

Contents

Introduction.....	1
User Interface.....	2
2.1 Some general information about DEEM.....	2
2.2 File Menu.....	3
2.3 Edit Menu	3
2.3.1 Edit commands.....	4
2.3.2 Property Window.....	4
2.4 Insert Menu	8
2.5 Special Menu.....	8
2.6 Zoom Menu	9
2.7 Compute Menu	9
2.7.1 Parameters.....	9
2.7.2 Measures.....	10
2.7.3 Transient Analysis.....	12
2.8 ? Menu.....	13
How to create and solve a Model: an example.....	14
3.1 A working example	14
3.2 Creation of the graphical Model	14
3.3 Properties of places, transitions and arcs.....	15
3.4 Values of the parameters and studies.....	17
3.5 Definition of the measures.....	19
3.6 Transient analysis	20
Results and output files.....	21
4.1 The PhN files	21
4.2 The files created for each phase.....	24
4.3 The files created for each study.....	25
Transient Solver.....	27
5.1 The analytical technique	27
5.2 The solution algorithm.....	27
Appendix.....	29
Installation	29
Tool Organization and File Structure.....	29
Bibliography.....	30

Introduction

Many systems devoted to control and management of critical activities have to perform a series of tasks that must be accomplished in sequence. Their operational life consists of a sequence of non-overlapping periods, called *phases*. These systems are often called Multiple-Phased Systems (MPS). They include several classes of systems that have been object of active research during the last decades, such as those known as Phased Mission Systems (PMS) and Scheduled Maintenance Systems (SMS). MPS are very general, since their phases can be distinguished along a wide variety of differentiating features.

- During a specific phase, an MPS is devoted to the execution of a particular set of tasks, which may be different from the activities performed within other phases.
- The performance and dependability requirements of an MPS can be completely different from one phase to another.
- During some phases the system may be subject to a particularly stressing environment, thus experiencing dramatic increases in the failure rate of its components.
- In order to accomplish its mission, a MPS may need to change its configuration over time, to adopt the most suitable one with respect to the performance and dependability requirements of the phase being currently executed, or simply to be more resilient to a hazardous external environment.
- The successful completion of a phase, as well as the activities performed therein, may bring a different benefit to the MPS with respect to that obtained with other phases.

Many examples of MPS can be found in various application domains. Representative examples are systems for the aided-guide of aircraft, whose mission-time is divided into several phases such as take-off, cruise, landing, with completely different requirements. A very important sub-class of MPS is represented by the so-called Scheduled Maintenance Systems encountered in almost all the application domains where an artefact is to be used for long time and is periodically subject to maintenance actions. An SMS is easily formulated as a MPS considering that the system is run for a number of operational phases, and then undergoes a maintenance phase.

DEEM (DEpendability Evaluation of Multiple-phased system) is a dependability modeling and evaluation tool specifically tailored for MPS. This tool supports the methodology proposed in [5, 6] for the dependability modeling tool and evaluation of MPS. This methodology relies upon Deterministic and Stochastic Petri Nets (DSPN) as a modeling tool and on Markov Regenerative Processes (MRGP) for the model solution. Due to their high expressiveness, DSPN models are able to cope with dynamic structure of MPS, and allow defining very concise models. DEEM models are solved with a very simple and computationally efficient analytical solution technique based on the separation of the MRGP underlying the DSPN of a MPS.

This manual is organized as follows. Chapter 2 describes the user interface, defining all menus with their functionalities. Chapter 3 shows how to create a model in DEEM and how to solve it. Chapter 4 specifies the files the results of the analysis are stored in, and it describes how to interpret those files. Chapter 5 describes the specialized solution algorithm implemented by DEEM. In Appendix all the steps to install DEEM are shown.

User Interface

This chapter is a reference guide for the various options provided by the Graphical User Interface of DEEM.

2.1 Some general information about DEEM

DEEM possesses a GUI inspired by [2] and realized using an X11 installation with Motif runtime Libraries which the user employs to define his model of a MPS. DEEM provides two logically separate parts to represent MPS models. One is the System Net (SN), which represents the failure/repair behaviour of system components, the other is the Phase Net (PhN), which represents the execution of the various phases. For this reason, the working area is split in two fields as shown in Figure 2.1.

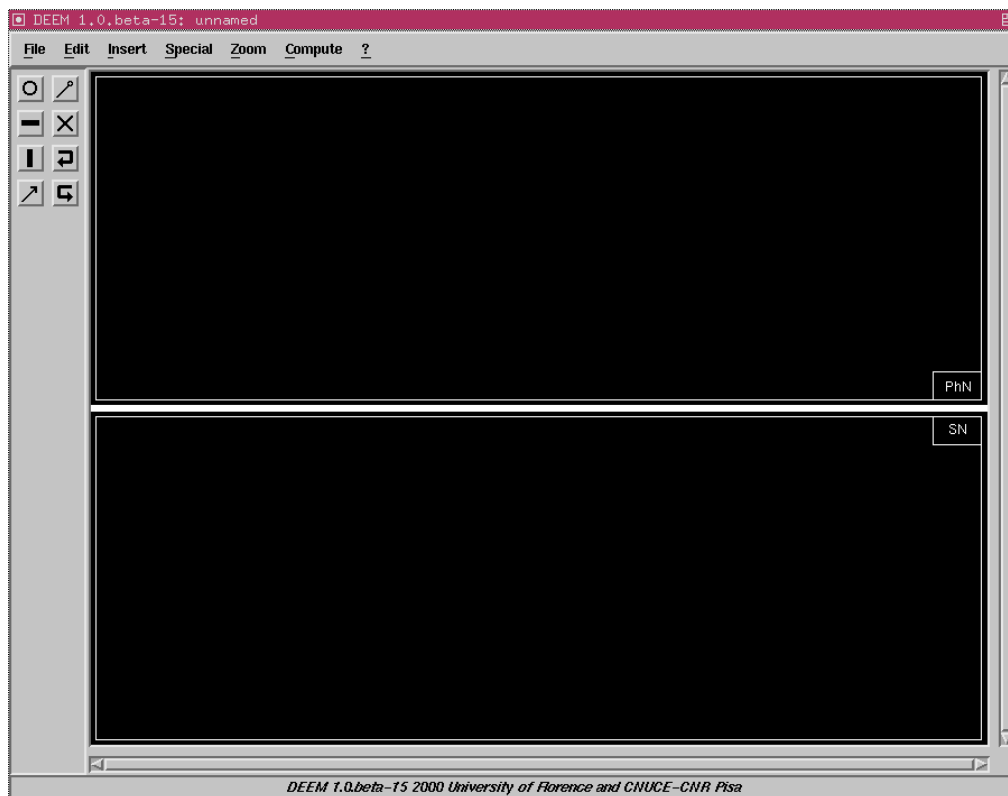


Figure 2.1: DEEM interface

The SN area may contain only exponentially distributed and immediate transitions, whereas in the PhN area the transitions may be deterministic or immediate. In a PhN model a token in a place represents the execution of a phase, and the firing of a deterministic transition the beginning of the next phase. In general, the execution of a phase is represented by each marking enabling only one deterministic transition.

DEEM employs the DSPN formalism [1] to model a MPS. DSPN extends Generalised Stochastic Petri Nets and Stochastic Reward Nets, so to allow the exact modelling of events with deterministic occurrence times.

According to this, the GUI uses the following objects to create a model:

- Places
- Transitions
- Arcs

Besides the introduction of deterministic transitions, DEEM makes available a set of modeling features that significantly improve DSPN expressiveness:

- the firing rates of timed transitions may be specified through arbitrary marking-dependent functions;
- these functions may be employed to include additional enabling conditions (*guards*) to the specification of the transitions;
- rewards rates can be defined as arbitrary marking-dependent functions;
- arcs cardinalities may depend on the marking of the model.

To draw a correct PhN it is necessary that the reachability graph associated with the PhN be a tree or an acyclic graph. Even if the behaviour of the system is represented by a loop (after n phases the some phases are repeated), that is the only condition to draw a correct PhN.

2.2 File Menu

The File Menu is used to create a new model, to open or close a model, to save the created or modified model, to assign a name to the net and to quit DEEM.

File->New (or Ctrl-N) creates a new model. If a model is already opened, it is closed (a confirmation box is shown) and a new net can be drawn.

File->Open (or Ctrl-L) opens a model from the directory *deem_models*.

File->Save (or Ctrl-S) saves in the current directory the opened model. The default name for new models is "unnamed".

File->Save As... With this option, a new name can be assigned to a model. The file is written in the directory *deem_models*. If a file with the same name exists in *deem_models*, a warning message (! *File already exists! Overwrite it?*) appears in a window, and the choice to overwrite the file is given.

File->Close closes the drawn model.


File->Save Options permits an automatic backup by making on automatic save every k steps (where k is a number).


File->Exit (or Ctrl-X) quits DEEM.

2.3 Edit Menu

The Edit Menu is used to undo or redo some actions, to refresh the window, to delete, cut, copy, paste one or more objects, to define the properties of places, transitions and arcs.


2.3.1 Edit commands

Edit->undo: (or Ctrl-U) removes, one by one, all the modifications to the model made since the last saved model. The same command can be executed by clicking the DEEM_button¹ .

Edit->redo: restores the modifications removed by Edit->undo. The same command can be executed by clicking DEEM_button .

Edit->Refresh (or Ctrl-R): refreshes the DEEM window.

To select an object click with the left button on it, and the following commands became available:

Edit->Delete (or Ctrl-D): removes the selected object. The same command can be executed by clicking the DEEM_button .

Edit->Copy Node (or Ctrl-E): copies the selected place.

Edit->Copy, Edit->Cut and **Edit->Paste:** to copy, cut and past the selected part of the net.

Edit->properties (or Ctrl-O): open the *property window* associated with the selected element. The property window of an object can also be opened by clicking with the right button on it.

2.3.2 Property Window

A property window is associated with each element (place, transition or arc).



The image shows a dialog box titled 'Property window associated with the place'. It has a light gray background and a dotted border. Inside, there are three rows of labels and text boxes: 'Place Name:' followed by a text box containing 'Place_1'; '# Tokens:' followed by a text box containing '1'; and 'Capacity:' followed by a text box containing 'no limit'. At the bottom of the dialog, there are two buttons: 'Apply' on the left and 'Cancel' on the right.

Figure 2.2: Property window associated with the place

The property window associated with a place (shown in Figure 2.2) contains the following fields:

- Ø **Place Name:** specifies the name of the place;
- Ø **# Token:** defines the number of tokens that the place contains at the beginning of the computation;

¹ N.B.: in the following we will call *DEEM_buttons* all the buttons shown by the DEEM interface in the working window.

- ∅ **Capacity:** specifies the maximum number of tokens that a place can have during the computation.

Figure 2.3: Property window associated with the transition

The property window associated with the transitions (shown in Figure 2.3) contains the following fields:

- ∅ **Transition Name:** specifies the name of the transition;
- ∅ **Orientation:** specifies the horizontal or vertical orientation of the transition;
- ∅ **Transition Type:** specifies the type of the transition. Transitions can be immediate, (represented by a thin line), or timed, in this case they may have exponentially distributed firing times, and they are represented by empty rectangles, or deterministic firing times (only for PhN), represented by filled rectangles. When a type is chosen, a new window appears to define in more details the transition. If the immediate type is chosen, the new window allows to specify the *probability* and *priority* associated with it. When a timed transition is chosen instead the new window allows to define the firing rate of the transition. The values can be constants or variables. To define a variable, the following syntax has to be respected: $VAR(variable_name)$;
- ∅ **Rate Function:** defines the firing rate of the transitions and can be an arbitrary marking-dependent function;
- ∅ **Enabling Function:** defines the enabling condition, (or guard), of the transition. It can be an arbitrary marking-dependent function;
- ∅ **Copy from list to:** adds the place selected from the list of all the places of the net in the “Rate Function” field or in the “Enabling Function” field by clicking the *copy* button.

In this window the **Help** button shows the syntax of the *probability*, *rate* and *enabling function*.

The probability and rate function syntax is:

```

func -> '(' func ')'
      | arith_op1 func
      | func arith_op2 func
      | arith_opn '(' func {',' func '}'
      | IF '(' predicate ')' THEN '(' func ')' [ELSE '(' func ')']
      | number
      | func_basic
      | func_name

arith_op1 -> '+' | 'SQRT'
arith_op2 -> '+' | '-' | '*' | '/' | '%' | '^'
arith_opn -> 'min' | 'max' | 'mean'

number -> digit{digit} ['. ' {digit}] [exponent]
      | {digit} '.' Digit {digit} [exponent]
digit -> '0' | ... | '9'
exponent -> 'e' ['+'|'-'] digit {digit}

func_basic ->
      'mark(' place_name ')' / number of Token in <place_name>
      | '#'(' place_name ')' / " " " " "
      | 'VAR(' variable_name ')' /declaration of variable <variable_name>

predicate -> '(' predicate ')'
      | 'NOT' predicate
      | predicate bool_arith predicate
      | func compare func
      | 'TRUE'
      | 'FALSE'

bool_arith -> 'AND' | 'OR' | 'EOR' | 'NOR' | 'NAND'
compare -> '=' | '<>' | '>' | '<' | '>=' | '<='

```


Same examples:

- $\text{VAR}(\lambda) * \text{mark}(\text{Place}_2)$
- $\text{IF} ((\text{mark}(\text{Place}_1) < 2) \text{AND} (\text{mark}(\text{Place}_2) = 0)) \text{ THEN } (\text{VAR}(\mu)) \text{ ELSE } (0.001)$

The Enabling Function syntax is:

`enab -> predicate`

`predicate -> '(' predicate ')'`

```

| 'NOT' predicate
| predicate bool_arith predicate
| func compare func
| 'TRUE'
| 'FALSE'

```

Same examples:

- $\text{mark}(\text{Place}_2) \geq 1$
- $(\text{mark}(\text{Place}_1) = 1) \text{ OR } (\text{mark}(\text{Place}_3) < 4)$

The property window associated with the arcs (Figure 2.4) contains the following fields:

- ∅ **Place:** specifies the name of the place the arc is attached to;
- ∅ **Transition:** specifies the name of the transition the arc is attached to;
- ∅ **Type:** specifies the type of the arc: *input*, *output* or *inhibit*;
- ∅ **Weight:** specifies the multiplicity of the arc;
- ∅ **Multiplicity Function:** defines the multiplicity of the arc as a function. It can be a marking-dependent function (example: $\text{mark}(\text{place_name})$).

The image shows a graphical user interface window titled 'Property window'. It contains several input fields and a dropdown menu. The fields are: 'Place' with the value 'Place_1', 'Transition' with the value 'Trans_1', 'Type' with a dropdown menu showing 'Input', 'Output', and 'Inhibit', 'Weight' with the value '1', and 'Multiplicity Function' which is currently empty. At the bottom of the window, there are two buttons: 'Apply' and 'Cancel'.

Figure 2.4: Property window associated with the arc

If a multiplicity function is defined, the field weight is not considered and the multiplicity of the arc is defined by this function.

2.4 Insert Menu

The Insert Menu is used to insert places, transitions or arcs.


Insert->Place (or Ctrl-P): draws a place in the DEEM window by clicking with the left button in the desired position. The same command is executed by clicking the

DEEM_button .



Insert->Transition Horizontal (or Ctrl-H): draws in the DEEM window a horizontal transition by clicking with the left button in the desired position. The same

command is executed by clicking the DEEM_button .

Insert->Transition Vertical (or Ctrl-V): draws in the DEEM window a vertical transition by clicking with the left button in the desired position. The same command is

executed by clicking the DEEM_button .

Insert-> ARC (or Ctrl-A): draws an input, output or inhibit arc. To draw an input arc, click with the left button on the starting place, then on the destination transition with the right button. To draw an output arc, click with the left button on the starting transition, then on the destination place with the right button. To draw an inhibit arc (from a place to a transition) the property window associated with the arc has to be

opened. Input/output arcs can also be inserted clicking the DEEM_button , whereas inhibit arcs can be inserted by clicking the DEEM_button .

2.5 Special Menu

The Special Menu is used to change the look of DEEM window or to move the net (or part of it).

Special->View->Designer:

border: shows/hides the border of the PhN and SN area;

Node names: shows/hides the places name;

Place init Value: shows/hides the initial number of tokens of each place;

Place capacity: shows/hides the places capacity;

Trans Rate/Probability: shows/hides the transitions rate/probability;

Multiplicity Arc: shows/hides the arcs multiplicity;

Trans Priority: shows/hides the immediate transitions priority;

Panel: shows/hides the panel on the left side of the DEEM window;

Special->Net Size: defines the size of the DEEM window (default: (800x600));

Special->Move Net: opens a new window with 4 arrows to move the global net (PhN and SN);

Special->Move Part of Net: moves the selected part of the net in the desired position (chosen by left-clicking).

Special->Colors: allows to choose the combination of colours for places, transitions etc... In the "Colour Groups" list, a set of pre-defined colour combinations is given. After a combination is chosen (by left-clicking on it), it can be assigned clicking on (<-). New combinations can be inserted in the list clicking on (->) and assigning them a name. A combination can be deleted by selecting it and clicking the "delete" button.

Special->Project Report: allows to create the documentation of the MPS model producing a LATEX file containing all model information.

2.6 Zoom Menu

The Zoom Menu is used to change the size of the global net by 25%, 50%, 75% or 100% (the original size);

2.7 Compute Menu

In *Compute Menu* is possible to set parameters, to define some measures of interest and to activate the transient analysis.

2.7.1 Parameters

Compute->Parameters: opens a new window in which all the variables defined within the model are shown. The variable *Time* is always present. When a variable is inserted in the model, it is automatically added to the variable list (the field *Name* showing the name of the variable and field *Value* the default "no_def").

When a variable is removed from the model, it is automatically removed from the variable list. A set of values for each variable represents a *study*. In the same window, it is possible to create a new *Study*, open an old one (selected from a list of studies) that becomes the current one, or remove a *Study*. The window shows the study list and the values of the variables of the current study (Figure 2.5).

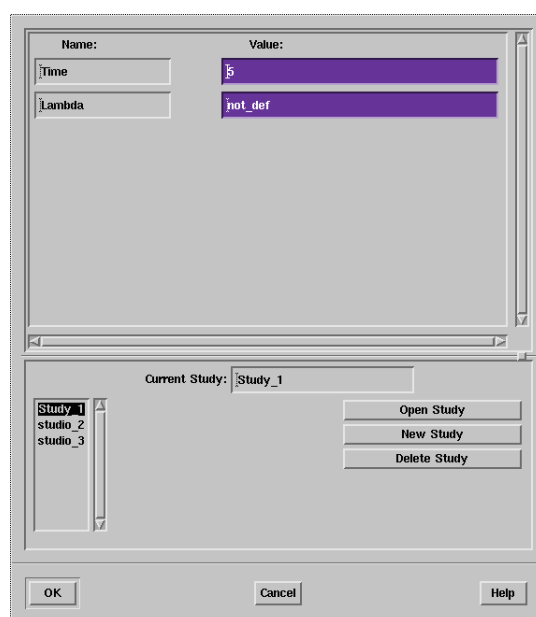


Figure 2.5: Parameters and studies window

The values of the variables can be specified in the following way:

```
value -> number / example  $\lambda = 0.01$ 
| range / example  $\lambda = [1,7,2] = 1,3,5,7$  or  $\lambda = [1,8,*2] = 1,2,4,8$ 
| value_set / example  $\lambda = \{0.0002, 0.009,0.002,0.09,0.02\}$ 
```

The first two values of *range* represent the interval of variability for the parameter λ ; the last value represents the incremental step (to be added or multiplied) to obtain the intermediate values in the set. As a default the incremental step is added and it is represented by a '+' (or nothing) before the last value. For a multiplying incremental step a '*' has to be put before the last value.

Each set of values of the variables (one numerical value for each variable) corresponds to an experiment (evaluation run).

2.7.2 Measures

Compute->Measures: permits to define the measures. Measures are defined by a Name, a reward function and an analysis type flag ("ist", "cum" or "tim_av" for instantaneous, cumulated and timed-averaged analysis respectively). The analysis type flag is set clicking on the relative button. Composed measures also can be defined through an expression composing the evaluated reward-based measures (referred with the construct FUN), as showed at bottom in Figure 2.6.

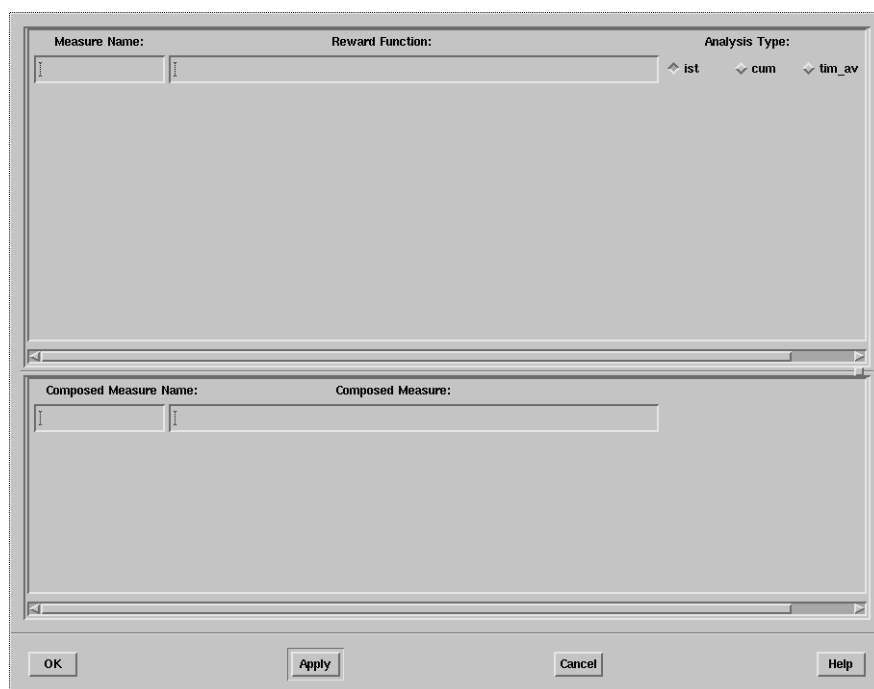


Figure 2.6: Measures window

The syntax of Reward Functions is:

```
Reward_Function ->
    IF '(' predicate ')' THEN (' func ') ['ELSE (' func ')']
| IF '(' impulse_predicate ')' THEN (' impulse_func ')'
```

where the syntax of the func and impulse_func is:

```
func -> '(' func ')'
      | arith_op1 func
      | func arith_op2 func
      | arith_opn '(' func {',' func ')' }
      | IF '(' predicate ')' THEN (' func ') ['ELSE (' func ')']
      | number
      | func_basic
      | func_name
```

```
impulse_func -> '(' impulse_func ')'
              | arith_op1 impulse_func
              | impulse_func arith_op2 impulse_func
              | arith_opn '(' impulse_func {',' impulse_func ')' }
              | number
              | VAR(' variable_name ')'
```

```
arith_op1 -> '+' | SQRT
arith_op2 -> '+' | '-' | '*' | '/' | '%' | '^'
arith_opn -> 'min' | 'max' | 'mean'
```

```
number -> digit{digit} ['. ' {digit}] [exponent]
        | {digit} '.' Digit {digit} [exponent]
digit -> '0' | ... | '9'
exponent -> 'e' ['+' | '-'] digit {digit}
```

```
func_basic ->
             mark(' place_name ') /number of Token in <place_name>
             | #(' place_name ') / " " " " "
```

```
bool_arith -> AND | OR | EOR | NOR | NAND
compare -> '=' | '<>' | '>' | '<' | '>=' | '<='
```

The syntax of predicate and impulse_predicate is:

```

predicate -> '(' predicate ')'
    | 'NOT' predicate
    | predicate bool_arith predicate
    | func compare func
    | 'TRUE'
    | 'FALSE'

impulse_predicate -> 'fire (' trans_name ')'2

```

Some examples:

- IF (mark(Place_1)=1) THEN (mark(Place_2)) ELSE (1)
- IF ((mark(Place_1)>1) AND (mark(Place_2)=0)) THEN (1)
- IF (fire(Trans_1)) THEN (5)

As a future work, it will be given the possibility also to define some derived measures, obtained as functions of the measures evaluated by the tool in phase of post-processing (example: measures of performability or cost).

2.7.3 Transient Analysis

Compute->Transient Analysis: permits to enter the parameters for the transient analysis (Figure 2.7).

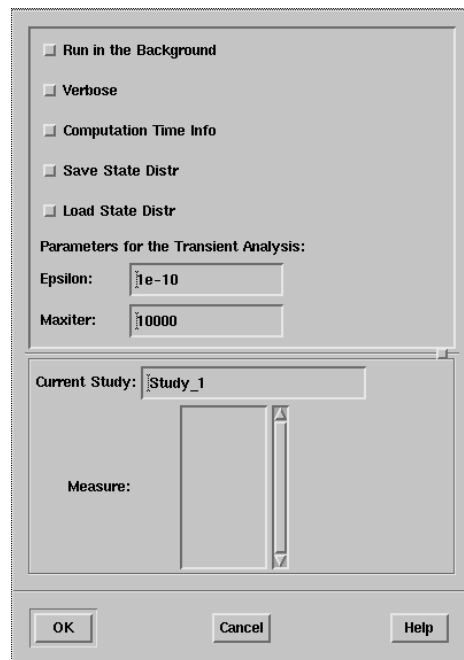


Figure 2.7: Transient analysis window

² Not yet implemented in the release 1.0.beta-15.

The field **Epsilon** represents the error tolerance, **Maxiter** the maximum number of iterations that has to be considered by the transient solution method.

When **Run in the Background** mode is selected, graphic interface of DEEM can still be used while the solver is running. In the **Verbose** mode, more information about the solution is showed on the shell while the solver is running. In **Computation Time Info** mode, information about the analysis times is produced.

Save State Distr mode permits to save in file netname.studyname.sdistro the state distribution of the net at the end of the transient analysis. **Load State Distr** mode permits to load from file netname.studyname.sdistri the state distribution of the net at initial time of the transient analysis.

The field **Current Study** shows the study selected for the analysis and the field **Measure** shows the measures that are calculated. The **OK** button makes the analysis to start.

2.8 ? Menu

The **? Menu** contains some information about DEEM (version, Xserver, Xclient, the creator...). The figure 2.8 shown this window.

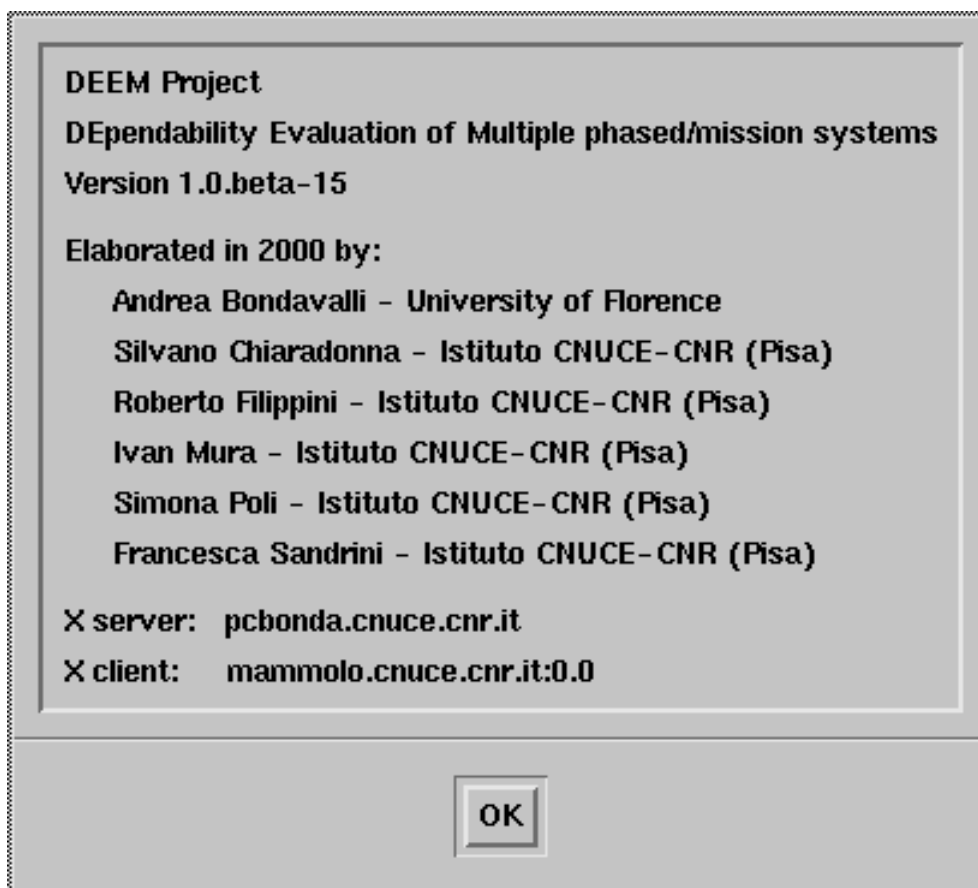


Figure 2.8 Information window of DEEM

How to create and solve a Model: an example

This chapter describes how to create and solve a model with DEEM, by applying the process to a real example of Scheduled Maintenance System (SMS) [3].

3.1 A working example

Consider a system equipped with two components. Component A is a primary unit providing some functionality to the system, and component B acts as a backup unit for component A. The system is equipped with a switching logic, so that when A fails, the control of operation is immediately passed to B.

The system executes cyclically two different types of mission. Mission 1 encompasses a single phase of fixed duration τ_{11} where Mission 2 is a two-phase mission, whose phases have duration τ_{21} and τ_{22} , respectively. The time to failure of component A is exponentially distributed, with parameter λ_{1A} during the mission of type 1, and parameter λ_{2A} during the mission of type 2, whereas the time to failure of component B is constantly λ_B .

The following scheduled maintenance actions are undertaken during system lifetime:

- The system is subject to a complete maintenance check every 100 missions;
- Primary unit A is replaced at the end of each mission, if failed. After the replacement, A takes again the role of primary unit;
- Backup unit B is subject to a partial check at the end of each α pairs of missions.

After 100 missions, all the components are checked, and the system is restored to the initial condition. We want to evaluate the *Reliability* R of the SMS defined as the not occurrence of a failure in both A and B.

3.2 Creation of the graphical Model

To execute DEEM, write the command **deem** (the window in Figure 2.1 will then appear). As already described in Chapter 2, the deem window for the model representation is composed of two parts: the PhN window (up part) for the model of the phases, and the SN window (low part) for the failure/repair model of the system components. Using the graphical commands (Insert Menu, introduced in Chapter 2), the models for the PhN and SN of our SMS running example can be constructed as illustrated in Fig. 3.9.

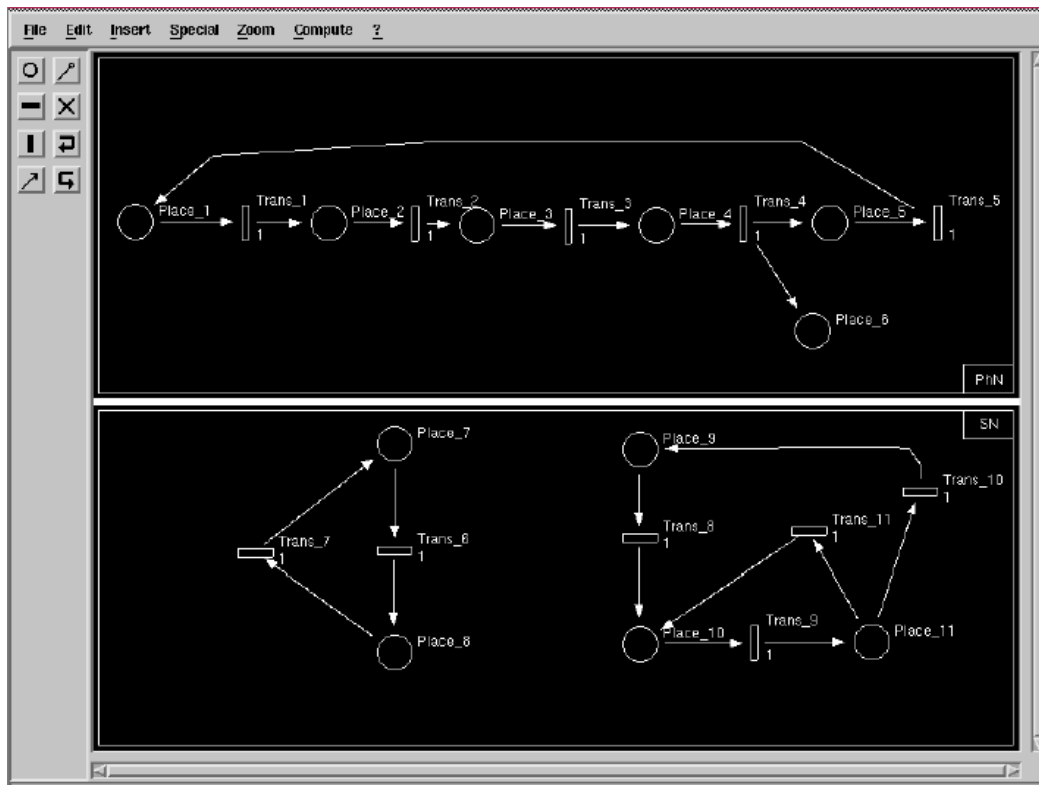


Figure 3.9: A sketch of the SMS model

The PhN has 6 places, 5 transitions and arcs linking places and transitions. The SN has 5 places, 6 transitions and arcs that linking places and transitions. Default names to places and transitions are automatically assigned by the tool. To save the model, click on **Save As...** (File Menu, in Chapter 2).

3.3 Properties of places, transitions and arcs

Now, properties have to be assigned to each object (places, transitions and arcs).

ü For each place, open the property window (select the place and a) click on Properties of the Edit Menu, or b) click with the right button of the mouse). Assign name, number of tokens and capacity to the place. Click **Apply** to confirm. Following the specifications of our SMS example, properties to the places of figure 3.9 are assigned as follows:

- Change the name Place_1 in P1. P1 represents the phase of Mission 1 and its initial number of tokens is 1 (The global mission starts at place P1);
- Change the name Place_2 in Stop1. Stop1 is necessary because Mission 2 can begin only if component A did not fail. If A failed, the system waits in Stop1 the repair of this component;
- Change the names Place_3 and Place_4 in P2 and P3, respectively. These places represent the two phases of Mission2;
- Change the name Place_5 in Stop2. Stop2 is necessary because Mission 1 can begin only if component A and B did not fail. If A and B failed, the system waits in Stop2 the repair of the component(s);
- Change the name Place_6 in Count. Count is the place that stores the number of cycles executed;
- Change the name Place_7 in Aok. Aok represents the state healthy of component A. Its initial number of tokens is 1 (at start time, the component is healthy);

- Change the name Place_8 in Afail. A token in Afail represents the failure of component A;
 - Change the name Place_9 in Bok. Bok represents the state healthy of component B. It's initial number of tokens is 1 (at start time, the component is healthy);
 - Change the name Place_10 in Bfail. A token in Bfail represents the failure of component B;
 - Change the name Place_11 in repair_B. A token in repair_B indicates that component B is under repair.
- ü For each transition, open the property window (select the place and a) click on Properties of the Edit Menu, or b) click with the right button of the mouse) and assign the name to the transition. By clicking on **Transition Type**, the choice between *immediate* or *timed* can be made. If *immediate* is selected, the firing probability and, possibly, the priority have to be inserted. If *timed* is selected instead, a new choice between *Deterministic* and *Exponential* has to be performed, both requiring the insertion of the transition rate. Click **Apply** to confirm. The probability/rate of firing can be marking-dependent; if this is the case, the field "Rate Function" must be filled, using the appropriate syntax defined in Chapter 2. If the firing of the transition is marking dependent, the field "Enabling Function" must be filled too. Following the specifications of our SMS example, properties to the transitions of figure 3.9 are assigned as follows:
- Trans_1: change the name in T1. T1 is a deterministic transition that model the time (10 hours) needed for the system to perform the phase of mission 1. This transition is enabled if the number of tokens in place Count is less than 50;
 - Trans_2: change the name in Tstop1. Tstop1 is an immediate transition with probability 1 and priority 1. It may be enabled only if the state of component A is healthy. This condition is obtained by giving the greater priority to the immediate transition (Yes_repair_A) that represents the repair of component A;
 - Trans_3 and Trans_4: change those names in T2 and T3 respectively. They are deterministic transitions that model the time (5 and 10 hours, respectively) needed for the system to perform the two phases of Mission 2;
 - Trans_5: change the name in Tcount. Tcount is an immediate transition with probability 1 and priority 1. It may be enabled only if the state of component A is healthy and it is the end of the $(\alpha*i)$ -th mission, $i=1, \dots, \lfloor 50/\alpha \rfloor$;
 - Trans_6: change the name in fA. fA is an exponential transition with firing time equal to 0.001 (λ_{1A}) if Mission 1 is the active mission, otherwise it is equal to 0.002 (λ_{2A}). fA is enabled if the number of tokens in place Count is less than 50;
 - Trans_7: change the name in Yes_repair_A. This transition represents the scheduled maintenance action for component A, which is replaced at the end of each mission, if failed. It is an immediate transition with probability 1 and priority 2 (it must fire before transitions "Tstop1" and "Tcount"). Yes_repair_A is enabled if: i) the place Bfail does not have tokens; ii) the number of tokens in place Count is less than 50; and iii) it is the end of Mission 1 or 2;
 - Trans_8: change the name in fB. fB is an exponential transition with firing time λ_B . fB is enabled if the number of tokens in place Count is less than 50;
 - Trans_9: change the name in yes_repair_B. This transition represents the possible scheduled maintenance action for component B (backup unit B is subject to a partial check at the end of each α pairs of missions). It is an immediate transition with probability 1 and priority 2 (it must fire before transitions "Tstop1" and "Tcount"). yes_repair_B is enabled if: i) the place Afail

does not have tokens, ii) it is the end of Mission 1 or 2, and iii) it is the end of one of the α pairs of missions;

- Trans_10 and Trans_11: change those names in ok_repair and nok_repair, respectively. ok_repair represents a successful repair action; it has a probability equal to p. nok_repair represents the failure of the repair action; it has a probability equal to 1-p. They are enabled only just before the starting of Mission 1 (there are not tokens in places Stop1 and Stop2!).

The figure 3.10 shows an example of rate and enabling function. The rate of the exponential transition is 0.001 if there is a token in place P1, otherwise it is 0.002.

Figure 3.10: Definition of rate and enabling function

- ü If there are arcs with a multiplicity different from 1, the property window associated with those arcs has to be opened (select the arc and a) click on Properties of the Edit Menu, or b) click with the right button of the mouse). To set the multiplicity, the field “Weight” can be filled with an integer number (for example 2) or, if the multiplicity is marking-dependent, insert the appropriate function in the field “Multiplicity Function” (for example mark(Place_1)). In this last case, the multiplicity changes as a function of the number of tokens of the place(s) it depends on. In the SMS example we are working on, all arcs have multiplicity 1.

The figure 3.11 shows the complete model obtained.

3.4 Values of the parameters and studies

To assign values to all the model parameters, select **Parameters** in *Compute Menu*. A window will appear, listing all the variables included in the model with associated an undefined value (*no_def*). Assignments to such variables may be performed through the insertion of: i) a constant number (it is the case of λ_{1A} and λ_{2A} , whose values are 0.001 and 0.002), or ii) a set of values (e.g., {0.00001,0.001,0.0015,0.002,0.5}), or iii) a range of values (e.g., [0.001,0.01,0.001], where the first two numbers represent the extreme values assumed by the variable, and the third number is the step to be added or multiplied to make the variable assume intermediate values in the range). If a

multiplicity of values is given, the variable assumes all the values in the set, and a different transient analysis is performed for each of them.

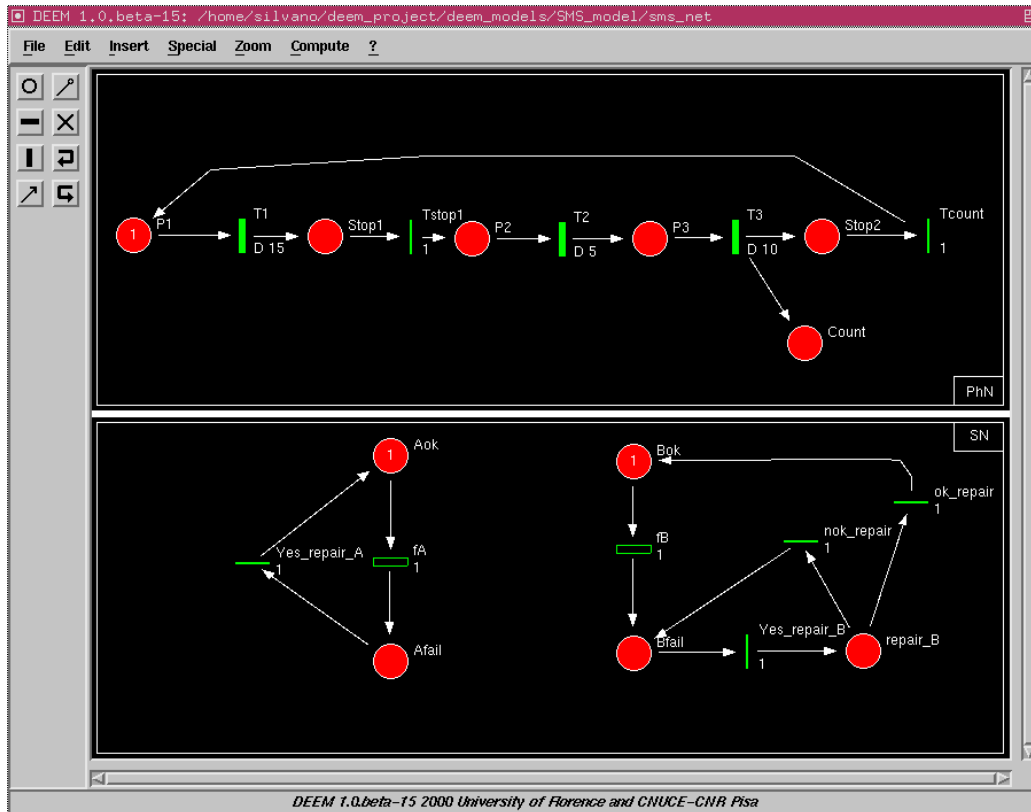


Figure 3.11: SMS model

To create a study, click on **New**. Insert the name of the study and click **Apply**. Now, assignments to the variables can be performed (as illustrated in Figs. 3.12a and 3.12b).

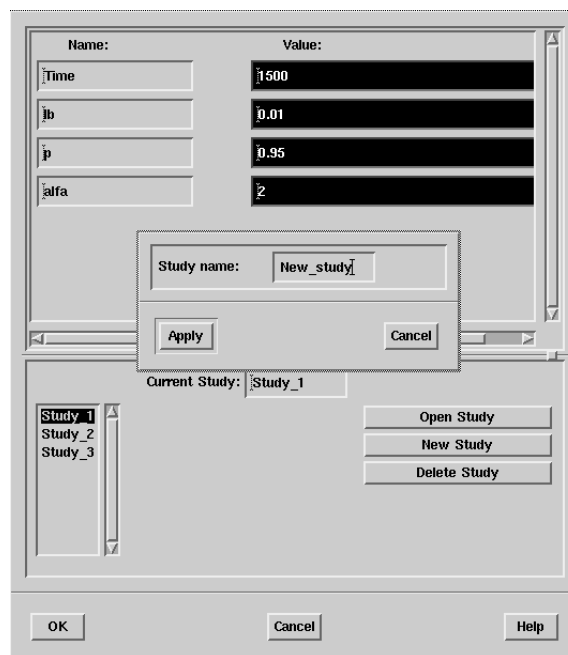


Figure 3.12a The window where insert the name of the new study

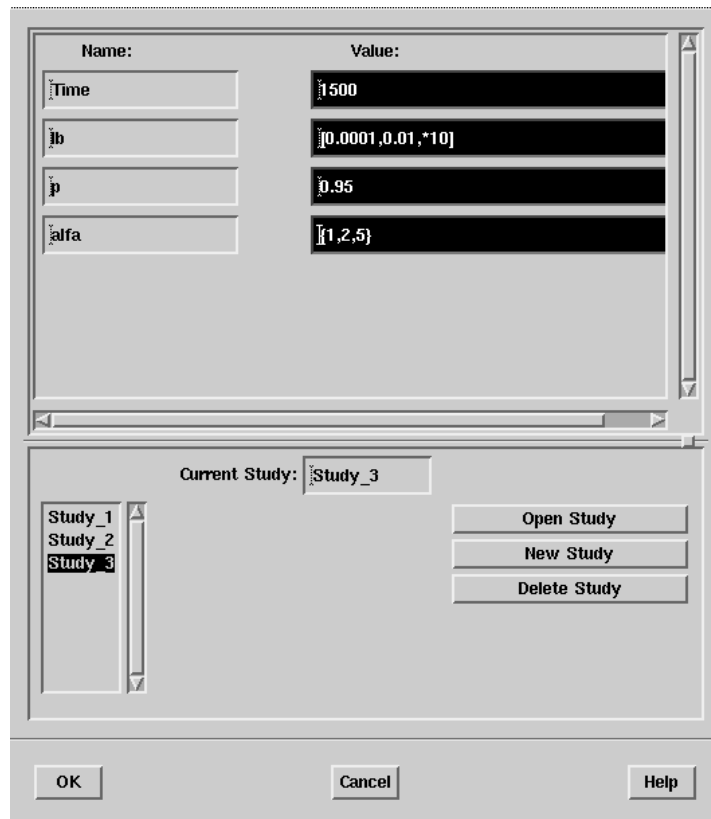


Figure 3.12b A definition of study

3.5 Definition of the measures

To define the measures to be evaluated, select **Measures** in *Compute Menu*. For each measure, it is required the definition of: i) a name, ii) a reward function, and iii) the analysis type flag (instantaneous, cumulated and timed-averaged analysis). In our example, we are interested in evaluating the Reliability of the system. Therefore:

- i) Reliability
- ii) IF (mark(Afail)+mark(Bfail)<2) THEN (1) (at least one component must be up!)
- iii) click on “instantaneous”

It is possible to define several measures to be evaluated together in a study. Click on the **Apply** button and a new row for inserting the name, reward function and analysis type flag will appear below the previously defined measure(s). E.g., if the Unreliability would have been of interest too, click on **Apply** and fill the new field with the appropriate values, that is:

- i) Unreliability
- ii) IF (mark(Afail)+mark(Bfail)>=2) THEN (1) (both components A and B are failed, thus leading to the failure of the system)
- iii) click on “instantaneous”

To confirm and close the Measure window, click on **OK**. The figure 3.13 shows how the measures are defined.

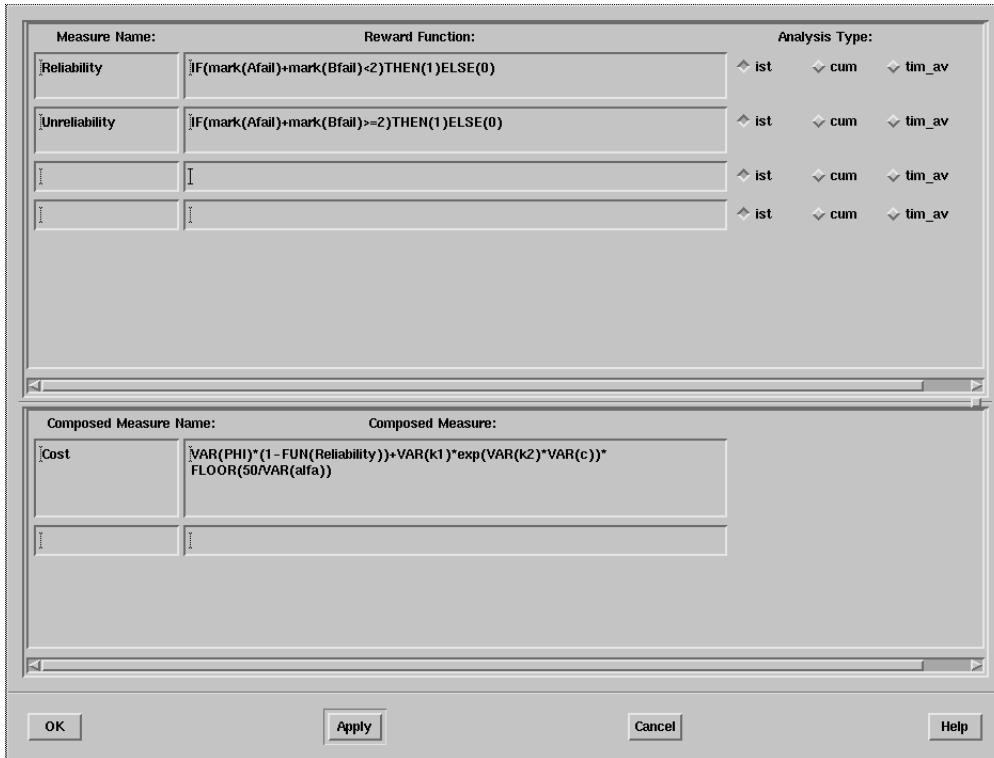


Figure 3.13: Measures defined for the SMS model

3.6 Transient analysis

The model evaluation follows the above described steps. To evaluate a model, select **Transient Analysis** in *Compute Menu*. The study under evaluation is the current study, as defined in the parameters window. To start the analysis, click on **OK**. The figure 3.14 shows an example.

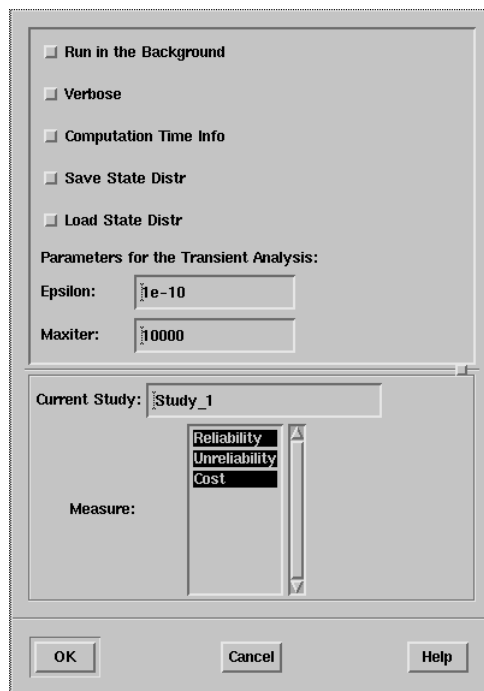


Figure 3.14: Transient analysis for the study Study_1

Results and output files

The Transient analysis produces various types of files (with different extensions). Some describe intermediate information useful for debugging a model or to trace the evaluation process (they are the files *.mc*, and *.rg*), others contain numerical results of the evaluation. This chapter lists and explains the output files of DEEM.

4.1 The PhN files

The Transient analysis creates two different PhN files.

- *Filename_PhN.mc*: describes the Markov Chain (MC) of the PhN. A part of this file is shown below. “_nstates” represents the number of states of the model. “_order = TOFROM” represents the type of order (arrive on state X if the current state is Y). The MC is represented by “_matrix” with the following syntax: "destination_state, origin_state:probability".

```
#####
#
#           DEEM Dependability Modelling and Evaluation           *
#
#           Version 1.0, xx.xx.00                                 *
#
#   (C) 2000 by Institute CNUCE, CNR Ghezzano(Pisa) ITALY      *
#
#                                                                 *
#####

This file was generated Tue Jun 27 12:57:22 2000
```

```
_firstindex = 0;
_nstates = 154;
_nentries = 153;
_order = _TOFROM;
_matrix =
0_0          ;
0_1          0_0:1.000000000000e+00;
0_2          0_1:1.000000000000e+00;
0_3          0_2:1.000000000000e+00;
0_4          0_3:1.000000000000e+00;
```

- *Filename_PhN.rg*: describes the Reachability Graph (RG) of the PhN. Two parts of these files are shown below. All places and transitions have associated an integer number used in “reachset” and “reachgraph”. In reachset “0_4_t 1:1 5:1” represents the marking of state 0_4_t. In this state there is a token within place 2 (P2) and a token within place 5 (Count). In reachgraph “0_1 0_0:6:1.000000e+00” means that state 0_1 can be reached from state 0_0 by the firing of transition 6 (T1) which rate is 1.000000e+00.

```

#*****
#
#           DEEM Dependability Modeling and Evaluation           *
#
#           Version 1.0.beta-2                                   *
#
#   (C) 2000 by Istituto CNUCE, CNR Ghezzano(Pisa) ITALY      *
#
#
#*****

```

This file was generated Tue Jul 11 15:02:24 2000

```

_nplace = 11;
_ntrans = 11;
_places =
    0: P1;
    1: P2;
    2: P3;
    3: Aok;
    4: Afail;
    5: Count;
    6: Stop2;
    7: Bok;
    8: Bfail;
    9: Stop1;
    10: repair_B;
_transitions =
    0: Yes_repair_A;
    1: yes_repair_B;

```



```

2: Tcount;
3: Tstop1;
4: ok_repair;
5: nok_repair;
6: T1;
7: T2;
8: fA;
9: T3;
10: fB;

_ntanmark = 150;
_nabsmark = 1;
_nvanmark = 0;
_nvanloop = ???;
_nentries = 150;
_reachset =
    0_0_t  0:1;
    0_1_t  1:1;
    0_2_t  2:1;
    0_3_t  0:1    5:1;
    0_4_t  1:1    5:1;
    0_5_t  2:1    5:1;
    0_6_t  0:1    5:2;

```

Second part of file:

```

_reachgraph =
    0_0;
    0_1    0_0:6:1.000000e+00;
    0_2    0_1:7:1.000000e+00;
    0_3    0_2:9:1.000000e+00;
    0_4    0_3:6:1.000000e+00;
    0_5    0_4:7:1.000000e+00;
    0_6    0_5:9:1.000000e+00;

```

4.2 The files created for each phase

The transient analysis creates three different files during the analysis.

- *Filename.mc*: describes the Markov Chain (MC). It has the same format of file .mc of PhN.
- *Filename.rg*: describes the Reachability Graph (RG). It has the same format of file .rg of PhN,
- *Filename.trsolv*: contains the state probability distribution at time **t** and average probability to being in that state at time **t**. A part of this file is shown below.

```

#*****
#
#           DEEM Dependability Modeling and Evaluation           *
#
#           Version 1.0.beta-2                                   *
#
#   (C) 2000 by Istituto CNUCE, CNR Ghezzano(Pisa) ITALY      *
#
#
#*****

```

This file was generated Tue Jul 11 15:02:25 2000

```
-----> PHASE NUMBER : 0
```

```
STATE:          PROBABILITY,      EXPECTED TOTAL TIME (TIMEAVG);
```

```
-----
```

```

_timepoint: 1.500000e+01
0_1:          8.478937040879e-01,      9.218563388611e-01,
0_2:          1.281427233714e-02,      6.757151638498e-03,
0_3:          1.372182355086e-01,      7.068102093397e-02,
0_4:          2.073788052551e-03,      7.054885648605e-04,

```

4.3 The files created for each study

The last files that the transient analysis creates are the files of the defined studies (one for each study).

- *Filename.studyname.spreadsheet*: contains the measures evaluated for the study *studyname*, in a format which can be further elaborated by spreadsheet programs.
- *Filename.studyname.gnuplot*: contains the measures evaluated for the study *studyname*, in a format which can be further elaborated by gnuplot .

It is possible to open the file *Filename.studyname.spreadsheet* using some spreadsheets programs like EXCEL Microsoft for any graphical representation. An example of this type of file is shown below.

```

*****
#
#           DEEM Dependability Modeling and Evaluation           *
#
#           Version 1.0.beta-2                                   *
#
#   (C) 2000 by Istituto CNUCE, CNR Ghezzano (Pisa) ITALY      *
#
#
*****

```

This file was generated Tue Jul 11 15:03:21 2000

STUDY_NAME: Study_3

STUDY_variables_setting:

```

Time      1.500000e+03
lb        [0.0001,0.01,*10]
p         9.500000e-01
alfa     {1,2,5}

```

REW_MEASURE_NAME: Reliability

PREDICATE: mark(Afail)+mark(Bfail)<2

REW_FUNCTION: 1

lb/alfa 1.000000e+00 2.000000e+00 5.000000e+00

1.000000e-04	9.941818e-01	9.907599e-01	9.813445e-01
1.000000e-03	9.440268e-01	9.132017e-01	8.353453e-01
1.000000e-02	6.015980e-01	4.783397e-01	3.020994e-01

REW_MEASURE_NAME: Unreliability
 PREDICATE: mark(Afail)+mark(Bfail)=2
 REW_FUNCTION: 1

lb/alfa	1.000000e+00	2.000000e+00	5.000000e+00
1.000000e-04	5.818176e-03	9.240130e-03	1.865547e-02
1.000000e-03	5.597324e-02	8.679829e-02	1.646547e-01
1.000000e-02	3.984020e-01	5.216603e-01	6.979006e-01

Figure 4.14 shows the plot of the Reliability, generated by EXCEL Microsoft from the output file of the DEEM transient analysis.

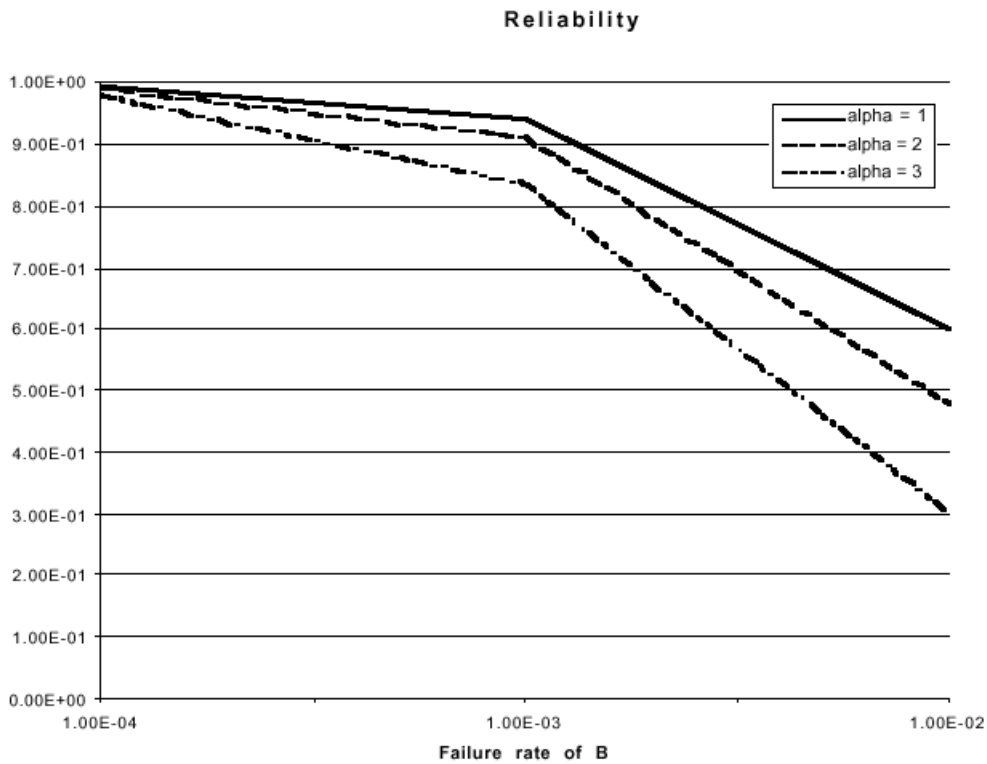


Figure 4.14: Reliability of the system

Transient Solver

DEEM provides a specific and efficient analytical solution for MPS models. This chapter describes the DEEM's solution algorithm.

5.1 The analytical technique

The specialised solution finds its ground by observing that the only deterministic transitions in a DSPN model of a MPS are the phase duration, and that these transitions are enabled one at a time. Thus, the marking process $\{M(t), t \geq 0\}$ of the DSPN is a Markov Regenerative Process (MRGP) [4] for which the firing times of the deterministic transitions are indeed regeneration points. Moreover, the following property holds of the DSPN model of a MPS:

Property 1: in every non-absorbing marking of the DSPN there is always one deterministic transition enabled, which corresponds to the phase being currently executed.

The general solution method for MRGP processes considers computing matrix $V(t)$, whose entry (m, m') is the occupation probability of marking m' at time $t \geq 0$ given the initial marking m . Matrix $V(t)$ is the solution to the generalised Markov renewal equation $V(t) = E(t) + K(t) * V(t)$, where $K(t)$ and $E(t)$ are the global and local kernel matrices [4] and $*$ is the convolution operator. Instead of directly attacking the solution of the generalised Markov renewal equation by numerical algorithms or Laplace-Stiltjes transform, DEEM computes matrix $V(t)$ according to the analytical method, proposed in [5, 6].

5.2 The solution algorithm

The following equation:

$$V_{i,j}(t) = \left(\prod_{h=1}^{r-1} e^{Q_{ph} \tau_{ph}} \Delta_{ph, ph+1} \right) e^{Q_j \delta}$$

allows to evaluate $V(t)$ through the separate analysis of the various alternative paths which compose the mission, and only requires the derivation of matrix exponentials $e^{Q_i t}$, and $\Delta_{i,j}$, $i, j = 1, 2, \dots, n$. which can be automatically obtained when the reachability graph is generated. The solutions of the DSPN model is thus reduced to the cheaper problem of solving a set of homogeneous, time-continuous smaller Markov chains.

To compute the dependability figures of the system, DEEM derives the probability vector $P(t)$ of each marking in SN at time t . We can obtain $P(t)$ from the transient probability matrix $V(t)$, with the equation $P(t) = P_0 \cdot V(t)$, where P_0 is the initial probability vector of the DSPN. To compute $P(t)$ and then the dependability figures of the system, the solution engine of DEEM takes as input the DSPN model and its initial probability vector P_0 , and performs the following algorithm:

1. Builds RGP, the reachability graph of the PhN sub-model. This graph has exactly one stable marking \dot{m}_i for each phase $i = 1, 2, \dots, n$ the MPS may perform.
2. Call `deem_solver(1, P0, 0)`.

The recursive algorithm of `deem_solver(i, Piinit, tiinit)` is:

1. Builds the reachability graph $RGS(\dot{m}_i^r)$ of the whole DSPN model when marking \dot{m}_i is the only one permitted for the PhN. From $RGS(\dot{m}_i^r)$ obtains the transition rate matrix Q_i of the continuous-time Markov chain describing the evolution of the DSPN during the execution of phase i .
2. If there are not phases next to phase i or $t \leq t_i^{init} + \tau_i$ then derivate the transient state probability vector $P_i(t) = P_i^{init} e^{Q_i(t-t_i^{init})}$ and return, else continue to step (3).
3. Derivate the transient state probability vector $P_i = P_i^{init} e^{Q_i \tau_i}$.
4. Builds the reachability graph $RGS(\dot{m}_i^r, next(\dot{m}_i^r))$, where $next(\dot{m}_i^r) = \{\dot{m}_{j_1}^r, \dots, \dot{m}_{j_m}^r\}$, of the whole DSPN model, when the initial marking of the PhN is \dot{m}_i , and transition t_i^{Det} is the only deterministic one allowed to fire. Each marking \dot{m}_{j_h} is reachable from \dot{m}_i through the firing of some instantaneous transition next the firing of t_i^{Det} .
5. For each stable marking \dot{m}_{j_h} (phase j_h) performs the following steps:
 - 5.1. From $RGS(\dot{m}_i^r, next(\dot{m}_i^r))$, obtains the branching probability matrix Δ_{i,j_h} for the transition from phase i to phase j_h .
 - 5.2. Derivate the initial state probability vector of the phase j_h : $P_{j_h}^{init} = P_i \Delta_{i,j_h}$.
 - 5.3. Call `deem_solver(j_h, Pj_hinit, tiinit + τ_i)`.

In this solution algorithm DEEM evaluates the specific dependability measure of interest for the MPS from P_i according to the standard computation algorithms.

The main computational cost of the DEEM solution algorithm is that required for the transient solutions, steps (2) and (3), and the multiplications in step (5.2) of the algorithm sketched in the previous section. Notice that the DEEM approach to generate the required matrices ever requires to handle the entire state space of the MRGP process.

Appendix

Installation

To use DEEM do the following:

- visit <http://bonda.cnuce.cnr.it/DEEM> to obtain the license file ".deemlicense" and drop this file in your home directory
- unzip and untar the file "deem-releasnumber.bin.arch.tar.gz"
- set the environment variable DEEM_HOME to the path of the installation directory of DEEM
- here is an example of lines that could be added to someone's .cshrc (assuming use of C shell) to execute deem:

```
alias deem 'cd $DEEMHOME/deem_models;$DEEMHOME/bin/deem&'  
setenv DEEM_HOME /where/is/DEEM
```

- to invoke DEEM run "deem" from the command line prompt

Please report any bugs to: silvano.chiaradonna@cnuce.cnr.it

Tool Organization and File Structure

This paragraph gives the organization of DEEM and its file structure (Figure A.1).

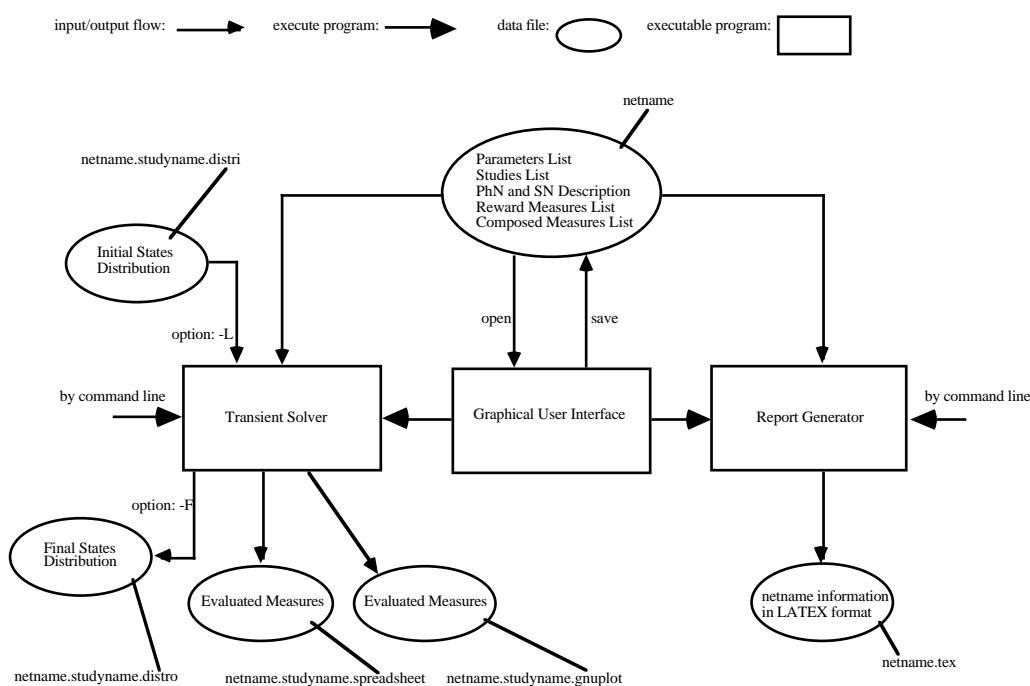


Figure A.1: Organization of DEEM and its file structure

Bibliography

- [1] M. Ajmone Marsan and G. Chiola, "On Petri nets with deterministic and exponentially distributed firing times," *Lecture Notes in Computer Science*, Vol. 266, pp. 132-145, 1987.
- [2] S. Allmaier and S. Dalibor, "PANDA - Petri net analysis and design assistant," in *Proc. Performance TOOLS'97*, Saint Malo, France, 1997.
- [3] A. Bondavalli, I. Mura and K.S. Trivedi, "Dependability Modelling and Sensitivity Analysis of Scheduled Maintenance Systems," in *Proc. EDCC-3 European Dependable Computing Conference*, Prague, Czech Republic, 1999, pp. 7-23.
- [4] H. Choi, V.G. Kulkarni and K.S. Trivedi, "Transient analysis of deterministic and stochastic Petri nets," in *Proc. 14th International Conference on Application and Theory of Petri Nets*, Chicago, IL, 1993, pp. 166-185.
- [5] I. Mura and A. Bondavalli, "Markov Regenerative Stochastic Petri Nets to Model and Evaluate the Dependability of Phased Missions," *IEEE Transactions on Computers*, to appear, Vol. pp. 2001.
- [6] I. Mura, A. Bondavalli, X. Zang and K.S. Trivedi, "Dependability Modeling and Evaluation of Phased Mission Systems: a DSPN Approach," in *Proc. DCCA-7 Dependable Computing for Critical Applications*, San Jose, CA, 1999, pp. 319-337.