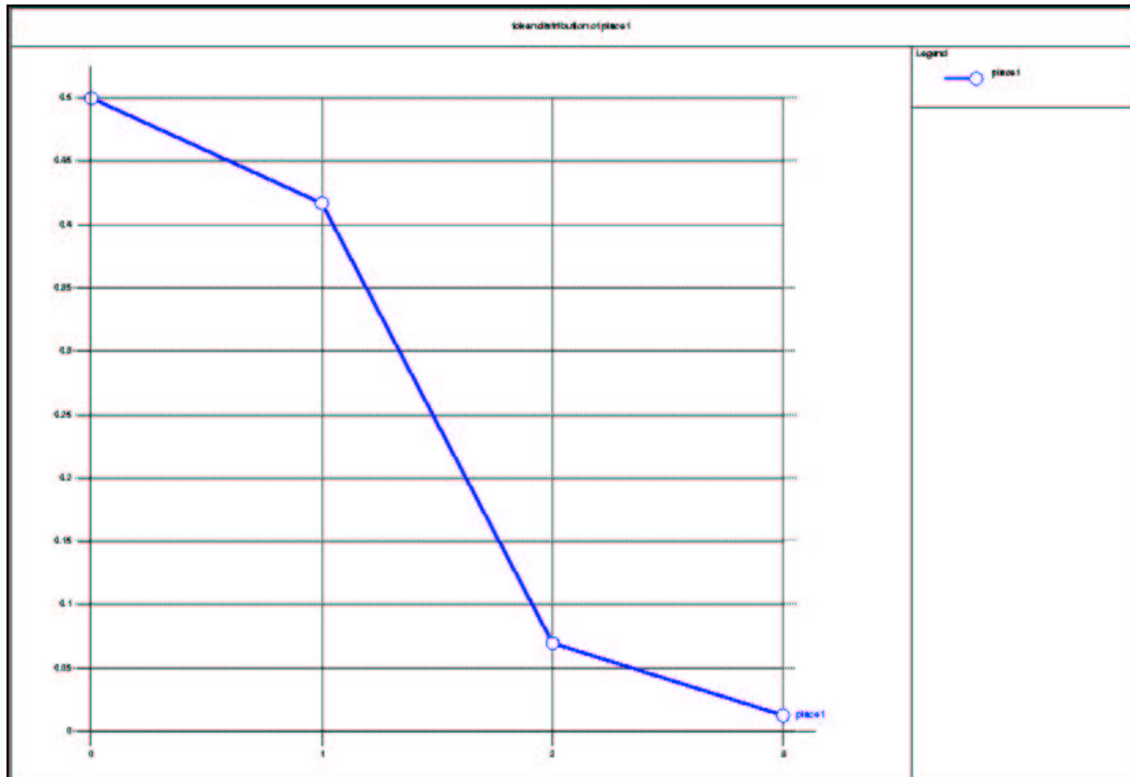
```
Simulation of "trivialPN.mos" by SPNP (simulation length: 100)

Results:
  avg_no_of_tokens_in_place1 = 0.582
  avg_no_of_tokens_in_place2 = 2.418
  usage_of_place1 = 0.483
  usage_of_place2 = 0.983

Distribution of "place1":
  P{place1 = 0} = 0.517
  P{place1 = 1} = 0.401
  P{place1 = 2} = 0.065
  P{place1 = 3} = 0.017

===============================================================================
```

D.8 Plot of trivialPN.igl



D.9 trivialPN_exp.mos

```
/* Constant + Parameter Part */
CONST rate  := 4;
CONST rate2 := 3.9;
CONST jobs := 3;
CONST init1 := 2;
CONST init2 := jobs-init1;

/* State Part */
NODE place1[jobs] := init1;
NODE place2[jobs] := init2;
ASSERT place1 + place2 = jobs;

/* Rule Part */
FROM place1 TO place2 RATE rate;
FROM place2 TO place1 RATE rate2;

/* Result Part */
PRINT avg_no_of_jobs_in_place1 := MEAN(place1);
PRINT avg_no_of_jobs_in_place2 := MEAN(place2);
PRINT usage_of_place1 := PROB(place1 != 0);
PRINT usage_of_place2 := PROB(place2 != 0);
// PRINT DIST place1;
// PRINT DIST place2;
```

Bibliography

[1] K. Begain, G. Bolch, H. Herold,
Practical Performance Modeling. Application of the MOSEL Language,
Kluwer Academic Publishers, 2001,
ISBN 0-7923-7951-9

[2] G. Ciardo, J. Muppala, K. Trivedi,
SPNP: Stochasic Petri Net Package,
in: *Proc. 3rd Int. Workshop on Petri Nets a. Performance Models (PNPM'89)*,
pp. 142-151,
Kyoto, Japan, Dec. 1989. IEEE Comp. Soc. Press.
http://www.cs.wm.edu/~ciardo/pubs/1989PNPM-SPNP.pdf (17.02.03, CD06)

[3] K. S. Trivedi,
SPNP User's Manual. Version 6.0,
Duke University, Durham, USA, 1999,
http://www.ee.duke.edu/~chirel/MANUAL/manual.pdf (06.02.03, CD01)

[4] B. Beutel,
Integration of the Petri Net Analysator TimeNET into the model analysis environment MOSEL,
University of Erlangen-Nürnberg, 2003,
http://www.linguistik.uni-erlangen.de/~bjoern/mosel-2.pdf (29.05.03, CD02)

[5] ***On the Simulation of Stochastic Petri Nets***,
Computer Science Department, College of William and Mary,
http://www.cs.wm.edu/~gli/Wpapers/PetriNet.ps (06.02.03, CD04)

[6] G. Bolch, M. Kirschnick,
PEPSY-QNS. Performance Evaluation and Prediction SYstem for Queueing NetworkS,
Universität Erlangen Nürnberg, 1993,
http://www4.informatik.uni-erlangen.de/TR/ps/TR-I4-92-21.ps.Z (06.02.03, CD05)

[7] C. A. Petri,
Kommunikation mit Automaten,
in: *Schriften des Institutes für instrumentelle Mathematik*,
Bonn, 1962

[8] B. Baumgarten,
Petri-Netze, Grundlagen und Anwendungen,
BI-Wissenschaftsverlag, 1990,
ISBN 3-411-14291-X

[9] W. Reisig,
Petrinetze, Eine Einführung,
Springer-Verlag, 1990,
ISBN 3-540-16622-X

[10] G. Bolch,
Leistungsbewertung von Rechensystemen mittels analytischer Warteschlangenmodelle,
Teubner-Verlag, 1989,
ISBN 3-519-02279-6

[11] P. Starke,
Analyse von Petri-Netz.Modellen,
Teubner-Verlag, 1990,
ISBN 3-519-02244-3

[12] G. Scheschonk,
Eine einführende Zusammenfassung der Petri-Netz-Theorie,
Technische Universität Berlin, 1977

[13] S. Greiner,
Modeling and Analysis of Operating Systems Using Extended QN Techniques and Petri Nets,
Universität Erlangen Nürnberg, 2001,
ISSN 0344-3515

[14] B. Page,
Diskrete Simulation,
Springer-Verlag, 1991,
ISBN 3-540-54421-6

[15] I. R. Chen,
Stochastic Petri Net Models,
in: *Lecture Slides on Modeling and Evaluation of Computer Systems*, pp. 150-174,
Virginia Polytechnic Institutute and State University, 2002
http://people.cs.vt.edu/~irchen/5214/pdf/p150-174.pdf (15.05.03, CD07)

[16] G. Ciardo,
Toward A Definition of Modeling Power for Stochastic Petri Net Models,
in: *Proc. of the Int. Workshop on Petri Nets and Performance Models, Madison, Wisconsin, August 1987*
Duke University, 1987,
http://www.cs.wm.edu/~ciardo/pubs/1987PNPM-ModelingPower.pdf (27.05.03, CD08)

[17] K.S. Trivedy, V.G. Kulkarni, G. Horton, D.M. Nicol,
Fluid stochastic Petri nets: Theory, applications, and solution techniques,
in: *European Journal of Operational Research 105*, pp. 184-201
1996/1998,
http://www.unc.edu/~vkulkarn/papers/fspn98.pdf (27.05.03, CD09)

[18] K. Jensen,
Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts,
Springer-Verlag, 1997,
ISBN 3-540-60943-1

[19] K. Jensen, G. Rozenberg,
High-level Petri Nets. Theory and Application,
Springer-Verlag, 1991,
ISBN: 3-540-54125-X or 0-387-54125-X

[20] B. W. Kernighan, D. M. Ritchie,
The C programming language,
Prentice-Hall, 1978,
ISBN: 0-13-110163-3

[20] G. Bolch,
Queueing networks and Markov chains,
Wiley, 1998,
ISBN: 0-471-19366-6

[21] C. Hipp,
Risikotheorie I,
TH Karlsruhe, 2001,
http://www.quantlet.de/scripts/riskt/pdf/rt1pdf.pdf (30.05.03, CD10)

[22] C. Williams,
Bayesian Methods for Parameter Estimation,
University of Edinburgh,
http://www.informatics.ed.ac.uk/teaching/modules/pmr/slides/bayespe.4up.pdf (30.05.03, CD11)

[23] W. Zucchini, F. Böker, A. Stadie,
Statistik III,
Universität Göttingen, 2001,
http://www.statoek.wiso.uni-goettingen.de/veranstaltungen/statistik3alt/daten/
(30.05.03, CD12)

[24] J. J. Filliben,
Exploratory Data Analysis,
in: *e-Handbook of Statistical Methods*,
NIST/SEMATECH, http://www.itl.nist.gov/div898/handbook/, 2002,
http://www.itl.nist.gov/div898/handbook/toolaids/pff/eda.zip (01.06.03, CD13)

[25] J. Eberspächer,
Lösung zur 8. Übung in Kommunikationsnetze 1,
TU München, 2003,
http://www.lkn.ei.tum.de/studium/veranstaltungen/kn1/uebungen/pdf/ub8_lsg.pdf
(01.06.03, CD14)

[26] S. M. Ross,
Introduction to Probability Models,
Academic Press / Elsevier Science, 2003,
ISBN: 0-12-598061-2

[27] V. Paxson, S. Floyd,
Wide Area Traffic: The Failure of Poisson Modeling,
in: *ACM TRANSACTI0NS ON NETWORKING, VOL.3, NO. 3*,
IEEE, 1995,
ftp://ftp.ee.lbl.gov/papers/WAN-poisson.ps.Z (02.06.03, CD15)

[28] S. Bodamer, J. Charzinski, K. Dolzer,
Dienstgütemetriken für elastischen Internetverkehr,
in: *Praxis der Informationsverarbeitung und Kommunikation (PIK), Band 25, Nr. 2*,
pp. 82-89
Universität Stuttgart(article) / Universität Mannheim(PIK), 2002,
http://www.ind.uni-stuttgart.de/Content/Publications/Archive/Bo_PIK2002_34653.pdf
(03.06.03, CD16)

[29] C. Hirel, B. Tuffin, K. S. Trivedi,
SPNP: Stochastic Petri Nets. Version 6.0,
Duke University,
http://www.ee.duke.edu/~chirel/PAPER/paperSpnp.ps (06.02.03, CD03)

[29] G. Bolch, S. Greiner, H. Jung, R. Zimmer,
The Markov Analyzer MOSES,
Universität Erlangen, 1994
http://www4.informatik.uni-erlangen.de/TR/pdf/TR-I4-94-10.pdf (06.02.03, CD17)

[30] A. Zimmermann,
TimeNET 3.0 - User Manual,
TU Berlin, 2001
http://pdv.cs.tu-berlin.de/~timenet/TimeNET-UserManual30.ps.gz (06.02.03, CD18)

[31] B. Tuffin, K. S. Trivedi,
Implementation of Importance Splitting Techniques in Stochastic Petri Net Package,
Duke University, Campus universitaire de Beaulieu,
ftp://ftp.eos.ncsu.edu/pub/ccsp/papers/ppr0003.PS (20.06.03, CD19)

[32] P. E. Heegaard,
Rare Event Provoking Simulation Techniques,
The Norwegian Institute of Technology, 1995,
http://www.item.ntnu.no/~poulh/publications/its.pdf (20.06.03, CD20)

[33] P. R. D'Argenio, H. Hermanns, J.-P. Katoen, R. Klaren,
MoDeST - A Modelling and Description Language for Stochastic Timed Systems,
in: *PAPM-PROMIV 2001, LNCS 2165*, pp. 87-104,
University of Twente (article) / Springer-Verlag, 2001,
http://wwwhome.cs.utwente.nl/~hermanns/papers/papm01.pdf (20.06.03, CD21)

Hint:

Due to the mostly short livetime of web offers, sources obtained online were archived on a CD-ROM enclosed to this report. This archive is meant as backup, not as publication of those contents. All sources were converted into Adobe® Portable Document Format (PDF) for readability and uniformity reasons.

Index

adaptations, 49
  basic.h, 49
  cmdline.c, 49
  cmdline.h, 50
  cspl.c, 50
  main.c, 59
  model.c, 61
  model.h, 63
  parser.y, 64
  result.c, 68
  scanner.l, 69
  timenet.c, 69
analysis
  numerical, 9

beta distribution, 11, 43
binomial distribution, 11, 43, 48

C-code, 5, 8
Cauchy distribution, 12, 43
changes, 29
  command line parameters, 29
  job distribution, 32
  new distributions, 30
  result parser, 29
clock, 9, 10, 17
code
  adaptations, *see* adaptations
  examples, 71
command line parameters, 29
conceptual model, 22
conclusion, 41
constant part, 22
CSPL, 5, 29, 40
  ac_reach(), 7
  ac_final(), 7
  ac_init(), 7
  assert(), 7
  net(), 6, 17
  options(), 5

CTMC, 4, 9

DES, 9
deterministic distribution, 12
discrete event simulation, 9
DIST, *see* job distribution
distribution, 11
  beta, 11, 43
  binomial, 11, 43, 48
  Cauchy, 12, 43
  deterministic, 5, 12
  Erlang, 13, 44, 48
  gamma, 13, 44
  geometric, 14, 44
  hyperexponential, 14, 44
  hypoexponential, 15, 44
  lognormal, 15, 44
  negative binomial, 47
  normal, 16, 45
  of jobs, *see* job distribution
  Pareto, 16, 45
  Poisson, 17, 45
  uniform, 5, 47
  Weibull, 47
DSPN, 5

eDSPN, 5
Erlang distribution, 13, 44, 48
ESPN, 5, 6, 9, 37
examples
  code, 71

FSPN, 5

gamma distribution, 13, 44
geometric distribution, 14, 44
GSMP, 9
GSPN, 4, 9

HL-SPN, 5
hyperexponential distribution, 14, 44

hypoexponential distribution, 15, 44

IGL, 24

job distribution, 32

lognormal distribution, 15, 44

Markov chain, 3, 22
Markov property, 4, 5, 9, 14, 17
memoryless, 9
MOSEL, 21
    changes, 29
    modeling, 21
    purpose, 21
    restrictions, 26

negative binomial distribution, 47
node part, 23
normal distribution, 16, 45
numerical analysis, 9

Pareto distribution, 16, 45
Petri net, 22
Petri nets, 3
    colored, 5
    deterministic stochastic, 5
    extended stochastic, 5
    fluid stochastic, 5
    generalized stochastic, 4
    high level, 5
    stochastic, 4
    untimed, 3
picture part, 24
PN, *see* Petri nets
Poisson distribution, 17, 45
PRD, 18, 45
PRI, 18, 31, 45
PRS, 19, 45

queuing network, 3, 22

re-enabling, *see* rsampling17
resampling, 6, 9, 10, 17, 45
result parser, 29
result part, 24
rule part, 23

speed up methods, 10
SPN, 4
SPNP, 3, 21
    evaluation methods, 9
    supported distributions, 11

tandem network, 3
TimeNET, 21, 37

uniform distribution, 5, 47

Weibull distribution, 47