

# **XPenelope User Guide**

for XPenelope Version 3.1

**Technical Report**

**Hermann de Meer, Hana Ševčíková**

November 1996

Department of Computer Science  
University of Hamburg

*demeer@informatik.uni-hamburg.de*  
*sevcikov@ro2.informatik.uni-hamburg.de*

## **Abstract**

XPENELOPE provides a user-friendly X window environment for PENELOPE. It supports the tasks of model creation, specification and control of series of experiments, and visualization of the results.

PENELOPE provides numerical and simulative methods based on the theory of Markov decision processes that are applicable for the optimization of performability measures. The optimization paradigm is based on the concept reconfigurability. *Transient* as well as *stationary control strategies* and *performance functions* can be computed.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Extended Markov Reward Models . . . . .	1
1.2	Features of PENELOPE . . . . .	1
1.3	Platforms . . . . .	3
<b>2</b>	<b>XPenelope User Guide</b>	<b>3</b>
2.1	Invocation . . . . .	3
2.2	The Model Index . . . . .	4
2.3	The Parameter Set Index . . . . .	5
2.4	The Experiment Index . . . . .	6
2.5	The Model Editor . . . . .	7
2.5.1	The File Menu . . . . .	8
2.5.2	The Edit Menu . . . . .	8
2.5.3	The Options Menu . . . . .	11
2.5.4	Arithmetic Expressions . . . . .	11
2.5.5	The Forward Declaration Dialog . . . . .	13
2.6	The Parameter Set Editor . . . . .	13
2.7	The Experiment Editor . . . . .	14
2.8	The Strategy Index . . . . .	18
2.8.1	The Strategy Editor . . . . .	19
2.9	The Graph Parameters Dialog . . . . .	20
2.9.1	The Forward Experiment Dialog . . . . .	24
2.9.2	Change Graph Dialog . . . . .	26
2.10	Resources . . . . .	26
<b>3</b>	<b>Examples</b>	<b>29</b>
3.1	Example 1 . . . . .	29
3.2	Example 2 . . . . .	35

**4 Conclusion**

**38**

**References**

**39**

# List of Figures

1	Model index. . . . .	4
2	Parameter set index. . . . .	5
3	Experiment index. . . . .	6
4	Model editor. . . . .	7
5	“Macros” dialog box. . . . .	9
6	“Parameters” dialog. . . . .	10
7	Macro parameters dialog. . . . .	11
8	“Forward Declaration” dialog. . . . .	13
9	Parameter set editor. . . . .	14
10	Experiment editor. . . . .	15
11	“Parameter Sets” dialog. . . . .	15
12	“Working...” dialog. . . . .	17
13	“Info” window. . . . .	17
14	Strategy index. . . . .	18
15	Strategy editor. . . . .	19
16	“Graph Parameters” dialog. . . . .	20
17	“Value”, “Difference Values” and “Which rewards” dialogs. . . . .	20
18	“Parameter Set” dialog. . . . .	22
19	“Which Experiment” dialog. . . . .	22
20	“Dif.Exp.Parameter” dialog. . . . .	22
21	A plot program displaying a strategy. . . . .	24
22	Forward Experiment dialog for equidistant mode. . . . .	24
23	Forward Experiment dialog for multiple mode. . . . .	25
24	“Change Graph” and “Change Lines” dialog. . . . .	26
25	A simple model . . . . .	29
26	Model editor containing six Markov states and one branching state . . . . .	30
27	States and transitions. . . . .	31

28	The completed model. . . . .	32
29	The parameter set editor. . . . .	33
30	The results of the experiment. . . . .	40
31	Model of the Erlang- <i>k</i> distribution . . . . .	40
32	Recursive nature of the Erlang- <i>k</i> distribution . . . . .	41
33	Macro for the Erlang- <i>k</i> distribution . . . . .	41
34	The <i>Connect</i> macro . . . . .	41
35	The <i>Erlang</i> macro . . . . .	42
36	A simple model with an invocation of the <i>Erlang</i> macro . . . . .	42

# 1 Introduction

PENELOPE is a software tool for the “**dependability evaluation and the optimization of performability**”. PENELOPE is based on the theory of *extended Markov reward models* [DEME 92]. The primary task of PENELOPE is the optimization of performability measures. The user may interact with PENELOPE either through a graphical user interface or through a command line interpreter.

## 1.1 Extended Markov Reward Models

Let  $Z = \{Z(t), t \geq 0\}$  denote a continuous time Markov chain with finite state space  $\mathbf{S}$ . To each state  $s \in \mathbf{S}$  a real-valued *reward rate*  $r(s)$ ,  $r : \mathbf{S} \rightarrow \mathbb{R}$ , is assigned, such that if the Markov chain is in state  $Z(t) \in \mathbf{S}$  at time  $t$ , then the *instantaneous reward rate* of the Markov chain at time  $t$  is defined as  $X(t) = r_{Z(t)}$ . In the time horizon  $[0, \dots, t)$  the *total reward*  $Y(t) = \int_0^t X(\tau) d\tau$  is accumulated. Note that  $X(t)$  and  $Y(t)$  depend on  $Z(t)$  and on an initial state. The distribution function  $\Psi(y, t) = P(Y(t) \leq y)$  is called the *performability*. A Markov chain  $Z$ , for which a reward function  $r$  has been defined, is called a *Markov reward model*. For ergodic models the instantaneous reward rate and the time averaged total reward converge in the limit to the same overall reward rate  $\lim_{t \rightarrow \infty} E[X(t)] = \lim_{t \rightarrow \infty} \frac{1}{t} E[Y(t)] = E[X]$ .

For each unit of time in which the system is in a certain state, the corresponding number of reward units  $X(t)$  is added to the total reward  $\int_0^t X(\tau) d\tau$ . Thus, it is possible to define a “yield measure” or a “loss measure” for the system by using appropriate reward rates.

*Extended Markov reward models* (EMRMs) enrich Markov reward models by the feature of *reconfiguration edges*: Whenever the system is in a state in which a reconfiguration edge originates, a decision must be made whether or not to switch to the target state of the reconfiguration edge. The optimal decision may be time dependent. Additionally, the Markov reward models are extended by so-called *branching states*. These are states in which the system does not spend any time. With the help of branching states, it is possible to assign reward values to state transitions (so-called *pulse rewards*).

EMRMs were derived in order to provide an abstraction from Markov decision theory. EMRMs are a marriage between performability modeling techniques and Markov decision theory and provide a general framework for the dynamic optimization of **reconfigurable**, dependable systems.

## 1.2 Features of PENELOPE

Reconfiguration edges denote options to reconfigure from one state to another. At every instant of time a different decision is possible for each reconfiguration edge. A strategy  $S(t)$  comprises the set of all possible decisions at a particular instant of time  $t$ ,  $0 \leq t \leq T$ . Strategies can be time dependent or stationary. In the latter case the parameter  $t$  can be dropped:  $S = S(t)$ . A strategy  $\hat{S}(t)$  is considered *optimal* if the performance of the system under strategy  $\hat{S}(t)$  is better or equals than the performance of the system under any other strategy  $S(t)$ .

PENELOPE offers two types of methods for the computation of **optimal strategies** and **performance functions**:

- **Transient Optimization:**

The system is investigated for a finite period of time, the so-called *mission time*  $T$ . Optimal strategies  $\hat{S}(t)$  are computed for every instant of time  $t$ ,  $0 \leq t \leq T$ . The results are depicted as  $\hat{S}(t')$ , which is a function of the **remaining** time  $t'$ ,  $t' = T - t$ , where  $t$  denotes the **elapsed** time. In addition, the expected accumulated reward  $E[Y_i(t)]$  is also computed for **all** initial states  $i$ , provided the optimal strategies were applied.

- **Stationary Optimization:**

The optimization is performed for an infinite time horizon. A strategy  $\hat{S}$  is optimal if no other strategy  $S$  exists such that the time averaged mean total reward  $E[X_i] = \lim_{T \rightarrow \infty} \frac{1}{T} E[Y_i(t)]$  gained under strategy  $S$  exceeds the one gained under  $\hat{S}$ .  $E[X_i]$  can be dependent or independent of state  $i$ . In the latter case  $E[X] = E[X_i]$  for all  $i$ , is called the (optimal) overall reward rate. The optimization itself is performed by the deployment of standard methods from Markov decision theory [TIJM 86]. The adaptation of these methods to the analysis of EMRMs is out of the scope of this reference manual. More details can be found in [KÄTK 92].

In addition, PENELOPE offers procedures for computations under **fixed** deliberately eligible **strategies**:

- **Simulation**

Under a fixed strategy, the behavior of the system is simulated. The mean total accumulated reward for an arbitrary initial state is computed.

- **Transient Analysis**

The transient analysis is carried out under a fixed strategy that can be deliberately specified. The strategy can be either of a transient or a stationary type.

- **Stationary Analysis**

The stationary analysis is carried out under a fixed strategy. The strategy can only be of a stationary type.

PENELOPE offers two possibilities for the user to interact with the system:

- The extended Markov reward models are defined by using a model description language. These models can be analyzed and optimized by invoking appropriate commands from the PENELOPE command line interpreter. The computed strategies are returned in a textual format with a clearly defined syntax. This enables automated preprocessing and postprocessing of model/strategy data.
- Using the graphical user interface XPenelope [MUNK 93], the models are created interactively. In this case, the user is working with familiar symbols such as circles (for states) and arrows (for state transitions).



Additionally, the graphical user interface offers a lot of extra functionality:

- Automated execution of experiment series.
- Graphical preparation of experiment results.
- Automated checking of model consistency.
- Definition and evaluation of hierarchical and iterative models, using a special macro mechanism.
- Definition of model parameters and an arbitrary number of parameter sets per model.
- Preparation of computed strategies.
- Operating system independent data management:
  - model data base
  - parameter set data base
  - experiment and results data base
  - strategy data base
- Printing of models and results for documentation purposes.

## **1.3 Platforms**

XPENELOPE runs under the following architectures:

- Sun 4 with SunOS 4
- Sun 4 with SunOS 5 (Solaris)

XPENELOPE requires an existence of the X Window System release 11 version 4 or higher and of the OSF/Motif GUI system version 1.1 or higher.

## **2 XPenelope User Guide**

### **2.1 Invocation**

XPenelope is invoked by executing the command

```
xpenelope
```

from the command line. This will bring up the model index (see section 2.2).

If `xpenelope` is called for the first time, the model database will be created and some sample models will be copied into the user's database. Normally, the location for the model database is the directory

```
$HOME/.xpenelope
```

This may be overridden by setting the environment variable `PENELOPEDB` to the desired path name.

## 2.2 The Model Index



Figure 1: Model index.

The model index is a list of all predefined models and macros (see Fig. 1). Macros are marked with the suffix “<M>”. One of the following buttons may be pressed:

**Create** Creates a new model. The model editor is opened (see Sec. 2.5).

**Copy** †Makes a copy of the selected model. Type in the desired model name in response to the prompt “Name of copy:”, and press **OK**. Then the whole model together with its parameter sets, experiment definitions and results of experiment series, and strategies is copied.

**Rename** †Renames a model. Type in the desired model name in response to the prompt “New model name:”, and press **OK**.

**Delete** †Deletes a model together with its parameter sets, experiment definitions and results of experiment series, and strategies.

**Open** †Brings up a pull-down menu, which offers the following choices:

**Model Editor...** Opens the model editor (see Sec. 2.5).

**Parameter Index...** Shows/Edits the list of all parameter sets defined for the model (see Sec. 2.3).

**Experiment Index...** Shows/Edits the list of all experiment definitions (see Sec. 2.4).

**Print** †Prints the model as a PostScript file. A dialog with the following choices appears:

**Send to printer:** Sends the PostScript file to the indicated printer.

**Send to file:** Stores the PostScript graphics in the specified file.

**Quit** Quits the program.

All operations marked with † require that a model (or a macro) has been previously selected from the list.

A double-clicking of a list item performs the **Open — Model Editor...** action.

## 2.3 The Parameter Set Index

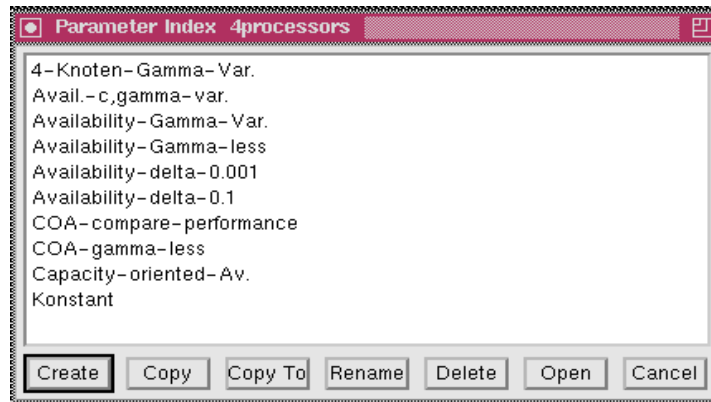


Figure 2: Parameter set index.

The parameter set index (see Fig. 2) is a list of all parameter sets available for the selected model. One of the following buttons may be selected:

**Create** Creates a new parameter set. The parameter set editor is opened (see Sec. 2.6).

**Copy** †Makes a copy of the selected parameter set. Type in the desired parameter set name in response to the prompt “Name of copy:” and then press **OK**.

**Copy To** †Makes a copy of the selected parameter set to an arbitrary model. Type in the desired parameter set name in response to the prompt “Name of copy:” and then press **OK**. Then the name of the model is prompted for.

**Rename** †Renames a parameter set. Type in the desired parameter set name in response to the prompt “New parameter set name:” and then press **OK**.

**Delete** †Deletes the selected parameter set.

**Open** †Opens the parameter set editor (see Sec. 2.6) for the selected parameter set.

**Cancel** Leaves the parameter set index.

All operations marked with † require that a parameter set has been previously selected from the list.

A double-clicking of a list item will perform the **Open** action.

## 2.4 The Experiment Index



Figure 3: Experiment index.

The experiment index (see Fig. 3) is a list of all experiments available for the selected model. One of the following buttons may be selected:

**Create** Creates a new experiment. A name of the new experiment is asked for. Then the experiment editor is opened (see Sec. 2.7).

**Copy** †Makes a copy of the selected experiment. Type in the desired experiment name in response to the prompt “Name of copy:” and then press **OK**. Then copies of the experiment definition and of the results of this experiment are made.

**Copy To** †Makes a copy of the selected experiment to an arbitrary model. Type in the desired experiment name in response to the prompt “Name of copy:” and then press **OK**. Then a name of the model is asked for.

**Rename** †Renames an experiment. Type in the desired experiment name in response to the prompt “New experiment name:” and then press **OK**.

**Delete** †Deletes the selected experiment.

**Open** †Opens the experiment editor (see Sec. 2.7) for the selected experiment.

**Cancel** Exits the experiment index.

All operations marked with † require that an experiment has been previously selected from the list.

A double-clicking of a list item will perform the **Open** action.

## 2.5 The Model Editor

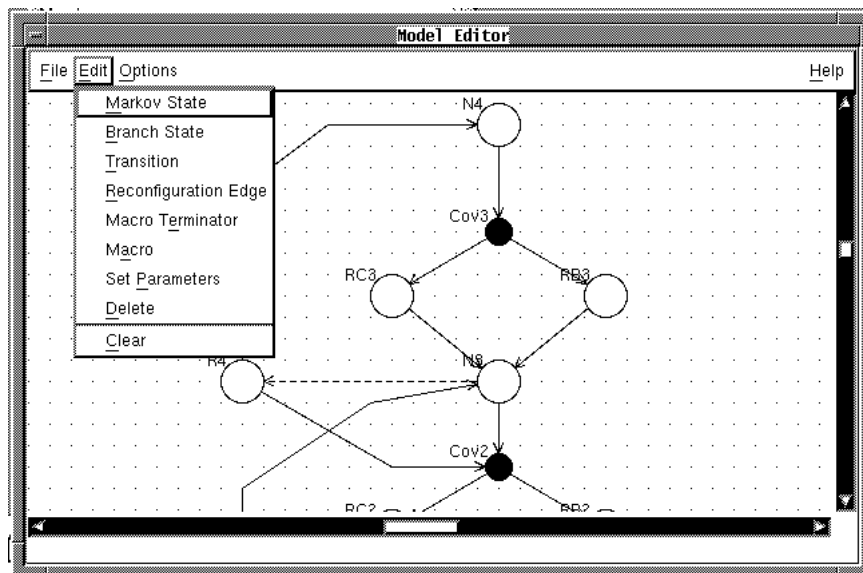


Figure 4: Model editor.

The model editor (see Fig. 4) is a “canvas” on which state-transition diagrams can be rendered. The following symbolism is used within the model editor:

Object	Representation
Markov State	○
Branching State	●
Transition	→
Reconfiguration Edge	- - - - ->
Macro Terminator	□
Macro	□ ↳   ↳ M
Missing Parameters	○ ?
Inconsistent Macro	□ ↳   ↳ M X

The arrows inside the macro symbols are called “sockets”. They point inward (outward), if the attached transition or reconfiguration edge is entering (leaving) the macro.

The menu bar of the model editor offers four choices:

- File** Loading, saving, and printing models.
- Edit** Creating, modifying, and deleting objects.
- Options** Grid and node name operations.
- Help** Currently, only the menu item **Help on Version** is offered. It displays the version number of the program.

The following sections discuss the individual menus.

### 2.5.1 The File Menu

- New...** Creates a new model.
- Open...** Loads a new model that is selected from the model index.
- Save** Saves the model. If the model has been previously saved, the same file name will be used. Otherwise, a file name is asked for.
- Save As...** Saves the model under a new file name. The file name is asked for.
- Print...** Prints the model as a Postscript file. Options are available to save the file or to send it directly to a printer.
- Exit** Leaves the model editor.

### 2.5.2 The Edit Menu

The first eight choices of the edit menu determine which action is performed if the left mouse button is pressed (the “select button”) within the model editor. A indicator to the left of the menu items shows which item currently is enabled.

- Markov State** Activates the Markov state mode. In this mode, a new Markov state is created if the select button is pressed.
- Branch State** Activates the branch state mode. In this mode, a new branching state is created if the select button is pressed.
- Transition** Activates the transition mode. In this mode, the following steps have to be performed in order to create a transition:
  1. Click at the start node.

2. If no intermediate vertex is needed, goto step 3, otherwise click anywhere in the canvas (except over other objects) in order to create a new vertex. Repeat this step until all intermediate vertices have been created.
3. Click at the target node. Now the transition will be displayed.

**Reconfiguration Edge** Activates the reconfiguration edge mode. In this mode, an equivalent procedure as in the transition mode is used in order to create a reconfiguration edge.

**Macro Terminator** Activates the macro terminator mode. In this mode, a new macro terminator is created if the select button is pressed. Models containing macro terminators are called “macros”; a macro terminator is the connection of the inside world of the macro to the outside world. If a macro is invoked by using the **Macro** item of the **Edit** menu, the macro terminators will be shown as little arrows (“sockets”) within a larger frame. The direction of an arrow indicates the direction of the transition which is attached to the corresponding terminator.

**Macro** Activates the macro mode. In this mode, a new macro invocation is created if the select button is pressed.

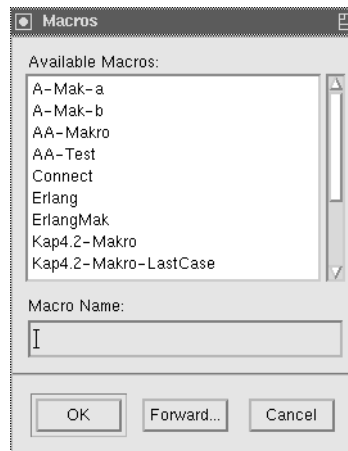


Figure 5: “Macros” dialog box.

Having the **Macro** menu item selected, the “Macros” dialog box pops up (see Fig. 5). It contains a list of available macros; select one of the list items and then press **OK**. Press **Cancel** if the operation should be aborted.

If a recursive macro definition is to be created a macro which has not yet been defined has to be used. The **Forward...** button can be selected in order to create a macro forward declaration (see Sec. 2.5.5).

If a macro invocation and a macro definition don’t match, the macro invocation will be displayed as a crossed-out box. In this case, the inconsistent macro invocation must be deleted and a new consistent invocation has to be created.

## Set Parameters

Activates the parameter modification mode. In this mode, any object may be selected in order to modify its parameters. (This does not apply to reconfiguration edges, which are parameterless.) If an object has been newly created, a question mark (“?”) is displayed on top of the object; this indicates that there are still parameters to be set.

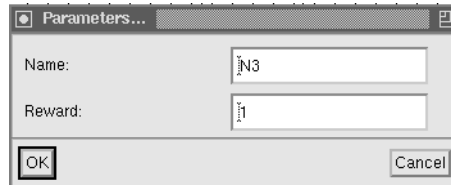


Figure 6: “Parameters” dialog.

If an object is selected a dialog box will appear allowing to change the object parameters (see Fig. 6). The **OK** button must be pressed in order to apply the new parameters to the object. No changes are made if the **Cancel** button is pressed.

- A Markov state or a branching state has the following parameters:
  - Name:** Name of state. A state name must start with a letter and may contain any of the characters A...Z, a...z, 0...9, -, (, ), ', [ and ].
  - Reward:** Reward rate; any arithmetic expression may be entered in this field (see Sec. 2.5.4).
- A transition has one of the following parameters:
  - Rate:** State transition rate (applies only to transitions originating in a Markov state).
  - Probability:** Probability of the transition (applies only to transitions originating in a branching state).

Any arithmetic expression may be entered.

- A macro terminator has the following parameters:
  - Name:** Name of the terminator.
- A macro invocation has the following parameters (see Fig. 7):
  - Name:** Name of the macro invocation.
  - Class:** Class of the invoked macro; this is the model name assigned to the macro definition.
  - Condition:** Any arithmetic expression. If the expression is zero, the macro invocation will be ignored. Otherwise, the macro invocation will be substituted by its definition (at experiment execution time). This field is initialized with 1.0.
  - Parameters:** A list of all parameters that appears in the macro definition. Each parameter may be substituted by



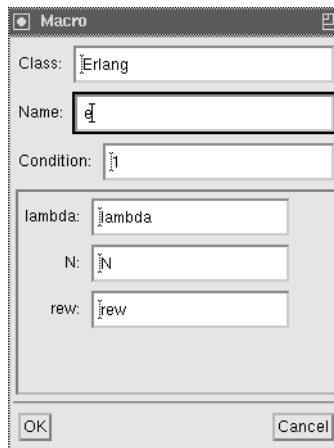


Figure 7: Macro parameters dialog.

an arithmetic expression; the substitution is performed at experiment execution time. Initially, each parameter is substituted by itself.

The names of all states, terminators, and macro invocations in a model must be unique.

**Delete** Activates the delete mode. In this mode, the select button may be pressed over an object in order to delete it.

**Clear** Erases the whole model.

### 2.5.3 The Options Menu

The options menu consists of five toggle buttons:

**Show Grid** Makes the grid visible/invisible.

**Snap to Grid** Activates/deactivates grid-snap mode. In this mode, all newly created objects will be centered with respect to the nearest intersection of the grid.

**Show Node Names** Shows/Hides the names of states and terminators.

**Show Node Rewards** Shows/Hides the reward values of states.

**Show Transition Rates** Shows/Hides the rates of transitions.

### 2.5.4 Arithmetic Expressions

An arithmetic expression consists of numbers, parameter names, operators, and function calls. The operators obey the usual precedence rules for arithmetic operators:

Operator	Precedence	Meaning
-	1 (highest)	unary minus
^	2	exponentiation
*	3	multiplication
/	3	division
+	4	addition
-	4	subtraction
<	5	less-than comparison
>	5	greater-than comparison
<=	5	less-than-or-equal comparison
>=	5	greater-than-or-equal comparison
=	5	test for equality
<>	5 (lowest)	test for inequality

Parentheses may be used to override the default precedence rules.

If the comparison succeeds the comparison operators on precedence level 5 return 1.0 otherwise they return 0.0. The comparison operators are generally only useful in the “Condition” field of the “Macro” dialog box.

The following functions are defined:

Function	Meaning
SQRT( $x$ )	square root: $\sqrt{x}$
EXP( $x$ )	power to the base $e$ : $e^x$
LN( $x$ )	natural logarithm: $\ln x$

A number is a sequence of digits with an optional embedded decimal point. An exponent suffix may be appended, i.e., the letter e, an optional sign, and a sequence of digits.

Examples:

3  
3.14  
0.0314e2  
314e-2

A parameter is a sequence of characters starting with a letter; the parameter name may contain letters, digits and underscores (“\_”). The actual value of a parameter is defined in the parameter set editor (see Sec. 2.6).

Examples:

$(1 + x)^3$	<code>(1 + x) ^ 3</code>
$2 \cdot a \cdot e^{1-x}$	<code>2 * a * EXP (1-x)</code>
$\sqrt{\frac{1+x}{1-x}}$	<code>SQRT ((1+x)/(1-x))</code>

## 2.5.5 The Forward Declaration Dialog

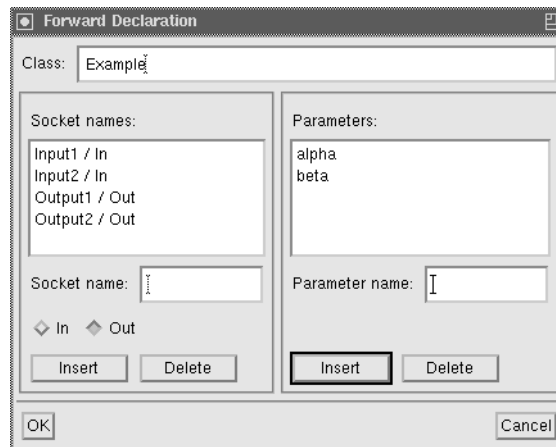


Figure 8: “Forward Declaration” dialog.

The purpose of the forward declaration dialog (see Fig. 8) is to declare a macro which has not yet been defined. The following information has to be provided:

- Class**                    The name of the model that will contain the macro definition.
- Socket names**        The names of the macro terminators. Use the button **Insert** to insert new socket names and the button **Delete** to delete the selected socket names. The radio buttons **In** and **Out** determine the direction of the socket.
- Parameters**            The names of the macro parameters. Use the button **Insert** to insert new parameter names and the button **Delete** to delete the selected parameter names.

Press the **OK** button in order to create the forward declaration, or press **Cancel** to abort the operation.

## 2.6 The Parameter Set Editor

The parameter set editor (see Fig. 9) is used to assign concrete values to the model parameters. Each parameter may have one or more values. For all possible value combinations one experiment will be executed.

The numbers entered in the parameter set editor consist of digits, an optional embedded decimal point, and an optional exponent suffix. For example, both 3.14 and 0.314e2 are valid numbers.

Each number is rounded to twenty digits:  $1e-21 = 0$ .

Select a parameter from the parameter index and activate one of the following radio buttons:

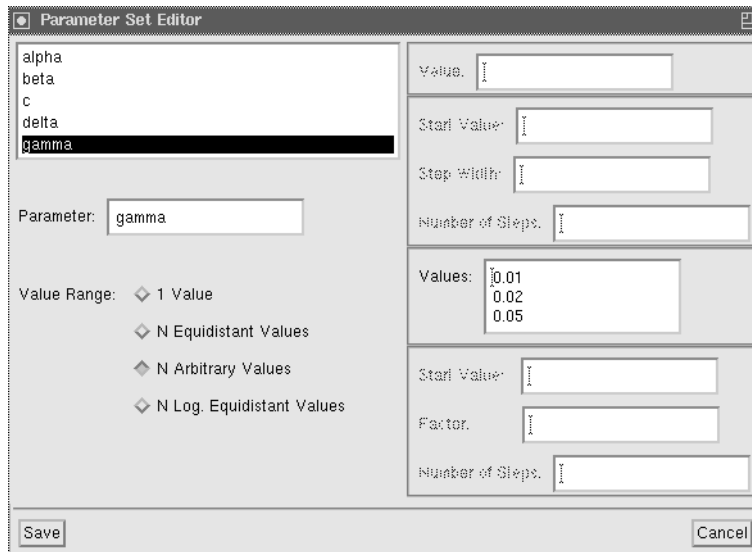


Figure 9: Parameter set editor.

**1 Value** There is exactly one value for this parameter (field **Value**).

**N Equidistant Values** The values for this parameter are taken from the set

$$\{x | x = a + s \cdot i, i = 0 \dots N - 1\}$$

where  $a$  is the value from the field **Start Value**,  $s$  from the field **Step Width** and  $N$  from the field **Number of Steps**.

**N Arbitrary Values** The values are taken from the field **Values**. This field can hold an arbitrary number of values. Each number must be in a separate line.

**N Logarithmic Equidistant Values** The values for this parameter are taken from the set

$$\{x | x = a \cdot f^i, i = 0 \dots N - 1\}$$

where  $a$  is the value from the field **Start Value**,  $f$  from the field **Factor** and  $N$  from the field **Number of Steps**.

Press the **Save** button in order to save the parameter set, or press **Cancel** to exit the parameter set editor.

## 2.7 The Experiment Editor

The experiment editor (see Fig. 10) is used to create experiment definitions, to run experiments, to save experiment definitions and results, to display results, and to save computed strategies.

An experiment definition requires the following information:

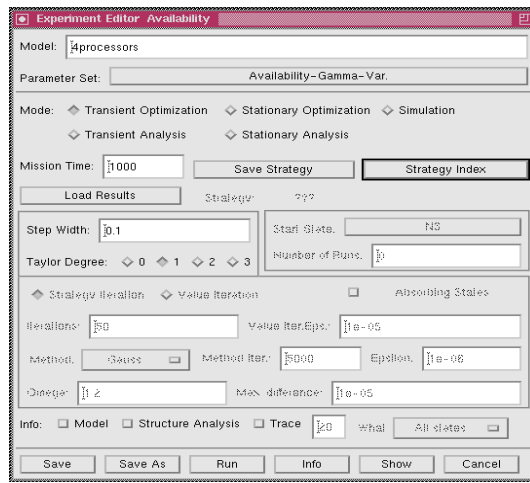


Figure 10: Experiment editor.

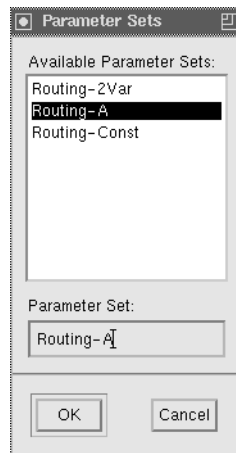


Figure 11: “Parameter Sets” dialog.

- Parameter Set** The name of the associated parameter set. Press this button in order to display the “Parameter Sets” dialog (see Fig. 11). One of the available parameter sets may be selected. Press **OK** to return to the experiment editor.
- Mode** One of the modes **Transient Optimization**, **Stationary Optimization**, **Simulation**, **Transient Analysis**, or **Stationary Analysis** can be selected.
- Mission Time** A mission time must be specified for the transient optimization, transient analysis, and for the simulation.
- Strategy** A fixed strategy may be specified for the transient and stationary analysis and for the simulation. It can be loaded in the Strategy Index (see Sec. 2.8 – Load). The strategy is considered as a **fixed** strategy for this experiment, no optimization will be performed.
- Step Width** The step width for the transient optimization and the transient analysis determining the accuracy can be specified.

<b>Taylor Degree</b>	The values 0, 1, 2 or 3 are possible. The default Taylor degree is 1.
<b>Start State</b>	The start state of the simulation.
<b>Number of Runs</b>	The number of runs for the simulation.
<b>Iteration Mode</b>	<b>Strategy Iteration</b> or <b>Value Iteration</b> may be selected for the stationary optimization.
<b>Absorbing States</b>	Click at this button if the model contains absorbing states for the appropriate computation method to be performed for the stationary optimization and stationary analysis.
<b>Iterations</b>	Number of iterations to be performed.
<b>Value Iter. Eps.</b>	Termination criterion for Value Iteration. Default: $10^{-5}$ .
<b>Method</b>	Available methods for the solution of linear equations: <b>Gauss</b> , <b>Power</b> , <b>Gauss-Seidel</b> , or <b>SOR</b> .
<b>Method Iter.</b>	Number of iterations for the iterative methods (Power, Gauss-Seidel, SOR).
<b>Epsilon</b>	Termination criterion for iterative methods for the solution of linear equations. Default: $10^{-6}$ .
<b>Omega</b>	Relaxation parameter for SOR method. Default: 1.2.
<b>Max. difference</b>	In the strategy iteration, two values $a$ and $b$ are considered as equal if $ a - b  < d$ , where $d$ is the value of the <b>Max. difference</b> field. Default: $10^{-5}$ .
<b>Info</b>	The toggle button <b>Model</b> determines whether the info file should contain the model description or not. The toggle button <b>Structure Analysis</b> determines whether a structure analysis of the model should be provided or not. If the toggle button <b>Trace</b> is on a trace of the computation is provided for the given time steps. Default: 20. In the menu <b>What</b> , one can specify the tracing states to be traced.

Meaning of the buttons:

<b>Save Strategy</b>	This button may be pressed if the experiment was run in the former and computed strategies exist. A strategy name is asked for.
<b>Strategy Index</b>	Shows/Edits the list of all strategies saved for this model (see Sec. 2.8).
<b>Load Results</b>	If the results of the experiment were previously saved they can be loaded (resp. reloaded) from the experiment database by clicking at this button.
<b>Save</b>	Saves the experiment definition, and the results of this experiment if there are any.

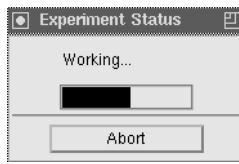


Figure 12: “Working...” dialog.

- Save As** Saves the experiment definition, and the results of this experiment if there are any under a new name. The experiment name is asked for.
- Run** Starts the experiment. While the experiment is performed, a “Working...” dialog appears (see Fig. 12). The experiment may be aborted with the **Abort** button of the “Working...” dialog.

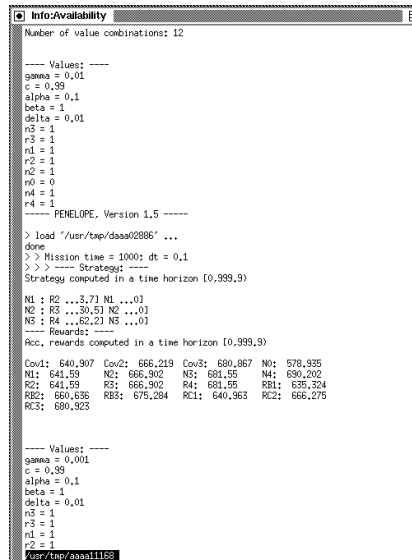


Figure 13: “Info” window.

- Info** Shows information on the performed experiment (see Fig. 13). The Info file contains:

- Parameter combinations.
- †Model (in PENELOPE’s model description language).
- †Structure analysis of the model.
- †Output of the trace function on defined states.
- Optimal strategies.
- Accumulated rewards.
- State probabilities.
- Error messages of PENELOPE.

All informations marked with † require that the corresponding Info toggle button is set on (see bellow).

The Info is displayed using the UNIX `less` program.

- Show** Shows the results of the experiment. The “Graph Parameters” dialog pops up (see Sec. 2.9).
- Cancel** Leaves the experiment editor.

## 2.8 The Strategy Index

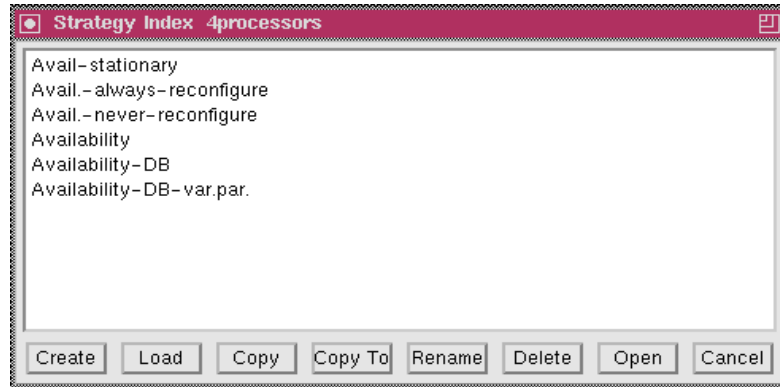


Figure 14: Strategy index.

The strategy index (see Fig. 14) is a list of all strategies available for the selected model. One of the following buttons may be pressed:

- Create** Creates a new strategy. A new strategy name is asked for. Then the strategy editor is opened (see Sec. 2.8.1).
- Load** †Loads the selected strategy in the experiment definition. The name of the loaded strategy appears in the experiment editor. If the selected strategy is of a transient type the mission time of this strategy appears in the widget “Mission Time” in the experiment editor.
- Copy** †Makes a copy of the selected strategy. A name of the copy is asked for.
- Copy To** †Makes a copy of the selected strategy to an arbitrary model. Type in the desired strategy name in response to the prompt “Name of copy:” and then press **OK**. Then a name of the model is asked for.
- Rename** †Renames a selected strategy. Type in the desired strategy name in response to the prompt “New strategy name:”, and then press **OK**.
- Delete** †Deletes the selected strategy.
- Open** †Opens the strategy editor for the selected strategy (see Sec. 2.8.1).
- Cancel** Exits the strategy index.

All operations marked with † require that a strategy has been previously selected from the list.

A double-clicking of a list item will perform the **Open** action.



## 2.8.1 The Strategy Editor

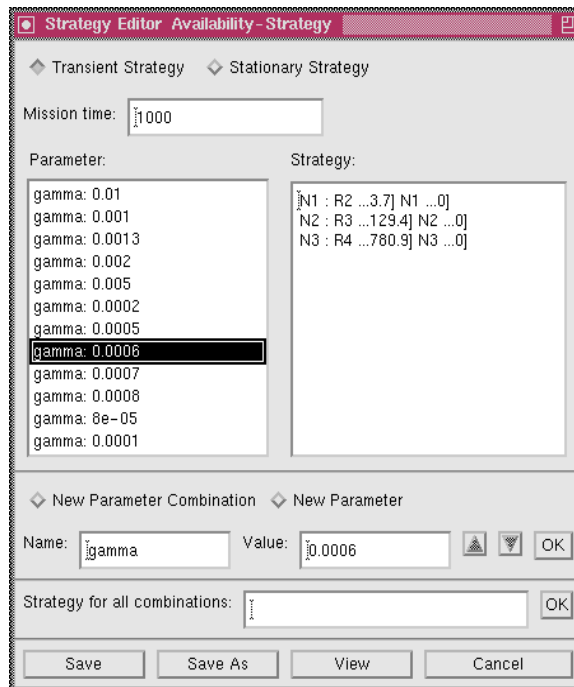


Figure 15: Strategy editor.

The strategy editor (see Fig. 15) is used to create, modify, and display strategies.

It contains:

- Toggle buttons **Transient Strategy** and **Stationary Strategy** which determine a type of the selected strategy.
- **Mission time**. The mission time is used only for transient strategies.
- A list of **Parameter** combinations. Each combination is represented by names and values of the varying parameters. Both, name and value, can be modified in the text widget **Name** and **Value**. By clicking at the **up arrow** (resp. **down arrow**), the previous (resp. next) parameter of the selected parameter combination appears. The button **OK** next to the arrows must be pressed in order to apply the parameter modification to the parameter combination list.

To create a new parameter of the selected combination click at the toggle button **New Parameter**, write a name, and a value and then press OK.

To create a new combination click at the toggle button **New Parameter Combination**, write a name, and a value of the first parameter in the new combination and then press OK. New combination is created. The remaining parameters of this combination can be put in by clicking at **New Parameter**.

- **Strategy.** This text widget contains a strategy of the selected parameter combination. For the syntax see [MAUS 90]. Both, transient and stationary syntax, are valid. The strategies can be modified by the user. Each line must be terminated with new line character.

In the text widget **Strategy for all combinations**, write a line that is equal for all parameter combinations. By pressing the button **OK** next to this text widget, strategies for all parameter combinations are modified in the line with the same reconfiguration state.

Meaning of the buttons:

**Save** Saves the strategy editor.

**Save As** Saves the strategy editor under a new name. The strategy name is asked for.

**View** Shows the current strategy. The “Graph Parameters” dialog pops up (see Sec. 2.9).

**Cancel** Leaves the strategy editor.

## 2.9 The Graph Parameters Dialog

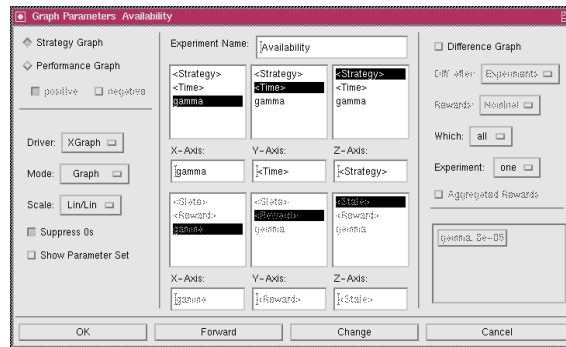


Figure 16: “Graph Parameters” dialog.

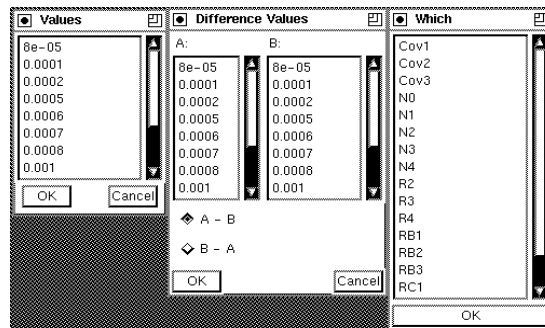


Figure 17: “Value”, “Difference Values” and “Which rewards” dialogs.

XPenelope uses 2D-plot programs to display the results of the experiments. The “Z-Axis” of the graph parameters dialog (see Fig. 16) represents the capability to plot multiple data sets per graph.

Select a kind of graph by clicking at one of toggle buttons:

**Strategy Graph** The three lists “X-Axis”, “Y-Axis” and “Z-Axis” contain the possible axis labels “<Time>”, “<Strategy>” and names of the parameters with more than one value.

**Performance Graph** By selecting the toggle button **positive** the values of reward rates will be displayed as positive values. By selecting the toggle button **negative** the values of reward rates will be displayed as negative values. The three lists “X-Axis”, “Y-Axis” and “Z-Axis” contain the possible axis labels “<State>”, “<Reward>” and names of the parameters with more than one value.

In the text widgets under each list, the user may change description of the axis.

If the model has more than one varying parameter, then fixed values must be selected for the parameters. This is done by clicking at one of the buttons marked with the parameter name, and by selecting one or more of the possible values from the “Values” dialog (see Fig. 17). Default value is the first value in the list of the possible values. If more than one value is selected the parameter gets a variable mode. No more than one parameter may have the variable mode.

In the **Driver** menu, a plot program can be selected. By default, the following drivers are installed:

**XGraph** Driver for the `xgraph` program (a freely distributable plot program).

**Gnuplot** Driver for the `gnuplot` program (also freely distributable).

**Text** Driver for a textual representation of the results; the text is displayed using the UNIX `view` program.

**ACE/GR** Driver for the ACE/GR program (a freely distributable plot program) [TURN 92].

**XGadd** Driver for the `xgadd` program, which has been developed for the PEPSY tool at the University of Erlangen-Nürnberg [MEIT 92].

In the **Scale** menu can be specified if the X (or Y) axis should have a linear or a logarithmic scale. There are four possibilities:

**Lin/Lin** X axis: linear; Y axis: linear.

**Lin/Log** X axis: linear; Y axis: logarithmic.

**Log/Lin** X axis: logarithmic; Y axis: linear.

**Log/Log** X axis: logarithmic; Y axis: logarithmic.

In the **Mode** menu can be chosen between a graph (**Graph**) or a bar chart (**Histogram**). The menus **Mode** and **Scale** do not apply to the **Text** driver.

The toggle button **Suppress 0s** determines whether strategy changes which always happen at time 0 should be suppressed or not. The default is to suppress these strategy changes.

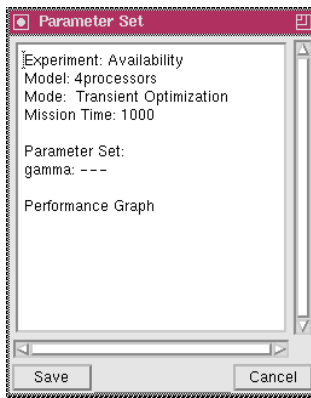


Figure 18: “Parameter Set” dialog.

The toggle button **Show Parameter Set** determines whether a window containing graph information (see Fig. 18) is shown by pressing the OK button or not. This window is for a documentation purpose. This information can be saved in a data file by clicking at the button **Save**. Then a file name is asked for.

In the text widget **Experiment Name** the inscription of the resulting graph may be changed.

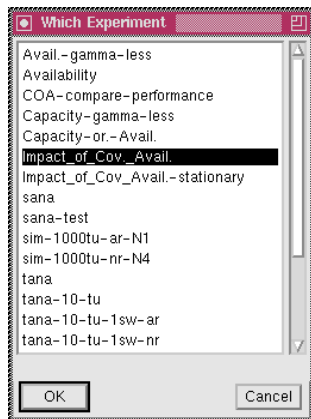


Figure 19: “Which Experiment” dialog.

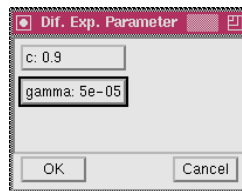


Figure 20: “Dif.Exp.Parameter” dialog.

The toggle button **Difference graph** determines whether the resulting graph a difference graph is or not.

The choices in the **Diff. after** menu:

- Param.** By clicking at one of the buttons marked with the parameter name the “Difference Values” dialog will appear (see Fig. 17). Select one value from the

list labeled **A**, and one or more values from the list labeled **B**. The toggle buttons **A-B** and **B-A** determine the subtrahend and minuend. The result lines are created as the difference between any two lines with the same name.

- Rewards** Choose exactly two rewards which the difference line is made from.
- Experiments** By clicking at this button the “Which Experiment” dialog (see Fig. 19) with a list of all possible experiments for this model appears. Choose one of the experiments and then press OK. If the selected experiment contains more than one varying parameter the “Dif.Exp.Parameter” dialog (see Fig. 20) pops up. By clicking at one of the buttons marked with the parameter name select a fixed value for each parameter. The result lines are created as the difference between any two lines with the same name. The subtrahend lines belong to the current experiment, the minuend lines belong to the experiment selected in the “Which Experiment” dialog.

The **Rewards** menu specifies if the reward axis has nominal scale (**Nominal**) or normalized (**Time average**).

The choices in the **Which** menu:

- all** The runs of all reward/strategy lines are displayed.
- part** Choose rewards/strategies from the “Which” dialog (see Fig. 17) theirs run should be displayed.

The choices in the **Experiment** menu:

- one** The results of one experiment (i.e. the current experiment) should be displayed.
- more** Choose one or more additional experiments from the “Experiments” dialog; the reward or strategy functions of the additional experiments will be displayed, together with the reward or strategy functions of the current experiment.

The toggle button **Aggregated Rewards** determines whether the rewards selected in the “Which” menu are aggregated or not.

Meaning of the buttons:

- OK** Displays the graph (see Fig. 21).
- Forward** Runs the experiment in another interval (see Sec. 2.9.1).
- Change** The user can change explicitly the result lines of a strategy graph (see Sec. 2.9.2).
- Cancel** Leaves the graph parameters dialog.

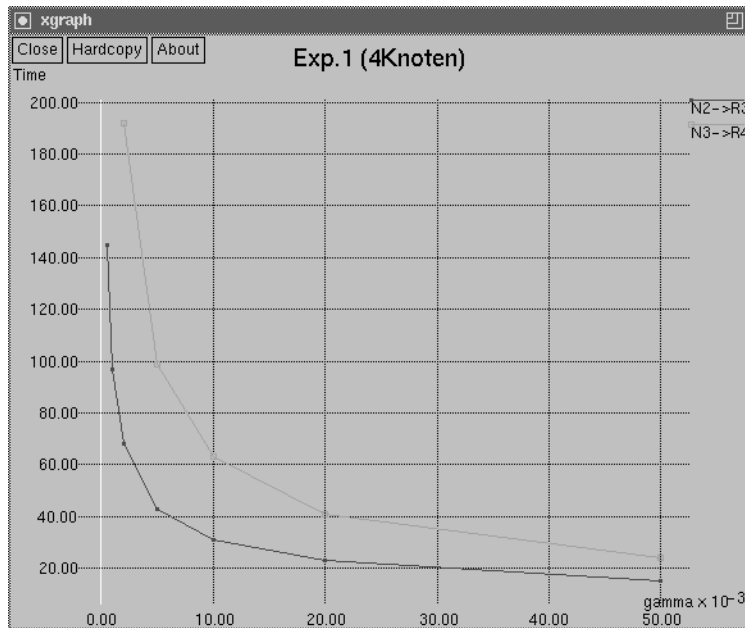


Figure 21: A plot program displaying a strategy.

In the graph, strategies are labeled as Node1->Node2, which means that the reconfiguration edge starting in Node1 and ending in Node2 is active.

A node N that appears in a macro named M is designated M/N. Node names such as M/M/M/N will automatically be abbreviated to M<sup>3</sup>/N.

If one of the varying parameters has a variable mode the line labels in the graph contain the value of this parameter in parentheses.

If the “Experiments” menu is set to “more” the line labels in the graph contain a number in parentheses that determines which experiment the lines belong to. The labels without such number determine lines from the current experiment.

### 2.9.1 The Forward Experiment Dialog

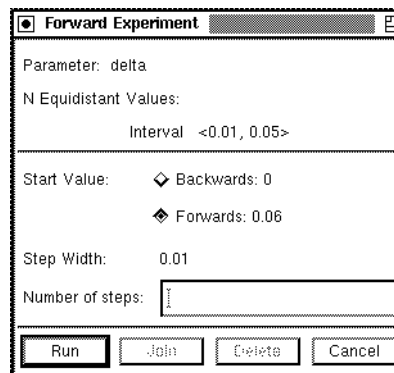


Figure 22: Forward Experiment dialog for equidistant mode.

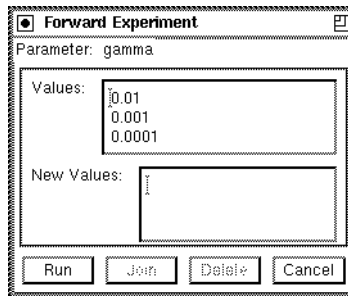


Figure 23: Forward Experiment dialog for multiple mode.

There are three different forward experiment dialogs depending on the parameter mode (equidistant, log. equidistant or multiple values) of the parameter selected in one of the axis lists.

**Equidistant Mode** The “Forward Experiment” dialog appears (see Fig. 22). The first three lines give an information about the selected parameter (name, mode, and interval of values) where the experiment had already run. Active one of the following radio buttons:

**Backwards:** *value* The parameter values are taken from the set

$$\{x|x = a - s \cdot i, i = 0 \dots N - 1\}$$

where  $a$  is the **Start Value**,  $s$  is the **Step Width** and  $N$  **Number of Steps**. The *Value* is the first possible parameter value ( $i=0$ ). If  $value < 0$  the radio button is made invisible. The value in field **Number of Steps** is accepted if  $N > 1$  and  $x \geq 0$  for  $i=N - 1$ .

**Forwards:** *value* The *value* is the start value for the new interval. The value in field **Number of Steps** is accepted if  $N > 1$ .

**Logarithmic Equidistant Mode** The “Forward Experiment” dialog appears. The informations in the first three lines is equal to the Forward Experiment dialog in the equidistant mode. If the “Backwards” radio button is selected the parameter values are taken from the set

$$\{x|x = a/f^i, i = 0 \dots N - 1\}$$

where  $a$  is the **Start Value**,  $f$  is the **Factor** and  $N$  is the **Number of Steps**. The value in field **Number of Steps** is accepted if  $N > 1$ .

**Multiple Mode** The “Forward Experiment” dialog appears (see Fig. 23). There are two text widgets. In the upper one, there are values which the experiment had already run for. This text widget is not editable. In the second one, new values may be inserted. At least two values have to be inserted. Each number must be in a separate line.

Press the button **Run** to run the experiment for the new values. Then the “Forward Experiment” and “Graph Parameters” dialog disappears and a “Working...” dialog (see Fig. 12) is shown. After the experiment is performed the new results can be seen in the “Graph Parameters” dialog

(see Sec. 2.9). Now, if the button “Forward” is pressed the new results can be joined with the previous by clicking at the button **Join** in the “Forward Experiment” dialog, or they can be deleted by clicking at the button **Delete**. To exit the “Forward Experiment” dialog the button **Cancel** must be pressed.

## 2.9.2 Change Graph Dialog

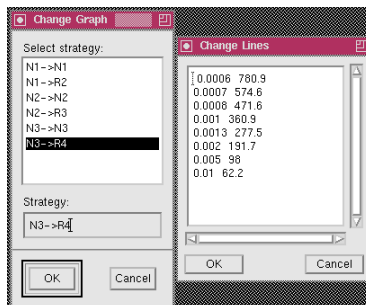


Figure 24: “Change Graph” and “Change Lines” dialog.

The user has a possibility to change interactively the course of strategy lines in the result graph. The “Change Graph” dialog (see Fig. 24) is a list of all strategies in the experiment. Select a strategy its line should be changed. By clicking at the button **OK**, the “Change Lines” dialog (see Fig. 24) appears.

Press the button **Cancel** to leave the change graph dialog.

**Change Lines Dialog** The Change Lines Dialog contains an editable text widget. Each curve in the graph is represented by a co-ordinate set in the text widget. The user may change this co-ordinates. Each line must be terminated with new line character. Between every two co-ordinate sets is an empty line.

Press the button **OK** to change the graph. To leave the Change Lines dialog without any change is done click at the button **Cancel**.

By clicking at the button Save in the Experiment Editor all changes are saved.

## 2.10 Resources

Some attributes of XPenelope can be customized by the user. In order to do this, entries must be made in the file

```
$HOME/.Xdefaults
```

and subsequently the X server must be restarted or the command



```
xrdb -load $HOME/.Xdefaults
```

must be executed.

Here is a list of the customizable resources, their default values, and their meaning:

```
XPen.geometry: +100+100
```

Position of the model index after invocation.

```
XPen.background: Gray90
```

Background color.

```
XPen.foreground: Black
```

Foreground color.

```
XPen.printCommand: lpr -P%s
```

Command to print a file; %s is a placeholder for the file name and **must** appear in the resource string.

```
XPen.PrinterName.value: psr0
```

Default printer name.

```
XPen.plotDrivers: \  
XGraph,xgraph.driver,LB \n\  
Text,text.driver,XY
```

Available plot drivers.

The `plotDrivers` resource contains an arbitrary number of driver specifications. Each specification is separated by a newline character. A specification has the general format

*Title , Driver , Capabilities*

where *Title* is the text that appears in the **Driver** menu of the graph parameters dialog. *Driver* is the name of the program which will be called to display the data. *Capabilities* is a string which contains any of the following characters:

- L** The driver is able to display logarithmically scaled axes.
- B** The driver is able to display barcharts.

**X** The driver is able to display alphanumerical strings as tick labels on the X axis.

**Y** The driver is able to display alphanumerical strings as tick labels on the Y axis.

Several arguments are passed to the driver program:

**\$1** Name of input file.

**\$2** Title.

**\$3** Name of X-axis.

**\$4** Name of Y-axis.

Additionally, the following options may be passed to the driver:

**-xalpha** Alphanumerical tick labels on the X axis.

**-yalpha** Alphanumerical tick labels on the Y axis.

**-logx** Logarithmically scaled X axis.

**-logy** Logarithmically scaled Y axis.

**-bar** Display bar chart instead of graph.

The input file contains XY-pairs, one for each line. The X and Y values are separated by one space character. The XY-pairs are grouped by their Z value; a line indicating the Z-value precedes the corresponding XY-pairs; this line starts with the character “:”.

Example:

```
:N1->N1
0.1 10
0.2 30
0.4 60
:N1->N2
0.1 15
0.2 40
0.4 80
```

This file stands for the XYZ-pairs (0.1,10,N1->N1), (0.2,30,N1->N1), ..., (0.4,80,N1->N2).

### 3 Examples

This section contains two step-by-step examples which show how to create and use models and macros with XPenelope.

#### 3.1 Example 1

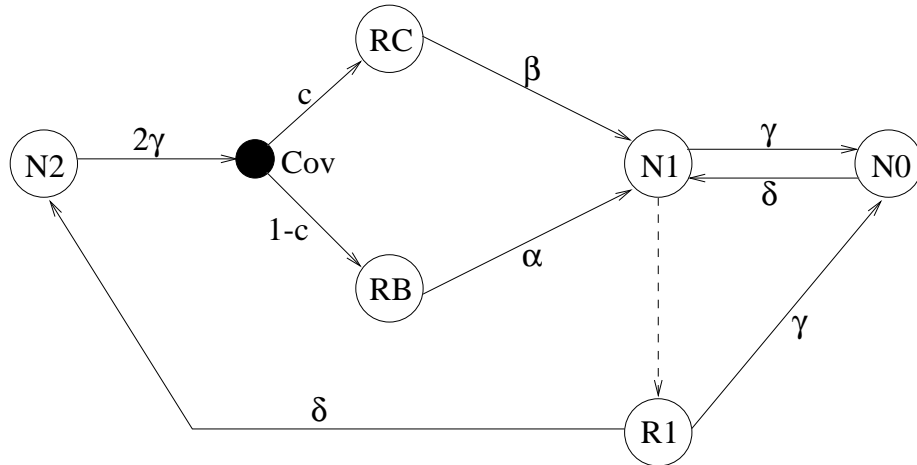


Figure 25: A simple model (from [DEME 92, p. 52])

The first example shows how to create the model depicted in figure 25. The following reward rates shall be assigned to the states:

State	Reward rate
N0	0
N1	1
R1	1
N2	1
RB	0
RC	0
Cov	0

*Phase 1: Create the states*

**Step 1:** Invoke XPenelope by entering the command

```
xpenelope
```

on the command line. After a few seconds, the model index should appear on the screen.

**Step 2:** Press the **Create** button in order to create a new model.

- Step 3:** Select the item **Markov State** from the **Edit** menu. This activates the *Markov state mode*; in this mode, every mouse click in the drawing area produces a new Markov state.
- Step 4:** For every Markov state in figure 25, click the left mouse button at the corresponding position in the drawing area. For every mouse click, a circle with a question mark appears on the screen.
- Step 5:** Select the item **Branch State** from the **Edit** menu. This activates the *Branch state mode*; in this mode, every mouse click in the drawing area produces a new branching state.
- Step 6:** Make one mouse click in the drawing area in order to create the state **Cov**. The drawing area should now look like figure 26.

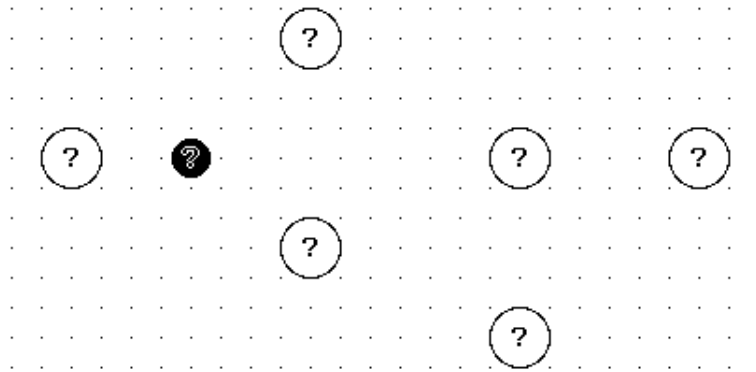


Figure 26: Model editor containing six Markov states and one branching state

*Phase 2: Assign names and rewards to the states*

- Step 7:** Select the item **Set Parameters** from the **Edit** menu. This activates the *Set parameters mode*; in this mode you may click at any object in order to modify its parameters.
- Step 8:** The question marks displayed on top of each state signal that the parameters for the states have not yet been entered. Thus, click at the state **N2** (the leftmost state). The “Parameters...” dialog appears. Enter the node name “N2” in the field **Name**; enter the reward rate “1” in the field **Reward**. Press the **OK** button.
- Step 9:** Repeat step 8 for each state.

*Phase 3: Create the transitions and reconfiguration edges*

- Step 10:** Select the item **Transition** from the **Edit** menu. This activates the *Transition mode*; in this mode, you may click at two states in order to create a transition between these states.

- Step 11:** First, we will create the transition between state **N2** and state **Cov**: click at state **N2**, then at state **Cov**. A transition with a question mark on top of it appears.
- Step 12:** Repeat step 11 for each transition. If you create the transition from **R1** to **N2**, you probably don't want to connect the states by a straight line. Therefore, you may proceed like this: click at state **R1**, then click somewhere between state **R1** and **N2**, and finally click at state **N2**. This will produce a transition consisting of two line segments.
- Step 13:** Select the item **Reconfiguration Edge** from the **Edit** menu. This activates the *Reconfiguration edge mode*; in this mode, you may click at two states in order to create a reconfiguration edge between these states.
- Step 14:** The example model contains only the reconfiguration edge between state **N1** and **R1**. Thus, first click at state **N1** and then at state **R1**. A reconfiguration edge between the two states appears. The drawing area should now look like figure 27.

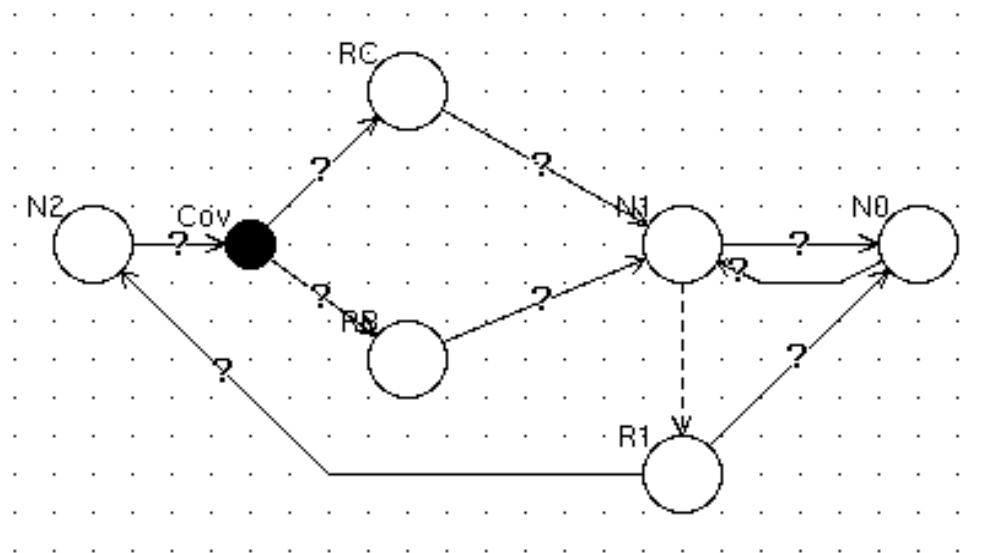


Figure 27: States and transitions.

*Phase 4: Assign transition rates and probabilities to the edges*

- Step 15:** Select the item **Set Parameters** from the **Edit** menu.
- Step 16:** Click at the transition between state **N2** and state **Cov**. The “Parameters...” dialog appears.
- Step 17:** Enter the transition rate “2 \* gamma” in the field **Rate**. Press the **OK** button.
- Step 18:** Repeat step 17 for each transition. If you click at a transition originating in a branching state, the “Parameters...” dialog will contain a field **Probability**. In this field you have to enter the probability of the transition (“c” or “1-c” in our example). The drawing area should now look like figure 28.

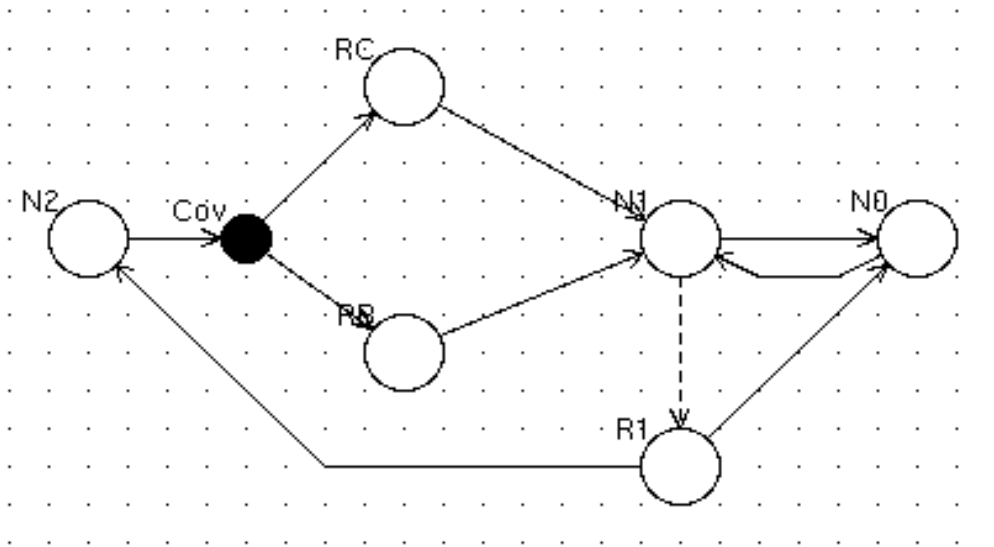


Figure 28: The completed model.

*Phase 5: Save the model*

**Step 19:** Select the item **Save** from the **File** menu. A dialog appears which prompts you for a model name. Enter “SimpleExample” and press the **OK** button.

**Step 20:** A message dialog with the text “Model saved” appears. Press the **OK** button.

*Phase 6: Create a parameter set*

We will now assign concrete values to the individual model parameters. The values we will choose are:

Model parameter	Value(s)
alpha	0.01
beta	0.1
c	0.99
delta	0.01
gamma	10 <sup>-5</sup> , 10 <sup>-4</sup> , 10 <sup>-3</sup> , 10 <sup>-2</sup> , 10 <sup>-1</sup>

**Step 21:** Select the item “SimpleExample” in the model editor and press the **Open** button. A pop-up menu appears; select the item **Parameter Index...**. The parameter index appears.

**Step 22:** Press the **Create** button of the parameter index in order to create a new parameter set. The parameter set editor shown in figure 29 appears.

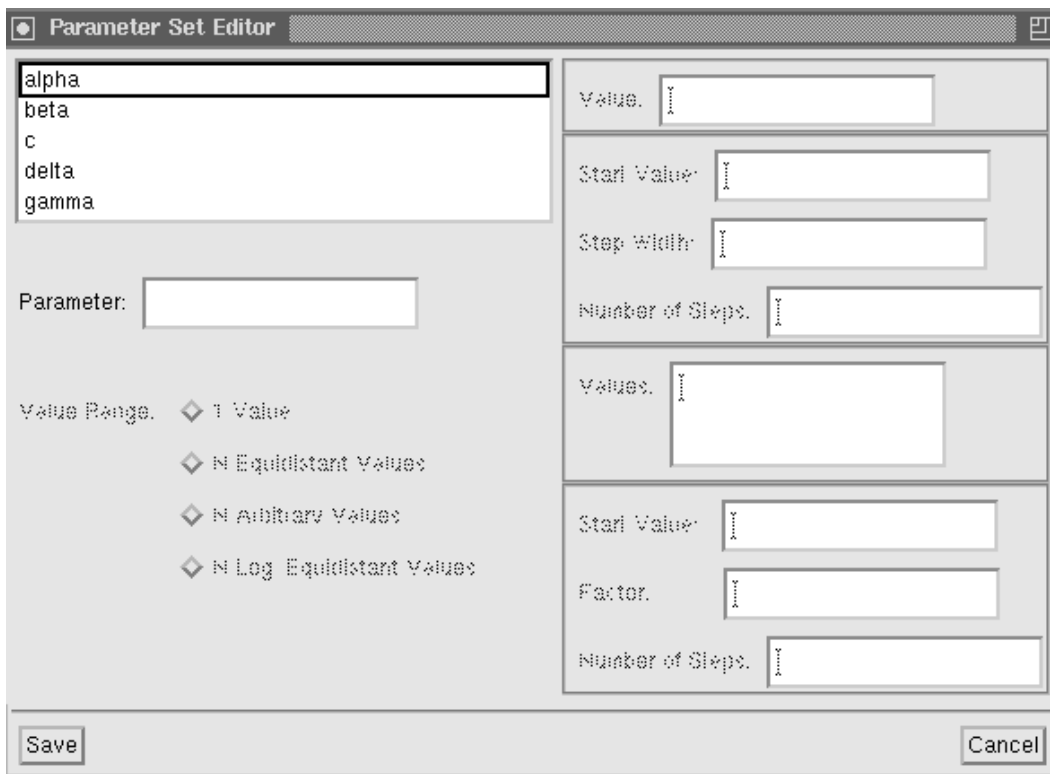


Figure 29: The parameter set editor.

- Step 23:** Click at the item “alpha” and enter the value “0.01” in the field **Value**.
- Step 24:** Repeat step 23 for the parameters “beta”, “c” and “delta”.
- Step 25:** Select the item “gamma” and press the button **N Arbitrary Values**. Now enter the values “1e-5”, “1e-4”, “1e-3”, “1e-2” and “1e-1” in the field **Values**, one value per line.
- Step 26:** Press the **Save** button.
- Step 27:** A dialog appears prompting you for the “Parameter Set Name”. Enter “PSet” and press the **OK** button. A message dialog with the text “Parameter Set Saved” appears; press the **OK** button.

*Phase 7: Create the experiment definition*

- Step 28:** Select the item “SimpleExample” in the model editor and press the **Open** button. A pop-up menu appears; select the item **Experiment Index...** The experiment index appears.
- Step 29:** Press the **Create** button of the experiment index in order to create a new experiment definition.
- Step 30:** A dialog appears prompting you for the “Experiment Name”. Enter “Exp” and press the **OK** button. The experiment editor appears.

- Step 31:** Press the button **Parameter Set**, which currently has the label **???**. The “Parameter Sets” dialog appears, which shows a list of the available parameter sets. Currently only the parameter set “PSet” is available. Select this parameter set and press the **OK** button. The label of the **Parameter Set** button has changed to “PSet” to reflect the currently selected parameter set.
- Step 32:** Enter the value “500” in the field **Mission Time**.
- Step 33:** Enter the value “0.1” in the field **Step Width**.
- Step 34:** Make sure that the radio button labeled **Transient Optimization** is active.

*Phase 8: Start the experiment*

- Step 35:** Press the **Run** button. The “Working...” dialog appears. Wait until this dialog disappears from the screen. Then the experiment has finished.
- Step 36:** Press the **Save** button. A message dialog with the text “Experiment Saved” appears; press the **OK** button.

*Phase 9: Show the results*

- Step 37:** Press the **Show** button. The “Graph Parameters” dialog appears.
- Step 38:** Make sure that the program “XGraph” has been selected in the **Driver** menu.
- Step 39:** Select the item **Log/Lin** from the **Scale** menu in order to scale the X-axis logarithmically.
- Step 40:** Press the **OK** button. After a few seconds, the graph shown in figure 30 appears.
- Step 41:** Interpret the results: The curve labeled “**N1**→**R1**” indicates that above this curve the reconfiguration edge from state **N1** to state **R1** is active. E.g. for  $\gamma = 0.001$  the reconfiguration edge is active from the start of the mission until 41.8 time units before the end of the mission.
- Step 42:** Remove the graph window by pressing the **Close** button.
- Step 43:** You may now modify the settings of the “Graph Parameters” dialog in order to modify the appearance of the graph. For example, you might select “<Time>” from the **X-Axis** list and “<Strategy>” from the **Y-Axis** list. If you now press the **OK** button, the graph appears with the X- and Y-axes swapped.
- Step 44:** Click at the radio button **Performance Graph** in the “Graph Parameters” dialog.
- Step 45:** You may select one or more performance curves by selecting the “part” item in the **Which** menu. The “Which” dialog appears. Select one or more items in the list of model states and press **OK**.
- Step 46:** Press the **OK** button in the “Graph Parameters” dialog. A performance graph for the selecting states appears.



Figure 31: Model of the Erlang- $k$  distribution

The second example shows how to create and use macros. In this example a macro which models the Erlang- $k$  distribution shall be created.

Figure 31 shows the model of the Erlang- $k$  distribution: it consists of a chain of  $k$  Markov states with a transition rate of  $k \cdot \mu$ . Erlang- $k$  distributions are generally used to model non-exponential distributions with a small variance.

An Erlang- $k$  distribution can be thought of as one Markov state which is connected to an Erlang- $k-1$  distribution (see figure 32). This means that a model for the Erlang- $k$  distribution can be built using a recursive macro.

The macro for the Erlang- $k$  distribution is depicted in figure 33. The macro contains a parameter  $N$ , which is incremented by one with each recursion step. The macro invokes itself if  $N < k$ ;

Figure 33: Macro for the Erlang- $k$  distribution

otherwise the macro *Connect* is called. *Connect* is a macro which simply connects the input with the output. The branching state *Com* is needed to connect the outputs of the two macros with *Out*, because it isn't allowed to connect more than one edge to a macro terminator.

The following paragraphs show how to create the Erlang macro.

*Phase 1: Create the Connect macro*

- Step 1:** Invoke XPenelope and press the **Create** button of the model index.
- Step 2:** Select the item **Macro Terminator** from the **Edit** menu. This activates the *Macro terminator mode*; in this mode, every mouse click in the drawing area produces a new macro terminator.
- Step 3:** Click the left mouse button twice in the drawing area in order to produce the terminators *In* and *Out*.
- Step 4:** Select the item **Set Parameters** from the **Edit** menu.

- Step 5:** Click at the first macro terminator. The “Set Parameters...” dialog appears. Enter “In” in the field **Name** and press the **OK** button.
- Step 6:** Repeat step 5 for the second macro terminator (name: “Out”).
- Step 7:** Select the item **Transition** from the **Edit** menu.
- Step 8:** Click at the terminator *In* and then at the terminator *Out*. A transition between the two terminators will appear (see figure 34).

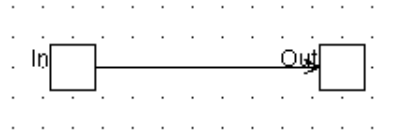


Figure 34: The *Connect* macro

- Step 9:** Select the item **Save** from the **File** menu and save the macro under the name “Connect”.

*Phase 2: Create the nodes of the Erlang macro*

- Step 10:** Create the terminators *In* and *Out* of the Erlang macro as shown in steps 2 to 5.
- Step 11:** Create the Markov state *A* and the branching state *Com* as shown in example 1.
- Step 12:** Select the item **Macro** from the **Edit** menu. This activates the *Macro mode*; in this mode, every mouse click in the drawing area produces a new macro invocation.
- Step 13:** The “Macros” dialog box appears. Select the item “Connect” from the list and press the **OK** button.
- Step 14:** Click the left mouse button while the cursor is in the drawing area. An invocation of the *Connect* macro will appear.

*Phase 3: Create a forward declaration for the Erlang macro*

At this point, we have the problem that we need to use the Erlang macro. This macro, however, has not yet been defined. Thus, we have to create a forward declaration, which informs the model editor about the names of the macro sockets and the names of the macro parameters. The parameters of the Erlang macro are:

Parameter	Meaning
k	Number of phases
mu	Transition rate
N	Recursion depth
rew	Reward rate

- Step 15:** Select the item **Macro** from the **Edit** menu. The “Macros” dialog box appears.
- Step 16:** Press the **Forward...** button. The “Forward Declaration” dialog appears.
- Step 17:** Enter “Erlang” into the **Class** field.
- Step 18:** Click at the **In** button and enter “In” in the **Socket name** field. Then press the **Insert** button. This inserts the input socket “In” into the **Socket names** list.
- Step 19:** Click at the **Out** button and enter “Out” in the **Socket name** field. Then press the **Insert** button. This inserts the output socket “Out” into the **Socket names** list.
- Step 20:** Enter “k” in the **Parameter name** field and press the **Insert** button. This inserts the parameter “k” in into the **Parameters** list.
- Step 21:** Repeat step 20 for the parameters “mu”, “N” and “rew”.
- Step 22:** Press the **OK** button.
- Step 23:** Click the left mouse button while the cursor is in the drawing area. An invocation of the *Erlang* macro will appear.

*Phase 4: Edit the macro parameters*

- Step 24:** Select the item **Set Parameters** from the **Edit** menu.
- Step 25:** Click at the invocation of the *Connect* macro. The “Macro” dialog appears.
- Step 26:** Enter the node name “C” in the **Name** field.
- Step 27:** Enter the condition “N=k” in the **Condition** field. The meaning of this condition is that this macro must not be invoked if  $k$  and  $N$  are not equal.
- Step 28:** Press the **OK** button.
- Step 29:** Click at the invocation of the *Erlang* macro. The “Macro” dialog appears.
- Step 30:** Enter the node name “E” in the **Name** field.
- Step 31:** Enter the condition “N<k” in the **Condition** field.
- Step 32:** Change the content of the field **N** from “N” to “N+1”. This means that  $N$  is incremented with each recursion step.
- Step 33:** Press the **OK** button.

*Phase 5: Complete the macro definition*

- Step 34:** Select the item **Transition** from the **Edit** menu.
- Step 35:** Create all necessary transitions as shown in figure 33.

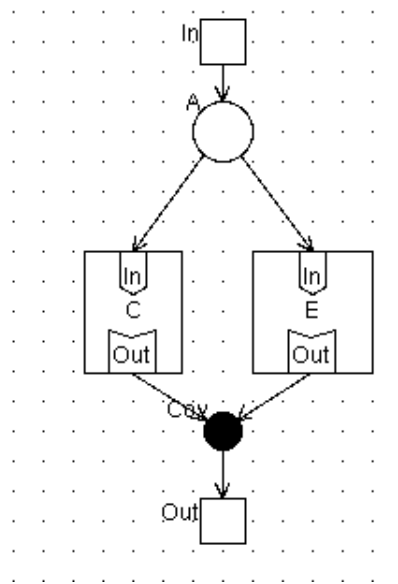


Figure 35: The *Erlang* macro

**Step 36:** Save the macro under the name “Erlang”. It is important that this name is the same as the name that has been specified in the forward declaration dialog. The completed macro definition should look like figure 35.

*Phase 6: Use the Erlang macro*

- Step 37:** Select the item **Macro** from the **Edit** menu. The “Macros” dialog appears.
- Step 38:** Select the item “Erlang” from the list and press the **OK** button.
- Step 39:** Click the left mouse button while the cursor is in the drawing area. An invocation of the *Erlang* macro appears.
- Step 40:** Select the item **Set Parameters** from the **Edit** menu.
- Step 41:** Click at the macro invocation. The “Macro” dialog appears.
- Step 42:** Enter a node name.
- Step 43:** Enter the value “1” in the field **N** in order to initialize the recursion depth counter.
- Step 44:** Modify the parameters “mu”, “rew” and “k” to suit your application.
- Step 45:** Build your model around this macro invocation. An example is shown in figure 36.

## 4 Conclusion

The combination of the tools Penelope and XPenelope enables a fast and comfortable creation and optimization of extended Markov reward models.

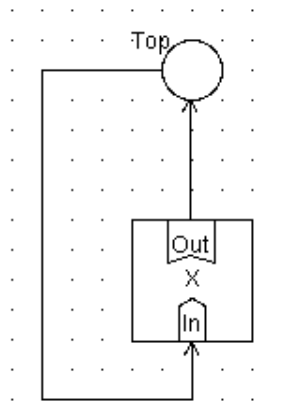


Figure 36: A simple model with an invocation of the *Erlang* macro

Penelope contains algorithms for the analysis and optimization of statically and dynamically reconfigurable systems. XPenelope supports the model creation, the experiment management and the visualization of the results. The macro mechanism of XPenelope makes it possible to split large models into smaller and simpler submodels.

The user-friendly X window interface, the operating system independent database, permanently executed consistency checks, and many other features make XPenelope a versatile working environment for the modeling and optimization of reconfigurable systems. Many application examples can be found in [DEME 94].

## References

- [KÄTK 92] Kätker, S.: “Entwicklung und Implementation ausgewählter Algorithmen der stochastischen Optimierung zur Erweiterung von PENELOPE”; Master thesis, University of Erlangen-Nürnberg, 1992.
- [MAUS 90] Mauser, H.: “Implementierung eines Optimierungsverfahrens für rekonfigurierbare Systeme”; Diplomarbeit, University of Erlangen-Nürnberg, 1990.
- [DEME 92] de Meer, H.: “Transiente Leistungsbewertung und Optimierung rekonfigurierbarer fehlertoleranter Rechensysteme”; *Arbeitsberichte des IMMD der Universität Erlangen-Nürnberg*, Vol. 25, No. 10, October 1992.
- [DEME 94] de Meer, H., Kishor S. Trivedi, and Mario Dal Cin: “Guarded Repair of Dependable Systems”; *Theoretical Computer Science, Special Issue on Dependable Parallel Computing*, Vol. 129, July 1994 ([http://www.informatik.uni-hamburg.de/TKRN/ro\\_homed.htm](http://www.informatik.uni-hamburg.de/TKRN/ro_homed.htm)).
- [MEMA 92] de Meer, H.; Mauser, H.: “A Modeling Approach for Dynamically Reconfigurable Systems”; *Proceedings of the 2nd Int. Workshop on Responsive Computer Systems*, Kamifukuaoka, Saitama, Japan, 1992.
- [MEIT 92] Meitinger M.: “Entwurf und Implementierung eines Programms zur graphischen Aufbereitung von Analyseergebnissen unter X-Windows”; Master thesis, University of Erlangen-Nürnberg, 1992.
- [MUNK 93] Munkert, F.: “Entwicklung und Implementierung einer benutzerfreundlichen X-Window-Oberfläche für das Leistungsbewerte- und Optimierungstool PENELOPE”; Master thesis, University of Erlangen-Nürnberg, 1993.
- [TIJM 86] Tijms, Henk C.: “Stochastic modelling and analysis : a computational approach”; Chichester : Wiley, 1986.
- [TURN 92] Turner, P.J.: “ACE/gr User’s Manual”; Oregon Graduate Institute of Science and Technology, Beaverton, Oregon, 1992.