Mercury Tool Manual
v4.7.0

MoDCS Research Group

[`http://www.modcs.org`]

CIn - Centro de Informatica UFPE

Cidade Universitaria - 50740-540 - Recife - Brazil

- Tel +55 81 2126.8430 -

November 30, 2018

Contents

1 Overview

This manual describes the **Mercury** tool: a software for supporting performance, dependability, and energy flow modeling in an easy and powerful way. The tool provides graphical interfaces for creating and evaluating Stochastic Petri Nets (SPN), Reliability Block Diagrams (RBD), Energy Flow Models (EFM), Continuous Time Markov Chains (CTMC), Discrete Time Markov Chain (DTMC) and Fault Tree (FT). EFM is proposed to compute the sustainability and cost estimates of data center power and cooling infrastructures whilst conforming to the energy constraints of each device.

Mercury has been developed by **MoDCS** (Modeling of Distributed and Concurrent Systems) Research Group at Informatics Center (CIn) of the Federal University of Pernambuco (UFPE) in Brazil since 2009

A comprehensive view of features is described here as well as steps for creating, editing, and evaluating the models supported by this tool. An overview of Mercury features is depicted below:

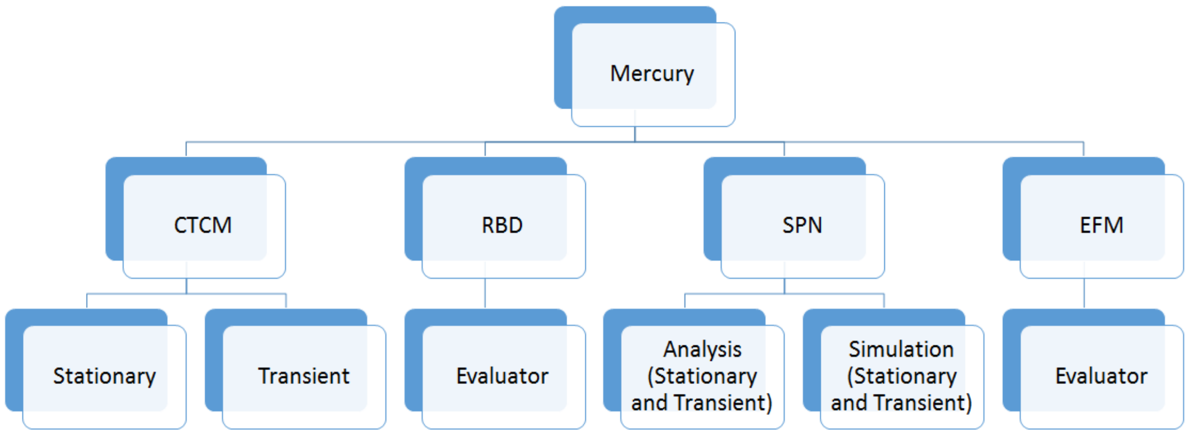

Mercury
CTCM
RBD
SPN
EFM
Stationary
Transient
Evaluator
Analysis (Stationary and Transient)
Simulation (Stationary and Transient)
Evaluator

Figure 1: Mercury Tool - Features

1.1 How to install the Tool

In order to install the newest version of the Mercury software, the user should access the URL `http://www.modcs.org/mercury`.

There is a license agreement document that must be signed and sent to Mercury creators before the access to the download page is granted to the user.

Mercury is made available in a compressed archive, containing the executable files (.jar and .bat), a folder containing the third-party libraries required for Mercury execution, and a folder with example models. All files require approximately 65 MB of disk space. The memory footprint of Mercury in runtime is about 10 MB, but it might increase depending on size of models and type of analysis executed with the tool.

Extract the archive, and double-click the executable file (**Mercury.jar** or **Mercury.bat**) file to open the Mercury tool. The **lib** folder must be kept in the same path of the jar file.

1.2 Graphical User Interface

Mercury has different views: (i) SPN, (ii) RBD, (iii) CTMC, (iv) DTMC, (v) EFM and (vi) FT. This section shows each one of these views by describing them in details.

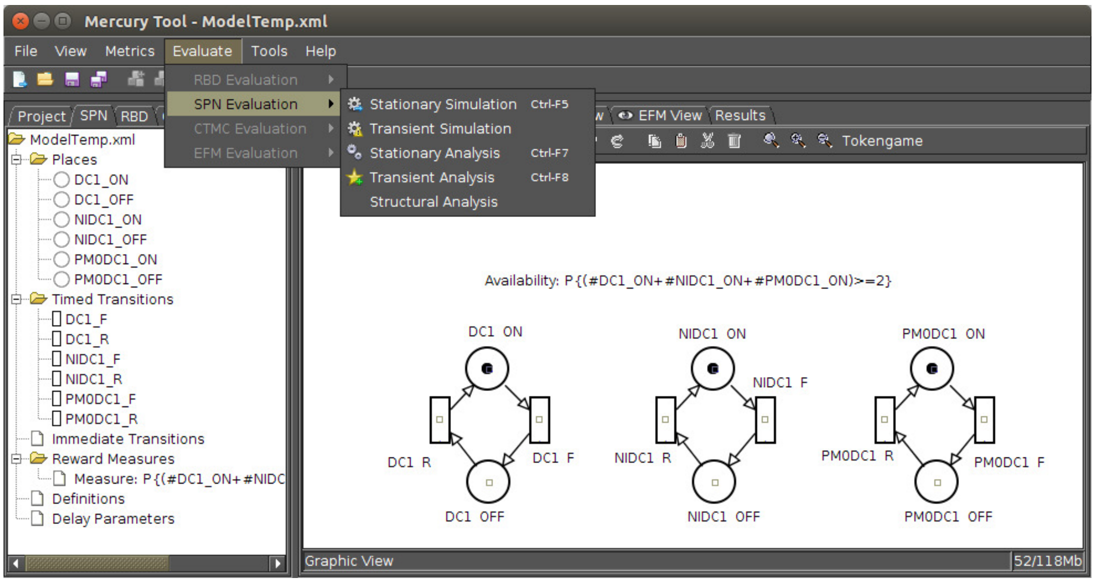


Figure 2: SPN View.

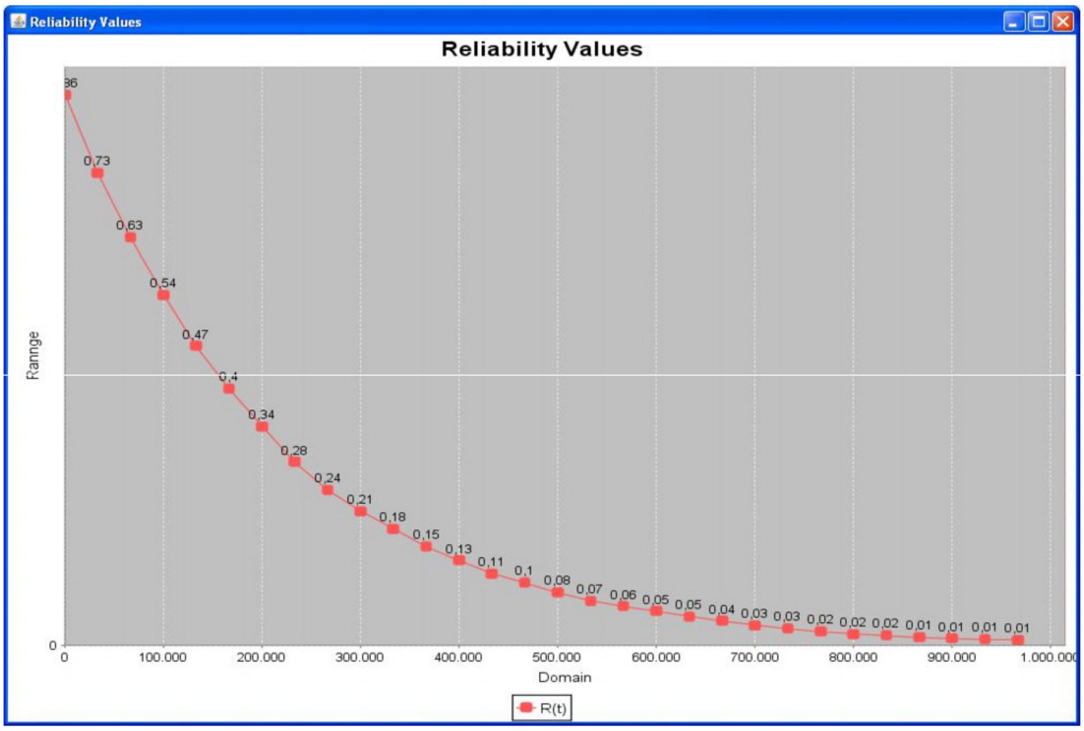


Figure 3: Mercury Tool - Transient Results.

1.2.1 SPN View

Regarding SPN models, Mercury tool performs dependability evaluation through simulation or analysis (i.e., numerical solution of underlying Markov chain). Both kinds of evaluation allow computing **transient** and

stationary metrics. Moreover, time-dependent metrics have been obtained through transient evaluation while steady-state metrics are the result of stationary evaluation. Figure 2 depicts the SPN View of the tool and Figure 3 shows a transient simulation result as an example.

1.2.2 CTMC View

The CTMC view (Figure 4) provides features for drawing and evaluating Continuous time Markov chains. The numerical solution of CTMCs may be carried out through stationary or transient analysis. For computing the stationary metrics, two methods are available: GTH (Grassmann-Taksar-Heyman) and Gauss-Seidel. The transient metrics are obtained through Uniformization (also known as Jensen's method) as default, but the user might also try the 4th-order Runge Kutta method. Sensitivity analysis is also available at CTMC view.

The rate of each transition might be defined by means of polynomial expressions with user-defined variables (called as parameters on Mercury). Parameter names can include greek letters. Besides states and transitions, users can define reward rates assigned to each state. In such a case, CTMCs become Markov Reward Models.

For models that have absorbing states, Mercury also enables computing probability of absorption and mean time to absorption. The user may create custom metrics with expressions that include state probabilities. Parameters and metrics are easily viewed and modified in the CTMC editor.

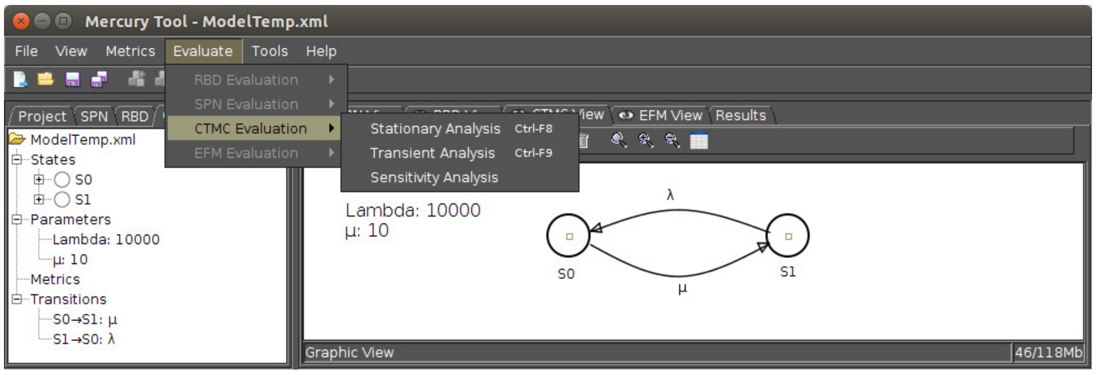


Figure 4: Mercury Tool - CTMC view

1.2.3 DTMC View

The DTMC view (Figure 4) provides features for drawing and evaluating Continuous time Markov chains. The numerical solution of DTMCs may be carried out through stationary or transient analysis. For computing the stationary metrics, two methods are available: GTH (Grassmann-Taksar-Heyman) and Gauss-Seidel.

Parameter names can include greek letters. Besides states and arcs, users can define reward rates assigned to each state. In such a case, DTMCs become Markov Reward Models.

For models that have absorbing states, Mercury also enables computing probability of absorption and mean time to absorption. The user may create custom metrics with expressions that include state probabilities. Parameters and metrics are easily viewed and modified in the DTMC editor.

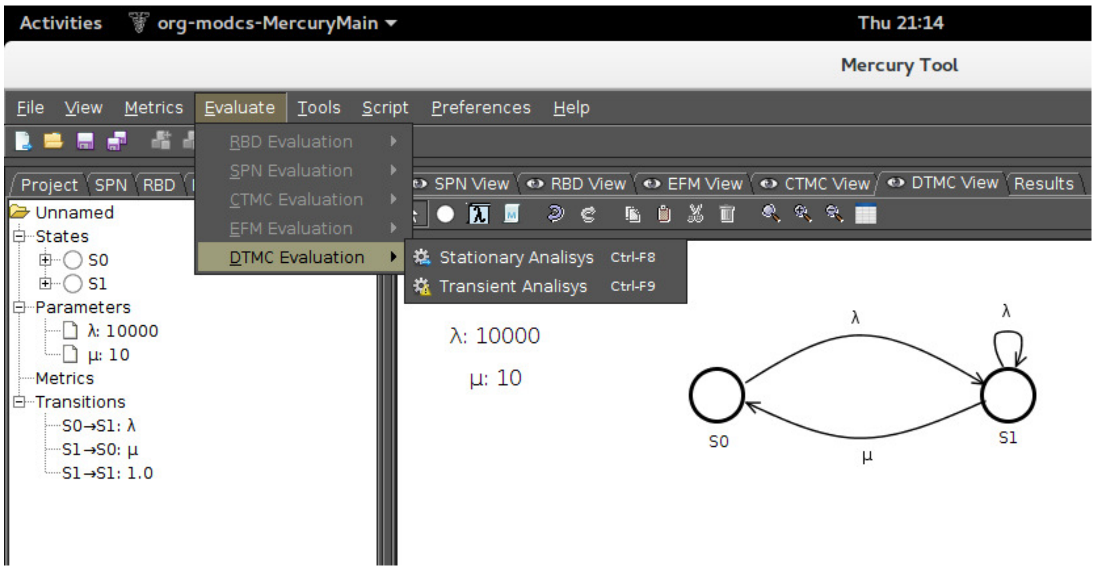


Figure 5: Mercury Tool - DTMC view

1.2.4 RBD View

Through RBD view, the user can perform reliability and availability analysis on large and complex systems using block diagrams. The types of reliability configurations supported by Mercury's RBD module are Series, Parallel, K out of N and Bridge. In addition, users can add labels and carry out experiments for a given component individually.

RBD allows one to represent component networks. It also provides the solution by closed form equations so that the results are usually obtained faster than simulation or numerical solution of other models. However, there are many situations (e.g., when there are dependency among the components) in which modeling using RBD is harder than modeling in SPN or CTMC. The Figure 6 shows the RBD view of Mercury.

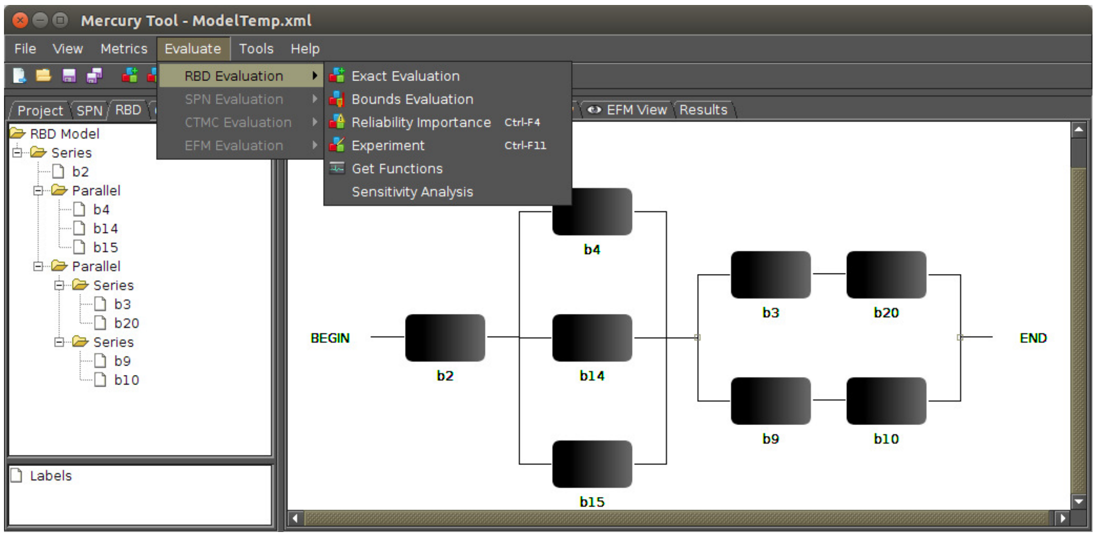


Figure 6: Mercury Tool - RBD View

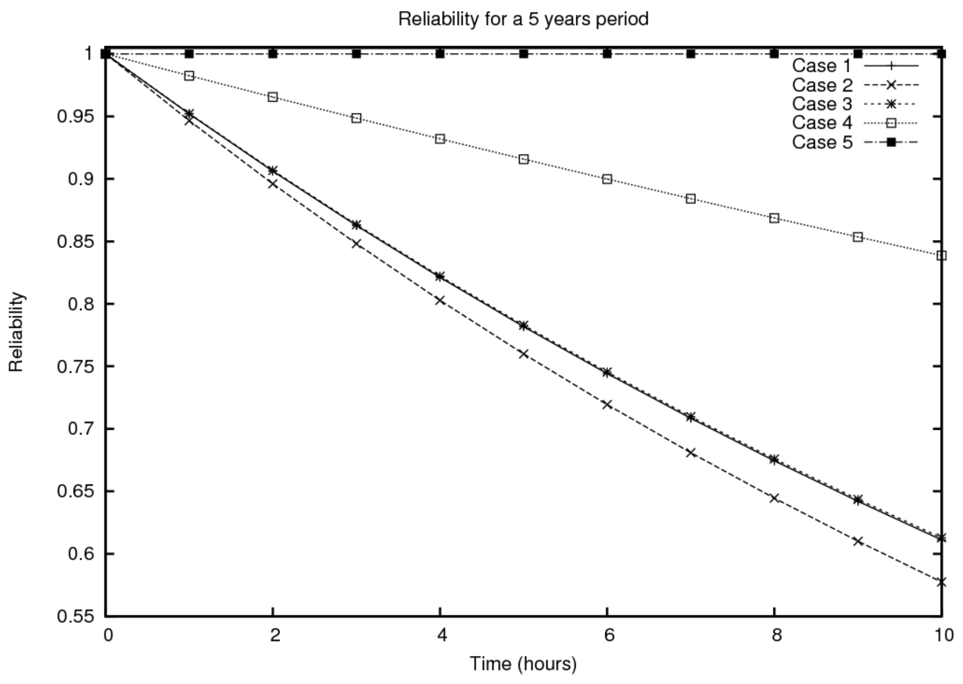


Figure 7: Reliability for a 5 years period

1.2.5 EFM View

The EFM models represent the energy flow between system components in terms of the respective efficiency and the maximum energy that each component can provide (for electrical devices) or the maximum cooling capacity (for cooling devices). Figure 8 depicts an example of an EFM model.

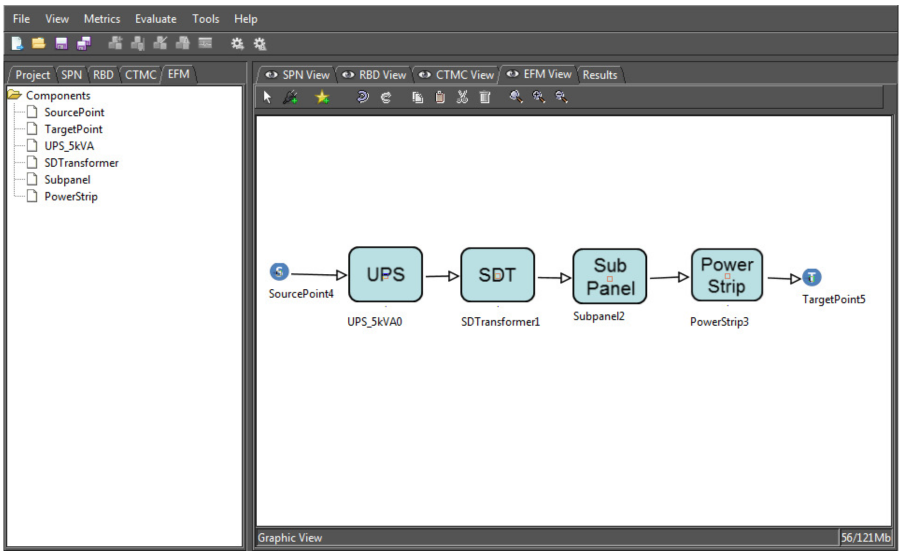


Figure 8: EFM View.

1.2.6 FT View

Fault Trees (FT) and RBDs models differ from each other in their purpose, while the FT focus on the events that lead something to enter into the failure node the RBDs focus on the events that make each block to be working.

In the FT view, the standard model presents a top event called **FAILURE**; this failure has the word **underfined** as a description since no event leads to It. The Figure 120 shows the FT view start screen.

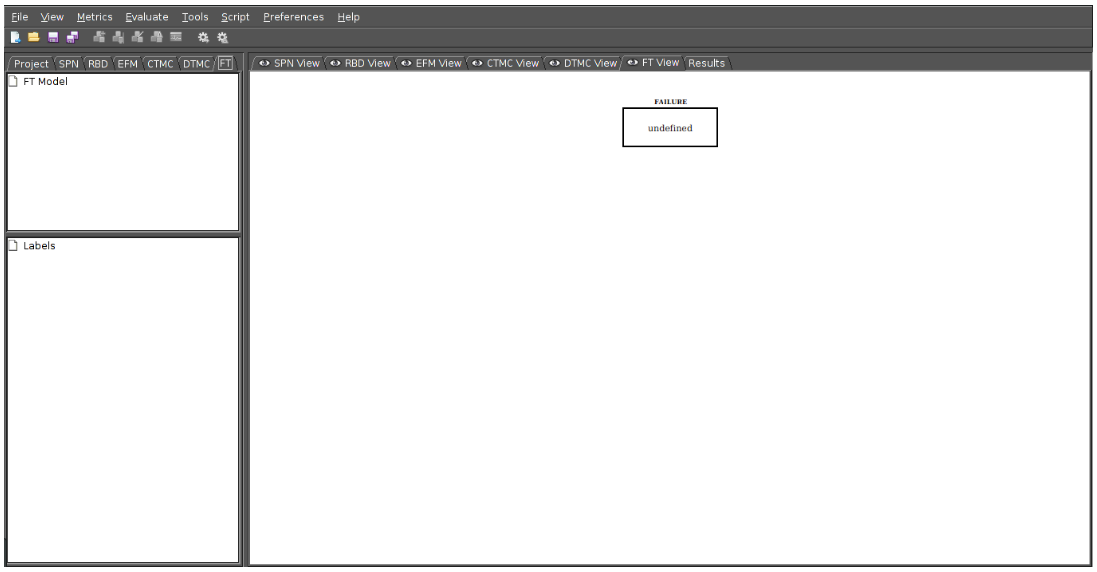


Figure 9: Initial FT Model

System Requirements

In order to run the Fault Tree view over Linux based distros an additional software is required: The JavaFX, that can be download through this **page** or install JavaFX through terminal Linux (sudo apt-get install javafx). For Microsoft Windows systems there is no need to install any additional package.

1.3 Description of menus

Mercury has been developed in Java language, which provides platform independence. The graphical interface allows one to model systems through one or more views: SPN, RBD, EFM, CTMC, DTMC or FT, wheres auxiliary tools (e.g., random variate generation) are also available. Thus users can adopt the most appropriate view according to their needs.

In addition, Mercury tool also provides an interesting functionality that allows importing models built in other engines that adopt the ".TN" standard format (used in tools such as TimeNet [1]). Moreover, there is also an option to export the models created on Mercury to a ".TN" file that is compliant to the standard. All projects developed in Mercury are saved in a ".dpn" file, that is a specific XML file for this tool and stores all information related to the created models.

1.4 Menus

This section details the functionality of each entry on the main menu, that is shown in Figure 10.

In order to activate each item of the menu, the user should click with the left mouse button in the menu item desired. After the click, a window will open showing other available options. Moreover, each menu item also has

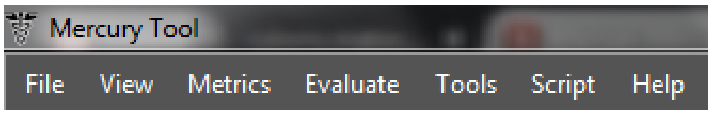


Figure 10: Menu Interface.

a keyboard shortcut. For example, Figure 11 presents the items in "File" menu, with each respective shortcut. These items are explained below:

- **New.** It is used to create a new project, in which all modeling views are initialized empty, except by the RBD view, which starts with one component. The keyboard shortcut: **Ctrl + N**.
- **Open.** It opens a previous project already created on Mercury. The tool only allows to open files with ".dpn" extension. In order to open files of ".TN" format, you must go to the "Import TN File" item. The keyboard shortcut for Open: Ctrl + O.
- **Open Recent** It shows a list of the five most recent projects created or opened on Mercury. The keyboard shortcut: Ctrl + R.
- **Save.** It is used to save the current project. At the first time a model is saved, a window appears for choosing the location and typing the name the model's file. The keyboard shortcut: Ctrl + S.
- **Save as.** It is used to save the model in the ".dpn" format, but with a different name or location than currently saved file. The keyboard shortcut: Ctrl + Shift + S.
- **Import TN File.** It is used to import models previously created in the ".TN" standard. The keyboard shortcut: Ctrl + I.
- **Export TN File.** It is used to export the current project to the ".TN" standard. The keyboard shortcut: Ctrl + E.
- **Close.** It is used to close the tool. The keyboard shortcut: Ctrl + Q.

In the menu **View**, the user can enable or disable the four views provided by Mercury tool (SPN, RBD, EFM, CTMC, DTMC, FT). Figure 12 shows the menu **View**. Moreover, the Figures 13 and 14 illustrate the four views enabled and disabled, respectively.

Figure 16 presents the menu **Tools** and their features are described as follows:

Figure 15 depicts the menu **Evaluate**, which has items for each model supported in Mercury. An item is enabled only when the corresponding model view is selected in the drawing area. The specific model evaluation items are detailed in the other sections ahead.

RVG stands for **random variate generator**. It is an additional tool responsible for generating random numbers, which contemplates several probability distributions. Statistical summaries are also provided for the generated numbers (e.g., standard deviation) and the data can be exported to files in order to be utilized

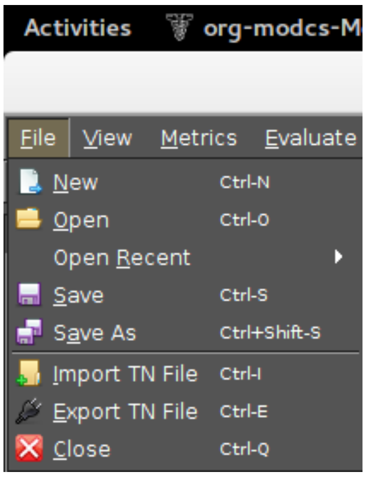


Figure 11: Menu File.

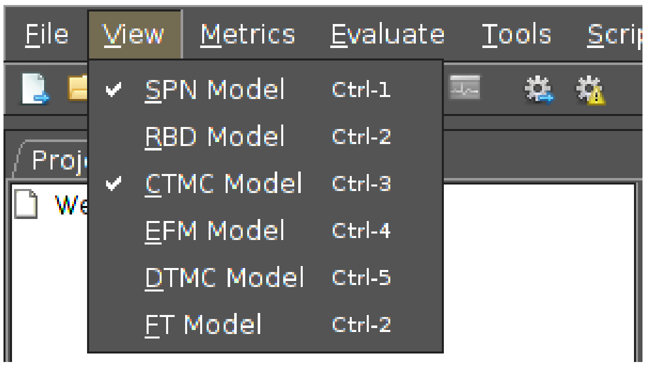


Figure 12: Menu View.

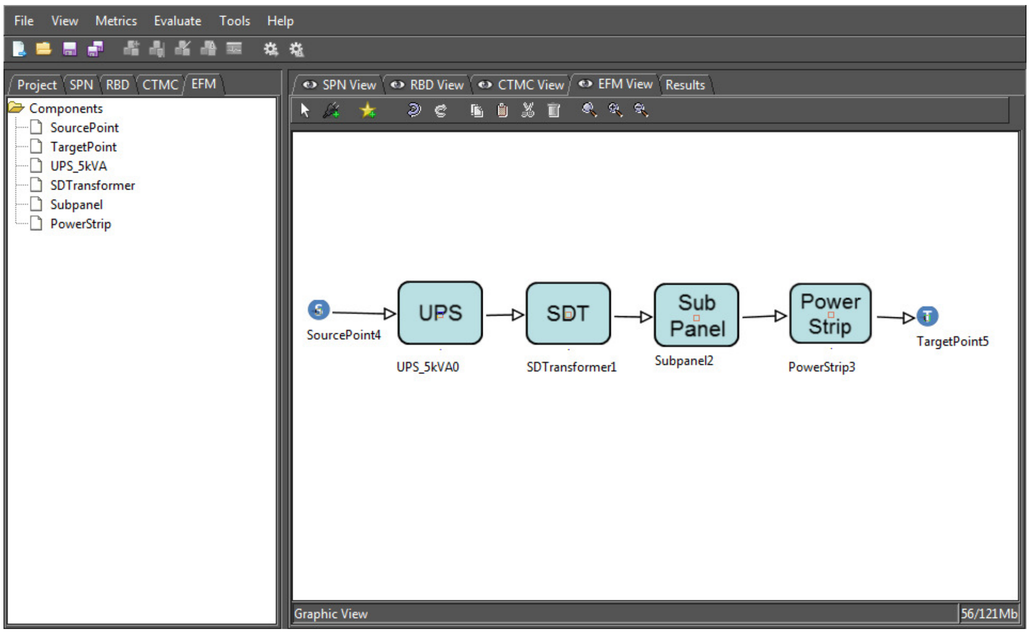


Figure 13: All four views enabled.

for external applications. Moreover, the generator is utilized by the SPN simulation, in which the user can choose different probability distributions associated to transitions.

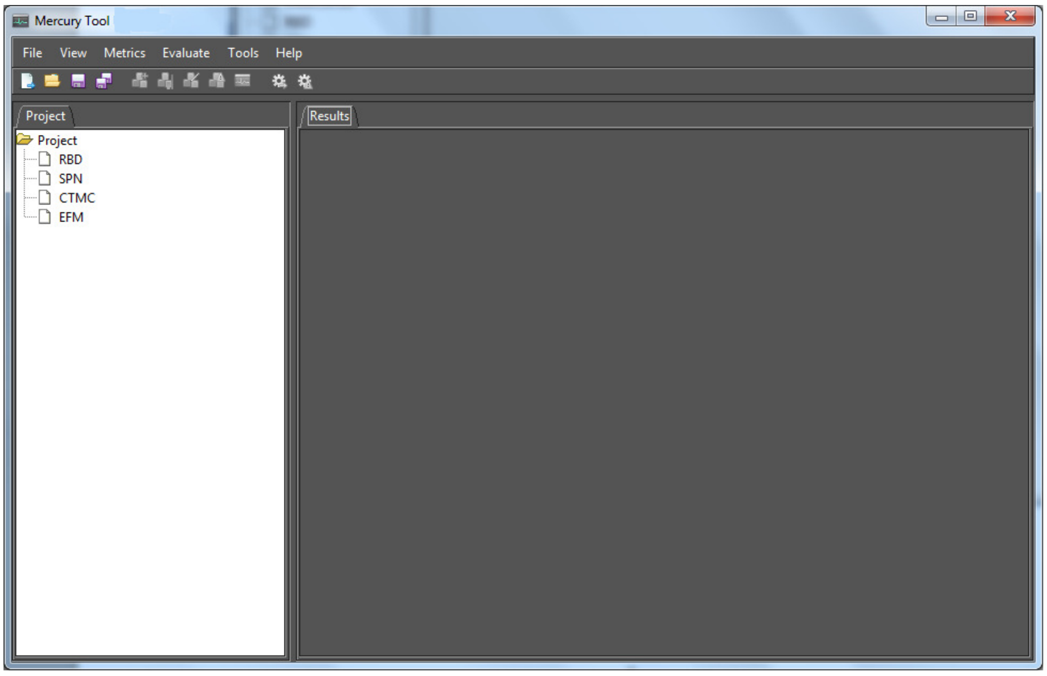


Figure 14: All four views disabled.

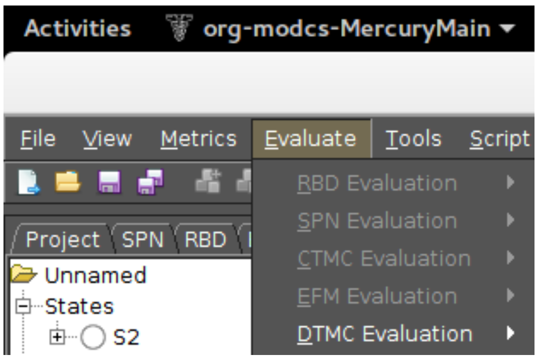


Figure 15: Menu Evaluate.

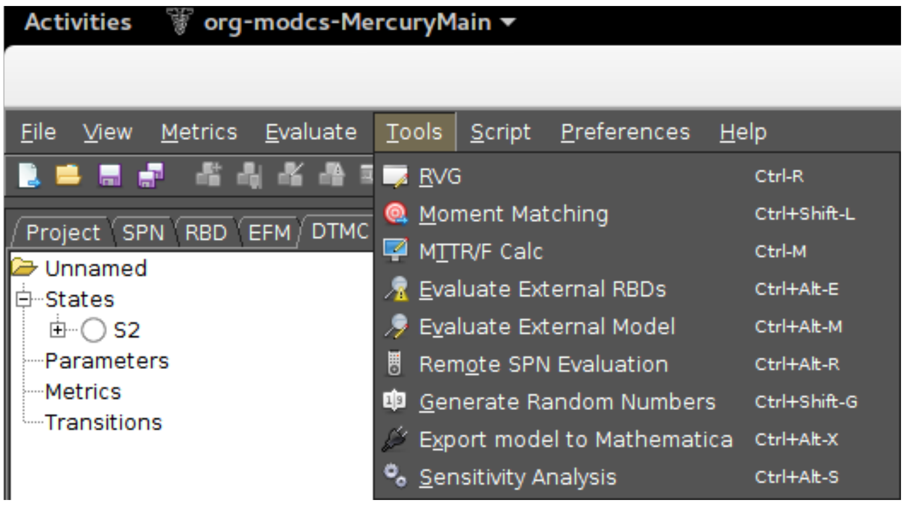

Activities
org-modcs-MercuryMain
File
View
Metrics
Evaluate
Tools
Script
Preferences
Help
Project
SPN
RBD
EFM
DTMC
Unnamed
States
S2
Parameters
Metrics
Transitions
RVG Ctrl-R
Moment Matching Ctrl+Shift-L
MTTR/F Calc Ctrl-M
Evaluate External RBDs Ctrl+Alt-E
Evaluate External Model Ctrl+Alt-M
Remote SPN Evaluation Ctrl+Alt-R
Generate Random Numbers Ctrl+Shift-G
Export model to Mathematica Ctrl+Alt-X
Sensitivity Analysis Ctrl+Alt-S

Figure 16: Menu Tools.

Moment matching [2] is another important auxiliary tool presented in Mercury environment. By adopting this tool, users can estimate which exponential-based probability distribution best fits the mean (first moment) and standard deviation (second moment) of user data for a given model parameter.

MTTR/F Calculator allows one to compute the MTTR (Mean Time To Repair) and MTTF (Mean Time To Failure) of the system in analysis by using availability and reliability curve as input parameters. The reliability is a time-dependent metric that gives the probability that an item will perform a required function, under stated conditions, for a stated period of time.

The system reliability $R(t)$ can be defined as follows [3]:

$$R(t) = exp\Big[\int_0^t \lambda(t)dt\Big]$$

where $\lambda(t)$ corresponds to the failure rate over the time t. However, if $\lambda(t)$ is constant the reliability can be evaluated as,

$$R(t) = e^{-\lambda t}.$$

For instance, suppose the system failure rate is $0.5[h^{-1}]$, then the reliability curve should be the following (Figure 1.4).

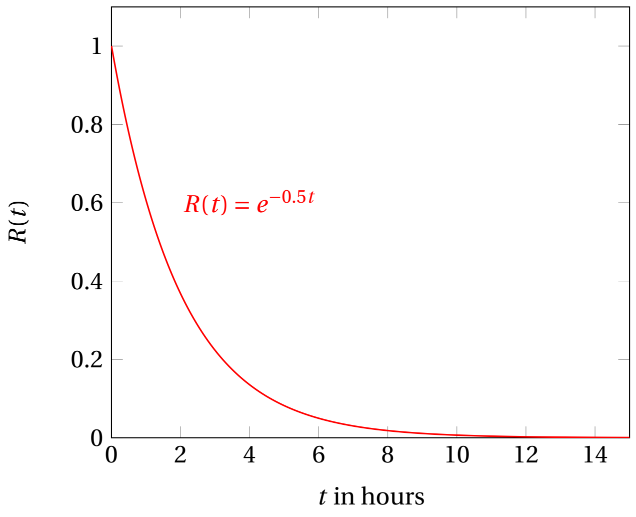


Figure 17: Reliability over time

The system MTTF and MTTR can be calculated using the following expressions [3]:

$$MTTF = \int_0^\infty R(t)dt$$

$$MTTR = \frac{MTTF}{avaialbility} - MTTF$$

Figure 18 shows the MTTR/F Calculator window, in which users should specify the *Availability* and a CSV file with no headers and two columns specifying the evaluation time and reliability value. Button *File* allow user to point to the file with time and reliability values.

After specifying the parameters, the user should compute the results by pressing the button *Calculate*. Figure 19 shows an example of the result obtained after computing MTTR and MTTF values. For a more detailed example, please refer to the following video *https://youtu.be/Hmu5DX3CJCg* .

Evaluate External RBDs is a tool for computing availability metrics from external RBD files, created following a specific format

Export model to Mathematica is an option for saving CTMC models in Wolfram Mathematica language.

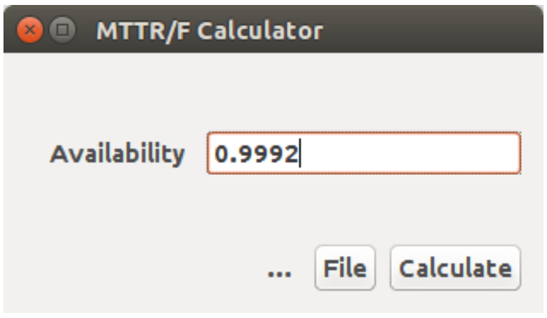
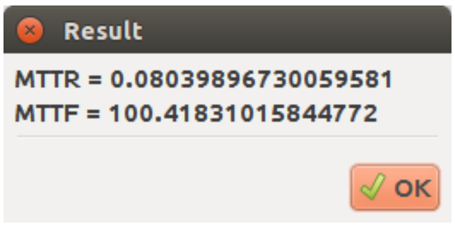


Figure 18: MTTR/F Calculator



```
MTTR = 0.08039896730059581
MTTF = 100.41831015844772
```

Figure 19: MTTR/F Calculator Results

1.4.1 Command Buttons

Command Buttons correspond to the buttons present on the top of the Mercury tool's window, right below the menu bar. These buttons are important and frequently used to create new projects or to save the current project, for example. Figure 20 shows the command buttons, in which each button is represented as an icon. The following items describe the functionality of each command buttons.

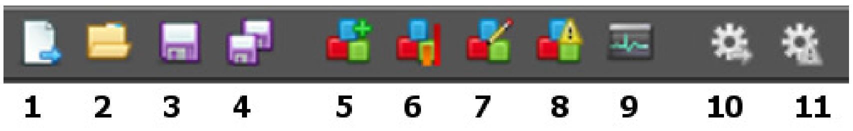
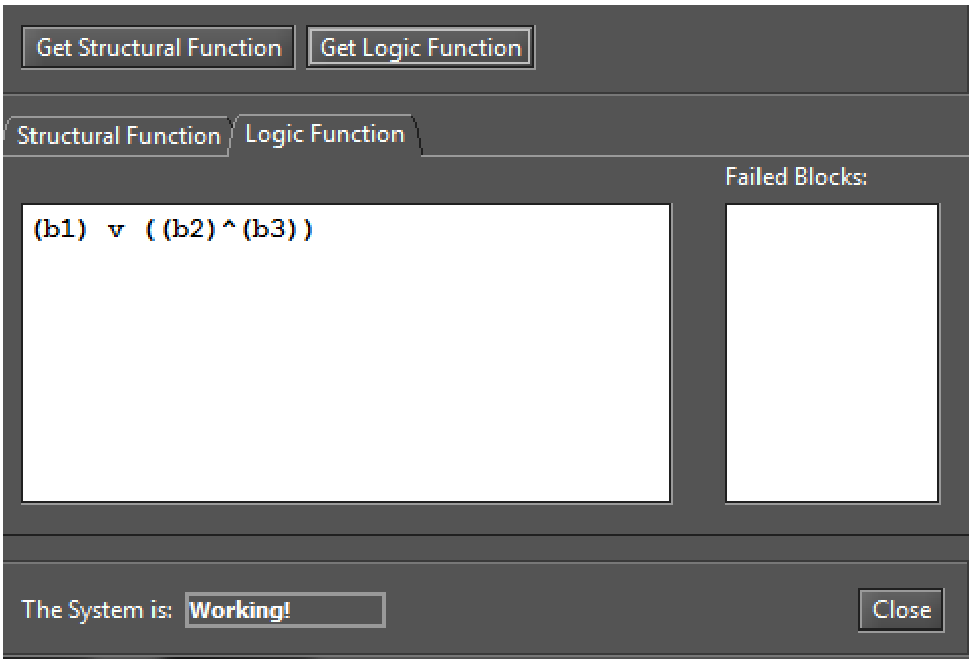


Figure 20: Command Buttons.

- **New (1):** After selecting this button, the user can create new projects. The drawing area is initially created empty, and so, models can be built. Shortcut: Ctrl+n.
- **Open (2):** In order to open a saved model, users should press the *open* button. Once pressed this button, a dialog window opens allowing users to open a saved model. Shortcut: Ctrl+o.
- **Save (3):** This button saves the changes of current project considering the model name shown in the title bar of the editor window. Shortcut: Ctrl+s.
- **Save As (4):** This button saves current model considering a new name, which can be typed in a subsequent dialog window. Shortcut: Ctrl+shift+s.
- **RBD Evaluation (5)** This button evaluates the model. Is only enabled when the program is in the RBD vision.
- **RBD Bounds Evaluation (6)** This button performs the bounds evaluation on the model. Is only enabled when the program is in the RBD vision.
- **RBD Experiment (7)** This button performs a Experiment. Is only enabled when the program is in the RBD vision.
- **Component Importance (8)** This button calculates the component importance on the model. Is only enabled when the program is in the RBD vision.
- **Functions (Structural and Logic) (9)** This button is used to get the functions of the system. The Structural Function is a function related to the states of the components. The system and each component may be in working or failed state. The system state is a binary random variable which is determined by the component states. If the states of the components are know, the system state is also know. The state can be toggled via the menu properties on the blocks and when the state of the block is faulty, the component is indicated by a broken icon. To get the functions, the user should click on this button and choose one option to show between logic or structural. The Figure 21 shows a example of a logic function of a system without faulty blocks.
- **Stationary Simulation (10)** This button starts the SPN stationary evaluation through the simulation of the model behavior. Is only enabled when the program is in the SPN vision.
- **Transient Simulation (11)** This button starts the SPN transient evaluation through the simulation of the model behavior. Is only enabled when the program is in the SPN vision. Only single metrics related to Probability and Expectation are accepted when transient simulations are adopted. For instance, metrics like P#P0=1 and E#P1 are accepted. However, metrics like (P#P0=1*2)/E#P3 and E#P1*3 are not allowed in transient simulations.



(b1) v ((b2)^(b3))

The System is: Working!

Figure 21: Mercury Tool - Structural Function

1.4.2 SPN Buttons

SPN buttons are the ones used for modeling GSPN nets. The icons (see Figure 22) are present in the object area on the Mercury tool main window. It is important to state that to view the GSPN buttons it is necessary to enable the GSPN view. Moreover, each button have specific functionality to model the GSPN nets and the descriptions related to them are detailed as follows.

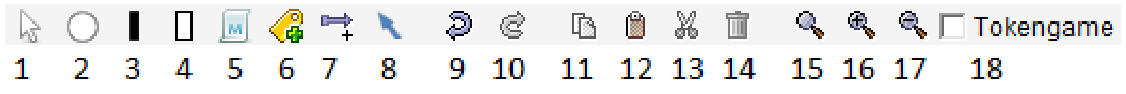


Figure 22: GSPN Buttons.

- **Default Cursor(1):** This generic button is used to activate the selection cursor mode.
- **Add Places(2):** Responsible for adding place components in the drawing area. In order to do this, the user should click on the Add Place button and after, on the desired place location in the drawing area.
- **Add Immediate transition(3):** This button is responsible for adding immediate transition components in the drawing area. In order to do this, the user should click on the Add Immediate transition button and after, on the desired location in the drawing area.
- **Add Exponential transition(4):** This button adds exponential transition components in the drawing area. For this, the user should click on the Add Exponential transition and after, on the desired location in the drawing area.
- **Add Measure(5):** Creates dependability measures.

- **Insert New Label(6):** This button is responsible for adding labels in the model. In order to do this, the user should click on the Insert New Label button and after, on the desired location.
- **Manager connecting components(7):** Allows to connect the components in the drawing area. For this, it is necessary to click and drag the arc from the source component to the next component.
- **Arc type(8):** Allows the user to choose the type of arc that will be used to model GSPN nets.
- **Undo(9):** cancel the recent actions in the drawing area. All recent changes are stored and can be rolled back, one after one. Keyboard shortcut: `Ctrl+Z`.
- **Redo(10):** Remake the last change in the drawing area. Keyboard shortcut: `Ctrl+`.
- **Copy(11):** Copies the currently selected model object. Keyboard shortcut: `Ctrl+C`.
- **Paste(12):** Insert the model component from the internal buffer to the current model. After added the component, it can be easily moved to the desired position. Keyboard shortcut: `Ctrl+V`.
- **Cut(13):** Remove the model component putting it in the internal buffer for further use. Keyboard shortcut: `Ctrl+X`.
- **Delete(14):** Remove definitely the model component.
- **Standard Scale(15):** Apply the standard scale in the drawing area.
- **Scale up Image(16):** Each click scales up the drawing image by 10% percent (zoom in).
- **Scale down Image(17):** Each click Scales down the drawing image by 10% percent (zoom out).
- **Token Game(18):** This functionality can be used to comprehend the behavior of the Petri model. When this option is enable, the user will interact with the model through firing the transitions. The places shown in the SPN view area contain their respective number of tokens in the current state, and enabled transitions flash. Double-clicking an enabled transition with the left mouse button causes it to fire, changing the marking consequently. This functionality is useful for debugging a model. Thus, the user can check whether it works as it is supposed to.

1.4.3 EFM Buttons

These buttons are present on the EFM view of Mercury Tool main window. Figure 22 shows the buttons and their description are detailed below.

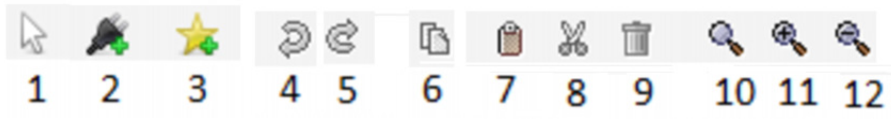


Figure 23: EFM Buttons.

- **Default Cursor(1):** This is a generic button to activate the selection mode.
- **Insert Component(2):** It adds data center components to the canvas. In order to do this, users should have power or cooling view selected.
- **Add new Component(3):** Add new data center components to the project. This button's click opens a window (see Figure 24) that allows the modeler to add components to the current project. In order to accomplish this, users should select the component as well as insert its dependability and values *MTTF* and *MTTR* as shown, the Sustainability Parameters and set if the component is working or not. Moreover, the added components will be available on left side as shown in Figure 25.

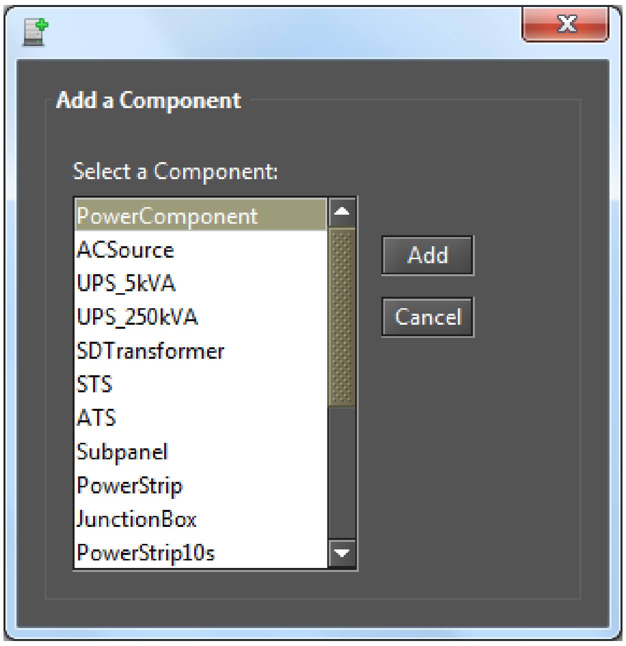


Figure 24: First Step - Add components in the EFM View.

- **Undo(4):** Undo the last changes in the drawing area.
- **Redo(5):** Remakes the last changes in the drawing area.
- **Copy(6):** Copies the currently selected model object.
- **Paste(7):** Inserts the model component.
- **Cut(8):** Removes the model component.
- **Delete(9):** Delete the selected component.
- **Standard Scale(10):** Applies the standard scale in the drawing area.
- **Scale up Image(11):** Each click scales up the drawing image by 10% percent (zoom in).
- **Scale down Image(14):** Each click Scales down the drawing image by 10% percent (zoom out).

1.5 Drawing Area

Mercury Tool has four views for modeling (SPN, RBD, CTMC, DTMC and EFM) and also another one for displaying the results (Results View). It significant to state that the user should click on the desired view tab to enable its visualization. Moreover, the views can be enabled or disabled. For this, the user should mark or cancel the mark of each view present on the Menu View.

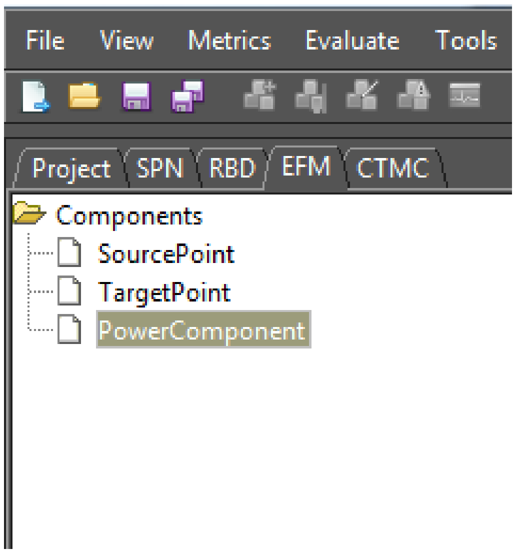


Figure 25: A new component is available.

In addition, on the left side of the tool there is a tab (See Figure 25), in which all components that are available in current project can be accessed or only visualized. Moreover, these components are divided by their current views (SPN, RBD, EFM, CTMC, DTMC and FT). In other words, the components of a SPN model are present on the tab of SPN components, for example.

2 SPN Modeling and Evaluation

Regarding the SPN view, the users can create models by adding components such as places and immediate or stochastic transitions. Moreover, the user can edit some properties (e.g., component names) related to the model. Figure 26 shows a model in which there are one timed and one immediate transitions. In order to view the component's properties, the user should press the right mouse button on the desired component as depicted on Figure 27. The properties are respectively: Name, Delay, Server Type and Probability Distribution. Moreover, depending on the chosen distribution, other properties will be available. The following lines detail those properties.

- **Name:** Component name.
- **Delay:** Transition Delay.
- **Server Type:** Firing Semantics.
- **Probability Distribution:** There are Normal, Erlang, Exponential and Empirical distributions available.

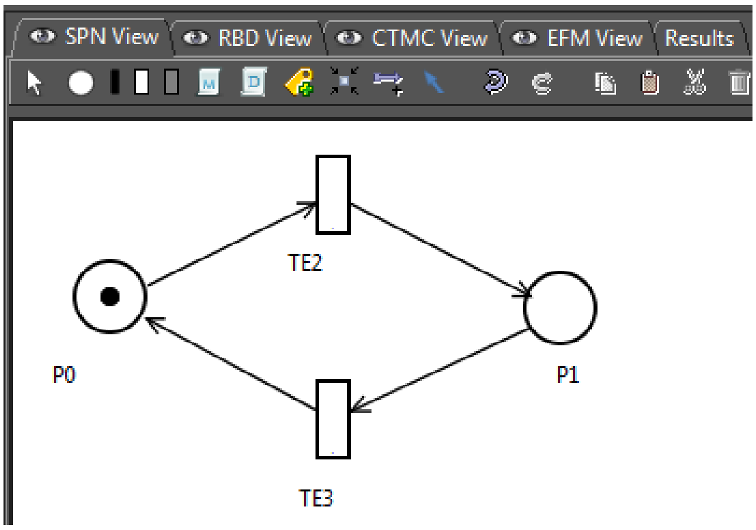


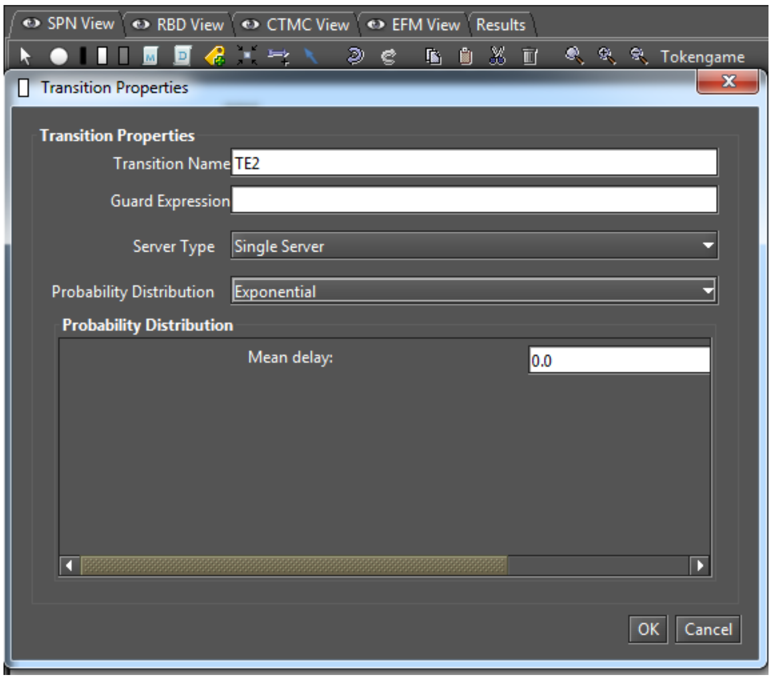
Figure 26: GSPN Model

In addition, Figure 28 shows the properties (e.g., Name and Marking) of a component place.

- **Name:** Component name.
- **Marking:** The number of marks (e.g., tokens) in the place.

Properties of the immediate transitions are detailed as follows. The properties are respectively: Name, Priority, Weight and Enabling Function. Figure 29 illustrated these properties.

- **Name:** Component name.
- **Priority:** It is possible to set the transition priority.



Transition Properties

Transition Name TE2

Guard Expression

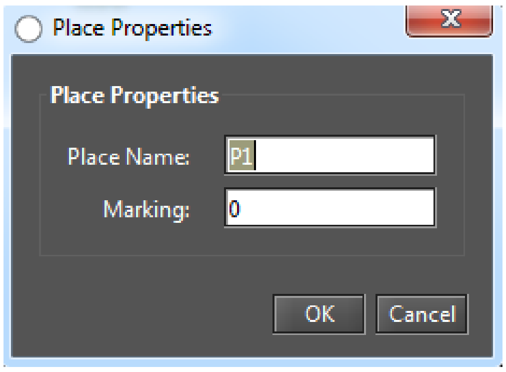
Server Type Single Server

Probability Distribution Exponential

Probability Distribution

Mean delay: 0.0

Figure 27: GSPN Model properties



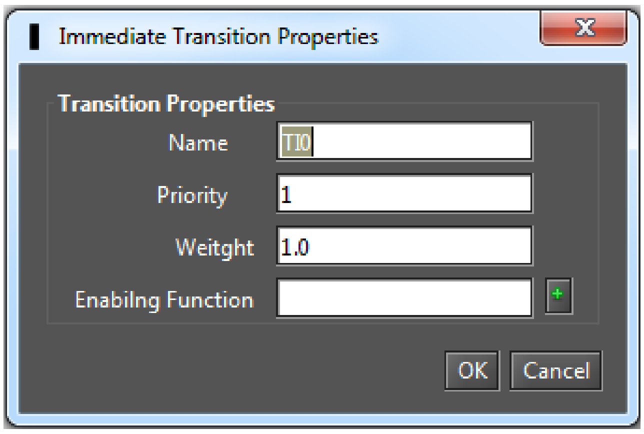
Place Properties

Place Name: P1

Marking: 0

Figure 28: GSPN Model properties

- **Weight:** It is adopted to set the weight on the transitions.
- **Enabling Function:** the function that enables this transition.



Transition Properties

Name TI0

Priority 1

Weitght 1.0

Enabilng Function

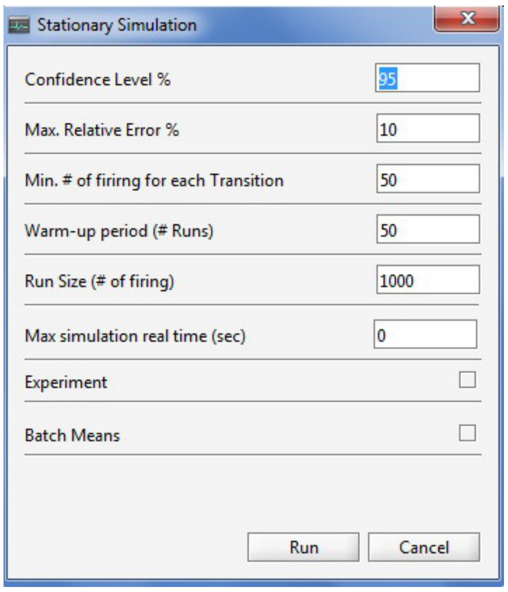
Figure 29: Immediate transition properties

In order to evaluate SPN models, the tool provides stationary and transient simulation, as well as stationary and transient analysis. These features are available in the menu “Evaluate”, option “SPN Evaluation“, or their respectively Command Buttons.

2.1 SPN Simulation

Figure 30 shows the Stationary Simulation input window and its input values are detailed as follows.

- **Confidence Level:** Sets the confidence level.
- **Max. Relative Error:** Sets the maximum percentage of relative error.
- **Min. of firing for each Transition:** Sets the minimum number of fires of each transition.
- **Warm-up:** Sets the Warm-up period (in number of runs).
- **Run Size:** The size of each run.
- **Max simulation real time:** It is used to set the maximum simulation runtime (sec).
- **Experiment:** Use or not the labels of the model.
- **Batch Means:** Use or not the method Batch Means when evaluate the model. This method remove the initial transient period.


Stationary Simulation

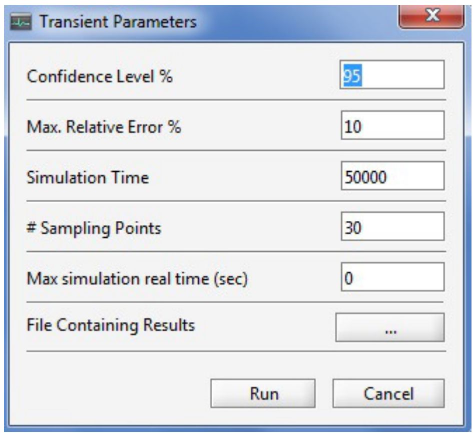
Confidence Level %	95
Max. Relative Error %	10
Min. # of firirng for each Transition	50
Warm-up period (# Runs)	50
Run Size (# of firing)	1000
Max simulation real time (sec)	0
Experiment	☐
Batch Means	☐

Figure 30: Stationary Simulation

In addition, Figure 31 shows the Transient Simulation input window. The parameters are detailed as follows.

- **Confidence Level:** Sets the confidence level.
- **Max. Relative Error:** Sets the maximum percentage of relative error.

- **Simulation Time:** Sets the simulation time unit (e.g., 8760 representing on year).
- **Sampling Points:** Sets the number of sampling points that will be considered in order to obtain the results.
- **Max simulation real time:** It is used to set the maximum simulation runtime (sec).
- **File Containing Results:** Select the file to save simulation results.


Transient Parameters

Confidence Level %	95
Max. Relative Error %	10
Simulation Time	50000
# Sampling Points	30
Max simulation real time (sec)	0
File Containing Results	...

Figure 31: Transient Simulation

2.2 SPN Analysis

The **Stationary Analysis** and **Transient Analysis** windows enable computing results through the generation of the underlying Markov chain that corresponds to the Stochastic Petri Net created by the user. Figure 32 shows the Stationary Analysis window, which computes steady-state metrics, useful for assessing the long-term average behavior of modeled system. This window has a combo box for selecting one of two solution methods available: **Direct - GTH** (Grassmann-Taksar-Heyman) or **Iterative - Gauss-Seidel**.

When solving the model through GTH, the user can change the **maximum error** used in the algorithm. The default value for the maximum error is 0.0000001 (10^-7). By clicking on the **Run** button, the solution algorithm is triggered and as soon as it finishes, the user-defined measures are updated on screen. Their values are shown in the drawing area, as illustrated in Figure 81, where the measure called **Availability** is presented on the right side.

Mercury also generates a plain text file with the results. The name of the result file is the Mercury model file name appended with the " "-StationaryAnalysis.txt" suffix.

When solving the model through Gauss-Seidel, besides maximum error, the user can also change the maximum number of iterations. The default value for such a parameter is -1, indicating that the algorithm only stops when convergence of results is reached, considering the informed error.

The SPN model can also be solved for a range of values for a given delay definition. This is accomplished by clicking in the **Experiment** check box on Stationary Analysis window and then on the **Run** button. For more

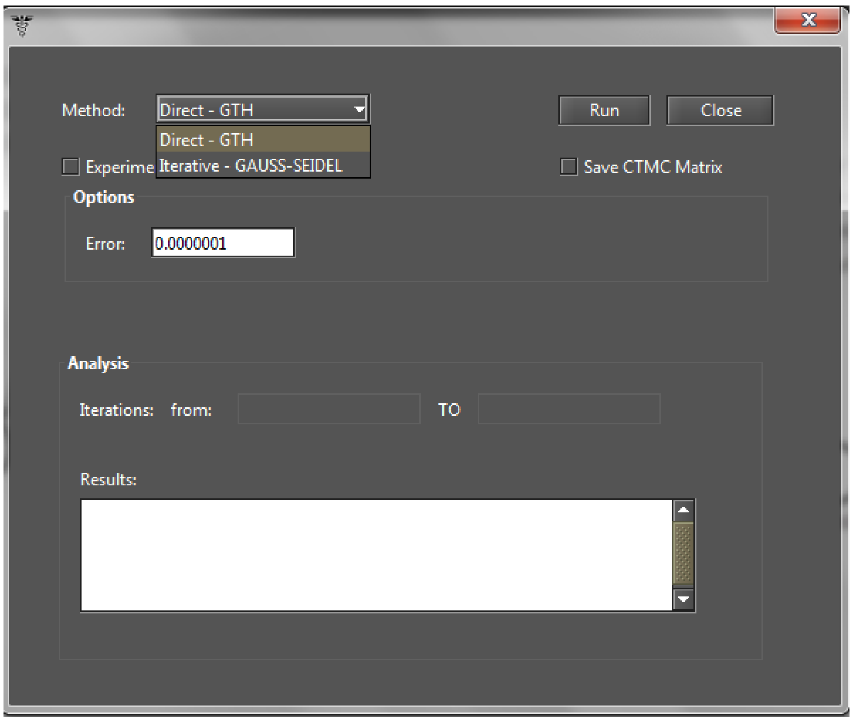


Figure 32: Stationary analysis window

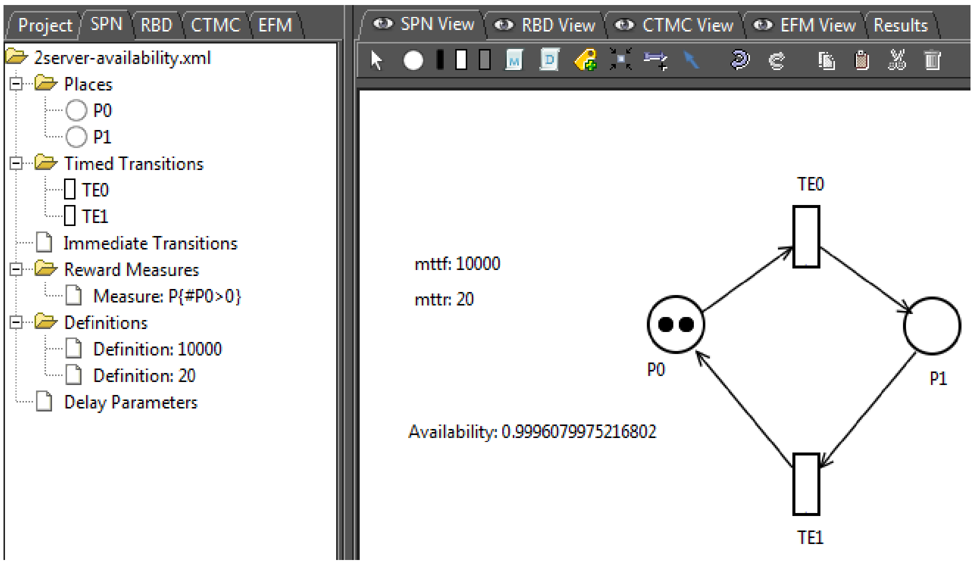


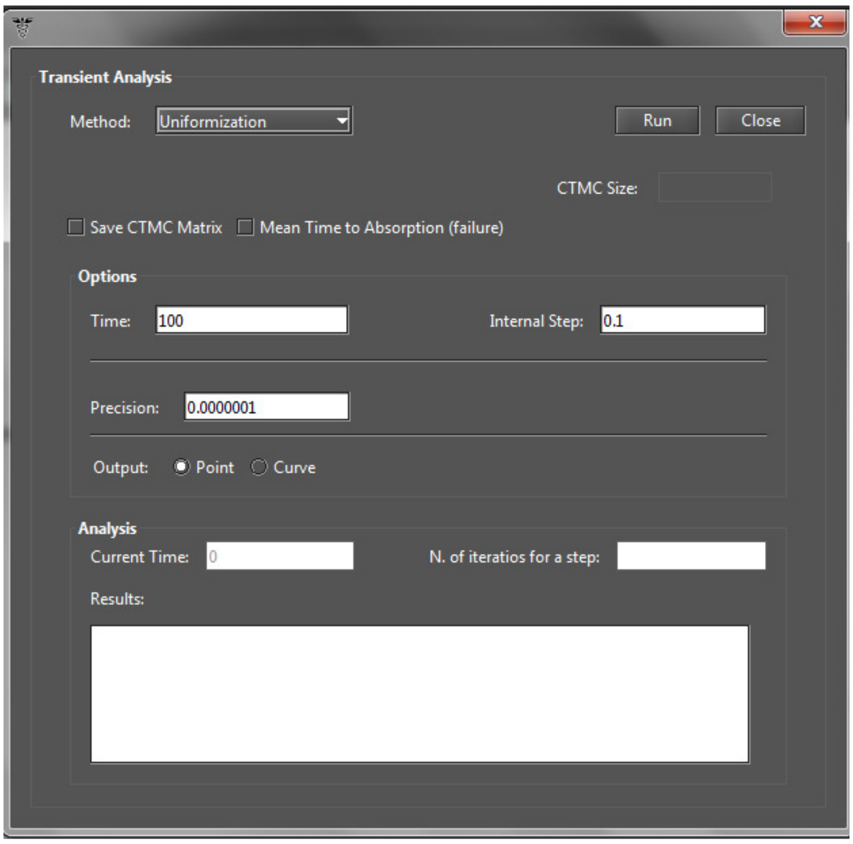
Figure 33: SPN and its metrics after model solution

details on the Experiment option, check the Section 4.3, that describes the Stationary Analysis for CTMC models.

Figure 34 shows the Transient Analysis window, which enables computing time-dependent results, useful for assessing the behavior of modeled system in a particular point in time. This window has a combo box for selecting one of two solution methods available: **Uniformization** (also known as Jensen's method) or **Runge-Kutta (4th order)**.

When solving the model through Uniformization, the user can define:

- The **time** for which the analysis will be carried out (default: 100);


Transient Analysis

Method: Uniformization

CTMC Size:

☐ Save CTMC Matrix ☐ Mean Time to Absorption (failure)

Options

Time: 100 Internal Step: 0.1

Precision: 0.0000001

Output: ● Point ○ Curve

Analysis

Current Time: 0 N. of iteratios for a step:

Results:

Figure 34: Transient analysis window

- the **internal step** size (default: 0.1);
- the **precision** of results (default:10¯7);

All those values are used by the Uniformization algorithm. Notice that the delay for obtaining results is proportional to the time informed for the analysis, because Uniformization is an iterative algorithm. The internal step size will affect accuracy of results and also execution time. By clicking on the **Run** button, the solution algorithm is triggered. As soon as it finishes, the measures are updated in the drawing area, and written in a plain text file with the name of Mercury model file appended with the "-TransientAnalysis.txt" suffix.

The **Transient Analysis** window also allows the user choosing between a **"Point"** or **"Curve**'' analysis. The **Point** analysis is the default, and shows results only for the specific point in time. The **Curve** analysis writes in a plain text file all measures computed in intermediate steps from time equals zero until the specified time instant.

The **mean time to absorption** (MTTA) is a metric that can be computed in the Transient Analysis window, by checking the **Mean Time to Absorption (failure)** checkbox. The MTTA value is printed in the results text area.

After reading the SPN modeling section, users can see a video demonstration of Mercury on the following links.

- **SPN Stationary Simulation -** `http://youtu.be/e9EaDjjQJUU`
- **SPN Stationary Analysis -** `http://youtu.be/1Y2ops_foNE`

- **SPN Transient Simulation -** `http://youtu.be/1zN17yT0ud8`
- **SPN Transient Analysis -** `http://youtu.be/UwQC356op1o`

2.3 SPN Structural Analysis

The Mercury tool provides a function to analyze SPNs without generating the reachability graph. The Structural Analysis enables us to prove some properties by means of **invariants** and **traps** techniques. Figure 35 shows how to use this functionality through Mercury menus.

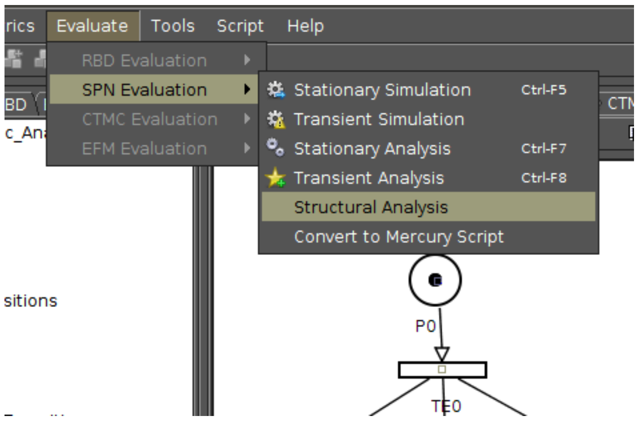
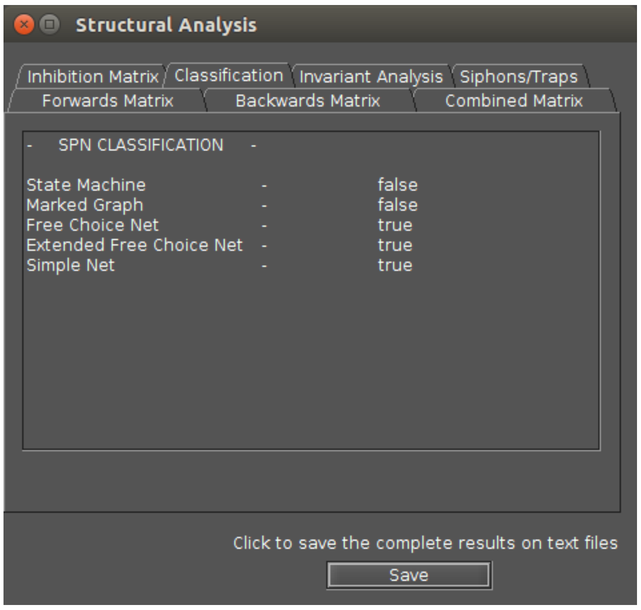


Figure 35: Structural Analysis menu entry.

When finished, the results window will pop-up with various panes showing SPN structural properties: **Forwards Matrix**, **Backwards Matrix**, **Combined Matrix**, **Inhibition Matrix**, **Classification**, **Invariant Analysis**, and **Siphons/Traps**.

Figure 36 shows one of the panes of the Structural Analysis result window. If desired, the user can save all results in a .txt file. To do this, just click **Save** button and select a folder and a file name.



```
-   SPN CLASSIFICATION   -

State Machine              -         false
Marked Graph               -         false
Free Choice Net            -         true
Extended Free Choice Net   -         true
Simple Net                 -         true
```

Figure 36: Structural Analysis results window.

2.4 Token Game

Token Game corresponds to the functionality of the tool in which users can simulate SPN models behavior. In order words, users are able, for instance, to debug the model that is under analysis. Thus, model construction mistakes can be easily discovered as well as their solution. Figure 37 shows an SPN model with the Token Game functionality off.

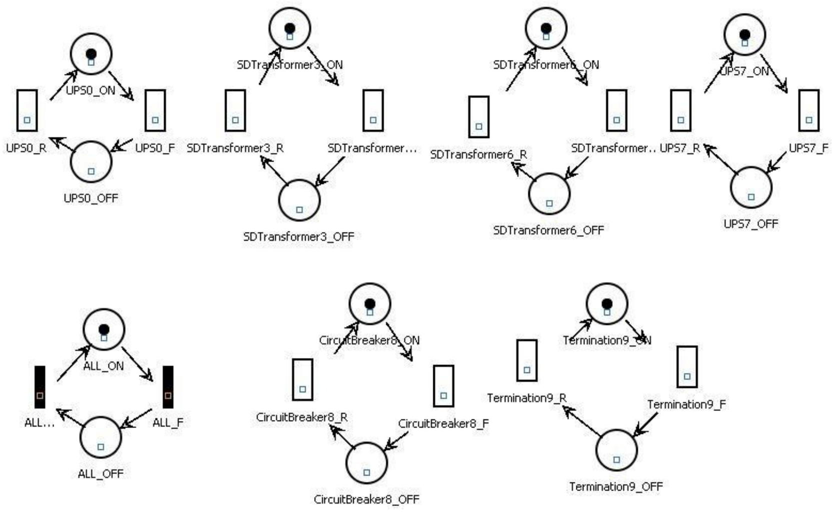


Figure 37: Correspondent SPN model.

Considering the Guard Expressions automatically generated, there is a transition ALL_F that corresponds to the one which represents the system behavior. In other words, a fire of that transition changes the system mode from the operational to the failure one. The guard expression of that transition (ALL_F) is represented as follows.

$((\#Termination9\_ON = 0)OR(\#CircuitBreaker8\_ON = 0)$

OR $(((\#UPS0\_ON = 0)OR(\#SDTransformer3\_ON = 0))$

AND $((\#UPS7\_ON = 0)OR(\#SDTransformer6\_ON = 0))))$

As already mentioned, with the token game functionality, one can visualize the behavior of Petri net models. In addition, users can simulate equipment failures as well as those correspondents consequences on the system availability. In order to start this functionality, users should mark the box related to Token Game as emphasized in Figure 38. Moreover, the reader should observe in Figure 38, that any component can fail through the fire of an enabled transitions (the green's one of that figure).

Considering that the user has fired the $transition\_9F$ (see Figure 39), meaning that the termination has failed. It is possible to observe that there is only one transition ready to fire (ALL_F), meaning that the system changed to the failure mode as expected. The reader should have in mind that this process can be applied to any Petri net model.

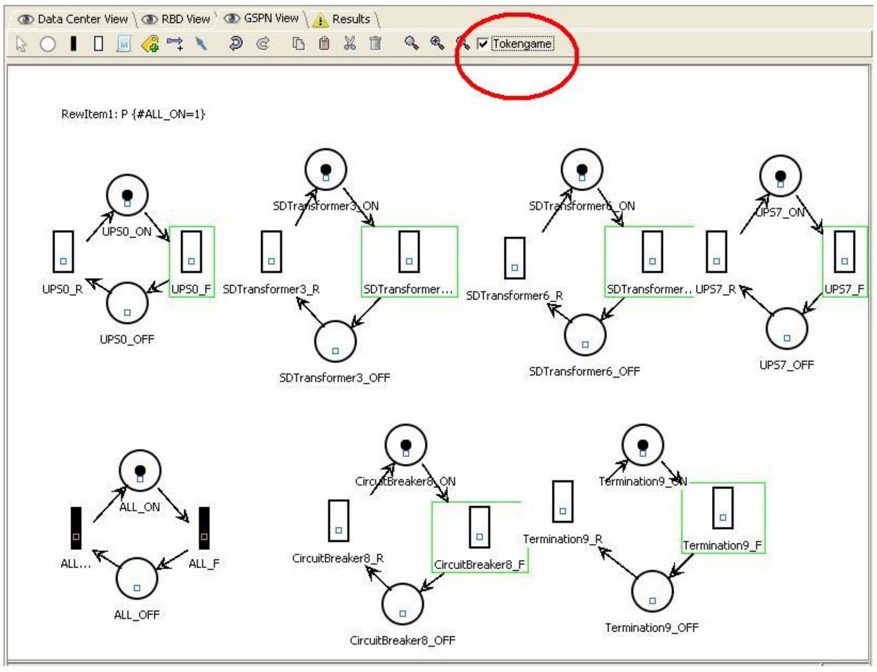


Figure 38: Property Screen Components

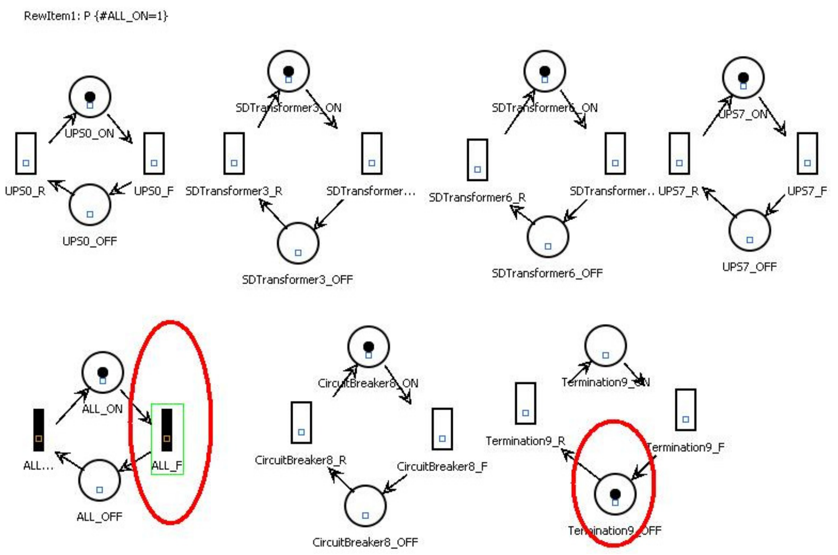
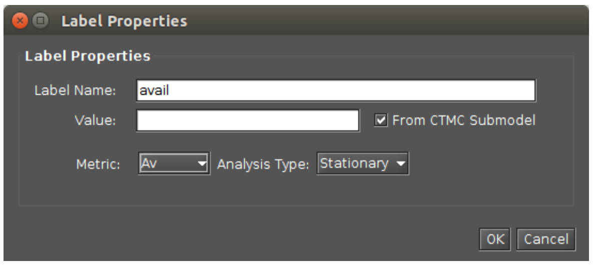


Figure 39: Token Game example.

2.5 Hierarchical composition with CTMC model

The user can to define the delay of transitions in the SPN model by means of metrics computed from a CTMC model, what we call as hierarchical composition. This is done through the inclusion of labels assigned to desired CTMC metric, as shown in Figure 40

The user must check the box "From CTMC submodel", and select one of the metrics previously defined in the CTMC. The type of analysis (Stationary or Transient) must be selected too. In case of transient analysis, the time parameter is also required.



Label Properties

Label Name: avail

Value:

[✓ From CTMC Submodel]

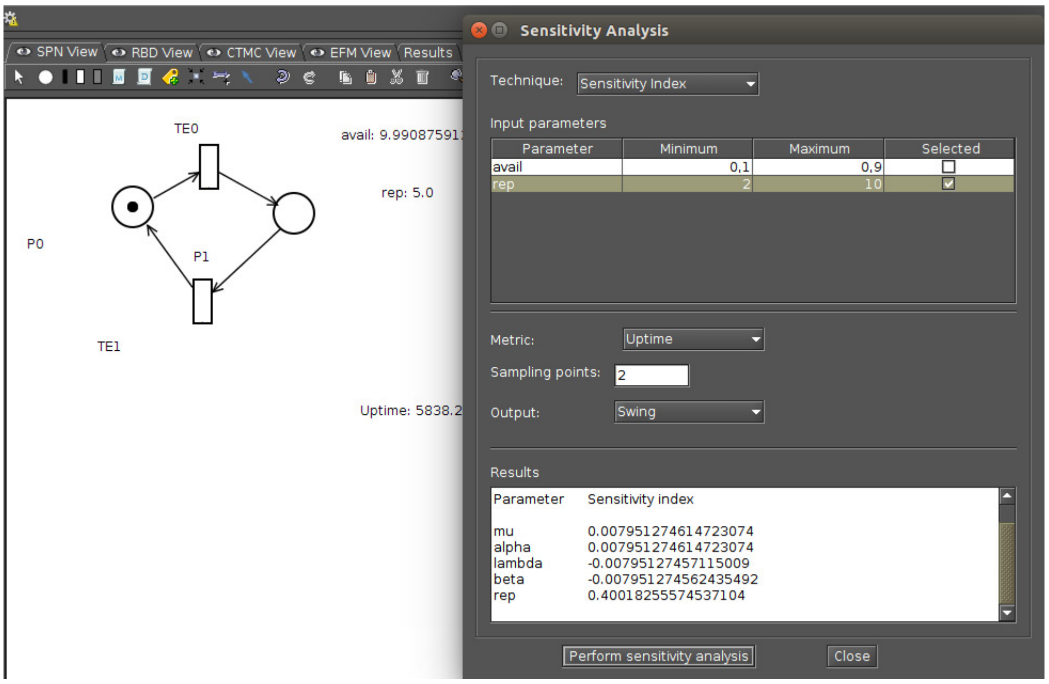
Metric: Av

Analysis Type: Stationary

Figure 40: SPN label for metric from CTMC submodel.

2.6 Sensitivity analysis

Mercury allows performing sensitivity analysis on SPN models, through the “Sensitivity analysis” item on the menu “Tools”. The Sensitivity analysis window is shown in Figure 41. In this window, the user must select between two sensitivity analysis methods: Design of Experiments or Sensitivity Indices. The first uses the standard method of analysis of variance to identify the effect of each factor on model results. The latter uses the technique of percentage difference, so it requires a minimum and maximum value for each parameter in order to compute the corresponding percentage variation on the chosen metric. Mercury is able to compute sensitivity of an SPN measure with respect to each delay parameter of the SPN model as well as with respect to the parameters of CTMC sub-models that provide the value for any SPN transition.



Technique: Sensitivity Index

Input parameters

Parameter	Minimum	Maximum	Selected
avail	0,1	0,9	☐
rep	2	10	✓

Metric: Uptime

Sampling points: 2

Output: Swing

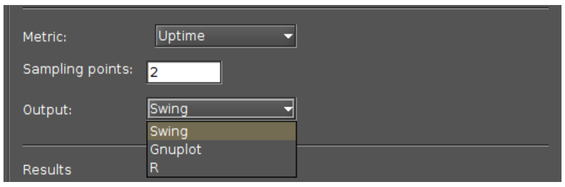
Results

```
Parameter    Sensitivity index

mu           0.007951274614723074
alpha        0.007951274614723074
lambda       -0.00795127457115009
beta         -0.007951274562435492
rep          0.40018255574537104
```

Figure 41: Sensitivity analysis example.

The results of the sensitivity analysis are presented in three possible output formats: Swing, R, and Gnuplot. The user must select one of those outputs, as depicted in Figure 42. The default option is “Swing”, which prints the results on the text area on the bottom of the Sensitivity Analysis window.



Metric: Uptime

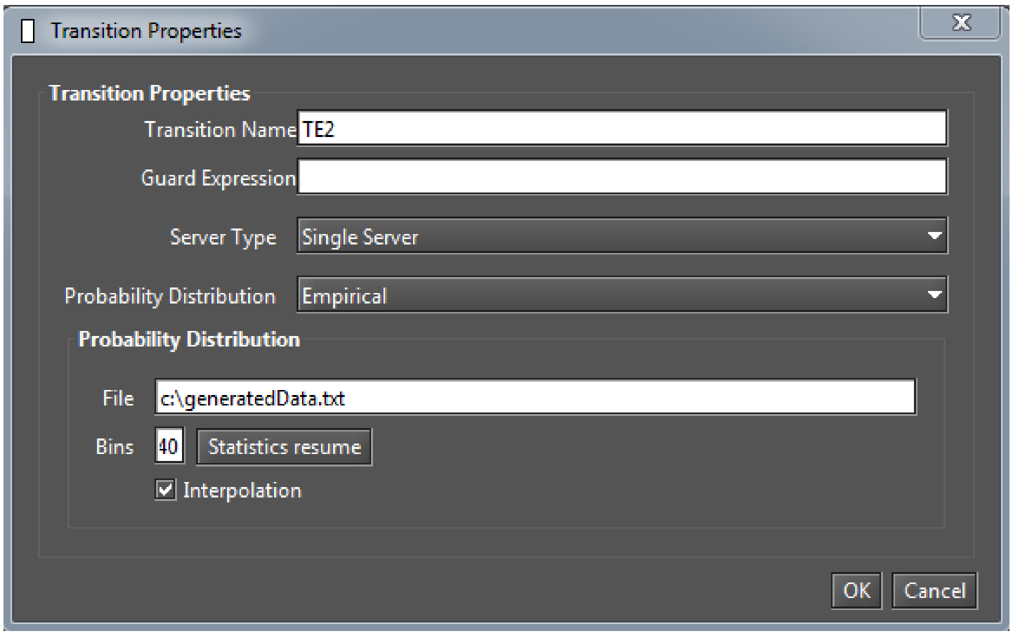
Sampling points: 2

Output: Swing

Figure 42: Output options for sensitivity analysis.

2.7 Transitions with Empirical Distributions

Mercury tool allows user data to represent their own distributions. It is possible to get output data from another model or data from a real system and create **Empirical distributions**. Data such as time failures from a server or customer arrivals in a queue.



Transition Properties

Transition Properties

Transition Name TE2

Guard Expression

Server Type Single Server

Probability Distribution Empirical

Probability Distribution

File c:\generatedData.txt

Bins 40

[✓] Interpolation

Figure 43: Empirical distribution parameters.

Figure 43 depicts the necessary parameters for enabling Empirical distributions. The *File* field set the absolute file path containing data. The file must contains *float* numbers using dots as decimal separators and a line break between values. The *Bins* field set the number of bins in which the samples will be split. The *Interpolation* check box is used to determine if the generated random variates will be interpolated (recomended). Finally, the *Statistics resume* button allows to view the data statistics.

3 RBD Modeling and Evaluation

In the RBD view, the standard model is presented with a directed block end to end as shown in Figure 44. In order to create a new block, it is necessary to press the right mouse click on the initial block already present. Moreover, Figure 45 shows how the users can add labels; insert (in a parallel or series representation), edit and remove blocks. Furthermore, users can also choose different connection options: *Bridge Block, Exponential Block and k-out-of-n block.*

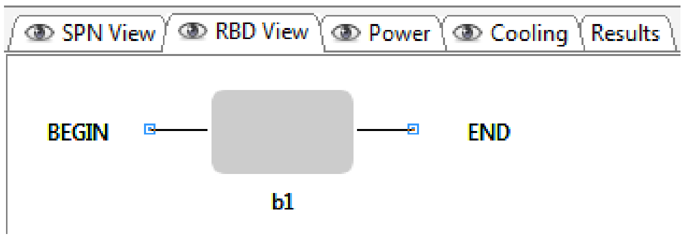


Figure 44: Initial RBD Model

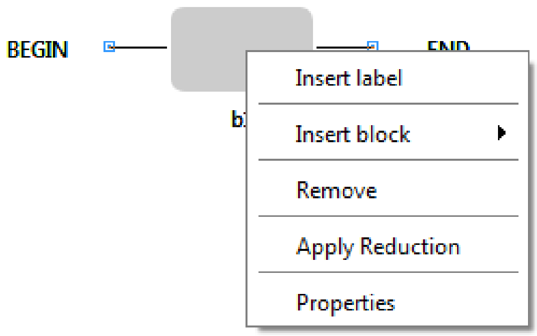


Figure 45: RBD Model properties

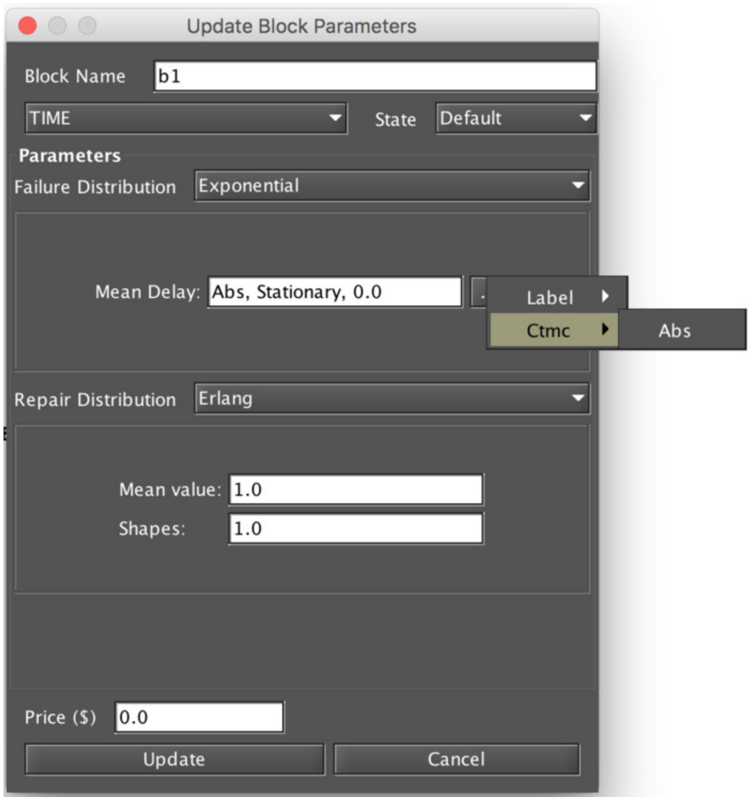
When a new component is added to the model, it is displayed the component's properties in order to allow the user to edit these values. The values that can be edited are *block's name, type parameters, state (default ou failed), distribution type and values for failure and repair, hierarchical parameters from submodels, and price.* Another way to edit the properties of a block is to click with the right button on its component. Figure 46 shows the block properties and Figure 49 shows the label properties. However, in case the component that has been edited is a label, it can be edit the label's *name and value.*

RBD allow to edit blocks with different distributions, just selecting its type and fill the correct parameters in the Edit Screen. This option adopt hierarchical analysis it is performed by SPN Analysis.

Block properties:

- **Block Name:** Component's name.
- **Parameters Type:** There are four parameters type available to be chosen: RATES, TIME, AVAILABILITY and RELIABILITY.

- **State:** The state can be default or failed.
- **Failure Parameters:** The theoretical distribution and values related to components failure.
- **Repair Parameters:** The theoretical distribution and values related to components repair.
- **Reliability or Availability:** Metrics values of the block.
- **"..." button :** Enable the user select any label declared or CTMC metrics defined in CTMC Models as hierarchical parameters from submodels.
- **Price:** Price of component.



Block Name: b1
TIME
State: Default
Failure Distribution: Exponential
Mean Delay: Abs, Stationary, 0.0
Repair Distribution: Erlang
Mean value: 1.0
Shapes: 1.0
Price ($): 0.0

Figure 46: RBD Block Properties

RBD View provides Exact Evaluation, Bounds Evaluation, Importance Measures, Experiment, Get Functions, and Sensitivity Analysis. All those options are available through the **Evaluation** menu.

3.1 Exact evaluation

Figure 47 shows the input window for Reliability Block Diagram Analysis. The tool have two methods to calculate the reliability, the user should choose between the standard mode or the method of Sum of Disjoint Products.

The metrics that users can choose to be computed are Mean Time To Failure, Mean Time to Repair, Uptime, Steady-State Availability, Instantaneous Availability, Downtime, Reliability and Unreliability.

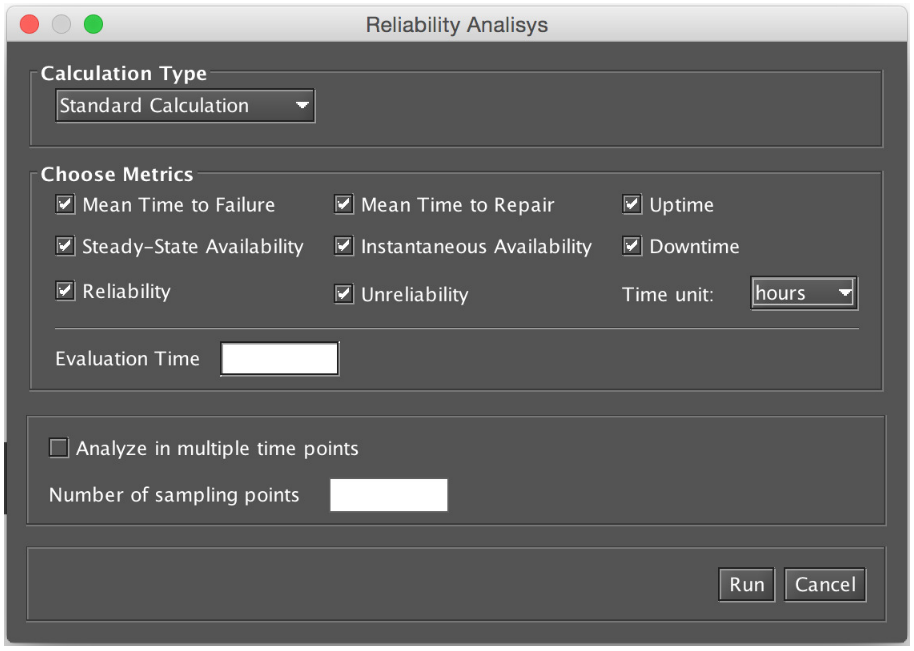


Figure 47: Input window for RBD Analisys

3.2 Resolution by Simulation

The resolution of a RBD through simulation occurs when the model components distributions (failure and repair) are not exponential, as Erlang, Gamma, etc. In this case, the not exponential block is converted to a SPN and resolved by simulation.

When this type of block is detected in the model by selecting any analysis of the RBD, the user is prompted to provide input parameters for simulation, according to the chosen metric. To this end, the simulation may be transient or stationary, as shown in Figure 48.

3.3 Experiment RBD

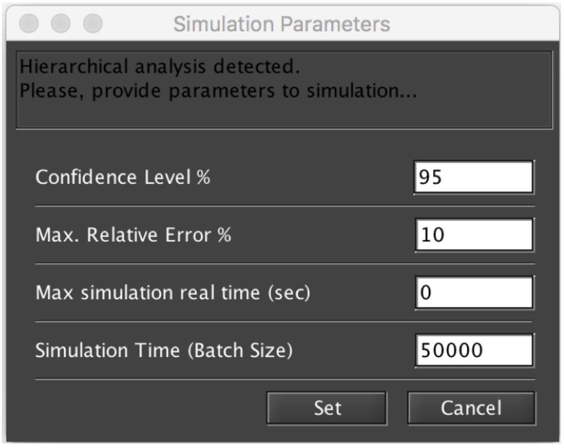
First of all, users should set the label properties. This properties is shown in Figure 49, and set it in a block.

Label Name: Component (block) name.

Value: The value of the component.

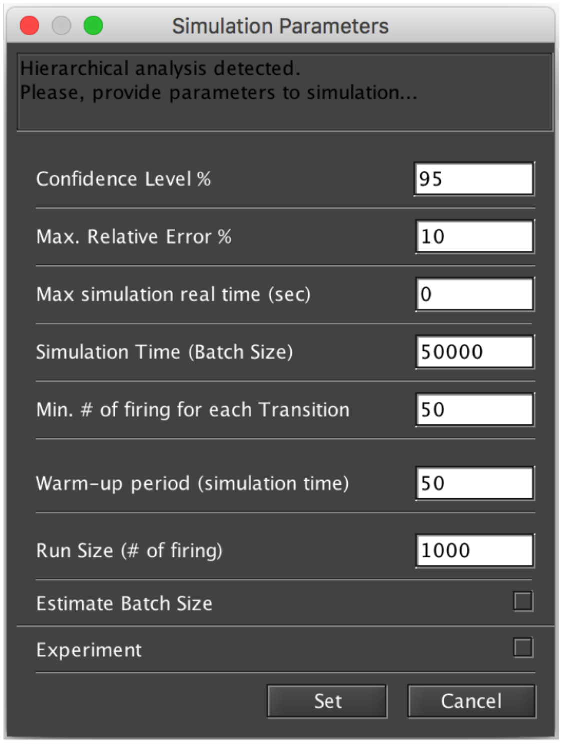
In order to perform this RBD evaluation, you must fill the information to the experiment to be carried out. Figure 50 shows the input window for RBD Experiment.

After select the label name as well as configure the metric, the minimal and maximum range, the interval and the evaluation time. It is important to state that users have to configure the evaluation time of analysis in order to execute the experiment.


Simulation Parameters

Hierarchical analysis detected.
Please, provide parameters to simulation...

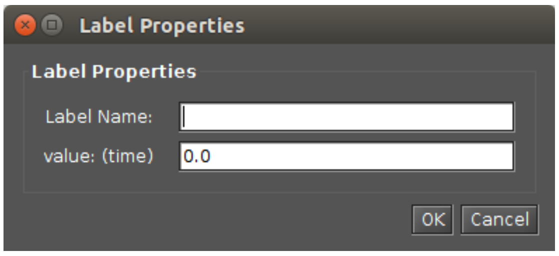
Confidence Level %	95
Max. Relative Error %	10
Max simulation real time (sec)	0
Simulation Time (Batch Size)	50000


Simulation Parameters

Hierarchical analysis detected.
Please, provide parameters to simulation...

Confidence Level %	95
Max. Relative Error %	10
Max simulation real time (sec)	0
Simulation Time (Batch Size)	50000
Min. # of firing for each Transition	50
Warm-up period (simulation time)	50
Run Size (# of firing)	1000
Estimate Batch Size	☐
Experiment	☐

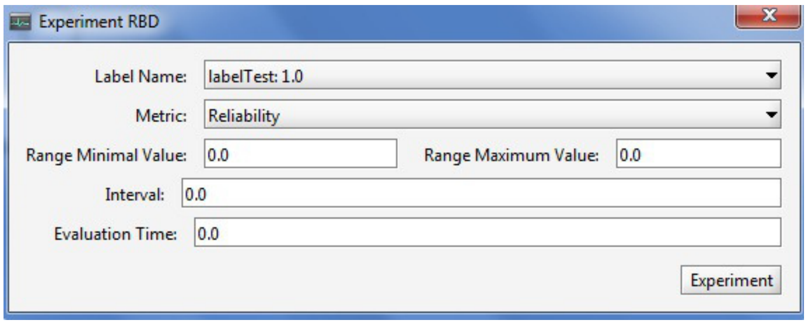
Figure 48: Input windows for Transient Simulation and Stationary Simulation


Label Properties

Label Properties

Label Name:	
value: (time)	0.0

Figure 49: RBD Labels Properties


Experiment RBD

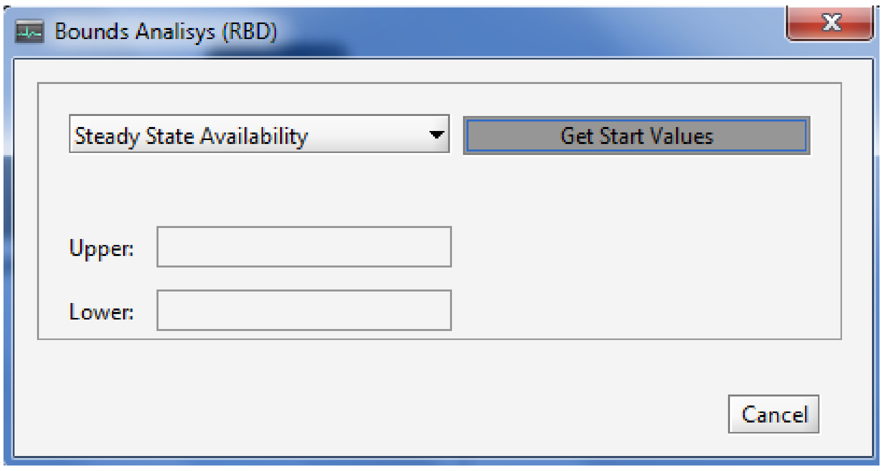
Label Name:	labelTest: 1.0
Metric:	Reliability
Range Minimal Value:	0.0
Range Maximum Value:	0.0
Interval:	0.0
Evaluation Time:	0.0

Figure 50: Experiment RBD

3.4 RBD Bounds Analysis

RBD Bounds Analysis is adopted to estimate the dependability metrics through the calculation of reliability and availability. Therefore, it is necessary to estimate the bounds values (upper and lower limits) for computing this analysis in which users quickly obtain the results.

Figure 51, Figure 52 and Figure 53 show the input window for “RBD Bounds Analysis"’. This new functionality should be performed when the model in analysis is very large. This method is subdivided in two parts: (i) computing the bounds values and (ii) adopting the sum of disjoint points to find the successive values and the number of iterations.



Steady State Availability

Upper:

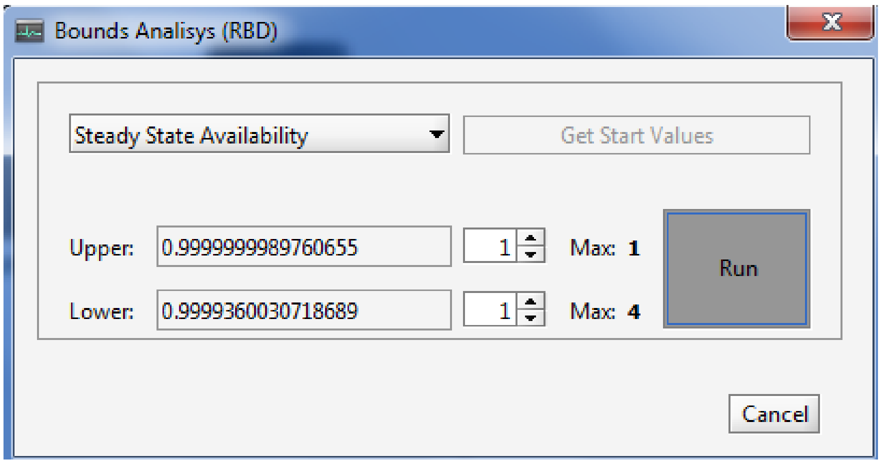
Lower:

Figure 51: RBD Bounds Analysis - Selection of Metric

First, it is computed the upper and lower values. This option adopts the first path and the first cut to provide the higher and lower values of the chosen metric. The paths are related to the lower bounds, in which a minimal set of components are chosen to guarantee the operational mode of the system. Meanwhile the cuts are related to the upper bounds, in which a minimal set of components that ensure the system on the failure mode is adopted. The user should choose the evaluation and click in the button “Get Start Values"’ as in the Figure 51, in which there is also information on how many iterations may be made to find the exact value.

After get the upper and lower values, the user should select the number of steps and click in the button “Run"’ as in the Figure 52.

Then, the user can see the result values in a window as in the Figure 53. Also, it’s possible to plot chart and export the results for an excel file.

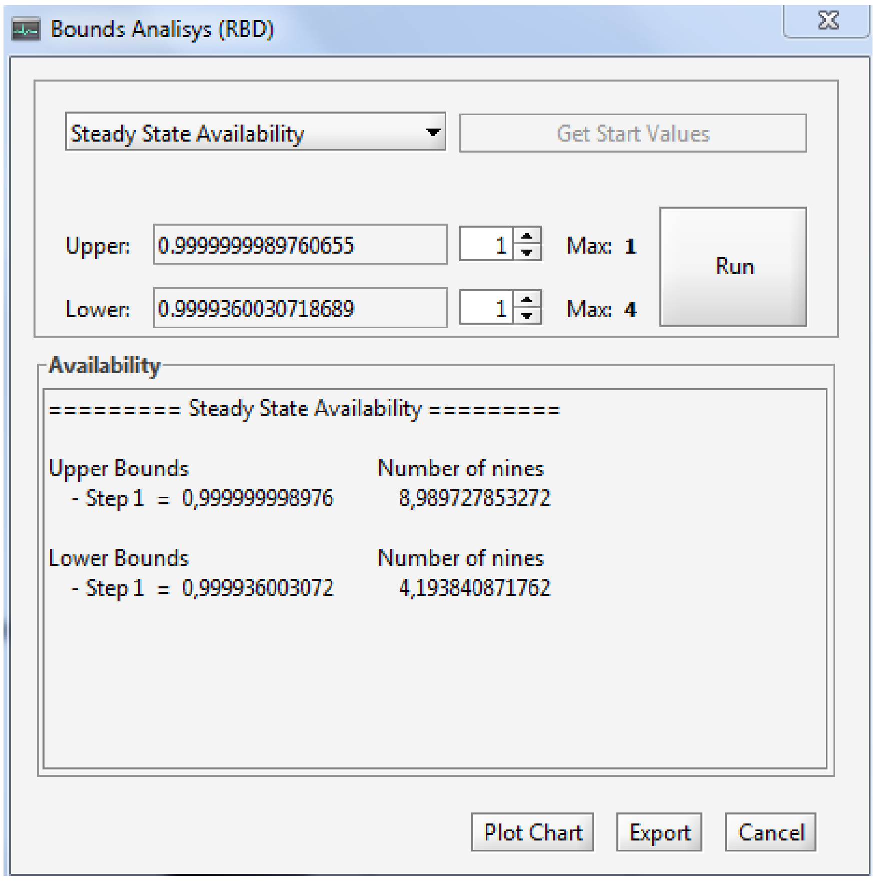


Steady State Availability

Upper: 0.9999999989760655 1 Max: **1**

Lower: 0.9999360030718689 1 Max: **4**

Figure 52: RBD Bounds Analysis - Start Values and Number of Iteractions



Steady State Availability

Upper: 0.9999999989760655 1 Max: 1

Lower: 0.9999360030718689 1 Max: 4

```
========= Steady State Availability =========

Upper Bounds                    Number of nines
  - Step 1  =  0,999999998976      8,989727853272

Lower Bounds                    Number of nines
  - Step 1  =  0,999936003072      4,193840871762
```

Figure 53: RBD Bounds Analysis - Result of Computation

Calculate successive values using Sum of Disjoint Products: The method adopted to find successive values and the number of iterations (values) is defined by the number of paths and cuts of the model. Increasing the number of iterations, the value found will be closer to the exact value. When the calculation is done for the last way or last cut, the exact value for the metric is found. Successive values can be calculated only for the paths, just for the cuts, or both.

- **Steps to calculate successive values:** After to calculate the values of the first path and the first cut the user should choose the number of iterations you are interested and click on “Run Iterations"’. The values of each iteration will be displayed at the area highlighted.

- **Steps to find the exact value:** After to calculate the values of the first path and the first cut the user should choose the max number of iterations that is interested (upper or lower) and click on “Run Iterations"’. The exact values will be the value found on last step..

3.5 RBD Reduction

In order to reduce the RBD complexity, a new functionality was created aiming to reduce the number of block diagrams by a reduction process. This feature can be used until there is a model with only one single block diagram. The model, for instance, may be reduced from its configuration such as series, parallel or bridge. The main goal is to simplify the model, However, the original model parameters may be lost and only the metric and characteristic of the original model will be preserved.

Figure 54 shows a RBD model before applied a reduction process, in which there are four block diagrams. In order to perform the RBD reduction, users should click on the mouse's right button over the submodel and select the option *Apply Reduction*. Figure 55 illustrates the model after the performed reduction process, in which it was possible to observe that the number of block diagrams were reduced to 3 blocks (b2 and b3 were reduced to just one block, namely, ssb6). Thus, the result is a new compressed submodel.

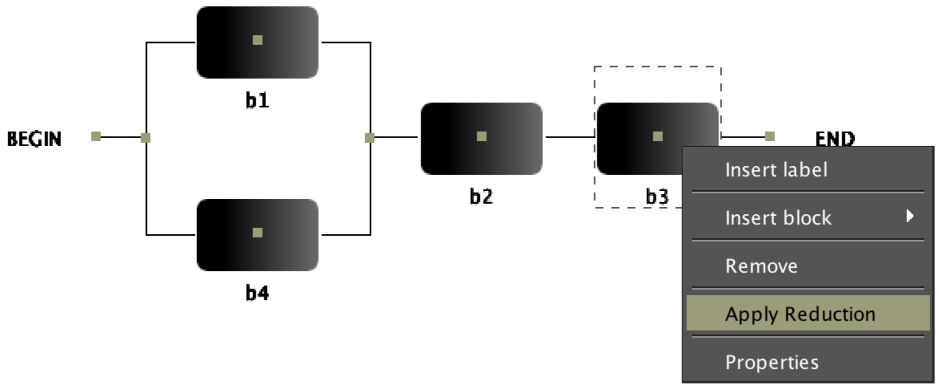


Figure 54: RBD diagram before the series submodel reduction.

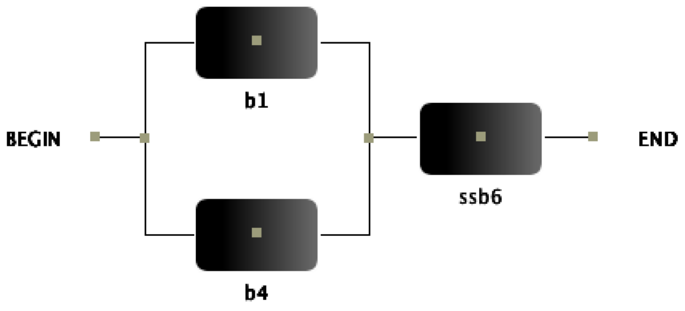


Figure 55: RBD diagram after the series submodel reduction.

In a sub-model series, the original features of the model are maintained. However, apply reductions in parallel sub-models, when the parameters of TIME or RATE have been edited, it is possible only to keep the characteristics at one point in time. So when the reduction is proceeded in a parallel model, new options are displayed as Figure 56, allowing to choose the metric to evaluate. If the reliability is further requested the TIME of reliability will be maintained as shown in Figure 57.

3.6 Component Importance and Total Cost of Acquisition

Component importance is a metric that indicates the impact of a particular component in the system's overall. With these importance values, the most important component (i.e., with the highest RI) should be improved to

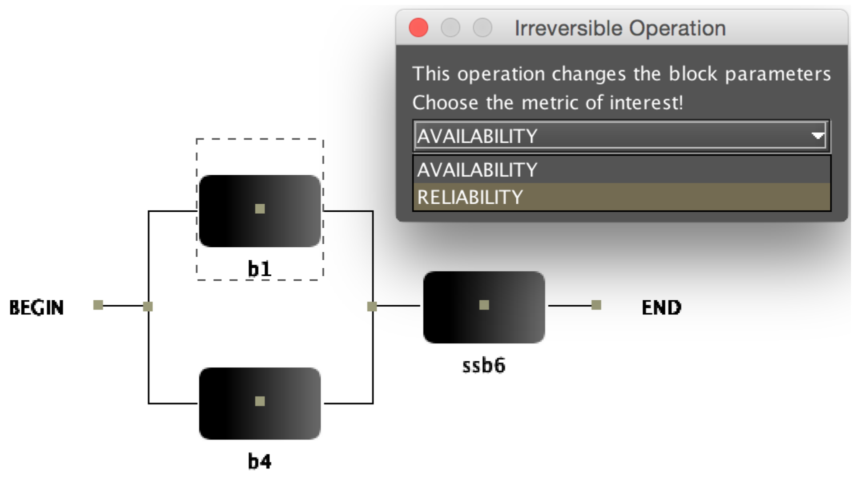

Irreversible Operation
This operation changes the block parameters
Choose the metric of interest!
AVAILABILITY
AVAILABILITY
RELIABILITY
b1
BEGIN
END
ssb6
b4

Figure 56: RBD diagram before parallel submodel reduction.

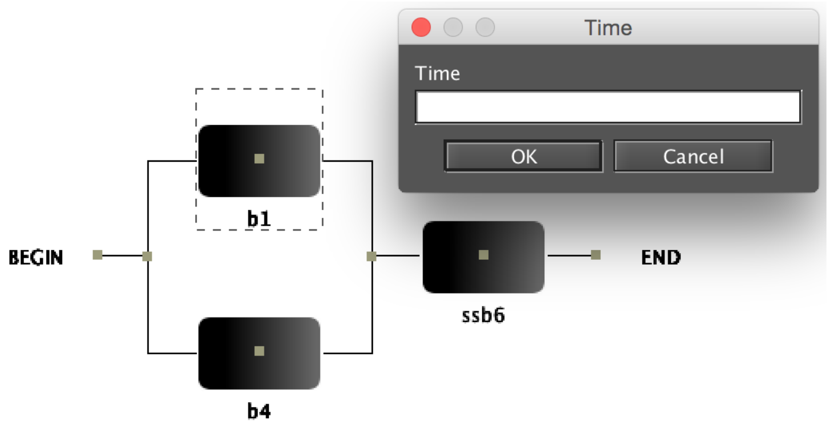


Figure 57: RBD diagram input time for parallel reduction.

increase system reliability. This evaluation can assist in maintenance actions.

This Mercury functionality is responsible for identifying the relative importance of each component in a system with respect to the overall system reliability or availability. If the parameter Cost was edited in the blocks, it is possible to evaluate the relationship between this metrics and investment costs.

It is possible to compute some importance measures as shown in Figure 58. Measures available are Reliability Importance (RI), Availability Importance (AI), and Criticallity (Reliability or Availability) Importance (CRI). Last measures can be obtained considering the system's failure or functioning.

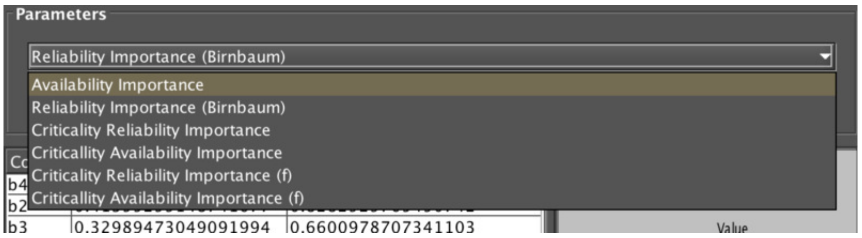
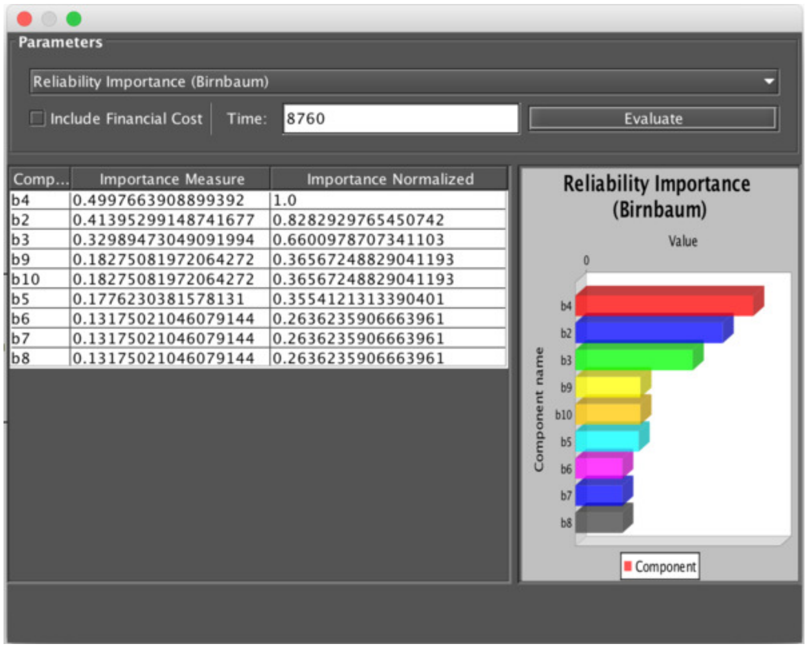

Parameters
Reliability Importance (Birnbaum)
Availability Importance
Reliability Importance (Birnbaum)
Criticallity Reliability Importance
Criticallity Availability Importance
Criticallity Reliability Importance (f)
Criticallity Availability Importance (f)
Co
b4
b2
b3
0.32989473049091994
0.6600978707341103
Value

Figure 58: Component Importance Measures

To use the feature, users should select the *Importance Measures* in the BRB Evaluation. After that, they should select one metric of interest on the Component Importance Screen and select Evaluate button. The results of this operation is shown in Figure 59. The results shown the importance value to each component and a graphical view as a ranking highlighting those most significant in the analysis.



Parameters

Reliability Importance (Birnbaum)

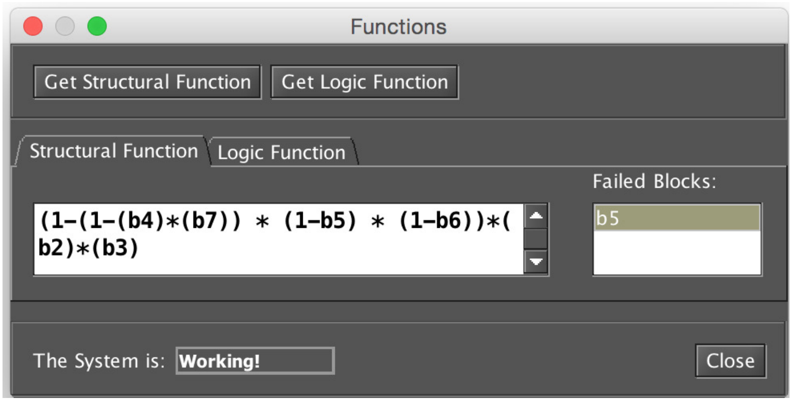
Include Financial Cost | Time: 8760

Comp...	Importance Measure	Importance Normalized
b4	0.4997663908899392	1.0
b2	0.41395299148741677	0.8282929765450742
b3	0.32989473049091994	0.6600978707341103
b9	0.18275081972064272	0.36567248829041193
b10	0.18275081972064272	0.36567248829041193
b5	0.1776230381578131	0.3554121313390401
b6	0.13175021046079144	0.2636235906663961
b7	0.13175021046079144	0.2636235906663961
b8	0.13175021046079144	0.2636235906663961

Figure 59: Reliability Importance Results.

3.7 Structural and Logical Functions

The Mercury Tool allows automatic generation of Structural Function (and Logical Funcion), which is a behavioral representation (in the failure of perspective and operation) of the system structure in the form of function, for RBD models. Combining the generation of structural function and the possibility to set the block status manually, it is possible to check the system status in relative to the block status. This function is accessible on menu *RBD Evaluation* and show the screen displayed in Figure 60 showing, in addition to the generated expression, the blocks that are marked as failed (inoperative) and the current state of the system.



```
(1-(1-(b4)*(b7)) * (1-b5) * (1-b6))*(
b2)*(b3)
```

Failed Blocks: b5

The System is: Working!

Figure 60: Structural Function of a RBD Model

3.8 Hierarchical Evaluation

RBD parameters can be set by evaluation of submodels such as CTMC ones. This kind of evaluation uses a metric result from a submodel as input to a desired RBD parameter. So, combinatorial and state-space models may be hierarchically combined in order to get the best of both models, e.g., a RBD could be used to represent the dependability relationship between independent subsystems, while more complex fail and repair mechanisms are modeled with CTMCs [4].

Mercury supports evaluation from CTMC as submodels to an RBD model. At evaluation time, the CTMC will be solved and the parameters will be gotten from that. Thus, it's necessary to have both CTMC and RBD models drawn inside of same Mercury project. For example, assume you have these following CMTC and RBD models as in Figure 61 and 62.

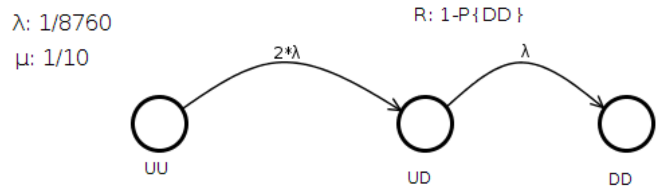


Figure 61: CTMC submodel example.

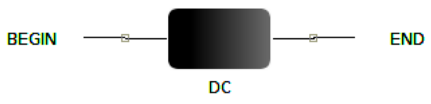


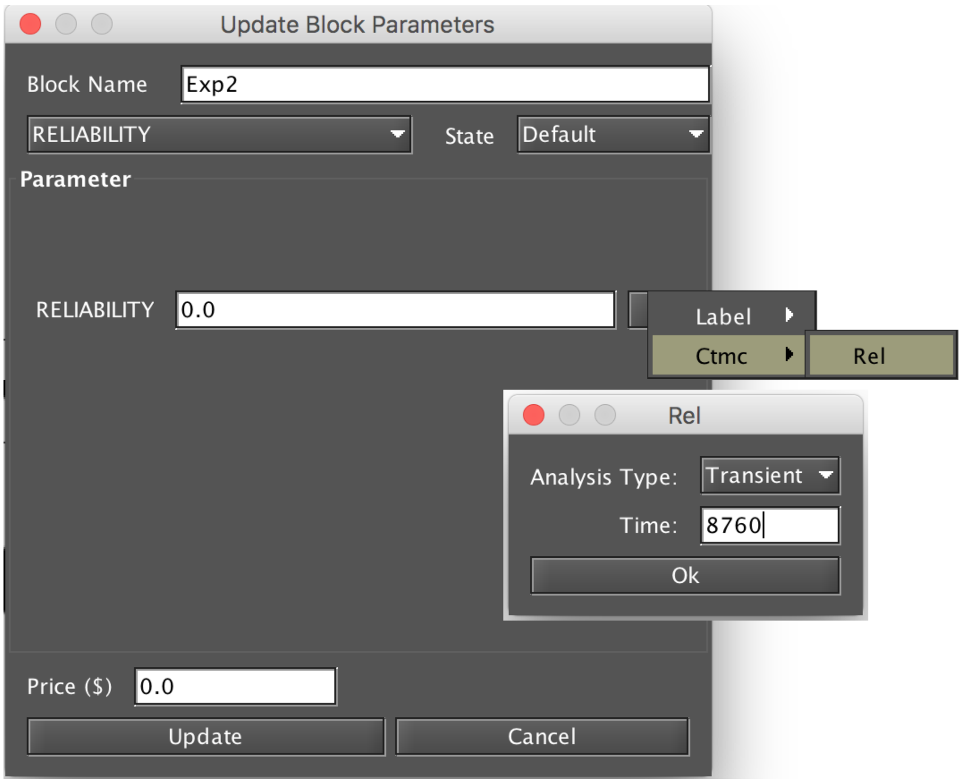
Figure 62: RBD model example.

When the submodel is detected, it's possible to get the parameter from it by clicking on the "..." button adjacent to parameter RBD as in the Figure 63. Following, one can choose the desired metric from the submodel on the drop down menu and then select the kind of analysis (stationary or transient).

In the example, it is shown the hierarchical to Reliability parameter, but it's also possible to get two parameters simultaneously from the submodel. That is the case when the block parameters are exponentially distributed to MTTF (or Failure Rate) and MTTR (or Repair Rate).

3.9 Sensitivity Analysis

The Evaluation menu has a **"Sensitivity Analysis"** option, that enables computing partial derivative sensitivity indices for the RBD model. Those indices indicate the impact that every input parameter has on the model's availability. If the model has hierarchical blocks, the input parameters of the CTMC sub-model are also consid-


Update Block Parameters

Block Name: Exp2

RELIABILITY

State: Default

Parameter

RELIABILITY: 0.0

Rel

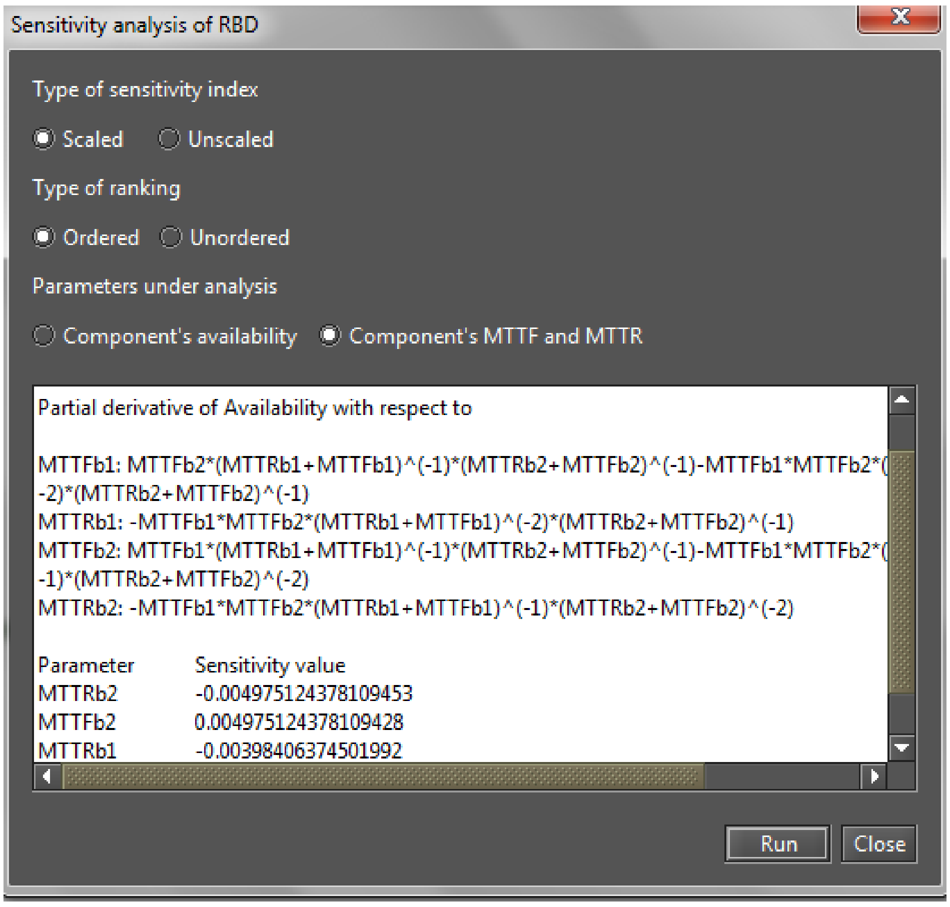
Analysis Type: Transient

Time: 8760

Price ($): 0.0

Figure 63: RBD block enabled to get parameters from a submodel.

ered in the Sensitivity Analysis. Figure 64 shows the RBD Sensitivity Analysis window, which presents the partial derivative of the structural equation to every parameter, as well as the sensitivity indices.


Sensitivity analysis of RBD

Type of sensitivity index

[✓ Scaled] Unscaled

Type of ranking

[✓ Ordered] Unordered

Parameters under analysis

Component's availability [✓ Component's MTTF and MTTR]

```
Partial derivative of Availability with respect to

MTTFb1: MTTFb2*(MTTRb1+MTTFb1)^(-1)*(MTTRb2+MTTFb2)^(-1)-MTTFb1*MTTFb2*(
-2)*(MTTRb2+MTTFb2)^(-1)
MTTRb1: -MTTFb1*MTTFb2*(MTTRb1+MTTFb1)^(-2)*(MTTRb2+MTTFb2)^(-1)
MTTFb2: MTTFb1*(MTTRb1+MTTFb1)^(-1)*(MTTRb2+MTTFb2)^(-1)-MTTFb1*MTTFb2*(
-1)*(MTTRb2+MTTFb2)^(-2)
MTTRb2: -MTTFb1*MTTFb2*(MTTRb1+MTTFb1)^(-1)*(MTTRb2+MTTFb2)^(-2)

Parameter        Sensitivity value
MTTRb2           -0.004975124378109453
MTTFb2           0.004975124378109428
MTTRb1           -0.00398406374501992
```

Figure 64: Sensitivity analysis of RBD model.

4 CTMC Modeling and Evaluation

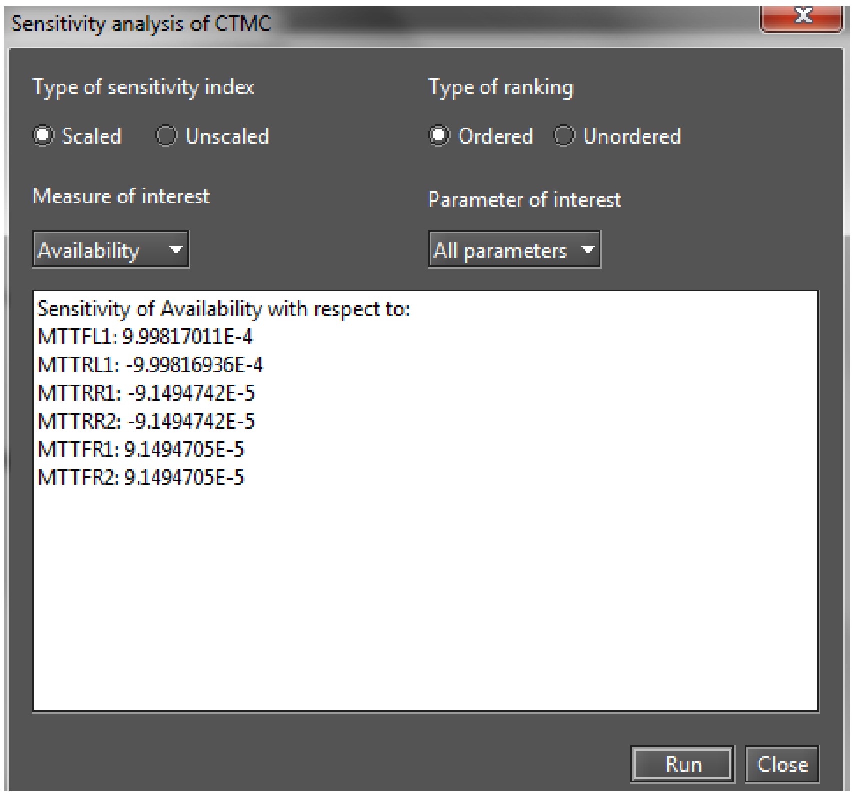
In the CTMC view, the user must click on the "Add New States" icon (see Figure 66), and then click on the drawing panel to create new states.

After adding the states, transitions are drawn by clicking in the center of the source state and dragging the created line until the target state, as shown in Figure 67.

The user can define the rate of each transition by right-clicking in the respective arc and clicking in the "Properties" item. Figure 68 shows the window where the transition rate can be defined.

Mercury also allows assigning reward rates to the states of the CTMC. The user must right-click on the selected state, and click on "Properties" item. Figure 69 shows the window where the reward rate assigned to the state can be defined. The default reward for every state is zero. The user can enter any real value or expression containing user-defined parameters. The state name can also be changed in the same window.

After all states and transitions are properly defined (see Figure 70), it is possible to perform stationary and transient analyses for this model. The user can also save the infinitesimal generator matrix (i.e., the transition rate matrix) to a text file, by clicking in the matrix icon shown in Figure 71.



Type of sensitivity index: [✓ Scaled] Unscaled

Type of ranking: [✓ Ordered] Unordered

Measure of interest: Availability

Parameter of interest: All parameters

```
Sensitivity of Availability with respect to:
MTTFL1: 9.99817011E-4
MTTRL1: -9.99816936E-4
MTTRR1: -9.1494742E-5
MTTRR2: -9.1494742E-5
MTTFR1: 9.1494705E-5
MTTFR2: 9.1494705E-5
```

Figure 65: Results of sensitivity analysis for CTMC model

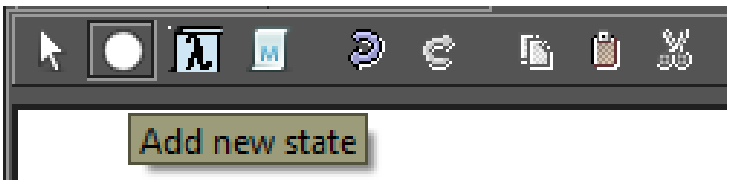


Figure 66: Adding a state to the CTMC

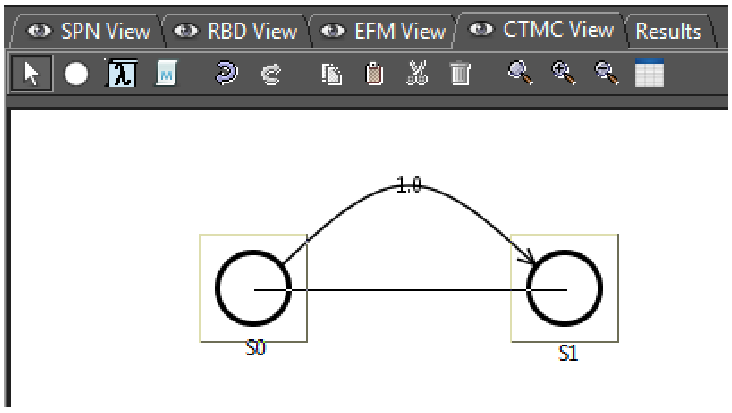
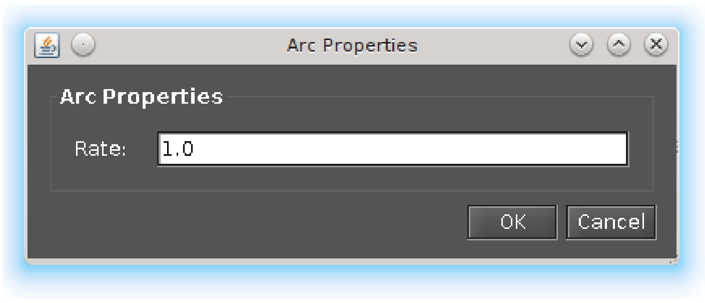


Figure 67: Adding a transition to the CTMC

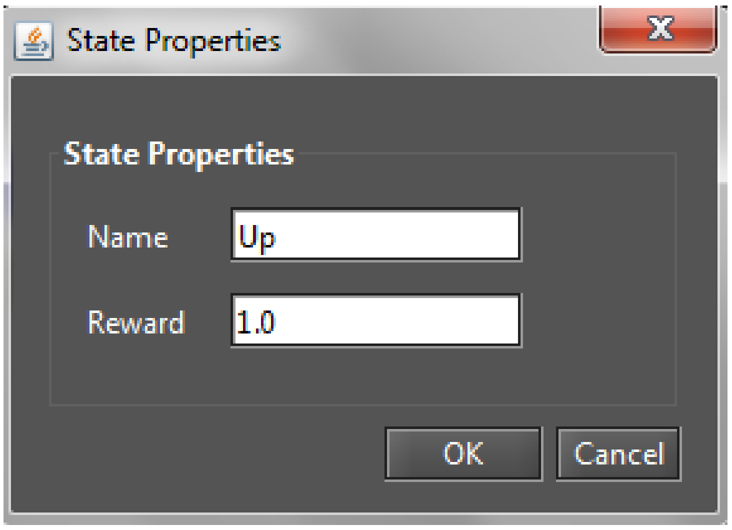
4.1 Input parameters

The transition rates of CTMCs can be defined through expressions containing both numbers and user-defined parameters. The button “Add new parameter definition” (see Figure 72) is represented by a λ symbol on CTMC menu bar, and it enables creating a symbolic parameter. After clicking in the button, click on any point in



Rate: 1.0

Figure 68: Defining the rate of a transition in the CTMC



Name: Up
Reward: 1.0

Figure 69: Modifying state properties (name and reward)

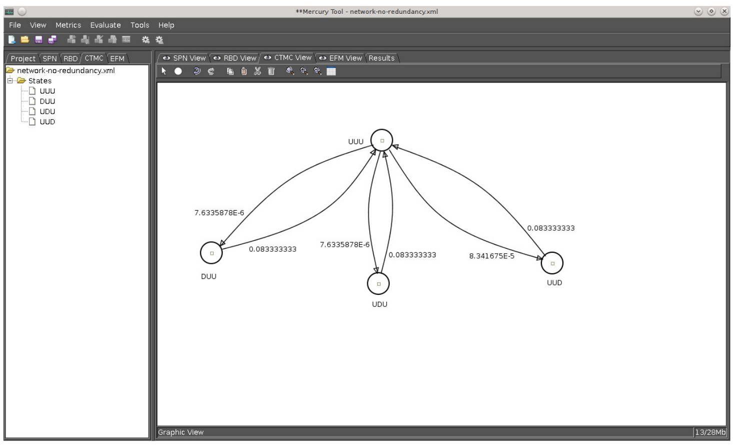


Figure 70: Example of CTMC model

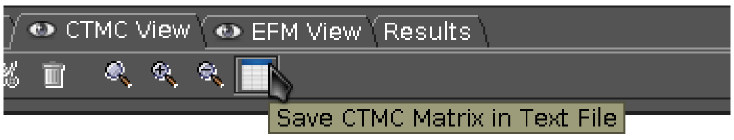


Figure 71: Saving the transition rate matrix to a file

CTMC drawing area, and a new parameter is created with name **Param0** (or **Param1**, etc, if other parameters were already created). In order to change the parameter name and define a value for it, go to the Properties on right-click menu. The name of the parameter can be defined by means of any combination of alphanumerical characters. Special characters (e.g., underscore, hyphen, ampersand) must be avoided. If names of greek letters are used for parameters, Mercury converts them to the proper symbol on greek lowercase alphabet (see Figures 73 and 95). The value assigned to the parameter can be a numerical expression. Any symbols or other parameter names are not allowed in parameter value.

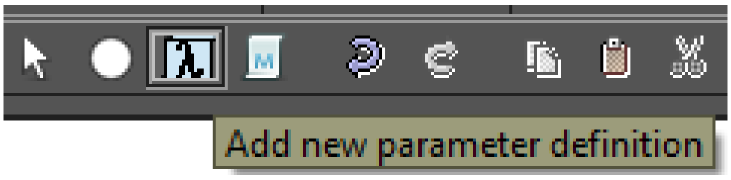
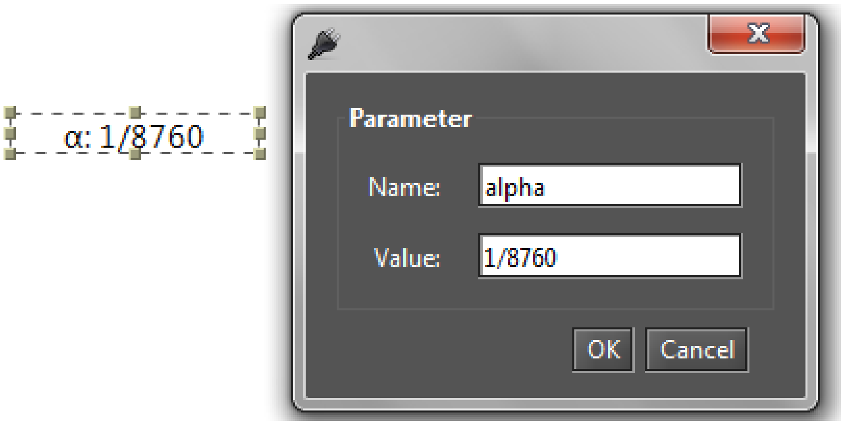


Figure 72: Adding a new user-defined parameter



Name: alpha

Value: 1/8760

Figure 73: Modifying parameter name and value

4.2 Metrics

The button "Add new metric definition" (see Figure 75) enables defining metrics related to the CTMC model results. After inserting a metric, the user can change the name and define the expression that will be used to compute the metric. The metric expression syntax is based on state probabilities (**P{*state-name*}**) and rewards (**R{*state-name*}**). The probabilities and rewards of any states can be combined (i.e., added, subtracted, etc) in the metric expression. In the example of Figure 76, the metric **AvB3** indicates the availability of the system represented by the two-state CTMC created with Mercury. The availability will be computed from expression **P{Up}**, that is the probability of being in state **Up**. When the model is evaluated through stationary or transient analysis, the metric value will be updated and displayed on screen, as shown in Figure 77. The system reward rate can be obtained using the metric expression **R{}**, and the system mean time to absorption (when there is at least one absorbing state) can be computed in a metric expression using the keyword **MTTA**.

The metric expression is still visible in the metric properties window, and in the **left-side panel** (see Figure

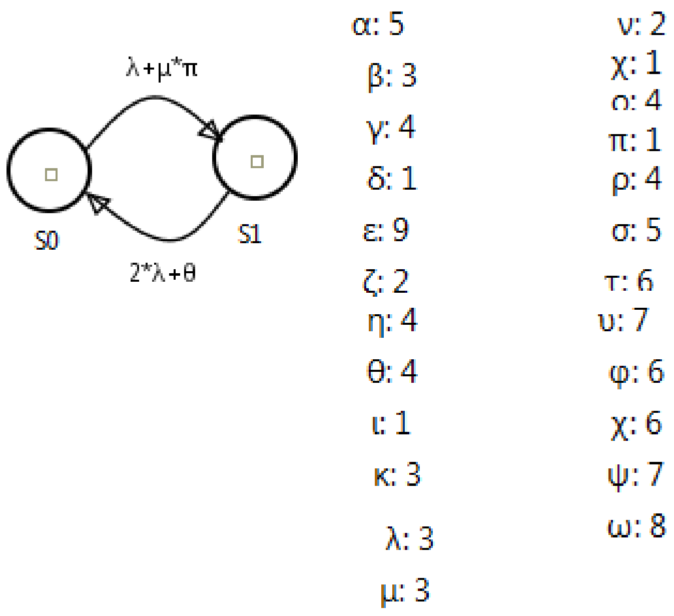


Figure 74: Parameter names defined with greek letters

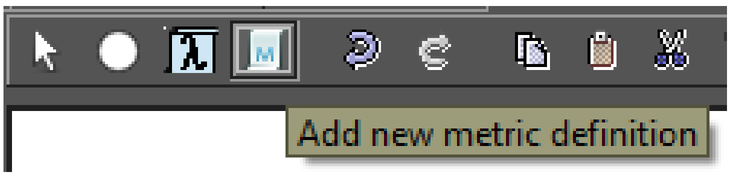


Figure 75: Adding a metric

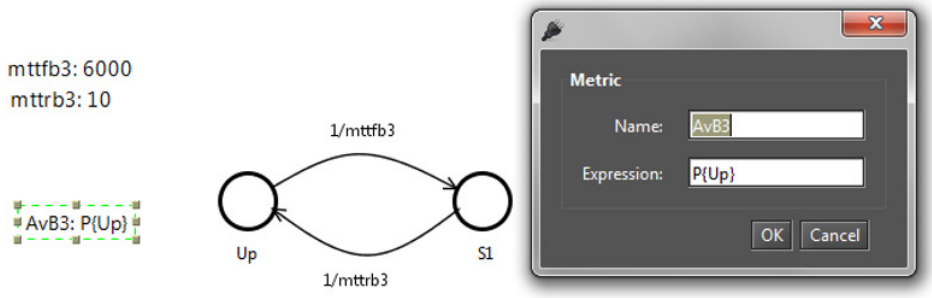


Figure 76: Defining metric name and expression

78). The panel shows all elements of the CTMC: state names and rewards; parameters names and values; metric names and expressions; and transitions. Every transition is represented in the side panel by the source and target states, followed by the expression or value assigned to that transition.

4.3 CTMC Stationary Analysis

When the CTMC View is active, the menu “Evaluation” shows three options for model analysis: **Stationary Analysis**, **Transient Analysis**, and **Sensitivity Analysis**. The Stationary Analysis computes steady-state probabilities, useful for assessing the long-term average behavior of modeled system. Figure 79 shows the window of

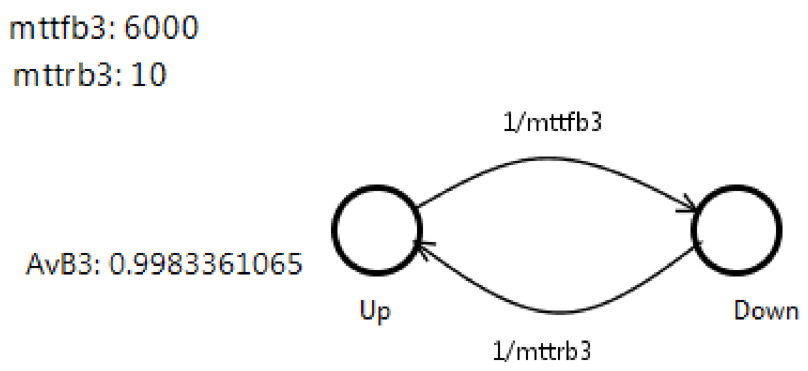
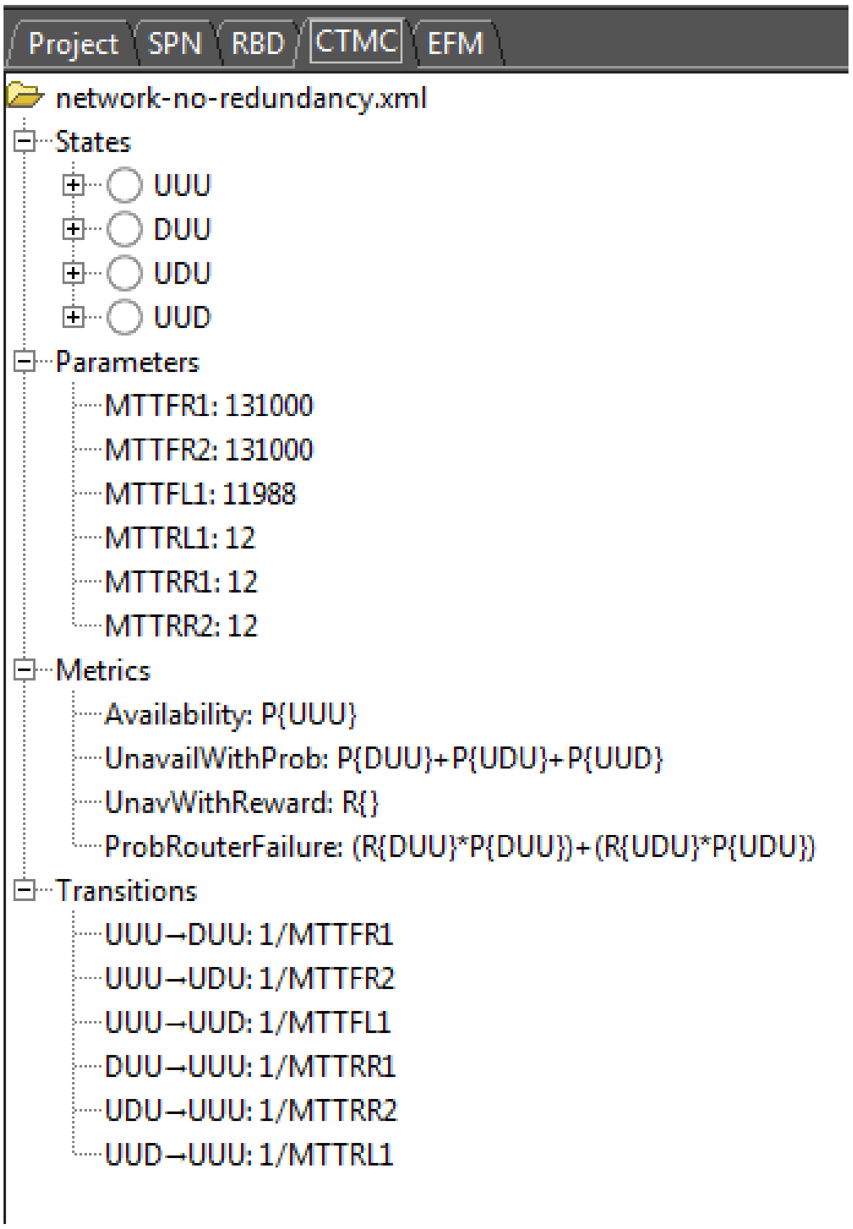


Figure 77: Metric value



```
network-no-redundancy.xml
States
    UUU
    DUU
    UDU
    UUD
Parameters
    MTTFR1: 131000
    MTTFR2: 131000
    MTTFL1: 11988
    MTTRL1: 12
    MTTRR1: 12
    MTTRR2: 12
Metrics
    Availability: P{UUU}
    UnavailWithProb: P{DUU}+P{UDU}+P{UUD}
    UnavWithReward: R{}
    ProbRouterFailure: (R{DUU}*P{DUU})+(R{UDU}*P{UDU})
Transitions
    UUU→DUU: 1/MTTFR1
    UUU→UDU: 1/MTTFR2
    UUU→UUD: 1/MTTFL1
    DUU→UUU: 1/MTTRR1
    UDU→UUU: 1/MTTRR2
    UUD→UUU: 1/MTTRL1
```

Figure 78: CTMC side panel

Stationary Analysis, which has a combo box for selecting one of two solution methods available: **Direct - GTH** (Grassmann-Taksar-Heyman) or **Iterative - Gauss-Seidel**.

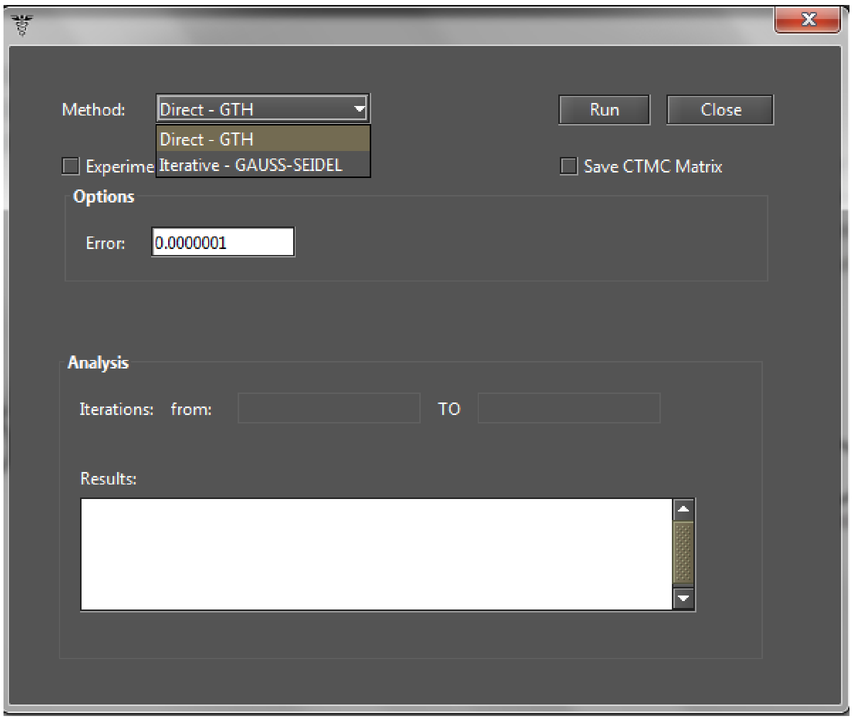
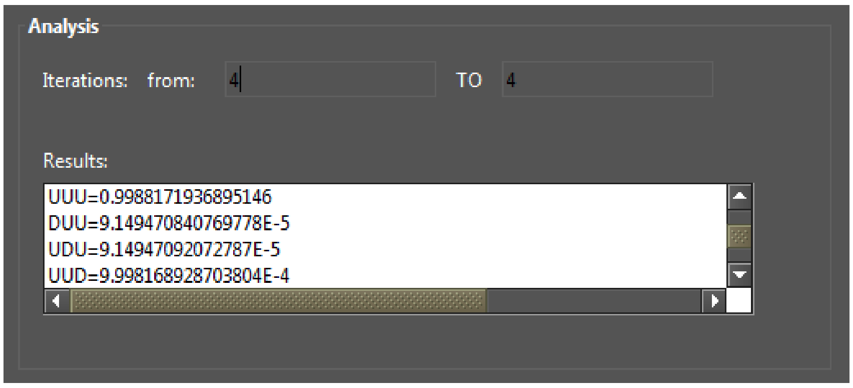


Figure 79: Stationary analysis window

When solving the model through GTH, the user can change the **maximum error** used in the algorithm. The default value for the maximum error is 0.0000001 (10^-7). By clicking on the **Run** button, the solution algorithm is triggered and as soon as it finishes, the results are presented in the text area in the bottom of analysis window (see Figure 80), and written in a plain text file with the name of Mercury model file appended with the " "-StationaryAnalysis.txt" suffix.



```
UUU=0.9988171936895146
DUU=9.149470840769778E-5
UDU=9.14947092072787E-5
UUD=9.998168928703804E-4
```

Figure 80: Stationary analysis results

When solving the model through Gauss-Seidel, besides maximum error, the user can also change the maximum number of iterations. The default value for such a parameter is -1, indicating that the algorithm only stops when convergence of results is reached, considering the informed error.

The user-defined metrics are updated as soon as the analysis completes, independently of the chosen solution method. Their values are shown in the drawing area, as illustrated in Figure 81, where the metrics are in right side.

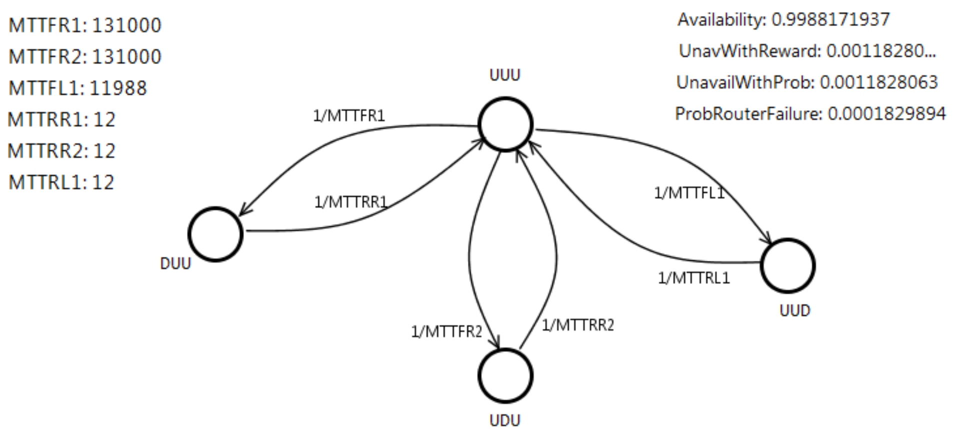
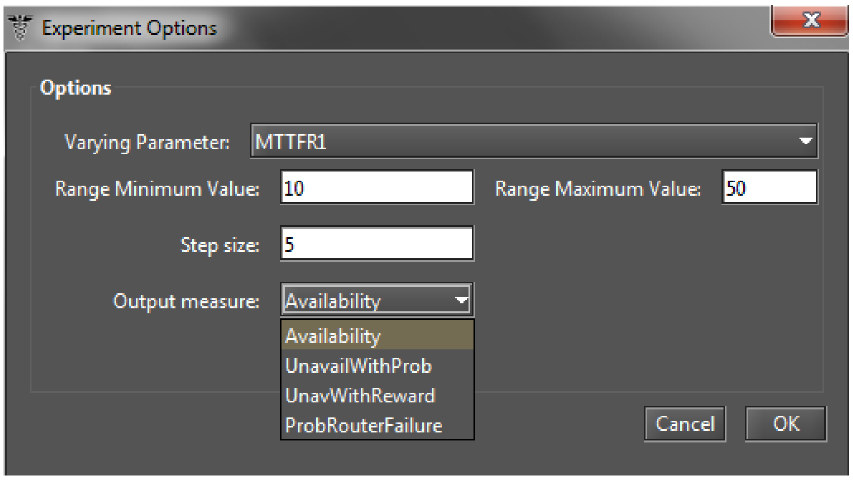


Figure 81: CTMC and its metrics after model solution

The CTMC model can also be solved for a range of values of user-defined parameters. This is accomplished by clicking in the **Experiment** check box on Stationary Analysis window and then on the **Run** button. A new dialog opens for definition of the experiment options (see Figure 82). The options are: the parameter that will vary; the minimum and maximum value for this parameter; the step size for incrementing the parameter value in the defined range; and the output metric that will be shown on experiment results and plotted on a graph (see Figures 83 and 84).


Experiment Options

Options

Varying Parameter: MTTFR1

Range Minimum Value: 10

Range Maximum Value: 50

Step size: 5

Output measure: Availability

Figure 82: CTMC Experiment Options

Another check box in the **Stationary Analysis** window allows saving the CTMC matrix to a plain text file just after the model solution.

4.4 CTMC Transient Analysis

The **Transient Analysis** option in the "Evaluate" menu computes time-dependent probabilities, useful for assessing the behavior of modeled system in a particular point in time. Figure 85 shows the window of Transient

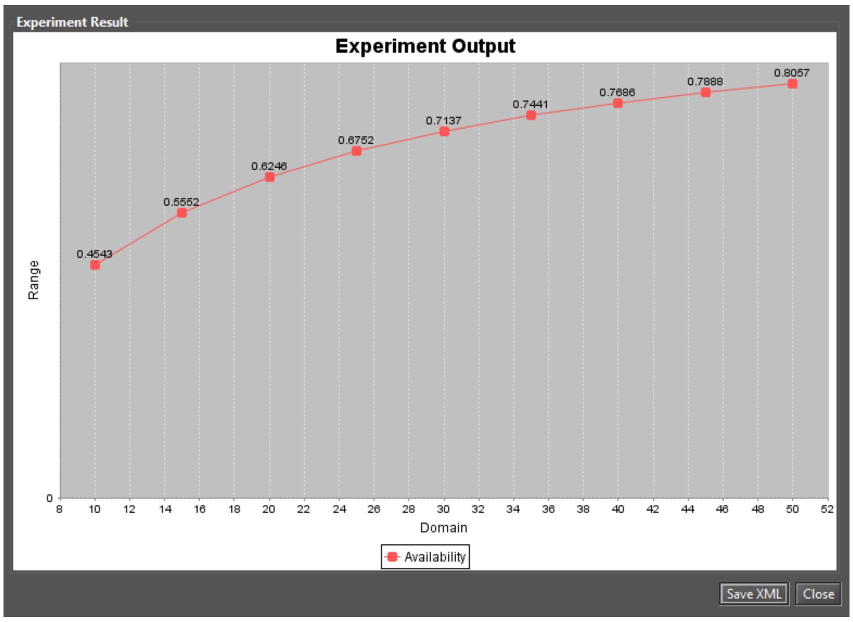
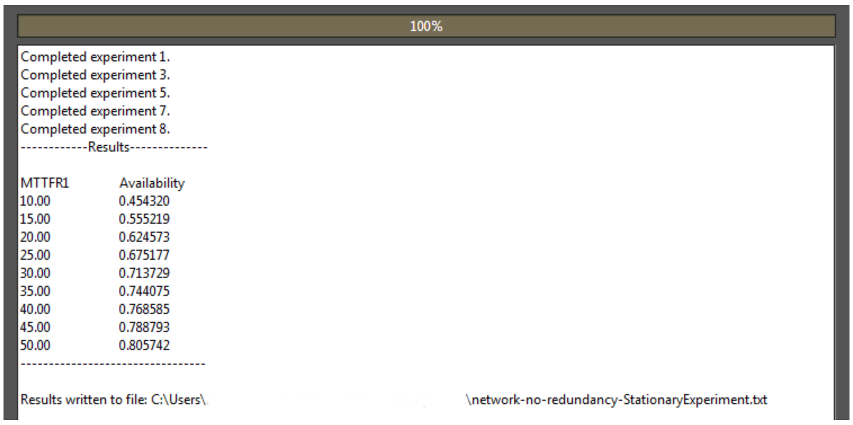


Figure 83: CTMC Experiment Graph



```
Completed experiment 1.
Completed experiment 3.
Completed experiment 5.
Completed experiment 7.
Completed experiment 8.
-----------Results-------------

MTTFR1        Availability
10.00         0.454320
15.00         0.555219
20.00         0.624573
25.00         0.675177
30.00         0.713729
35.00         0.744075
40.00         0.768585
45.00         0.788793
50.00         0.805742
-------------------------------

Results written to file: C:\Users\                    \network-no-redundancy-StationaryExperiment.txt
```

Figure 84: CTMC Experiment Textual Results

Analysis, which has a combo box for selecting one of two solution methods available: **Uniformization** (also known as Jensen's method) or **Runge-Kutta (4th order)**.

When solving the model through Uniformization, the user can define:

- The **time** for which the analysis will be carried out (default: 100);
- the **internal step** size (default: 0.1);
- the **precision** of results (default:10¯7);
- the **initial state probabilities** (default: 1 for first added state, 0 for the remaining states).

All those values are used by the Uniformization algorithm. Notice that the delay for obtaining results is proportional to the time informed for the analysis, because Uniformization is an iterative algorithm. The internal step size will affect accuracy of results and also execution time. By clicking on the **Run** button, the solution algorithm is triggered. As soon as it finishes, the results are presented in the text area on the bottom

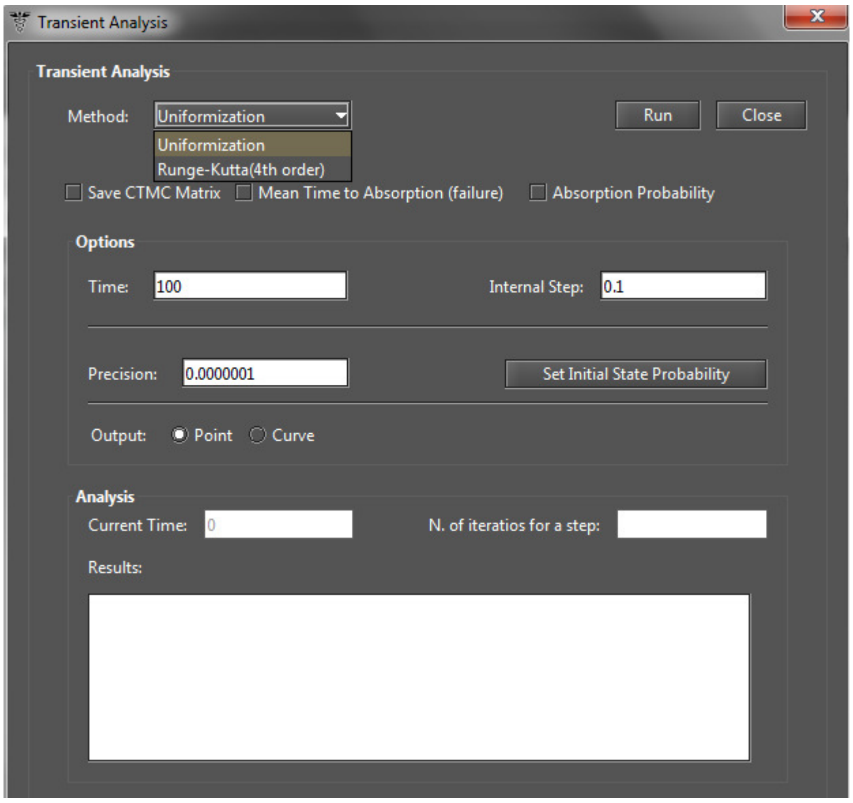


Figure 85: Transient analysis window

of analysis window, and written in a plain text file with the name of Mercury model file appended with the "-TransientAnalysis.txt" suffix.

The **Transient Analysis** window also allows the user choosing between a **"Point"** or **"Curve**" analysis. The **Point** analysis is the default, and shows the state probabilities only for the specific point in time. The **Curve** analysis writes in a plain text file all state probabilities computed from time equals zero until the specified time instant.

The **mean time to absorption** (MTTA) is a metric that can be computed in the Transient Analysis window, by checking the **Mean Time to Absorption (failure)** checkbox. The MTTA value is printed after state probabilities on Transient Analysis results text area. The user might also define a metric on drawing area, using the keyword **MTTA** as expression. The **absorption probability** for each state in the specified point in time is another metric that is accessible in Transient Analysis window.

4.5 Sensitivity Analysis

The Evaluation menu has a **"Sensitivity Analysis"** option, that enables computing partial derivative sensitivity indices for the CTMC model. Those indices indicate the impact that every input parameter has on one of the model's output metric.

Figure 86 depicts a CTMC built for representing the availability of a network comprising two routers and one link. This model was proposed in [5], and it has six parameters that affect the system availability. Those parameters are the mean time to failure (MTTF) and mean time to repair (MTTR) of each component: router 1

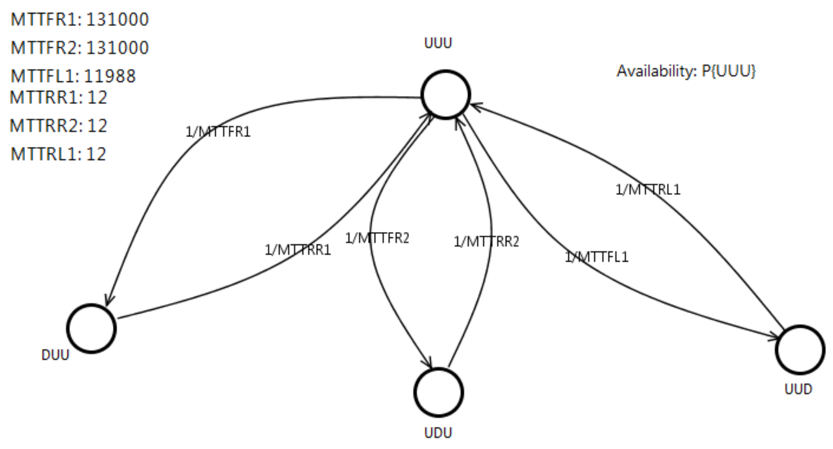


Figure 86: CTMC model for a network comprising two routers and one link

(R1), router 2 (R2), and link (L1). Their sensitivity ranking is computed in Mercury as illustrated in Figure 65. The sensitivity analysis windows has four options:

- The **type of sensitivity index** can be scaled or unscaled. When the user chooses scaled indices, every partial derivative is multiplied by the ratio of respective parameter value and metric value. This removes the influence of parameter units and provides sensitivity in a non-dimensional view. Unscaled indices are the raw results of partial derivatives. For more details on scaled and unscaled indices, please refer to [5] and [6].
- The **type of ranking** might be ordered or unordered. Usually ordered rankings are preferred for fast identification of most important parameters as well as those which have little impact on chosen metric.
- The **measure of interest** can be any user-defined measure of the CTMC model, for which the user is interested in assessing the sensitivity to input parameters. Please notice that if no measure was defined, no sensitivity analysis is possible. All measures can be evaluated at once.
- The **parameter of interest** can be any of the CTMC's input parameters. The user must choose between view the sensitivity of chosen measure with respect to just one parameter, or to all parameters.

5 DTMC Modeling and Evaluation

In the DTMC view, the user must click on the “Add New States” icon (see Figure 87), and then click on the drawing panel to create new states.

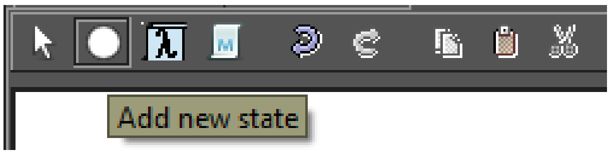


Figure 87: Adding a state to the DTMC

After adding the states, arcs are drawn by clicking in the center of the source state and dragging the created line until the target state, as shown in Figure 88.

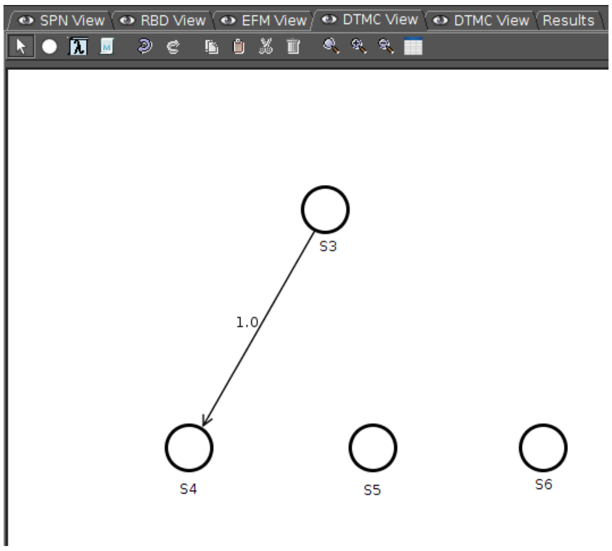


Figure 88: Adding a transition to the DTMC

The user can define the probability of each arc by right-clicking in the respective arc and clicking in the “Properties” item. Figure 89 shows the window where the arc probability can be defined.

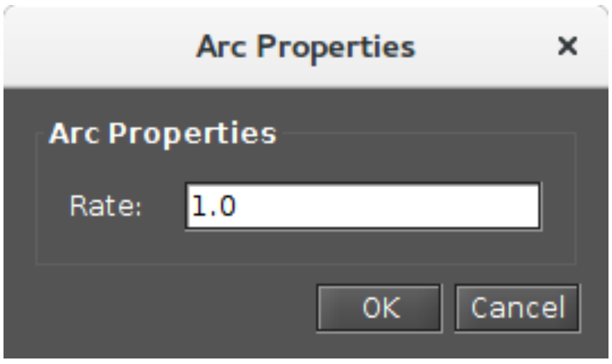


Figure 89: Defining the probability of the DTMC

After all states and arcs are properly defined (see Figure 90), it is possible to perform stationary and transient analyses for this model. The user can also save the probability matrix to a text file, by clicking in the matrix icon shown in Figure 91.

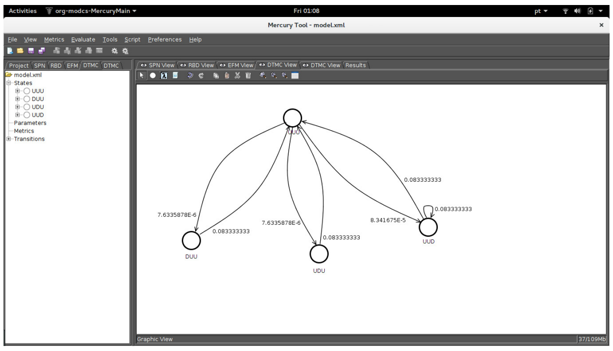


Figure 90: Example of DTMC model

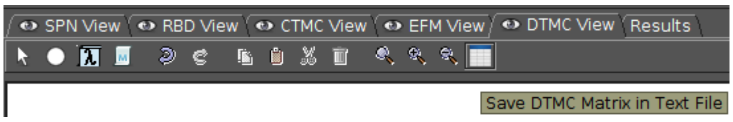


Figure 91: Saving the probability matrix to a file

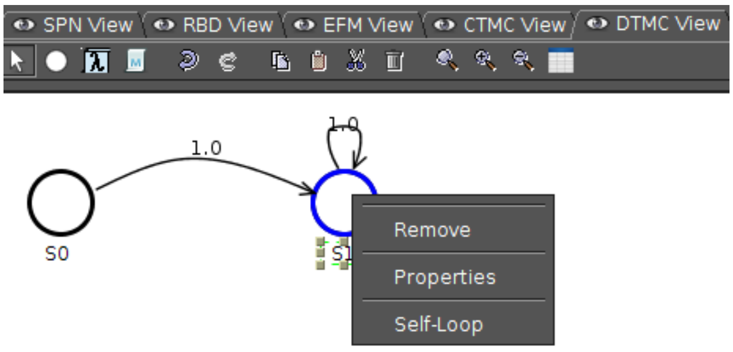


Figure 92: Adding a self-loop

It is worth mentioning that in the DTMC, the sum of the probabilities of the output arcs must be equal to 1. This restriction is verified before any evaluation of the DTMC model.

5.1 Input parameters

The arc probabilities of DTMCs can be defined through expressions containing both numbers and user-defined parameters. The button "Add new parameter definition" (see Figure 93) is represented by a λ symbol on DTMC menu bar, and it enables creating a symbolic parameter. After clicking in the button, click on any point in DTMC drawing area, and a new parameter is created with name **Param0** (or **Param1**, etc, if other parameters were already created). In order to change the parameter name and define a value for it, go to the Properties on right-click menu, as well as you can add self-loop in the desired place by right-clicking above the place and selecting self-loop. The name of the parameter can be defined by means of any combination of alphanumerical characters. Special characters (e.g., underscore, hyphen, ampersand) must be avoided. If names of greek letters are used for parameters, Mercury converts them to the proper symbol on greek lowercase alphabet (see Figures 94 and 95). The value assigned to the parameter can be a numerical expression. Any symbols or other parameter names are not allowed in parameter value.

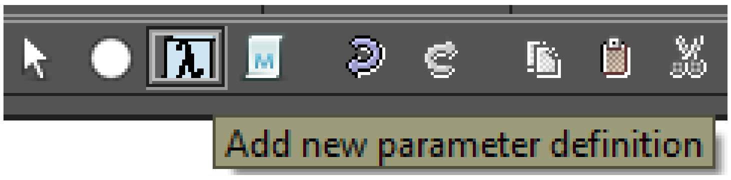
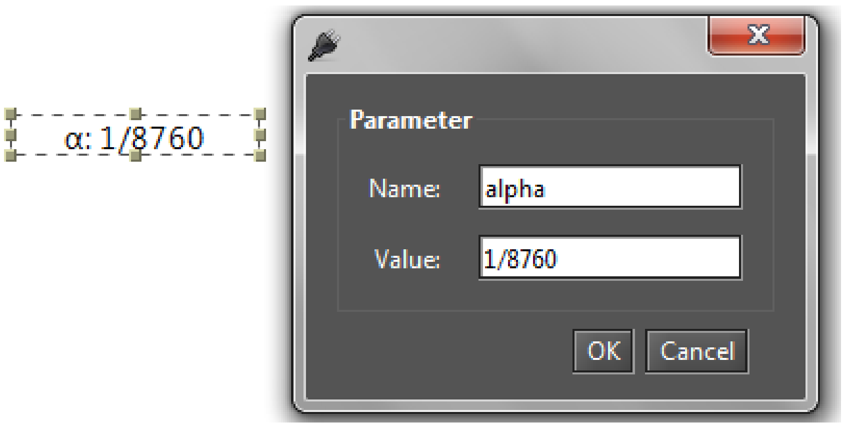


Figure 93: Adding a new user-defined parameter


α: 1/8760

Parameter

Name: alpha

Value: 1/8760

Figure 94: Modifying parameter name and value

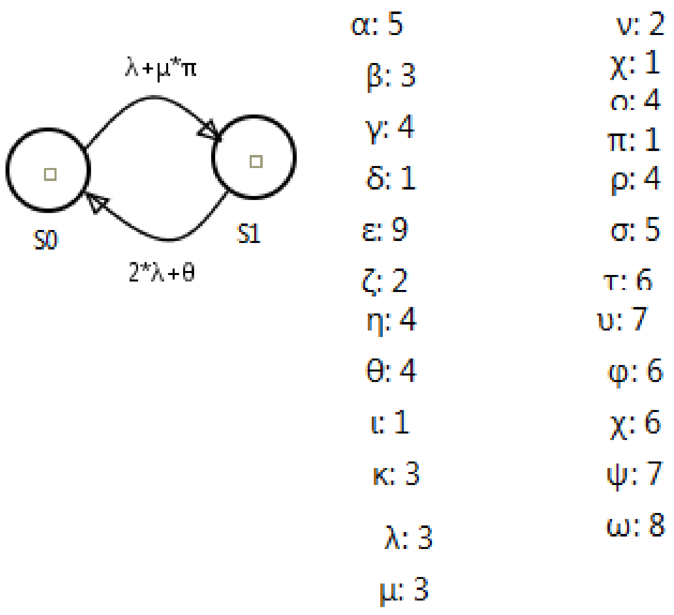


Figure 95: Parameter names defined with greek letters

5.2 Metrics

The button “Add new metric definition” (see Figure 96) enables defining metrics related to the DTMC model results. After inserting a metric, the user can change the name and define the expression that will be used to compute the metric. The metric expression syntax is based on state probabilities (**P{*state-name*}**). The probabilities of any states can be combined (i.e., added, subtracted, etc) in the metric expression. In the example of Figure 97, the metric **AvB3** indicates the availability of the system represented by the two-state DTMC created with Mercury. The availability will be computed from expression **P{Up}**, that is the probability of being in state **Up**. When the model is evaluated through stationary or transient analysis, the metric value will be updated and displayed on screen, as shown in Figure 98. The system mean time to absorption (when there is at least one absorbing state) can be computed in a metric expression using the keyword **MTTA**.

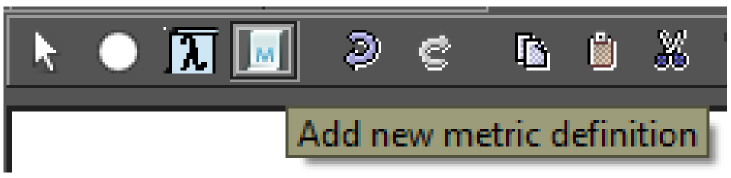
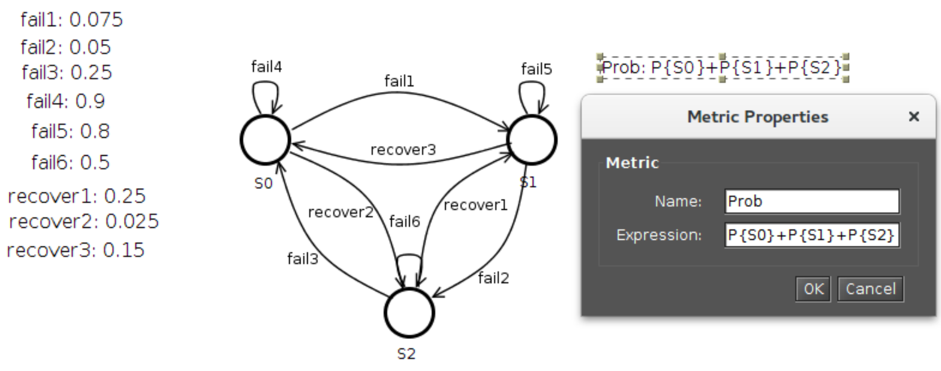


Figure 96: Adding a metric



Name: Prob
Expression: P{S0}+P{S1}+P{S2}

Figure 97: Defining metric name and expression

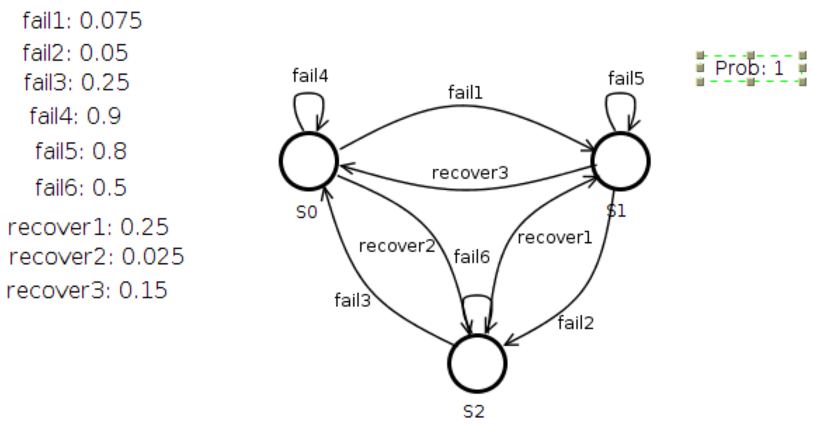
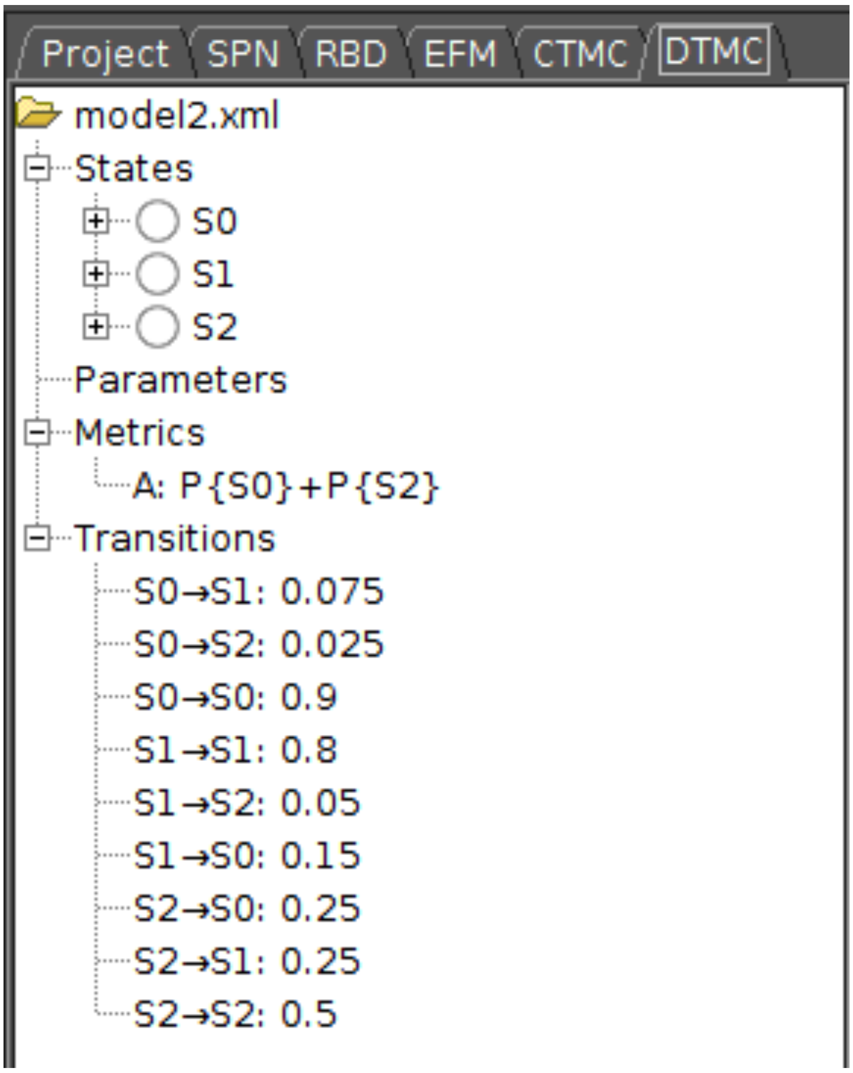


Figure 98: Metric value

The metric expression is still visible in the metric properties window, and in the **left-side panel** (see Figure 99). The panel shows all elements of the DTMC: state names; parameters names and values; metric names and expressions; and acs. Every arc is represented in the side panel by the source and target states, followed by the expression or value assigned to that arc.



```
model2.xml
States
    S0
    S1
    S2
Parameters
Metrics
    A: P{S0}+P{S2}
Transitions
    S0→S1: 0.075
    S0→S2: 0.025
    S0→S0: 0.9
    S1→S1: 0.8
    S1→S2: 0.05
    S1→S0: 0.15
    S2→S0: 0.25
    S2→S1: 0.25
    S2→S2: 0.5
```

Figure 99: DTMC side panel

5.3 DTMC Stationary Analysis

When the DTMC View is active, the menu "Evaluation" shows two options for model analysis: **Stationary Analysis** and **Transient Analysis**. The Stationary Analysis computes steady-state probabilities, useful for assessing the long-term average behavior of modeled system. Figure 100 shows the window of Stationary Analysis, which has a combo box for selecting one of two solution methods available: **Direct - GTH** (Grassmann-Taksar-Heyman) or **Iterative - Gauss-Seidel**.

When solving the model through GTH, the user can change the **maximum error** used in the algorithm. The default value for the maximum error is 0.0000001 (10^-7). By clicking on the **Run** button, the solution

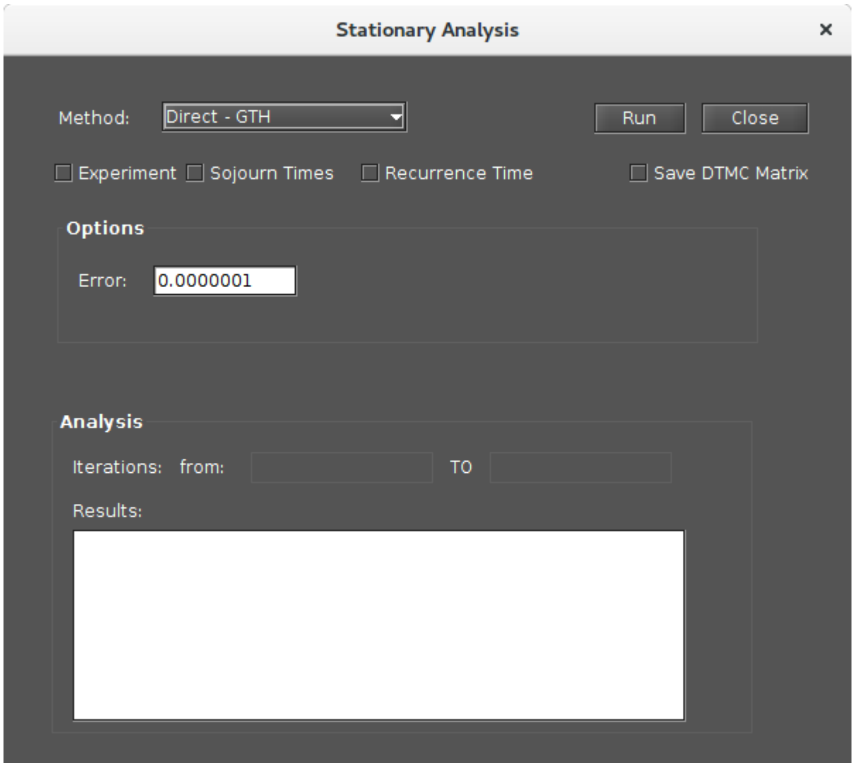
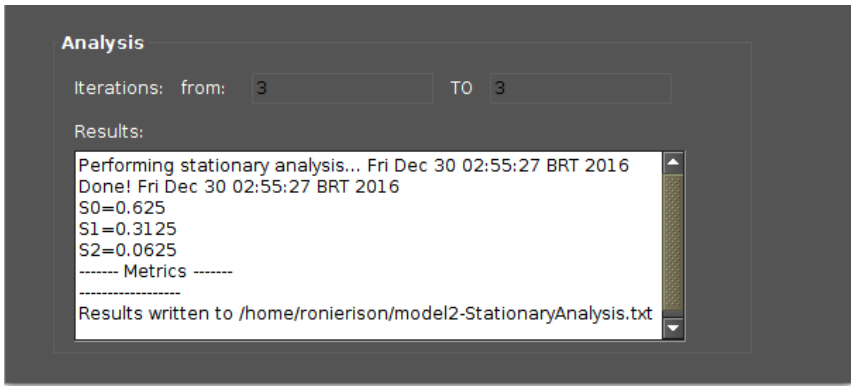


Figure 100: Stationary analysis window

algorithm is triggered and as soon as it finishes, the results are presented in the text area in the bottom of analysis window (see Figure 101), and written in a plain text file with the name of Mercury model file appended with the " "-StationaryAnalysis.txt" suffix.



```
Performing stationary analysis... Fri Dec 30 02:55:27 BRT 2016
Done! Fri Dec 30 02:55:27 BRT 2016
S0=0.625
S1=0.3125
S2=0.0625
------- Metrics -------
-------------------
Results written to /home/ronierison/model2-StationaryAnalysis.txt
```

Figure 101: Stationary analysis results

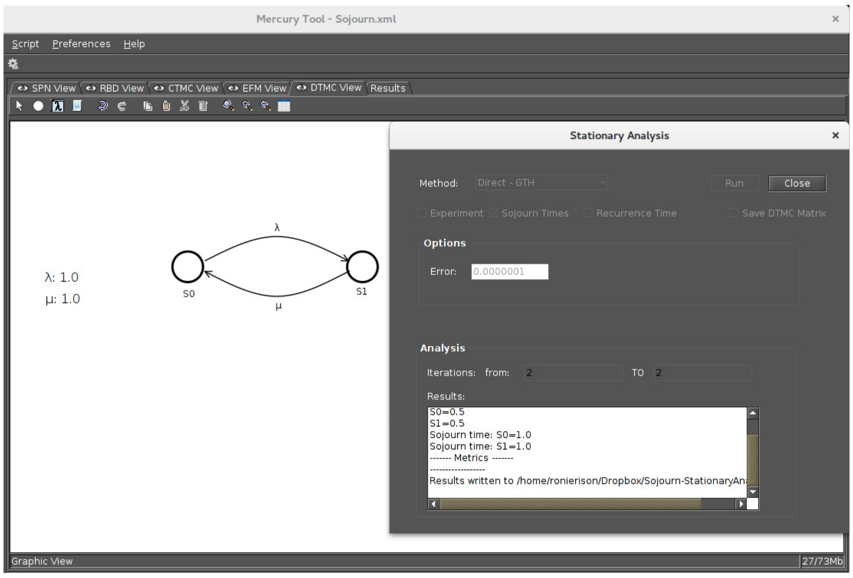
When solving the model through Gauss-Seidel, besides maximum error, the user can also change the maximum number of iterations. The default value for such a parameter is -1, indicating that the algorithm only stops when convergence of results is reached, considering the informed error.

The user-defined metrics are updated as soon as the analysis completes, independently of the chosen solution method. Their values are shown in the drawing area, as illustrated in Figure 81, where the metrics are in right side.

The DTMC model can also be solved for a range of values of user-defined parameters. This is accomplished by clicking in the **Experiment** check box on Stationary Analysis window and then on the **Run** button. A new dialog opens for definition of the experiment options (see Figure 82). The options are: the parameter that will vary; the minimum and maximum value for this parameter; the step size for incrementing the parameter value in the defined range; and the output metric that will be shown on experiment results and plotted on a graph (see Figures 83 and 84).

Mercury also computes the following metrics when the user requests the stationary analysis of a DTMC:

Sojourn times: A state i is said to be a sojourn state if there exists a state $j, i \neq j$, which is accessible from state i, but not vice versa, that is, state i is not accessible from state j. In other words, a state is a sojourn, if once entering its state, the process can never return to this state again. A communication class consisting of sojourn states is called a permanent class, "the process will leave this class and never return with probability 1."



Method: Direct - GTH

Experiment [✓ Sojourn Times] Recurrence Time Save DTMC Matrix

Options

Error: 0.0000001

Analysis

Iterations: from: 2 TO 2

Results:

```
S0=0.5
S1=0.5
Sojourn time: S0=1.0
Sojourn time: S1=1.0
------- Metrics -------
-----------------
Results written to /home/ronierison/Dropbox/Sojourn-StationaryAn
```

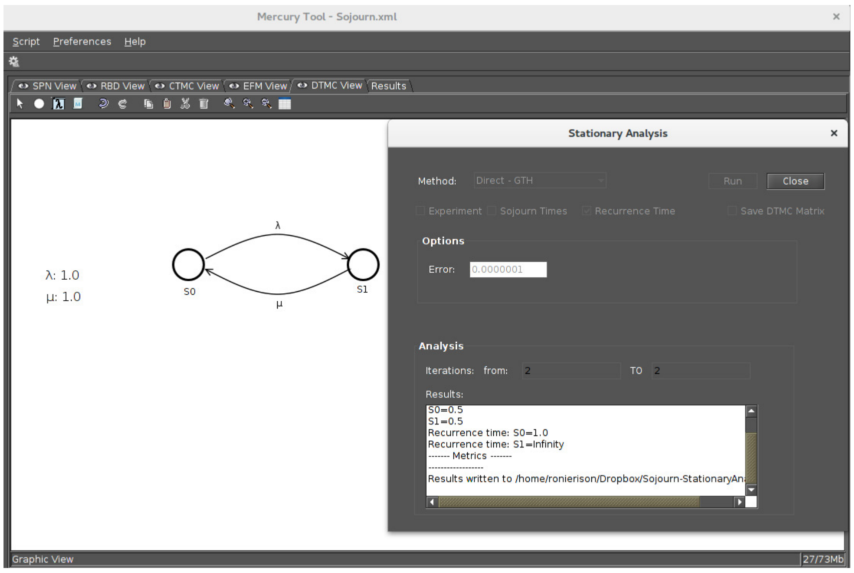
Figure 102: Sojourn Time

Recurrence time: A state i is said to be a recurring state if, once entering, it will certainly return to that state again. A Markov chain is recurrent if each state in the chain is recurrent. A recurrent class is a communication class consisting of recurring states, a Markov chain never leaves a recurring class, once it has entered. A state is recurrent if the chain visit this state with probability 1. This means that the chain visits a recurring state countless times, however, if the state i is then the sojourn time of the next visit of the state i is a geometric random variable.

5.4 DTMC Transient Analysis

The Transient Analysis option in the "Evaluate" menu computes time-dependent probabilities, useful for assessing the behavior of modeled system in a particular point in time. Figure 104 shows the window of Transient Analysis, where when solving the model the probability of the time step n is found by multiplying the *n-power* of the probability matrix by the initial state probability.

When solving the model, the user can define:



Method: Direct - GTH

Experiment [✓ Recurrence Time] Sojourn Times Save DTMC Matrix

Options

Error: 0.0000001

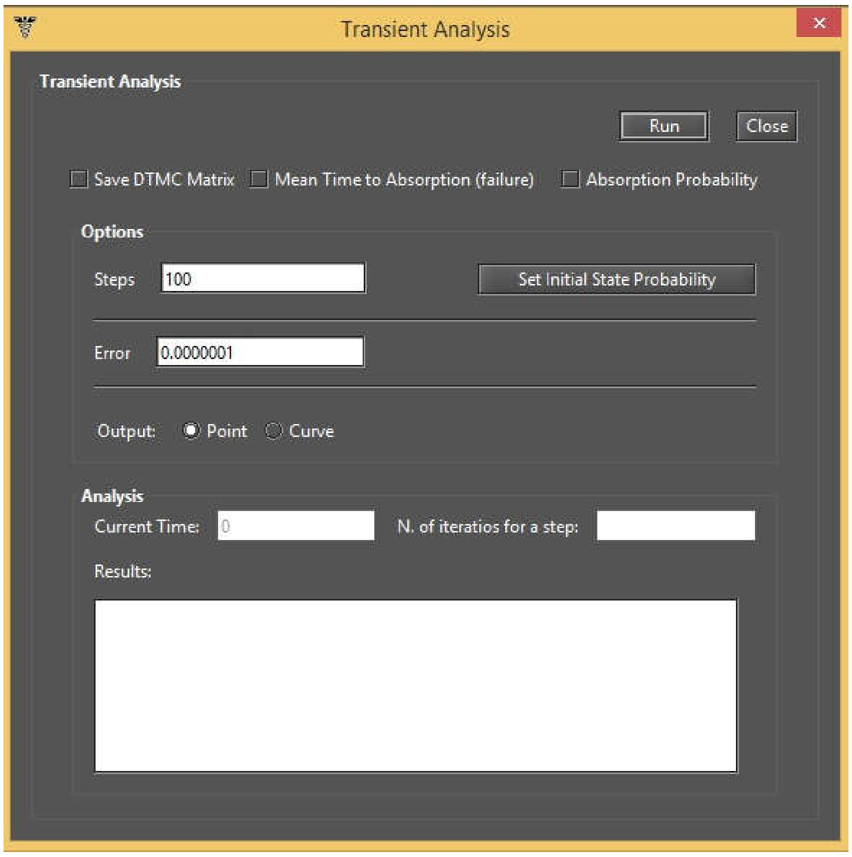
Analysis

Iterations: from: 2 TO 2

Results:

```
S0=0.5
S1=0.5
Recurrence time: S0=1.0
Recurrence time: S1=Infinity
------- Metrics -------
------------------
Results written to /home/ronierison/Dropbox/Sojourn-StationaryAn
```

Figure 103: Recurrence Time



Transient Analysis

Save DTMC Matrix Mean Time to Absorption (failure) Absorption Probability

Options

Steps 100

Error 0.0000001

Output: [✓ Point] Curve

Analysis

Current Time: 0 N. of iteratios for a step:

Results:

Figure 104: Transient analysis window

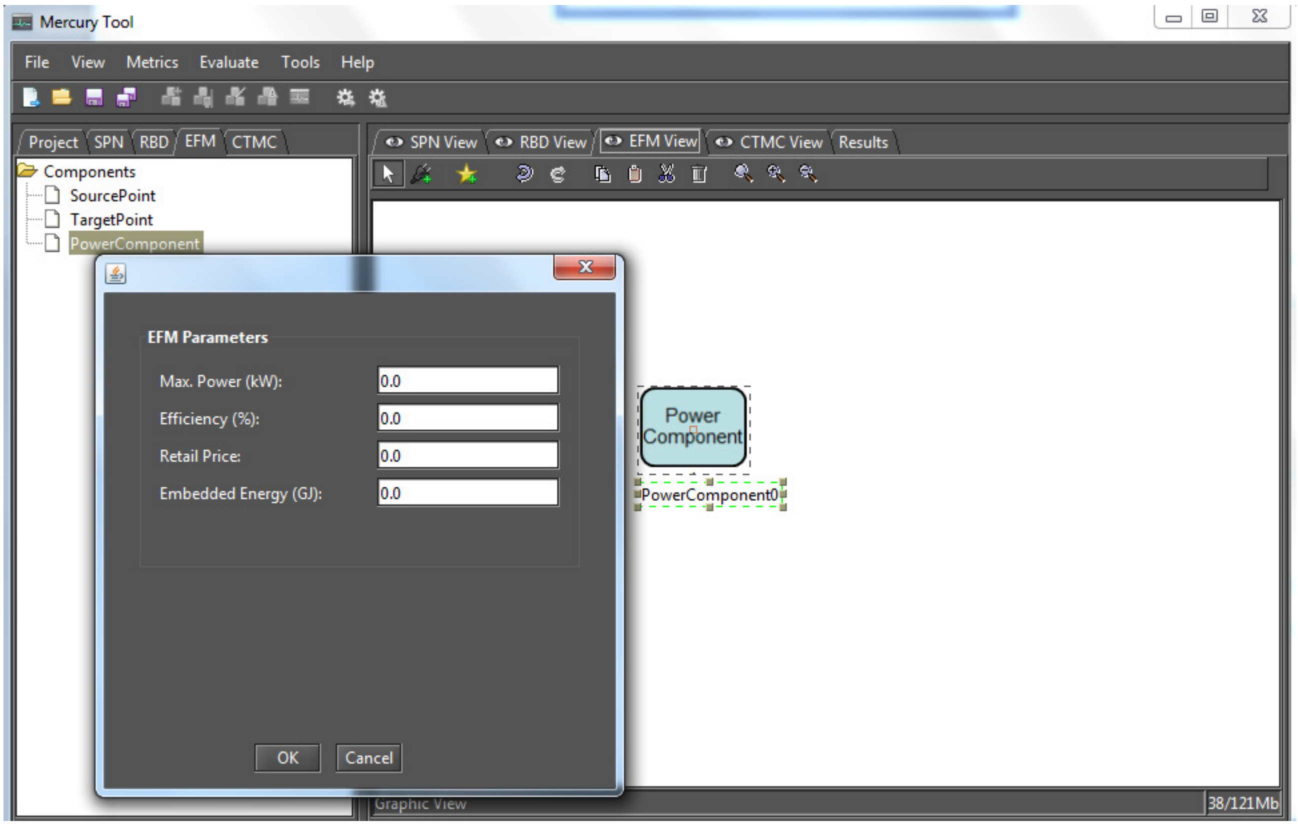
- The **steps** for which the analysis will be carried out (default: 100);
- the **error** of results (default:0.0000001)
- the **initial state probabilities** (default: 1 for first added state, 0 for the remaining states).

The internal step size will affect accuracy of results and also execution time. By clicking on the Run button, the solution algorithm is triggered. As soon as it finishes, the results are presented in the text area on the bottom of analysis window, and written in a plain text file with the name of Mercury model file appended with the "-TransientAnalysis.txt" suffix.

The Transient Analysis window also allows the user choosing between a "Point" or "Curve" analysis. The Point analysis is the default, and shows the state probabilities only for the specific point in time. The Curve analysis writes in a plain text file all state probabilities computed from time equals zero until the specified time instant.

The mean time to absorption (MTTA) is a metric that can be computed in the Transient Analysis window, by checking the Mean Time to Absorption (failure) checkbox. The MTTA value is printed after state probabilities on Transient Analysis results text area. The user might also define a metric on drawing area, using the keyword MTTA as expression. The absorption probability for each state in the specified point in time is another metric that is accessible in Transient Analysis window.

6 EFM Modeling and Evaluation



EFM Parameters

Max. Power (kW): 0.0

Efficiency (%): 0.0

Retail Price: 0.0

Embedded Energy (GJ): 0.0

OK Cancel

Figure 105: Property Screen Components

Energy Flow Model (EFM) is proposed to estimate the sustainability impact and cost of data center architectures without overstepping the energy constraints of each device. This is accomplished with algorithms that traverse the EFM and compute the cost, estimate the environmental impact and verify the energy flow.

The EFM evaluation functionality is responsible for estimating sustainability impacts of a system (e.g., model) in terms of its lifetime exergy (available energy) consumption. This functionality also computes the total cost that is composed by Initial cost and Operational cost. The initial cost represents the budged needed to obtain the system components in order to build the system. The operational cost is the cost to maintain the system in the operational mode.

In order to perform the sustainability evaluation, users should create an EFM model. To add power or cooling components in the model, users should click on the start icon. Once clicked in that button, a window in which it is possible to select the component to be added is shown (see Figure 106). The next step is to add the component to the model as depicted in Figure 107. Once the components are added to the model the user should connect the components including SourcePoints and TargetPoint as shown in Figure 108. After the addition of a component, a list is shown on the left side of the screen. A right click on the component provides a way to edit their properties as shown in Figure 105.

Additionally, it is important to stress that users have to set the demanded power on the TargetPoint (for

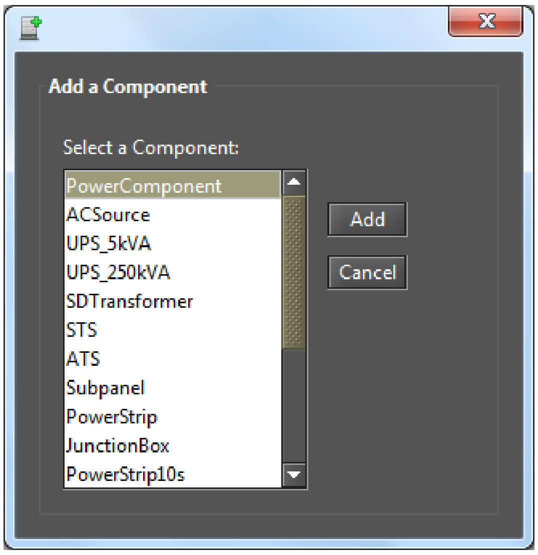


Figure 106: Selecting the component to be added.

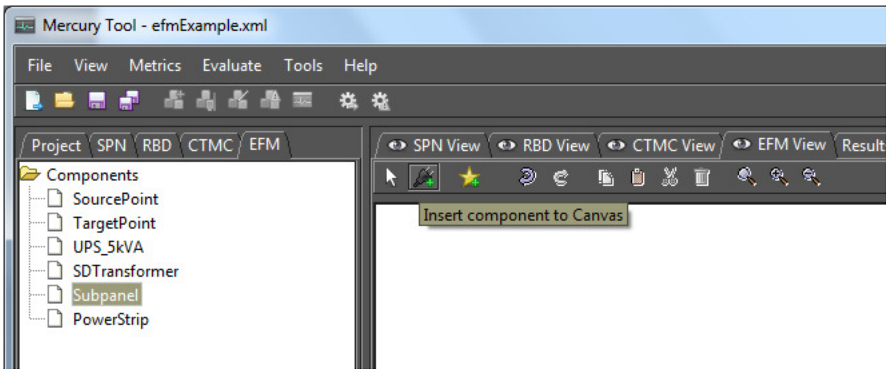


Figure 107: Inserting the component to the model.

power system) or on the SourcePoint (for cooling system). This is done by with a right click in the Source or Target points and selecting the option Properties. After that the evaluation can be conducted. The evaluation is performed through the Menu option shown in Figure 109. Once selected the combined option, the EFM evaluation of cost, exergy and energy flow is selected. To conduct those evaluations, users have to provide the EFM parameters depicted in Figure 110. Finally, the results is presented (see Figure 111). In this example, the result indicates that the energy flow evaluation is true meaning that the power constraints present on each device were respected. However, in case this result is false, the system shows the component in which the constraint was crossed (see Figure 112). In this window of results, the user has also an option to export the results to an spreadsheet (e.g., a file .xls).

6.1 Power Load Distribution Algorithm - PLDA

A Power Load Distribution Algorithm (PLDA) is proposed to minimize the electrical energy consumption of the EFM models [7]. The PLDA is based on the Ford-Fulkerson algorithm, which computes the maximum possible flow in a flow network [8]. The network is represented by a graph, where the transport capacity of the devices is defined in the edges. The algorithm begins by traversing the graph, searching for the best flows between two

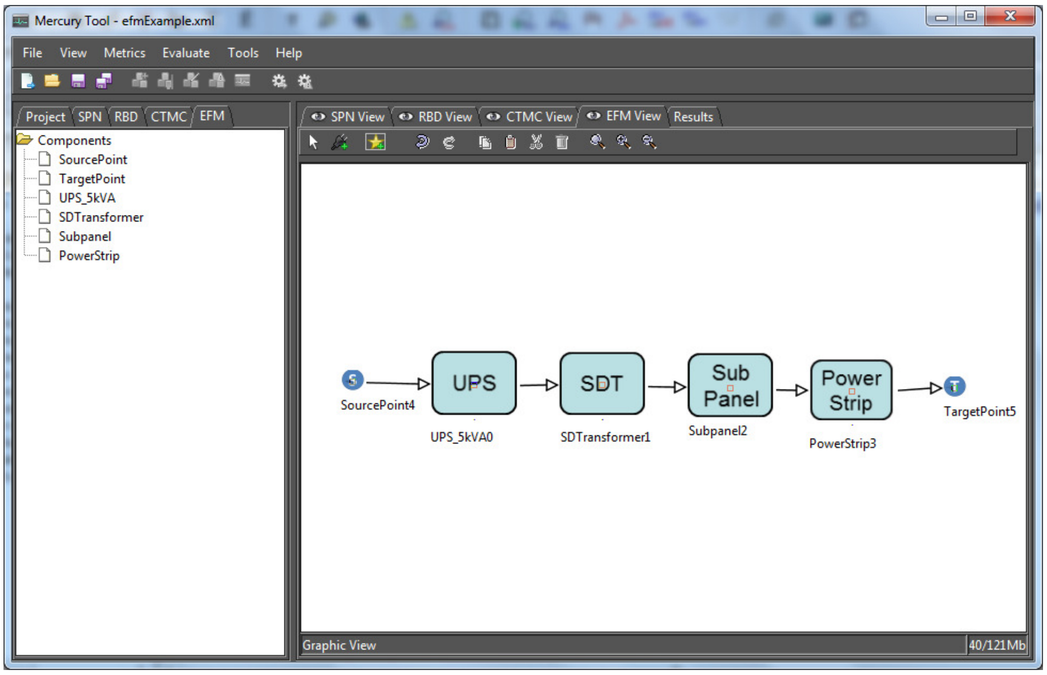


Figure 108: EFM example.

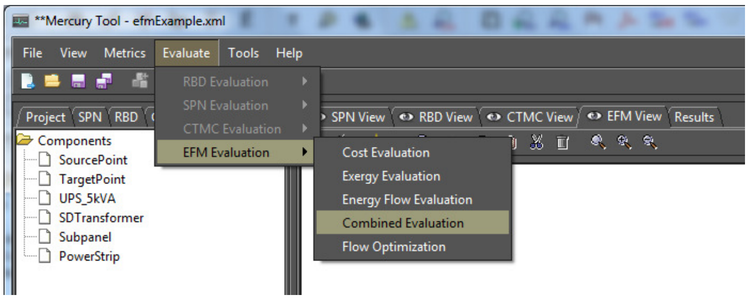


Figure 109: EFM menu.

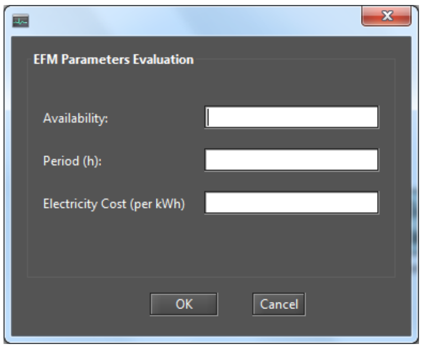
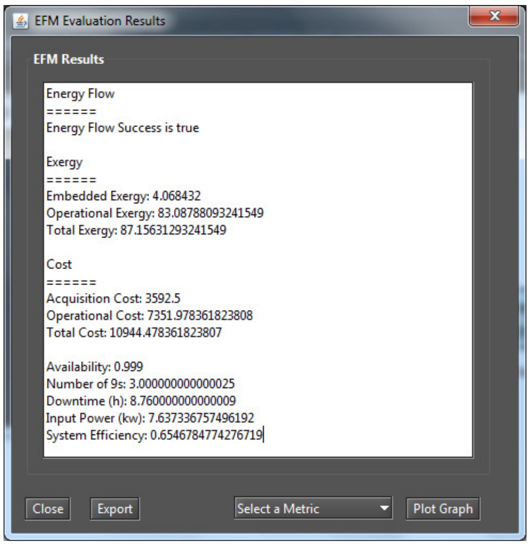


Figure 110: EFM parameters.

specific points in the graph. If a particular path lacks the capacity to support all of the flow demanded, then the residual flow is redirected to other paths. The Priority First Search (PFS) is the adopted method for selecting the path between the nodes [9, 10]. The PFS chooses the path according to the highest electrical capacities of nodes in the graph [11].

The Figue 113 show how to call the PLDA function.



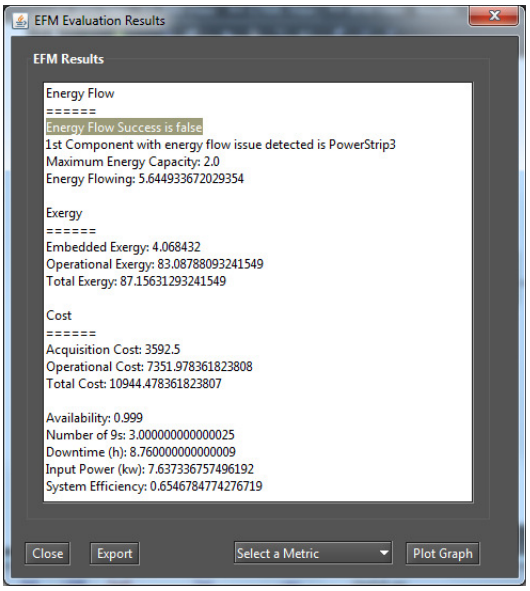
```
Energy Flow
======
Energy Flow Success is true

Exergy
======
Embedded Exergy: 4.068432
Operational Exergy: 83.08788093241549
Total Exergy: 87.15631293241549

Cost
======
Acquisition Cost: 3592.5
Operational Cost: 7351.978361823808
Total Cost: 10944.478361823807

Availability: 0.999
Number of 9s: 3.000000000000025
Downtime (h): 8.760000000000009
Input Power (kw): 7.637336757496192
System Efficiency: 0.6546784774276719
```

Figure 111: EFM results.



```
Energy Flow
======
Energy Flow Success is false
1st Component with energy flow issue detected is PowerStrip3
Maximum Energy Capacity: 2.0
Energy Flowing: 5.644933672029354

Exergy
======
Embedded Exergy: 4.068432
Operational Exergy: 83.08788093241549
Total Exergy: 87.15631293241549

Cost
======
Acquisition Cost: 3592.5
Operational Cost: 7351.978361823808
Total Cost: 10944.478361823807

Availability: 0.999
Number of 9s: 3.000000000000025
Downtime (h): 8.760000000000009
Input Power (kw): 7.637336757496192
System Efficiency: 0.6546784774276719
```

Figure 112: EFM results with the energy flow evaluation false.

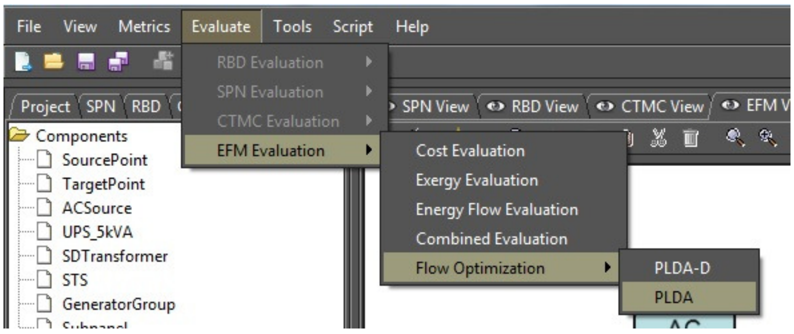
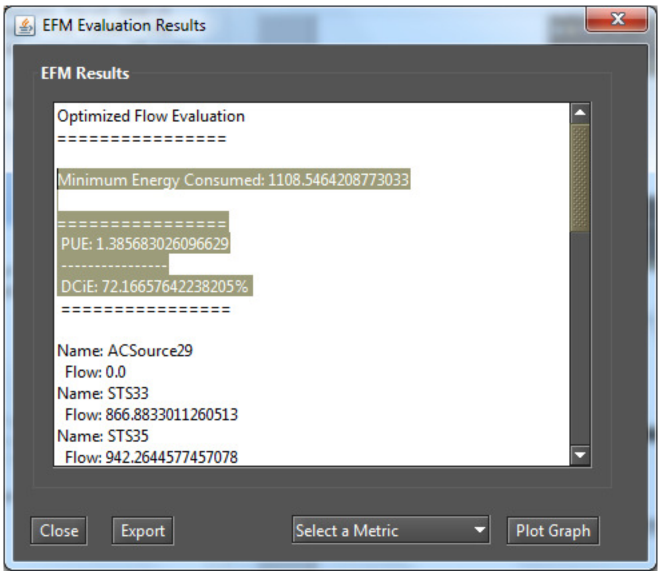


Figure 113: PLDA optimization function

Figure 114 depicts the results of the PLDA algorithm, with the mininum energy consumed, PUE and DCiE highlighted.



```
Optimized Flow Evaluation
================

Minimum Energy Consumed: 1108.5464208773033

================
 PUE: 1.385683026096629
 ---------------
 DCiE: 72.16657642238205%
 ================

Name: ACSource29
  Flow: 0.0
Name: STS33
  Flow: 866.8833011260513
Name: STS35
  Flow: 942.2644577457078
```

Figure 114: PLDA optimization results

6.1.1 Example of PLDA execution

Figure 115 illustrates the EFM model of a specified architecture. In the example, all the edge weights are set to the default value, one. The power flow is computed by traversing the graph from the target to the source node.

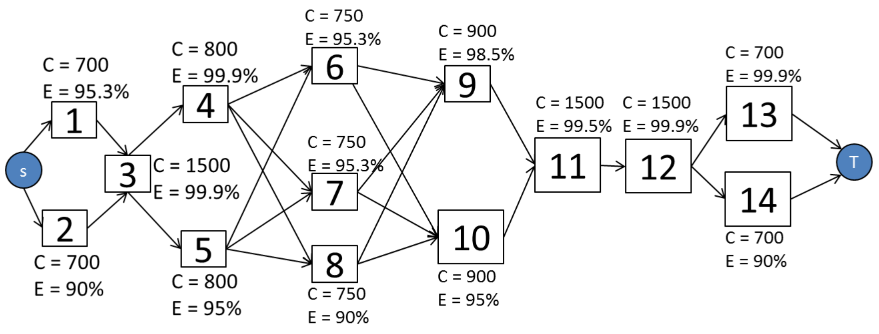


Figure 115: EFM model.

Figure 116 depicts the EFM model after the execution of the PLDA. It should be noted that the weights on the edges have changed, optimizing the power flow through a best weights distribution.

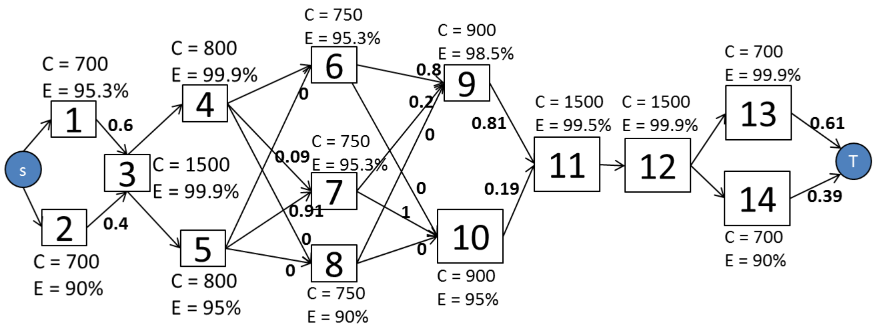


Figure 116: EFM model after PLDA execution.

Table 1 presents a summary of the results obtained by the PLDA. Column "*Improvement*" depicts the improvement. The system efficiency is improved by over 4.2%; consequently, the associated cost and sustainability figures are improved by 4.2% and 20.4%, respectively. Availability results were also computed with RBD/SPN models, but are not included here.

Table 1: Summary results before and after PLDA execution.

Metric	**Before**	**After**	**Improvement (%)**
Availability (%)	0.99999226	0.99999226	0
Number of 9 s	5.111	5.111	0
Downtime (hs)	0.0677	0.0677	0
Input Power (kW)	1,312.63	1,259.64	4.2
System Efficiency (%)	76.18	79.38	4.2
Operational Cost (USD)	1,264,849	1,213,784	4.2
Operational Exergy (GJ)	9,859.32	8,188.11	20.4

6.2 Power Load Distribution Algorithm in Depth search (PLDA-D)

A Power Load Distribution Algorithm - Depth (PLDA-D) is proposed to minimize the electrical energy consumption of the EFM models [7]. Is an evolution of the PLDA 6.1 algorithm, apply for the same problem but with a big differ in technique of graph search. In the PLDA-D the model EFM is searched in depth, choosing always the best path in a depth search to distribute the weights of the edges.

The PLDA-D is based in the Bellman [12], Ford and Fulkerson [8] flow algorithm, but with many adaptations. The PLDA-D is divided in three phases: Initialize, kernel and the search of best path.

The Figue 117 show how to call the PLDA function.

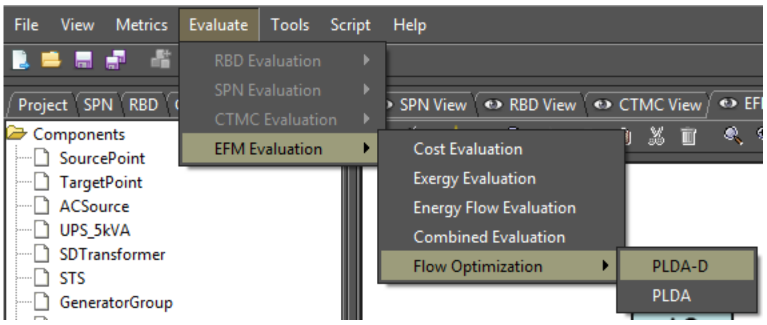
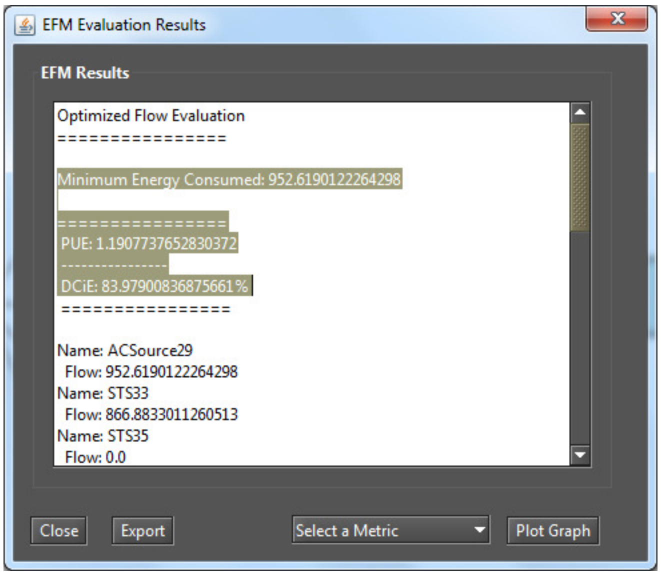


Figure 117: PLDA-D optimization function

Figure 118 depicts the results of the PLDA-D algorithm, with the mininum energy consumed, PUE and DCiE highlighted.



```
Optimized Flow Evaluation
================

Minimum Energy Consumed: 952.6190122264298

================
 PUE: 1.1907737652830372
 ---------------
 DCiE: 83.97900836875661 %
 ================

Name: ACSource29
  Flow: 952.6190122264298
Name: STS33
  Flow: 866.8833011260513
Name: STS35
  Flow: 0.0
```

Figure 118: PLDA-D optimization results

6.2.1 Example of PLDA-D Execution

Figure 119 illustrates the step-by-step of the PLDAD execution, highlighting some variables, edges and weights. Lets consider the model represent by Figure 119.a with three electrical components A, B, C, each one with efficiency 80, 90 and 95 % respectively. This means that if a component has efficiency of 90%, 10% of the energy that passes through it is lost. The others symbols of the model are S for the node $Source$, with can be represented by an electrical utility and T, for the node $Target$, with can be represented by a computer room.

In the example, the demand (Dem) and the Efficiency (Ef) values are know. The value of the $Target$ node Acc is set to one. The others accumulated cost (Acc) are set do zero and the edge weights are set to the default value one, respectively, as depicted in Figure 119.a, a perfect representation of a EFM model. The phase one of the PLDAD algorithm is represented by the Figure 119.b when all the variables are initialized in all vertices. Actual Cost ($ActCost$) to infinite, child to null ($Child$) and Accumulated cost to zero (Acc).

Phase two starts in the Figure 119.c following to the Figure 119.h. At this stage the best path is selected according to the efficiency of each component, through a scan in depth and respecting the limits of capacity of each equipment. In Figure 119.c the values of the $ActualCost$ and $AccumulatedCost$ are computed and the best child is choose, according the lower value of the variable $ActCost$. This value is used to select the best child for a given node.

A very important step in this phase is represented by Figure 119.g. After the calculations of the variables Acc and $ActCost$, it was verified that the $ActCost$ for the current path (3.39) was less than the $ActCost$ of the previous path (3.68) to get to the $Source$ node. Thus, the $Source$ node has change in the values of its variables and the best $Child$ now is the node B and not more node A. In other words, it is less costly to get to the source node for B by A, so B represents a better path than A.

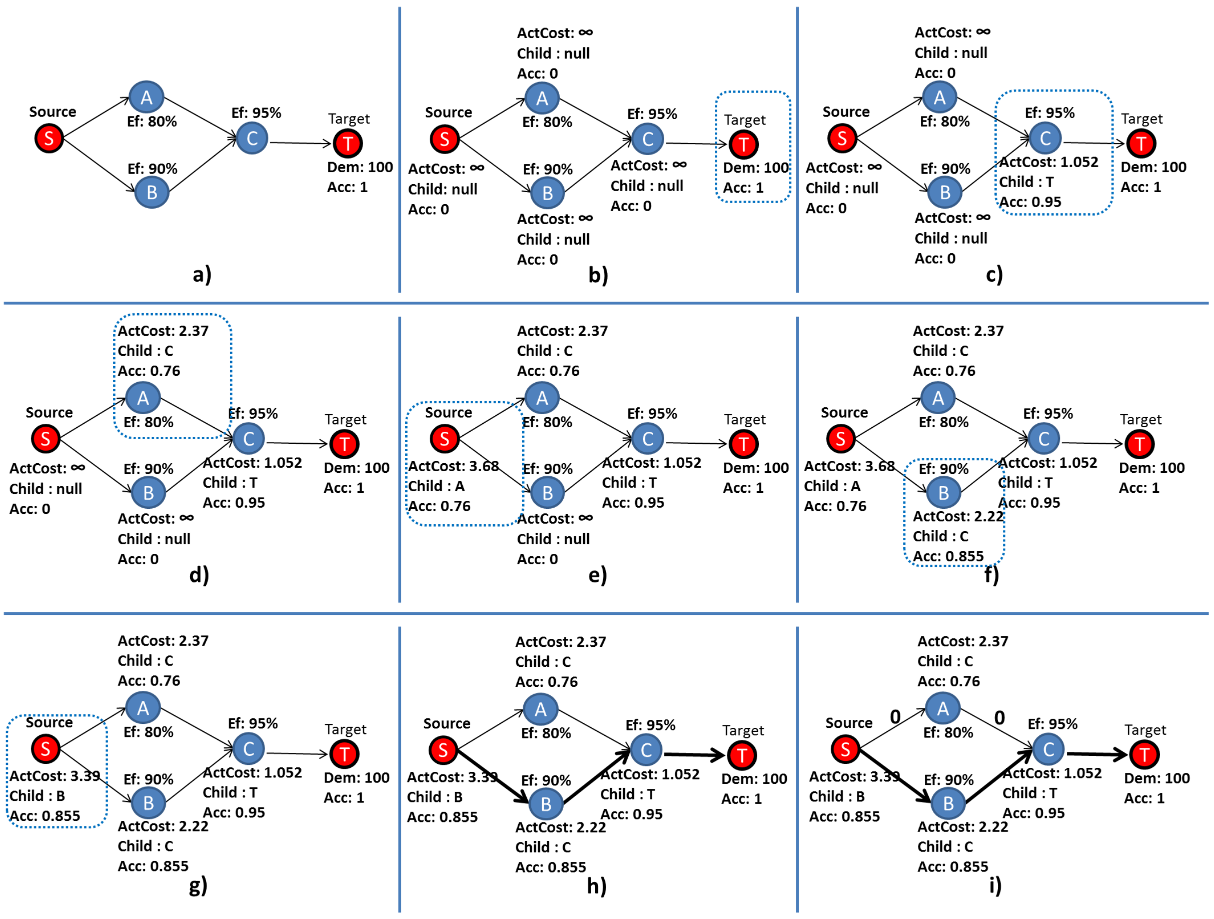


Figure 119: Exemple PLDAD execution

The Figure 119.h represents the end of the phase three. For this example, the best path from note Target to Source is: $Target, C, B, Source$. In the Figure 119.i the flow is distributed, according to the weights of the edges. With these values, the EFM computes the minimum possible value for the input power, reducing all the values associated with data center power consumption.

7 FT Modeling and Evaluation

Fault Trees (FT) and RBDs models differ from each other in their purpose, while the FT focus on the events that lead something to enter into the failure node the RBDs focus on the events that make each block to be working. In the FT view, the standard model presents a top event called **FAILURE**; this failure has the word **underfined** as a description since no event leads to It. The Figure 120 shows the FT view start screen.

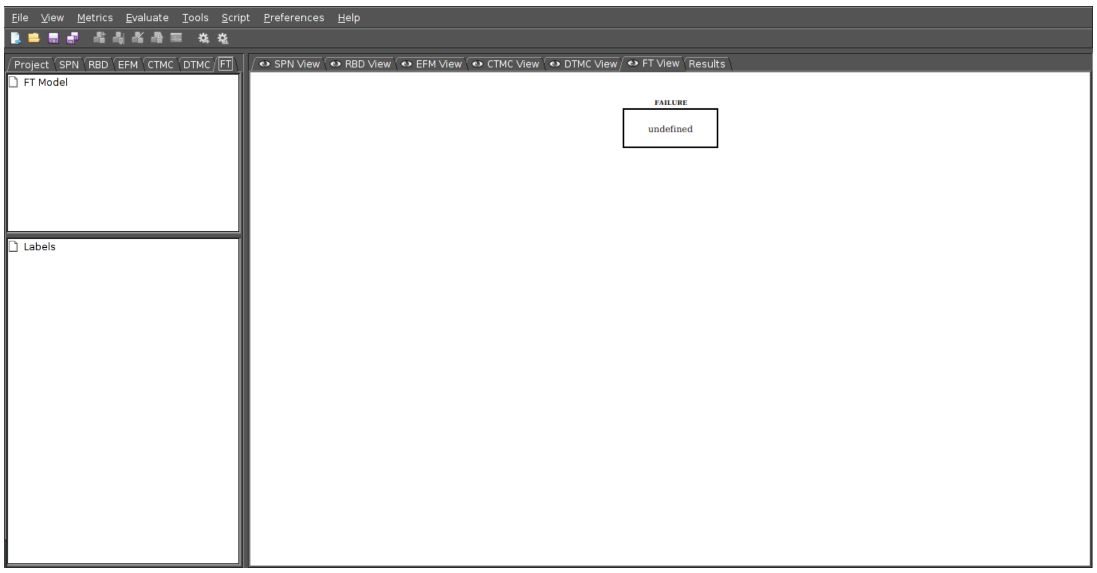


Figure 120: Initial FT Model

The events that lead to the **FAILURE** node must be bound to a **GATE**, in order to evaluate the probability of an event to happen based on the leaf junctions probability and other gates connected to the main gate. In order to create a gate the user must press the right mouse button on the **FAILURE** event, Figure 121 shows how the adding new gate screen. Furthermore, users can choose between three different gates: *AND, OR, KooN*.

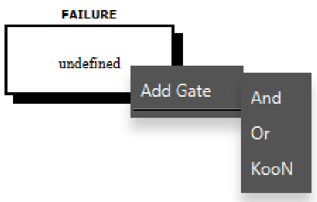


Figure 121: Adding New Gates

One of the essential features related to the act of adding gates is the fact that we need at least two events to evaluate a gate, when you add a new gate in your model you need to choose a number of leaf nodes bigger than one, the Mercury adds two leaf nodes automatically if you leave the field empty.

After choosing the gate that fills the model needs, the user reaches the next screen, the Figure 122 presents the gate properties. Each gate has a **Name** as an Identification parameter, which cannot be modified; the **(N)umber of Events** similar to this gate; a **Description** field that may be used to store information about the event that is the result of the gate. Also, It is important to highlight that the **K Value** field is only used by KooN gates to present how many events must occur in order to failure happen.

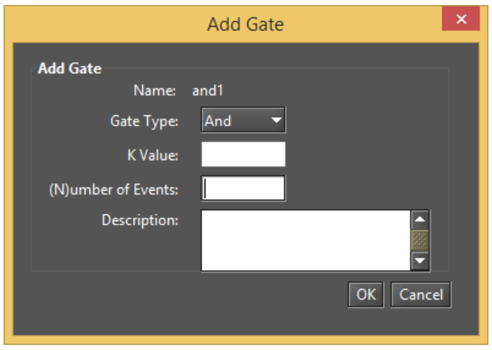


Figure 122: Add Gate

A double left click over an event node leads to the description edition screen. The Figure 123 shows the new gate properties screen, which resembles the Add Gate screen (Figure 122). Also, it is important to mention that the description field is the only field enabled on this screen.

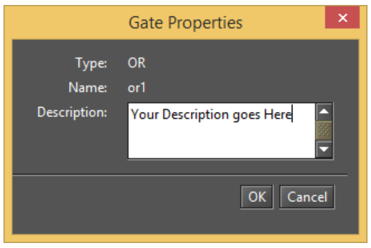
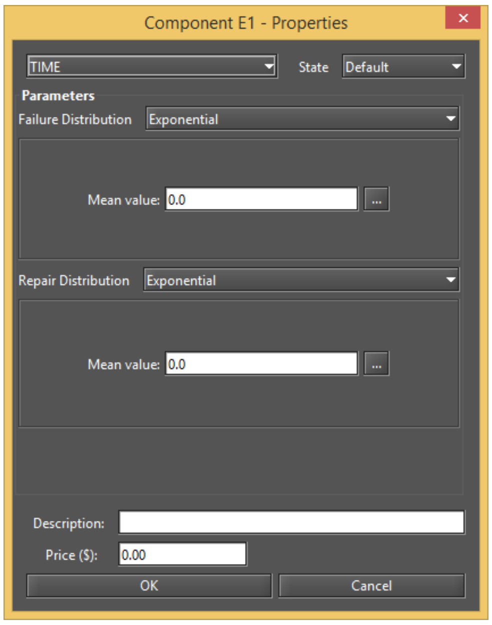


Figure 123: Gate Properties

Two restrictions follow the adding gates propriety: **(i)** You cannot add a new gate to a KooN gate related event; this is just possible to events related to AND or OR gates. **(ii)** You cannot edit an event leaf name, which means that a solo event related to a gate is unable to receive another name when you double-click it, then the properties screen opens, the Figure 124 shows it.

The leaf node properties is a similar screen to the one presented in the RBD view; later the user is going to see the conversion process to transform an FT into an RBD. The FT allows the edition of leaf nodes with different distributions. The properties that can be modified are:

- **Parameters Type:** There are four parameters type available to be chosen: RATES, TIME, AVAILABILITY, and RELIABILITY.
- **State:** The state can be Default or Failed.
- **Failure Parameters:** The theoretical distribution and values related to components failure.
- **Repair Parameters:** The theoretical distribution and values related to components repair.
- **Reliability or Availability:** Metrics values of the block.


Component E1 - Properties

TIME | State: Default

Parameters

Failure Distribution: Exponential

Mean value: 0.0

Repair Distribution: Exponential

Mean value: 0.0

Description:

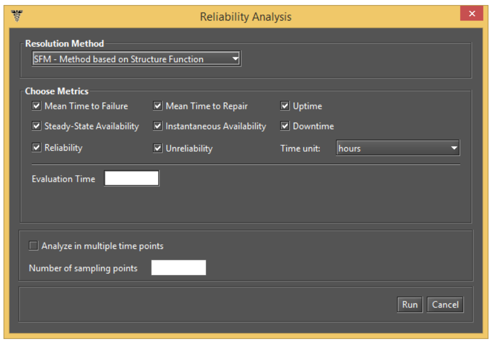
Price ($): 0.00

Figure 124: Leaf node Properties

- **"..." button:** Enable the user select any label declared or CTMC metrics defined in CTMC Models as hierarchical parameters from submodels.
- **Price:** Price of component.

7.1 Exact Evaluation

The Figure 125 presents the input window for Fault Trees Analysis. The tool has two methods to calculate the reliability; the user should choose between the **Standard** mode or the method of **Sum of Disjoint Products**. The metrics that users can choose to be computed are Mean Time To Failure, Mean Time to Repair, Uptime, Steady-State Availability, Instantaneous Availability, Downtime, Reliability, and Unreliability.


Reliability Analysis

Resolution Method: SFM - Method based on Structure Function

Choose Metrics

✓ Mean Time to Failure ✓ Mean Time to Repair ✓ Uptime

✓ Steady-State Availability ✓ Instantaneous Availability ✓ Downtime

✓ Reliability ✓ Unreliability Time unit: hours

Evaluation Time:

☐ Analyze in multiple time points

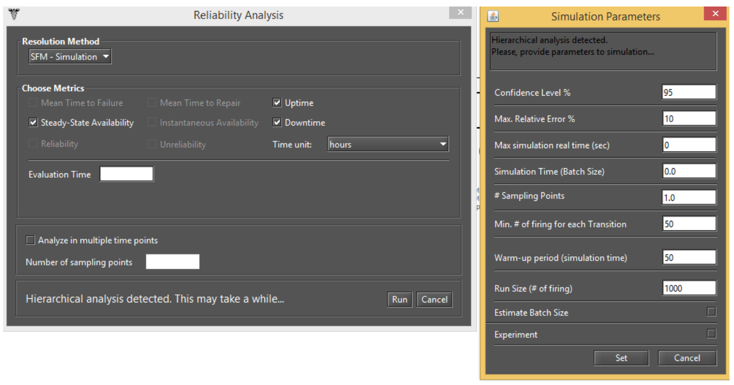
Number of sampling points:

Figure 125: Input screen for FT Analysis

7.2 Resolution by Simulation

The resolution of an FT through simulation occurs when the model components distributions (failure and repair) are not exponential, like Erlang, Gamma, and so on. In this case, the not exponential leaf node is converted to an SPN and resolved by simulation.

When the system detects this type of leaf nodes in the model by selecting any analysis of the FT, the user is prompted to provide input parameters for simulation, according to the chosen metric. The Figure 126 shows the simulation process.


Reliability Analysis

Resolution Method: SFM - Simulation

Choose Metrics
- [ ] Mean Time to Failure
- [ ] Mean Time to Repair
- [x] Uptime
- [x] Steady-State Availability
- [ ] Instantaneous Availability
- [x] Downtime
- [ ] Reliability
- [ ] Unreliability

Time unit: hours

Evaluation Time

- [ ] Analyze in multiple time points

Number of sampling points

Hierarchical analysis detected. This may take a while...

Simulation Parameters

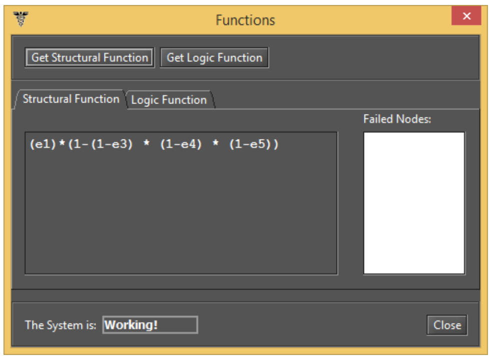
Hierarchical analysis detected.
Please, provide parameters to simulation...

Confidence Level %	95
Max. Relative Error %	10
Max simulation real time (sec)	0
Simulation Time (Batch Size)	0.0
# Sampling Points	1.0
Min. # of firing for each Transition	50
Warm-up period (simulation time)	50
Run Size (# of firing)	1000
Estimate Batch Size	☐
Experiment	☐

Figure 126: Input screens for Transient Simulation and Stationary Simulation

7.3 Structural and Logical Functions

The Mercury Tool allows automatic generation of Structural Function (and Logical Function), which is a behavioral representation (in the perspective of failure and operation) of the system structure in the form of function, for FT Models. The Mercury Tool combines the generation of structural function and the possibility to set the leaf node status manually; it is possible to check the system status relative to the leaf node status. This function is accessible on menu *FT Evaluation* and by clicking over the Get Functions, the user reaches the screen displayed in Figure 127, in addition to the generated expression, the user can set a leaf node as failed (inoperative) and working (operation), which is the results of the current state of the system.


Functions

```
(e1)*(1-(1-e3) * (1-e4) * (1-e5))
```

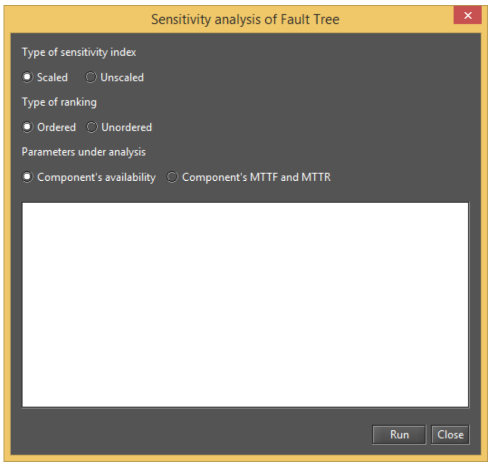
Failed Nodes:

The System is: Working!

Figure 127: Structural Function of a FT Model

7.4 Sensitivity Analysis

The Evaluation menu has a **"Sensitivity Analysis"** option that enables computing partial derivative sensitivity indices for the FT Model. Those indices indicate the impact that every input parameter has on the model's availability. Figure 128 shows the FT Sensitivity Analysis screen.


Sensitivity analysis of Fault Tree

Type of sensitivity index

[✓ Scaled] Unscaled

Type of ranking

[✓ Ordered] Unordered

Parameters under analysis

[✓ Component's availability] Component's MTTF and MTTR

Figure 128: Sensitivity analysis screen for a FT Model

7.5 FT Modeling

To add leaf node see Figure 121, the next step we define the leaf nodes values, click with a right mouse button in the **Properties** menu. See Figure 124 to more.

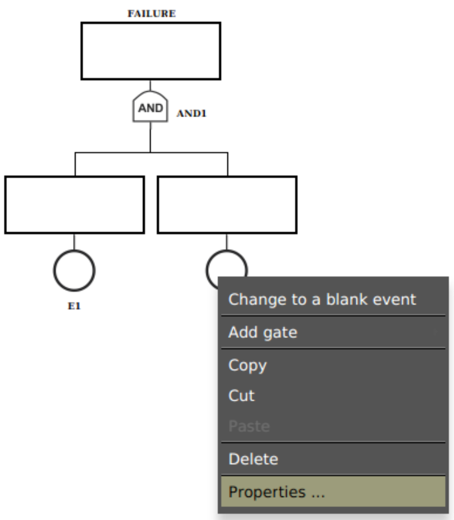


Figure 129: Add FT values

After nodes values insert, the object changes color indicating were values inserted.

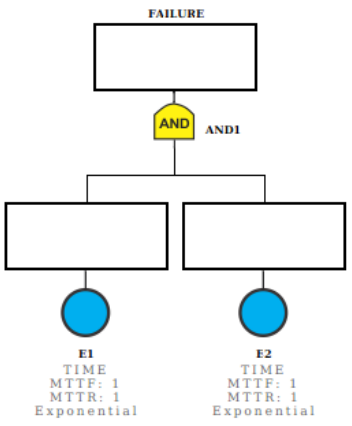


Figure 130: Add FT values

Below shown how to insert others ports in Fault Tree. Just click right mouse bottom the node.

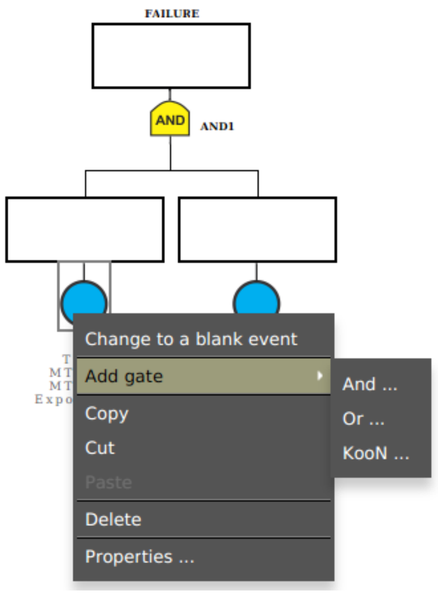


Figure 131: Add FT values

As also logic port change, click a port to change para AND or OR.

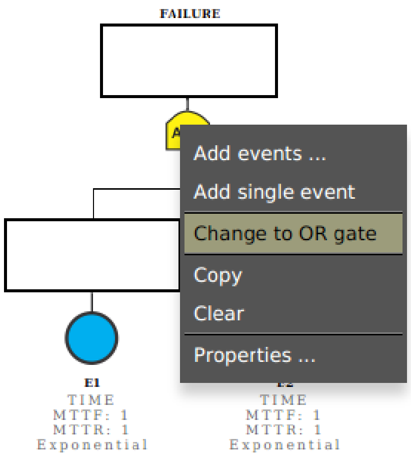


Figure 132: Add FT values

It's possible to convert the Fault Tree (FT) model to Reliability Block Diagram (RBD) model, see Figure 133.

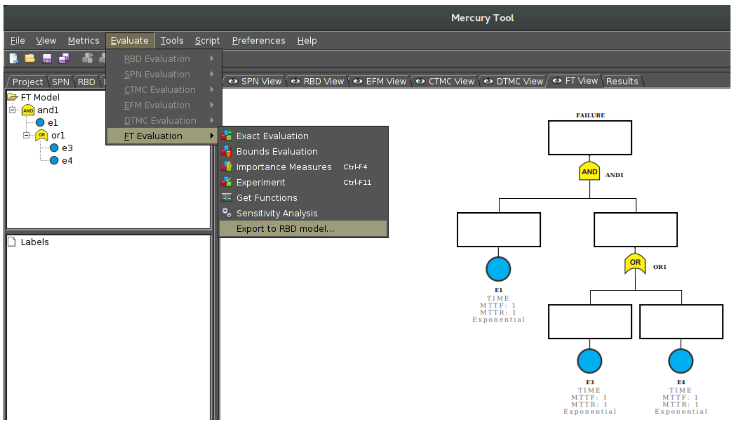


Figure 133: Convert FT to RBD

The Fault Tree converted to the Reliability Block Diagram in Figure 134.

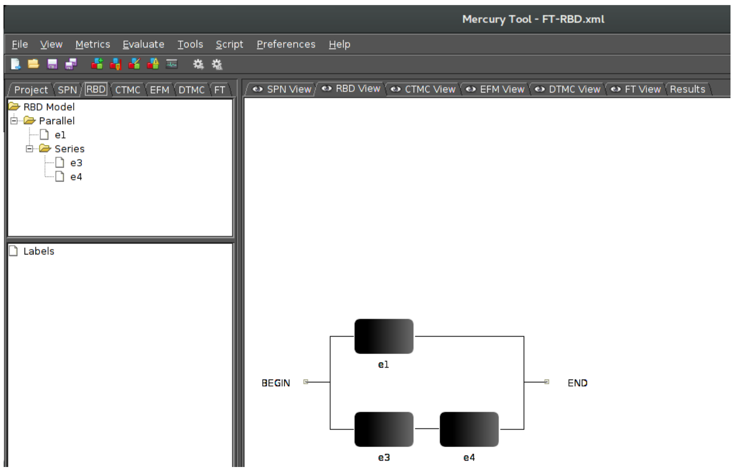

Mercury Tool - FT-RBD.xml
File View Metrics Evaluate Tools Script Preferences Help
RBD Model
Parallel
e1
Series
e3
e4
Labels
BEGIN
END
e1
e3
e4

Figure 134: Model RBD

8 Mercury scripting language

8.1 Introduction

The Mercury scripting language was conceived to allows a greater flexibility on model evaluation. To run Mercury scripts, we can use a command line interface (CLI) tool, or use the *Script* menu inside the graphical interface. The advantage of using this language in conjunction with the CLI tool is the possibility of automatize a project workflow, evaluating models, extracting metrics, and generating reports and charts automatically.

Moreover, there is another advantages offered by the language, that are not supported by the graphical interface

- Increased support to hierarchical modeling: Virtually, any model can call another model and use its results as internal parameter;
- Increased support to symbolic evaluation and experiments. Parameters of a model can be defined as variable left open. We can alter those variables and re-evaluate the model to measure the impact of those parameters on certain metrics;
- Support to Petri Net transitions with a phase-type delay. This family of distributions can be used to approximate any distribution that does not fit into a exponential distribution;
- Support to hierarchical transitions on Stochastic Petri Net models. This type of transitions can be used to reduce the complexity of models, or to express a recurring structure in the model to be reused more easily. Some tools [1] [13] offer support to hierarchical Petri Net models only to Coloured Petri Nets.

8.2 Script structure

We define the syntax of a script using the BNF notation as follows:

Listing 1: Grammar for a Mercury script

```
<script> ::= <models> <main_block>


<models> ::= <model> <models> |
             <model>


<model> ::= <SPN_model> | <CTMC_model> | <RBD_model>
```

A script is formed by a section for declaring models, that contains one or more models of type: CTMC (Continuous Time Markov Chain), RBD (Reliability Block Diagrams) or SPN (Stochastic Petri Net). Support for EMF formalism will be included in the next version. At the end of models section, there is the main block, that has the following syntax:

Listing 2: Grammar for the main block

```
<main_block> ::= <main_block> "{"
<main_statements>
"}"

<main_statements> ::= <statement> ";" <main_statements> |
                      <statement> ";"

<statement> ::= <print_statement> |
                <attribution_statement> |
                <for_statement>
```

In the main block we can modify variables, solve models and print the obtained results. We modify variables to set parameters for a model, and to collect results from metrics using the *solve* function. The for command was included to allow executing experiments. With this command we can modify a variable according to a list of values.

In Figure 135, we show a CTMC model, and on Listing 3, we show the corresponding Mercury script. First, we define a CTMC model named *CTMCModel*, declare its states, transitions and metrics. The rates of transitions are defined as function of *lambda* and *mu* parameters. On the main block, we set values for those parameters, and evaluated the metric m1 of the CTMC model. The result was stored in the variable named *aval*. Finally, we print on console the contents of this variable with the *println* command.

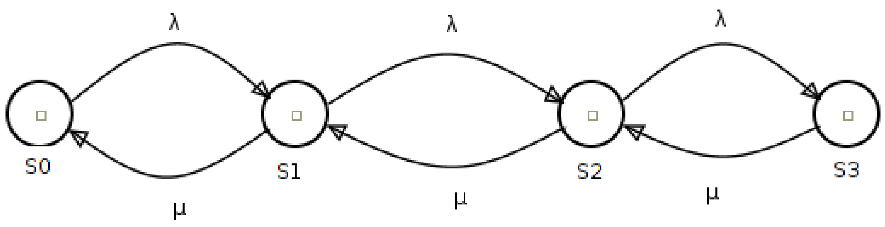


Figure 135: CTMC model example

Listing 3: Source code for CTMC model

```
markov CTMCModel{

  state S0;
  state S1;
  state S2;
  state S3;

  transition S0 -> S1( rate = lambda);
  transition S1 -> S0( rate = mu);
```

```
    transition S1 -> S2( rate = lambda);
    transition S2 -> S1( rate = mu);
    transition S2 -> S3( rate = lambda);
    transition S3 -> S2( rate = mu);

    metric m1 = stationaryProbability( st = S0 );

}

main{
    lambda = 0.00001;
    mu = 0.01;

    aval = solve( model = CTMCModel, metric = m1 );
    println(aval);
}
```

8.2.1 Reserved words

In the Table 2, we list the reserved words of the language.

state	transition	rate	markov	up
RBD	block	hierarchy	series	parallel
top	model	MTTF	MTTR	main
print	println	for	in	out
metric	solve	value	SPN	SubNet
place	timedTransition	immediateTransition	substitutionTransition	tokens
subnet	inputs	outputs	delay	inhibitors
weight	priority	enablingFunction	serverType	

Table 2: Reserved words

This words cannot be used as: identifiers (of models, variables, functions), as parameter metrics, as user defined functions, or as keys of a dictionary structure. For example, on the CTMC metric for stationary probability, we use the key "st" to inform the state that we wish to evaluate the probability. We cannot use the "state" word, because this is a reserved word, used for declaring states in a CTMC model.

On the following sections, we describe the syntax for each supported formalism: Continuous Time Markov Chains, Reliability Block Diagrams, and Stochastic Petri Networks.

8.3 Continuous Time Markov Chain

In the Listing 4, we describe the syntax for declaring CTMC models. A CTMC modelo contains definitions of states, with the *state* reserved word, transitions, with the *transition* reserved word, and metrics. An state can additionaly receive an *up* annotation after the identifier, for availability models. This annotation define states where the system is considered operational (free of failures). All remaing states correspond to the system into a failure state (down).

Listing 4: Grammar for CTMC models

```
<CTMC_block> ::= "CTMC" "{"
<ctmc_statemnts>
"}"


<ctmc_statements> ::= <ctmc_statement> ";" <ctmc_statements> |
                                          <ctmc_statement> ";"


<ctmc_statement> ::= <state_statement> |
                        <transition_statement> |
                        <metric> ;


<state_statement> ::= "state" <identifier> |
                             "state" <identifier> "up"


<transition_statement> ::= "transition" <identifier> "->" <identifier>
                        "(" "rate" "=" <numeric_exp> ")"



<metric> ::= "metric" <identifier> = <metric_name> "(" <metric_parameters> ")" |
                                    <metric_name>
```

The supported metrics for CTMC models are: i) availability; ii) reward rate for states; iii) stationary probability; iv) transient probability.

8.3.1 Availability

The availability metric does not take any parameter. This metric returns the sum of all stationary probability for states annotated with the *up* keyword. In the following script, we show an availability model for a redundant private cloud manager. This example was extracted from [].

Listing 5: Availability metric for a CTMC model

```
markov RedundantGC{
    state fu up;
    state fw;
    state ff;
    state uf up;
    state uw up;

    transition fw -> fu(rate = sa_s2);
    transition fu -> ff(rate = lambda_s2);
    transition ff -> uf(rate = mu_s1);
    transition uf -> uw(rate = mu_s2);
    transition uw -> fw(rate = lambda_s1);

    transition fw -> uw(rate=mu_s1);
    transition uw -> uf(rate=lambdai_s2);
    transition uf -> ff(rate=lambda_s1);

    transition fw -> ff(rate=lambdai_s2);
    transition fu -> uw(rate=mu_s1);

    metric aval = availability;
}
```

8.3.2 Reward metric

This metric computes the sum of rates associated with each state. The parameters defined to this metric are a list of pairs: $<state\_name>=<valor>$. The metric computes the sum of products of each rate and the stationary probability associated with the state. The states that does not receive any rate, are associated with a rate zero, implicitly.

Below, we list an example of model with a reward metric. The reader is suggested to check that metrics m1 and m3 produce the same result.

Listing 6: Reward metric for a CTMC model

```
markov Teste{

    state s1 up;
    state s2 up;
    state s3;

    transition s1 -> s2 (rate = lambda);
    transition s2 -> s3 (rate = lambda);
    transition s3 -> s2 (rate = mu);
    transition s2 -> s1 (rate = mu);

    metric m1 = availability;
    metric m2 = reward ( s1 = 1/5, s2 = 1/4, s3 = 1/3 );
    metric m3 = reward ( s1 = 1, s2 = 1 );
}
```

8.3.3 Stationary and transient probabilities

The most recurrent metrics used with CTMCs are stationary and transient probabilities associated with the states. The stationary probability of a state *S* corresponds to the proportion of time that the model is in this state. The transient probability of a state *S* within a time *T*, corresponds to the probability is in the state *S*, after *T* units of time from initial time (t = 0).

Inside the Mercury language, we use the *stationaryProbability(st = S)* metric to obtain the stationary probability associated with a state *S*. For transient probability, we need also to provide the time *T*, and initial probabilities for each state. That is, the probability that at time T = 0, the model is in each state. In our syntax, the states that are not declared in the list of initial probabilities, receive a initial probability equals to 0. It is important that the sum of all initial probabilities be equals to 1, otherwise an exception will be thrown.

The following listing illustrates the metrics for stationary and transient probabilities:

Listing 7: Stationary and transient metrics for a CTMC model

```
markov Test3{
    state s0;
    state s1;
    state s2;
    state s3;
    state s4;

    transition s0 -> s2 (rate = a);
    transition s2 -> s1 (rate = b);
    transition s1 -> s4 (rate = a);
    transition s2 -> s3 (rate = b);

    transition s3 -> s4 (rate = c);
    transition s4 -> s0 (rate = c);

    metric m1 = reward( s0 = 1, s1 = 2 );
    metric m2 = stationaryProbability ( st = s2 );


    metric t0 = transientProbability (
        time = 100,
        st = s0,
        initialProbabilities = ( s0 = 0.5, s3 = 0.5 )
    );
}
```

8.4 Reliability Block Diagram

An RBD model is composed by:

- Exponential blocks that represents components with an associated mean time to failure and mean time to repair parameters;
- Hierarchical blocks that are evaluated by calling another external models;
- Series/parallel arrangements of another blocks;
- Declaration of the top level block;
- RBD metrics.

In the following listing, we show the RBD model grammar:

Listing 8: RBD model grammar

```
<RBD_model> ::= "RBD" "{" <rbd−statements> "}"


<rbd−statements> ::= <rbd_statement> ";" <rbd_statements>|
                              <rbd_statement> ";"


<rbd_statement> ::= <block_statement>
          | <series_block_statement>
          | <parallel_block_statement>
          | <top_block_statement>
          | <rbd_metrics>


<block_statement> ::= <exp_block_statement> |
                         <hierarchy_block_statement>
```

```
<exp_block_statement> ::= "block" <identifier>
    "(" "MTTF" "=" <numeric_exp> ","
        "MTTR" "=" <numeric_exp> ")"

<hierarchy_block_statement> ::= hierarchy <identifier> "("
                "availability" "=" <numeric_expression> ") ";" |
                 hierarchy <identifier> "("
                "reliability" "=" <numeric_expression> ") ";"



<series_block> ::= "series" <identifier> "(" <identifire_list> ")"  ";"


<parallel_block> ::= "parallel" <identifier> "(" <identifire_list> ")"  ";"


<top_block> ::= "top" <identifier> ";"
```

For illustrating the RBD capabilities of the language, we will show an example of hierarchical model. The RBD of Figure 136 represents an availability/reliability model for a private cloud with an front end system, and a set of nodes. The front end is the administrative component of the private cloud, and the nodes are the machines where the client's virtual machines are deployed. This is a top level model, and all those blocks are evaluated by calling another models. The front end (GC block) is modelled by the *RedundantGC* markov model. For each node, the RBD of the Figure 137 is used (*Node* model). The failure of each node is represented by the series composition of the following blocks: Hardware (Hw), Operating System (OS), Hypervisor (KVM) and Node Controller (NC). The failure of any component provokes the failure of the node. For the failure of top level model, it is necessary that the GC subsystem fails, and all nodes fail. The Mercury script for represent those models is shown on Listing 10.

Listing 9: Hierarchical model for a cloud infrastructure

```
RBD Node{
    block hw(MTTF=mttfhw,MTTR=mttrhw);
    block so(MTTF=mttfso,MTTR=mttrso);
    block kvm(MTTF=mttfkvm,
      MTTR=mttrkvm);
    block nc(MTTF=mttfnc,MTTR=mttrnc);

    series b1(hw, so, kvm, nc);
    top b1;

    metric aval = availability;
}

RBD Cloud{
    hierarchy gc(
      availability = solve(
         model=RedundantGC,
         metric = aval
      )
    );
    hierarchy node1(
      availability = solve(model = Node,
      metric = aval)
    );
    hierarchy node2(
      availability = solve(model = Node,
      metric = aval)
    );
    hierarchy node3(
      availability = solve(model = Node,
      metric = aval)
    );
    hierarchy node4(
      availability = solve(model = Node,
      metric = aval)
    );
    hierarchy node5(
      availability = solve(model = Node,
      metric = aval)
    );

    parallel b1(node1, node2,
       node3, node4, node5);
    series b2(gc, b1);

    top b2;

    metric aval = availability;
}
```

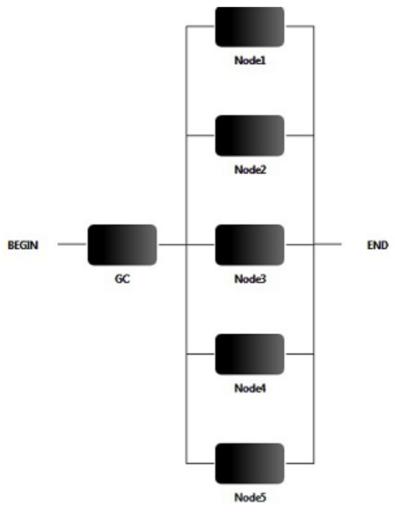


Figure 136: Reliability block diagram for a private cloud [4]

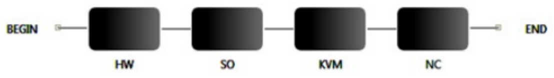


Figure 137: Reliability block for a cloud node [4]

Listing 10: Hierarchical model for a cloud infrastructure

8.4.1 RBD metrics

We have four metrics available to RBD models:

- Availability;
- Mean time to failure;
- Mean time to repair;
- Reliability.

We show the previous example of *Node* model with the four metrics in the following listing. The three first metrics does not take any parameters: steady state availability, MTTF and MTTR. The reliability and instantaneous availability metrics use a *time* parameter.

Listing 11: RBD metrics

```
RBD Node{
    block hw(MTTF = mttfhw,MTTR = mttrhw);
    block so(MTTF = mttfso,MTTR = mttrso);
    block kvm(MTTF = mttfkvm,
      MTTR = mttrkvm);
    block nc(MTTF = mttfnc,MTTR = mttrnc);

    series b1(hw, so, kvm, nc);

    top b1;

    metric m1 = availability;
    metric m2 = mttf;
    metric m3 = mttr;
    metric m4 = availability( time = 50 );
    metric m5 = reliability( time = 100 );
}
```

8.5 Stochastic Petri Nets

In our scripting language, a Petri Net is described in terms of places and transitions. Places can be defined with a (optional) initial marking. Transitions can be of three types: immediate, timed and substitution. Substitution transitions allow us to create modular and reusable Petri nets. This functionality is present only in the scripting language. Another exclusive feature of the language is the support to *phase-type* distributions. In this section we will show an example of simple SPN containing only exponential and immediate transitions, and in the next sub-sections we will describe transitions with phase-type delay, and substitution transitions.

In the listing 12 we show the grammar in BNF notation for describing Stochastic Petri Net models in the language. Basically, we have three distinct statements: place statements, transition statements and metric statements. The arcs connecting transitions and places are defined inside the transitions, as parameters: *inputs*, *outputs* and *inhibitors*. Timed transitions has as parameters: the delay and the server type. If ommited, the server type parameter will be *"SingleServer"* by default. Immediate transitions has as (optional) parameters: weight, priority and an enabling function. The metrics are defined in terms of an string representing a reward metric, the same used in the graphical interface.

Listing 12: Grammar for SPN models

```
<SPN–Model>    ::= "SPN" "{"
<spn_statements>
"}"

<spn_statements> ::= <spn_statement> ";" <spn_statements> |
                     <spn_statement> ";"

<spn_statemnt> ::= <place_statement> |
                   <transition_statement> |
                   <metric_statement>

<place_statement> ::= "place" <identifier> |
                      "place" <identifier> "(" <numeric_exp> ")"

<transition_statement> ::= <timed_transition> |
                           <immediate_transition> |
                          <substitution_transition>

<timed_transition> ::=  "timedTransition" <identifier> "{"
                        [ "inputs" "=" "(" <arc_list> ")" "," ]
                        [ "outputs" "=" "(" <arc_list> ")" "," ]
                        [ "inhibitor" "=" "(" <arc_list> ")" "," ]
                        [ "serverType" "=" { "SingleServer" | "InfiniteServer" } "," ]
                        [ "delay" "=" <delay_exp> "," ]
"}"

<immediate_transiton> ::=  "immediateTransition" <identifier> "{"
                        [ "inputs" "=" "(" <arc_list> ")" "," ]
                        [ "outputs" "=" "(" <arc_list> ")" "," ]
                        [ "inhibitor" "=" "(" <arc_list> ")" "," ]
                        [ "weight" "=" <numeric_exp> "," ]
                        [ "priority" "=" <numei_exp> "," ]
"}"
```

To illustrate the syntax for modelling SPNs in the script language, we a model for a M/M/1/K queue, extracted from [14]. The model is depicted in the Figure 138. The transition *generate* creates the tokens that corresponds to service requests. Each generated token is placed in *generated* and, thenceforth, a choice is made. The token can be queued to be processed by the server, if there is a free position in the queue. If the queue is full, the token will be discarded by the immediate transition *loss*. The place *free* controls the firing of this transition, using the inhibitor arc. The tokens waiting in the queue are placed in *buffer*. The transition *service* represents the processing of the queued requests by the server. Considering that is a M/M/1/K queue, we have only one server for attending all requisitions. Therefore, the server semantic for the *service* transition is single server. In the Listing 13 we show the script for the SPN model described earlier.

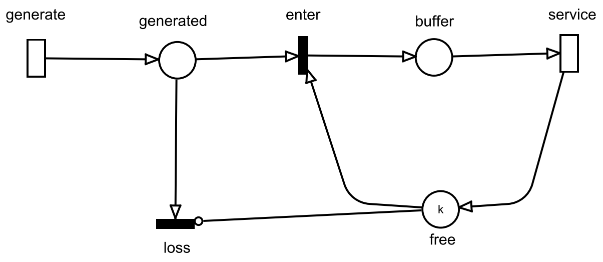


Figure 138: Stochastic Petri Net model for a M/M/1/k queue [14]

Listing 13: Script for SPN model

```
SPN Petri{
    place generated;
    place buffer;
    place free(tokens= k );

    timedTransition generate(
        outputs = [generated],
        delay = lambda
    );

    timedTransition service(
        inputs = [buffer],
        outputs = [free],
        delay = mu
    );

    immediateTransition loss(
        inputs = [generated],
        inhibitors = [free]
    );

    immediateTransition enter(
        inputs = [generated, free],
        outputs = [buffer]
    );

    metric m1 =
      stationaryProbability(
          expression = "P{#buffer>0}"
      );
}

main {
    k = 10;
    mu = 2;
    lambda = 1;
    througput =  solve( model = Petri,
        metric  = m1 ) /  mu;

    println(througput);
}
```

References

[1] R. German, C. Kelling, A. Zimmermann, and G. Hommel, “Timenet: a toolkit for evaluating non-markovian stochastic petri nets,” *Performance Evaluation*, vol. 24, no. 1, pp. 69–87, 1995.

[2] A. Desrochers and R. Al-Jaar, *Applications of Petri Nets in Manufacturing Systems: Modeling, Control, and Performance Analysis*. IEEE Press, 1995.

[3] H. Pham, “System reliability concepts,” in *System Software Reliability*. Springer, 2006, pp. 9–75.

[4] J. Dantas, R. Matos, J. Araujo, and P. Maciel, “An availability model for eucalyptus platform: An analysis of warm-standy replication mechanism,” in *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1664–1669.

[5] R. Matos Junior, A. Guimaraes, K. Camboim, P. Maciel, and K. Trivedi, “Sensitivity analysis of availability of redundancy in computer networks,” in *CTRQ 2011, The Fourth International Conference on Communication Theory, Reliability, and Quality of Service*. IARIA, Apr 2011, pp. 115–121. [Online]. Available: http://www.thinkmind.org/index.php?view=article&articleid=ctrq_2011_6_10_10047

[6] R. S. Matos, P. R. M. Maciel, F. Machida, D. S. Kim, and K. S. Trivedi, “Sensitivity analysis of server virtualized system availability,” *IEEE Transactions on Reliability*, vol. 61, no. 4, pp. 994–1006, 2012.

[7] G. Callou, P. Maciel, D. Tutsch, and J. Araujo, "Models for dependability and sustainability analysis of data center cooling architectures," in *Dependable Systems and Networks (DSN), 2012 IEEE International Conference on*, Jun 2012, pp. 1 –6.

[8] L. Ford and D. R. Fulkerson, *Flows in networks.* Princeton Princeton University Press, 1962, vol. 1962.

[9] J. Ferreira, G. Callou, and P. Maciel, "A power load distribution algorithm to optimize data center electrical flow," *Energies*, vol. 6, no. 7, pp. 3422–3443, 2013.

[10] J. Ferreira, G. Callou, J. Dantas, R. Souza, and P. Maciel, "An algorithm to optimize electrical flows," in *Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics.* IEEE Computer Society, 2013, pp. 109–114.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *et al.*, *Introduction to algorithms.* MIT press Cambridge, 2001, vol. 2.

[12] R. Bellman, "On a routing problem," DTIC Document, Tech. Rep., 1956.

[13] A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, and K. Jensen, "Cpn tools for editing, simulating, and analysing coloured petri nets," in *Applications and Theory of Petri Nets 2003.* Springer, 2003, pp. 450–462.

[14] R. German, "A concept for the modular description of stochastic petri nets (extended abstract," in *Proc. 3rd Int. Workshop on Performability Modeling of Computer and Communication Systems*, 1996, pp. 20–24.

A Syntax of CTMC measures, parameters, state names, and state rewards

Output measures for CTMC models created in the GUI must be defined according to the following notation:

"P{"<state_name>"}" = Probability of being in the state called <state_name>

"R{"<state_name>"}" = Reward rate of state called <state_name>

"R{}" = Steady-state reward of the system

The formal syntax for output measures, names of states and parameters, as well as for transition rates is defined as follows:

Listing 14: Syntax of components for CTMC model

```
<output_measure> ::= <output_value>
                   | ‘‘−’’ <output_measure>
                   | ‘‘(’’ <output_measure>‘‘)’’
                   | <output_measure> <num_op> <output_measure>


<output_value> ::= <probability_measure>
```

```
                  | <reward_measure>
                  | <real_constant>
                  | <integer_value>

<probability_measure> ::= ‘‘P{’’<state_name>‘‘}’’

<reward_measure> ::= ‘‘R{’’ {<state_name>} ‘‘}’’

<state_name> ::= {<identifier>}+

<parameter_name> ::= {<identifier>}+

<transition_rate> ::= <expression>

<reward_rate> ::= <expression>

<expression> ::= <real_value>
              | ‘‘−’’ <expression>
              |‘‘(’’ <expression>‘‘)’’
              | <expression> <num_op> <expression>

<num_op> ::= ‘‘+’’ | ‘‘   ’’ | ‘‘*’’ | ‘‘/’’

<real_value> ::= <parameter_name>
                  |<real_constant>
                  |<integer_constant>

<real_constant> ::= {<digit>}+‘‘.’’{<digit>}+

<integer_constant> ::= {<digit>}+

<identifier> ::= {letter | digit}+

<letter> ::= ‘‘A’’\textendash‘‘Z’’ | ‘‘a’’\textendash‘‘z’’

<digit> ::= ‘‘0’’\textendash‘‘9’’
```

The basic symbols have the following meanings:

"symbol": terminal symbol

< symbol > : non-terminal symbol

symbol1 | symbol2 : symbol 1 or symbol2

{symbol}+ one or more of occurrences of symbol

symbol1–symbol2 : range of values between symbol1 and symbol2

B Syntax of SPN metrics, guard expressions and marking dependent arc multiplicities.

This section presents the specification related to SPN metrics, guard expressions, and marking dependent arc multiplicities. We present a formal syntax description by using Backus-Naur Form (BNF).

Three different expressions can be utilized in Mercury tool (See list:spngrammar). SPN expressions are represented as Metrics, GuardExpressions, and MarkingDependentMultiplicites. Metrics can be a probability or an expectation and are used for representing evaluation metrics. GuardExpressions are adopted to reprensent logic expressions to enable/disable transitions firings. MarkingDependentMultiplicites are numeric expressions that are evaluated depending on the SPN marking to specify arc multiplicities.

Listing 15: Syntax of components for SPN model

```
<Metric> ::=         ''P{''<logic_condition>''}''
                   | ''E{''<marking_function>''}''



<MarkingDependentMultiplicity> ::= <if_else_exp>


<GuardExpression> ::= <logic_expression>


<if_exp> ::= {''IF(''<logic_condition>''):(''<expr>'')''}


<if_list> ::= <if_exp> | <if_list> <if_exp>


<if_else_exp> ::= <if_list> +''ELSE(''<expr>'')''
                    | <expr>


<expr> ::= <real_value>
             |''-''<expr>
             |''(''<expr>'')''
```

```
          |``(''<expr>``)''<num_op>``(''<expr>``)''

<real_value> ::= <identifier>
               | <real_const>
               | <int_value>

<real_const> ::= {<digit>}+``.''{<digit>}+

<logic_condition> ::= <comp>
               | ``(''<logic_condition>``)''
               | ``NOT(''<logic_condition>``)''
               | ``(''<logic_condition>``)''AND``(''<logic_condition>``)''
               | ``(''<logic_condition>``)''OR``(''<logic_condition>``)''

<comp> ::= <mark_function><comp_op><mark_function>

<comp_op> ::= ``/='' | ``='' | ``<'' | ``>'' | ``<='' | ``>=''

<mark_function> ::= ``(''<mark_function>``)''<num_op>``(''<mark_function>``)''
               | ``(''<mark_function>``)''
               | <int_value>

<num_op> ::= ``+'' | ``   '' | ``*'' | ``/''

<int_value> ::= <int_const>
               |<identifier>
               |<mark>

<int_const> ::= {<digit>}+

<identifier> ::= {<letter>|<digit>}+

<letter> ::= ``A''\textendash``Z'' | ``a''\textendash``z''

<digit> ::= ``0''\textendash``9''
```

<mark> ::= ‘‘#’’<identifier>

B.1 GENERAL COMMENTS SPN ABOUT SYNTAX

In this syntax, all the elements of a given expression are separated by parentheses. For instance, suppose we want to evaluate the probability of having more than two tokens on place P1 and zero tokens on place P2. The corresponding expression is:

$$P\{(\#P1>2)AND(\#P2=0)\}//\text{CORRECT SYNTAX}$$

IMPORTANT. Empty spaces inside the expressions are not allowed. Therefore, the following expression is not allowed.

$$P\{\#P1>2AND\#P2=0\}//\text{WRONG SYNTAX}$$

Generally guard expressions are composed of different comparisons composed by by ANDs, ORs, and NOTs. For instance, the following expression.

$$(\#P1=1)AND(\#P2=2)//\text{CORRECT SYNTAX}$$

This expression can be adopted as a guard function to allow a transition to fire only if the place P1 has one token and P2 has two tokens. Again, empty spaces inside the expressions are not allowed and all the sub-expressions must be combined by using parentheses.

$$\#P1=1AND\#P2=2//\text{WRONG SYNTAX}$$

Regarding if else expressions. The language supports if else expressions to represent MarkingDependentMultiplicity. This component is used to represent number of tokens in places. When used in language, these expressions may change the place marking based on other place markings. For instance, suppose a net with two places P1 and P2, and the marking of P1 is one if P2 has no tokens and zero if P2 has tokens. In this case, P1 marking should be

$$IF(\#P2=0):(1)ELSE(0)$$

It is also possible to have nested if else expressions. To explain that, let’s extend the previous example and consider that the net has 4 places (P1, ..., P4) and the marking of P1 will be one if P2 has no tokens, zero if P3 has one token, two if P4 has no tokens and three otherwise. The corresponding expression should be.

$$IF(\#P2=0):(1)IF(\#P3=1):(0)IF(\#P4=0):(2)ELSE(3)$$

This expression is similar to nested if, elsif, ..., else expressions in standard programming languages like C or Java.