



Mercury Tool Manual

v5.1

MoDCS Research Group

[<http://www.modcs.org>]

CIn - Centro de Informatica UFPE
Cidade Universitaria - 50740-540 - Recife - Brazil
- Tel +55 81 2126.8430 -



August 9, 2021

Contents

| | | |
|----------|--|-----------|
| 1 | Overview | 1 |
| 1.1 | How to Install the Tool | 2 |
| 1.1.1 | Linux System Requirements | 2 |
| 1.2 | Graphical User Interface (GUI) | 3 |
| 1.2.1 | RBD View | 3 |
| 1.2.2 | FT View | 4 |
| 1.2.3 | EFM View | 5 |
| 1.2.4 | SPN View | 6 |
| 1.2.5 | CTMC View | 7 |
| 1.2.6 | DTMC View | 8 |
| 1.3 | Main Menu | 9 |
| 1.4 | Main Toolbar | 16 |
| 1.5 | Drawing Area | 17 |
| 2 | SPN Modeling and Evaluation | 20 |
| 2.1 | SPN Simulation | 38 |
| 2.1.1 | Stationary Simulation | 38 |
| 2.1.2 | Transient Simulation | 46 |
| 2.1.3 | MTTA Simulation | 54 |
| 2.2 | SPN Analysis | 58 |
| 2.2.1 | Stationary Analysis | 58 |
| 2.2.2 | Transient Analysis | 63 |
| 2.3 | SPN Structural Analysis | 65 |
| 2.4 | Token Game | 66 |
| 2.5 | Sensitivity Analysis | 68 |
| 3 | RBD Modeling and Evaluation | 70 |
| 3.1 | RBD Reduction | 78 |
| 3.2 | RBD Evaluation | 81 |
| 3.2.1 | Exact Evaluation | 82 |
| 3.2.2 | RBD Experiment | 88 |
| 3.2.3 | Bounds for Dependability Analysis | 93 |
| 3.2.4 | Component Importance and Total Cost of Acquisition | 96 |
| 3.2.5 | Structural and Logical Functions | 98 |
| 3.2.6 | Sensitivity Analysis | 100 |

| | | |
|----------|--|------------|
| 4 | FT Modeling and Evaluation | 102 |
| 4.1 | FT Evaluation | 115 |
| 4.1.1 | Exact Evaluation | 116 |
| 4.1.2 | FT Experiment | 122 |
| 4.1.3 | Bounds for Dependability Analysis | 128 |
| 4.1.4 | Component Importance and Total Cost of Acquisition | 133 |
| 4.1.5 | Structural and Logical Functions | 137 |
| 4.1.6 | Sensitivity Analysis | 140 |
| 4.1.7 | Export to RBD model | 141 |
| 5 | CTMC Modeling and Evaluation | 143 |
| 5.1 | Input Parameters/Definitions | 146 |
| 5.2 | Metrics | 147 |
| 5.3 | CTMC Evaluation | 149 |
| 5.3.1 | CTMC Stationary Analysis | 150 |
| 5.3.2 | CTMC Transient Analysis | 156 |
| 5.3.3 | Sensitivity Analysis | 159 |
| 6 | DTMC Modeling and Evaluation | 161 |
| 6.1 | Input Parameters | 164 |
| 6.2 | Metrics | 165 |
| 6.3 | DTMC Evaluation | 167 |
| 6.3.1 | DTMC Stationary Analysis | 167 |
| 6.3.2 | DTMC Transient Analysis | 174 |
| 7 | EFM Modeling and Evaluation | 177 |
| 7.1 | Power Load Distribution Algorithm - PLDA | 183 |
| 7.1.1 | Example of PLDA execution | 185 |
| 7.2 | Power Load Distribution Algorithm in Depth search (PLDA-D) | 186 |
| 7.2.1 | Example of PLDA-D Execution | 187 |
| 8 | Mercury Scripting Language | 189 |
| 8.1 | Introduction | 189 |
| 8.2 | Script Structure | 189 |
| 8.2.1 | Reserved Words | 191 |
| 8.3 | Continuous Time Markov Chain | 192 |
| 8.3.1 | Availability | 193 |
| 8.3.2 | Reward Metric | 193 |

| | |
|---|------------|
| 8.3.3 Stationary and Transient Probabilities | 194 |
| 8.4 Reliability Block Diagram | 195 |
| 8.5 Stochastic Petri Nets | 198 |
| A Syntax of CTMC Measures, Parameters, State Names, and State Rewards | 205 |
| B Syntax of SPN Metrics, Guard Expressions, and Arc Multiplicity Dependent on Marking. | 207 |
| B.1 GENERAL COMMENTS ABOUT SPN SYNTAX | 208 |

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 user interfaces for creating and evaluating stochastic Petri nets (SPNs), continuous-time Markov chains (CTMCs), discrete-time Markov chains (DTMCs), reliability block diagrams (RBDs), fault trees (FTs), and energy flow models (EFMs).

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 the tool. An overview of Mercury features is depicted below:

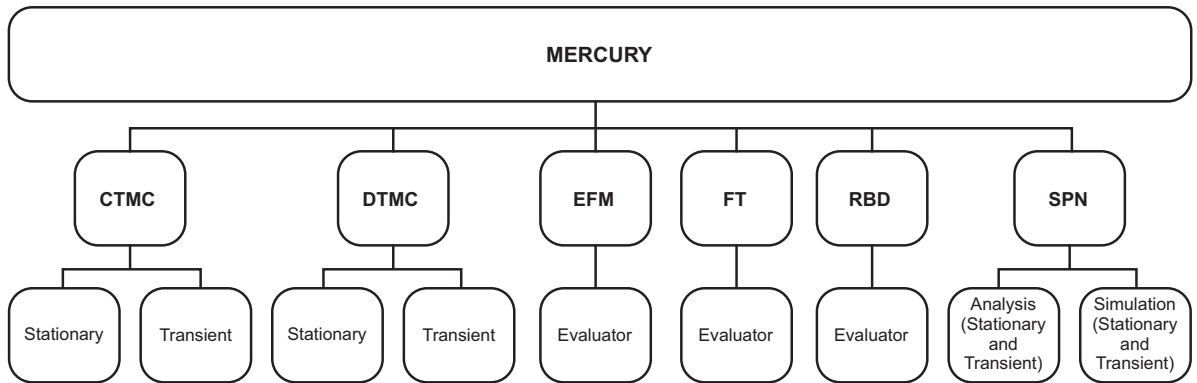


Figure 1: Mercury Tool - Features

Mercury has been developed in Java language, which provides platform independence. Graphical interface allows one to model systems by using one or more views — RBD, FT, EFM, SPN, CTMC, or DTMC — whereas auxiliary modules (e.g., random variate generator and moment matching) are also available. Thus, users can adopt the most appropriate view according to their needs.

In addition, Mercury also provides a feature 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 that standard. All projects developed on Mercury are saved in a “.xml” file that stores all information related to the created models.

1.1 How to Install the Tool

The first step in order to install the newest version of Mercury is to access the URL https://www.modcs.org/?page_id=2392. There is a license agreement document that must be signed and sent to the Mercury creators before access to download page is granted for 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 80 MB of disk space. The memory footprint of Mercury in runtime is about 60 MB, but it might increase depending on the size of models and type of analysis executed by the tool. Extract the archive, and double-click the executable file (**Mercury.jar** or **Mercury.bat**) file to open Mercury. Folder **lib** must be kept in the same path of the jar file. By starting Mercury, the initialization screen depicted in Figure 2 is shown.



Figure 2: Initialization Screen

1.1.1 Linux System Requirements

This subsection specifies the minimum Linux system configuration required to run Mercury. Ensure that the system meets these minimum requirements: 1) Java Runtime Environment (JRE) or Java Development Kit (JDK) in version 1.8+; and 2) OpenJFX package. OpenJFX package is necessary to run the Fault Tree module. On Ubuntu, the admin may use the following command in order to have OpenJFX installed: *sudo apt-get install openjfx*. Fault Tree module will not be available if the last requirement is not met.

1.2 Graphical User Interface (GUI)

Mercury provides six different views: (i) RBD, (ii) FT, (iii) EFM, (iv) SPN, (v) CTMC, and (vi) DTMC. This section briefly describe each one of these views. Each formalism has its own section and more details regarding each view are available in its respective section.

1.2.1 RBD View

Reliability Block Diagram (RBD) is a success-oriented modeling approach and it makes it possible to create a visual representation of a system showing how components contribute to the failure or success of a system. RBD view (see Figure 3) provides features for performing reliability and availability analysis on large and complex systems by using blocks. The types of blocks configurations supported by the tool are series, parallel, and k-out-of-n. It also provides the solution by closed-form equations so that results are usually obtained faster than simulation or numerical solutions of other models. In addition, users can add labels and carry out experiments for a given component individually. By creating a project, the default RBD model is created with an empty block. In the RBD view, the default RBD model has an empty block named b1. The color of this block is gray and it represents that its properties have not yet been defined. See Section 3 for further information about the support provided by Mercury for RBDs.

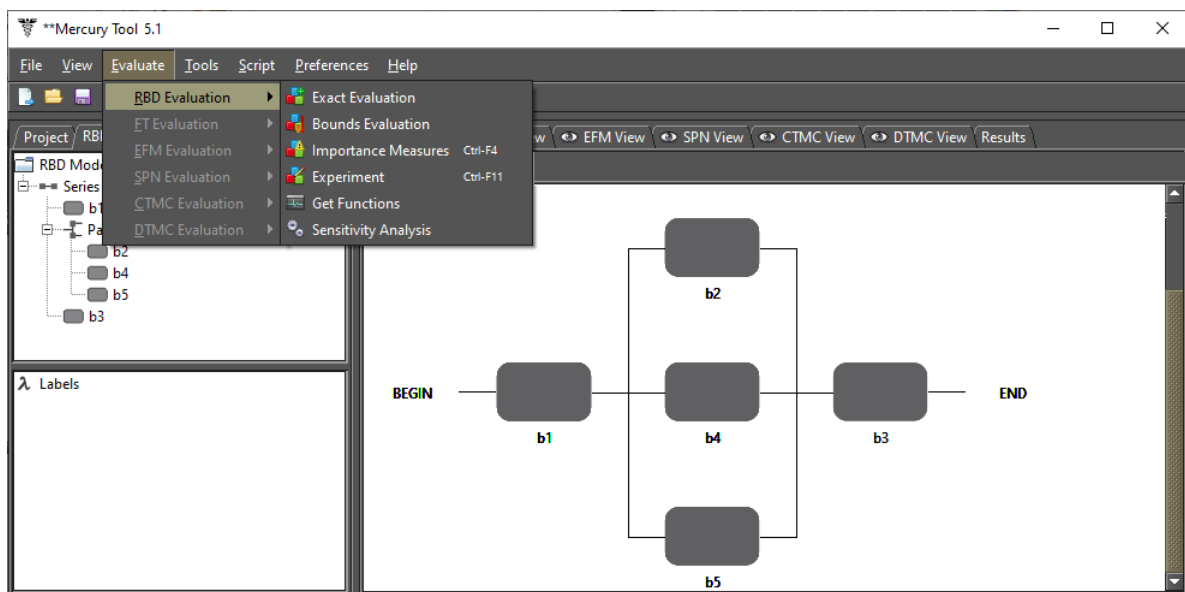


Figure 3: RBD View

1.2.2 FT View

Fault Trees (FTs) and RBDs differ from each other in their purpose. FT is a top-down logical diagram and it makes possible to create a visual representation of a system showing the logical relationships between associated events and causes lead that lead a system to fail. By creating a project, a default FT model is created with an empty top event (see Figure 4). In the FT view, the default model presents a **FAILURE** top event. This event is named "undefined" since no event leads to it. See Section 4 for further information about the support provided by Mercury for FTs.

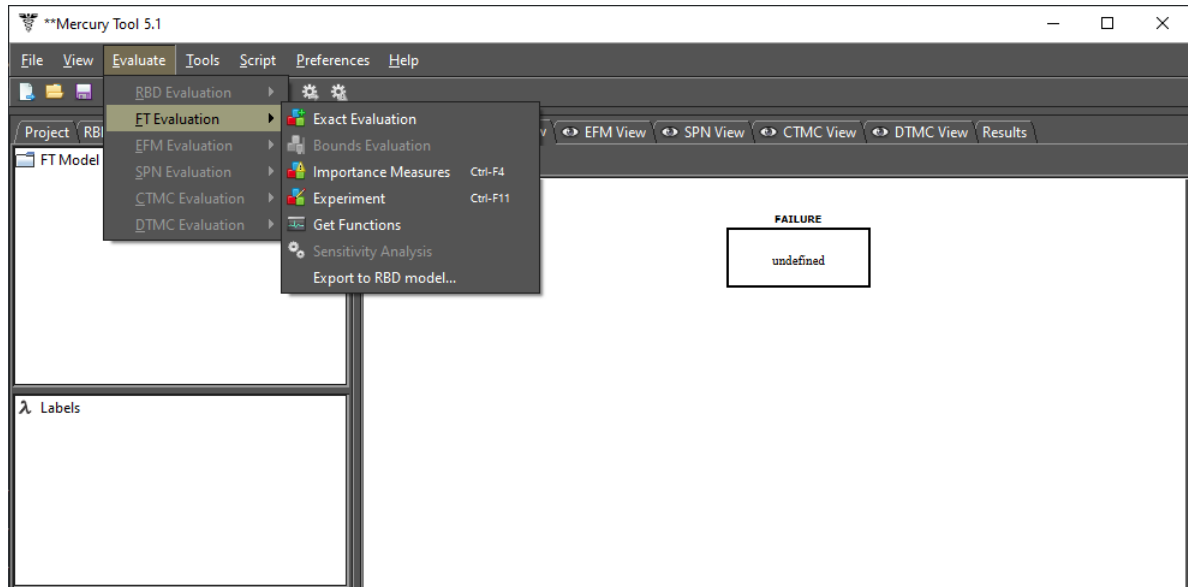


Figure 4: FT View

As we aforementioned, it is necessary to install the JavaFX package in order to make the Fault Tree module available on Linux-based distributions. It can be downloaded by accessing the following URL <https://www.oracle.com/technetwork/pt/java/javafx/downloads/index.html> or installed by using a Linux terminal. For Microsoft Windows systems, there is no need to install any additional packages.

1.2.3 EFM View

Energy Flow Model (EFM) view provides features for computing sustainability and cost estimates of data center power and cooling infrastructures while conforming to the energy constraints of each device. EFM models represent the energy flow between system components in terms of the respective efficiency and maximum energy that each component can provide (for electrical devices) or maximum cooling capacity (for cooling devices). Figure 5 depicts an example of an EFM model. See Section 7 for further information about the support provided by Mercury for EFM.

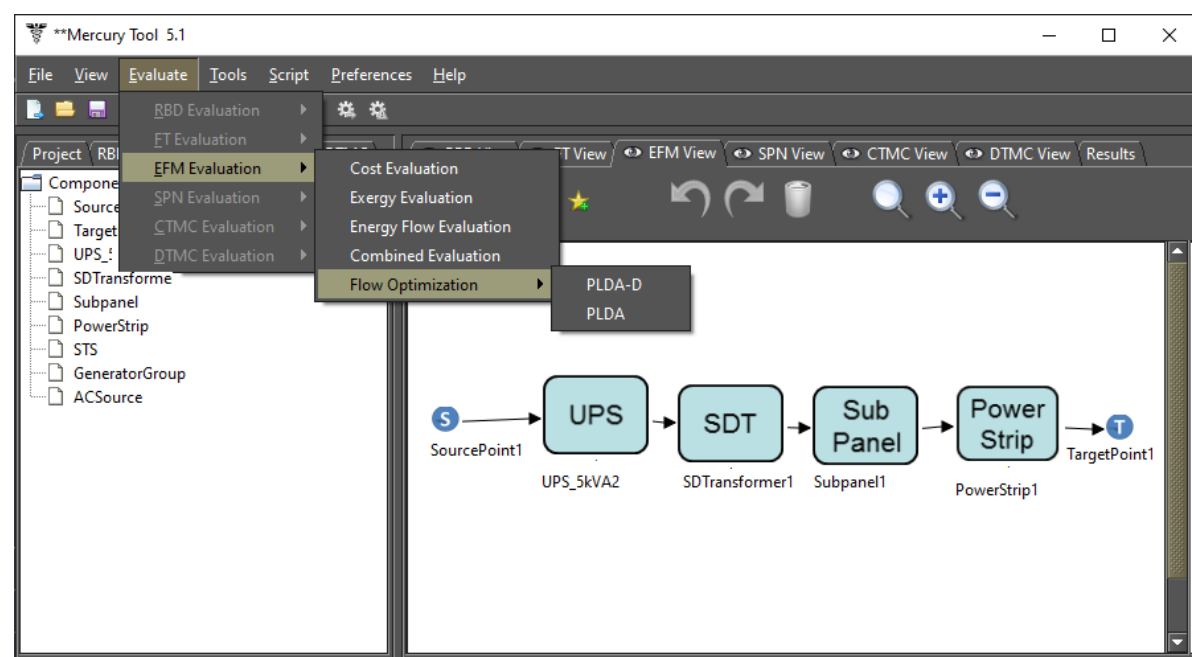


Figure 5: EFM View

1.2.4 SPN View

Regarding stochastic Petri nets, Mercury makes it possible to carry out evaluations by performing simulation or numerical analysis (i.e., numerical solution of underlying Markov chains). Both kinds of evaluations allow to compute **transient** and **stationary** metrics. Time-dependent metrics are obtained through transient evaluations while steady-state metrics are obtained by performing stationary evaluations. Figure 6 depicts the SPN View containing an SPN model as an example. See Section 2 for further information about the support provided by Mercury for SPNs.

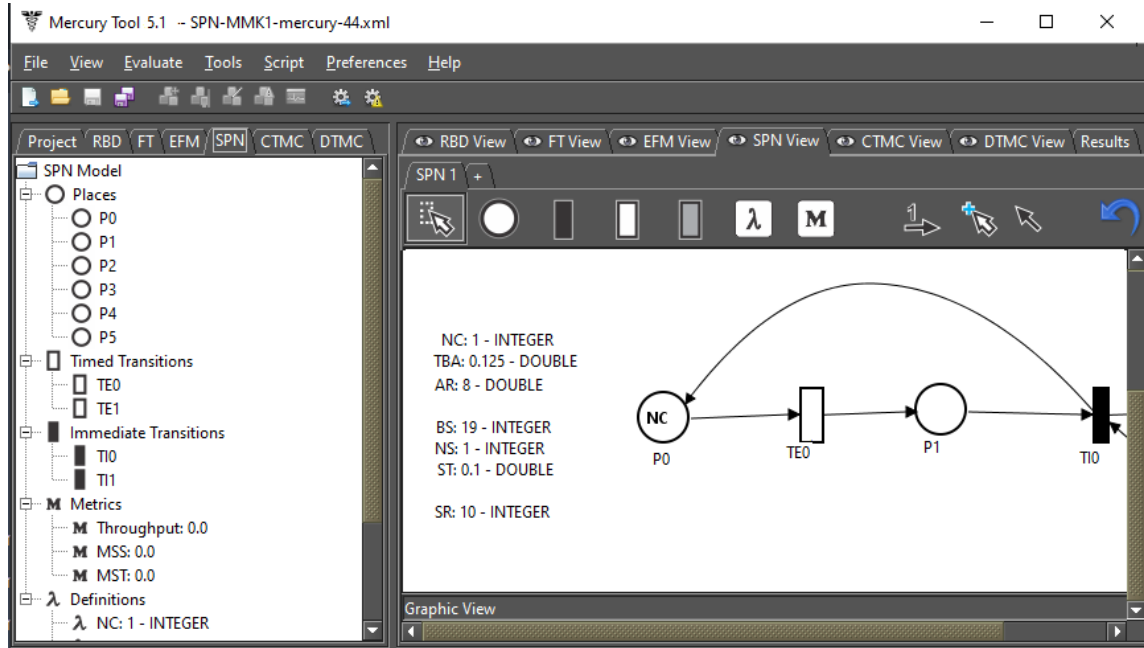


Figure 6: SPN View

1.2.5 CTMC View

CTMC view provides features for drawing and evaluating continuous-time Markov chains (see Figure 7). Numerical solutions of CTMCs may be carried out by performing stationary or transient analysis. For computing stationary metrics, two methods are available: GTH (Grassmann-Taksar-Heyman) and Gauss-Seidel. Transient metrics are obtained by using “Uniformization” method (also known as Jensen’s method) as default, but the user might also try “4th-order Runge Kutta” method. Sensitivity analysis is also available at CTMC view. The rate of each state transition might be defined by means of polynomial expressions referencing user-defined variables (named as parameters/definitions on Mercury). Parameter names can include Greek letters. Besides states and transitions, users can define reward rates assigned to states. In such a case, CTMCs become Markov reward models. For models having absorbing states, Mercury also enables computing absorption probability and mean time to absorption. The user may create custom metrics by formulating expressions that may include state probabilities. Parameters and metrics are easily viewed and modified in the CTMC editor. See Section 5 for further information about the support provided by Mercury for CTMCs.

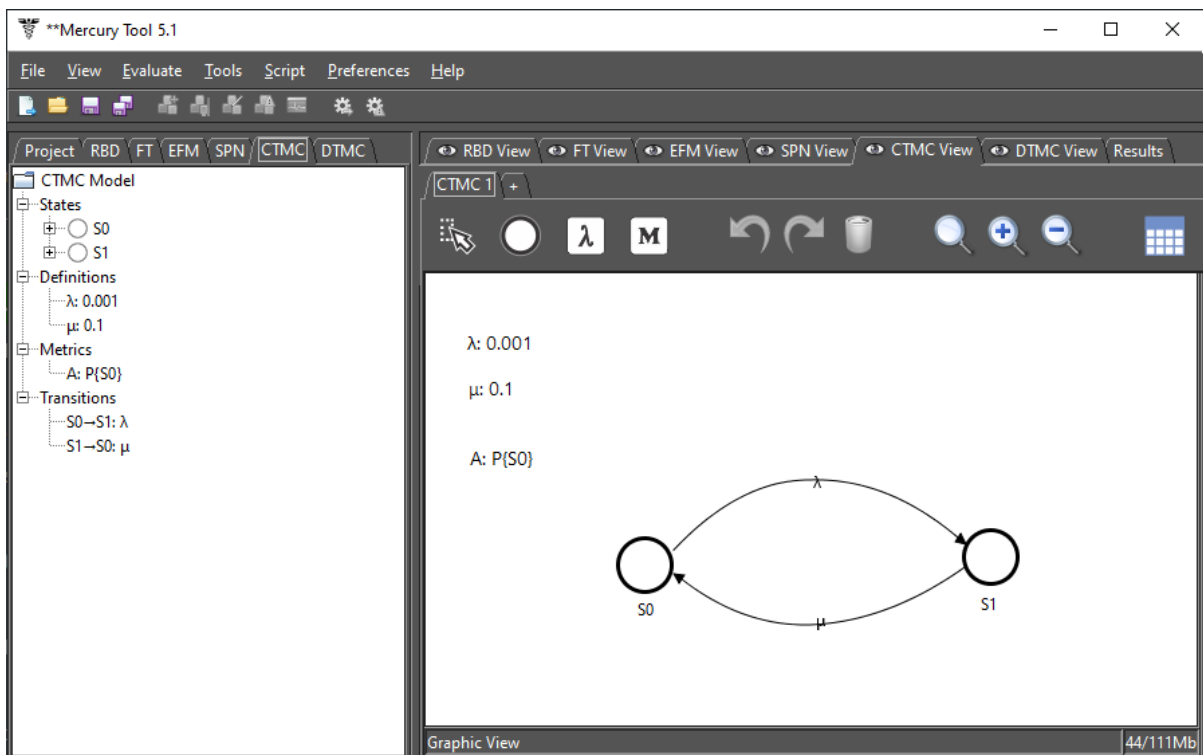


Figure 7: CTMC View

1.2.6 DTMC View

DTMC view provides features for drawing and evaluating discrete-time Markov chains (see Figure 8). Numerical solutions of DTMCs may be carried out by performing stationary or transient analysis. For computing stationary metrics, two methods are available: GTH (Grassmann-Taksar-Heyman) and Gauss-Seidel. Parameter names can include Greek letters. For models having absorbing states, Mercury also enables computing absorption probability and mean time to absorption. The user may create custom metrics by formulating expressions referencing state probabilities. Parameters and metrics are easily viewed and modified on the DTMC editor. See Section 6 for further information about the support provided by Mercury for DTMCs.

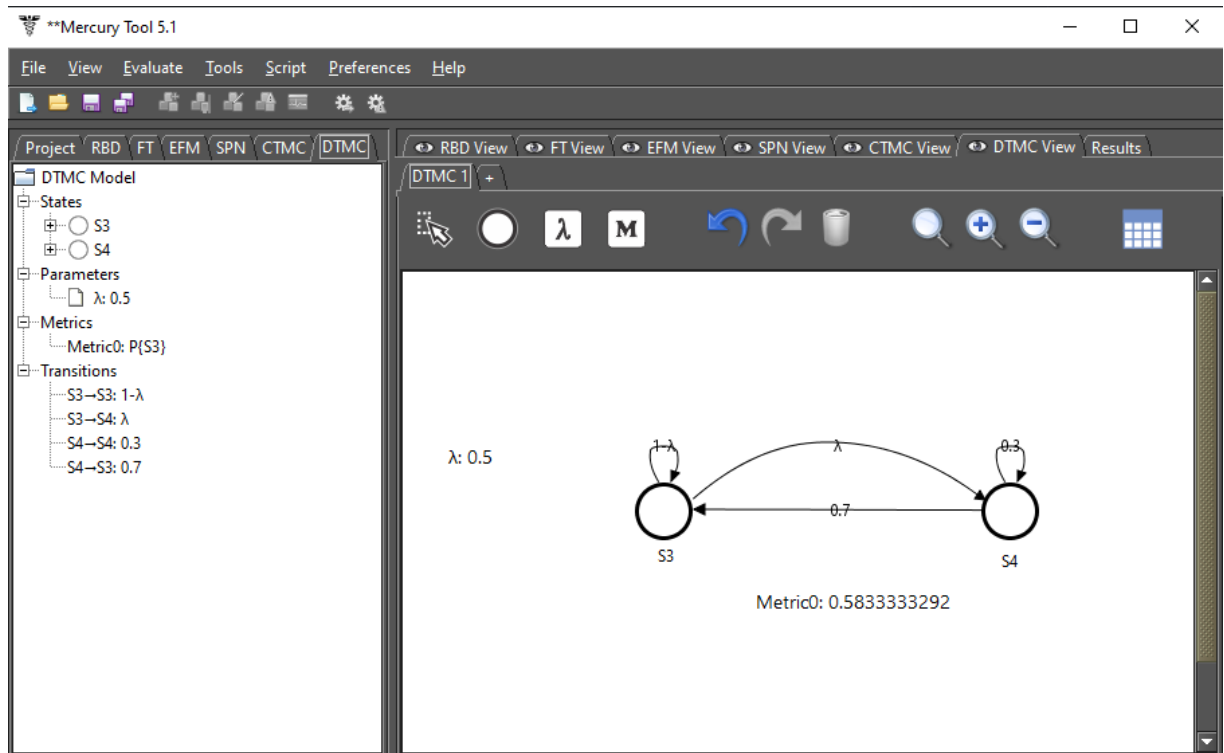


Figure 8: DTMC View

1.3 Main Menu

Main menu of Mercury is shown by Figure 9. As we can see, Mercury has seven main menu items. Some menu items on each main menu item have keyboard shortcuts associated to them. As a demonstration, Figure 10 depicts the options available in menu **File**.

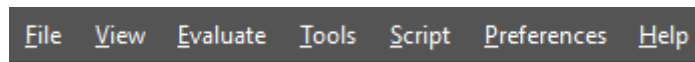


Figure 9: Main Menu

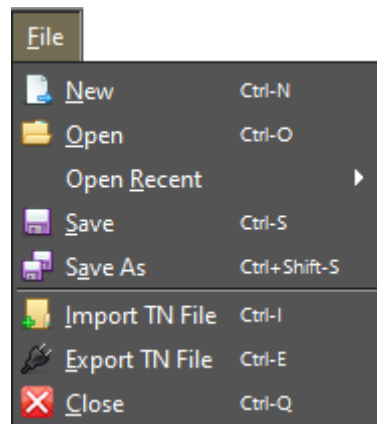


Figure 10: Menu File

Next we describe the options available in menu File.

New. Create a project. By creating a project, all modeling views are made available and initialized empty, except for RBD and FT views that starts each one with a default component with no probability assigned to it.

Shortcut: Ctrl + N

Open. Open a project. Mercury only allows opening files in the Mercury project file format with the “.xml” extension. However, Mercury allows us to import models built in other engines that adopt the “.TN” standard format. Choose “Import TN File” option in order to import files from this format.

Shortcut: Ctrl + O

Open Recent. Here, the user can see a list of the twenty-five most recent projects.

Save. Save recent project changes to the current file or to a new one for a new project. For the first time a project is saved, a window is displayed for the user to choose a location and enter a name for the file to be created.

Shortcut: Ctrl + S

Save As. Save the current project to a new file by defining a new location and name for it.

Shortcut: Ctrl + Shift + S

Import TN File. Import files in the “.TN” standard.

Shortcut: Ctrl + I

Export TN File. Export the project to a file in the “.TN” standard..

Shortcut: Ctrl + E

Close. Close to tool.

Shortcut: Ctrl + Q

Now, let us describe the menu View (see Figure 11). In this menu, the user can show/hide the views provided by the tool: RBD, FT, EFM, SPN, CTMC, and DTMC. Figures 12 and 13 depict the main window with the six views visible and hidden, respectively.

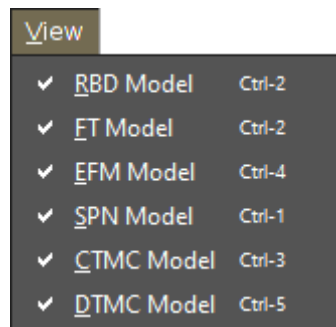


Figure 11: Menu View

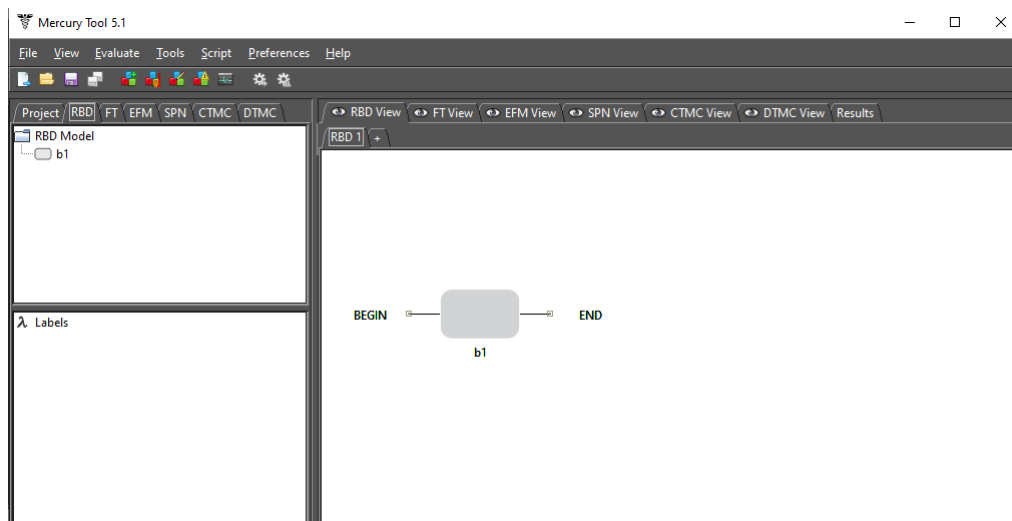


Figure 12: Mercury with the Six Views Visible

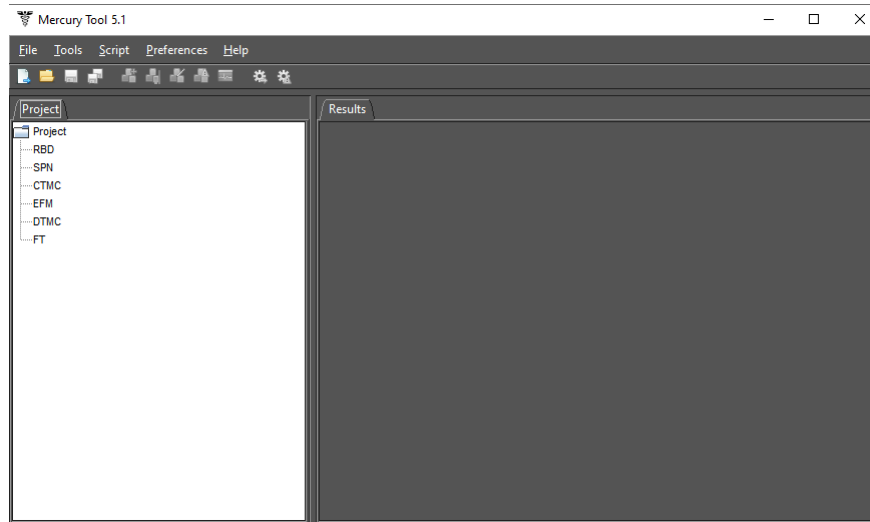


Figure 13: Mercury with the Six Views Hidden

Now, let us describe the menu Evaluate (see Figure 14). This menu has a menu group for each formalism supported by the tool. A menu group is enabled only when the respective model view is active on the main window. Menu items available in each menu group for each formalism are described in the sections ahead.

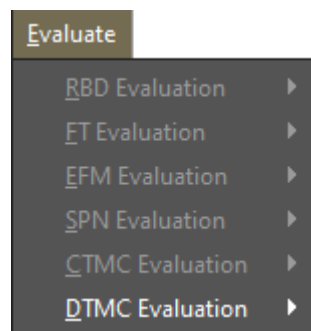


Figure 14: Menu Evaluate

Next we describe the options available in the menu Tools (see Figure 15).

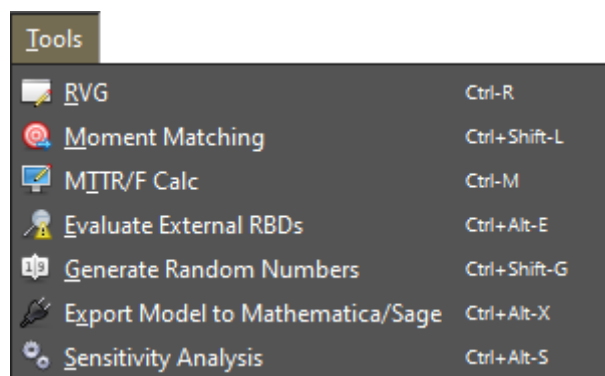


Figure 15: Menu Tools

RVG stands for **Random Variate Generator**. A module for supporting the generation of random numbers, providing a large number of probability distributions. Statistical summaries are also provided taking into

account the generated numbers, such as standard deviation, variance, mean, skewness, and kurtosis (see Figure 16). Results can be exported for supporting evaluations by using other software. RVG is used by the SPN simulator which supports the evaluation of models with non-exponential times.

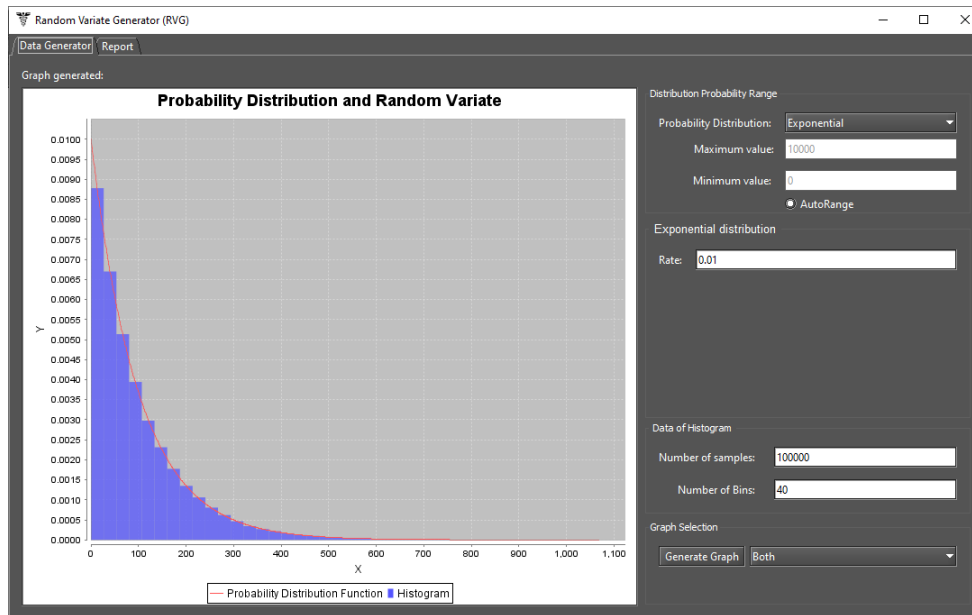


Figure 16: RVG Module

Moment Matching [2]. Supports estimates of which exponential-based probability distribution best fits the mean (first moment) and standard deviation (second moment) of an empirical distribution (see Figure 17). By supporting numerical evaluations of metrics on models with non-exponential times associated with them.

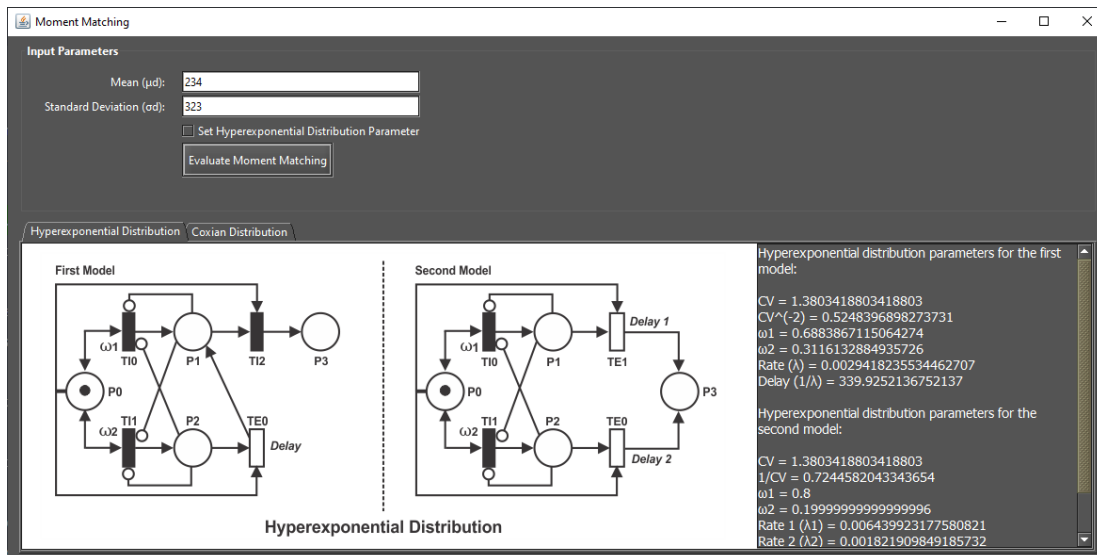


Figure 17: Moment Matching

MTTR/F Calculator. Allows us to compute MTTR (mean time to repair) and MTTF (mean time to failure) by using availability and reliability curve as input parameters. Reliability is a time-dependent metric that

gives the probability to perform something, under stated conditions, for a stated period of time.

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

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

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

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

For instance, by supposing the system failure rate is $0.5[h^{-1}]$, then reliability curve should be the following (see Figure 15):

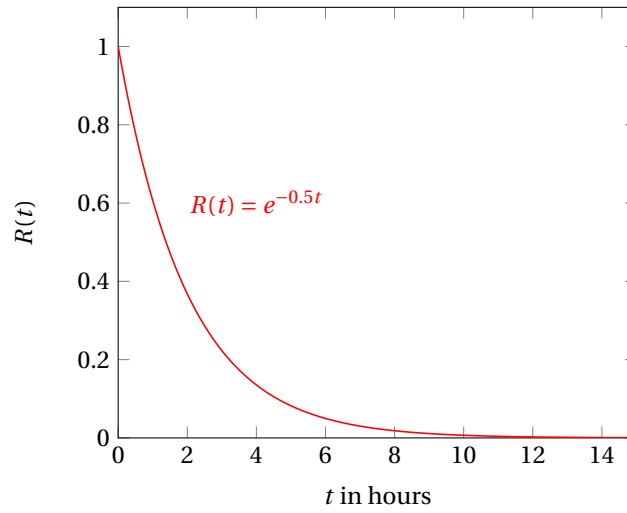


Figure 18: Reliability Over Time

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

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

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

Figure 19 shows the MTTR/F Calculator window, on which users should specify *Availability* and a CSV file with no headers and two columns specifying the evaluation time and reliability value. Button *File* allows us to point to a file containing time and reliability values.

By specifying the parameters, the user should compute results by pressing the button *Calculate*. Figure 20 shows an example of result obtained by computing MTTR and MTTF values. For a more detailed example, please refer to the following video <https://youtu.be/Hmu5DX3CJCg>.

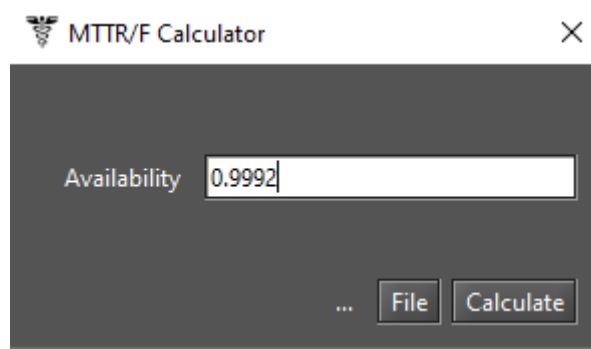


Figure 19: MTTR/F Calculator

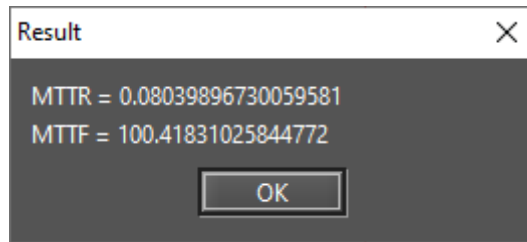


Figure 20: MTTR/F Calculator Result

Evaluate External RBDs. Module that allows us to compute availability metrics of external RBD files created following a specific format.

Generate Random Numbers. This module allows us to generate random numbers following an specific probability distribution. It supports a large number of probability distributions. Firstly, the user needs to enter the size of the sample and select the probability distribution that will guide the generation of numbers. After that, it is necessary to define the value for each parameter of the chosen distribution. Before starting the process of number generation, the user needs to select the location where the file containing the generated sample will be saved.

Export Model to Mathematica/Sage. This feature exports CTMC models to Wolfram Mathematica language/Sage-Math format for supporting external evaluations.

Sensitivity Analysis. Perform sensitivity analysis on the active model in the main window. Two sensitivity analysis methods are available: “Design of Experiments” and “Sensitivity Indices”. The former uses the standard method of analysis of variance to identify the effect of each factor on 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.

Now, let us describe the menu Script (Figure 21). In this menu the user can generate the representation of the active model in the scripting language format or create a script from scratch. By clicking on menu “Generate script”, Mercury converts the active model in a script, thus it can be evaluated and modified in the Script Editor. Figure 22 shows the Script Editor with a script representation of an SPN model. Also, Mercury offers scripts as

examples. To open them, just click on the menu item representing the script and it will be opened in the Script Editor. See Chapter 8 for more information about the Mercury scripting language and its grammar.

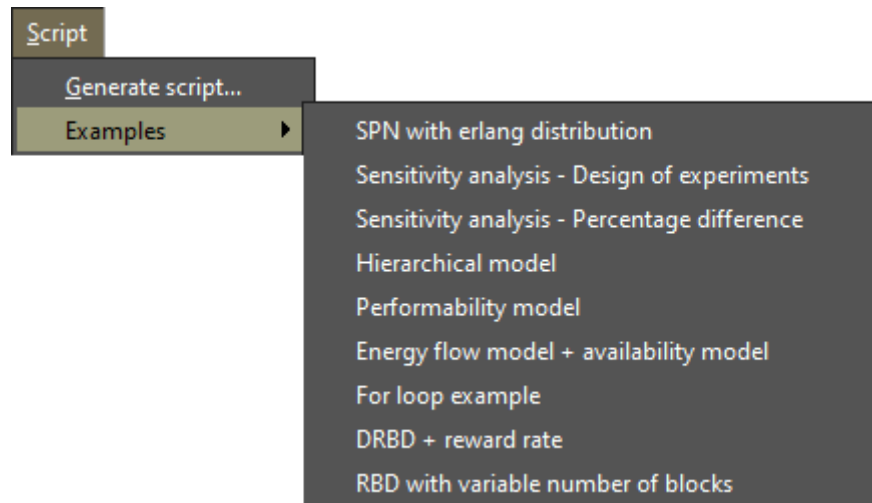


Figure 21: Menu Script

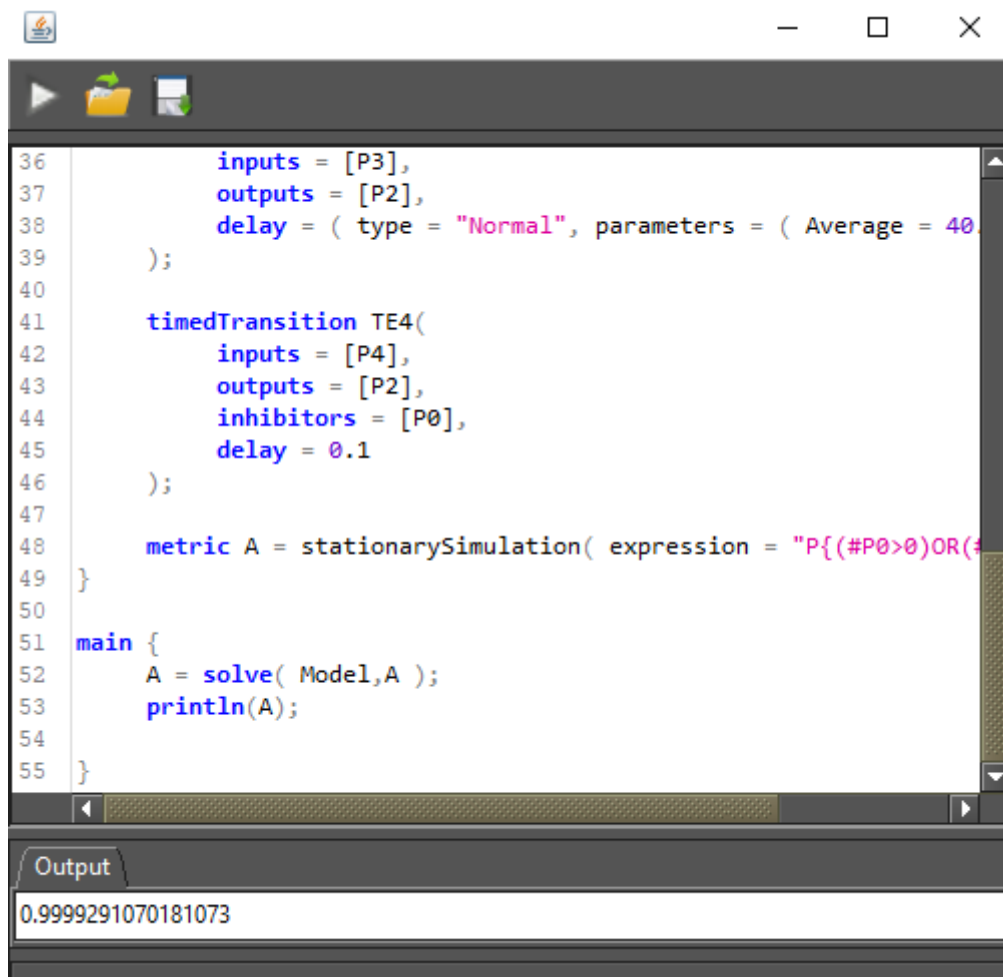


Figure 22: Script Editor

1.4 Main Toolbar

Main toolbar provides access to some of the most used features in Mercury, such as create or save a project. It is displayed at the top of the main window, just below the menu bar. Figure 23 shows the command buttons on this toolbar, which each one is represented by an icon. The following items describe each of them.

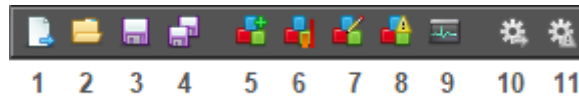


Figure 23: Main Toolbar

1. **New.** Create a project.

Shortcut: Ctrl + N

2. **Open.** Open a project.

Shortcut: Ctrl + O

3. **Save.** Save project changes to the current file or to a new one in case of a new project.

Shortcut: Ctrl + S

4. **Save As.** Save the project to a new file.

Shortcut: Ctrl + Shift + S

5. **RBD Evaluation.** Carry out a myriad of evaluations on the RBD model. It is only enabled when the RBD view is active.

6. **RBD Bounds Evaluation.** Perform bounds evaluations on the RBD model. It is only enabled when the RBD view is active.

7. **RBD Experiment.** Perform experiments on the RBD model. It is only enabled when the RBD view is active.

Shortcut: Ctrl + F11.

8. **RBD Importance Measures.** Calculate importance for each component in the RBD model. It is only enabled when the RBD view is active.

Shortcut: Ctrl + F4

9. **Structural and Logic Functions.** Obtain the structural and logic functions of the current RBD model. It is only enabled when the RBD view is active. Structural function is a function related to the states of RBD blocks. The system and its component may be in working or failed state. System state is a binary random variable determined by considering states of its components. If the states of the components are known, then the state of the system is also known. State of a component may be changed by accessing the properties of the block. When the state of a component is “failed”, the component is marked by a explosion icon on the block. As an example, Figure 24 shows the logical function of a system with no failed blocks.

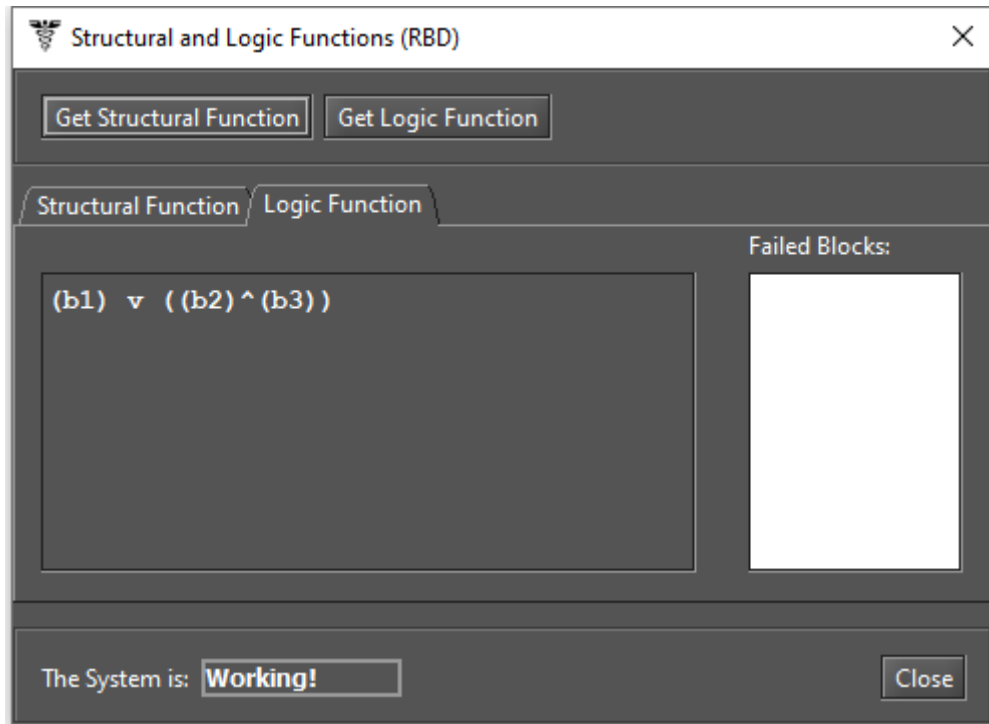


Figure 24: Logic Function of a Model

10. **SPN Stationary Simulation.** Start the SPN stationary simulator. It is only enabled when the SPN View is active.
11. **SPN Transient Simulation.** Start the SPN transient simulator. It is only enabled when the SPN View is active.

1.5 Drawing Area

Mercury supports six formalism and provides a modeling view for each of them — RBD, FT, EFM, SPN, CTMC, and DTMC. In addition, it provides another view to display results generated by the simulators. The user should click on the view tab on the main window to enable the respective visualization. Views can become visible and hidden. To do that, the view must be checked or unchecked on the menu View.

The drawing area is a blank area where components related to a formalism may be inserted. In general, in order to add a component, it is necessary to click on a button representing the component on the toolbar. Right after a click on the desired location inside the drawing area must be performed in order to put the selected component there — except for RBDs and Fault Trees. Figure 25 shows the drawing area of the SPN view.

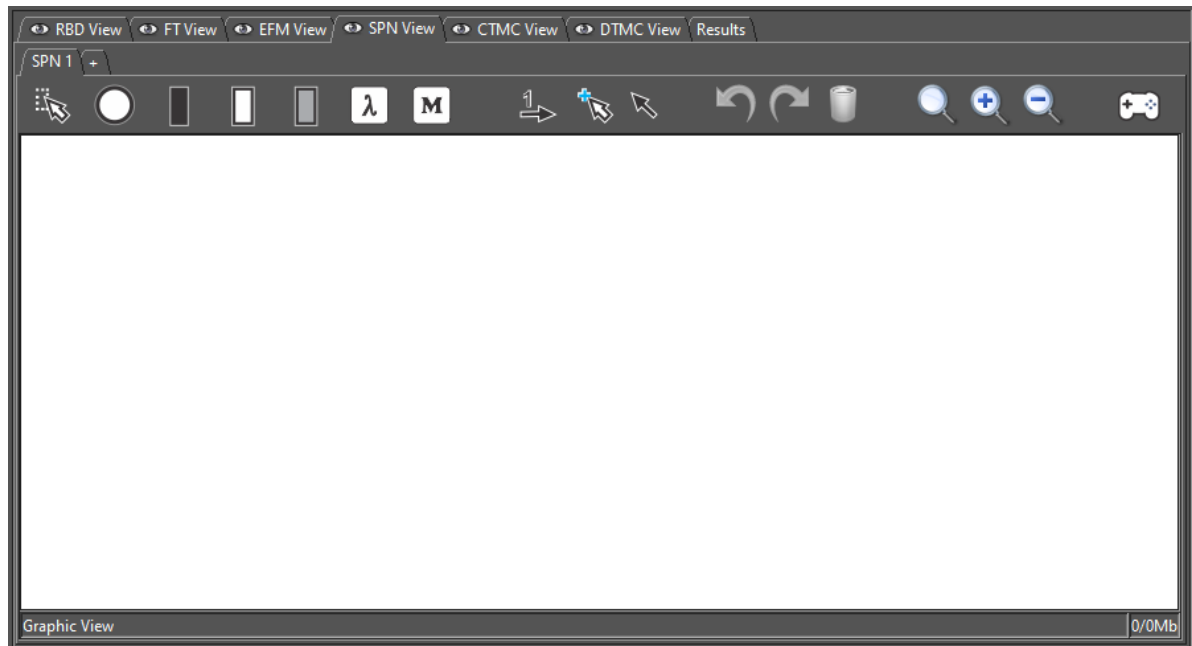


Figure 25: SPN Drawing Area

On the left side of the main window, there is a tab for each formalism, as we can see by looking at Figure 26. By accessing these tabs, all components composing the current model are presented. The properties of a component may be accessed by double-clicking on it. In this example, the SPN tab is active. Each time a component is inserted into the drawing area or any of its properties are updated, the panel on the left side is also updated.

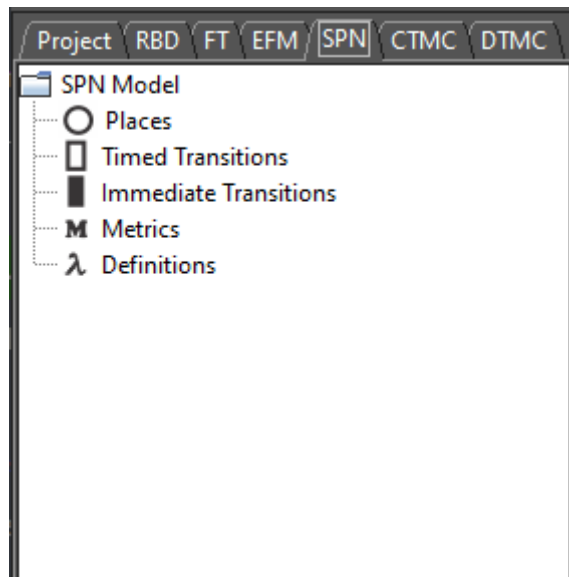


Figure 26: Left-Side Panel

Mercury now supports the creation of projects, supporting the creation of multiple models of the same formalism in a single file. A new model is created by clicking the button “+” under the tab of the selected formalism, as depicted in Figure 27. After that, a new tab is created (see Figure 28). The user also has the option to rename, remove, or duplicate the model, and these options are available from the popup menu that appear when right-clicking on the tab of the model (see Figure 29). If the user selects the option “Rename” or “Duplicate”, the new name of the model may be entered on the dialog (see Figure 30).

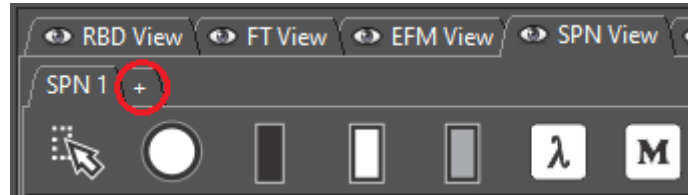


Figure 27: Adding a new tab

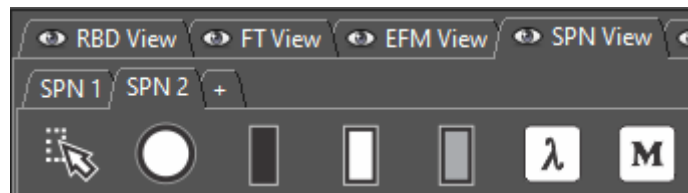


Figure 28: SPN with two models

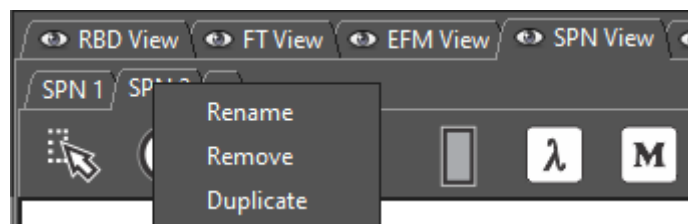


Figure 29: Popup menu for tabs

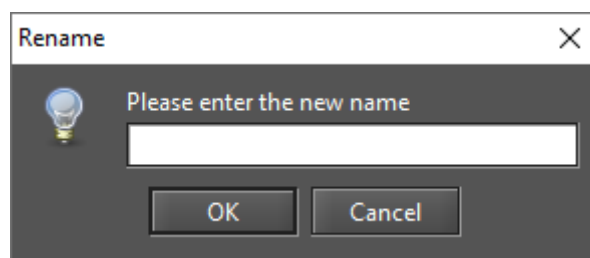


Figure 30: Renaming a model

2 SPN Modeling and Evaluation

Mercury is a complete tool for modeling SPNs. In the SPN view, users can create models by adding components such as places and transitions. Figure 31 shows a model with two transitions — one timed and one immediate —, and two places. Below, we detail the process for modeling SPNs by using Mercury.

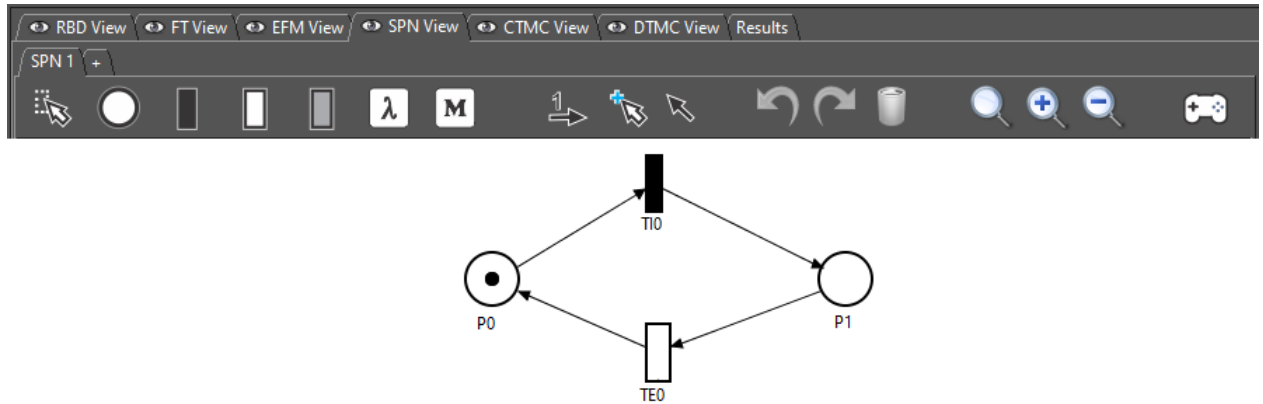


Figure 31: SPN Model

Figure 32 shows the SPN toolbar. Some buttons on the SPN toolbar are the ones used to model SPNs. This toolbar is visible when the SPN view is active. Next, we describe each of its buttons.



Figure 32: SPN Toolbar

1. Selection.



Turns the selection mode on. This mode allows us to select components in the drawing area. When this mode is on, it is possible to select more than one component in the drawing area by holding the SHIFT key down while clicking on the components. Another way to select a set of components is by creating a selection area. A selection area is created by holding the left mouse button pressed while the mouse is moved. All components inside this area will be selected.

2. Place.



Inserts places into the model.

Users should click on the “Place” button and then click on the desired location into the drawing area.

For default, new places do not have tokens.

3. Immediate Transition.



Inserts immediate transitions into the model.

Users should click on the "Immediate Transition" button and then click on the desired location into the drawing area.

4. Exponential Transition.



Inserts exponential transitions into the model.

Users should click on the "Exponential Transition" button and then click on the desired location into the drawing area.

5. Non-Exponential Transition.



Adds non-exponential transitions into the model.

Users should click on the "Non-Exponential Transition" button and then click on the desired location into the drawing area.

6. Definition.



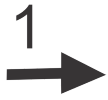
Inserts definitions into the model. Users should click on the "Definition" button and then click on the desired location into the drawing area. A definition is a variable that stores a numeric value. It may be attached to some properties of others SPN components. In this case, Mercury considers the current value of the definition for the property making reference. References are made by entering the name of the definition as the value of the property or inside an expression. Definitions may be attached to markings for places, priorities and guard expressions for transitions, distribution parameters for exponential transitions, weights for immediate transitions, multiplicity expressions for arcs, and expressions for metrics. More than one property may refer to the same definition. Definitions are useful for supporting experiments. In this case, by changing the value of a definition, it is possible to evaluate the impact of that change on a metric.

7. Metric.



Inserts metrics into the model. Users should click on the "Metric" button and then click on the desired location into the drawing area. Basically, a metric is an expression for evaluating a characteristic of the model. A metric may be useful for evaluating whether a specific state is reached or the time spent to perform some activity.

8. Show/Hide Arcs Labels.



Shows/hides the labels on top of the arcs. This type of label shows the multiplicity of the arc. Multiplicity of arcs represents how many tokens will be consumed or created in places. In case of inhibitor arcs, it represents how many tokens a place must have in order to avoid a transition to become enabled.

9. Turn Connection Mode On/Off.



Turns the connection mode on/off.

When the connection mode is on, it is possible to connect places to transitions and vice versa by using arcs.

10. Turn Default/Inhibitor Arc Mode On/Off.



This button allows one to choose the type of arc to be used in order to connect transitions and places. Users can choose between two types of arcs: standard and inhibitor. Inhibitor arcs must only be used to connect places to transitions. By creating a new project, the standard arc is active by default.

11. Undo.



Undo recent changes from the model.

All recent changes are stored and can be rolled back, one after another.

Shortcut: Ctrl+Z

12. Redo.



Redo recent changes undone to the model.

Shortcut: Ctrl+Y

13. Remove.



Removes selected components from the model. By selecting a set of components, all of them will be removed by clicking on this button. Also, users may delete components by pressing DEL key or right-clicking on the selected component and choosing "Remove." By removing a place or transition, its arcs are removed too.

14. Default Scale.



Apply standard scale into the drawing area.

15. Scale Up.



Each click scales the drawing image up by 10% percent (zoom in).

16. Scale Down.



Each click scales the drawing image down by 10% percent (zoom out).

17. Token Game.



Token Game is a feature that allows us to evaluate graphically the behavior of an SPN model. By turning the Token Game on, transitions enabled for the current marking will be highlighted, and the user can double-click on one of them in order to fire it. By firing, a new marking is reached, and, depending on that, new transitions may become enabled and others become disabled. By continuing this firings process, it is possible to check whether the model behaves as expected.

Now, let us see the interaction inside the drawing area. By right-clicking on any SPN component, a pop-up menu is displayed. All components have a popup menu associated to them that provides at least two items:

- **Remove.** Removes the selected component from the model.
- **Properties.** Shows the “Properties” dialog for the user to change the properties of the selected component.

Figure 33 shows the pop-up menu of transitions.

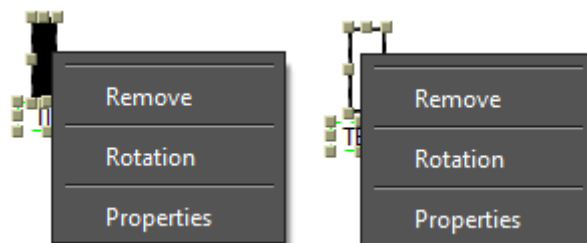


Figure 33: Popup Menu of Transitions

This menu displays three menu items. Rotation is an action only available for transitions.

- **Rotation.** Rotates the selected transition. Transitions can be positioned horizontally or vertically. Figure 34 shows an immediate transition positioned horizontally. All arcs of the transition are readjusted when it is rotated.

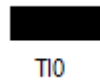


Figure 34: An Immediate Transition Positioned Horizontally

Now, let us describe the properties of timed transitions. To view them, the user should press the right mouse button on that transition, as shown in Figure 33, and then click on the item “Properties”. Another way to do that is by double-clicking on the timed transition. In addition, a third option is by double-clicking on the transition representation in the left-side panel. All roads lead to Rome. It is important to highlight that this last option is available for all components of the model. Figure 35 shows the properties of a timed transition.

Timed Transition [X]

Properties

Name:

Priority:

Guard Expression: ...

Server Type:

Description:

Probability Distribution:

Distribution Parameter(s)

Mean delay:

Figure 35: Properties of Timed Transitions

Next, we describe each one.

- **Name.** Name for the transition. It is used to identify the component in the model. Mercury only accepts

alphanumeric characters and underscore. Also, the name needs to start with an alpha character. An error will occur by not following this rule. Besides that, it is not possible to assign a name already in use by another component of the same type.

- **Priority.** Firing priority assigned to the transition. The higher the priority, the higher the precedence to fire. It is important to highlight that immediate transitions always take precedence over timed transitions.
- **Guard Expression.** A boolean expression to allow a transition to become enabled and can be fired. In addition to the fact that the current marking makes this possible, a transition only becomes enabled and can fire when the guard expression assigned to it is evaluated as true.

The current version of Mercury does not support floating-point literal in the guard expression. Figure 36 depicts an example about how it occurs. In order to overcome this limitation, it is necessary to insert a definition/variable with the floating-point value. By considering our example, we have defined a double definition named **X**. Once it has been created, the definition may be referenced in a guard expression as depicted in Figure 37. We are working to solve this issue and will release a new version when it has been solved.

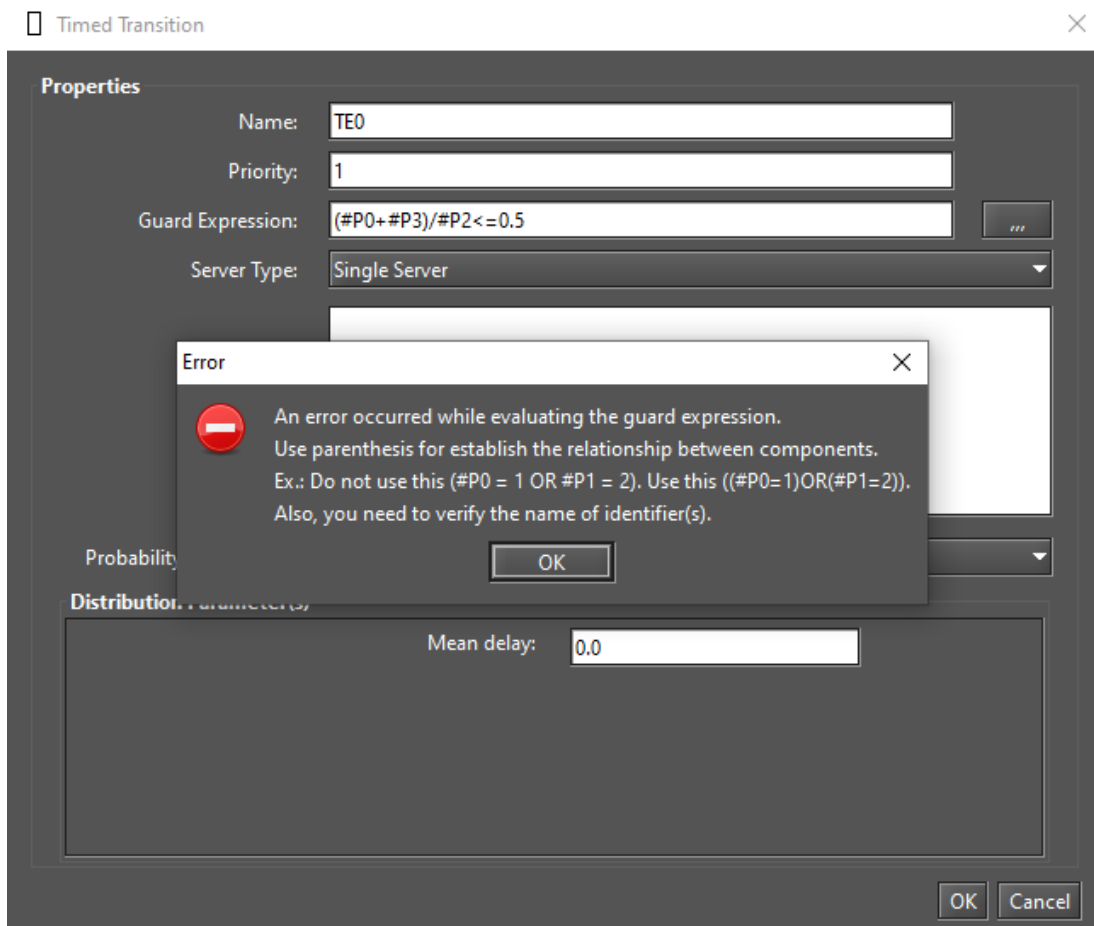


Figure 36: Error when using floating-point literal in Guard Expressions

TE0

X: 0.5 - DOUBLE

Timed Transition
✕

Properties

Name:

Priority:

Guard Expression: ...

Server Type: Single Server

Description:

Probability Distribution: Exponential

Distribution Parameter(s)

Mean delay:

OK
Cancel

Figure 37: Solution to refer floating values in Guard Expressions

- **Server Type.** The firing semantic assigned to the timed transition. Users can choose one of the two options available. These options are single server semantic (SSS) and infinite server semantic (ISS). In SSS, a transition only becomes enabled and can fire only once every instant. In ISS, the number of tokens in the input places of a transition defines the enabling degree for that transition. The enabling degree defines the degree of parallelism of the transition.
- **Description.** A description may be assigned to each component of the model. It represents additional information about the component itself or the real-world component/subsystem/action represented by the component. Description aims to increase the comprehension about the model under construction. It does not have semantic value when evaluating the model. It is only plain text attached to a component.
- **Probability Distribution.** Mercury supports a large number of probability distributions. When all timed transitions are exponential, the model can be evaluated by carrying out numerical analyses or simulations. On the other hand, when there is at least one non-exponential timed transition, the model can only be

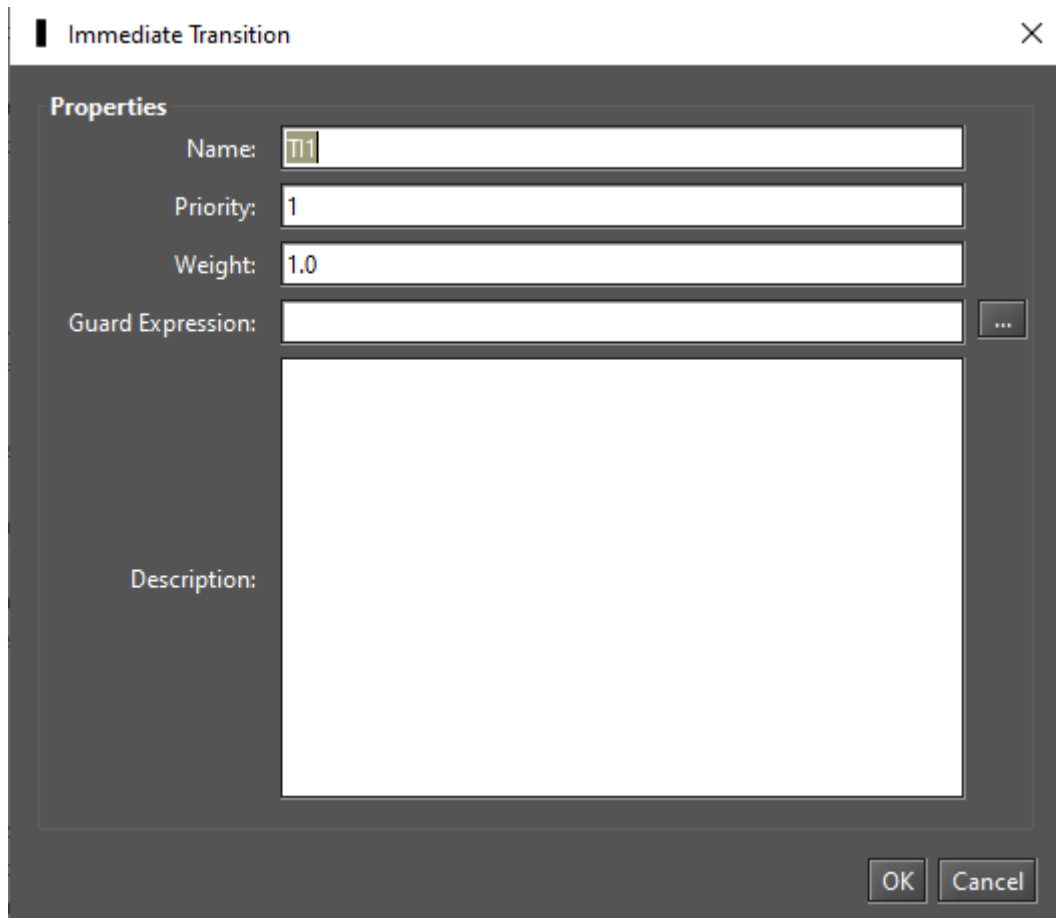
evaluated by simulation. Depending on the chosen distribution, fields related to the parameters of that distribution appear for the user to enter their values. Only non-negative real values may be entered to each parameter. Mercury supports the following probability distributions:

- Beta
- Binomial
- Burr
- Cauchy
- Chi-squared
- Deterministic
- Discrete Uniform
- Erlang
- Exponential
- F Fisher–Snedecor
- Frechet
- Gamma
- Generalized Extreme Value
- Generalized Pareto
- Geometric
- Hypergeometric
- Logistic
- Log-logistic
- Log-normal
- Nakagami
- Normal
- Pareto
- Pearson Type 6
- Poisson
- Rayleigh
- Student's t-distribution
- Triangular
- Uniform

– Weibull

Some probability distributions require only a parameter named “Delay” that corresponds to the delay to fire the transition. In addition to the delay, other parameters may be required depending on the chosen distribution. For example, exponential distribution requires only the mean delay. On the other hand, Erlang distribution requires two parameters: mean delay and shape. Normal distribution requires two parameters: mean and standard deviation. Each probability distribution has its own parameters and they must be entered by the user before performing evaluations.

Now, we describe the properties of immediate transitions (see Figure 38).



The image shows a software dialog box titled "Immediate Transition" with a close button (X) in the top right corner. The dialog is divided into two main sections. The top section, labeled "Properties", contains four input fields: "Name:" with the value "T1", "Priority:" with the value "1", "Weight:" with the value "1.0", and "Guard Expression:" which is empty and has a small menu icon (three dots) to its right. The bottom section, labeled "Description:", contains a large, empty text area. At the bottom right of the dialog are "OK" and "Cancel" buttons.

Figure 38: Properties of Immediate Transitions

Following we describe each one. For the sake of conciseness, we will describe only those properties that we have not yet described.

- **Name.** Name for the immediate transition. It is used to identify the transition in the model.
- **Priority.** See page 25.
- **Weight.** Weight of the transition.
- **Guard Expression.** See page 25.

- **Description.** See page 26.

Now, let us see the properties of places (see Figure 39).

- **Name.** Name for the place. It is used to identify the component in the model.
- **Marking.** The number of tokens assigned to the place. Only non-negative integer values may be entered.
It is possible to attach an integer definition to the marking property once the definition has been created.
To do that, the user only have to enter the name of the definition into this field.
- **Description:** See page 26.

Figure 39: Properties of Places

Figure 40 shows the pop-up menu of arcs.

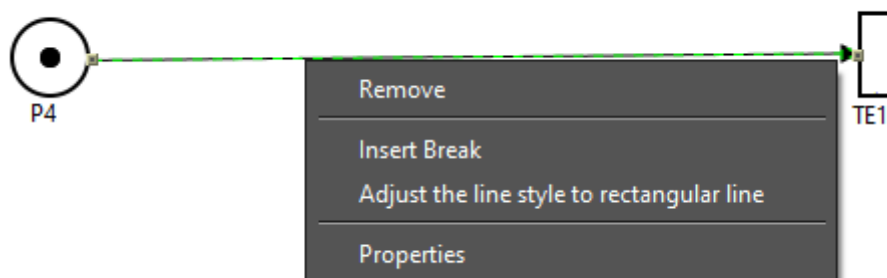


Figure 40: Menu of Arcs

This menu contains four menu items. “Insert break” and “Adjust the line style to...” are items specifics for arcs.

- **Insert Break.** Inserts a break point into the clicked location. By clicking on the break point and holding the mouse button pressed, it is possible to move that point to the desired location. Thus, the shape of the arc can be changed.
- **Adjust the line style to rectangular/curved line.** Mercury supports two line styles for arcs: rectangular and curved. Curved line format is the default style. To change to the rectangular style just click on the corresponding menu item for the chosen arc. Figure 41 shows an SPN with a rectangular arc and Figure 42 shows an SPN with a curved arc.



Figure 41: SPN with a Rectangular Arc

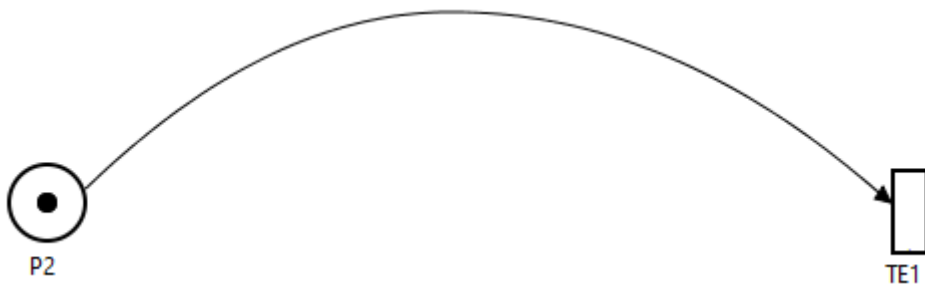


Figure 42: SPN with a Curved Arc

Figure 43 shows the properties of arcs. As we can see, arcs only have one property:

- **Multiplicity:** Multiplicity of the arc, that is, the weight for that arc. It represents the number of tokens required when the arc is an output arc of a place or it represents the number of tokens created in a place when the arc is an input arc to that place.

In some dialog boxes, we can see a button with an ellipsis as a description. This button is always next to a text field as highlighted in Figure 44.

By clicking on this button opens the Expression Editor (see Figure 45).

Expression editor is a text editor that makes it easy to create expressions to define guard expressions and metrics. It is a simple editor that highlights parentheses, brackets, and braces as well as some keywords. The editor has a button that reduces the font of the text and another one that increases it. Thus, making easy to define large and complex expressions.

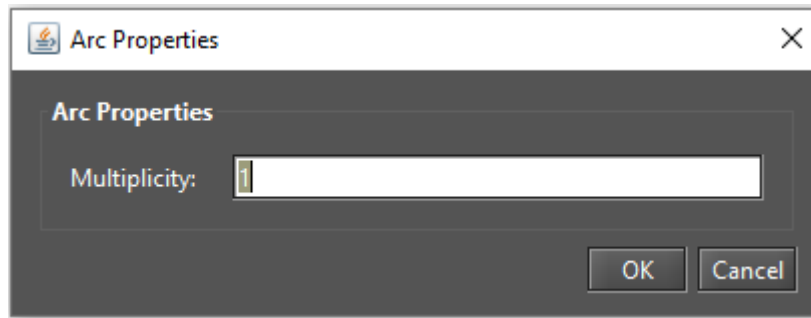


Figure 43: Properties of Arcs



Figure 44: Accessing the Expression Editor

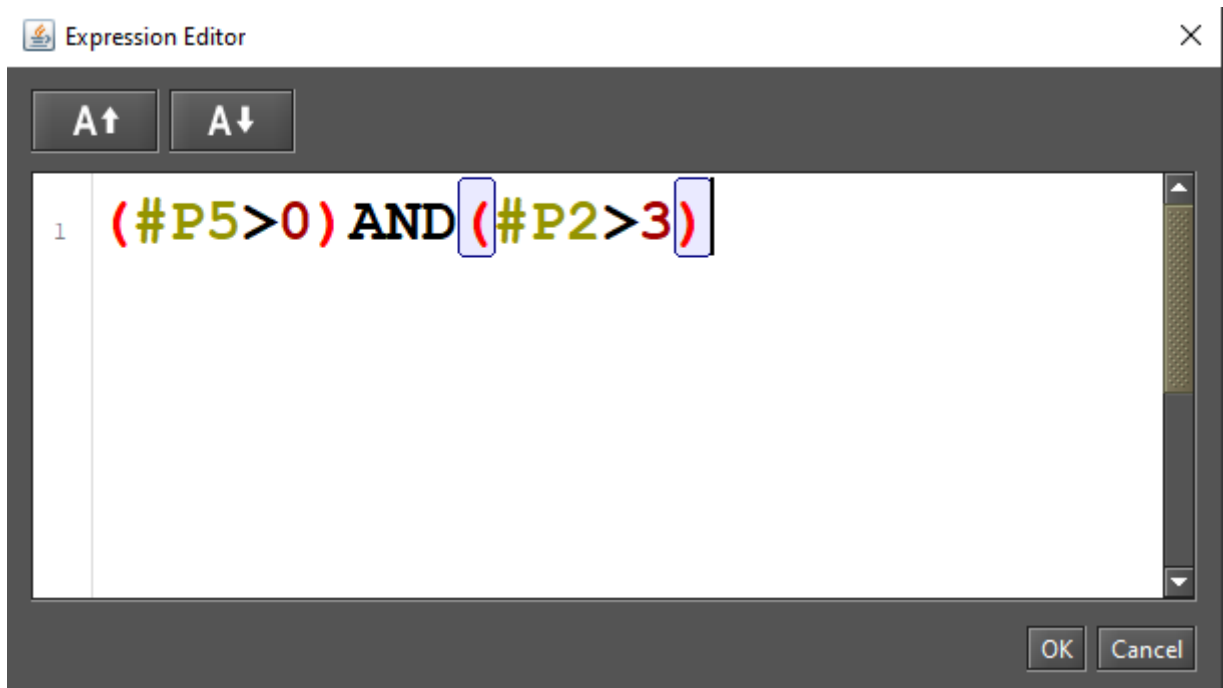
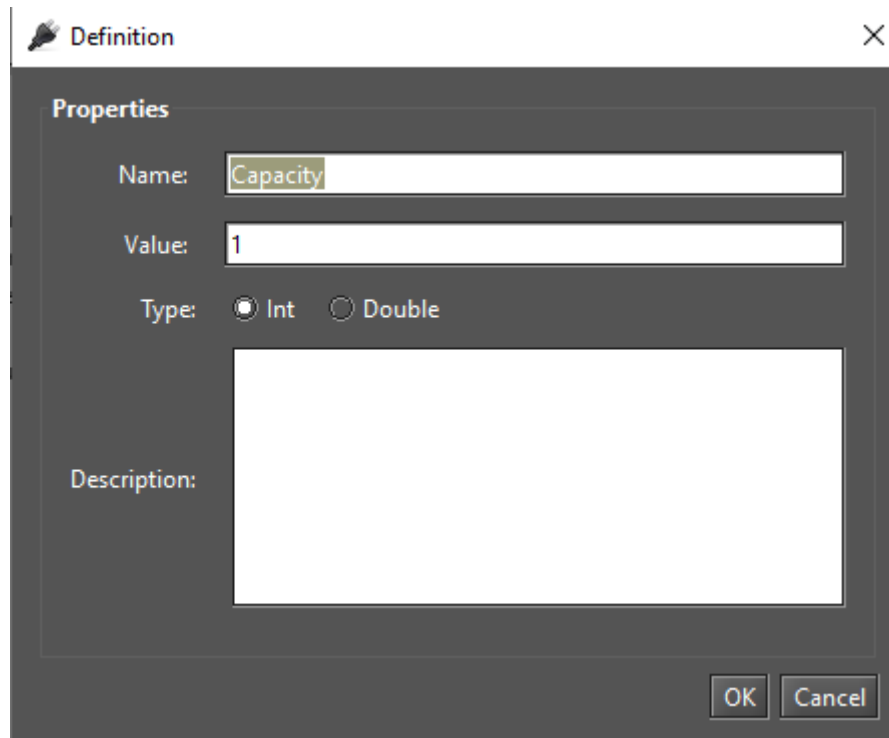


Figure 45: Expression Editor

Now, let us take a look at the properties of the component “Definition.” We have already described this component at the page 21. When accessing the properties of a definition, the dialog presented in Figure 46 is shown. Following, we describe each property of this component.



The image shows a software dialog box titled "Definition". It has a close button (X) in the top right corner. The main content area is labeled "Properties" and contains the following elements:

- Name:** A text input field containing the word "Capacity".
- Value:** A text input field containing the number "1".
- Type:** Two radio buttons labeled "Int" and "Double". The "Int" radio button is selected.
- Description:** A large, empty rectangular text area.

At the bottom right of the dialog, there are two buttons: "OK" and "Cancel".

Figure 46: Properties of Definitions

- **Name.** Name for the definition. It is used to identify the definition in the model.
- **Value.** Value represented by the definition.
- **Type.** A variable can store two types of numeric values: integer and double. By defining the properties of a definition, the user must choose the appropriate type accordingly to the value entered. An error will occur when choosing the type INT and entering a double value.
- **Description.** See page 26.

An important point to have in mind is that definitions are variables that store numeric values, as we already mentioned in this section. A definition cannot make a reference to another definition, but it may be referenced by other components of other types. By updating the properties of a definition, in case it is referenced by other components, a confirmation dialog appears. When the definition is not referenced, if the values entered in the fields are valid, the respective properties are updated accordingly. Figure 47 shows what happens when the user try to update a referenced definition.

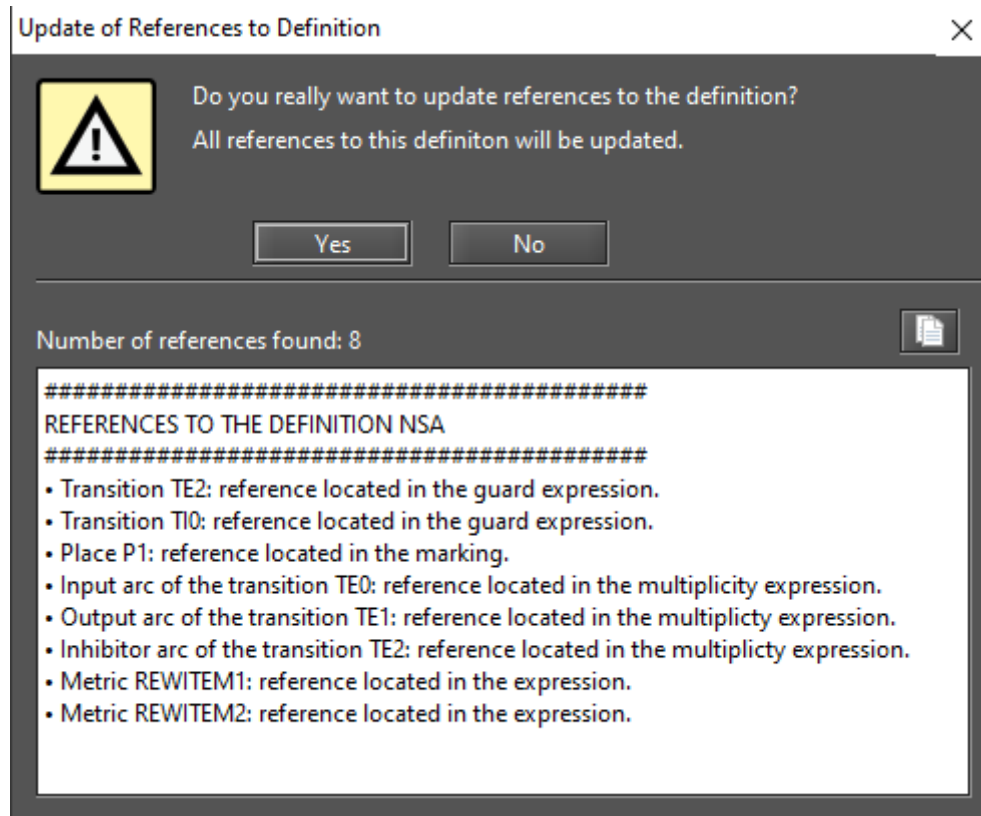


Figure 47: Confirmation to update the references to a Definition

As we can see, all references found to the definition are presented. The user needs to confirm in order to update the definition and all properties of other components making references to it. By cancelling, the definition will not be updated. This behavior aims to prevent the model from being broken. When referring to a definition that does not exist or storing an invalid value for the property, evaluations cannot be carried out. Figure 48 shows the updating log that is shown after the updating process has been successfully finished.

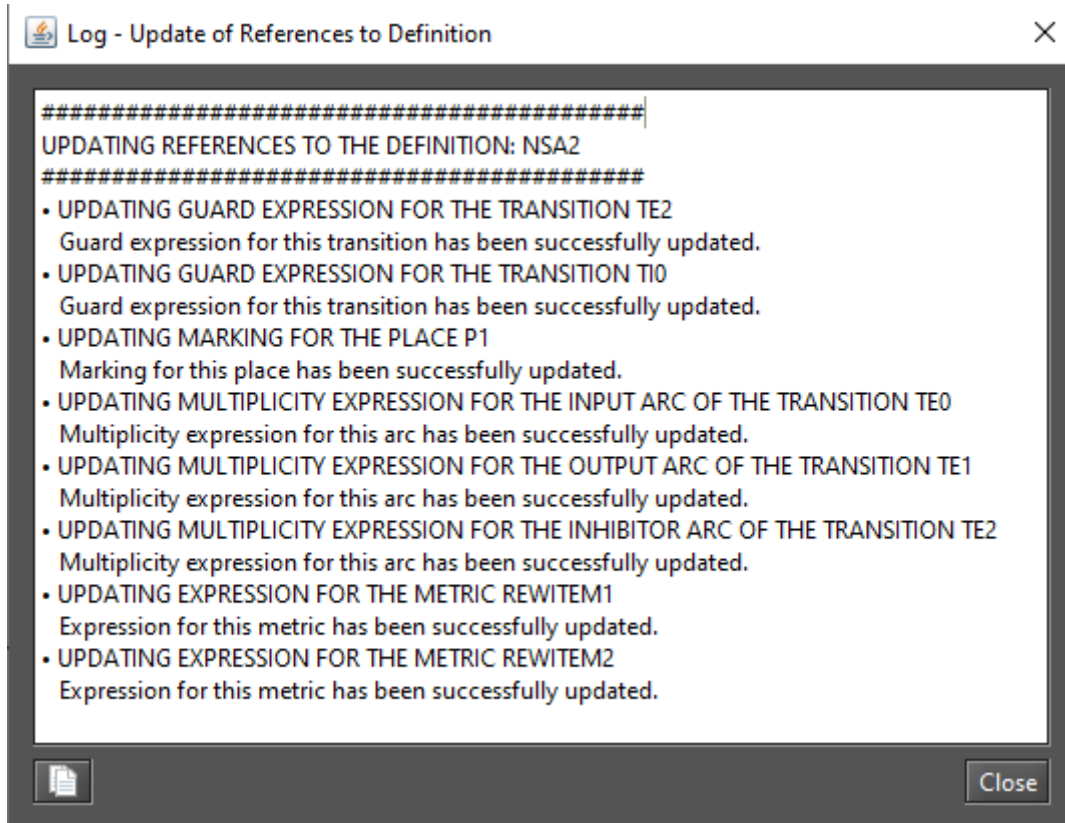


Figure 48: Updating of references to a Definition

In this dialog, the user can see whether all references have been successfully updated. If errors occur in this process then they are presented in the log. An example of an error occurrence is the case when an INT definition with a positive integer value has been defined and referenced by the marking property of a place. By changing the type of the definition to DOUBLE or entering a negative value, the new value of the definition are not accepted to the marking property. As well known, it is not possible to define a negative integer value as marking, thus the new value would not be accepted. In cases like this, the properties making references to are set to their default values, as we can see in Figure 49.

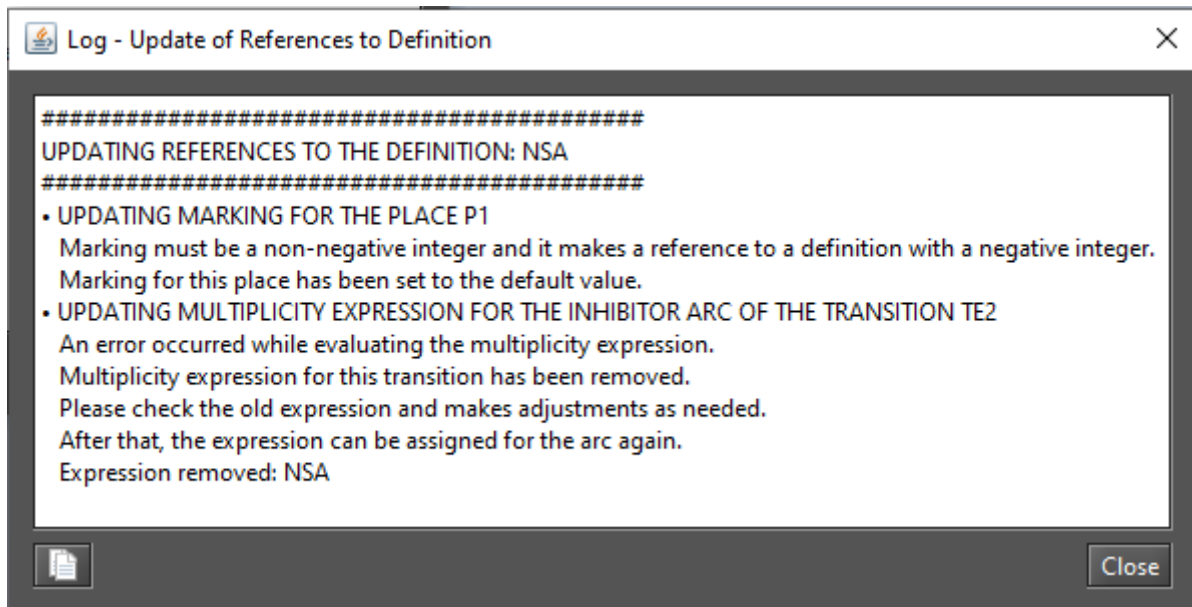


Figure 49: References Updating with Errors

By removing a definition, the same confirmation dialog is presented, as we can see by looking at Figure 50. If the user confirm the removal then all properties making references to the definition are set to their default values (see Figure 51).

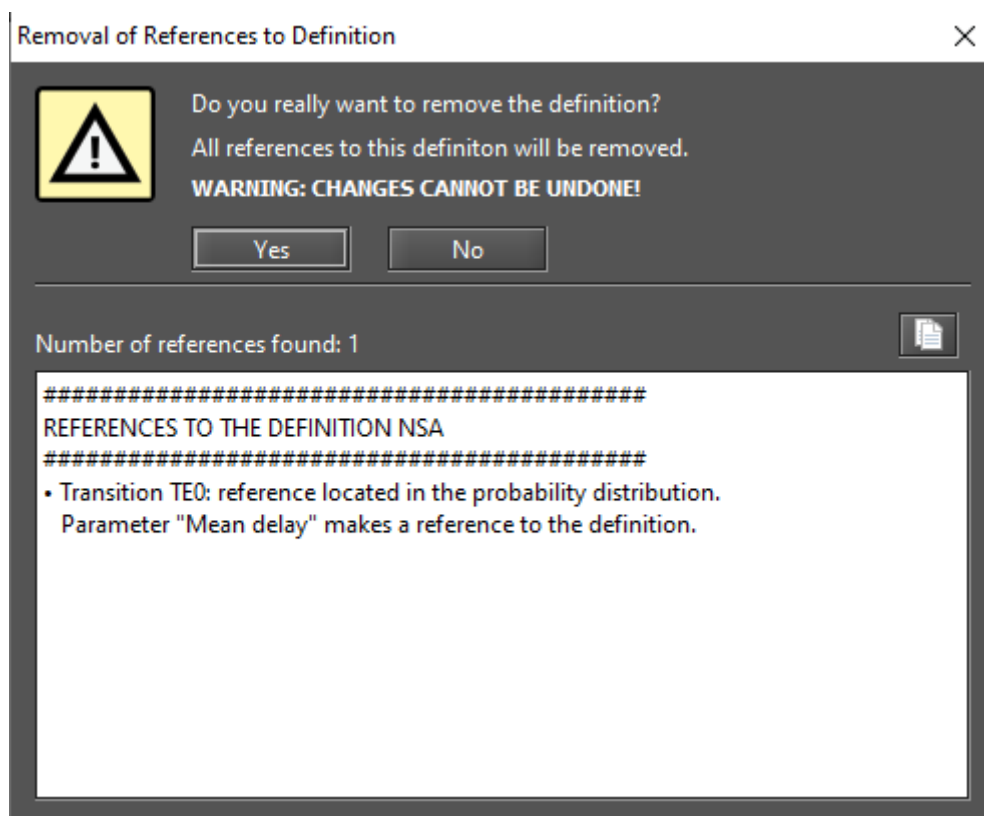


Figure 50: Confirmation to Remove a Definition

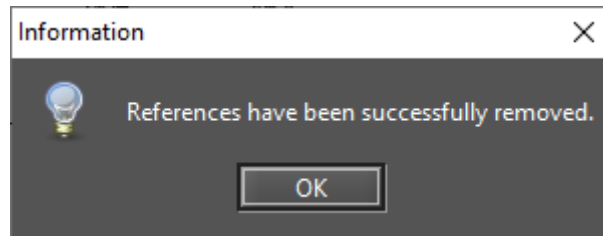


Figure 51: Invalid references are removed

Now, let us take a look at the properties of the component “Metric.” We have already described this component at the page 21. When accessing the properties of a metric, the dialog presented in Figure 52 is opened. Following, we describe each property of metric.

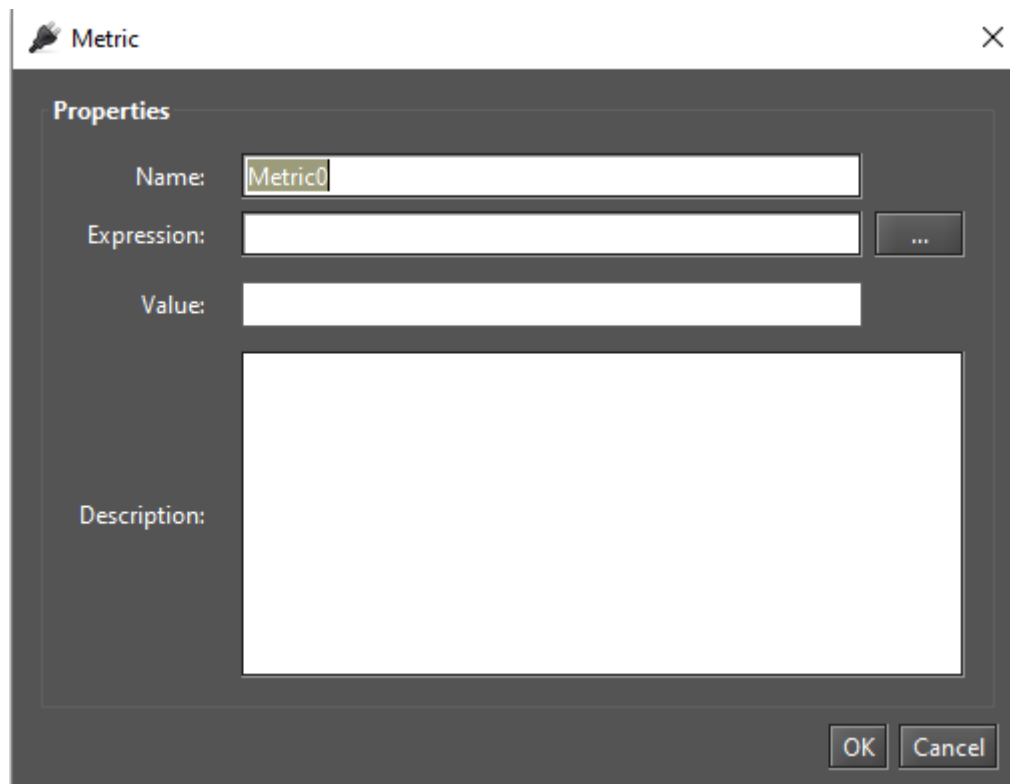


Figure 52: Properties of Metrics

- **Name.** Name for the metric. It is used to identify the metric in the model.
- **Expression.** Expression to be evaluated by performing analyses or simulations. The expression may be used to obtain the state of the model at some point or the time spent to perform an activity, for example. At the appendices, the user can find the syntax to support the creation of simple and complex expressions.
- **Value.** Stores the value of the metric obtained from the last analysis or simulation.
- **Description.** See page 26.

Mercury has a feature that increases the usability of the tool itself as well as the readability of models. Once an SPN component has been inserted, it is possible to read its properties in the drawing area by positioning the

mouse cursor over it. After that, a tooltip appears showing all properties of the component. As we can see in Figure 53, all properties of a transition appear in the tooltip. Mercury provide this feature for all components of all supported formalism.

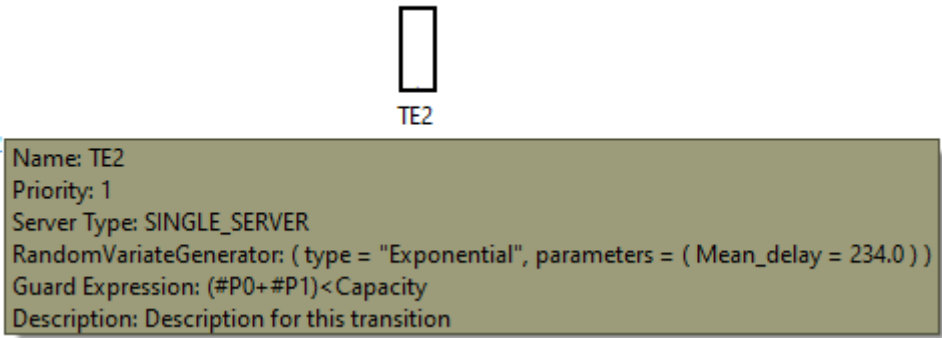


Figure 53: Tooltip for a Transition

Mercury offers for evaluations of SPN models, analyses and simulators. For both, stationary and transient metrics can be evaluated. These features are available in menu “Evaluate” on the option “SPN Evaluation”. Also, they may be accessed by clicking on the command buttons on the main toolbar (see Figure 54 and 55). Next, we will present the simulators and, afterward, the analyses.

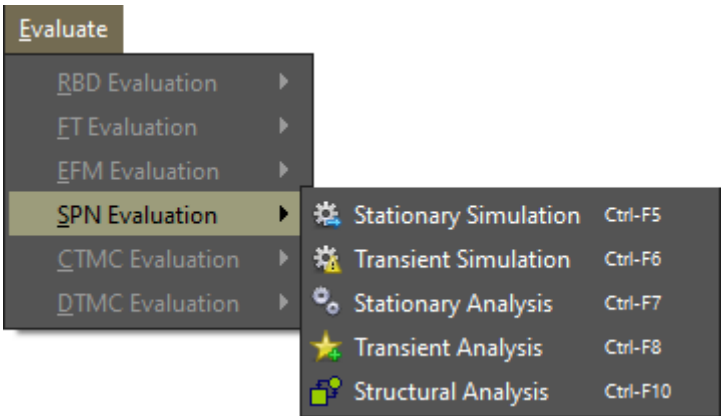


Figure 54: SPN Menu

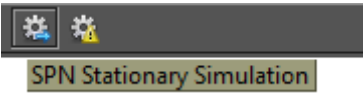


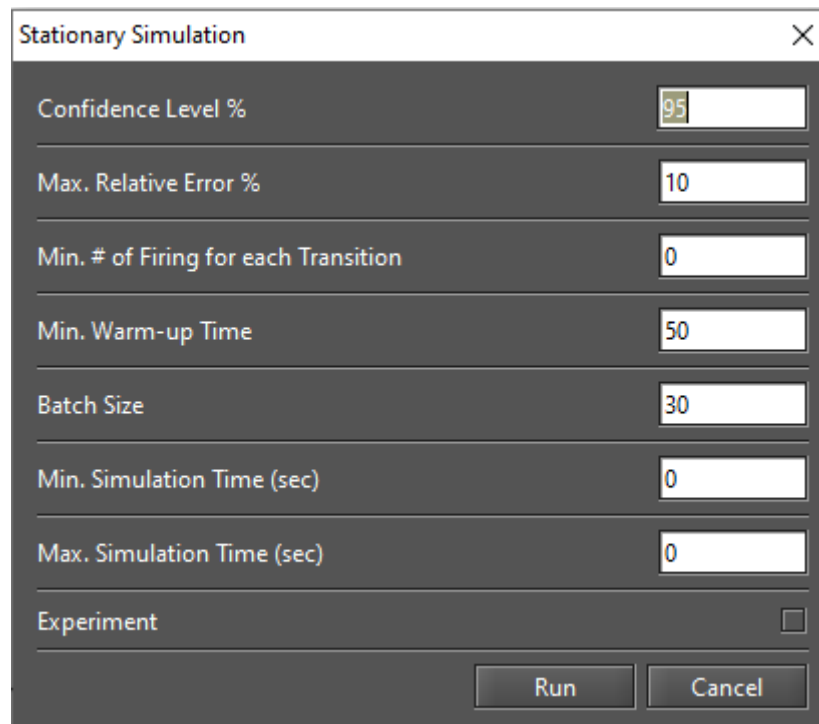
Figure 55: Accessing the SPN Simulators

2.1 SPN Simulation

Models with non-exponential transitions can only be evaluated by performing simulations. Mercury provides two types of simulators. Stationary simulator obtains stationary metrics and the transient simulator obtains time-dependent metrics. Next, we will present the stationary simulator and, in the next section, we will describe the transient simulator.

2.1.1 Stationary Simulation

Figure 56 shows the input parameters for the stationary simulator. These parameters are detailed below.

A screenshot of a software dialog box titled "Stationary Simulation". The dialog box has a dark gray background and a light gray border. It contains several input fields and a checkbox. The fields are: "Confidence Level %" with a value of 95, "Max. Relative Error %" with a value of 10, "Min. # of Firing for each Transition" with a value of 0, "Min. Warm-up Time" with a value of 50, "Batch Size" with a value of 30, "Min. Simulation Time (sec)" with a value of 0, and "Max. Simulation Time (sec)" with a value of 0. There is also a checkbox labeled "Experiment" which is currently unchecked. At the bottom right of the dialog box are two buttons: "Run" and "Cancel".

| Parameter | Value |
|--------------------------------------|--------------------------|
| Confidence Level % | 95 |
| Max. Relative Error % | 10 |
| Min. # of Firing for each Transition | 0 |
| Min. Warm-up Time | 50 |
| Batch Size | 30 |
| Min. Simulation Time (sec) | 0 |
| Max. Simulation Time (sec) | 0 |
| Experiment | <input type="checkbox"/> |

Figure 56: Stationary Simulator

- **Confidence Level.** The confidence interval for obtaining the metrics.
- **Max. Relative Error %.** Defines the maximum relative error in order to stop the simulation.
- **Min. firing for each Transition.** Sets the minimum number of firings for each transition. This is another condition in order to stop the simulation. By entering a value greater than 0, the simulation only stops when the number of firings for each transition is equal to or greater than the defined value and the error criteria has been reached, or the maximum elapsed time is reached, if defined. When setting "0" to this input parameter, the simulator does not consider this stopping condition.
- **Min. Warm-up Time.** Defines the minimum warm-up period. The warm-up phase is the period when the model is not considered to be in steady-state, and metrics are not collected in that period. There are some methods for supporting the evaluation of the moment that the model enters in stationary state,

but Mercury requires the user to define the period of the transient phase. In future versions, we plan to implement some estimation methods to detect the end of the transient phase. As we are evaluating stochastic models, it is expected that the warm-up time is not an deterministic value when a set of simulations is performed. Therefore, the user defines a minimum warm-up time and, once the global simulation time is equal to or greater than the warm-up time defined by the user, the simulation begins to generate batches, collect metrics, and calculate statistics.

- **Batch Size.** Defines the number of samples that will compose each batch in the simulation.
- **Min. Simulation Time (sec).** This time corresponds to the physical time and must be defined in seconds. This time may help us to run simulations on the occasions when the model has possible rare events. Rare events occur when the difference between the delays assigned to the transitions is huge. Rare events may be the reason why there may be no variation in the simulation error. Therefore, the simulator cannot proceed with the simulation by estimating the necessary number of batches to achieve the desired relative error. By entering a minimum simulation time avoids the simulator to stop the simulation when the initial number of batches has no variation in the error. By setting a value greater than 0, the simulation stops when the global time is higher than this time and the simulation error is lower to or equal than the relative error, or other stop conditions is met. If there is no error variation until the minimum time is reached, then the simulation stops. When setting 0 to this parameter, this stopping criteria is not considered.
- **Max. Simulation Time (sec).** It is used to define the maximum time of a simulation. This time corresponds to the physical time and must be defined in seconds. If one of the stopping conditions is not met before this time is reached (minimum simulation time, maximum relative error, and number of firings for each transition), then the simulation stops when this time is reached. When setting 0 to this parameter, this stopping criteria is not considered.
- **Experiment.** Experiment allows us to perform a set of simulation by changing the value of a given parameter into each simulation. The value of the parameter is changed by considering a step size and a minimum and maximum value. At the end of an experiment, Mercury plots a graph showing the impact of each value variation on the selected metric. In this graph, the user can see the average value and confidence intervals of each point.

Figure 57 shows the stationary simulator in action.

The information displayed on this window is self-describing. The stationary simulator has two tabs. “*Batches and Errors*” tab displays the logs of the batches processed as well as the relative error of the simulation up to that point (see Figure 58). “*Transitions Firings*” tab shows the number of firings for each fired transition as well as the percentage of firing in relation to the other fired transitions (see Figure 68).

The simulation finishes when one of the following criterias is reached: maximum relative error, minimum simulation time when there is no variation in the error, maximum simulation time, or the minimum number

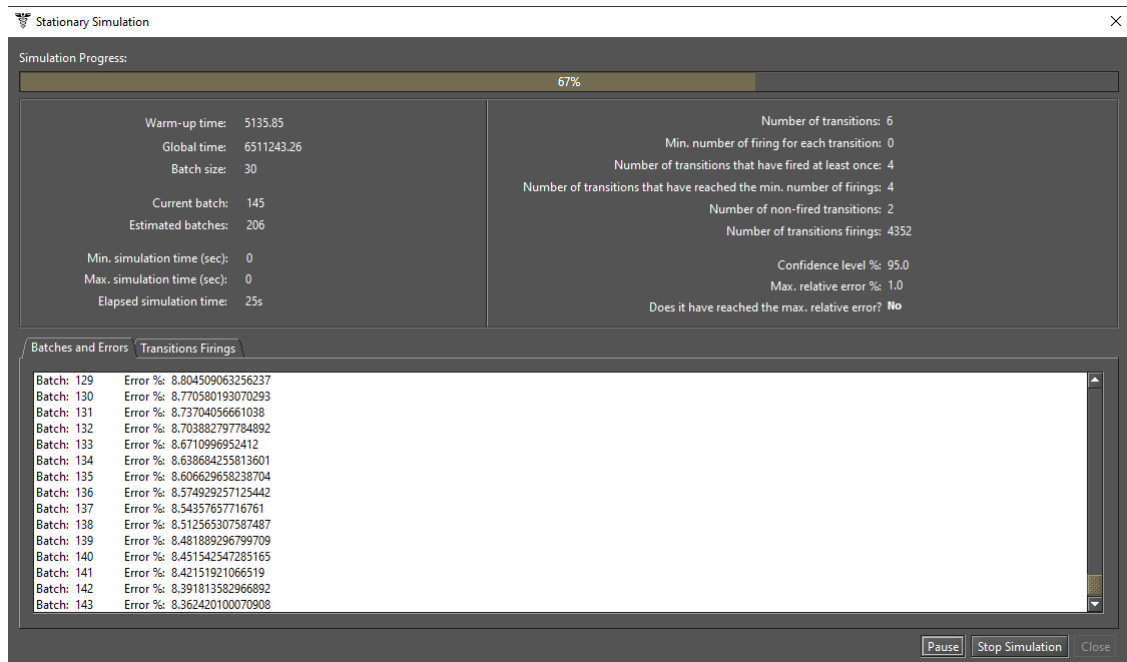


Figure 57: Stationary Simulator

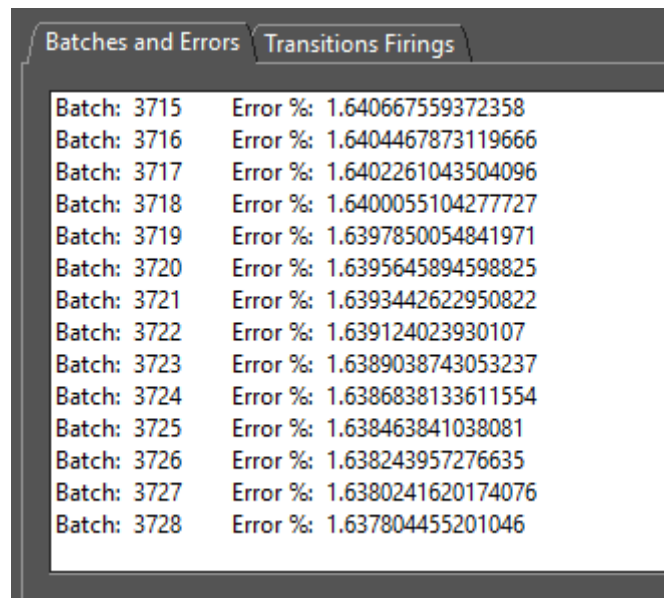


Figure 58: Batches and Errors

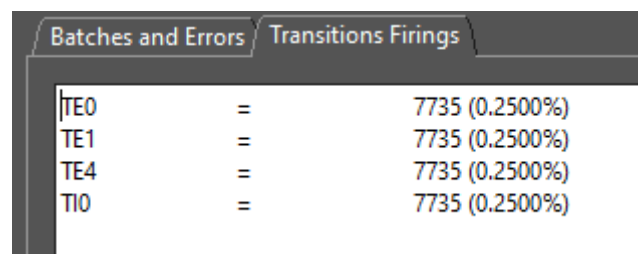


Figure 59: Transitions Firings

of firings for each transition and the error has also been reached, whatever occurs first. There is a progress bar on the top of that window that shows the progress of the simulation. This simulation progress undergoes

adjustments depending on the simulation, as the previously estimated number of batches to reach the relative error can be re-estimated, thus changing the overall progress of the simulation. In addition, the user can pause and continue the simulation, as well as stop it at any time (see Figure 69).

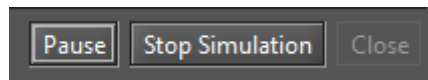


Figure 60: Buttons on the Stationary Simulator

When a simulation finishes, the user can export the result as plain text or as a spreadsheet (MS Excel) (see Figure 61). A large number of statistics is computed by considering the result of a simulation.

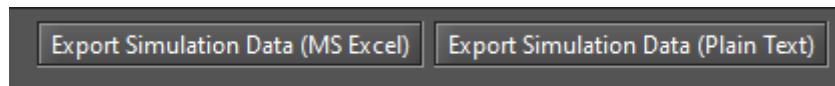


Figure 61: Export Buttons

Some statistics generated by the simulator are:

- Sample Size
- Mean
- Midrange
- Minimum
- 1st Quartile
- 2nd Quartile
- 3rd Quartile
- Maximum
- IQR (interquartile range)
- Range
- RMS (root mean square)
- Variance
- Standard Deviation
- Mean Absolute Deviation
- Coeff. Of Variation
- Sum

- Sum of Squares
- Skewness
- Kurtosis
- Standard Error
- Relative Error

At the end of the simulation, the result is shown in the "Result" tab of the main window. Figure 62 demonstrates the example model we have used in the simulator. Listing 2 shows an example of the output generated by the simulator. In this example, only one metric has been evaluated.

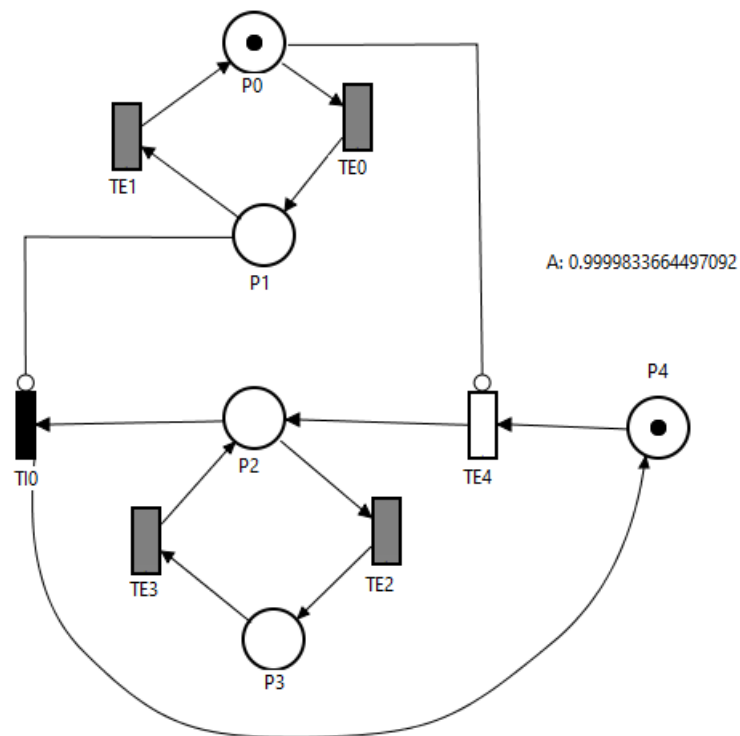


Figure 62: SPN Model for Stationary Simulation

Listing 1: Stationary Simulation Result

STATIONARY SIMULATION RESULT

Confidence Level %: 95.0

Max. Relative Error %: 1.0

Min. Firing **for** each Transition: 0

Max. Simulation Time: 0

Min. Warm-up Period: 50

Warm-up Period: 5135.85

Global Time: 453749665.85

Batch Size: 30

Batches: 10000

Transitions Firings: 300001

Fired Transitions: 4

Non-Fired Transitions: 2

Fired Transitions

TE0 = 75001 (0.2500%)

TE1 = 75000 (0.2500%)

TE4 = 75000 (0.2500%)

TI0 = 75000 (0.2500%)

Non-Fired Transitions

TE2

TE3

Descriptive Statistics

Metric : A, $P\{(\#P0>0)OR(\#P2>0)\}$

Result: 0.9999833664497092

Nines: 4.7790150443977675

Confidence Interval: [0.9999833628996808,0.9999833699997376]

Standard Error: 1.811053049610908E-9

Error %: 1.0

Sample Size , n: 10000
 Midrange: 0.9999817534036552
 Minimum: 0.9999784761307453
 1st Quartile: 0.9999833204467997
 2nd Quartile: 0.9999833471453827
 3rd Quartile: 0.9999833889670042
 Maximum: 0.9999850306765653
 IQR: 6.852020451031393E-8
 Range: 6.5545458199922635E-6
 RMS: 0.9999833664497279
 Variance , s^2 : 3.2799131485049696E-14
 Standard Deviation , s: 1.811053049610908E-7
 Mean Absolute Deviation: 6.560655693113038E-8
 Coeff. Of Variation: 1.8110831743539695E-7
 Sum: 9999.833664497091
 Sum Sq: 9999.667331761308
 Skewness: -11.6910502101944
 Kurtosis: 233.18498785918288

Now, we will demonstrate how to execute experiments in the stationary simulator. Figure 63 shows the dialog box that appears when confirming the input parameters for the simulation and checking the “Experiment” option.

Figure 63: Executing an experiment in the Stationary Simulation

In this window, the user must select the parameter that will be changed, its minimum and maximum value as well as a step size. In addition, the user must select the metric that will be evaluated. At the end of the simulation, a graph will be plotted considering the value of the metric for each change in the parameter’s value. As we can see in Figure 64, the mean value and its confidence intervals are presented for each point. When placing the

mouse cursor over the point that represents the mean, the values of the confidence intervals are displayed as a tooltip.

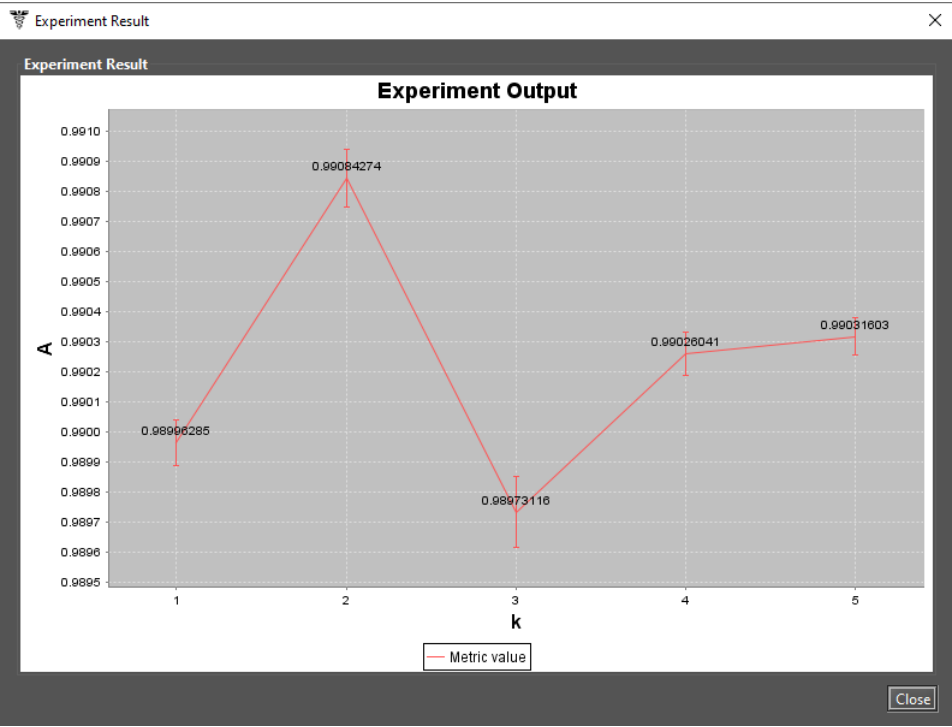
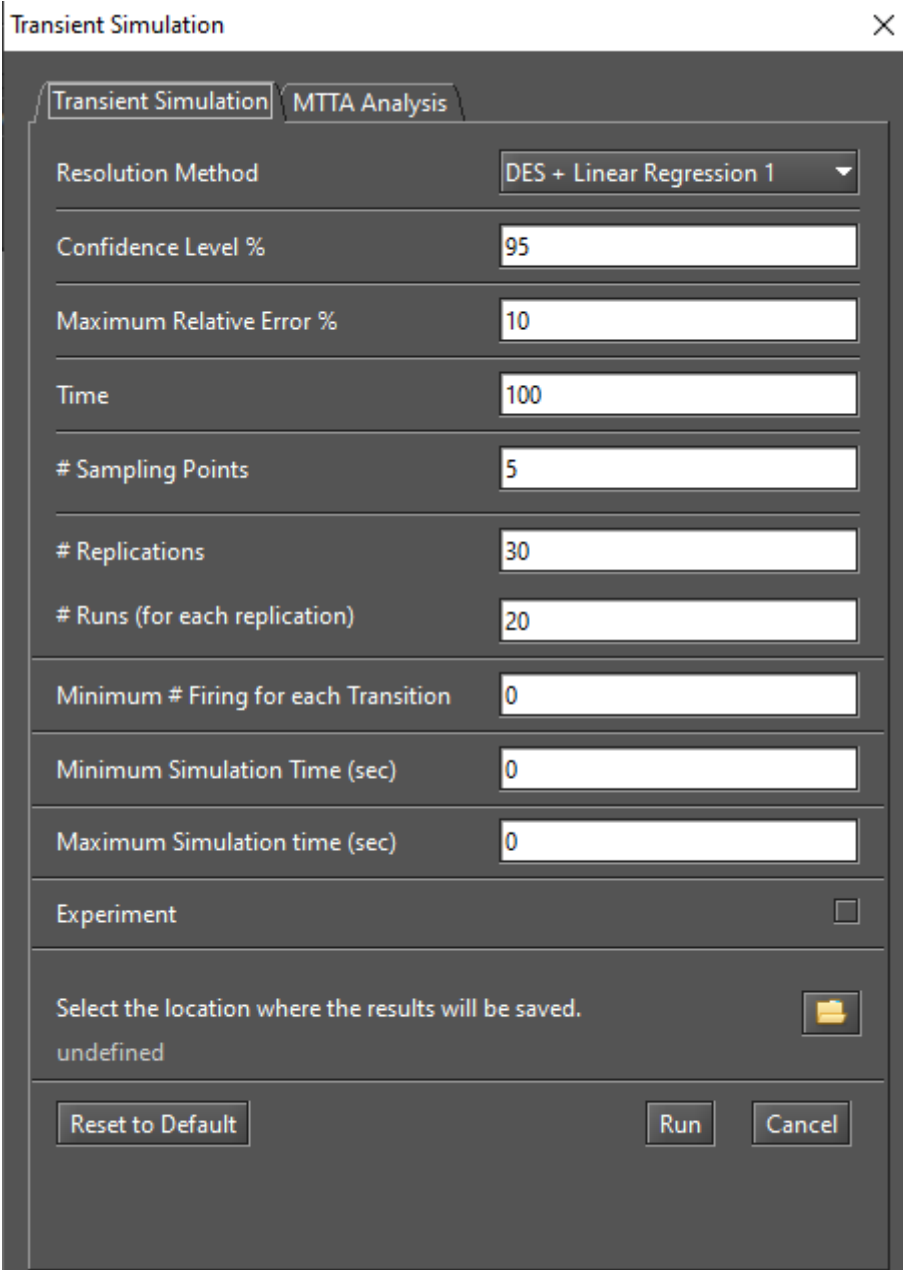


Figure 64: Output of a Stationary Simulation Experiment for an SPN Model

2.1.2 Transient Simulation

Models with non-exponential transitions can only be evaluated by performing simulations. Transient simulations may be used when the user is interested in evaluate metrics considering a specific point in time. A transient simulation is composed by a set of replications and each replication is composed by a set of runs. Each run runs from time 0 until the time t' is reached, which is defined by the user in the parameter "Time". When the current set of runs is finished, the value of each sampling point of the current replication is computed. The replication represents the mean values of the points in its set of runs.

Transient simulator may be accessed in the menu Evaluate -> SPN Evaluation -> Transient Simulation. Figure 65 shows the input parameters for the transient simulator. Each parameter is described below.



The image shows a software dialog box titled "Transient Simulation" with a close button (X) in the top right corner. It contains two tabs: "Transient Simulation" (selected) and "MTTA Analysis". The dialog lists several input parameters, each with a text field or a dropdown menu:

- Resolution Method: DES + Linear Regression 1 (dropdown)
- Confidence Level %: 95 (text field)
- Maximum Relative Error %: 10 (text field)
- Time: 100 (text field)
- # Sampling Points: 5 (text field)
- # Replications: 30 (text field)
- # Runs (for each replication): 20 (text field)
- Minimum # Firing for each Transition: 0 (text field)
- Minimum Simulation Time (sec): 0 (text field)
- Maximum Simulation time (sec): 0 (text field)
- Experiment: ☐ (checkbox)
- Select the location where the results will be saved: undefined (text field with a folder icon button)

At the bottom of the dialog, there are three buttons: "Reset to Default", "Run", and "Cancel".

Figure 65: Input Parameters for the Transient Simulator

- **Resolution Method.** Mercury supports two methods for computing the values of the points in the transient simulation.

“DES + Linear Regression 1” computes the value of each point at the end of each run by adopting linear interpolation between two known points. When the number of runs of a replication has been executed, the values of each point are collected in the current set of runs, and its mean value is assigned to that same point in the current replication.

“DES + Linear Regression 2” calculates the value of each sampling point of the current replication when its set of runs has been executed. Oposite to the first method, this one computes the value of each point of the current replication considering its entire set of runs. This method adopts linear regression between multiple known points. For each sampling point, this method considers two set of events. The first one comprises the set of last events occurred before the evaluated point. The second one is the set of the first events occurred in the evaluated point or after it. In each run it is collected the event occurred before the evaluated point, and the event that occurred in the evaluated point or after it. For each set of points is computed the mean value of the metric and the mean time of events occurrences. After that, the value of the metric is estimated for the current sampling point.
- **Confidence Level.** The confidence interval for obtaining the metrics.
- **Max. Relative Error %.** Defines the maximum relative error in order to stop the simulation.
- **Time.** Defines the evaluation time (t'). Each run starts from time 0 until the time t' is reached. This time may be divided in various intermediare points and each metric is evaluated for each point. The intermediare points is defined by the number of sampling points.
- **Sampling Points.** Defines the number of sampling points evaluated during the simulation. The time interval from time 0 to time t' is devided into intermediare points considering this number of sampling points. When the user chooses to evaluate only one sampling point, it is only considered the value of the metric at time t' .
- **Replications.** Defines the initial number of replications of the simulation. If the initial number of replication is reached and the simulation error has not yet been reached, then the simulator estimates a new of replications considering the current state of the simulation in order to achieve the desired error. The simulator always reestimate the number of expected replications in order to achieve the simulation error when the number from the last estimation is reached and the error has not yet been reached.
- **Runs.** Defines the number of runs for each replication. As we aforementioned, each replication is composed by a set of runs. Each run starts from time 0 until the time t' is reached. When the number of runs is reached for the current replication, the values for the point t' and intermediare points, if any, are computed and assigned to the current replication. After that, a new replication will be initiated whether any stop criteria have not been met.

- **Min. firing for each Transition.** Defines the minimum number of firings for each transition. This is another condition in order to stop the simulation. By entering a value greater than 0, the simulation only stops when the number of firings for each transition is equal to or greater than the defined value and the error criteria has been reached, or the maximum elapsed time is reached, if defined. When setting “0” to this input parameter, the simulator does not consider this stopping condition.
- **Minimum Simulation Time (sec).** This stopping criteria defines the minimum elapsed time of a simulation. This time corresponds to the physical time and must be defined in seconds. This criteria may help us to run simulations when the model has possible rare events. Rare events occur when the difference between the delays assigned to the transitions is huge. Rare events may be the reason why there may be no variation in the simulation error. Therefore, the simulator cannot proceed with the simulation by estimating the necessary number of replications in order to achieve the desired relative error, as the relative error has not changed since the beginning (it is 0). By entering a minimum elapsed time avoids the simulation to be stopped when this situation occurs. If the minimum time is reached and there was no error variation, then the simulation finishes. Otherwise, the simulation runs until the error or other stopping criteria is reached. By setting “0” to this input parameter, the simulator does not consider this stopping condition.
- **Maximum Simulation Time (sec).** This stopping criteria defines the maximum elapsed time of a simulation. This time corresponds to the physical time and must be defined in seconds. If any stopping criteria are not met before this time is reached (minimum simulation time, maximum relative error, and number of firings for each transition), then the simulation finishes when this time is reached. By setting “0” to this input parameter, the simulator does not consider this stopping criteria.
- **Experiment.** Experiment allows us to perform a set of simulation by changing the value of a given parameter into each simulation. The value of the parameter is changed by considering a step size and a minimum and maximum value. At the end of an experiment, Mercury plots a graph showing the impact of each value variation on the selected metric. In this graph, the user can see the average value and confidence intervals of each point.
- **Location.** Before starting the simulation, the user needs to specifies the location where the results will be saved.

Figure 66 shows the transient simulator in action.

The information displayed on this window is self-describing. The transient simulator has two tabs. “*Replications and Errors*” tab displays the logs of the replications that have been processed as well as the relative error up to that point (see Figure 67). “*Transitions Firings*” tab shows the number of firings for each fired transition as well as the percentage of firing in relation to the other fired transitions (see Figure 68).

The simulation finishes when one of the following criterias is reached: maximum relative error, minimum

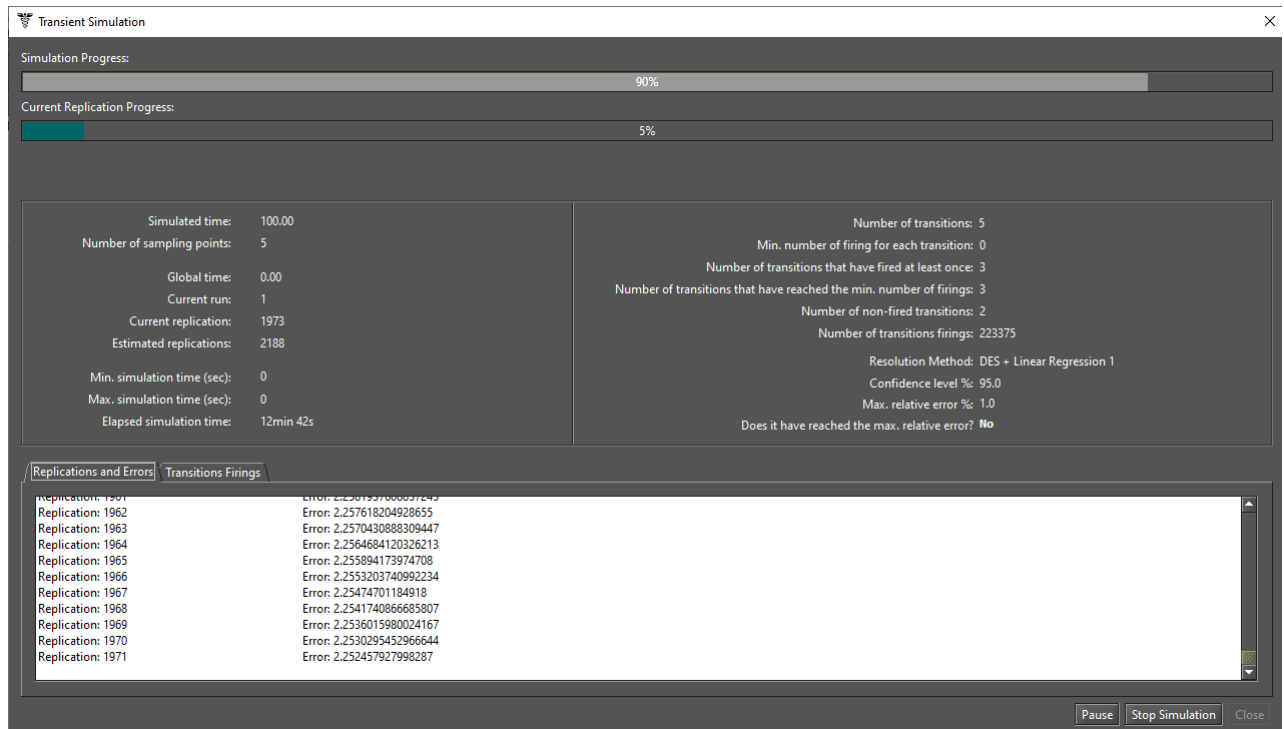


Figure 66: Transient Simulator

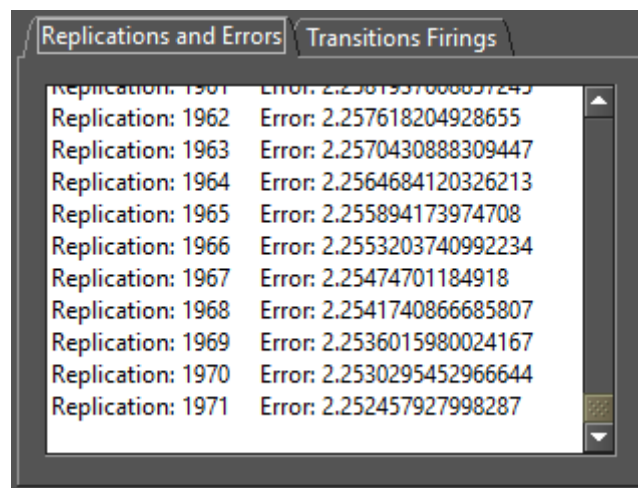


Figure 67: Replications and Errors

simulation time when there is no variation in the error, maximum simulation time, or the minimum number of firings for each transition and the error has also been reached, whatever occurs first. There is a progress bar on the top of that window that shows the progress of the simulation. That simulation progress undergoes adjustments depending on the simulation, as the previously estimated number of replications to reach the relative error can be re-estimated, thus changing the overall progress of the simulation. In addition, the user can pause and continue the simulation, as well as stop it at any time (see Figure 69).

A large number of statistics is computed by considering the result of a simulation. Some statistics generated by the simulator are:

| Replications and Errors | | Transitions Firings |
|-------------------------|---|---------------------|
| TE0 | = | 117151 (0.4738%) |
| TE1 | = | 60223 (0.2436%) |
| TE2 | = | 0 (0.0000%) |
| TI0 | = | 69879 (0.2826%) |
| TI1 | = | 0 (0.0000%) |

Figure 68: Transitions Firings

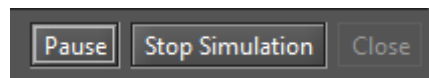


Figure 69: Buttons on the Transient Simulator

- Sample Size
- Mean
- Midrange
- Minimum
- 1st Quartile
- 2nd Quartile
- 3rd Quartile
- Maximum
- IQR (interquartile range)
- Range
- RMS (root mean square)
- Variance
- Standard Deviation
- Mean Absolute Deviation
- Coeff. Of Variation
- Sum

- Sum of Squares
- Skewness
- Kurtosis
- Standard Error
- Relative Error

At the end of a simulation, the result is shown in the "Result" tab of the main window. Listing 2 shows an example of an output generated by the transient simulator. In this example, only one metric has been evaluated.

Listing 2: Transient Simulation Result

```
#####
TRANSIENT SIMULATION RESULT
#####

Results have been successfully saved in the following directory:
G:\Modelos\

-----

Input Parameters
-----

Resolution Method: DES + LINEAR REGRESSION 1
Confidence Level %: 90.0
Max. Relative Error %: 10.0
Simulated Time: 100.0
Number of Sampling Points: 5
Number of Replications: 30
Number of Runs: 20
Min. Firing for each Transition: 0
Min. Simulation Time(sec): 0
Max. Simulation Time(sec): 0

-----

Result
-----

Replications: 100 (the simulation has been finished on
this replication)
```

Transitions firings: 11219

Fired transitions: 3

Non-fired transitions: 2

Transitions Firings Log

| | | |
|-----|---|----------------|
| TE0 | = | 5338 (0.4758%) |
| TE1 | = | 2714 (0.2419%) |
| TE2 | = | 0 (0.0000%) |
| TI0 | = | 3167 (0.2823%) |
| TI1 | = | 0 (0.0000%) |

Descriptive Statistics

Metric: MRT, $((E\{P0\}) + (E\{P3\})) / ((1 / \text{Arrival}) * (1 - (P\{P1=0\})))$

Simulated Time: 100.0

Result: 47.525

Nines: NaN

Confidence Interval: [45.39652245547025,49.65347754452975]

Standard Error: 1.2819133229968283

Error %: 10.0

Sample Size, n: 100

Midrange: 53.75

Minimum: 25.0

1st Quartile: 38.75

2nd Quartile: 45.0

3rd Quartile: 56.25

Maximum: 82.5

IQR: 17.5

Range: 57.5

RMS: 49.20683387498123

Variance, s^2 : 164.33017676767705

Standard Deviation, s: 12.819133229968282

Mean Absolute Deviation: 10.179999999999996

Coeff. Of Variation: 0.26973452351327265

Sum: 4752.5

Sum Sq: 242131.25

Skewness: 0.5220765348073639

Kurtosis: -0.0317526461636799

2.1.3 MTTA Simulation

Mercury provides a type of evaluation in the transient simulator that allows us to evaluate the behavior of absorbing models and from there generating a large number of results. Figure 70 shows an example of an absorbing model.

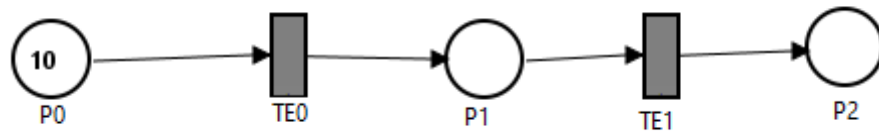


Figure 70: Absorbing SPN Model

Mean time to absorption (MTTA) simulation is accessed by following the menu depicted in Figure 71.

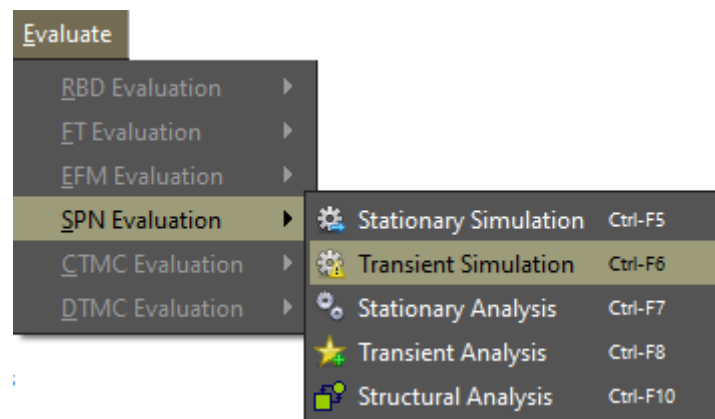


Figure 71: Accessing the Transient Simulator

By accessing this menu, a window with two tabs is displayed (see Figure 72). The first tab contains the input parameters for the default transient simulator (see the previous section). The second tab ("MTTA Analysis") shows the input parameters for MTTA simulation, and these parameters are described below:

- **Confidence Level %.** Confidence interval for generating the statistics.
- **Number of Samples.** Number of samples that the simulator will collect. After collecting the samples, statistics are generated from them.
- **Relative Error %.** Maximum relative error to be considered. The MTTA simulation only stops when the relative error of the simulation is equal to or lower than the relative error defined by the user.

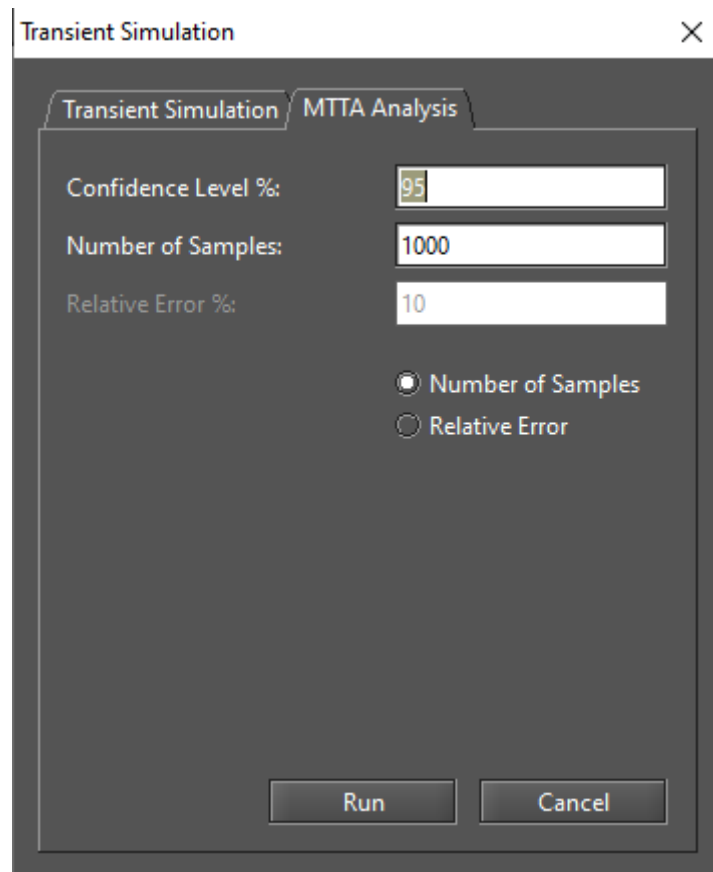


Figure 72: MTTA Analysis Dialog

At the end, a window shows the results of the transient simulation for the absorbing model under evaluation. In tab "Summary" are presented statistics about the simulation. As we can see in Figure 73, a large number of statistics is computed. Listing 3 shows the output of an MTTA simulation in detail.

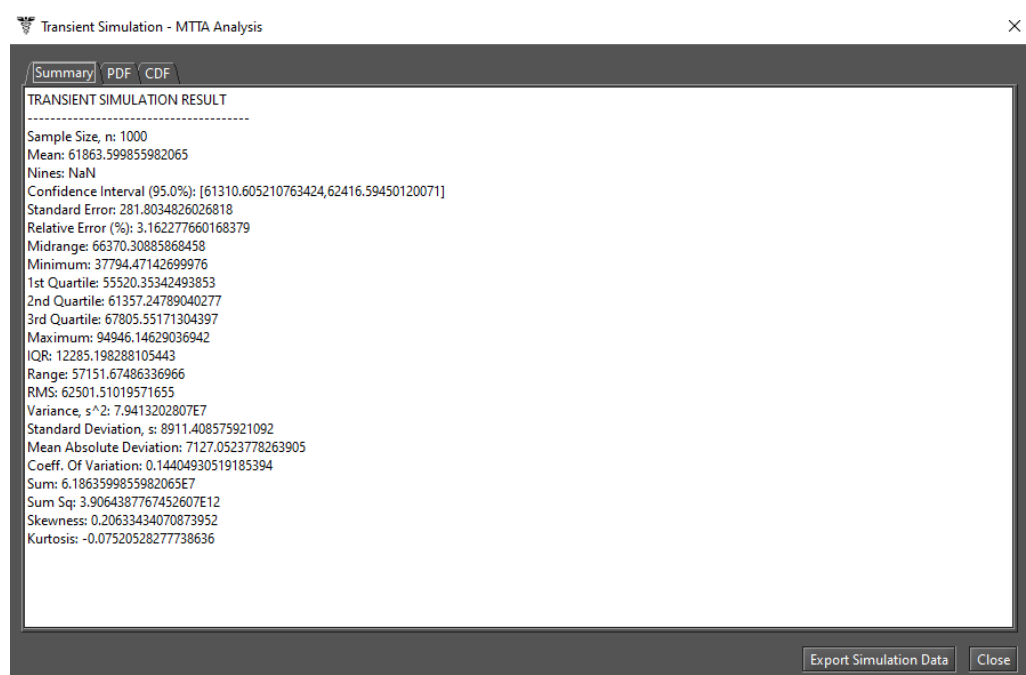


Figure 73: MTTA Result - Summary

Listing 3: MTTA Result

```

MTTA TRANSIENT SIMULATION RESULT
-----

Sample Size , n: 1000
Mean: 61863.599855982065
Nines: NaN
Confidence Interval (95.0%): [61310.605210763424,62416.59450120071]
Standard Error: 281.8034826026818
Relative Error (%): 3.162277660168379
Midrange: 66370.30885868458
Minimum: 37794.47142699976
1st Quartile: 55520.35342493853
2nd Quartile: 61357.24789040277
3rd Quartile: 67805.55171304397
Maximum: 94946.14629036942
IQR: 12285.198288105443
Range: 57151.67486336966
RMS: 62501.51019571655
Variance, s^2: 7.9413202807E7
Standard Deviation, s: 8911.408575921092
Mean Absolute Deviation: 7127.0523778263905
Coeff. Of Variation: 0.14404930519185394
Sum: 6.1863599855982065E7
Sum Sq: 3.9064387767452607E12
Skewness: 0.20633434070873952
Kurtosis: -0.07520528277738636

```

"PDF" tab displays the probability distribution function of the generated data (see Figure 74). The cumulative distribution function of these data is shown in the tab "CDF" (see Figure 75). When placing the cursor on any blue point on the plotted curve, the tool shows the values of the x and y axes as a tooltip (see Figure 76). The user has also an option to export the result to an MS Excel spreadsheet (a .xls file).

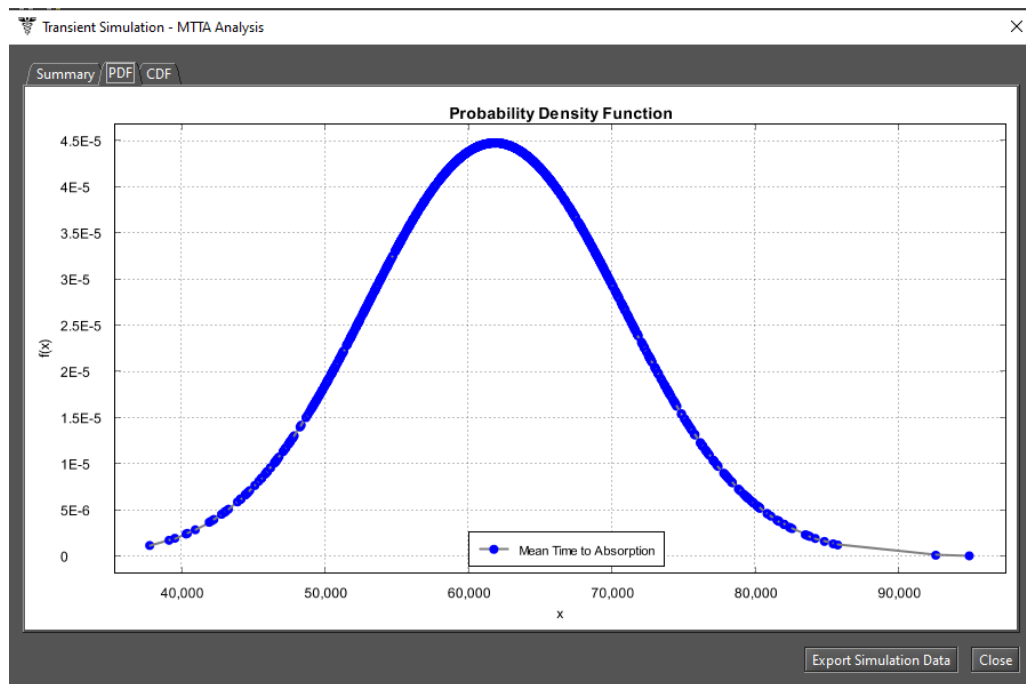


Figure 74: MTTA Result - PDF

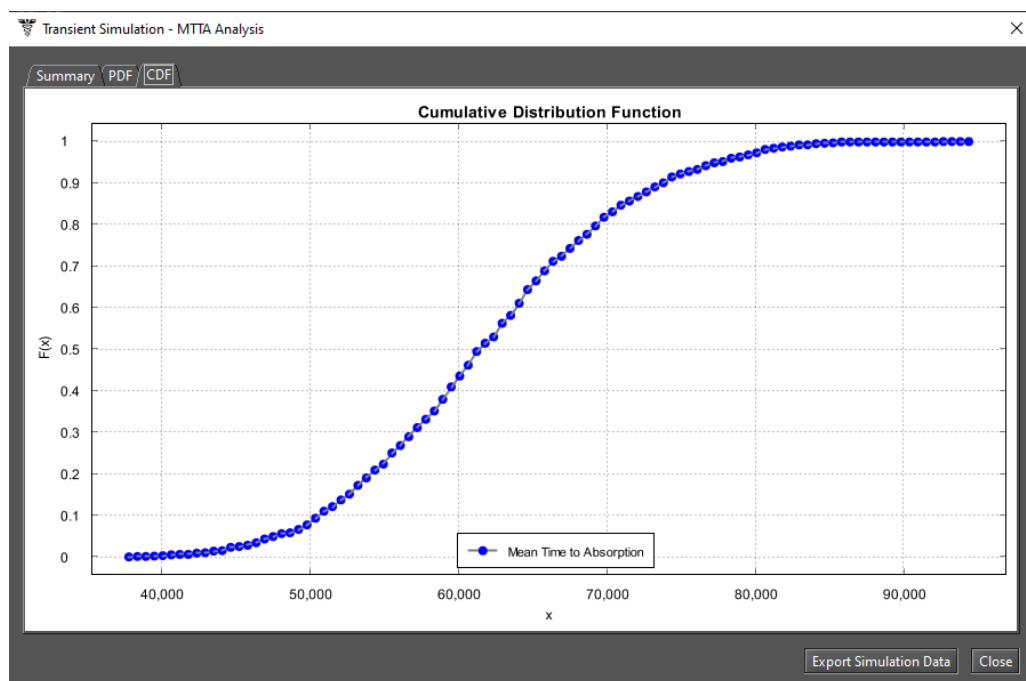


Figure 75: MTTA Result - CDF

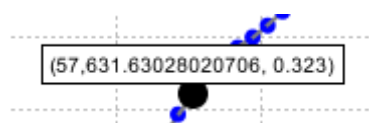


Figure 76: X and Y Axis Values

2.2 SPN Analysis

Stationary Analysis and **Transient Analysis** both compute results by generating the underlying CTMC related to the state space of the SPN model under evaluation. Stationary analysis computes steady-state probabilities, useful for evaluating the long-term average behavior of modeled systems. On the other hand, transient analysis computes time-dependent probabilities, useful for evaluating the behavior of modeled systems at a particular point in time.

2.2.1 Stationary Analysis

Figure 77 shows the “Stationary Analysis” window, which has a combo box for selecting one of two solution methods available: **Direct - GTH** (Grassmann-Taksar-Heyman) and **Iterative - Gauss-Seidel**.

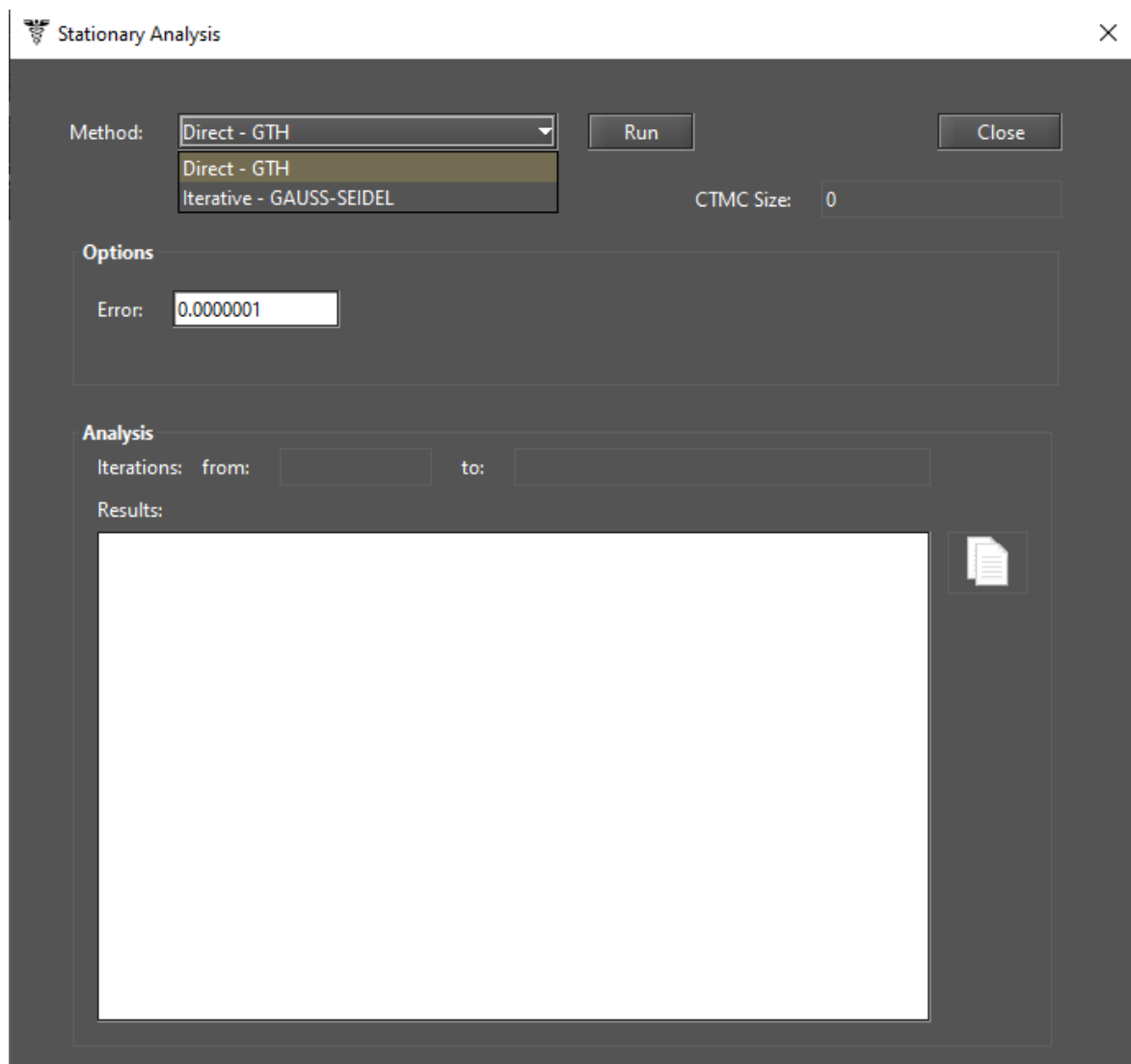


Figure 77: Stationary Analysis Window

When solving a 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 button “Run”, the solution algorithm is triggered and as soon as it finishes, results are presented in the text area at the bottom of the window (see

Listing 4).

Listing 4: Stationary Analysis for an SPN

```
Tue Feb 11 07:01:25 BRT 2020
Performing stationary analysis ...
Generating CIMC...
CIMC generated... (1s)
Executing GIH numerical method...
Done! (elapsed time: 1s)
S0=0.9903691816162996
S1=0.009630818383700439
```

When solving the model through Gauss-Seidel, besides the 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 the convergence of results is reached, considering the error entered in the input dialog (see Figure 78).

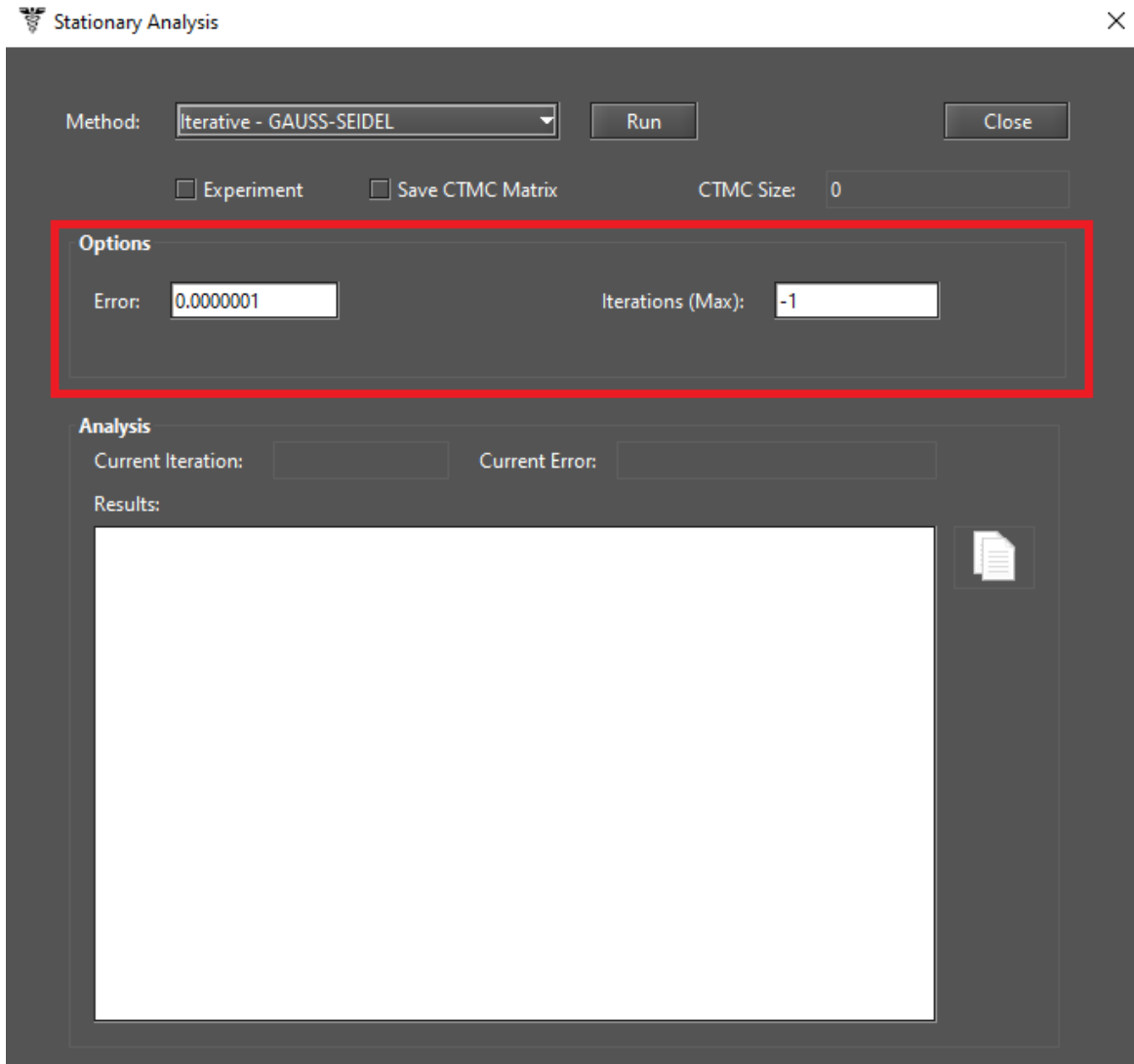


Figure 78: Stationary Analysis Window - Gauss-Seidel Method

Metrics are updated as soon as the analysis is completed, independently of the chosen method, their values are updated in the drawing area, as depicted by Figure 79, where a metric named “Availability” has been defined.

SPN models can also be solved for a range of values of user-defined parameters. This is accomplished by checking “Experiment” on the “Stationary Analysis” window and then clicking on button “Run”. A new dialog is displayed in order for the user to define the input parameters for the experiment to be carried out (see Figure 80).

Below, we describe each of them.

- **Varying Parameter.** Parameter (definition) that will have its value changed.
- **Output measure.** Metric to be evaluated.
- **Range Minimum Value.** Initial value to be assigned to the selected parameter.
- **Range Maximum Value.** Final value to be assigned to the selected parameter.
- **Interval.** It is the step-size for changing the value of the parameter. The parameter starts with the

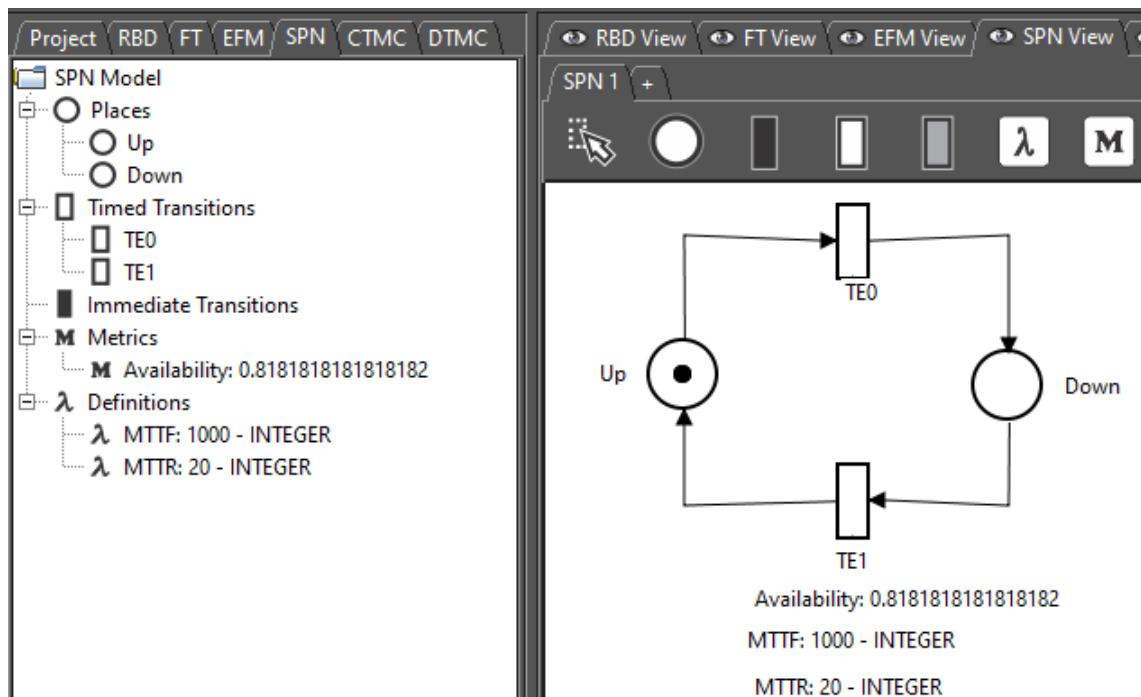


Figure 79: An SPN Model for Analysis

Figure 80: SPN Experiment

minimum value and its value is increased by considering the stepsize. At each change, the selected metric is evaluated. Experiment ends when the maximum value for the parameter is reached.

At the end of the experiment, results are shown and a graph is plotted, as we can see by looking at Figures 81 and 82, respectively.

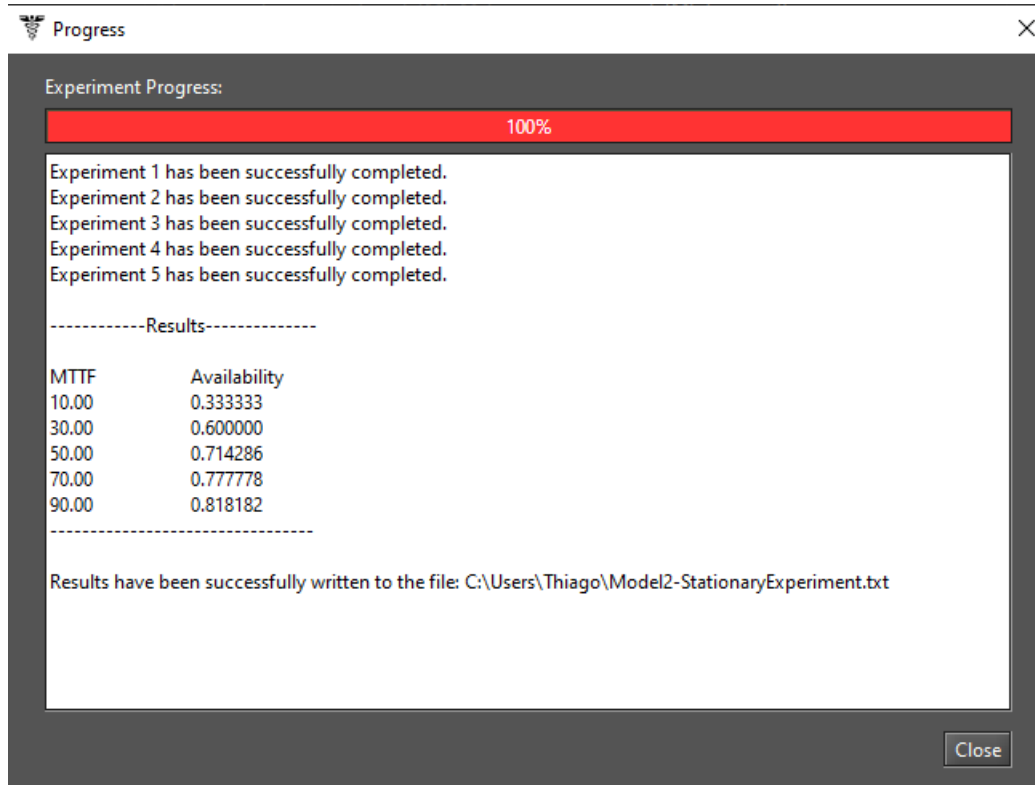


Figure 81: Results of an SPN Experiment

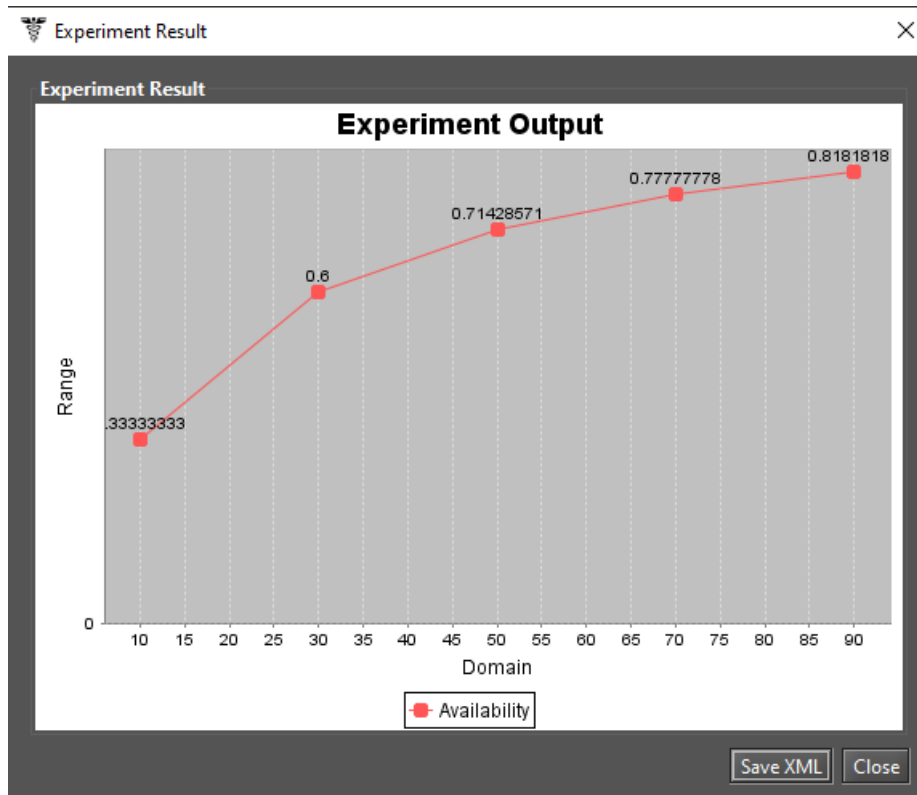


Figure 82: Graph of an SPN Experiment

2.2.2 Transient Analysis

Figure 83 shows the window “Transient Analysis” , which has a combo box for selecting one of the two solution methods available: **Uniformization** (also known as Jensen’s method) and **Runge-Kutta (4th order)**.

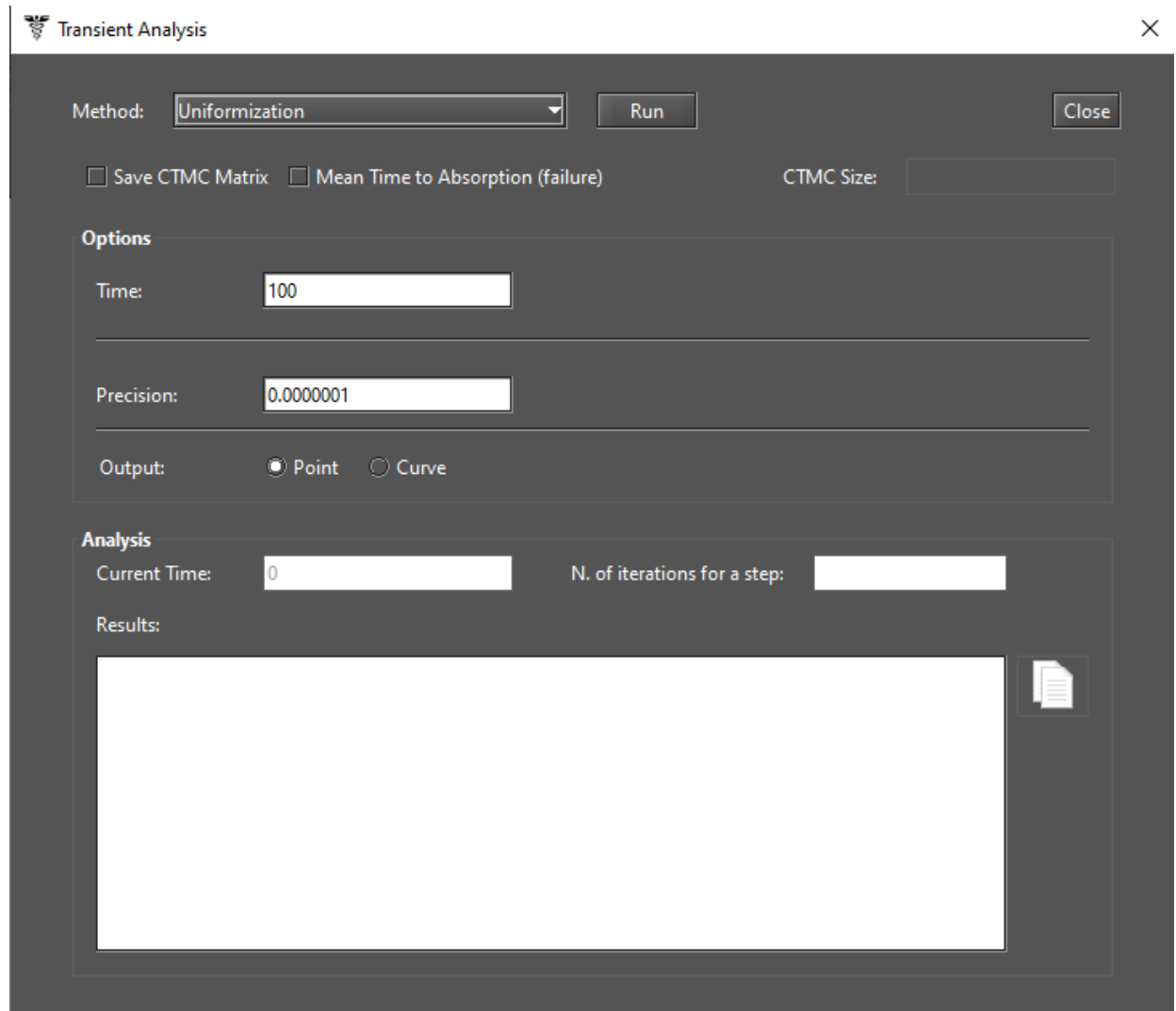


Figure 83: Transient Analysis Window

When solving a model, the user can define:

- **Time** for which the analysis will be carried out (default: 100).
- **Precision** of results (default: 10^{-7}),

By selecting the Uniformization method, note that the time required for obtaining results is proportional to the time entered by the user for the analysis because Uniformization is an iterative algorithm.

By clicking on button “Run”, the solution algorithm is triggered. As soon as it finishes, results are presented in the text area at the bottom of the window, also they are written in a plain text file having the filename of the project appended with the “-TransientAnalysis.txt” suffix.

This window also allows the user to choose between a **Point** or **Curve** analysis. **Point** analysis is the default, and it shows results only for the specific point in time. **Curve** analysis writes in a plain text file all measures values computed in intermediate steps from time equals zero until the specified time.

Mean time to absorption (**MTTA**) is a metric that can be computed for absorbing SPNs by checking “Mean Time to Absorption (failure)”. MTTA is presented after the state probabilities in the “Results” text area.

2.3 SPN Structural Analysis

Mercury provides a feature for analyzing SPNs without generating the reachability graph, but by considering the structure of the model. “Structural Analysis” makes it possible to prove some properties of an SPN model by means of **invariants** and **traps** techniques. It is accessible in menu Evaluate -> SPN Evaluation -> Structural Analysis (see Figure 84).

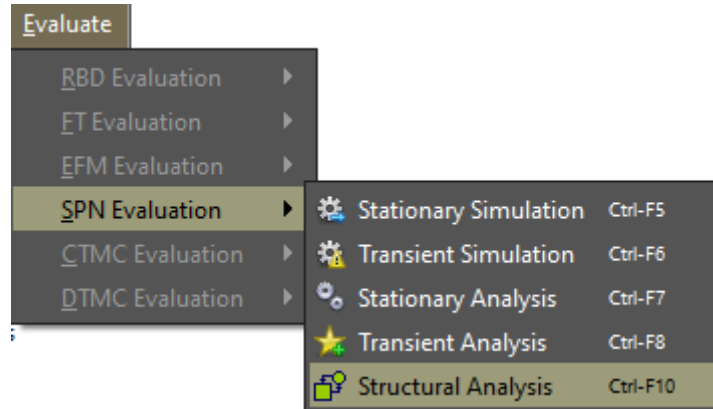


Figure 84: Structural Analysis Function

When the structural analysis is finished, the “Structural Analysis” window is displayed containing various tabs each one having information about the SPN structural properties: **Matrix O (output matrix)**, **Matrix I (input matrix)**, **Matrix C (incidence matrix)**, **Matrix H (inhibitor matrix)**, **Classification**, **Invariant Analysis**, and **Siphons/Traps**. On the same window it is possible to export the result to a plain text file by clicking on button "Save to File".

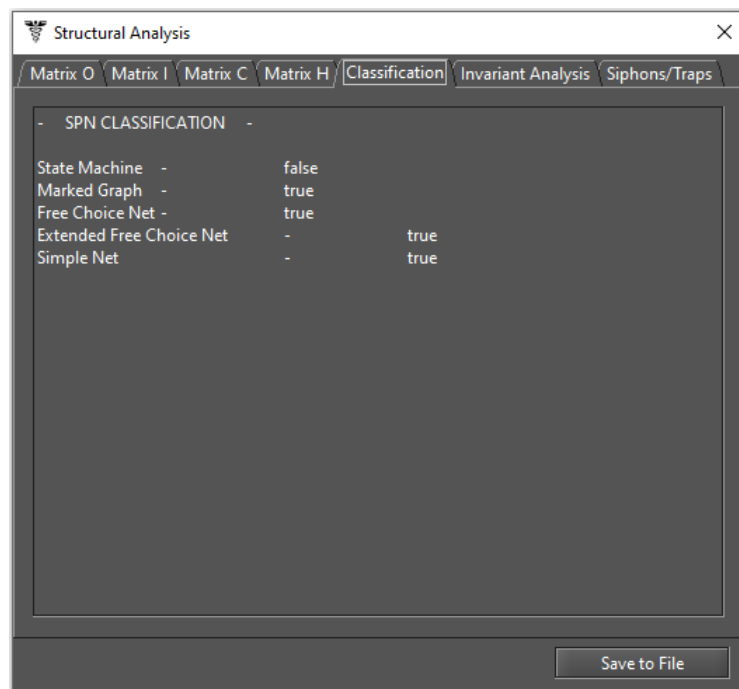


Figure 85: Structural Analysis Window

2.4 Token Game

Token Game lets us to simulate the behavior of SPN models. In other words, users are able to debug the model. Thus, mistakes on the construction phase of that model can be easily discovered and some improvements may be made in order to fix them. Figure 86 shows the model we have adopted as an example.

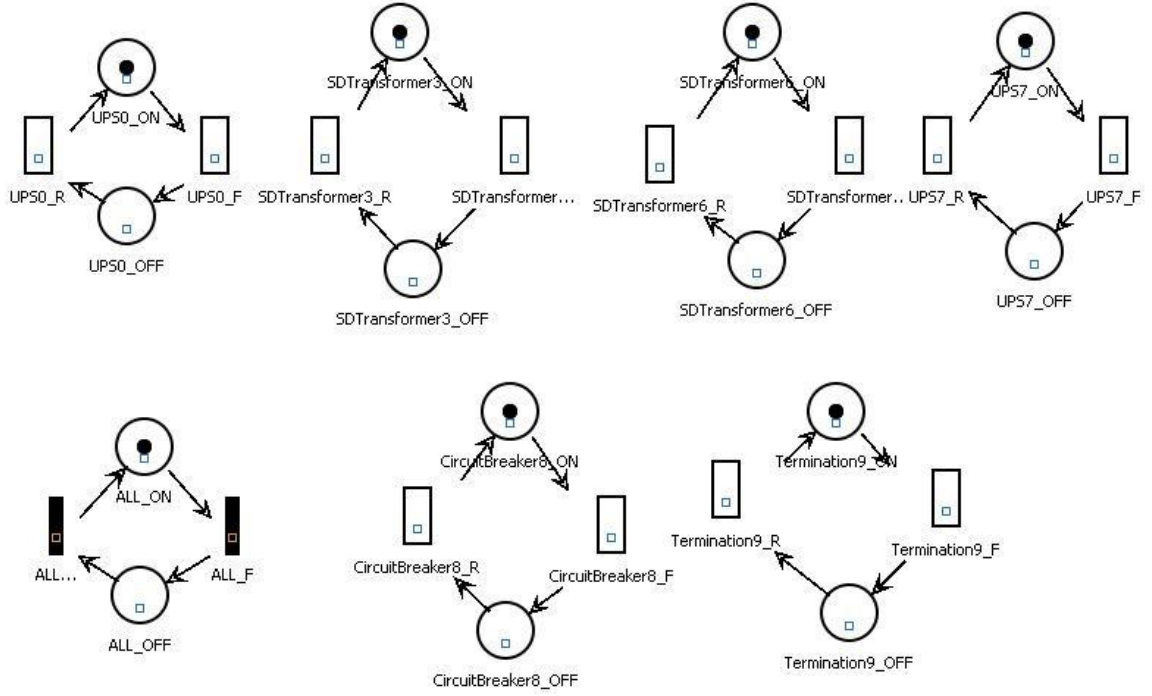


Figure 86: SPN Model

Considering the guard expression defined below, there is a transition *ALL_F* representing the behavior of a system. In other words, when it fires, the mode of that system is changed from operational to failure. The guard expression assigned to that transition (*ALL_F*) is represented as follows.

$$\begin{aligned}
 & ((\#Termination9_ON = 0) OR (\#CircuitBreaker8_ON = 0)) \\
 & OR (((\#UPS0_ON = 0) OR (\#SDTransformer3_ON = 0)) \\
 & AND ((\#UPS7_ON = 0) OR (\#SDTransformer6_ON = 0))))
 \end{aligned}$$

By considering our SPN model and using Token Game, users can simulate equipment failures as well as the corresponding consequences on the availability of a system. In order to turn Token Game on/off, the user must click on button “Token Game” highlighted in Figure 87. Moreover, by observing that figure, we can see any component can fail by the firing of any enabled transition (the green’s ones on it).

By considering the transition “Termination9_F” has been fired (see Figure 88), it means that the termination has failed. So, we can see there is only one transition ready to be fired (*ALL_F*), meaning the system has changed to the failure mode, as expected.

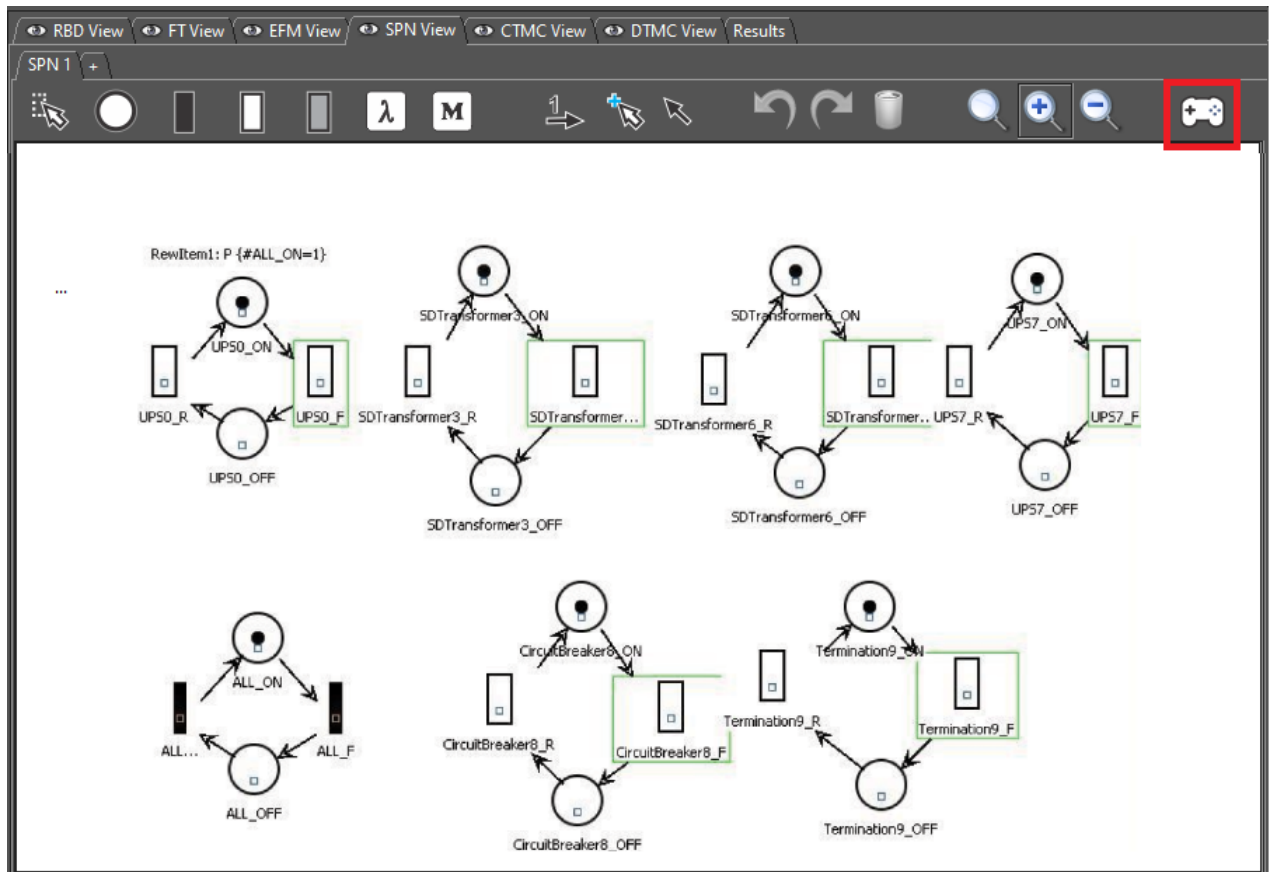


Figure 87: Turning Token Game On

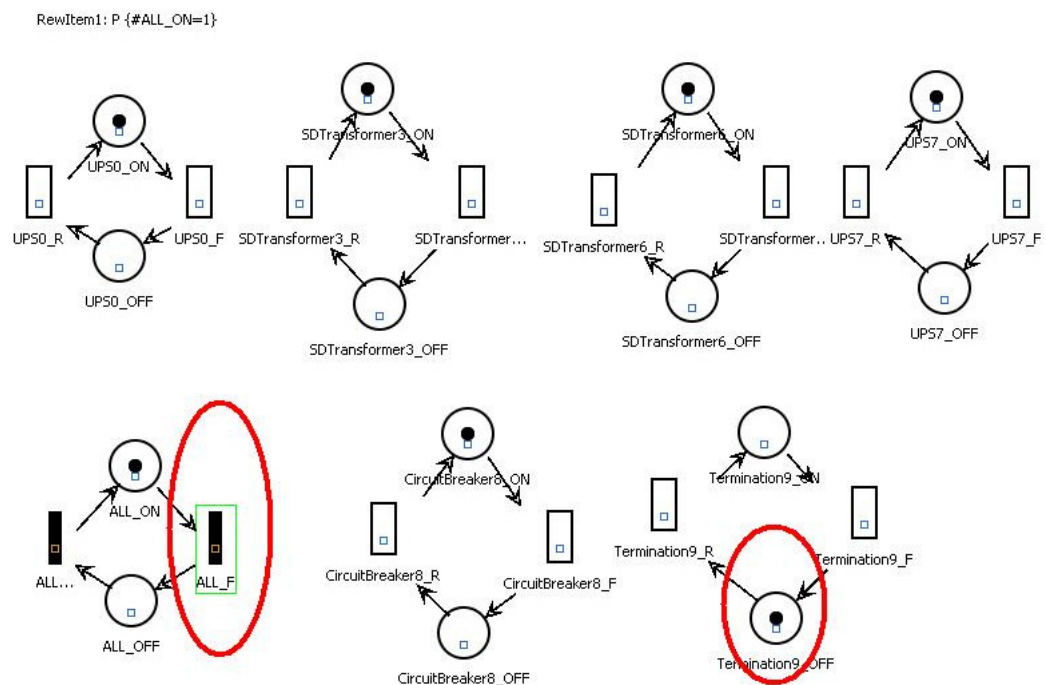


Figure 88: Example of a Token Game

2.5 Sensitivity Analysis

Mercury allows us to perform sensitivity analysis on SPNs computing partial derivative sensitivity indices for them. This analysis is accessible in the menu Tools -> Sensitivity Analysis. Figure 89 shows the window “Sensitivity Analysis”.

Users must select between two sensitivity analysis methods: “Design of Experiments” and “Sensitivity Indices”. The former uses the standard method of analysis of variance to identify the effect of each factor on 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 for an SPN measure with respect to each SPN delay parameter.

| Parameter | Minimum | Maximum | Selected |
|-----------|---------|---------|-------------------------------------|
| avail | 0.1 | 0.9 | <input type="checkbox"/> |
| repo | 2 | 10 | <input checked="" type="checkbox"/> |

| Parameter | Sensitivity index |
|-----------|-----------------------|
| mu | 0.007951274614723074 |
| alpha | 0.007951274614723074 |
| lambda | -0.00795127457115009 |
| beta | -0.007951274562435492 |
| rep | 0.400182555745377104 |

Figure 89: Sensitivity Analysis for an SPN Model

Results of the sensitivity analysis are presented in three possible output formats: “None”, “JFreeChart”, and “R”. Users must select one of those outputs, as depicted in Figure 90. Default option is “None”, which prints results in the text area on the bottom of the window.

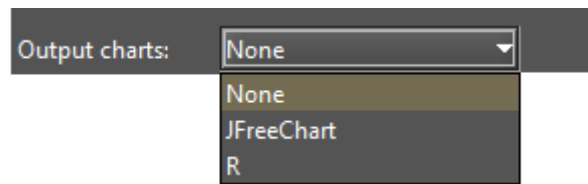


Figure 90: Output Options for Sensitivity Analysis

3 RBD Modeling and Evaluation

Reliability Block Diagram (RBD) is a success-oriented modeling approach for supporting dependability evaluations. By evaluating RBDs, users can see how the failure or success of individual components contributes to the overall reliability and availability. By creating a project, the default RBD model contains only an empty block. Going to the RBD view, the user can see that this default model contains a light gray block named b1 (see Figure 91). When fulfilled by this color, it represents that the properties of that block have not yet been defined. RBD is evaluated from left to right.

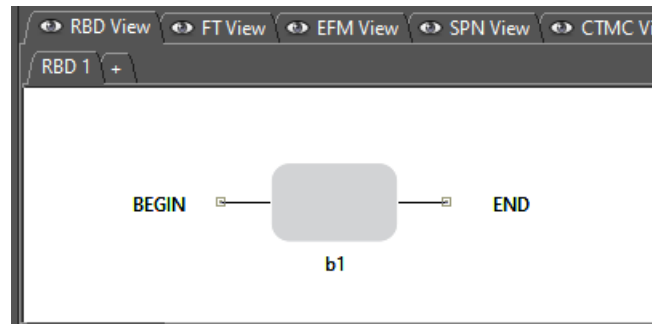


Figure 91: Default RBD Model

As opposed to the other formalisms, the RBD view does not offer a toolbar for the user to select components and make changes into the model. All operations for changing the model are performed by selecting menu items by using the mouse. For example, the user must right-click over the initial block in order to create a block. As we can see by looking at Figure 92, on the pop-up menu some options are available. Changes in the model are performed by selecting the appropriate action on the respective menu item. Among the available options, the user can find the basic operations that are: insert, edit (properties), and remove blocks.

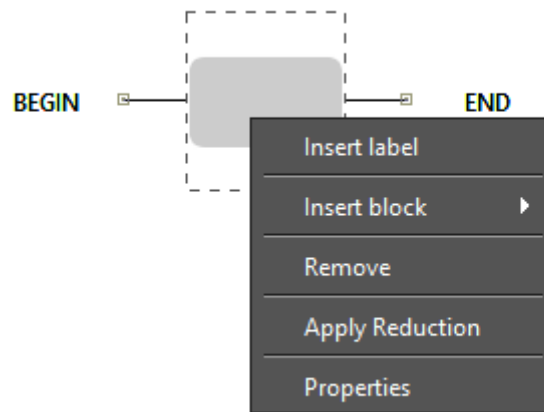


Figure 92: Popup Menu of RBD Blocks

In order to insert a block, the menu “Insert block” must be selected. Depending on how the new block will be connected to the other ones, menu “Series” or “Parallel” must be selected. For each of them, there are two types of blocks that can be created: “Simple Block” and “k-out-of-n Block” (see Figure 93).

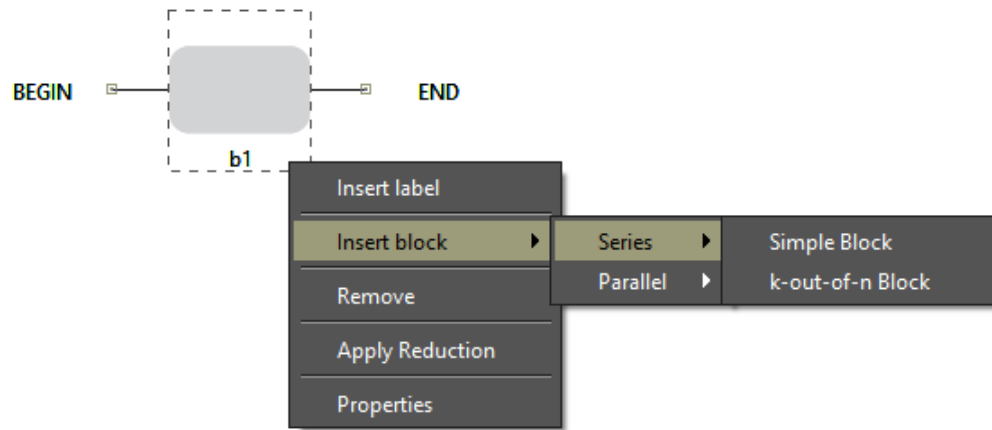



Figure 93: Inserting an RBD Block

By choosing the type of block to be inserted, a window appears for the user to set the properties of the new block(s). Figure 94 shows the dialog presented when inserting a simple block, and Figure 95 shows the dialog presented when inserting a k-out-of-n block. As we can see, the only difference between them is two new fields (K and N) presented when the option “k-out-of-n Block” is chosen. Besides the properties of the block itself, the user must define how many blocks will be inserted at the same time. In this case, all new blocks will have the same properties.

 Insert new Block

Number of Blocks:

1

↑

↓

Description:

TIME

State: Default

Parameters

Failure Distribution: Exponential

Mean value: 0.0

Repair Distribution: Exponential

Mean value: 0.0

Price (\$): 0.00

Insert

Cancel

Figure 94: Inserting a Simple Block

Insert new Block

Number of Blocks: 1

Description:

TIME State: Default

Parameters

Failure Distribution: Exponential

Mean value: 0.0

Repair Distribution: Exponential

Mean value: 0.0

Price (\$): 0.00 K: 1 N: 1

Insert Cancel

Figure 95: Inserting a K-Out-Of-N Block

Once a block has been created, one way to edit its properties is by right-clicking on it and selecting the menu “Properties.” Another way to do that is by double-clicking on it. Also, a third way is available, the user may perform a double-click over the component representation in the top-left-side panel under the icon “RBD Model” (see Figure 96). Figure 97 shows the properties of a simple block already created. At the moment a block is being inserted, the “Block Name” field is not presented for the user as the name for that block is automatically defined by Mercury (see Figure 94). Once it has been created, its name may be changed by accessing its properties. Next, let us do an overview of the properties of blocks.

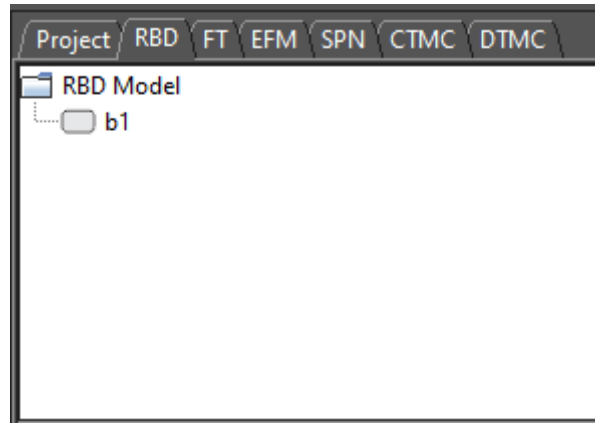


Figure 96: Top-Left-Side Panel of RBDs

- **Number of Blocks.** At the moment a block is being created, the number of blocks to be created is required. The same parameter values will be assigned for all blocks when inserting a value greater than 1.
- **Block Name.** Name for the block. At the moment a block is being created, Mercury defines its name. Once created, the name can be changed by the user by accessing the properties of the block.
- **Description.** A description is additional information about the block itself or the component/subsystem represented by it. It aims to increase the understanding of the model and has no semantic value when evaluating the model. It is just plain text attached to the block.
- **Parameters Type.** Blocks accept four types of parameters: RATES, TIME, AVAILABILITY, and RELIABILITY. Just one of them may be selected at a given moment. The default type is TIME. When parameter type is TIME or RATES, the user can enter the appropriate values for the failure and repair parameters (see Figures 94, 95, and 97). On the other hand, when the type is AVAILABILITY or RELIABILITY, the user can enter the respective value by considering the selected type, as we can see in Figure 98. Considering the context of the last figure, the user must enter the availability of the component represented by the block.
- **State.** State of the block. There are two states available: DEFAULT or FAILED. The default state is DEFAULT what means the block is working properly. On the other hand, when the state is FAILED, it indicates that that block has failed.
- **Failure Parameters.** Mercury supports a large number of probability distributions. Depending on the chosen distribution, fields representing the parameters of the chosen distribution appear in order for the user to enter their values. A label may be attached to each failure parameter. Button "... " allows us to select a label that has already been declared.
- **Repair Parameters.** Fields representing the parameters of the chosen distribution appear for the user to enter their values. A label may be attached to each repair parameter. Button "... " allows us to select a label that has already been declared.

Update Block Parameters

Block Name:

Description:

TIME State:

Parameters

Failure Distribution:

Mean value:

Repair Distribution:

Mean value:

Price (\$):

Figure 97: Properties of Simple Blocks

- **Price.** Cost regarding the component represented by the block. The cost of blocks is considered by evaluating the model through the "component importance and the total cost of acquisition" method. See Section 3.2.4 for further information about this type of evaluation.
- **K and N.** When handling k-out-of-n blocks, two fields appear. A KooN block represents a set of identical components (N) in a single block. All components in this set have the same failure and repair parameters. Using this type of block, users can define how many components at least (K) should be working so that there is no failure. Figure 99 depicts how a KooN block is represented. As we can see, the values of the parameters K and N are presented by side of the block name in the graph.

Figure 100 depicts an RBD having two blocks in series and Figure 101 depicts an RBD having two blocks in

Update Block Parameters

Block Name:

Description:

AVAILABILITY State:

Parameter

| Parameter | Value | ... |
|--------------|-------|-----|
| AVAILABILITY | 0.0 | ... |

Price (\$):

Figure 98: Defining the availability of a Block



Figure 99: An Exponential KooN Block

parallel. The color of the block changes by assigning all parameters required. Evaluations can only be carried out when all required parameters of all blocks are entered.

Mercury has a feature to increase the readability of models. Once the parameters of a block have been assigned, it is possible to read them in the drawing area by positioning the mouse cursor over the block. After

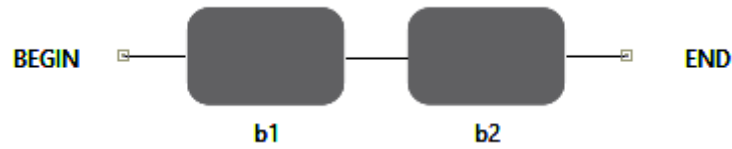


Figure 100: Two Blocks in Series

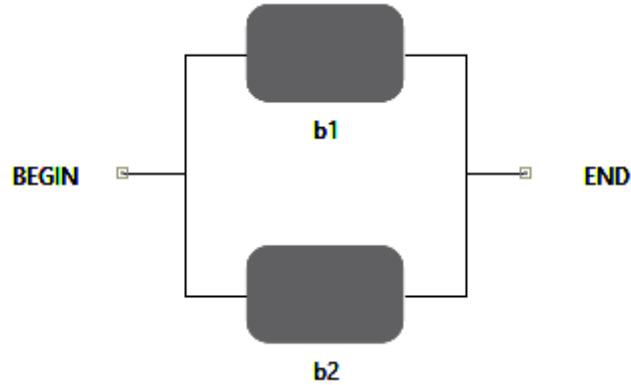


Figure 101: Two Blocks in Parallel

this, a tooltip appears showing all properties of that block. As we can see by looking at Figure 102, all properties appear in the tooltip. All types of components of all formalism supported by Mercury provide this feature.

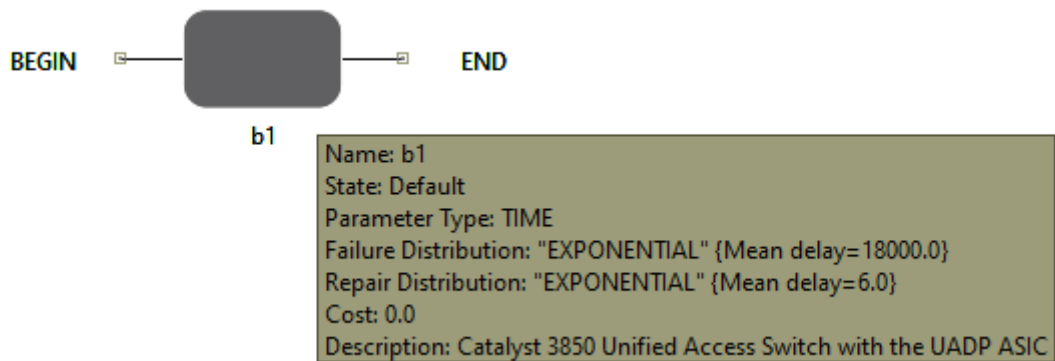


Figure 102: Tooltip for a Block

Finally, let us see the types of blocks and how they are graphically represented. Figure 103 shows the six types of RBD blocks. Figures 103.a and 103.b show blocks with no parameters assigned. Figures 103.c and 103.d demonstrate exponential blocks but in c) the state of the block is defined as "Default" while in d) it is defined as "Failed." Figures 103.e and 103.f demonstrate non-exponential blocks, but in e) the state of the block is defined as "Default" while in f) it is defined as "Failed." As we can see, blocks with no failure/repair parameters are depicted by a light gray block. Exponential blocks are depicted by a dark gray block. Finally, non-exponential blocks are depicted by a blue block.

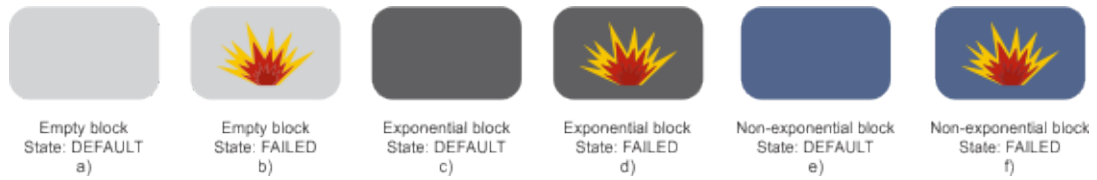


Figure 103: Types of RBD Blocks

3.1 RBD Reduction

In order to reduce the complexity of RBD models, a function is available for supporting a reduction process aiming to reduce the number of blocks. This function can be used until there is only one single block in the model. The model may be reduced from its configuration such as series or parallel. The main objective of this function is to simplify the model itself. However, original parameters of blocks may be lost and only metrics and characteristics of the original model will be preserved.

Figure 104 shows a model before the reduction process is applied, in which there are four blocks. This function is applied by performing a right-click over the block and selecting “Apply Reduction” (see Figure 105). Figure 106 illustrates the model after the reduction has been applied. It is possible to see that the number of blocks was reduced to three (b2 and b3 were reduced to just one block, namely, ssb6).

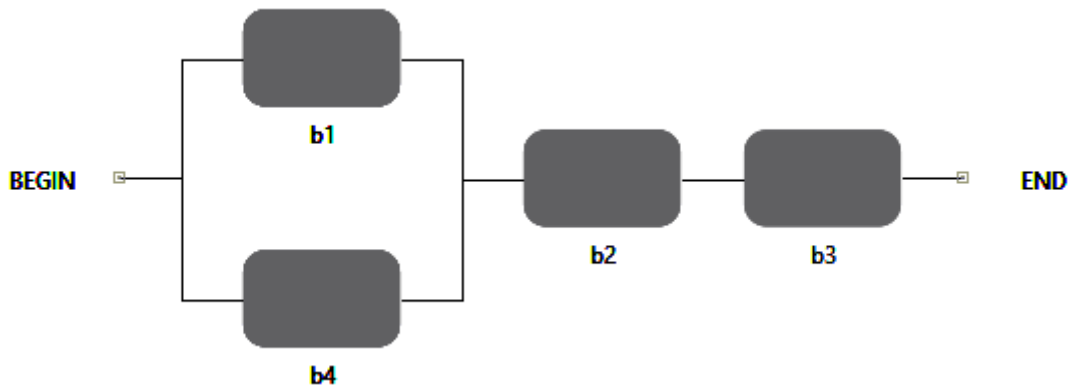


Figure 104: RBD before the series submodel is reduced

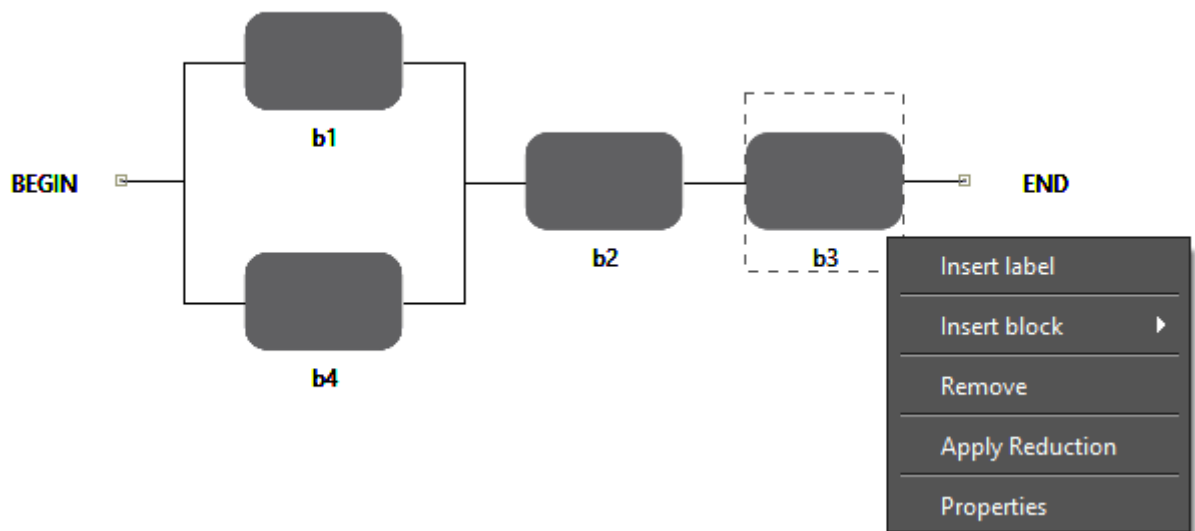


Figure 105: Applying reduction to the RBD

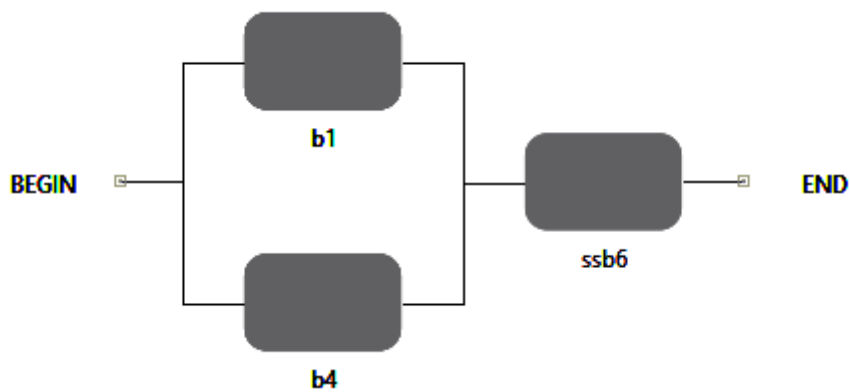


Figure 106: RBD after the series submodel has been reduced

In a series sub-model, the original characteristics of the model are maintained. However, by applying 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. Therefore, when applying a reduction in a parallel submodel, new options are displayed as depicted in Figure 107. In this case, users must choose the metric to be evaluated. If reliability is selected, then the time for reliability estimation is requested, as shown in Figure 108. Figure 109 shows the model after a reduction in the parallel submodel has been applied.

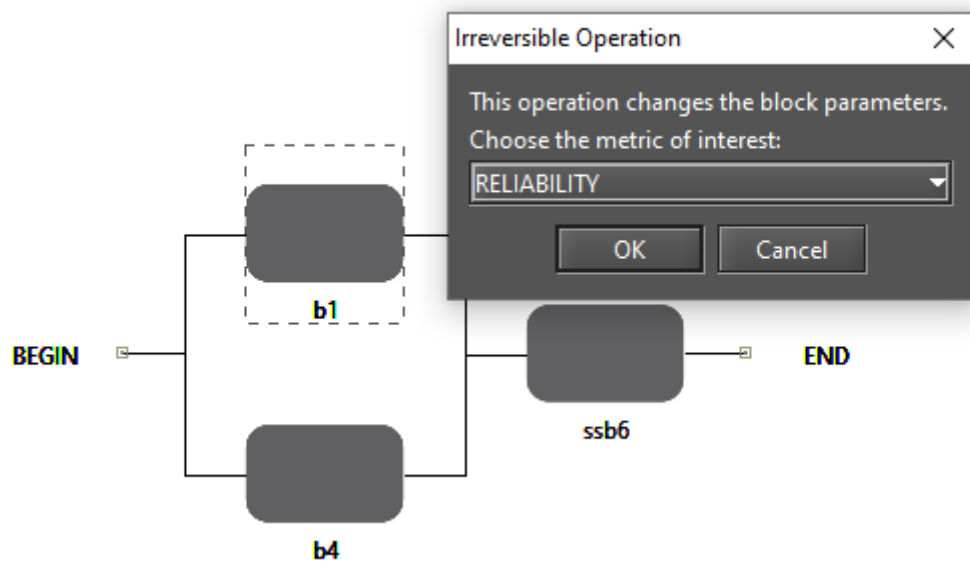


Figure 107: RBD before the parallel submodel is reduced

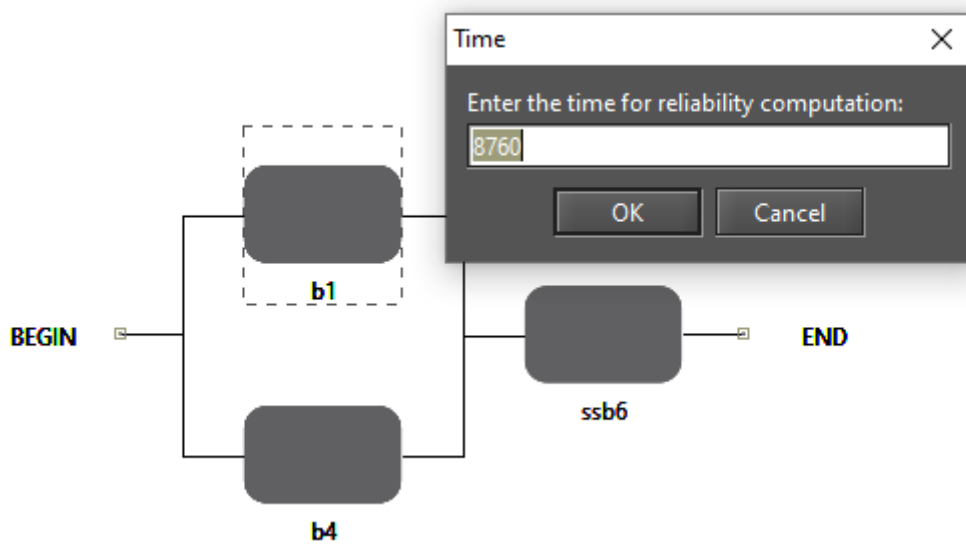


Figure 108: Entering the time for reliability computation

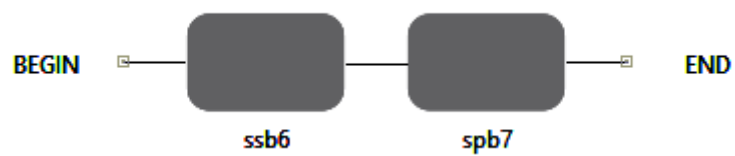


Figure 109: RBD after the parallel submodel has been reduced

3.2 RBD Evaluation

Mercury provides a large number of evaluations for RBDs:

- Exact Evaluation,
- Bounds Evaluation,
- Importance Measures,
- Experiment,
- Get Functions, and
- Sensitivity Analysis.

These evaluations are available in the menu Evaluate -> RBD Evaluation, as depicted in Figure 110. We introduce each one in the next subsections.

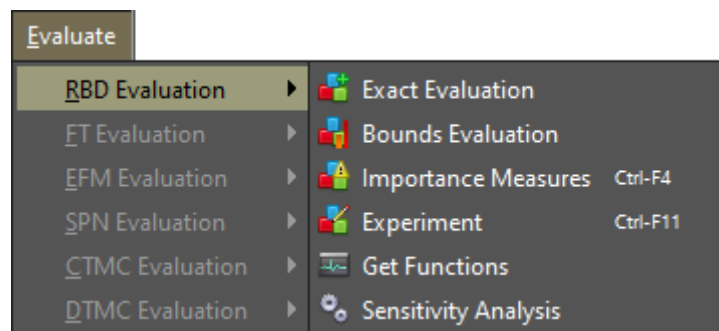


Figure 110: RBD Evaluation Menu

3.2.1 Exact Evaluation

Through the Exact Evaluation a large number of dependability analyses may be performed on RBDs. Exact Evaluation can be accessed in the menu Evaluate -> RBD Evaluation -> Exact Evaluation. Figure 111 shows the window for performing dependability evaluations.

Reliability Analysis of RBD

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

Run Cancel

Figure 111: Exact Evaluation of RBDs

As we can see, users can evaluate eight metrics: mean time to failure, mean time to repair, steady-state availability, instantaneous availability, reliability, unreliability, uptime, and downtime. Besides that, the user can choose the time unit to be considered when computing uptime and downtime: seconds, minutes, hours, and days. When time-dependent metrics are chosen — reliability, unreliability, or instantaneous availability —, the time parameter is required. In addition, there is an option to analyze time-dependent metrics by considering multiple points on time. Time interval goes from 0 to the evaluation time, as the last one is divided by the number of points entered. The metric is computed for each point.

Mercury provides two methods for computing dependability metrics. It is possible to choose between SFM (Method based on Structure-Function) and SDP (Sum of Disjoint Products), as depicted in Figure 112. SFM computes metrics taking into account the structural function of the model. On the other hand, the SDP method, based on Boolean algebra, computes metrics considering minimal cuts and minimum paths.

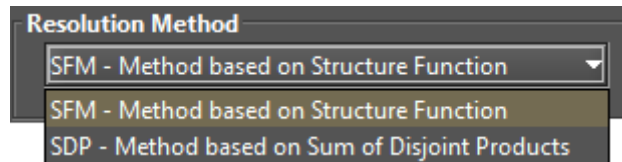


Figure 112: Resolution Method

After choosing the options and entering the evaluation time and number of sampling points, if required, the user must click on the button “Run” in order to Mercury perform evaluations. Once done, a window appears showing the result, as we can see in Figure 113. The results are divided into two groups. These are “Steady-state Results” for steady-state metrics and “Instantaneous Results” for time-dependent metrics. Listing 5 shows a result obtained by evaluating an RBD as an example.

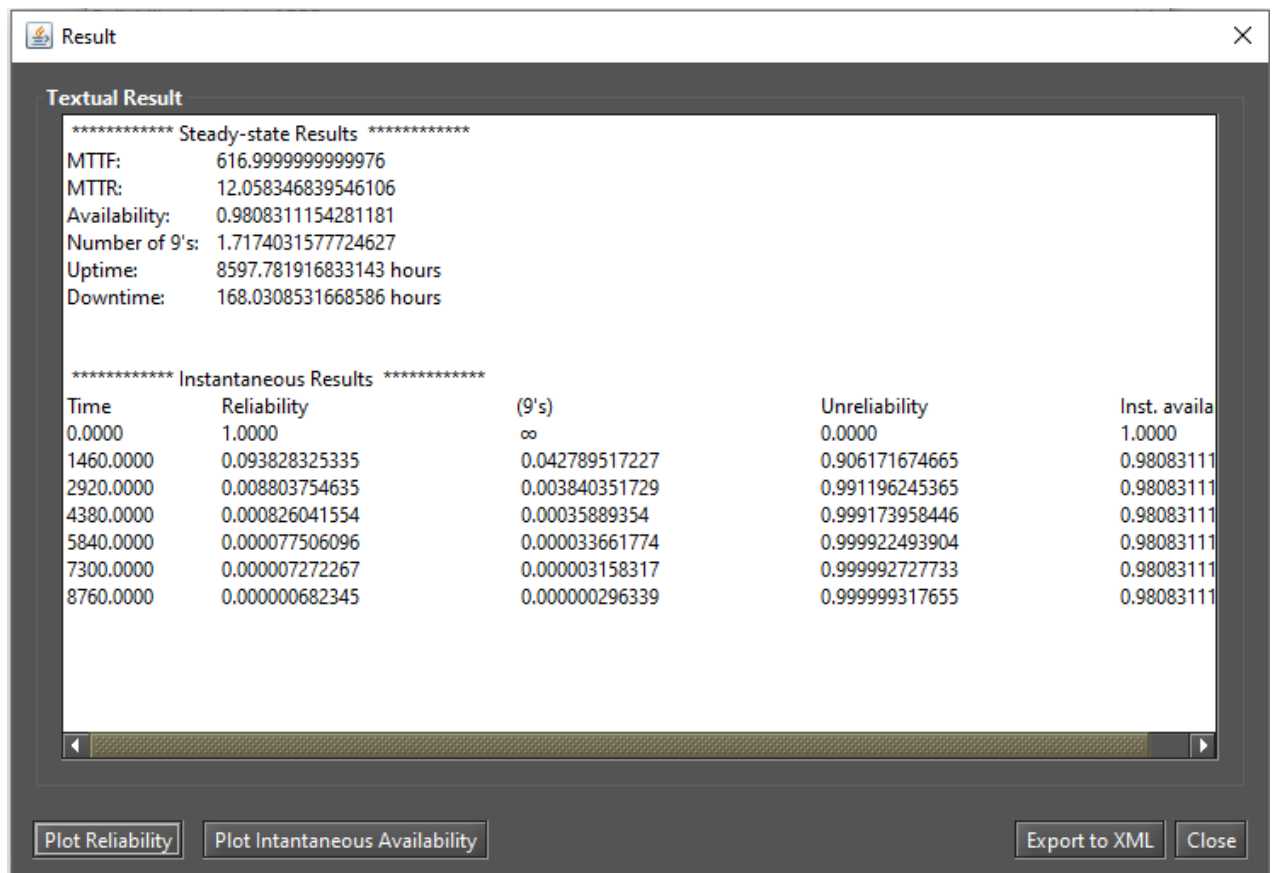


Figure 113: Dependability Results

Listing 5: Dependability Results for an RBD

```

***** Steady-state Results *****

MTTF:                616.9999999999976
MTTR:                12.058346839546106
Availability:        0.9808311154281181
Number of 9's:      1.717403157724627

```

| | | | |
|-----------------------------------|-------------------------|----------------|----------------|
| Uptime: | 8597.781916833143 hours | | |
| Downtime: | 168.0308531668586 hours | | |
| ***** Instantaneous Results ***** | | | |
| Time | Reliability | (9's) | Unreliability |
| 0.0000 | 1.0000 | infinity | 0.0000 |
| 1460.0000 | 0.093828325335 | 0.042789517227 | 0.906171674665 |
| 2920.0000 | 0.008803754635 | 0.003840351729 | 0.991196245365 |
| 4380.0000 | 0.000826041554 | 0.00035889354 | 0.999173958446 |
| 5840.0000 | 0.000077506096 | 0.000033661774 | 0.999922493904 |
| 7300.0000 | 0.000007272267 | 0.000003158317 | 0.999992727733 |
| 8760.0000 | 0.000000682345 | 0.000000296339 | 0.999999317655 |

Figure 114 shows the “Reliability Chart” dialog displayed by clicking on the button “Plot Reliability.” Figure 115 shows the “Instantaneous Availability Chart” dialog displayed by clicking on the button “Plot Instantaneous Availability.” Both buttons are only visible when the respective reliability metric is selected on the input dialog. The number of points on the plotted lines is determined by the number of points entered by the user.

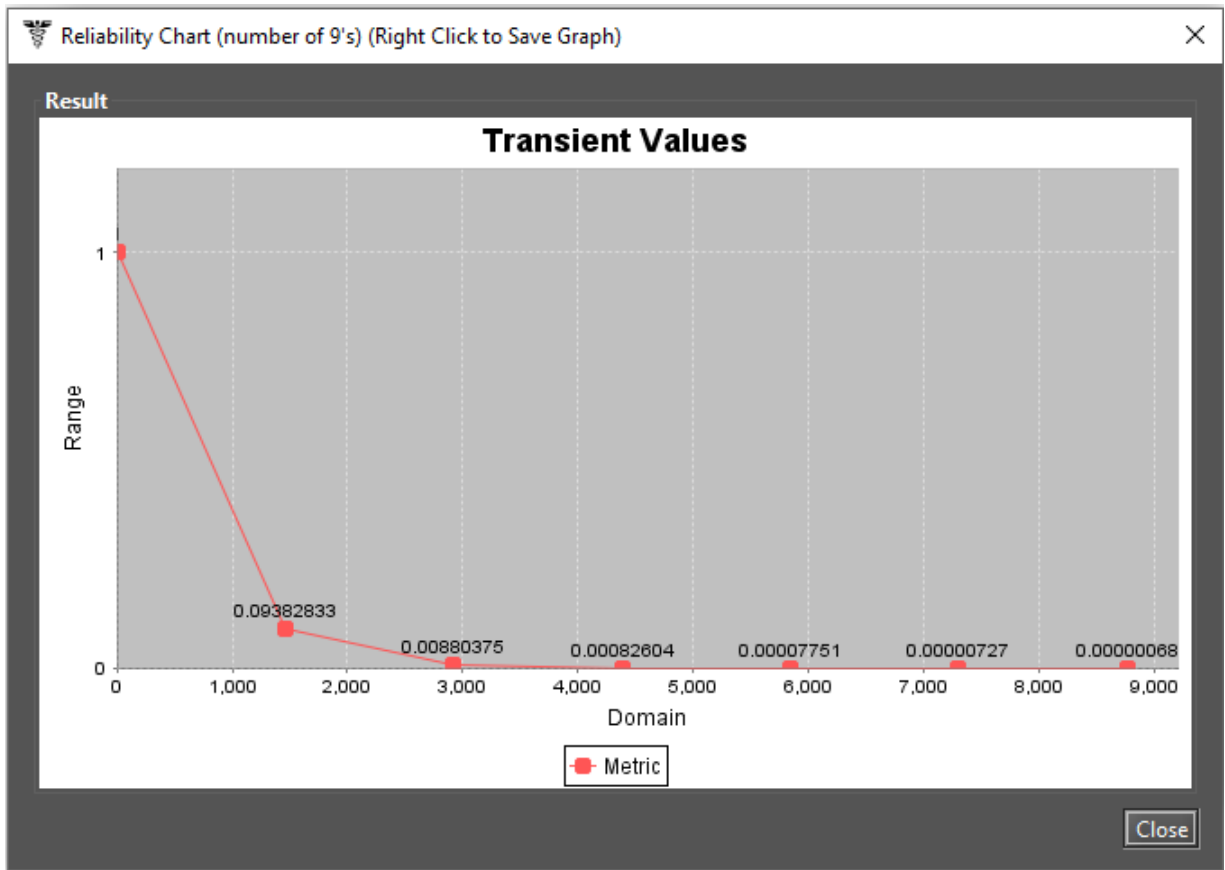


Figure 114: Reliability Chart

Resolutions of models through simulation are required when non-exponential probability distributions are associated to some block. In this case, these non-exponential blocks are converted into SPNs and the model is solved through simulation. Two resolution methods are available: *SFM - Simulation* and *SDP - Simulation*. By detecting this situation, Mercury displays the message "Non-exponential distributions detected" at the bottom of the Reliability Analysis screen (see Figure 116).

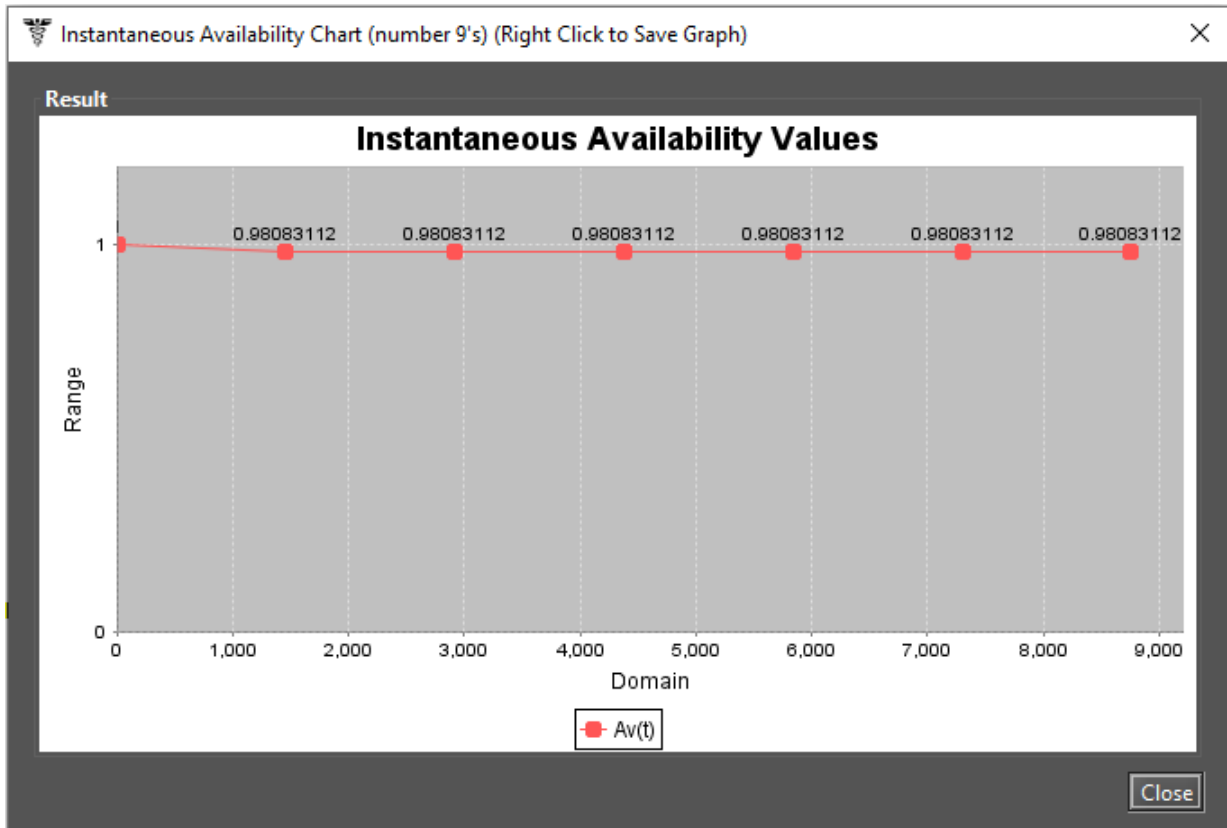


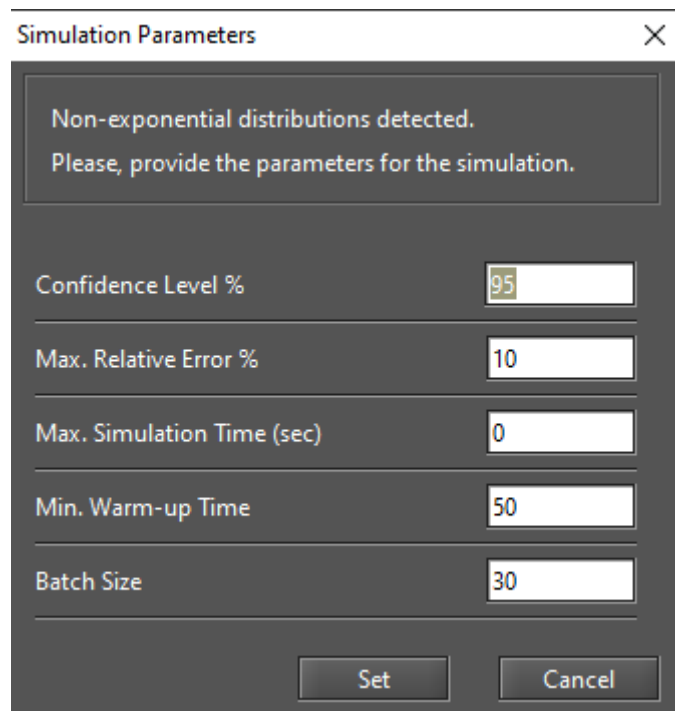
Figure 115: Instantaneous Availability Chart

The figure shows a software window titled "Reliability Analysis of RBD". It contains several configuration options for a simulation:

- Resolution Method:** A dropdown menu with "SFM - Simulation" selected. A secondary dropdown shows "SFM - Simulation" and "SDP - Simulation".
- Analysis Options:**
 - ☐ Mean Time to Failure
 - ☐ Mean Time to Repair
 - ☒ Uptime
 - ☒ Steady-State Availability
 - ☐ Instantaneous Availability
 - ☒ Downtime
 - ☒ Reliability
 - ☒ Unreliability
- Time unit:** A dropdown menu set to "hours".
- Evaluation Time:** An empty text input field.
- Analyze in multiple time points:** An unchecked checkbox.
- Number of sampling points:** An empty text input field.
- Message:** "Non-exponential distributions detected." in red text.
- Buttons:** "Run" and "Cancel" buttons.

Figure 116: RBD Analysis through Simulation

By clicking on the button “Run”, some parameters must be entered for supporting the simulation. The required parameters depend on the chosen metrics. By selecting only steady-state metrics, Mercury shows the window depicted in Figure 117. On the other hand, when selecting one or more time-dependent metrics, Mercury shows the window depicted in Figure 118. Results are presented once the parameters have been defined and the simulation has been finished. It is important to highlight that the following metrics cannot be solved by performing simulation: MTTE, MTTR, and Instantaneous Availability. For further information about simulations, please see Section 2.1.



The image shows a 'Simulation Parameters' dialog box with a close button (X) in the top right corner. Inside the dialog, there is a message box that says 'Non-exponential distributions detected. Please, provide the parameters for the simulation.' Below this message, there are five input fields with labels and values: 'Confidence Level %' with value '95', 'Max. Relative Error %' with value '10', 'Max. Simulation Time (sec)' with value '0', 'Min. Warm-up Time' with value '50', and 'Batch Size' with value '30'. At the bottom of the dialog, there are two buttons: 'Set' and 'Cancel'.

| Parameter | Value |
|----------------------------|-------|
| Confidence Level % | 95 |
| Max. Relative Error % | 10 |
| Max. Simulation Time (sec) | 0 |
| Min. Warm-up Time | 50 |
| Batch Size | 30 |

Figure 117: Simulation for Steady-State Metrics

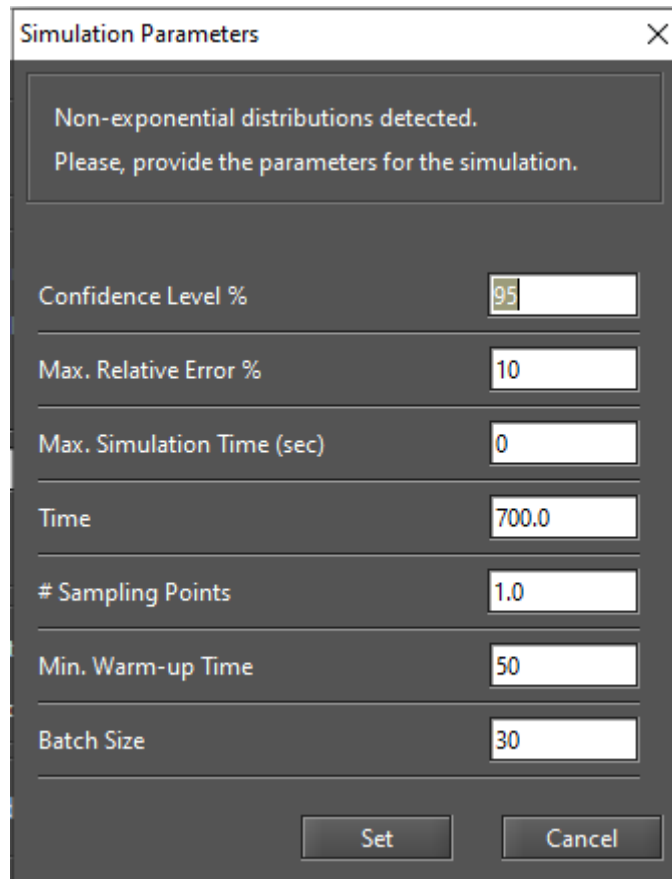
A screenshot of a 'Simulation Parameters' dialog box. At the top, a message box states: 'Non-exponential distributions detected. Please, provide the parameters for the simulation.' Below this, there are seven input fields with corresponding labels: 'Confidence Level %' (value: 95), 'Max. Relative Error %' (value: 10), 'Max. Simulation Time (sec)' (value: 0), 'Time' (value: 700.0), '# Sampling Points' (value: 1.0), 'Min. Warm-up Time' (value: 50), and 'Batch Size' (value: 30). At the bottom right, there are two buttons: 'Set' and 'Cancel'.

Figure 118: Simulation for Time-Dependent Metrics

3.2.2 RBD Experiment

Mercury allows us to evaluate the impact of varying some parameters on the model. Now, we demonstrate how to use the experiment function.

The first step is to define one or more labels. Labels are variables that store numeric values and can be attached to the failure/repair parameters of blocks. The value of a label is changed considering a stepsize and at each change, the selected metric is evaluated. A label is inserted by right-clicking on an RBD block and selecting “Insert label”, as depicted in Figure 119. Another way to do that is by right-clicking on the label area in the left-side panel and selecting “Insert label.”

Once this is done, the “Label Properties” window is shown (see Figure 120). There, the user can set the properties of the label.

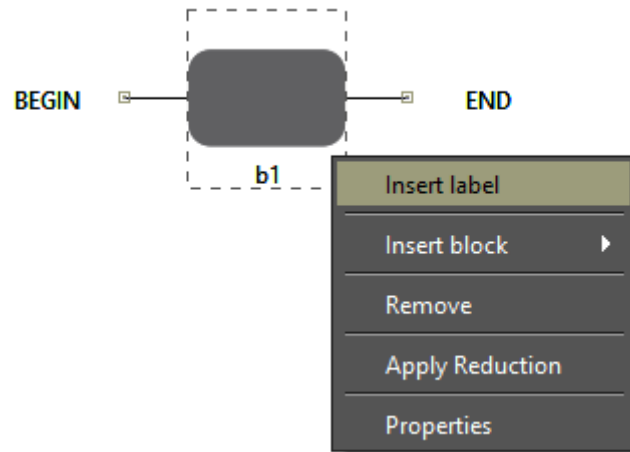


Figure 119: Inserting a Label into the RBD Model

The 'Label Properties' dialog box is shown with a title bar containing a small icon and the text 'Label Properties'. The main area is titled 'Properties' and contains three input fields: 'Name:' (empty), 'Value:' (containing '0.0'), and 'Description:' (a large empty text area). At the bottom right, there are 'OK' and 'Cancel' buttons.

Figure 120: Properties of an RBD Label

Once a label has been inserted, it is available in the left-side panel on the RBD tab. Now, it can be associated with one or more block parameter (see Figure 121).

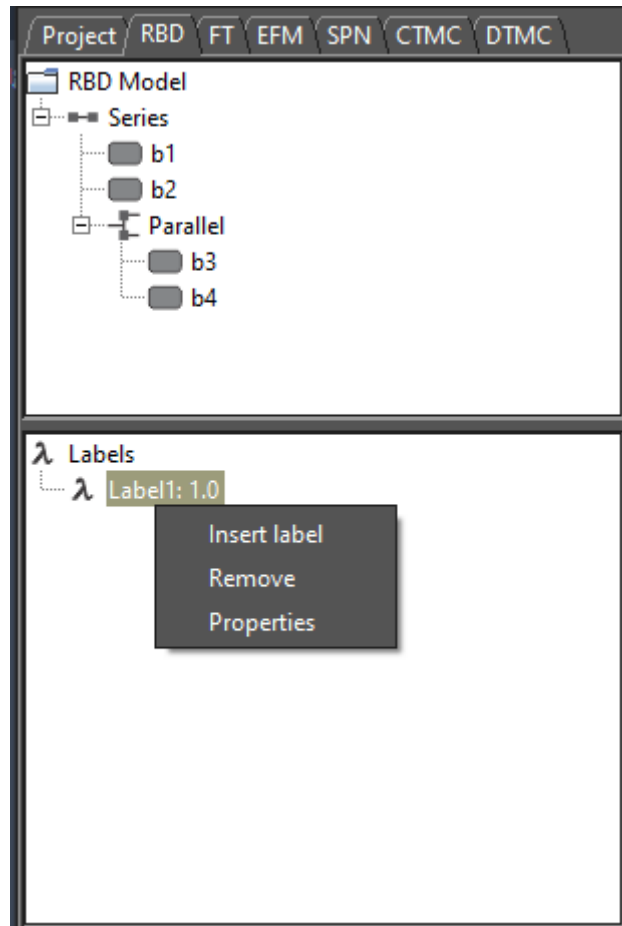


Figure 121: RBD Panel in the Left-Side

By right-clicking on any label, a pop-up menu with three menu items appears (see Figure 121). Below, we describe each of them.

- **Insert label.** Show the window “Label Properties” allowing the user to insert a label.
- **Remove label.** Remove the selected label.
- **Properties.** Show the window “Label Properties” allowing the user to change the properties of the label.

After defining a label, it is necessary to attach this label to the failure/repair parameter of the block under evaluation. Figure 122 demonstrates how to attach a label to a block parameter.

Update Block Parameters

Block Name:

Description:

TIME: State:

Parameters

Failure Distribution:

Mean value:

Repair Distribution:

Mean value:

Price (\$):

Figure 122: Attaching a Label to a Block Parameter

Experiment of RBDs can be accessed in the menu Evaluate -> RBD Evaluation -> Experiment. Figure 123 shows the “RBD Experiment” window. In order to perform experiments, the user must enter values for all fields required.

Below, we describe each of them.

- **Parameter.** The label to have its value changed at each experiment iteration.
- **Metric.** Metric to be evaluated.
- **Minimum Value.** Initial value for the selected label.
- **Maximum Value.** Final value for the selected label.

The image shows a software dialog box titled "Experiment of RBD". It has a dark gray background and a light gray border. At the top right is a close button (X). The dialog contains several input fields and two buttons at the bottom right.

| Field | Value |
|------------------|-------------|
| Parameter: | Label1: 1.0 |
| Metric: | Reliability |
| Minimal Value: | 1000 |
| Maximum Value: | 10000 |
| Interval: | 1000 |
| Evaluation Time: | 7000 |

Buttons: Run Experiment, Cancel

Figure 123: RBD Experiment

- **Interval.** Step-size to be considered for changing the value of the label. The label starts with the minimum value and its value is increased considering this interval. At each change, the selected metric is evaluated. The experiment finishes when the maximum value for the label is reached.
- **Evaluation Time.** Evaluation time to be considered for computing time-dependent metrics. For time-dependent metrics — reliability, unreliability, instantaneous availability — it is required to enter the time parameter.

After defining the input parameters, the user must click on the button “Experiment” in order to start the experiment. Once the experiment has been finished, the “Experiment Result” dialog is shown (see Figure 124).

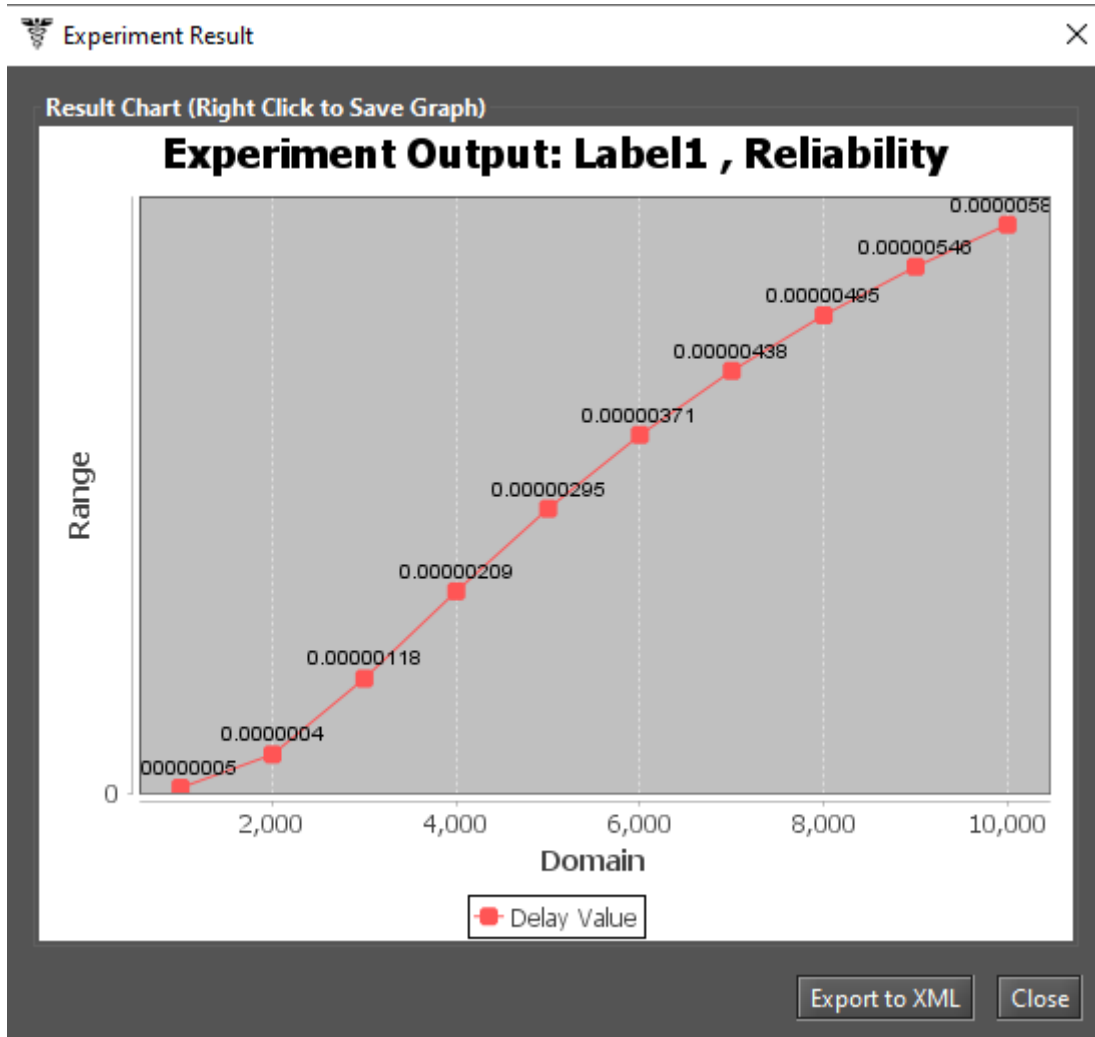


Figure 124: Experiment Result Dialog

3.2.3 Bounds for Dependability Analysis

Bounds for Dependability Analysis is used to estimate dependability metrics through the calculation of reliability, availability, or downtime. Therefore, it is necessary to estimate the bounds values (upper and lower limits) for computing this analysis, in which users quickly obtain results. This analysis should be performed when the model is huge. This analysis is subdivided into 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 required.

Users can access this analysis going to the menu Evaluate -> RBD Evaluation -> Bounds Evaluation. Figure 125 shows the “Bounds for Dependability Analysis” window. As we can see in Figure 126, it is possible to evaluate four metrics: steady-state availability, instantaneous availability, reliability, and downtime. The time parameter is required when choosing time-dependent metrics. Once a metric has been selected and the time entered if required, the user must click on the “Get Start Values” button in order to start the evaluation.

First, it is necessary to compute 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 is chosen to guarantee the operational mode of the system. Meanwhile,

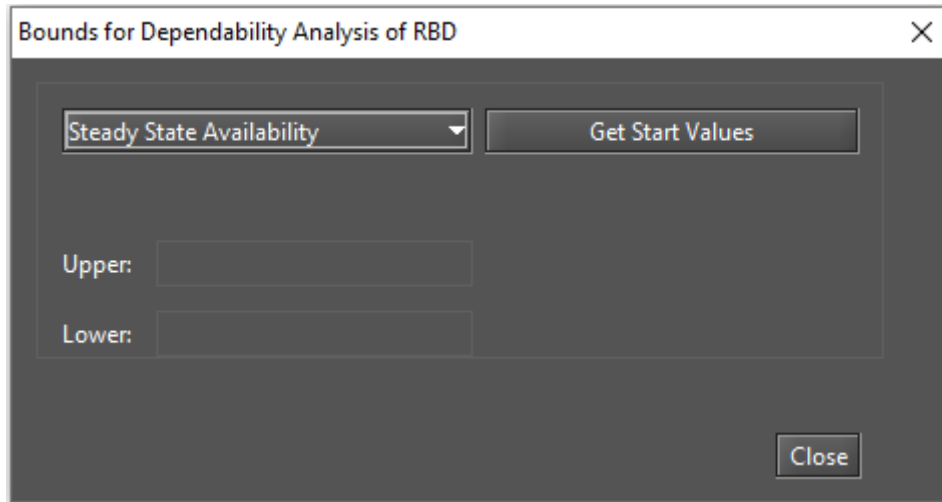


Figure 125: Bounds for Dependability Analysis

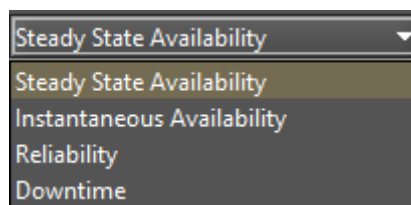


Figure 126: Metrics for Bounds Evaluation

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. After getting the upper and lower values, the user can define the number of steps for the upper and lower values and click on the button “Run” (see Figure 127). Once done, the user can see the result as demonstrated in Figure 128. The user can plot a chart and export the result to an MS Excel file.

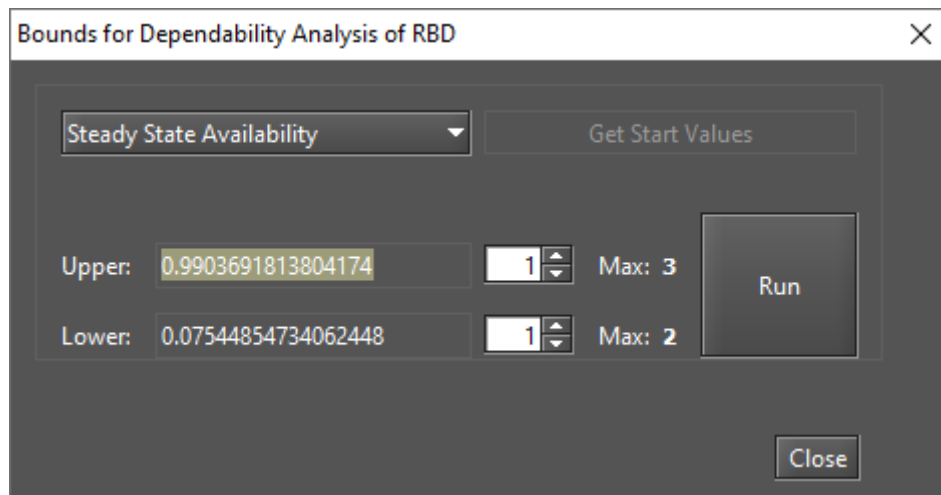


Figure 127: Bounds for Steady-State Availability Metric

The method adopted to find successive values and the number of iterations 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. Once the computation for the last path or last cut has been finished, the exact value for the metric can be

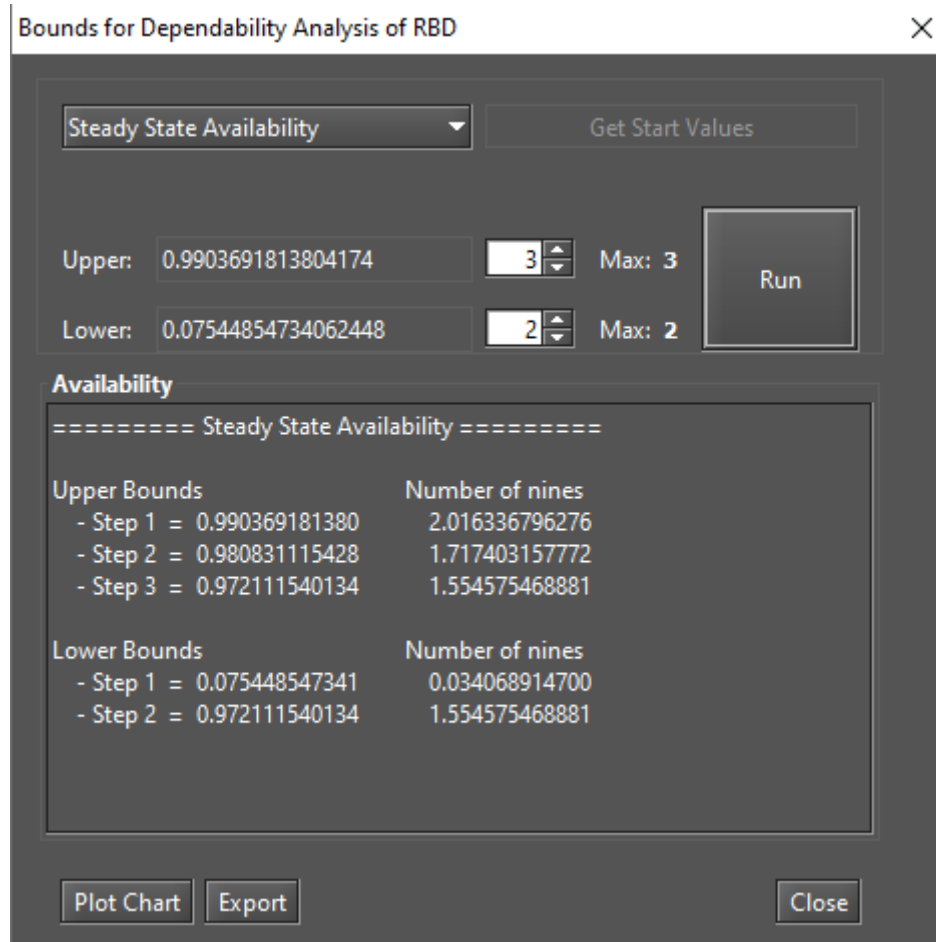


Figure 128: Bounds for Steady-State Availability Metric - Steps Result

found. The exact value will be the value found in the last step.

3.2.4 Component Importance and Total Cost of Acquisition

Component importance is a metric that indicates the impact of a particular component on the system. By considering importance values, the most important component (i.e., that one with the highest importance) should be improved in order to increase the reliability or availability of the system. This evaluation may be applied for example to assist in maintenance actions.

Importance measures make it possible to identify the relative importance of each component with respect to the overall system reliability or availability. Users can access this evaluation by going to the menu Evaluate -> RBD Evaluation and select "Importance Measures." After that, users must select one metric on the "Component Importance Measures" window, after which click on the button "Evaluate" (see Figure 129). If the parameter "Cost" has been set, it is also possible to evaluate the relationship between metrics and investment costs. The time parameter is required in order to evaluate reliability metrics.

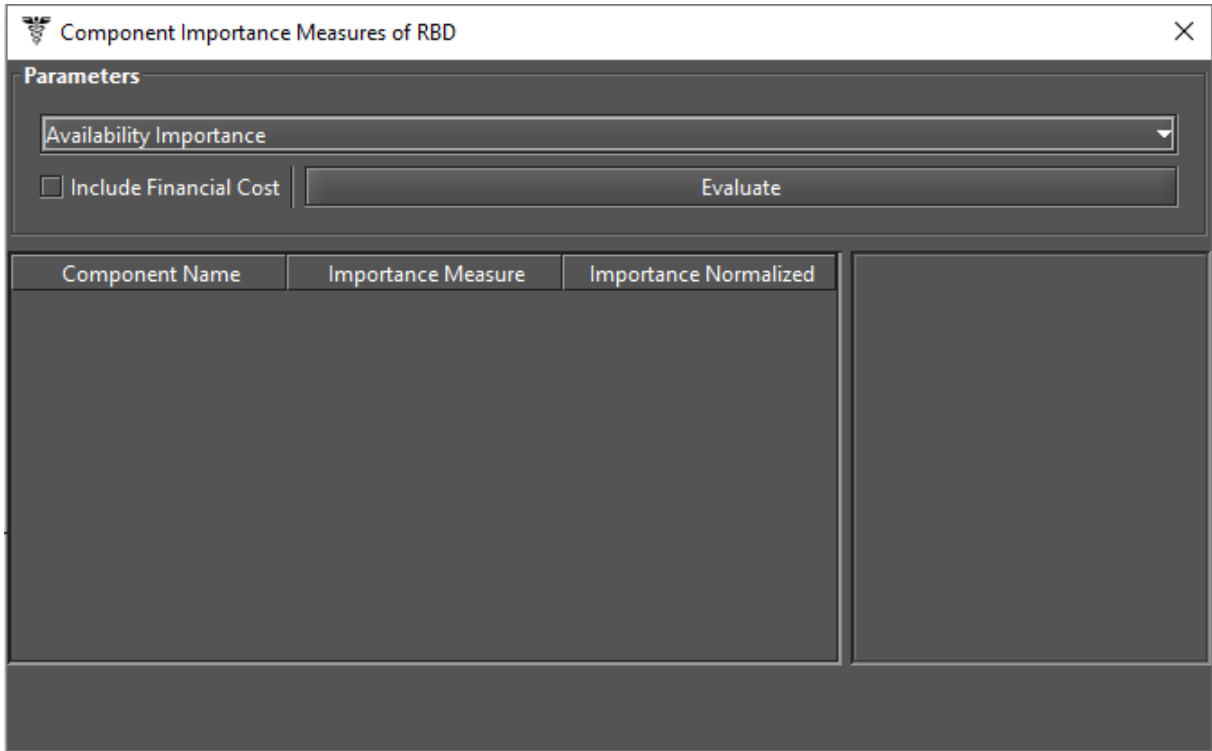


Figure 129: Component Importance Measures

As shown in Figure 130, users can choose between some types of measures. The types are: "Availability Importance", "Reliability Importance (Birnbbaum)", and "Criticality (Reliability or Availability) Importance (CRI)". The last one can be obtained by considering the system in failure "(f)" or working.

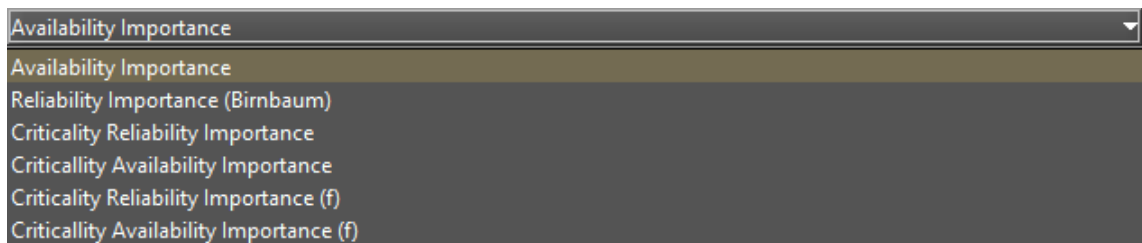


Figure 130: Component Importance Measures - Metrics of Interest

Results of evaluations like this are shown in Figure 131. The results show the importance value for each component and a graphical view as a ranking highlighting those most significant in the analysis.

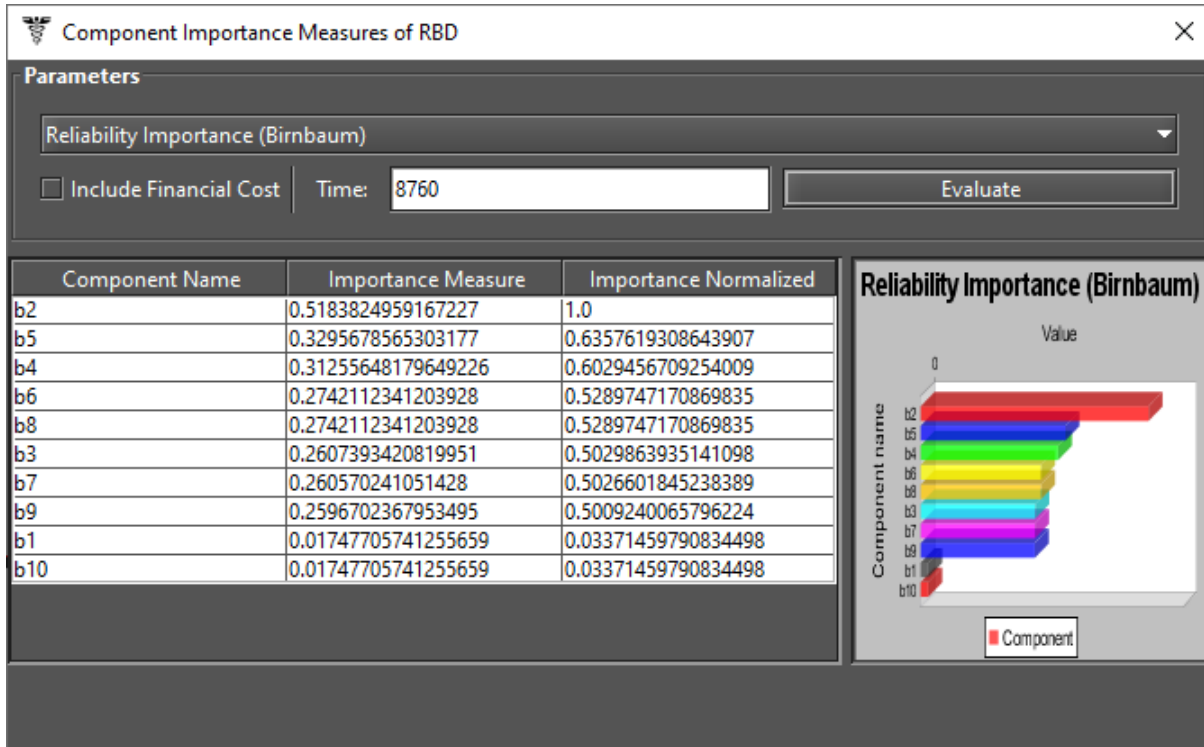


Figure 131: Result of Reliability Importance Evaluation

3.2.5 Structural and Logical Functions

Mercury generates structural and logical functions of RBD models. Both functions are a behavioral representation of the system represented by the model and are related to the states of each component. By applying them, it is possible to evaluate the impact on the system considering the components that have failed.

The system and its components must be in one of the following states: working (default) or failed. The state of the system is a binary random variable determined by the states of its components. If the state of each component is known, then the state of the system is also known.

The state can be toggled by accessing the properties of the component (see Figure 132). When the state of a block is failed, the component is depicted by a fire icon over the block, as already mentioned in this manual.

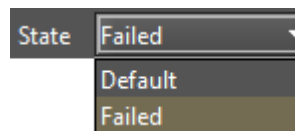


Figure 132: Block State

Now, let us demonstrate how to get these functions by using Mercury. Figure 133 shows an RBD model having blocks in series and in parallel. As we can see, there is a failed block in this model (block b5).

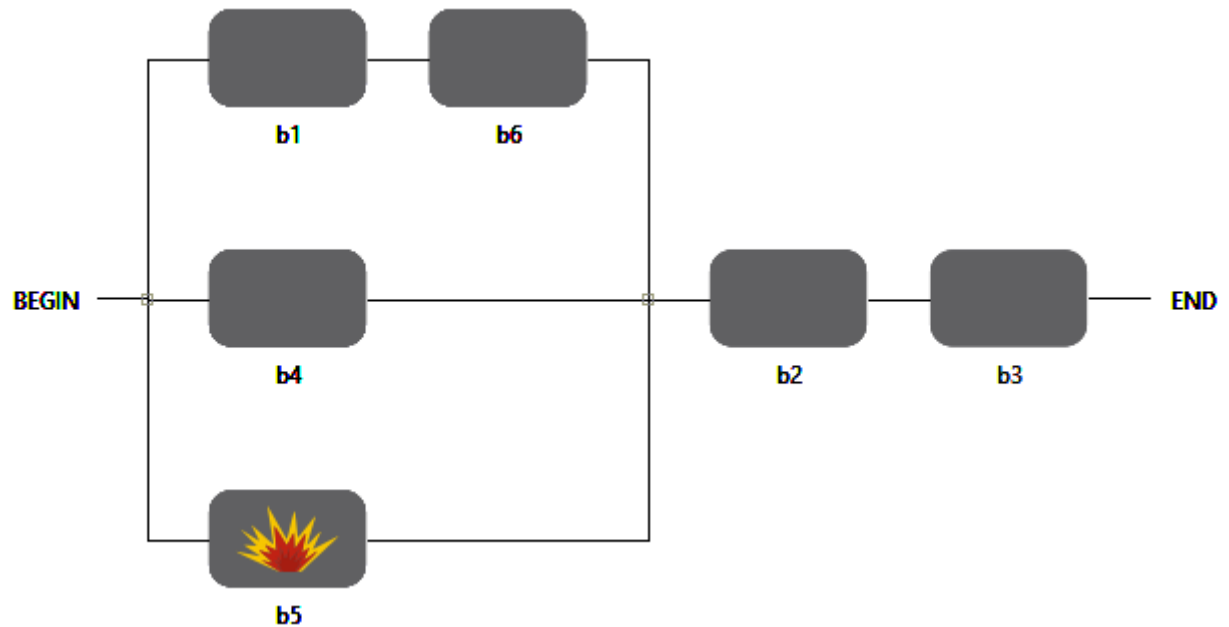


Figure 133: RBD Model for Structural and Logic Function Computation

The structural and logical functions can be accessed in the menu Evaluate -> RBD Evaluation -> Get Functions. Figures 134 and 135 show the structural and logic functions of the RBD model depicted above, respectively. In addition to the expressions, the tool indicates the blocks marked as faulty (inoperative) and the current state of the system. By considering the example, the failed block (b5) does not impact the state of the system.

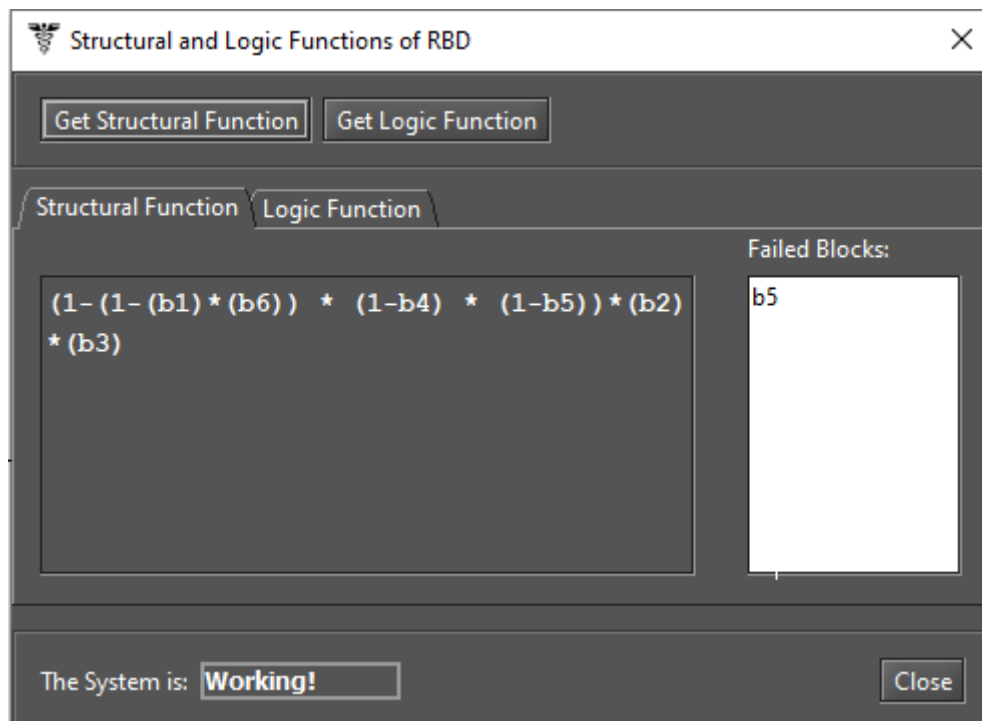


Figure 134: Structural Function

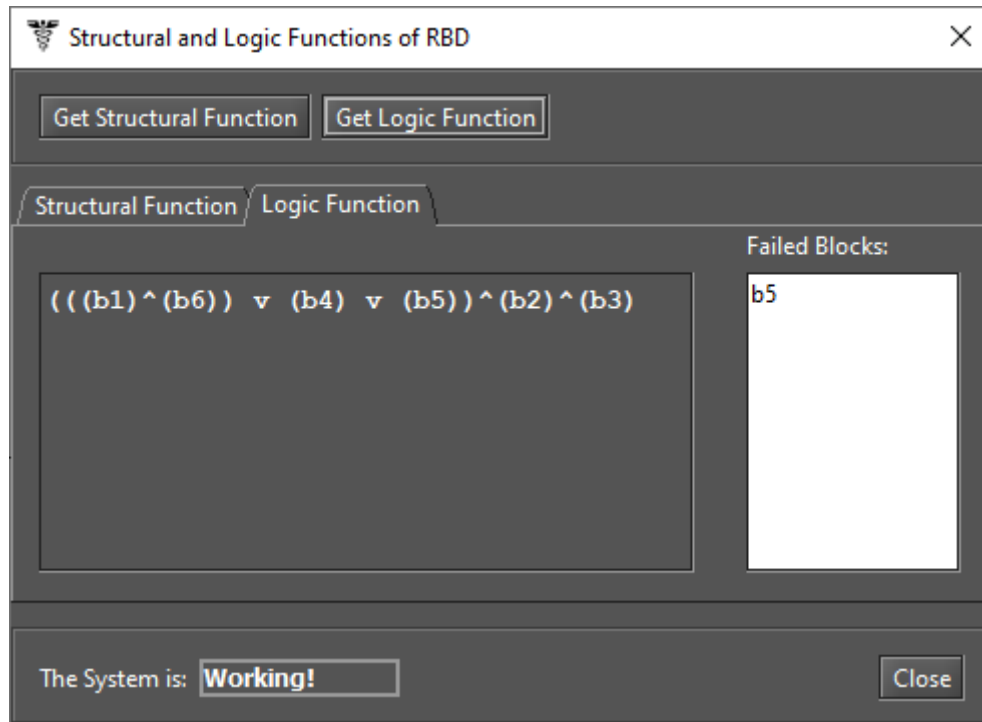


Figure 135: Logic Function

3.2.6 Sensitivity Analysis

Mercury computes partial derivative sensitivity indices of RBDs through sensitivity analysis. Those indices indicate the impact that every input parameter has on the availability of the model. Sensitivity analysis can be accessed in the menu Evaluate -> RBD Evaluation -> Sensitivity Analysis. Figure 136 shows the “RBD Sensitivity Analysis” window, which shows the partial derivative of the structural equation for each parameter, as well as the sensitivity indices. Users can use the sensitivity analysis only if the model has only exponential blocks.

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

| Parameter | Sensitivity value |
|-----------|-----------------------|
| MTTRb1 | -0.009630818619582666 |
| MTTRb2 | -0.009630818619582666 |
| MTTFb1 | 0.009630818619582386 |

Run
Close

Figure 136: Sensitivity Analysis of an RBD Model

4 FT Modeling and Evaluation

Fault Trees (FTs) and RBDs differ from each other in their purposes. FT is a top-down logical diagram and it makes possible to create a visual representation of a system showing the logical relationships between associated events and causes lead that may lead the evaluated system to a failure state. By creating a project, the default FT model contains only an empty top event FAILURE named "undefined" as shown in Figure 137. Which means that no event leads to it. Thus, starting from this point, the user can define the model by using components.

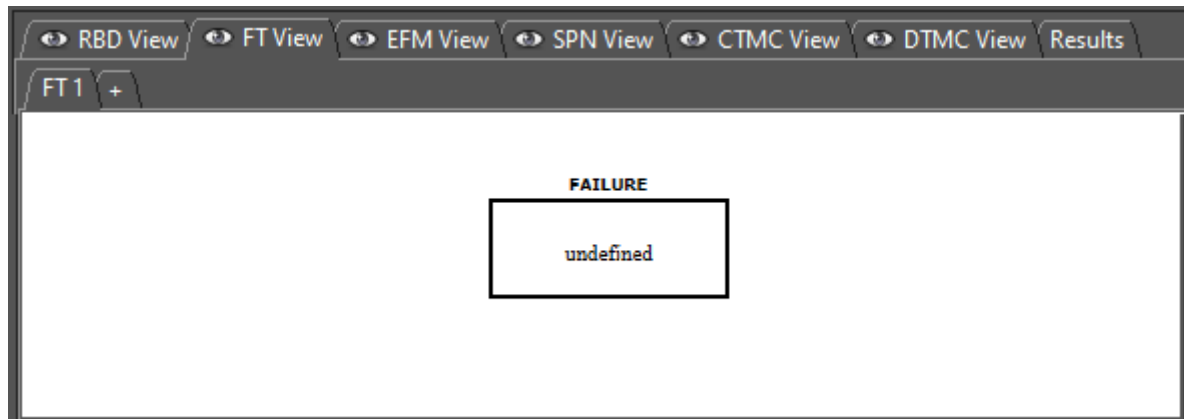


Figure 137: Default Fault Tree Model

By using the Mercury tool, we can handle two types of nodes: basic events and gates (logic ports). Basic events are represented as leaf nodes, as depicted in Figure 138. On the other hand, each supported gate has its own graphical representation. As we can see by looking at Figure 139, Mercury supports three types of gates: AND, OR, and K-out-of-N (KooN).

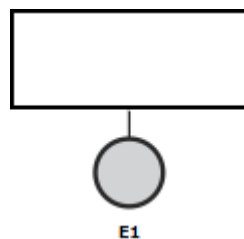


Figure 138: Basic Event



Figure 139: Gates

The events leading to the top-event **FAILURE** must be directly linked to a **GATE**, making it possible to evaluate the probability of an event happening based on the probability obtained by joining basic events and child gates.

As opposed to the other formalisms, the FT view does not offer a toolbar for the user to select components and make changes into the model. All operations for changing the model are performed by selecting menu items by using the mouse. For example, the user must right-click over the top event in order to create gates and event nodes. Changes in the model are performed by selecting the appropriate action on the respective menu item. Among the available options, the user can find the basic operations that are: insert, edit (properties), and remove components.

In order to create a gate, the user must right-click on the **FAILURE** event. On this pop-up menu, there is only one menu item: “Add Gate.” Figure 140 demonstrates how to insert a gate into the top-event. From there, users can choose between three different types of gates: *AND*, *OR*, and *KooN*.

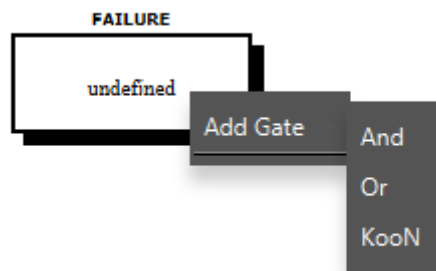
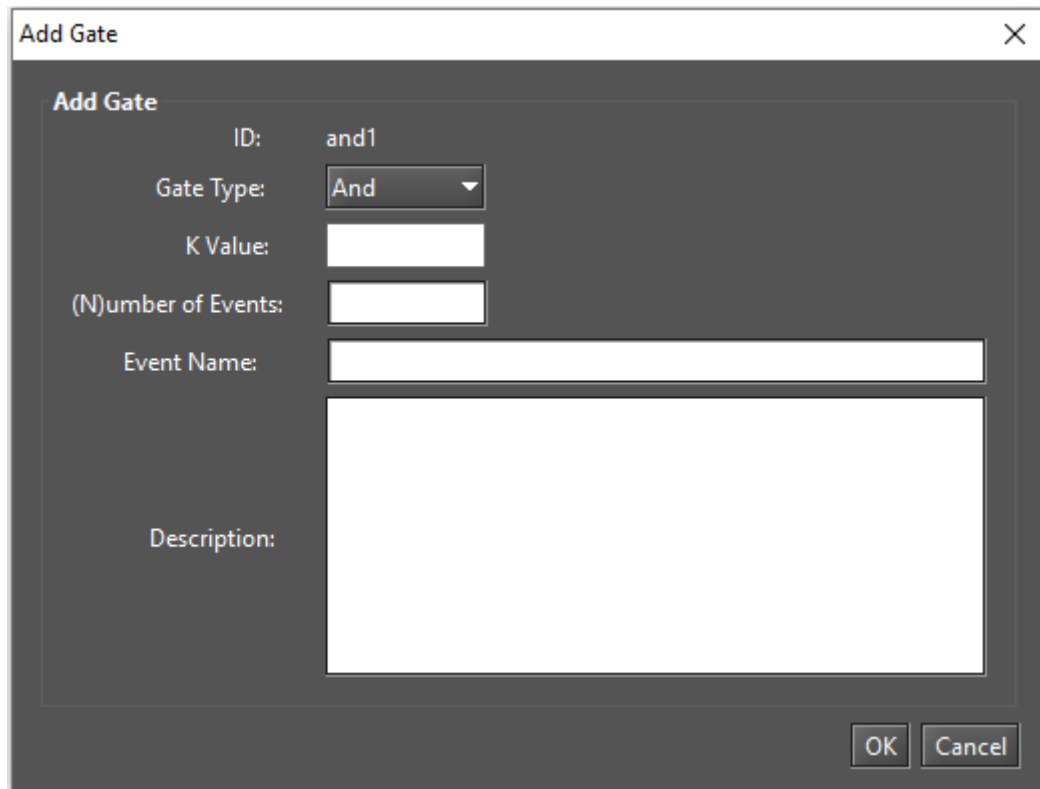


Figure 140: Adding a gate into the empty model

Once the type of gate is chosen, the “Add Gate” dialog is shown (see Figure 141). Below, we describe each field on this dialog.

- **ID.** ID for the gate to be inserted in the model. The ID is generated by Mercury, so users do not have the option to change it later. Each node in the FT graph has an ID that uniquely identifies it.
- **Gate Type.** Type of gate to be inserted. Users can change the gate type by clicking on the drop down button.
- **K Value.** When inserting a KooN gate, this field becomes enabled. A KooN gate represents a set of identical components (N) in a single node. All components in this set have the same failure and repair parameters. Using this type of gate, users should define how many components at least (K) must fail so that the set of components failed. Figure 142 depicts how a KooN gate is represented. As we can see, the values of the parameters K and N are presented by side of the gate id in the graph. In the current version of Mercury, it is not possible to add child gates to a KooN gate. The KooN gate can only have one basic child event that represents the set of components. As soon as possible, we intend to overcome this limitation.
- **(N)umber of Events.** When adding a gate, it is necessary to enter the number of basic events to be inserted into the gate as child nodes. Each gate must have at least two nodes. Once inserted, a basic event can be replaced with a gate. Thus, it is possible to define an FT model with a large number of levels and components.



The 'Add Gate' dialog box contains the following fields:

- ID:** and1
- Gate Type:** And (dropdown menu)
- K Value:** (empty text box)
- (N)umber of Events:** (empty text box)
- Event Name:** (empty text box)
- Description:** (empty text area)

Buttons: OK, Cancel

Figure 141: Add Gate Window

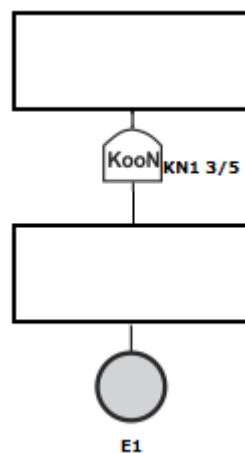


Figure 142: KooN Gate

- **Event Name.** Name for the component/event. It is displayed within the rectangular area above the node. Name can be changed at any time.
- **Description.** A description is additional information about the node itself or the component/subsystem represented by it. It aims to increase the understanding of the model and has no semantic value when evaluating the model. It is just plain text attached to the node.

Once entering the required fields and confirming by clicking on the button OK, Mercury inserts the nodes and updates the FT graph. Figure 143 depicts an AND gate having two basic events.

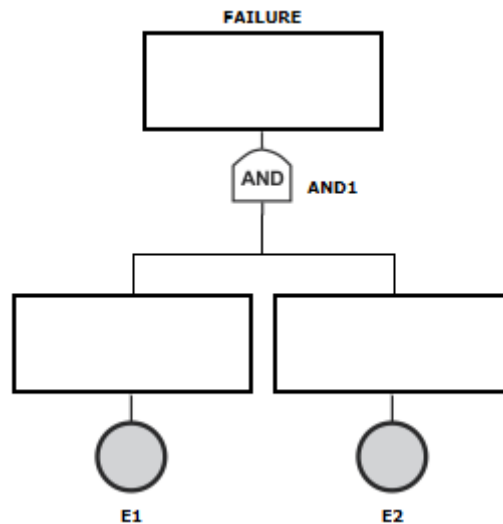


Figure 143: AND Gate with Two Basic Events

By double-clicking on a gate opens the “Gate Properties” dialog (see Figure 144).

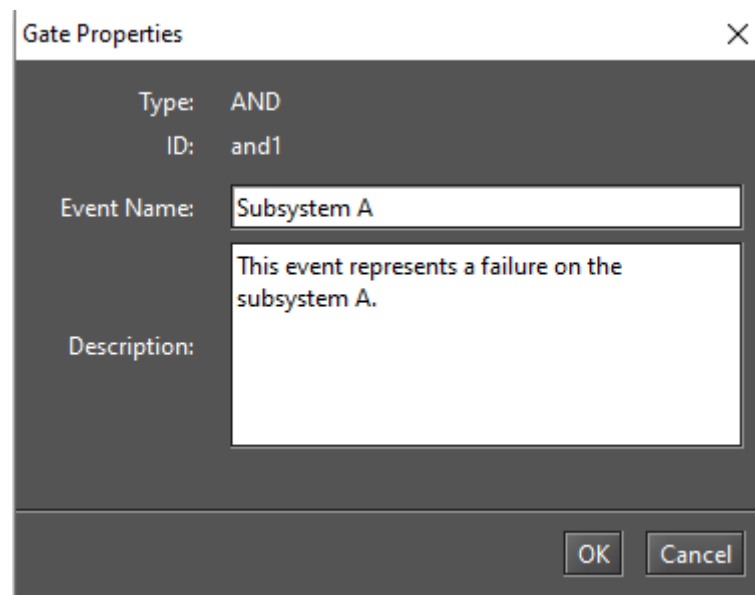


Figure 144: Gate Properties Window

By changing the name and description, the model is updated accordingly. Figure 145 shows the updated model.

Another way to edit the properties of a gate is by right-clicking on it and choosing “Properties...” from the menu that appears. As we can see by looking at Figure 146, some options are available in this menu.

Below, we describe the options available in this popup menu.

- **Add events.** By selecting it, a dialog appears (see Figure 147) where users must enter the number of events to be inserted into the selected gate. Once done, it is necessary to define the properties (failure/repair parameters) for each new basic event.

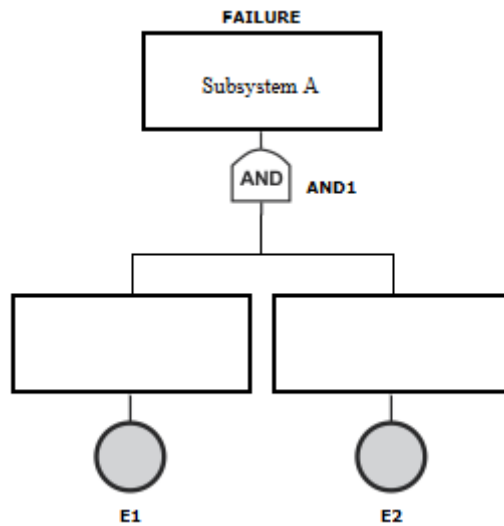


Figure 145: Name and Description Updated

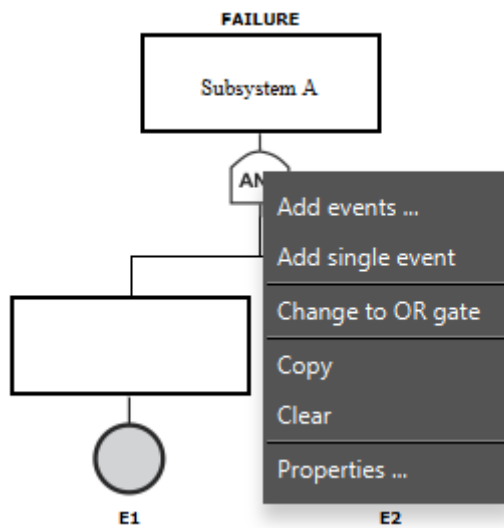


Figure 146: Gate Popup Menu

The image shows a software dialog box titled 'Add Events'. It has a standard title bar with a close button (X). The main content area has a label 'Add Events' and a text input field preceded by the label '(N)umber of Events:'. At the bottom right of the dialog are two buttons: 'OK' and 'Cancel'.

Figure 147: Add Events Dialog

- **Add single event.** Insert an empty event into the gate. By considering our example (see Figure 145), after selecting “Add single event” from the popup menu of the AND gate, a basic event is inserted into this gate, as we can see highlighted in Figure 148.

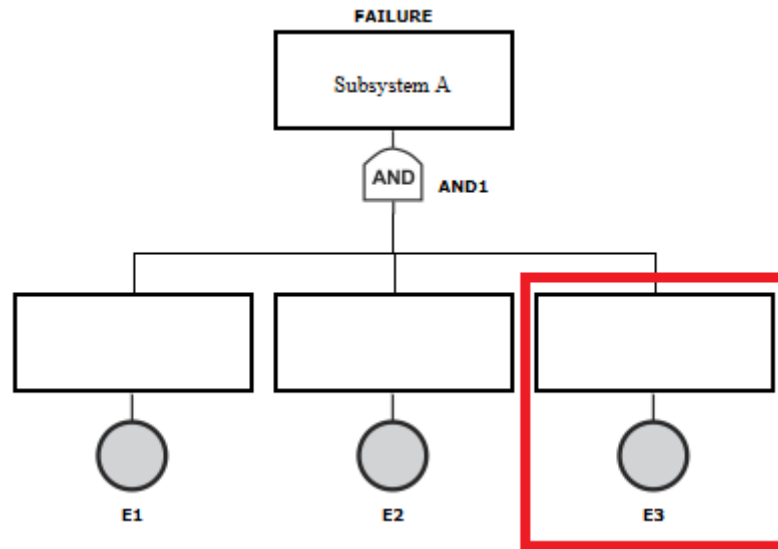


Figure 148: AND Gate with a New Event

- **Change to {OR, GATE} gate.** Change the type of the selected gate. When a port logic OR is selected, the only available option is to change it to an AND gate. The opposite applies for an AND gate. As an example, by considering our model, when selecting this option, the tool changes the AND gate (see Figure 148) to an OR gate (see Figure 149).

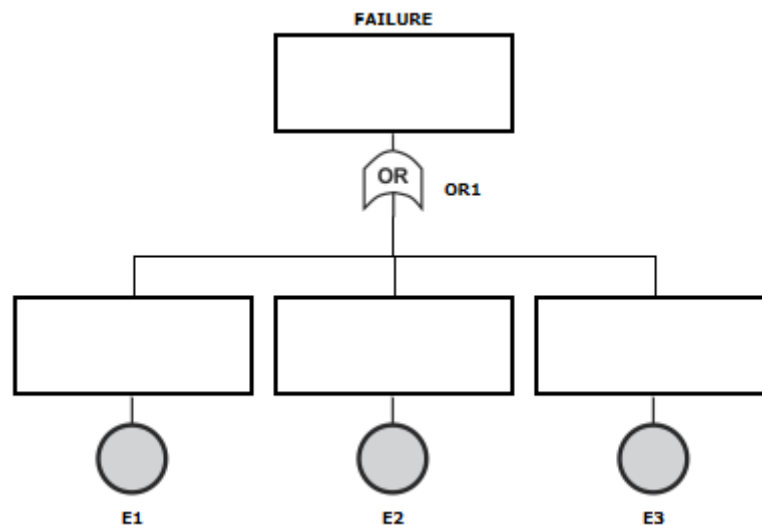
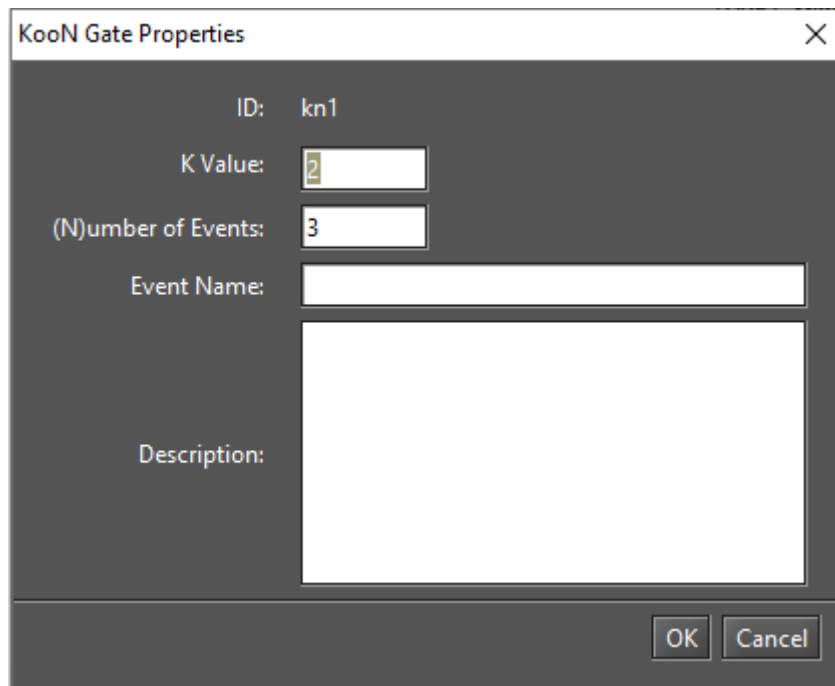


Figure 149: Changing the Type of the Gate

- **Copy.** Copy the selected gate and all its child components to clipboard.
- **Clear.** Empty the model. This option is only available when the selected gate is the top-level gate.
- **Properties.** By selecting this option, the user can change the properties of the selected gate. Figure 144 shows the “Gate Properties” window for AND and OR gates. For a KooN gate, in addition to "Event Name" and "Description", it is possible to change parameters K and N , as we can see in Figure 150.

A screenshot of a software dialog box titled "KooN Gate Properties". The dialog has a dark gray background and a light gray border. At the top right is a close button (X). The main area contains several labels and input fields: "ID:" followed by the text "kn1"; "K Value:" followed by a text box containing the number "2"; "(N)umber of Events:" followed by a text box containing the number "3"; "Event Name:" followed by an empty text box; and "Description:" followed by a large empty text area. At the bottom right are two buttons: "OK" and "Cancel".

KooN Gate Properties

ID: kn1

K Value: 2

(N)umber of Events: 3

Event Name:

Description:

OK Cancel

Figure 150: KooN Gate Properties

By right-clicking over a non-top gate, a slightly different popup menu is displayed, as we can see by looking at Figure 151.

Below, we describe the options that we have not yet presented and that are available for non-upper gates.

- **Change to a blank event.** Replace the gate and all its child nodes with an empty event.
- **Cut.** Cut the selected gate to clipboard.
- **Paste.** Replace the selected gate with nodes on clipboard. This option is only enabled when components have been copied to clipboard.
- **Delete.** Remove the selected gate and all its child nodes.

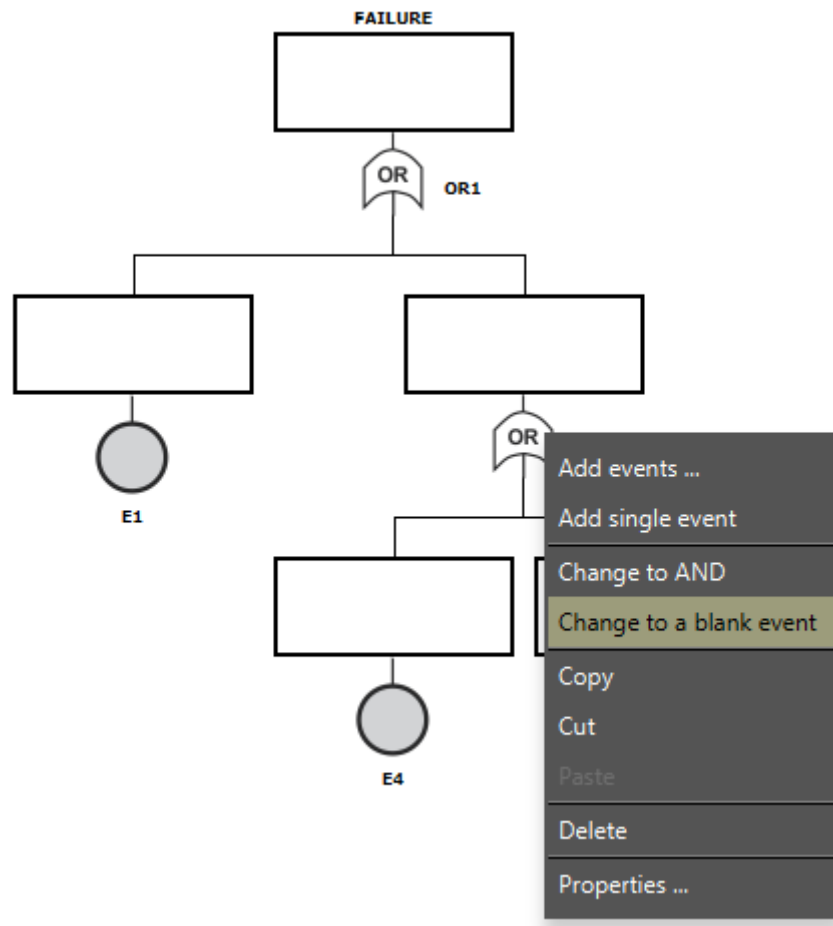


Figure 151: Popup Menu for Non-Top Level Gates

By right-clicking on any basic event, a pop-up menu appears, as we can see by looking at Figure 152.

Below, we describe each menu item.

- **Change to a blank event.** Replace the selected node with an empty basic event.
- **Add gate.** Replace the selected basic event with a gate. After choosing the type of gate to replace the basic event from the submenu, the “Add Gate” dialog is shown (see Figure 141).
- **Copy.** Copy the selected event to clipboard.
- **Cut.** Cut the selected event to clipboard. It is only possible when its parent gate has at least three direct child nodes. An error occurs by selecting this option when there are only two direct nodes at the gate, as depicted by Figure 153.

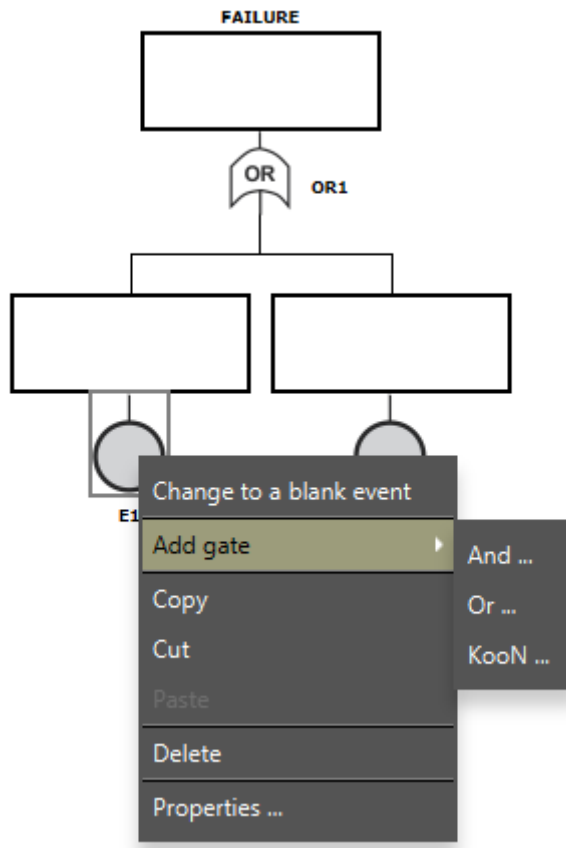


Figure 152: Popup Menu for Basic Events

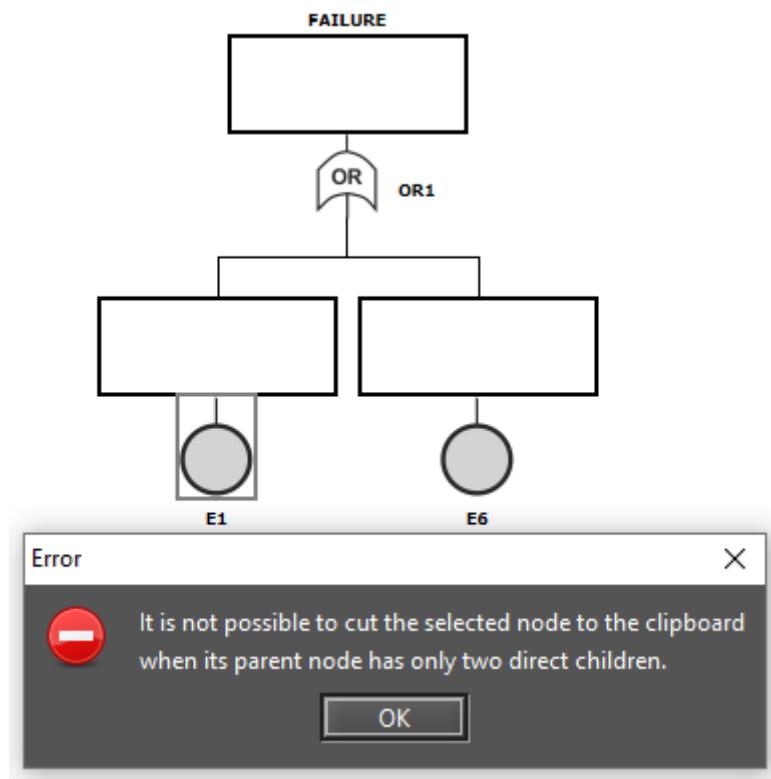


Figure 153: An error occurred while trying to cut a node to clipboard

- **Paste.** Replace the select node with components on the clipboard. It is only enabled when some component has been copied to the clipboard.
- **Delete.** Remove the selected node from the model. It is important to highlight each gate needs at least two direct nodes. Thus, an error occurs when trying to remove a node when the model has only two nodes, as we can see by looking at Figure 154.

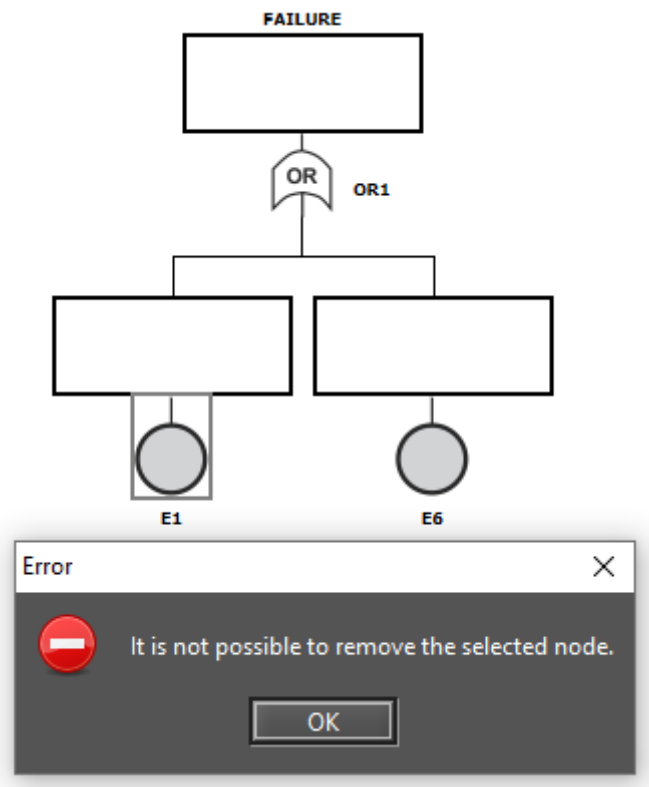


Figure 154: An error occurred when trying to delete a basic event

- **Properties.** Show the dialog box for changing properties of basic events (see Figure 155).

Event E1 - Properties

Event Name:

Description:

TIME State: Default

Parameters

Failure Distribution: Exponential

Mean value: 0.0

Repair Distribution: Exponential

Mean value: 0.0

Price (\$): 0.00

Figure 155: Properties Dialog for Basic Events

Next, let us do an overview of the properties of basic events.

- **Parameters Type.** Basic events accept four types of parameters: RATES, TIME, AVAILABILITY, and RELIABILITY. Just one of them may be selected at a given moment. The default type is TIME. When parameter type is TIME or RATES, the user can enter the appropriate values for the failure and repair parameters (see Figure 155). On the other hand, when the type is AVAILABILITY or RELIABILITY, the user can enter the respective value by considering the selected type, as we can see in Figure 156. Considering the context of the last figure, the user must enter the availability of the component represented by the basic event.
- **State.** State of the basic event. There are two states available: DEFAULT or FAILED. The default state is DEFAULT what means the component is working properly. On the other hand, when the state is FAILED, it indicates that that component has failed.

Event E1 - Properties

Event Name:

Description:

AVAILABILITY State: Default

Parameter

AVAILABILITY ...

Price (\$):

OK Cancel

Figure 156: Defining the Availability for a Basic Event

- **Failure Parameters.** Mercury supports a large number of probability distributions. Depending on the chosen distribution, fields representing the parameters of the chosen distribution appear in order for the user to enter their values. A label may be attached to each failure parameter. Button "..." enables us to select a label already declared.
- **Repair Parameters.** Fields representing the parameters of the chosen distribution appear for the user to enter their values. A label may be attached to each repair parameter. Button "..." enables us to select a label already declared.
- **Price.** Cost regarding the component represented by the basic event. The cost of events is considered when evaluating the model through the "component importance and the total cost of acquisition" method. See Section 4.1.4 for further information about this type of evaluation.

Finally, let us see the types of basic events and how they are graphically represented. Figure 157 shows the six

types of basic events. Figures 157.a and 157.b show events with no parameters assigned. Figures 157.c and 157.d demonstrate exponential events but in c) the state of the event is defined as “Default” while in d) it is defined as “Failed.” Figures 157.e and 157.f demonstrate non-exponential events, but in e) the state of the event is defined as “Default” while in f) it is defined as “Failed.” As we can see, events with no failure/repair parameters are depicted by a light gray circle. Exponential events are depicted by a dark gray circle. And, non-exponential events are depicted by a blue block.



Figure 157: Types of Basic Events

When the required parameters of all basic events linked to a father gate are entered, the color for that gate changes to yellow as depicted in Figure 158.

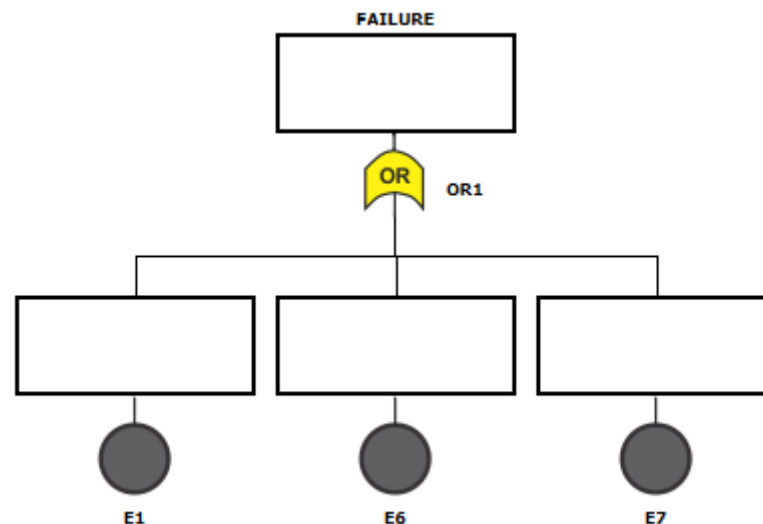


Figure 158: FT Model

Mercury has a feature to increase the readability of models. Once the parameters of a node have been assigned, it is possible to read them in the drawing area by positioning the mouse cursor over the node. After this, a tooltip appears showing all properties of that node. As we can see by looking at Figure 159, all properties appear in the tooltip. All types of components of all formalism supported by Mercury provide this feature.

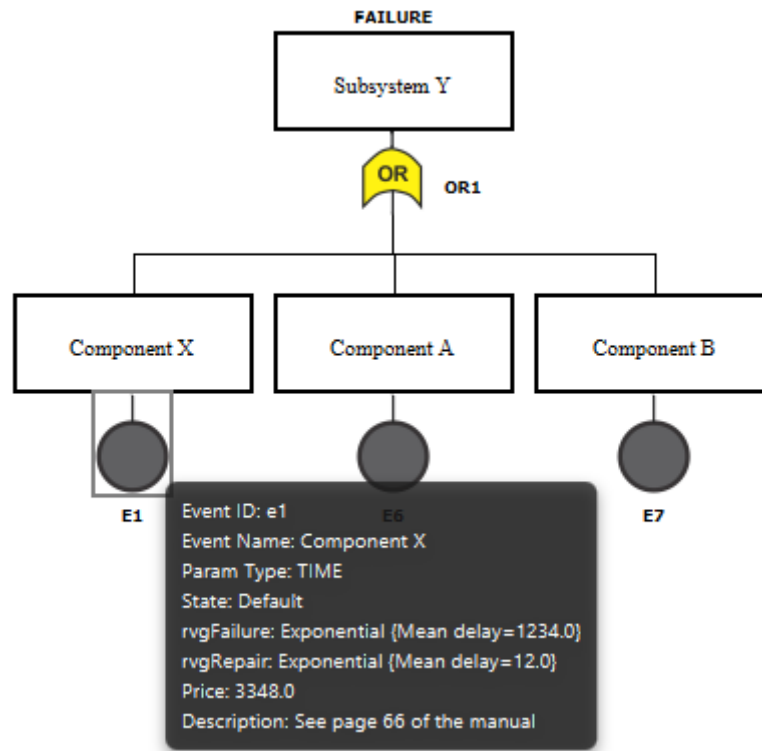


Figure 159: Tooltip for a FT Node

4.1 FT Evaluation

Mercury provides a large number of evaluations for FTs:

- Exact Evaluation,
- Bounds Evaluation,
- Importance Measures,
- Experiment,
- Get Functions,
- Sensitivity Analysis, and
- Export to RBD model.

These evaluations are available in the menu Evaluate -> FT Evaluation, as depicted in Figure 160. We introduce each one in the next subsections.

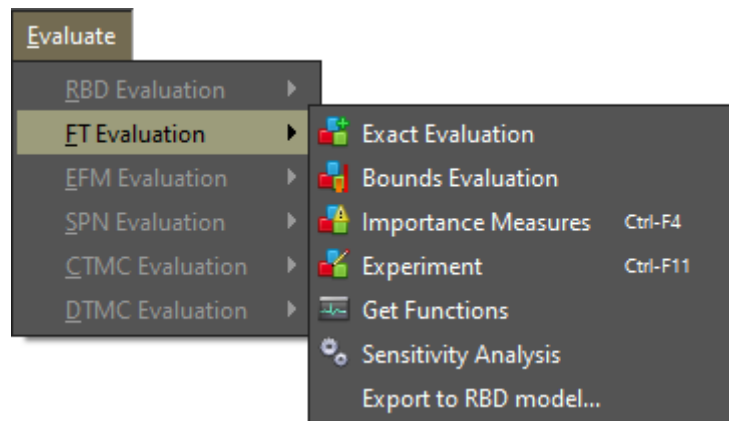


Figure 160: FT Evaluation Menu

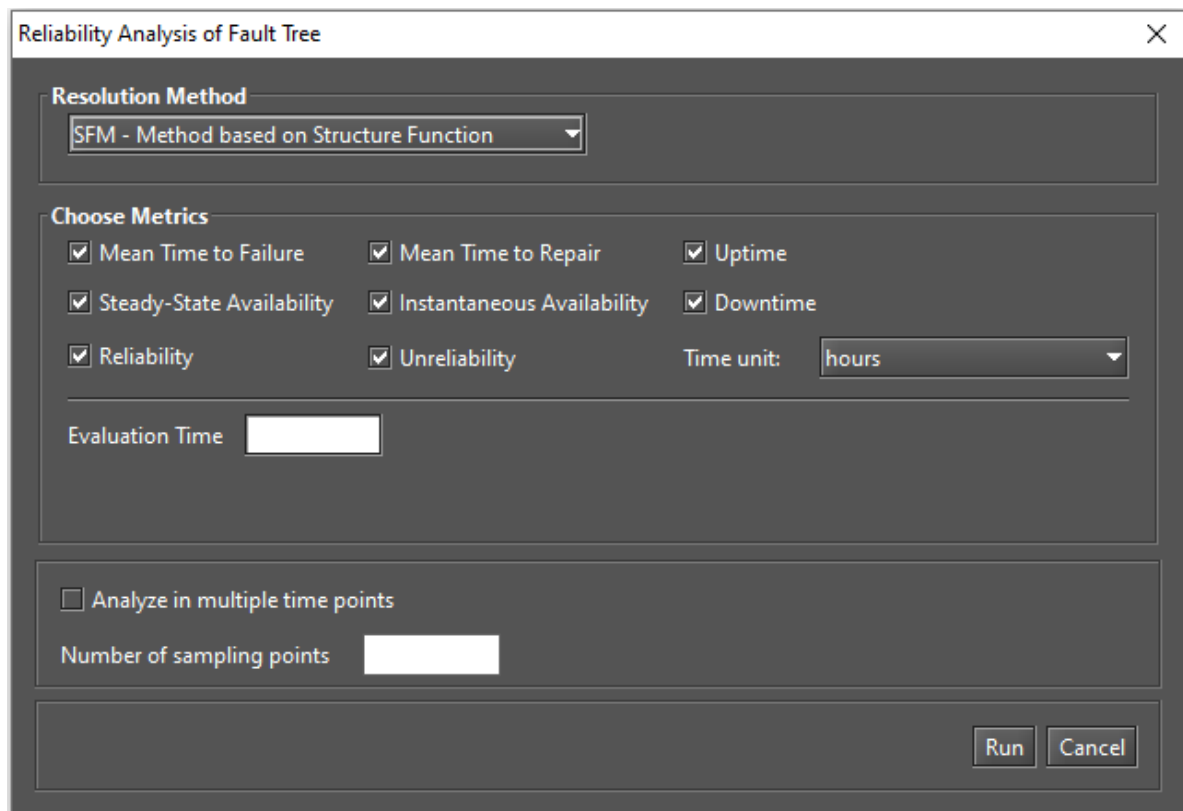


Figure 161: Exact Evaluation of FTs

4.1.1 Exact Evaluation

Through the Exact Evaluation a large number of dependability analyses may be performed. Exact Evaluation can be accessed in the menu Evaluate -> FT Evaluation -> Exact Evaluation. Figure 161 shows the window for performing dependability evaluations.

As we can see, users can evaluate eight metrics: mean time to failure, mean time to repair, steady-state availability, instantaneous availability, reliability, unreliability, uptime, and downtime. Besides that, the user can choose the time unit to be considered when computing uptime and downtime: seconds, minutes, hours, and days. When time-dependent metrics are chosen — reliability, unreliability, or instantaneous availability —,

the time parameter is required. In addition, there is an option to analyze time-dependent metrics considering multiple points on time. Time interval goes from 0 to the evaluation time, as the last one is divided by considering the number of points entered. The metric is computed for each point.

Mercury provides two methods for computing dependability metrics. It is possible to choose between SFM (Method based on Structure-Function) and SDP (Sum of Disjoint Products), as depicted in Figure 162. SFM computes metrics considering the structural function of the model. On the other hand, the SDP method, based on Boolean algebra, computes metrics considering minimal cuts and minimum paths.

After choosing the options and entering the evaluation time and number of sampling points, if required, the user must click on the button “Run” in order to Mercury perform evaluations.

Reliability Analysis of Fault Tree

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

Run Cancel

Figure 162: Window for FT Evaluation

Now, let us demonstrate how to carry out evaluations on FTs by using the Mercury tool. We have considered a model composed by four events, each one contributing to the overall failure of the evaluated system (see Figure 163). As we can see, the system fails when events **e1** or **e8** or both events into the AND gate — **e3** and **e4** — occur. Node **e8** represents a k-out-of-n component. In order to event **e8** occurs, it is necessary that at least 3 of the 5 components fail.

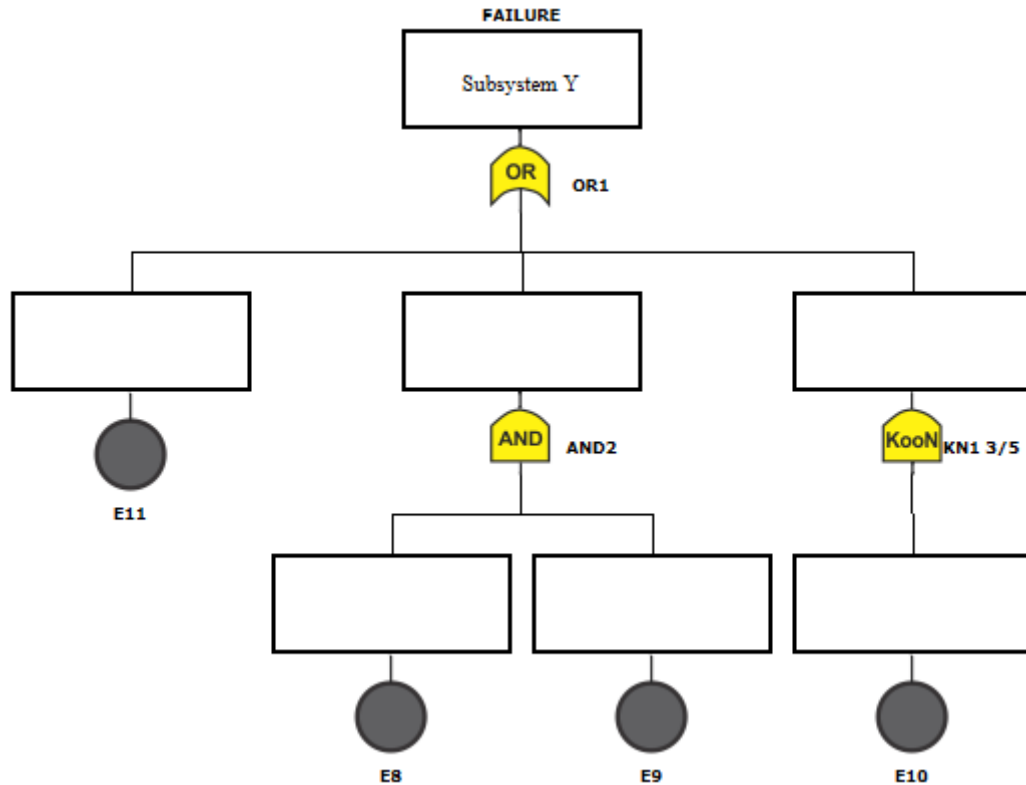


Figure 163: FT Model for Dependability Evaluation

We performed an evaluation by considering the time parameter equal to 4380h, “days” as the time unit, and considering six sampling points (see Figure 164). At the end of the evaluation, a window appears showing the result, as we can see in Figure 165. The results are divided into two groups. These are “Steady-state Results” for steady-state metrics and “Instantaneous Results” for time-dependent metrics. Listing 6 shows a result obtained by evaluating a FT as an example.

Reliability Analysis of Fault Tree

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: days

Evaluation Time 4380

☒ Analyze in multiple time points
 Number of sampling points 6

Run Cancel

Figure 164: FT Analysis

Listing 6: Dependability Results for a FT Model

```

***** Steady-state Results *****
MTTF:          543.5476190476171
MTTR:          5.341457343291265
Availability:   0.9902686033061311
Number of 9's: 2.011824823358696
Uptime:        361.68788227219 days
Downtime:      3.554316727810011 days

***** Instantaneous Results *****
Time    Reliability    (9's)    Unreliability    Inst. availability
0.0000  1.0000          infinity  0.0000          1.0000
730.0000 0.265623054979  0.134080965804  0.734376945021  0.990268603306
1460.0000 0.02725934473   0.012002932597  0.97274065527   0.990268603306
2190.0000 0.001955731123  0.000850194882  0.998044268877  0.990268603306
2920.0000 0.000119844142  0.000052050769  0.999880155858  0.990268603306
3650.0000 0.000006795692  0.000002951342  0.999993204308  0.990268603306
4380.0000 0.000000370113  0.000000160738  0.999999629887  0.990268603306

```

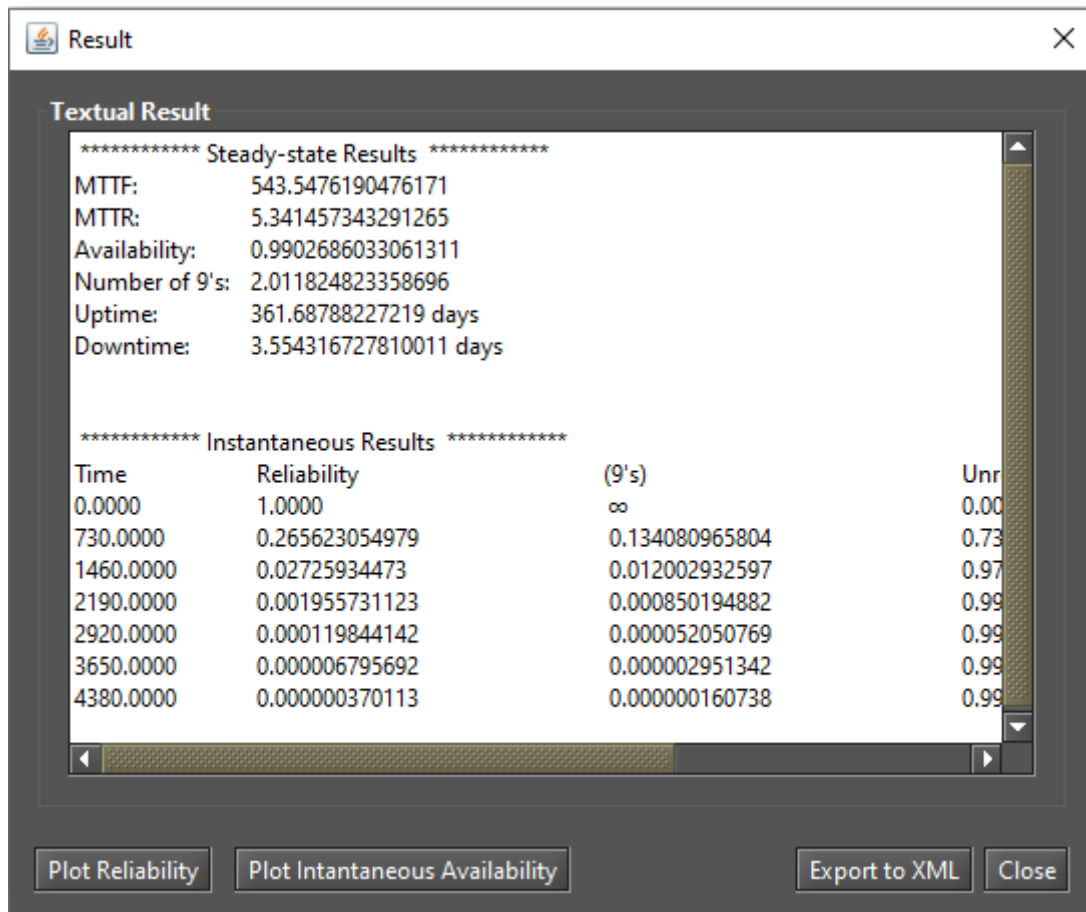


Figure 165: Dependability Results

Figure 166 shows the “Reliability Chart” dialog displayed by clicking on the button “Plot Reliability.” This button is only visible when some reliability metric is selected on the input dialog. The number of points on the plotted lines is determined by the number of points entered by the user.

Resolutions of models through simulation are required when non-exponential probability distributions are associated to some event. Figure 167 depicts a model with a non-exponential node (node e1). In this case, these non-exponential nodes are converted into SPNs and the model is solved through simulation. Two resolution methods are available: *SFM - Simulation* and *SDP - Simulation*. By detecting this situation, Mercury displays the message “Non-exponential distributions detected” at the bottom of the Reliability Analysis screen (see Figure 168).

By clicking on the button “Run”, some parameters must be entered for supporting the simulation. The required parameters depend on the chosen metrics. By selecting only steady-state metrics, Mercury shows the window depicted in Figure 169. On the other hand, when selecting one or more time-dependent metrics, Mercury shows the window depicted in Figure 170. Results are presented once the parameters have been defined and the simulation has been finished. It is important to highlight that the following metrics cannot be solved by performing simulation: MTTE, MTTR, and Instantaneous Availability. For further information about simulations, please see Section 2.1.

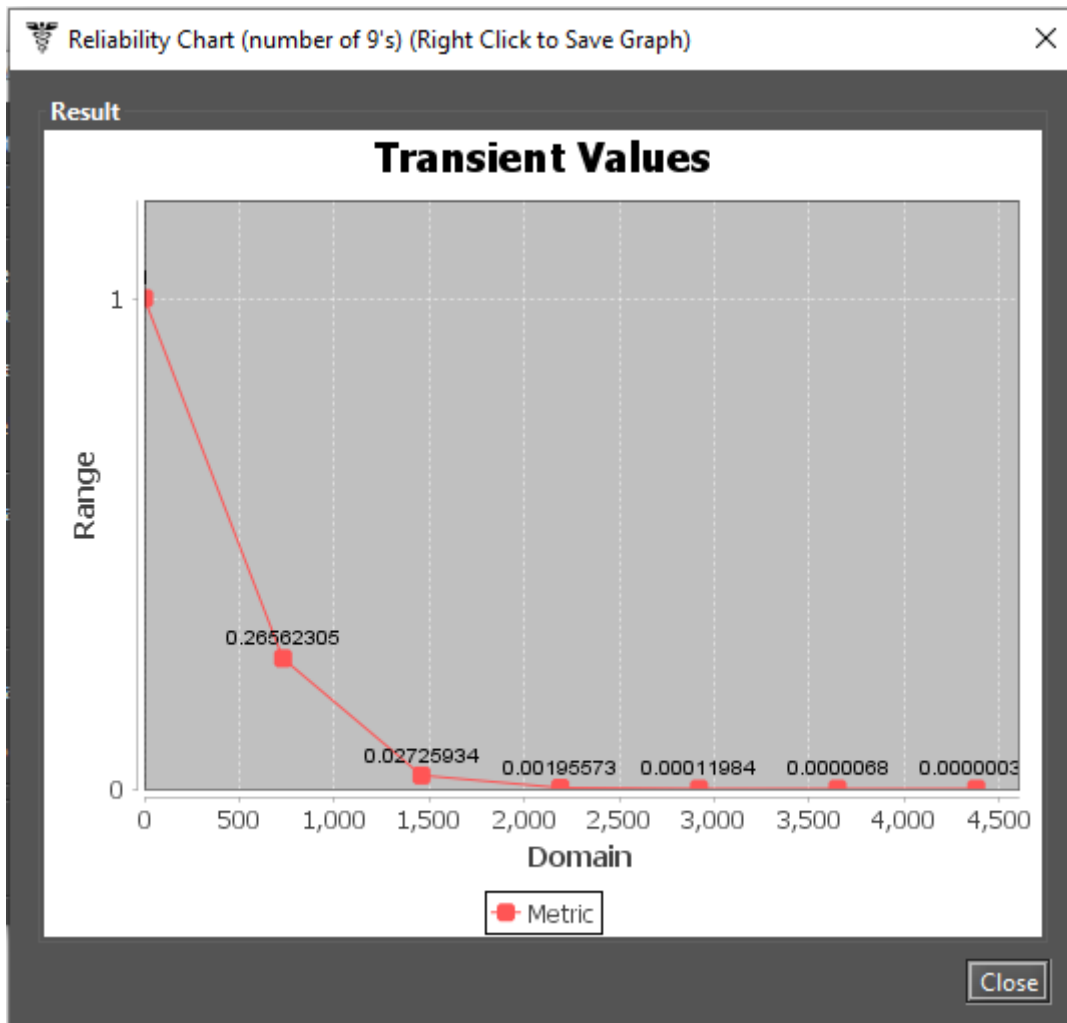


Figure 166: Reliability Chart

4.1.2 FT Experiment

Mercury allows us to evaluate the impact of varying some parameters in the model. Now, we demonstrate how to use the experiment function.

The first step is to define one or more labels. Labels are variables that store numeric values and can be attached to the failure/repair parameters of events. The value of a label is changed considering a stepsize and at each change, the selected metric is evaluated. Another way to do this is by right-clicking on the label area in the left-side panel and selecting “Insert label”, as depicted in Figure 171.

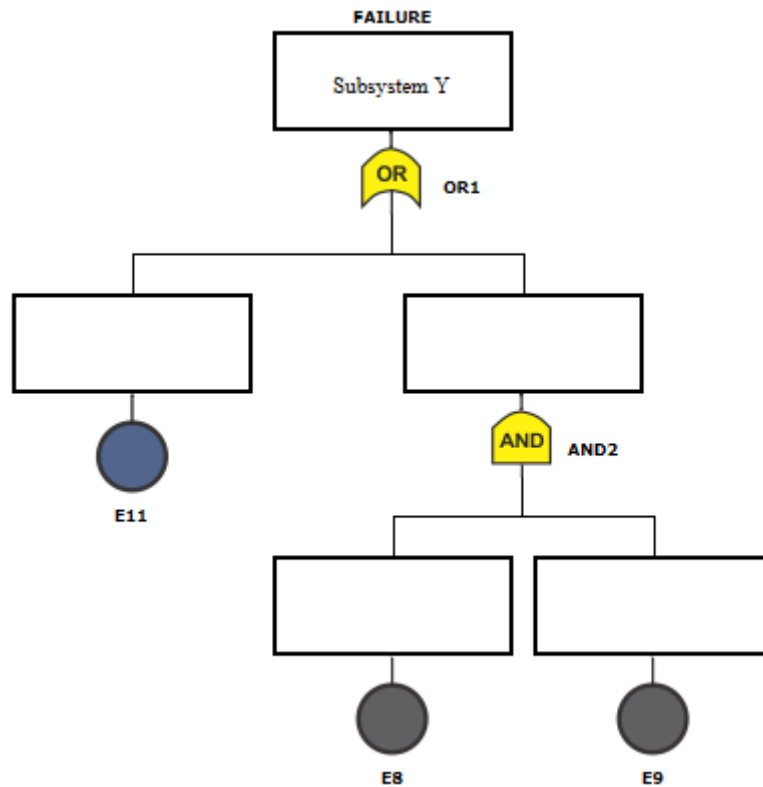


Figure 167: Non-Exponential Node in the Model

Reliability Analysis of Fault Tree

Resolution Method

SFM - Simulation

SFM - Simulation

SDP - Simulation

☐ 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

Non-exponential distributions detected.

Run Cancel

Figure 168: FT Analysis through Simulation

Once this is done, the “Label Properties” window is shown (see Figure 172). There, the user can set the properties of the label.

Simulation Parameters
✕

Non-exponential distributions detected.
Please, provide the parameters for the simulation.

| | |
|----------------------------|----|
| Confidence Level % | 95 |
| Max. Relative Error % | 10 |
| Max. Simulation Time (sec) | 0 |
| Min. Warm-up Time | 50 |
| Batch Size | 30 |

Set Cancel

Figure 169: Simulation for Steady-State Metrics

Simulation Parameters
✕

Non-exponential distributions detected.
Please, provide the parameters for the simulation.

| | |
|----------------------------|--------|
| Confidence Level % | 95 |
| Max. Relative Error % | 10 |
| Max. Simulation Time (sec) | 0 |
| Time | 8760.0 |
| # Sampling Points | 1.0 |
| Min. Warm-up Time | 50 |
| Batch Size | 30 |

Set Cancel

Figure 170: Simulation for Time-Dependent Metrics

Once a label has been inserted, it is available in the left-side panel on the FT tab. Now, it can be associated with one or more parameter of events. By right-clicking on any label, a pop-up menu with three menu items

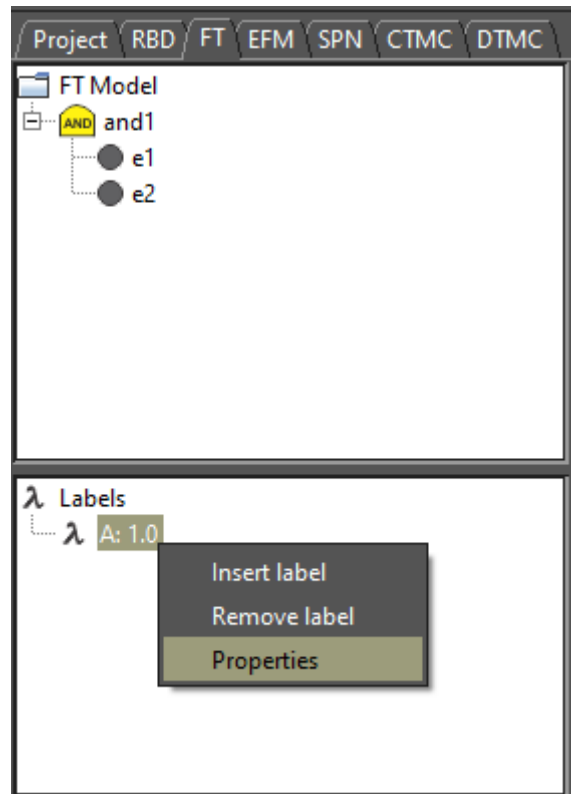


Figure 171: FT Panel in the Left-Side

Figure 172: Inserting a label in the FT Model

appears (see Figure 171). Below, we describe each of them.

- **Insert label.** Show the window “Label Properties” allowing the user to insert a label.
- **Remove label.** Remove the selected label.
- **Properties.** Show the window “Label Properties” allowing the user to change the properties of the label.

After defining a label, it is necessary to attach this label to the failure/repair parameter of the component under evaluation. Figure 173 demonstrates how to attach a label to a parameter of an event.

The screenshot shows the 'Event E1 - Properties' dialog box. It contains the following fields and controls:

- Event Name:** A text input field.
- Description:** A large text area.
- TIME:** A dropdown menu.
- State:** A dropdown menu currently set to 'Default'.
- Parameters:** A section containing:
 - Failure Distribution:** A dropdown menu set to 'Exponential'. Below it, the 'Mean value' is 1234.0, and a label 'A' is attached to the right of the input field.
 - Repair Distribution:** A dropdown menu set to 'Exponential'. Below it, the 'Mean value' is 12.0, and an ellipsis button '...' is to the right of the input field.
- Price (\$):** A text input field set to 0.0.
- Buttons:** 'OK' and 'Cancel' buttons at the bottom.

Figure 173: Attaching a label to the parameter of an event

Experiment of FTs can be accessed in the menu Evaluate -> FT Evaluation -> Experiment. Figure 174 shows the "FT Experiment" window. In order to perform experiments, the user must enter values for all required fields.

Below, we describe each of them.

- **Parameter.** The label to have its value changed at each experiment iteration.
- **Metric.** Metric to be evaluated.
- **Minimum Value.** Initial value for the selected label.

- **Maximum Value.** Final value for the selected label.
- **Interval.** Step-size for changing the value of the label. The label starts with the minimum value and its value is increased by considering this interval. At each change, the selected metric is evaluated. The experiment finishes when the maximum value for the label is reached.
- **Evaluation Time.** Evaluation time to be considered for computing time-dependent metrics. For time-dependent metrics — reliability, unreliability, instantaneous availability — it is required to enter the time parameter.



Experiment of Fault Tree

Parameter: A: 1.0

Metric: Reliability

Minimum Value: 0.0 Maximum Value: 0.0

Interval: 0.0

Evaluation Time: 0.0

Run Experiment Cancel

Figure 174: FT Experiment

After defining the input parameters, the user must click on the button “Experiment” in order to start the experiment. Once the experiment has been finished, the “Experiment Result” dialog is shown (see Figure 175).

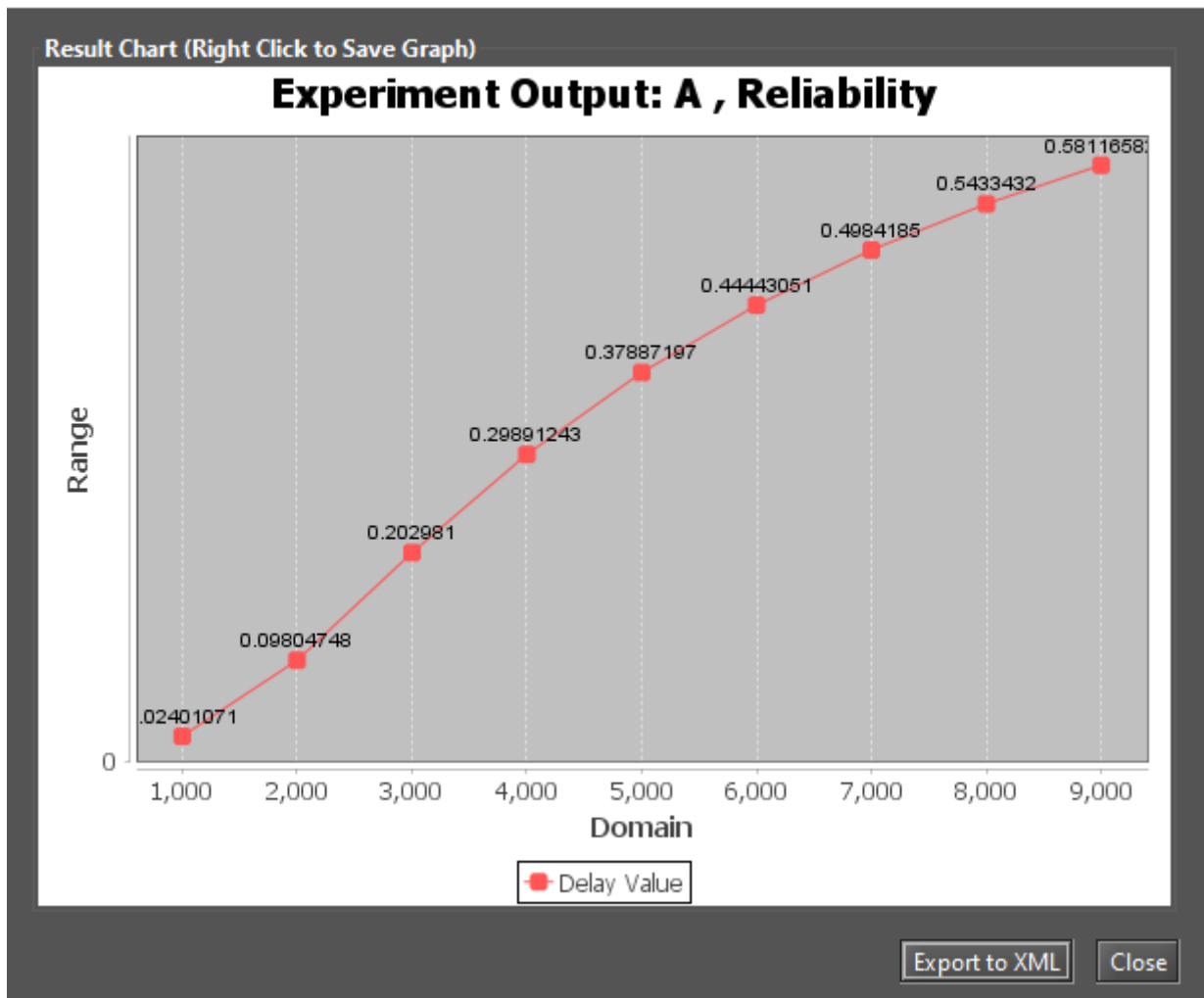


Figure 175: Experiment Result Dialog

4.1.3 Bounds for Dependability Analysis

Bounds for Dependability Analysis is used to estimate dependability metrics through the calculation of reliability, availability, or downtime. Therefore, it is necessary to estimate the bounds values (upper and lower limits) for computing this analysis, in which users quickly obtain results. This analysis should be performed when the model is huge. This analysis is subdivided into 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 required.

Users can access this analysis going to the menu Evaluate -> FT Evaluation -> Bounds Evaluation. Figure 176 shows the “Bounds for Dependability Analysis” window. As we can see in Figure 177, it is possible to evaluate four metrics: steady-state availability, instantaneous availability, reliability, and downtime. The time parameter is required when choosing time-dependent metrics. Once a metric has been selected and the time entered, if required, the user must click on the “Get Start Values” button in order to start the evaluation.

Bounds for Dependability Analysis of Fault Tree

Metric: Steady State Availability Get Start Values

Upper:

Lower:

Close

Figure 176: Bounds for Dependability Analysis

Steady State Availability

Steady State Availability

Instantaneous Availability

Reliability

Downtime

Figure 177: Metrics for Bounds Evaluation

Now, let us demonstrate how to perform bounds evaluation by using Mercury. We have considered a model composed by five events, each one contributing to the overall failure of the system (see Figure 178). As we can see, the system fails when events **e1**, **e6** and at least one of the following events occur: **e3**, **e4**, or **e5**. We have evaluated the bounds for this model by considering the time parameter equal to 8760h (see Figure 179).

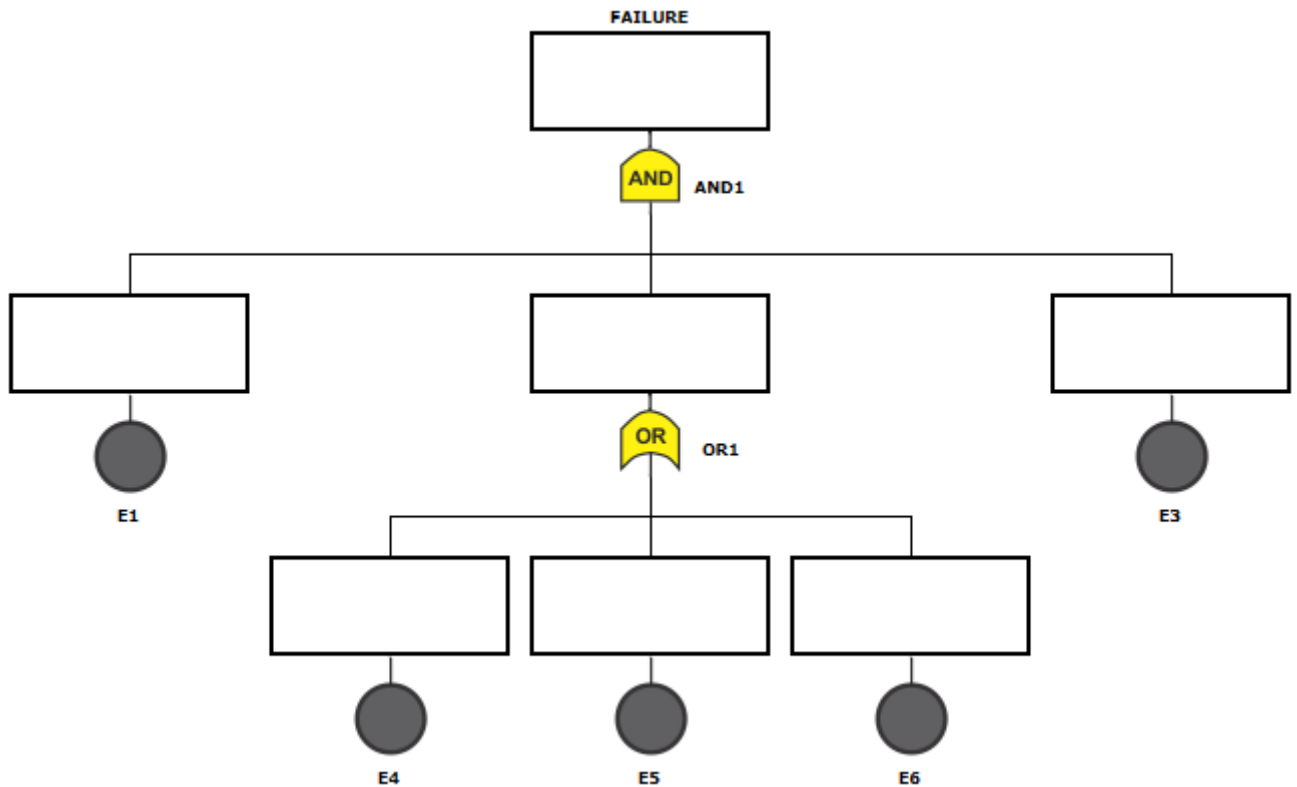


Figure 178: FT Model for Bounds Evaluation

Bounds for Dependability Analysis of Fault Tree

Metric: Reliability

Time: 8760

Upper: 0.7070295425162266

Lower: 8.260415540683038E-4

Get Start Values

1 Max: 3

1 Max: 3

Run

Close

Figure 179: Bounds for Reliability Metric

First, it is necessary to compute 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 is 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. After getting the upper and lower values, we have defined the number of steps to three for the upper and lower values. Thus, we obtained the result demonstrated in Figure 180. We have highlighted the values for the upper and lower bounds for the last step — step 3. As we can see, they are the same. By clicking on

the “Plot Chart” button, we can see the lower and upper bounds for three steps converging to the exact value (see Figure 181).

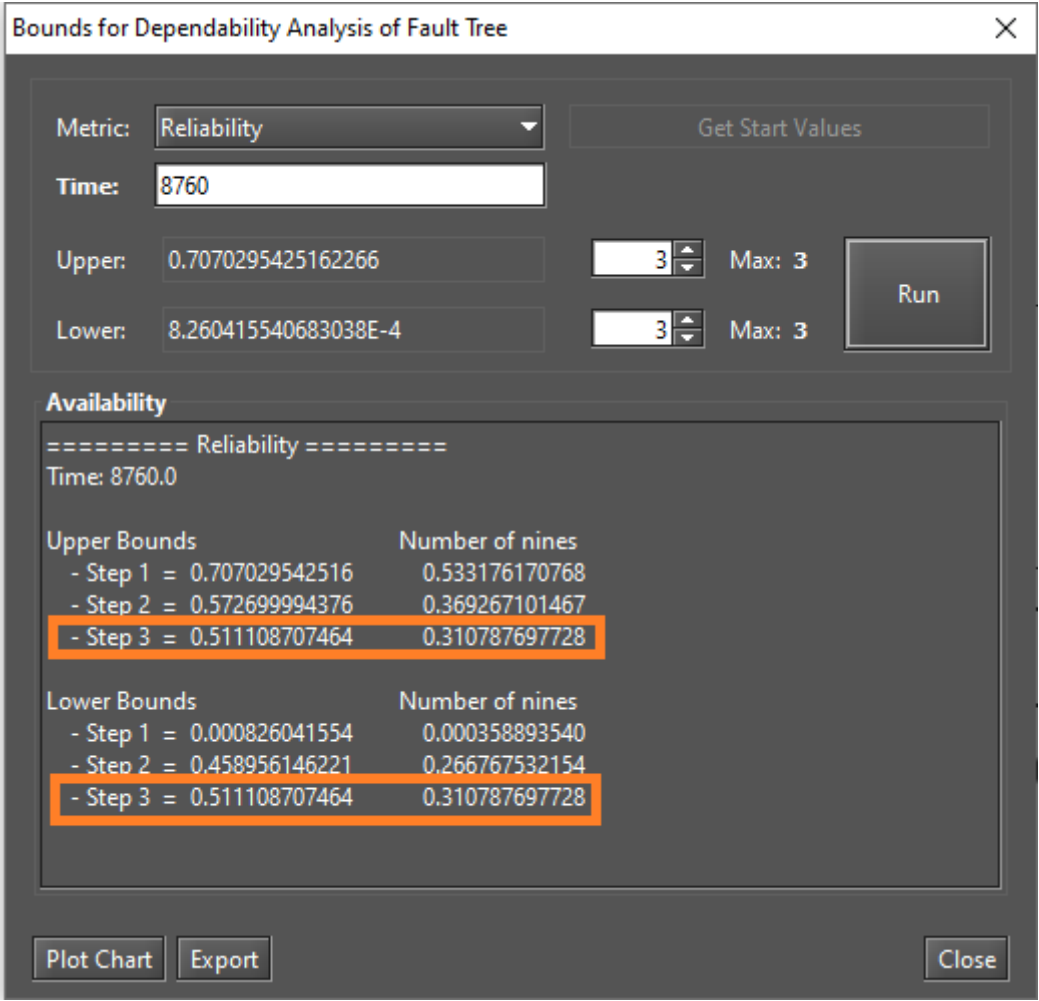


Figure 180: Bounds for Reliability - Result

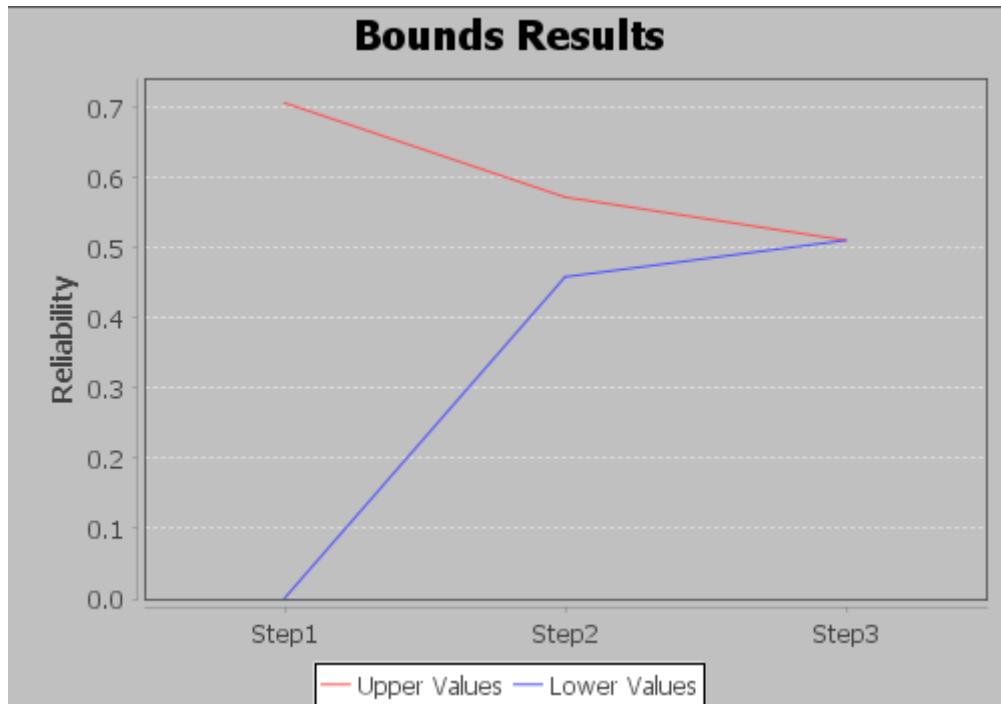


Figure 181: Plotting Bounds Results

The method adopted to find successive values and the number of iterations 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. Once the computation for the last path or last cut has been finished, the exact value for the metric can be found. The exact value will be the value found in the last step. We have performed the exact evaluation to obtain the reliability at 8760h. Figure 182 shows that reliability at 8760h is equal to the values obtained in the bounds evaluation for the maximum number of steps (see Figure 180).

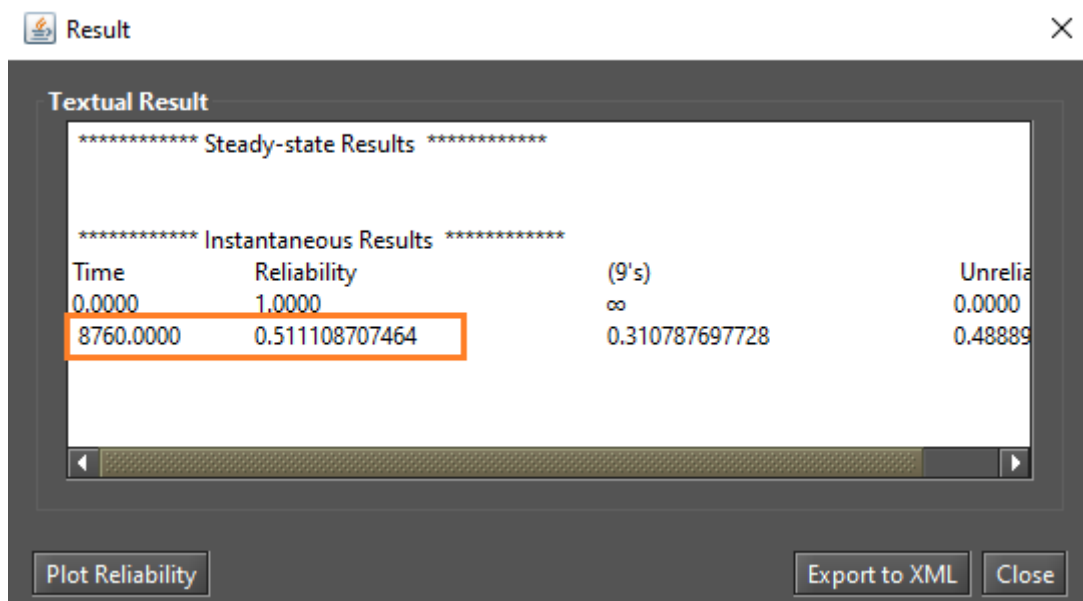


Figure 182: Reliability at 8760h

4.1.4 Component Importance and Total Cost of Acquisition

Component importance is a metric that indicates the impact of a particular component on the system. By considering importance values, the most important component (i.e., that one with the highest importance) should be improved in order to increase the reliability or availability of the system. This evaluation may be applied for example to assist in maintenance actions.

Importance measures make it possible to identify the relative importance of each component with respect to the overall system reliability or availability. Users can access this evaluation by going to the menu Evaluate -> FT Evaluation and select "Importance Measures." After that, users must select one metric on the "Component Importance Measures" window, after which click on the button "Evaluate" (see Figure 183). If the parameter "Cost" has been defined, it is also possible to evaluate the relationship between metrics and investment costs. The time parameter is required in order to evaluate reliability metrics.

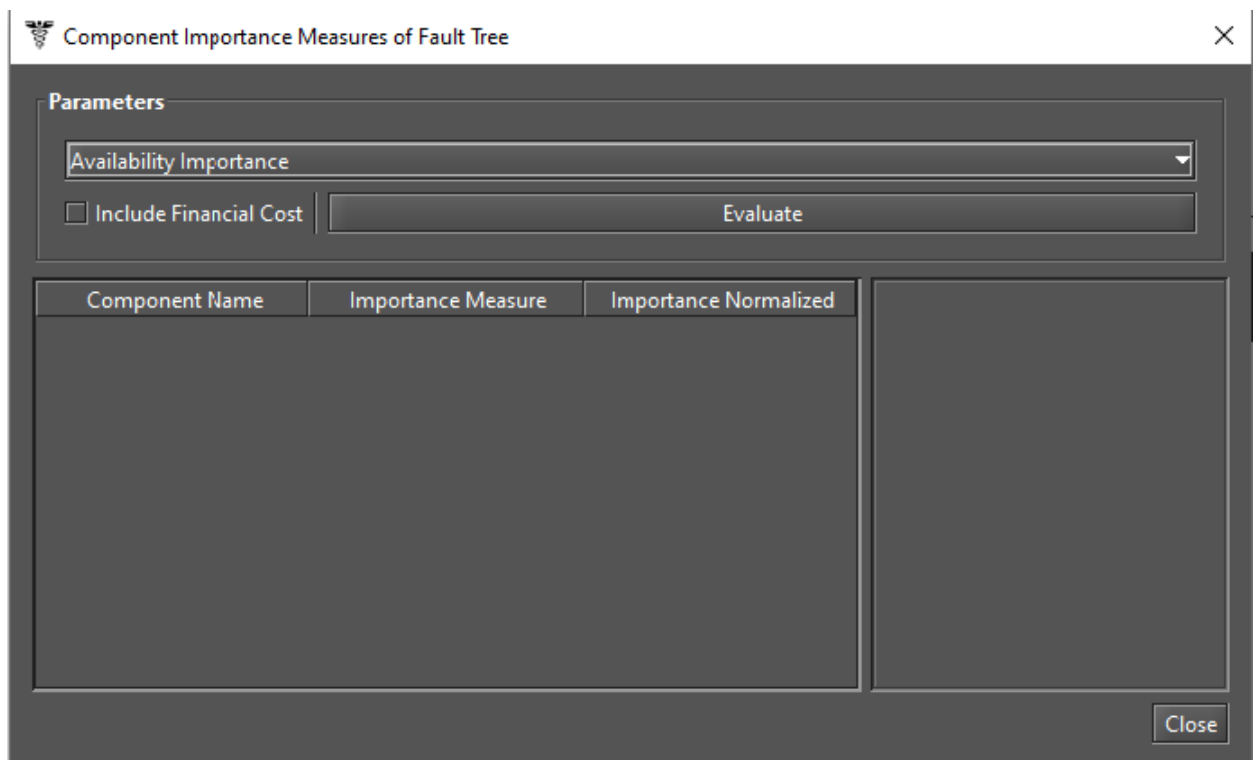


Figure 183: Component Importance Measures

As shown by Figure 184, users can choose between some types of measures available. The types are: "Availability Importance", "Reliability Importance (Birnbaum)", and "Criticality (Reliability or Availability) Importance (CRI)". The last one can be obtained considering the system in failure "(f)" or working.

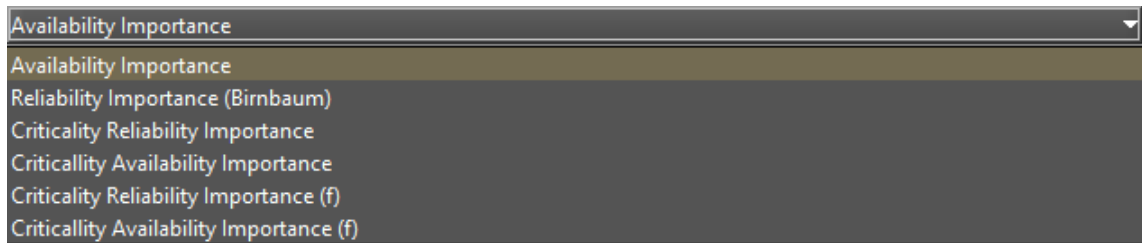


Figure 184: Component Importance Measures - Metrics of Interest

Following, we demonstrate how to execute “Component Importance” evaluations. We performed this evaluation considering the metric “Reliability Importance (Birnbaum)” for the model shown in Figure 185.

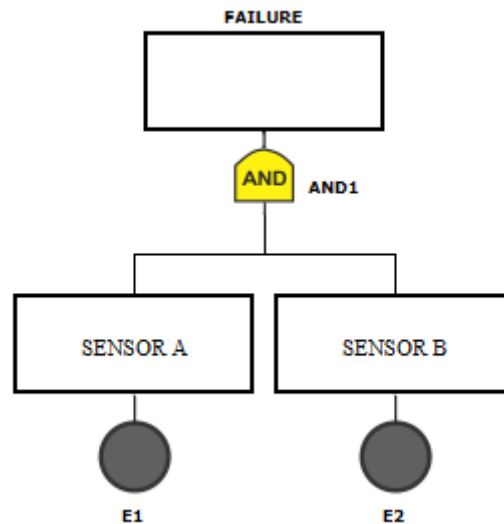


Figure 185: FT Model for Component Importance Evaluation

As we can see by looking at Figure 185, the evaluated system has two sensors and each one is attached with the only gate of the model. In order for the system to fail, it is necessary that at least one sensor fails. “Sensor A” has MTTF equal to 17520h and MTTR equal to 72h. “Sensor B” has MTTF equal to 6000h and MTTR equal to 24h. Being all times exponentially distributed. When performing the “Component Importance” evaluation considering the time parameter equal to 8760h, we obtained the results presented in Figure 186.

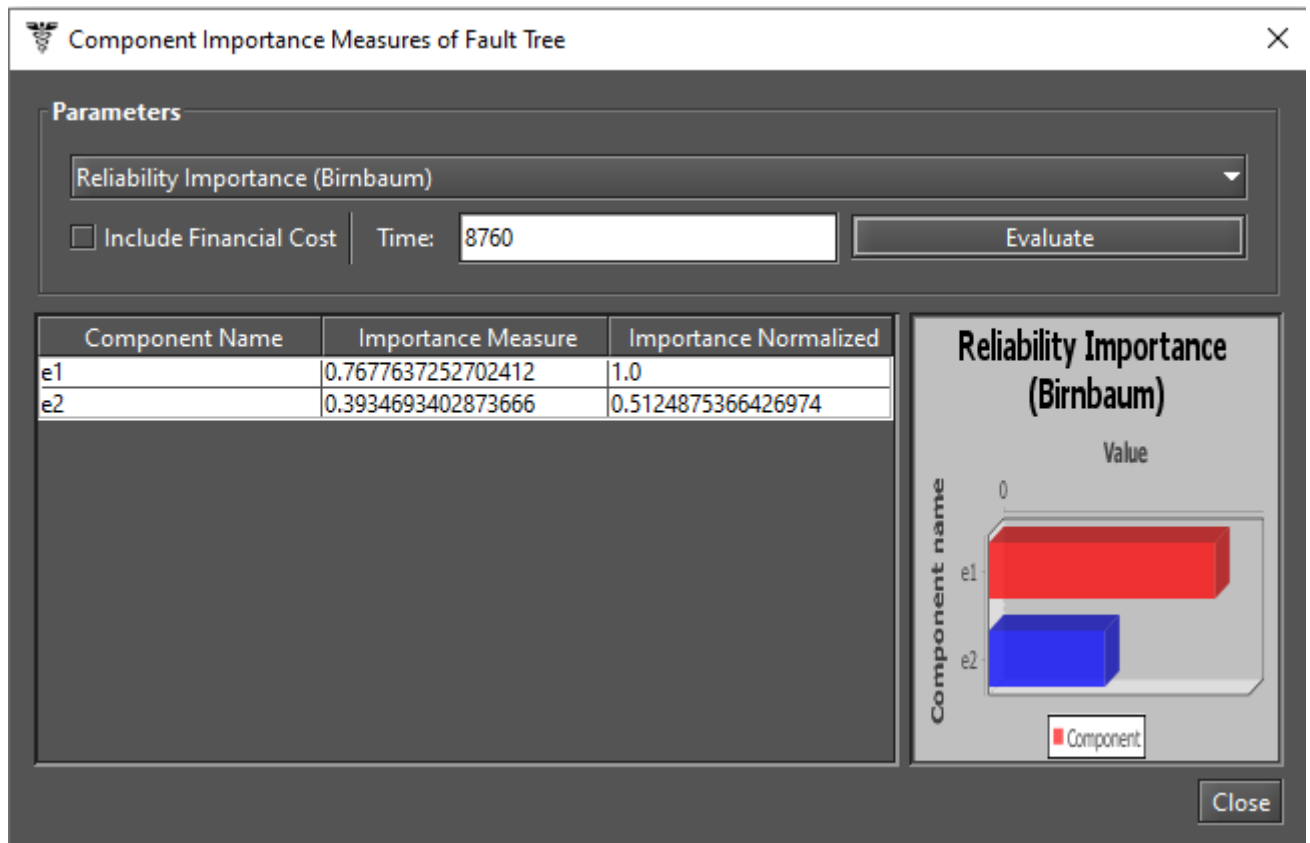


Figure 186: Reliability Importance Results

The results show the importance value for each component and a graphical view depicts as a ranking highlighting those most significant. As we can see, the most impacting component for reliability is “Sensor B” (e6). By replacing this component with another one having the same MTTR, but an MTTF equal to 12000h, this changes the result as depicted in Figure 187.

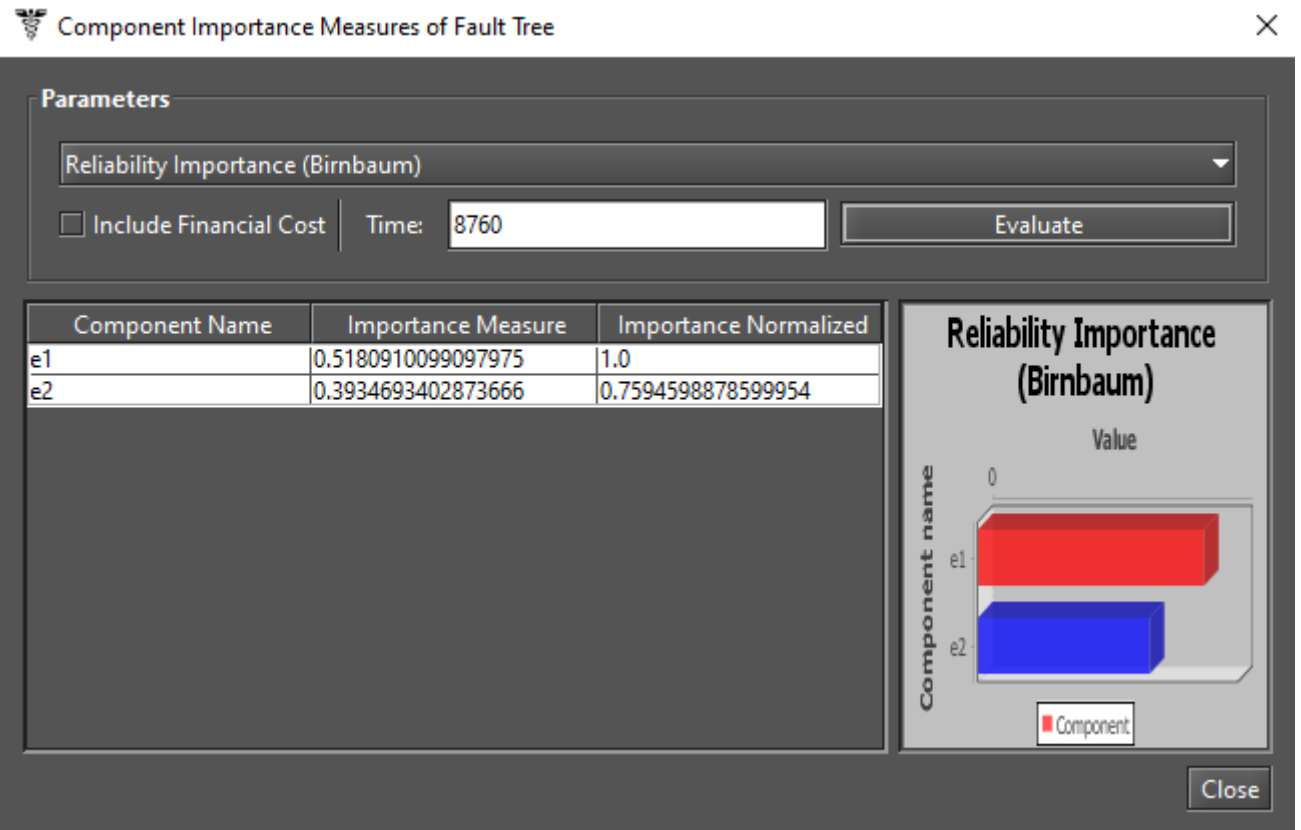


Figure 187: Reliability importance when replacing a component

We can see the criticality of “Sensor A” (e1) has increased. This means that, considering a time interval of 8760h, “Sensor B” is more critical for overall reliability when compared to the “Sensor C.”

4.1.5 Structural and Logical Functions

Mercury generates structural and logical functions of FT models. Both functions are a behavioral representation of the system represented by the model and are related to the states of each component. By applying them, it is possible to evaluate the impact on the system considering the components that have failed.

The system and its components should be in one of the following states: working (default) or failed. The state of the system is a binary random variable determined by the states of its components. If the state of each component is known, then the state of the system is also known.

The state can be toggled by accessing the properties of the component (see Figure 188). When the state of a component is failed, the component is depicted by a fire icon over the node, as already mentioned in this manual.

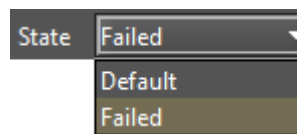


Figure 188: State

Now, let us demonstrate how to get these functions by using Mercury. Figure 189 shows a model having an AND gate and an OR gate. As we can see, there is a failed node in this model (event e3), and that node is child of the OR gate.

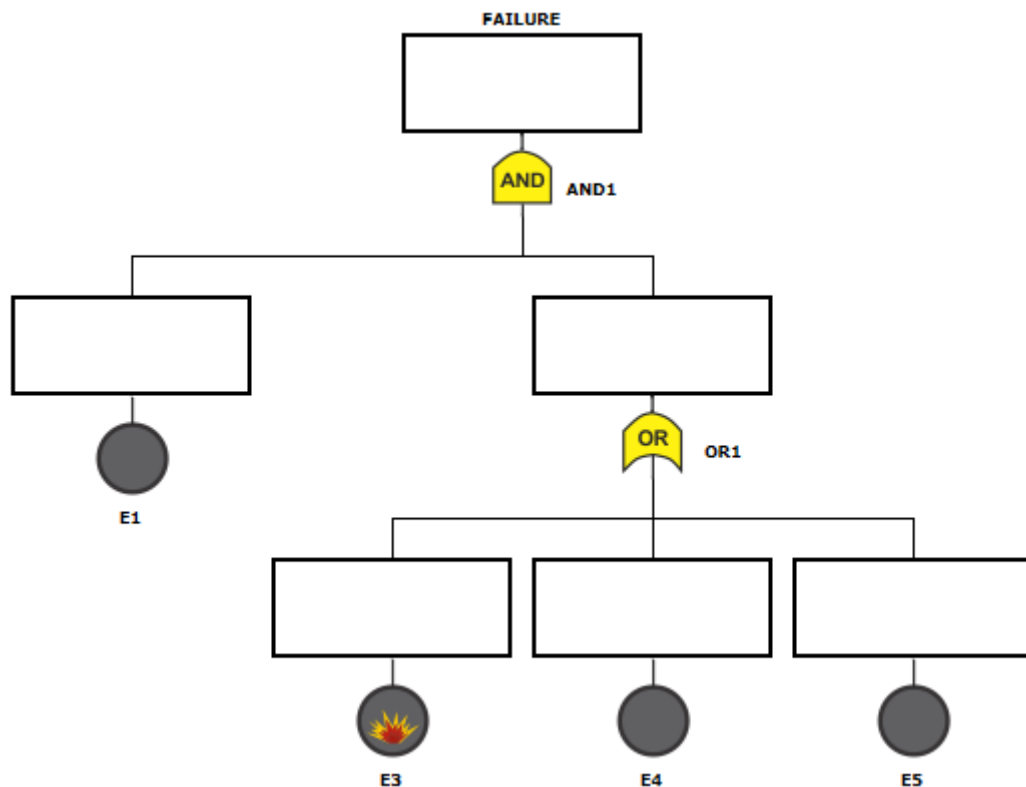


Figure 189: FT Model

The structural and logical functions can be accessed in the menu Evaluate -> FT Evaluation -> Get Functions. Figures 190 and 191 show the structural and logic functions of the FT model depicted above, respectively. In addition to the expressions, the tool indicates the event nodes marked as faulty (inoperative) and the current state of the system. By considering the example, the failed node (e3) does not impact the state of the system.

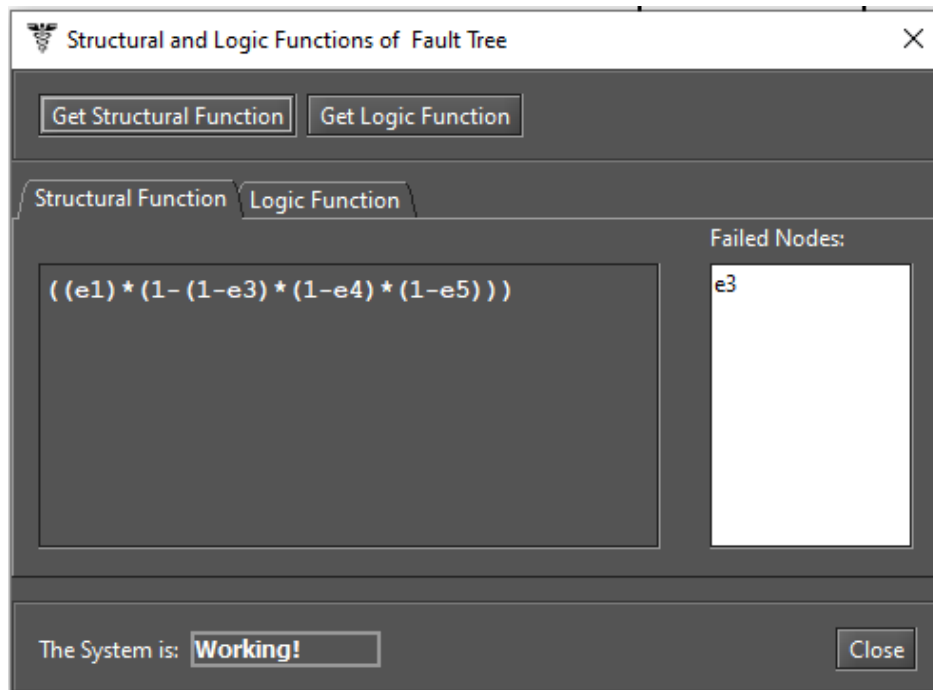


Figure 190: Structural Function

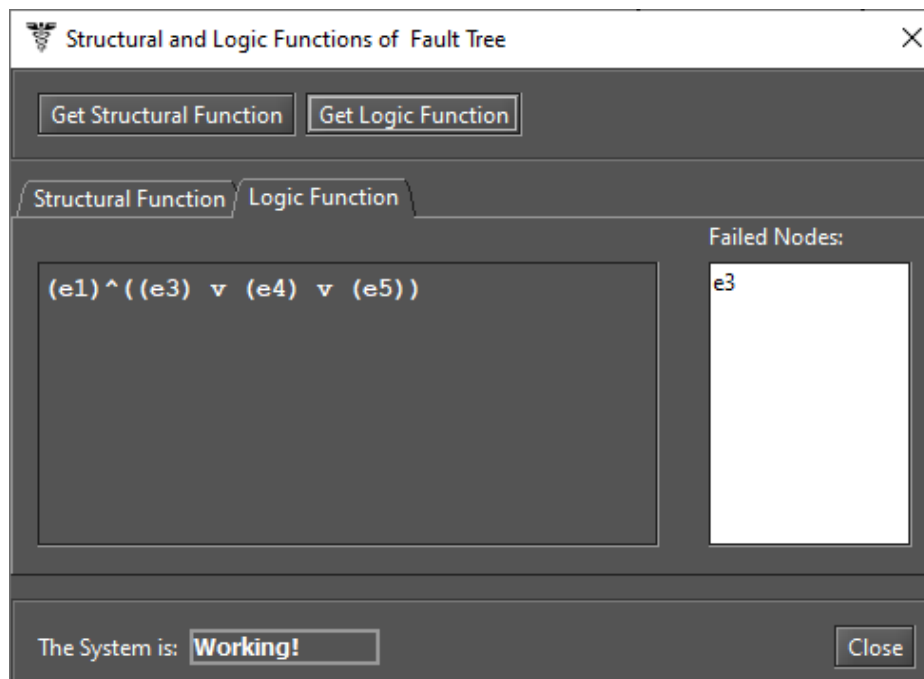


Figure 191: Logic Function

On the other hand, we can see that by changing the state of the event node e1 to failed (see Figure 192), it changes the state of the system to failed (see Figure 193).

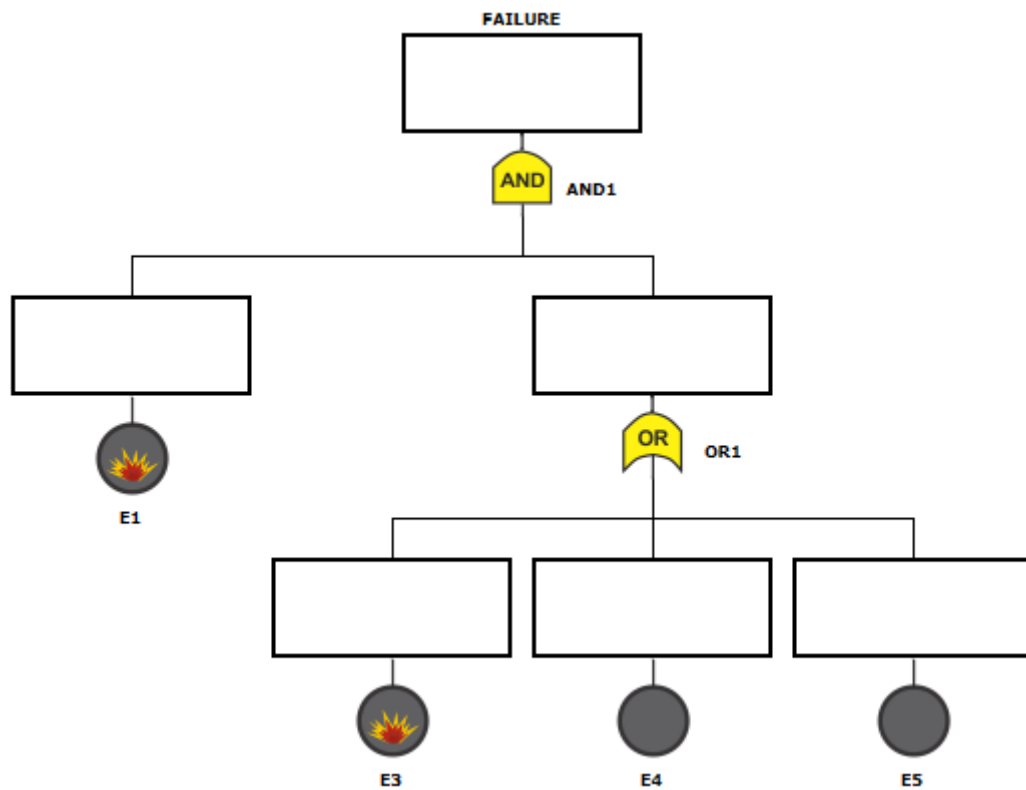


Figure 192: FT Model

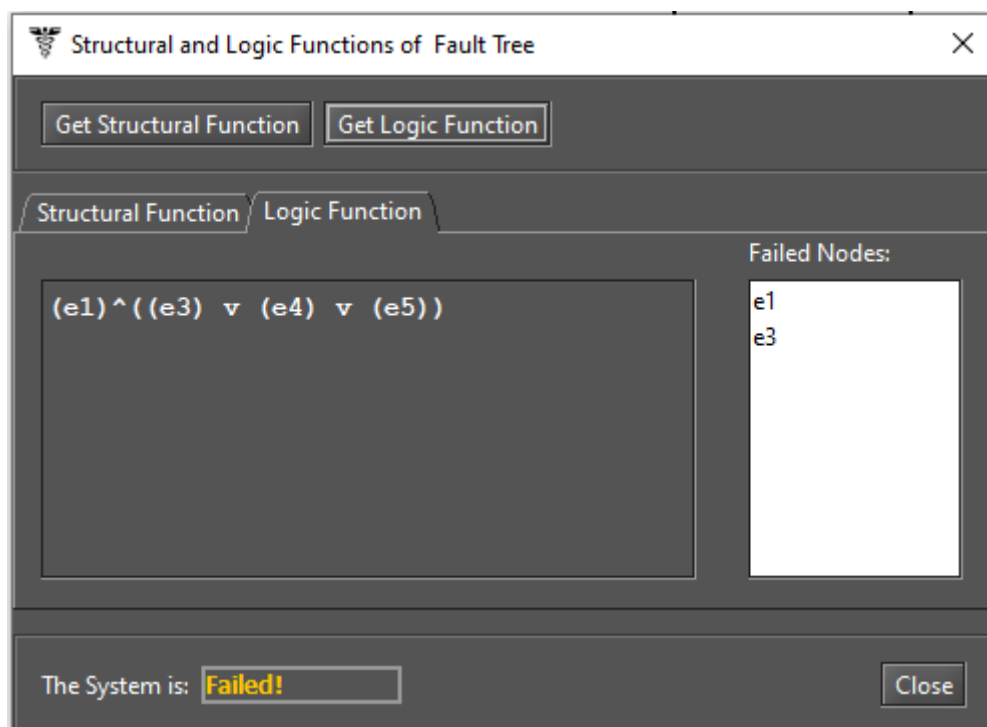


Figure 193: Logic Function and System State as Failed

4.1.6 Sensitivity Analysis

Mercury computes partial derivative sensitivity indices of FTs through sensitivity analysis. Those indices indicate the impact that every input parameter has on the availability of the model. Sensitivity analysis can be accessed in the menu Evaluate -> FT Evaluation -> Sensitivity Analysis. Figure 194 shows the “Sensitivity Analysis of FT” window, which shows the partial derivative of the structural equation for each parameter, as well as the sensitivity indices. Users can use the sensitivity analysis only if the model has exponential events.

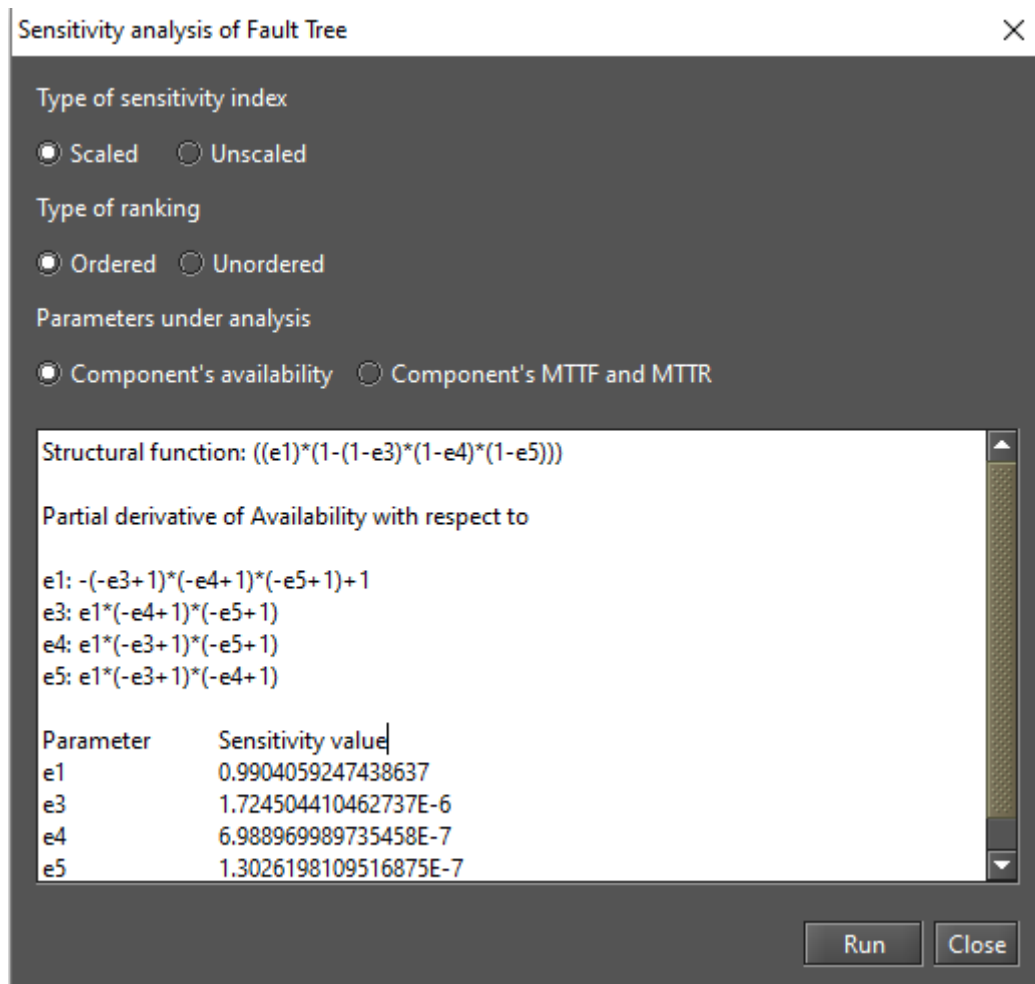


Figure 194: Sensitivity Analysis of Fault Tree

4.1.7 Export to RBD model

Users can convert FTs to RBDs. This can be done by accessing the menu Evaluate -> FT Evaluation -> Export to RBD model. The conversion process must be confirmed, as shown in Figure 195. Once done, the user must choose the location and enter the name of the file to be created.

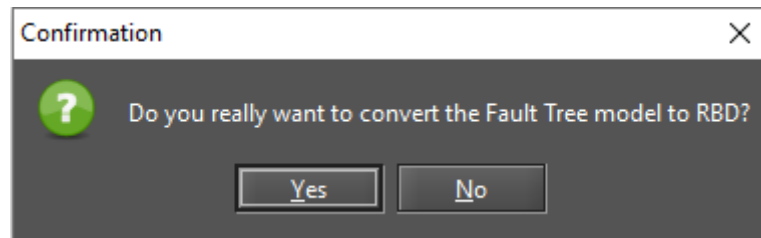


Figure 195: Converting FT to RBD

Figure 197 shows an RBD converted from the FT presented in Figure 196.

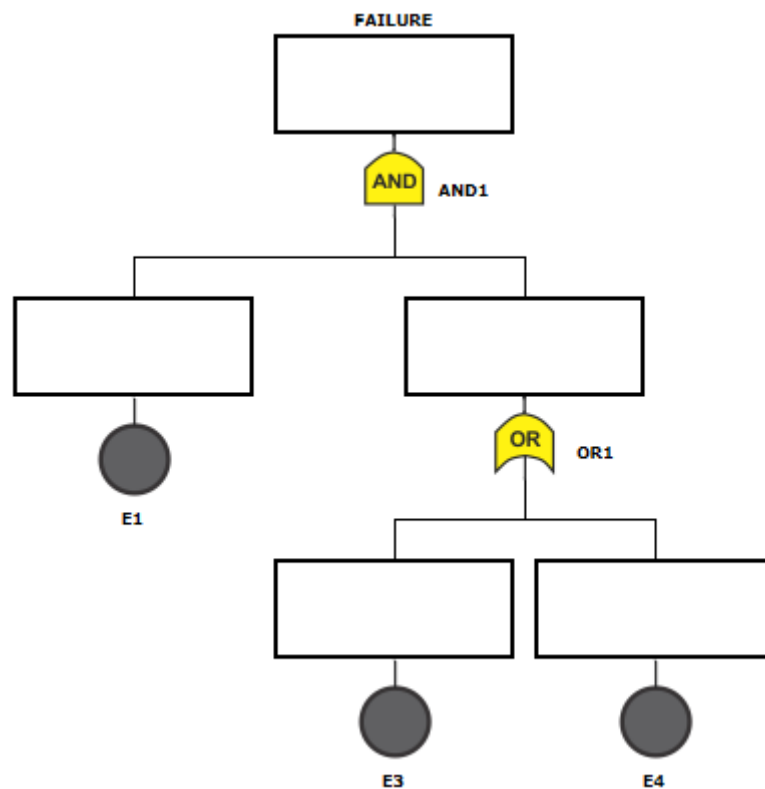


Figure 196: FT Model

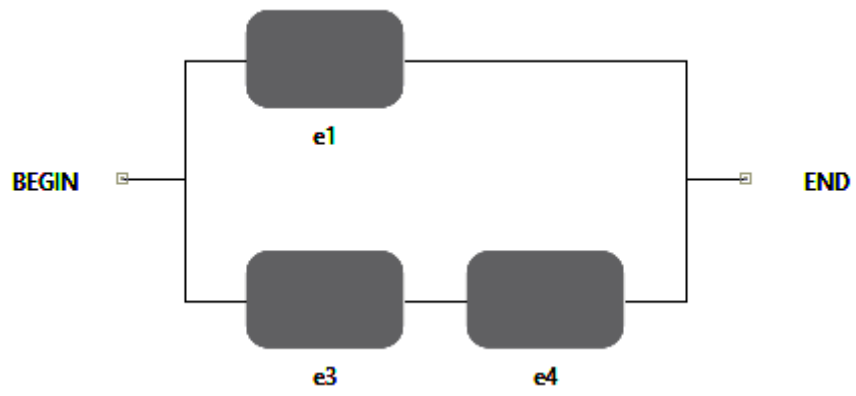


Figure 197: RBD generated from a FT Model

5 CTMC Modeling and Evaluation

The first step in order to start modeling CTMC models on Mercury is to put states into the graph. On the CTMC view, the user must click on the button “State”, available on the toolbar (see Figure 198), and then click on the desired location into the drawing area in order to create a state there.



Figure 198: Adding a CTMC State

After adding states, transitions between them are drawn by clicking on the center of the source state, only after cursor changes to a hand symbol, and then dragging the line until the target state, as depicted in Figure 199. After that, a directed arc is created between the two states as depicted in Figure 200.

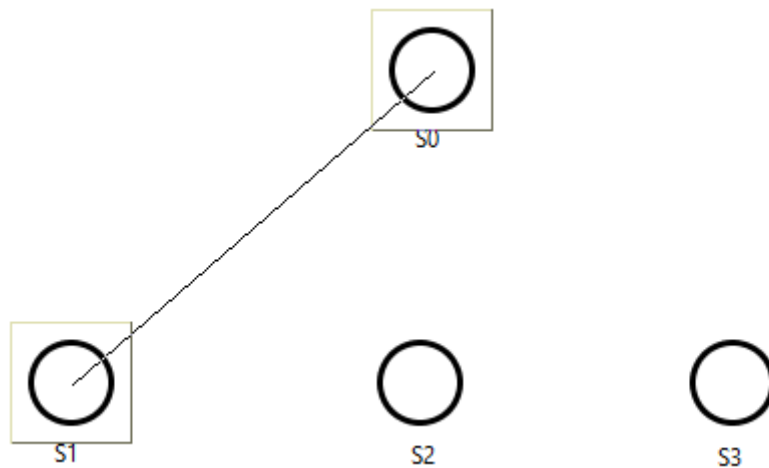


Figure 199: Adding a Transition Between States

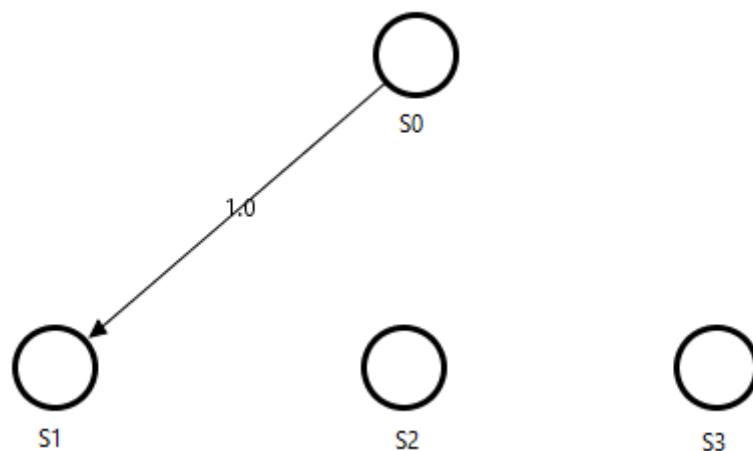


Figure 200: A Transition Between States

Users can define the rate of each transition by double-clicking on the respective arc or right-clicking on it and selecting “Properties.” Figure 201 shows the window where transition rate can be defined.



Figure 201: Defining a Rate of a State Transition

Mercury also allows us to assign reward rates to states. It is performed by double-clicking on the selected state, or right-clicking on it and selecting “Properties.” Figure 202 shows the window “State” where a reward rate can be assigned to a state. Default reward for each state is zero. It is possible to enter any real value or expression containing user-defined parameters on this window. The name of the state can also be changed.

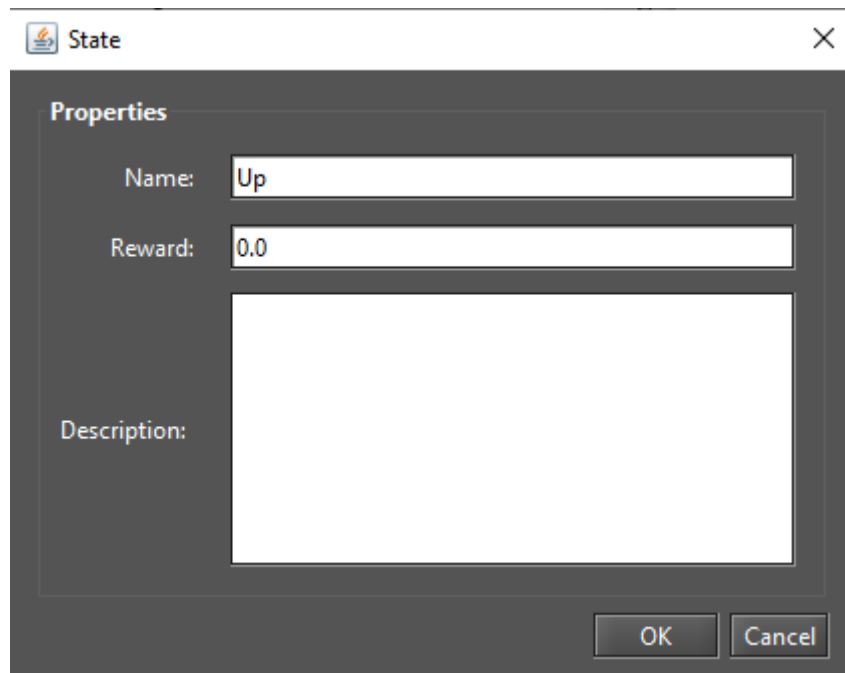


Figure 202: Modifying the Properties of a State

After all states and transitions have been properly defined (see Figure 203), stationary and transient analyses can be carried out.

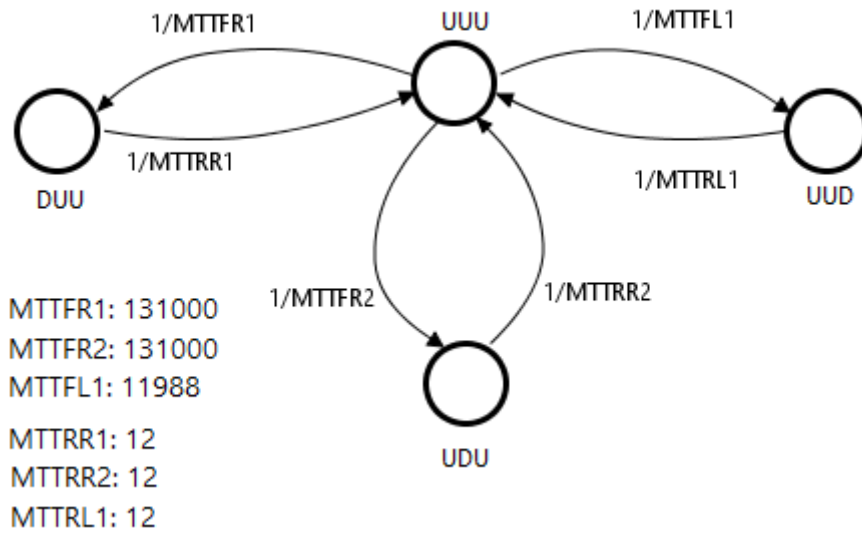


Figure 203: Example of CTMC Model

Users can export the infinitesimal generator matrix (transition rate matrix) of the model to a text file by clicking on the matrix icon on the toolbar, as shown in Figure 204.



Figure 204: Exporting the Transition Rate Matrix

Mercury has a feature for increasing the usability of the tool. Once a CTMC component has been inserted, it is possible to read its properties in the drawing area by positioning the mouse cursor over it. After that, a tooltip appears showing all properties of the component. As we can see by looking at Figure 205 that depicts a tooltip showing the properties of a state.

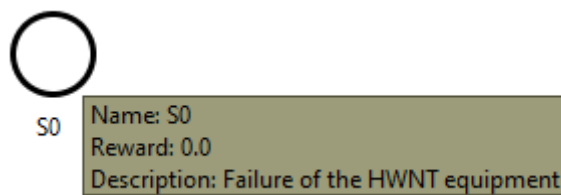


Figure 205: Tooltip for a CTMC State

5.1 Input Parameters/Definitions

Transition rates can be defined by using expressions containing both numbers and user-defined parameters. Button “Definition” on the toolbar is represented by a λ icon and it creates a symbolic parameter (see Figure 206).



Figure 206: Adding a CTMC Definition

After clicking on that button, the user must click on any point into the drawing area in order to put the definition there. Thus, a new parameter is created named **Param0** (or **Param1**, and so on, in case other parameters have been already created). By double-clicking on it or selecting “Properties” on the pop-up menu of the definition, it is possible to access its properties.

The name of the parameter can be defined by means of a combination of alphanumeric characters. Identifiers on Mercury must start with at least one alpha character. Special characters (e.g., a hyphen, or ampersand) are not allowed, except for underscores. In case names of Greek letters are used, Mercury converts them to the proper symbol on the Greek lowercase alphabet (see Figures 207 and 208). The value assigned to the parameter can be a numerical expression. Any symbols or parameter names are not allowed in the value field.

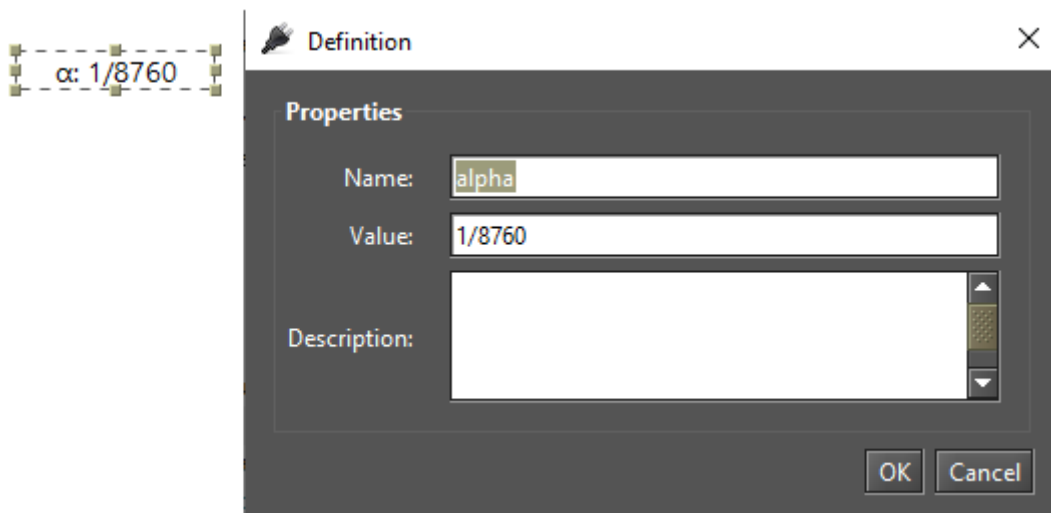


Figure 207: Modifying a CTMC Parameter

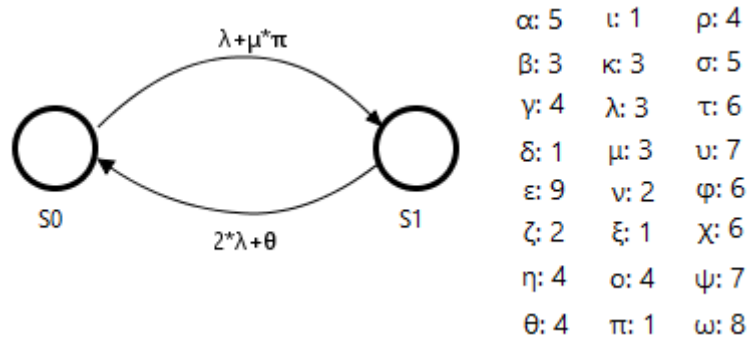


Figure 208: Parameters Named by Using Greek Letters

5.2 Metrics

Button “Metric” allows us to define metrics to extract some characteristic of the model (see Figure 209).



Figure 209: Adding a CTMC Metric

Once a metric has been inserted, users can change its name, description, and define the expression to compute its value. Syntax for metric expressions is based on state probabilities ($\mathbf{P}\{\textit{state-name}\}$) and rewards ($\mathbf{R}\{\textit{state-name}\}$). Probabilities and rewards for any states can be combined in the expression (i.e., added, subtracted, and so on). Considering the example depicted in Figure 210, the metric **AvB3** indicates the availability of the system (state “Up”) represented by the two-state model.

In this case, availability is computed considering the expression $\mathbf{P}\{\mathbf{Up}\}$ — it is the probability for remaining in the state **Up**. Once the model is evaluated by using stationary or transient analysis, metrics are updated in the drawing area accordingly (see Figure 211).

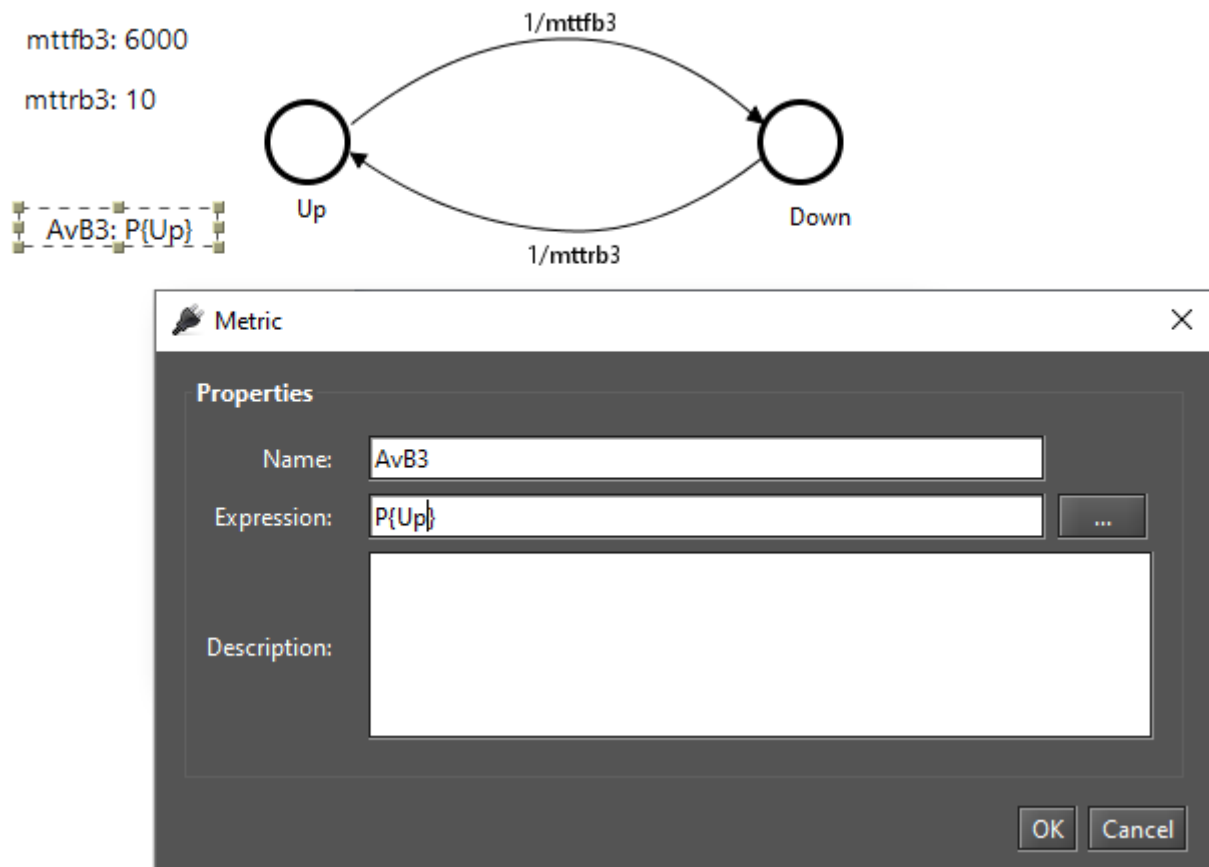


Figure 210: Defining Name and Expression for a Metric

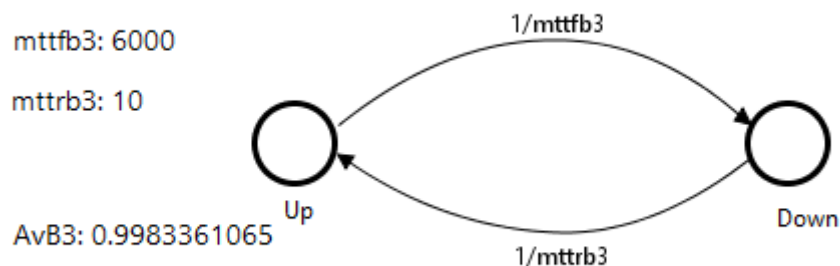


Figure 211: A CTMC Metric Solved

Reward rate can be obtained by using the metric expression $R\{ \}$, and mean time to absorption — when there is at least one absorbing state — can be computed by defining a metric having the expression $MTTA$.

Metric expressions are still visible in the “Metrics” group in the left-side panel on the CTMC tab (see Figure 212). That panel shows all components composing the CTMC model: states, parameters, metrics, and transitions. State transitions are represented in this panel by the source and target states followed by the expression or value assigned to that transition.

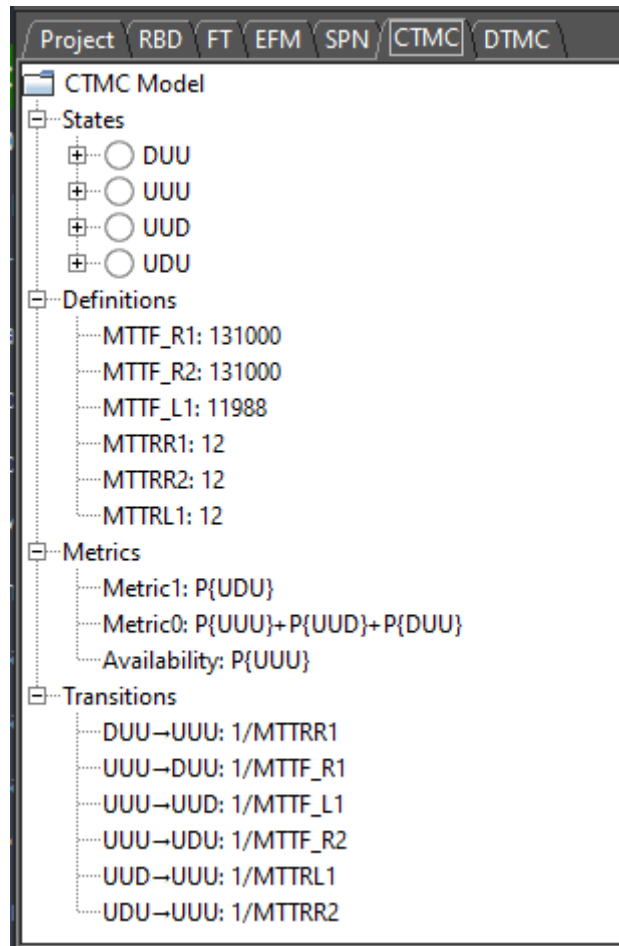


Figure 212: CTMC Panel on the Left Side of the Main Window

5.3 CTMC Evaluation

Mercury makes it possible to carry out a large number of evaluations on CTMCs. The tool provides three functionalities for CTMC evaluations: “Stationary Analysis”, “Transient Analysis”, and “Sensitivity Analysis.” These evaluations are available in menu Evaluate -> CTMC Evaluation (see Figure 213). In the next subsections, we will introduce each of them.

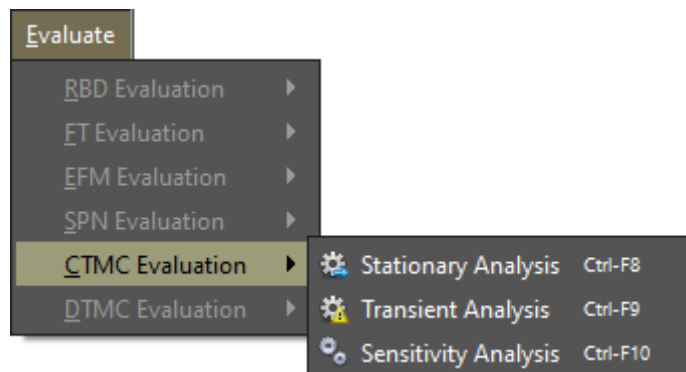


Figure 213: CTMC Evaluation Menu

5.3.1 CTMC Stationary Analysis

Stationary analysis computes steady-state probabilities, useful for evaluating the long-term average behavior of modeled systems. Figure 214 shows the window “Stationary Analysis”, which has a combo box for selecting one of two supported solution methods: **Direct - GTH** (Grassmann-Taksar-Heyman) and **Iterative - Gauss-Seidel**.

When solving CTMCs through GTH, it is possible to change the **maximum error** used in the algorithm. Default value for the maximum error is 0.0000001 (10^{-7}). By clicking on button “Run”, the solution algorithm is triggered and as soon as it finishes, results are presented in the text area at the bottom of the window (see Listing 7), and written in a plain text file with the name of the project file appended with the “-StationaryAnalysis.txt” suffix.

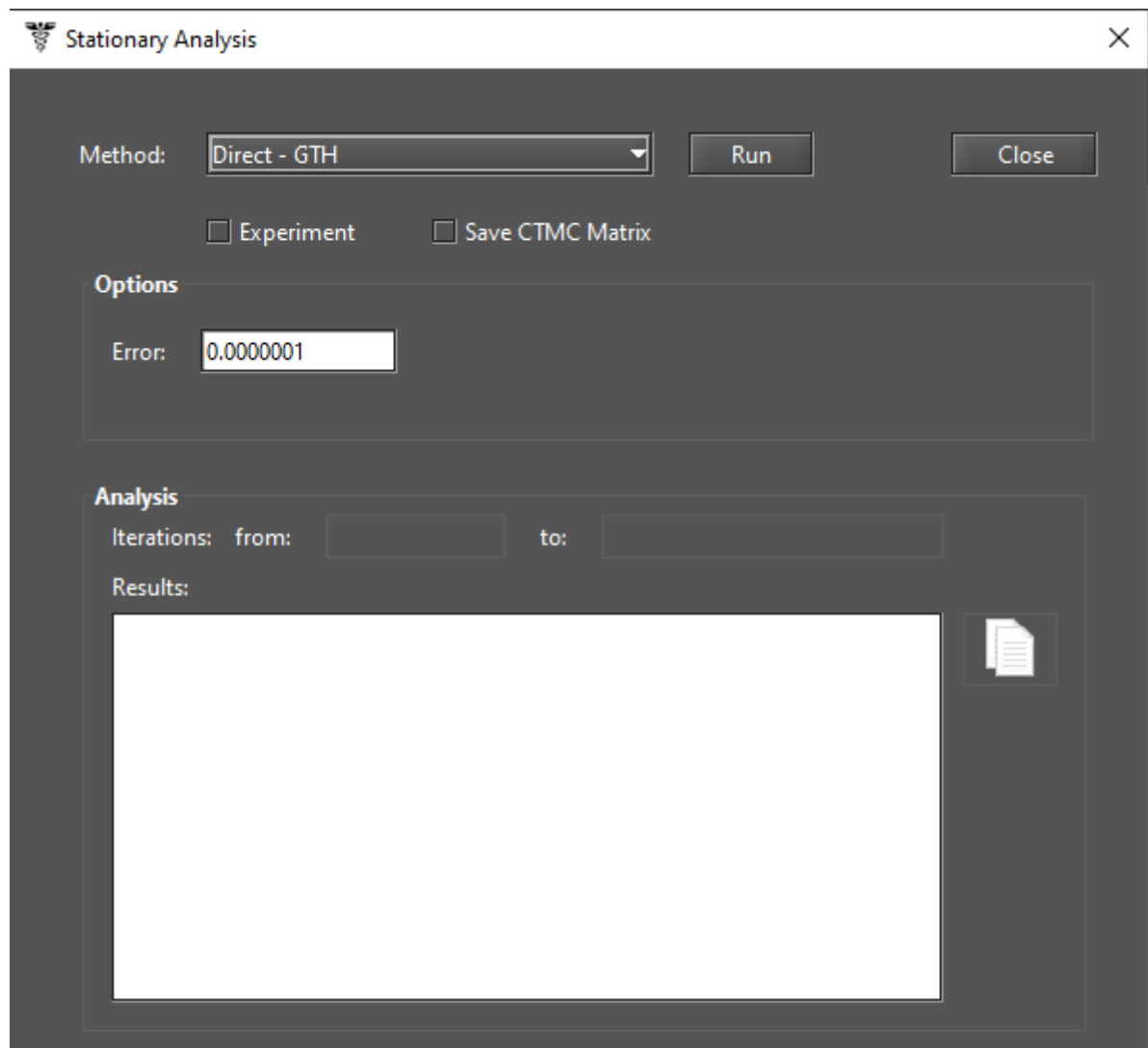


Figure 214: Stationary Analysis Window

Listing 7: Stationary Analysis of a CTMC

```

Mon Aug 14 13:09:56 BRT 2020

Performing stationary analysis...

Done! (elapsed time: 1s )

Matrix Q has been written into the file :

C:\Users\Thiago\Chapter_CTMC_Model1-MatrixQ.txt

#####

DUU=9.149470622218616E-5
UUU=0.9988171936427845
UUD=9.998170168890783E-4
UDU=9.149463410419709E-5

----- Metrics -----

Availability=0.9988171936427845

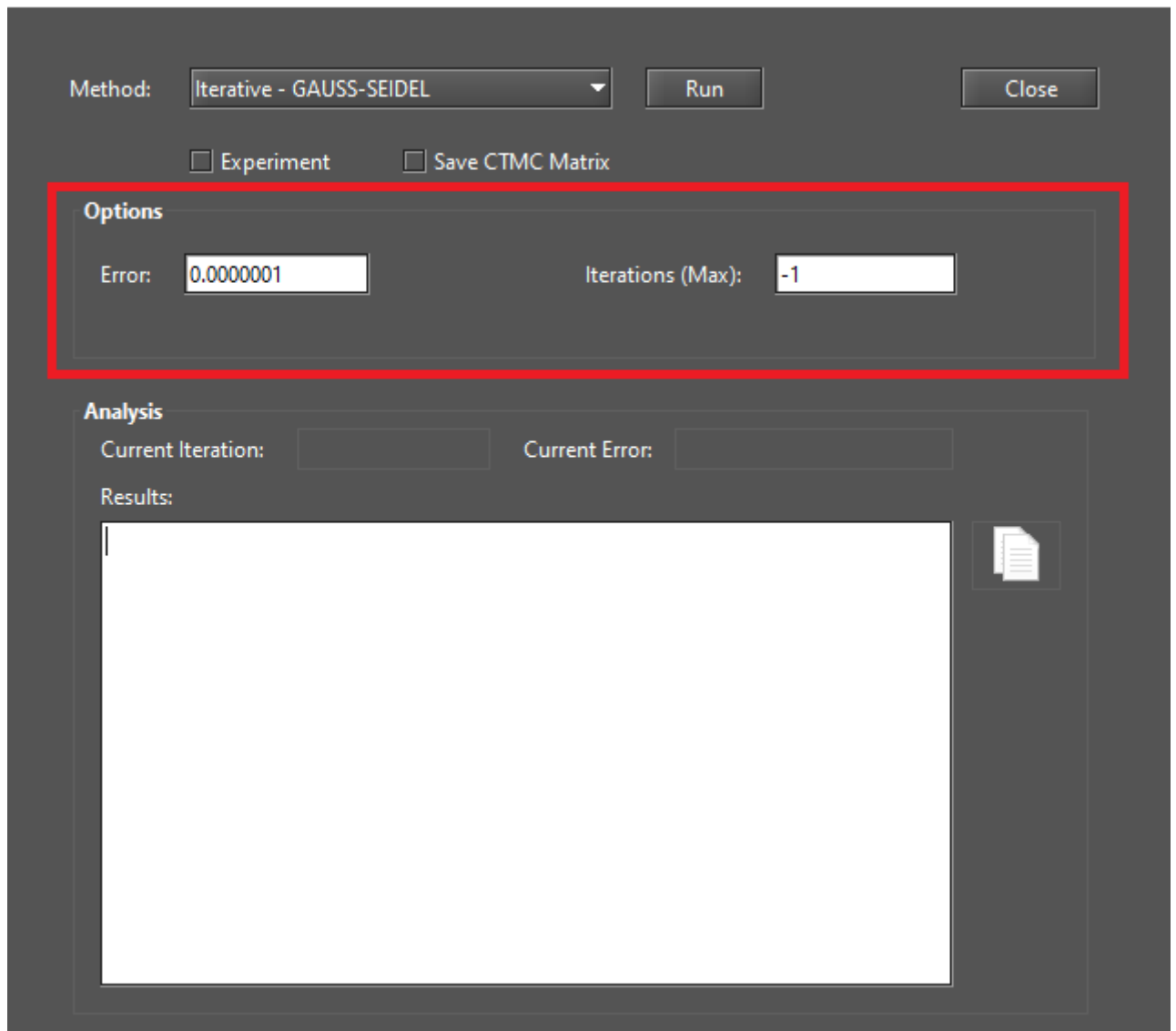
-----

Results have been written into the file :

C:\Users\Thiago\Chapter_CTMC_Model1-StationaryAnalysis.txt

```

When solving CTMCs through Gauss-Seidel, besides the maximum error, it is possible to change the maximum number of iterations. Default value for such a parameter is “-1”, indicating the algorithm only stops when the convergence of results is reached, considering the entered error (see Figure 215).



The image shows a software window titled "Stationary Analysis" with a close button (✕) in the top right corner. The window has a dark gray background. At the top, there is a "Method:" dropdown menu set to "Iterative - GAUSS-SEIDEL", a "Run" button, and a "Close" button. Below these are two checkboxes: "Experiment" and "Save CTMC Matrix", both of which are unchecked. A red rectangular box highlights the "Options" section, which contains two input fields: "Error:" with the value "0.0000001" and "Iterations (Max):" with the value "-1". Below the "Options" section is the "Analysis" section, which includes "Current Iteration:" and "Current Error:" labels followed by empty input boxes. Below these is a "Results:" label followed by a large, empty white rectangular area. To the right of this area is a small icon of a document with lines, representing a file or export function.

Figure 215: Stationary Analysis Window - Gauss-Seidel Method

Metrics are updated in the drawing area as soon as the analysis has been completed (see Figure 216 where the metric is on the left side of the model: "Availability").

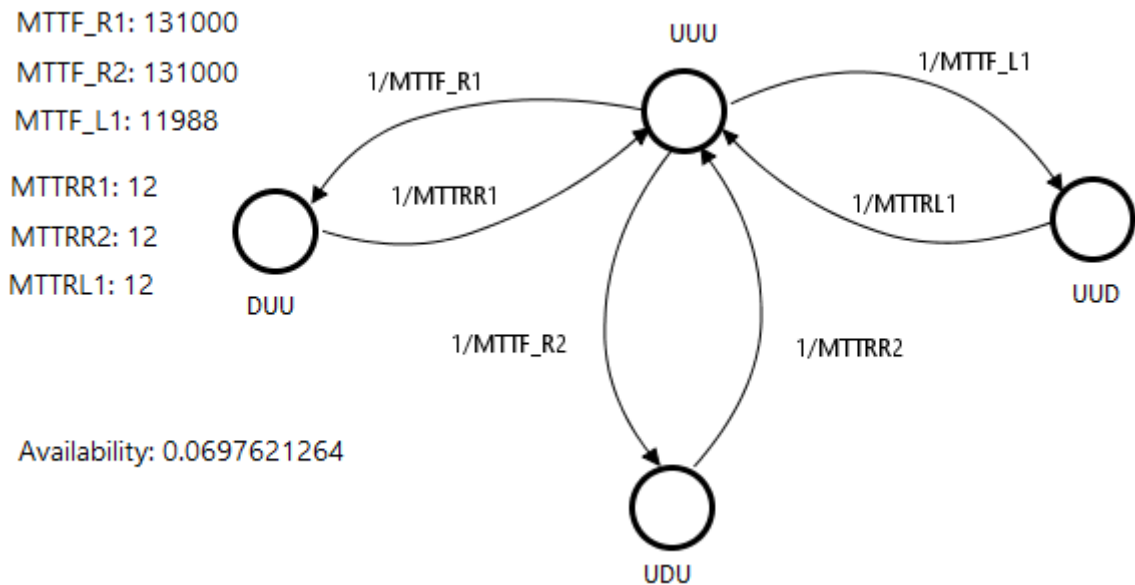


Figure 216: CTMC Metrics Updated

CTMC models can also be solved for a range of values of an user-defined parameter. This is accomplished by checking “Experiment” and clicking on the button “Run” on the window “Stationary Analysis”. A new window is displayed in order for the user to set the input parameters for the experiment (see Figure 217).

Options for Experiment

Options

Parameter: MTTF_R1

Minimum Value: 0.0 Maximum Value: 0.0

Interval (step size): 0.0

Metric: Availability

OK Cancel

Figure 217: CTMC Experiment

Below, we describe each field on this window.

- **Varying Parameter.** Parameter to have its value changed.
- **Output Measure.** Metric to be evaluated.
- **Range Minimum Value.** Initial value to be assigned to the selected parameter.
- **Range Maximum Value.** Final value to be assigned to the selected parameter.

- **Interval.** It is the step-size for changing the value of the parameter. Parameter starts with the minimum value and its value is increased considering the entered interval. At each change, the selected metric is evaluated. The experiment finishes when the maximum value for the parameter is reached.

At the end of the experiment, results are shown and a graph is plotted, as we can see by looking at Figures 218 and 219, respectively.

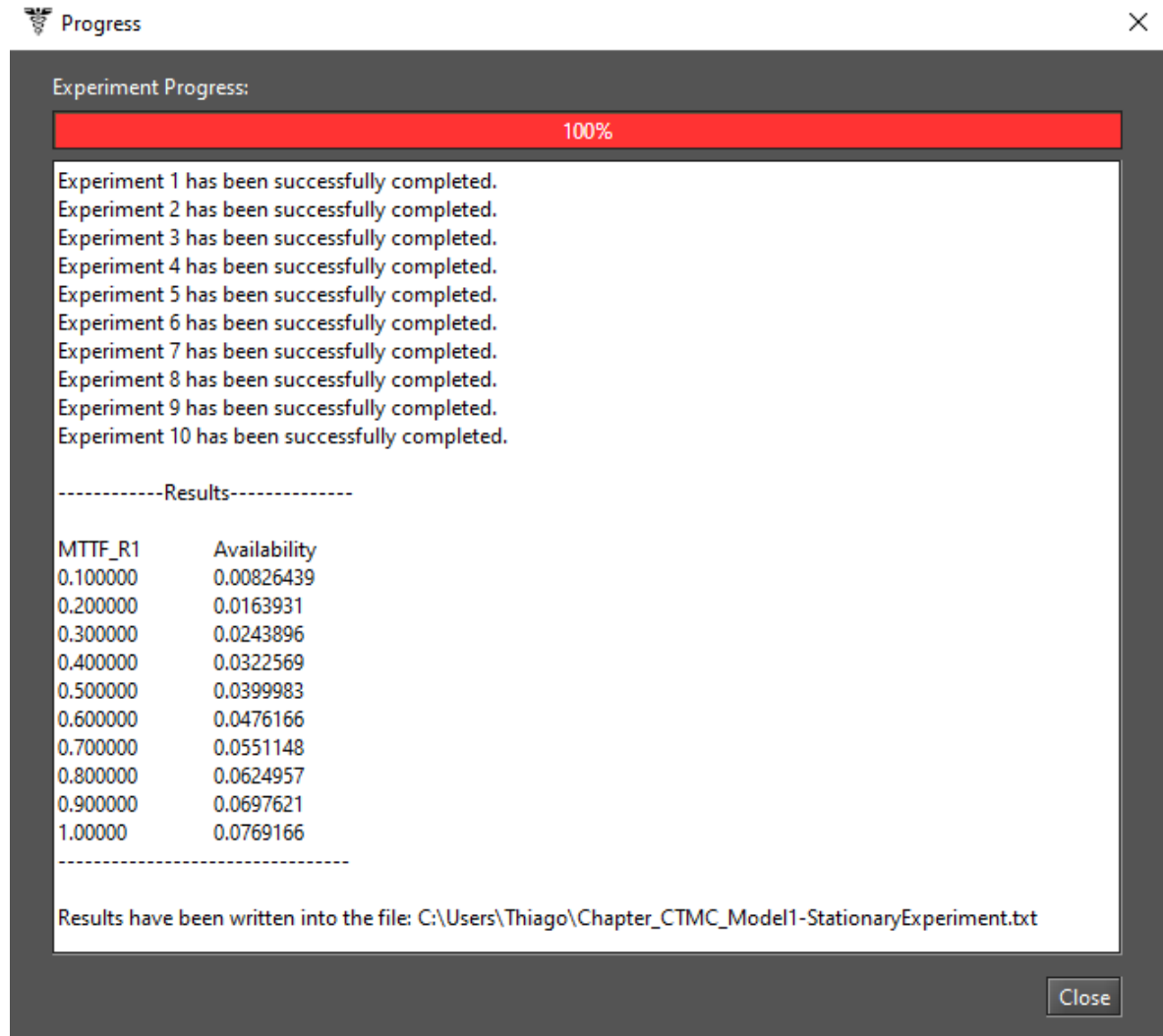


Figure 218: Results of a CTMC Experiment

Another option available on the window “Stationary Analysis” allows us to save the CTMC matrix to a file. It is written into a plain text file with the name of the project file appended with the “-MatrixQ.txt” suffix.

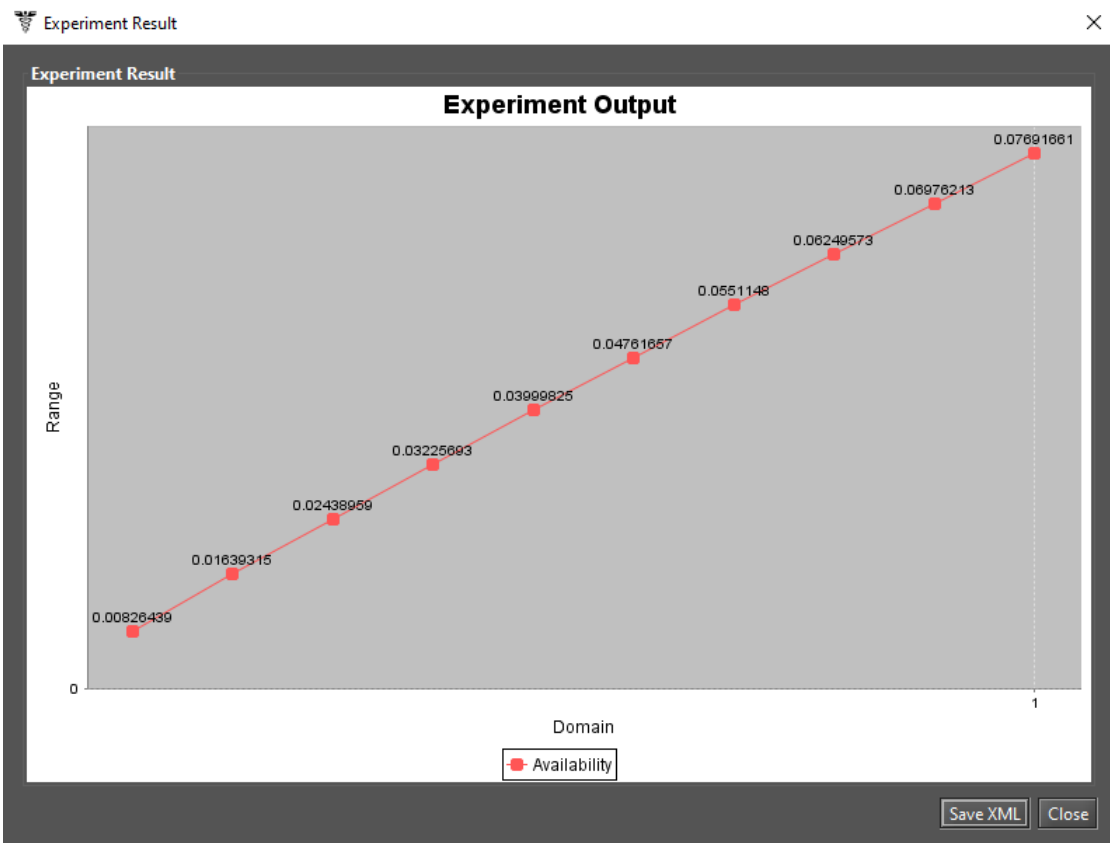


Figure 219: Graph of a CTMC Experiment

5.3.2 CTMC Transient Analysis

Transient analysis computes time-dependent probabilities, useful for evaluating the behavior of modeled systems at a particular point in time. Figure 220 shows the window “Transient Analysis”, which has a input field for selecting one of the two solution methods available: **Uniformization** (also known as Jensen’s method) and **Runge-Kutta (4th order)**.

When solving transient metrics of CTMCs, the user can define:

- **Time** for which the analysis will be carried out (default: 100).
- **Precision** of results (default: 10^{-7}).
- **Initial state probabilities** (default: 1 for the initial state, 0 for the other states). These probabilities are defined by clicking on button “Set Initial State Probability” (see Figure 221).

Transient Analysis
×

Method:

Uniformization

Uniformization
Runge-Kutta(4th order)

Run
Close

☐ Save C
☐ (ure)
☐ Absorption Probability

Options

Time:

100

Precision:

0.0000001

Set Initial State Probability

Output:
☒ Point
☐ Curve

Analysis

Current Time:

0

N. of iterations for a step:

Results:

Figure 220: Transient Analysis Window

Initial State Probability
×

Probabilities

| State | Initial Probability |
|-------|---------------------|
| UUU | 1.0 |
| DUU | 0.0 |
| UDU | 0.0 |
| UUD | 0.0 |

Ok

Cancel

Figure 221: Initial State Probability Window

When selecting Uniformization method, note that the time required for obtaining results is proportional to the time entered for the analysis because Uniformization is an iterative algorithm.

By clicking on button “Run”, the solution algorithm is triggered. As soon as it finishes, results are presented in the text area at the bottom of the window “Transient Analysis”, also they are written in a plain text file having the filename of the project appended with the “-TransientAnalysis.txt” suffix.

This window also allows us to choose between a **Point** or **Curve** analysis. **Point** analysis is the default, and it shows state probabilities only for the specific point in time. **Curve** analysis writes in a plain text file all state probabilities computed from time equals zero until the specified time.

Mean time to absorption (**MTTA**) is a metric that can be computed by checking “Mean Time to Absorption (failure).” MTTA is presented after the state probabilities in the “Results” text area. For MTTA calculation, the user might also define a metric with the expression **MTTA**. “Absorption Probability” for each state is another metric that is available in transient analysis.

5.3.3 Sensitivity Analysis

Sensitivity Analysis enables computing partial derivative sensitivity indices for CTMCs. Those indices indicate the impact that every input parameter has on a metric. This function is available by accessing menu Evaluate -> CTMC Evaluation -> Sensitivity Analysis. Figure 222 depicts a CTMC that represents the availability of a network comprising two routers and one link.

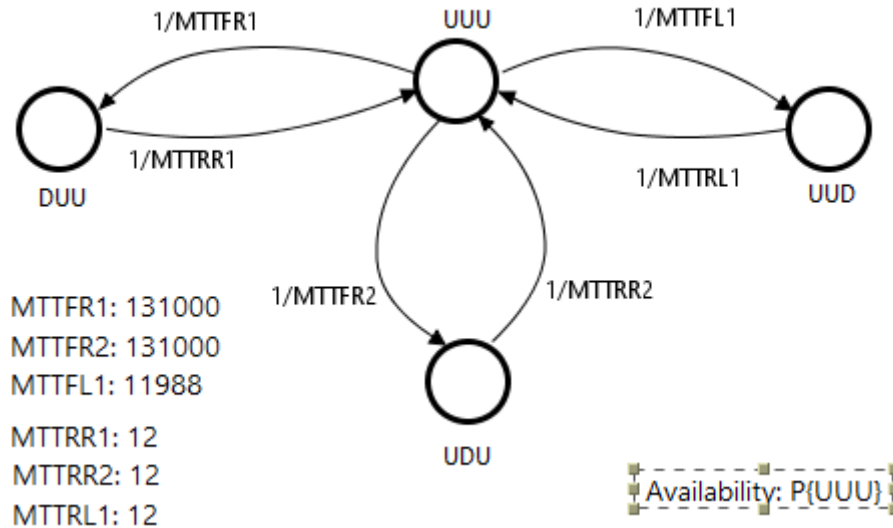


Figure 222: CTMC Model for a Network

This model was proposed in [4], and it has six parameters that affect system availability. Those parameters are mean time to failure (MTTF) and mean time to repair (MTTR) of each component: router 1 (R1), router 2 (R2), and link (L1). Their sensitivity ranking was computed by using Mercury as illustrated by Figure 223.

The window “Sensitivity Analysis” has four options:

- **Type of sensitivity index** can be scaled or unscaled. When the user chooses scaled indices, every partial derivative is multiplied by the ratio of the 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 [4] and [5].
- **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 the chosen metric.
- **Measure of interest** can be any user-defined CTMC measure, for which the user is interested in assessing sensitivity to input parameters. Please notice that if no measure has been defined, no sensitivity analysis can be carried out. All measures can be evaluated at once.
- **Parameter of interest** can be any parameter in the model. The user must choose between viewing the sensitivity of the chosen measure with respect to just one parameter, or to all parameters.

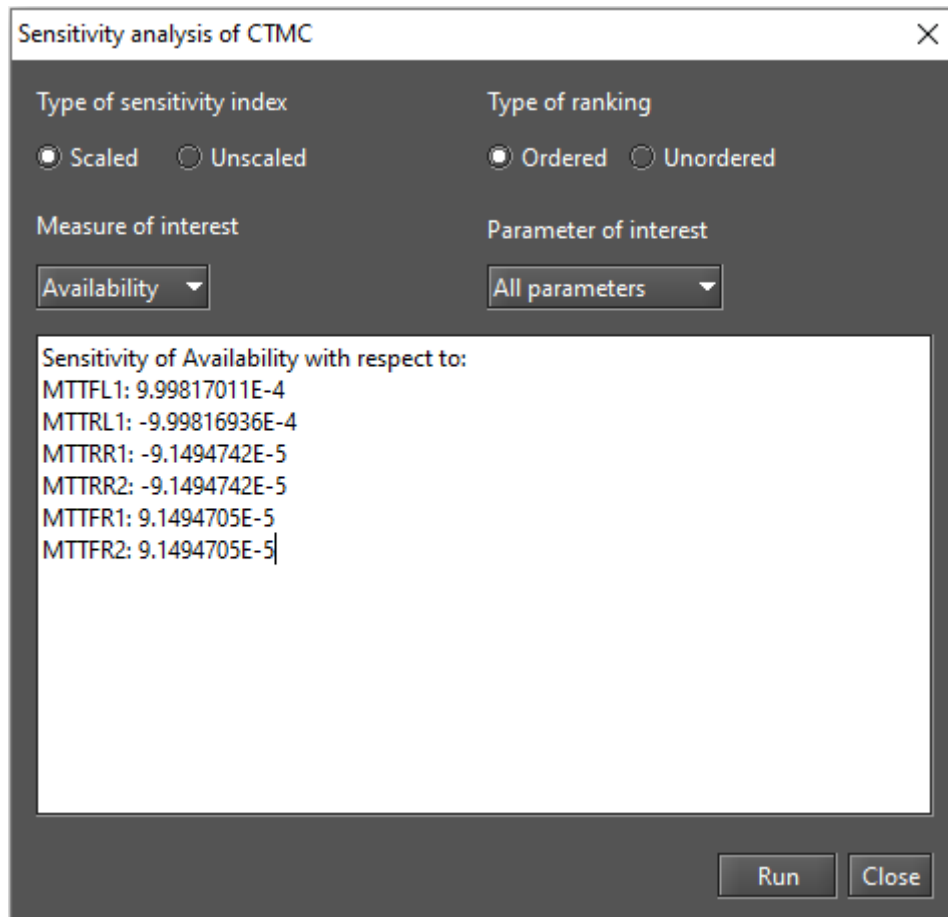


Figure 223: Results of Sensitivity Analysis for a CTMC

6 DTMC Modeling and Evaluation

The first step in order to start modeling DTMC models on Mercury is to put states into the graph. On the DTMC view, the user must click on the button “State”, available on the toolbar (see Figure 224), and then click on the desired location into the drawing area in order to create states there.



Figure 224: Adding a DTMC State

Transitions between them are drawn by clicking on the center of the source state, only after cursor changes to a hand symbol, and then dragging the line until the target state, as depicted in Figure 225. After that, a directed arc is created between the two states as depicted in Figure 226.

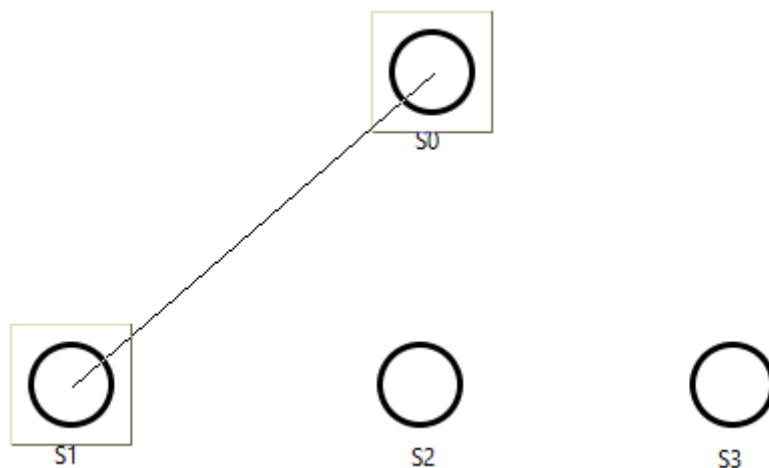


Figure 225: Adding a Transition Between States

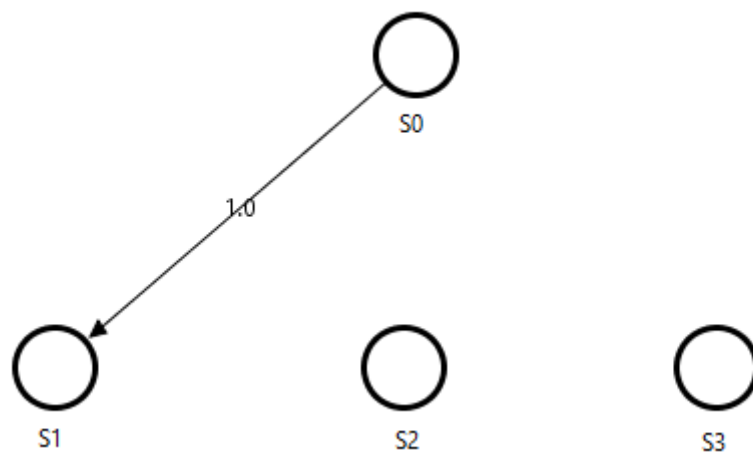


Figure 226: A Transition Between States

It is possible to define the probability of remaining in the current state once it has been reached by using self-loops. A self-loop for a state is defined by right-clicking on the state and selecting “Self-Loop”, as depicted in Figure 227.

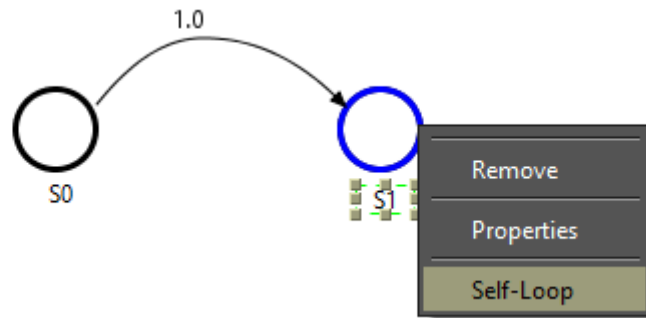


Figure 227: Defining a Self-Loop to a State

After that, a self-loop arc is drawn for the state, as we can see in Figure 228. The left-side panel on the DTMC tab is updated accordingly.

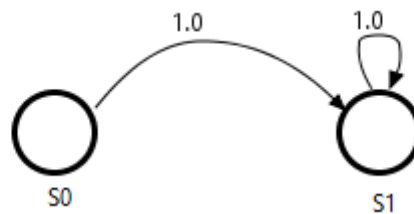


Figure 228: State with a Self-Loop Transition

The user can define probability for each transition by double-clicking on the respective arc or right-clicking on it and choosing “Properties”. Figure 229 shows the window which a transition probability can be defined.



Figure 229: State Transition Probability

Figure 230 shows a DTMC model having some parameters and a metric defined. By defining all states and transitions properly, it is possible to carry out stationary and transient analyses.

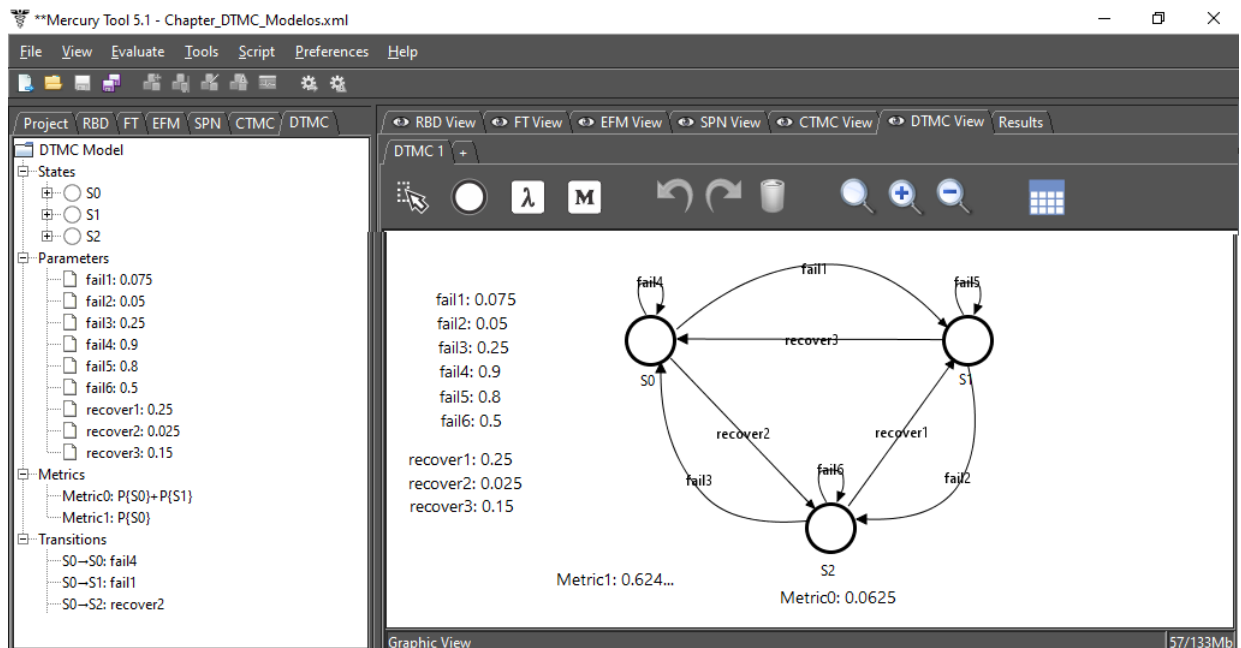


Figure 230: Example of DTMC Model

Users can export the probability matrix of a model to a text file, by clicking on the button represented by a matrix icon on the toolbar as shown in Figure 231.



Figure 231: Exporting the DTMC Probability Matrix

It is worth mentioning that when modeling DTMCs, the sum of probabilities for all output arcs for each state must be equal to one. Otherwise, it is not possible to carry out evaluations on the model. Figure 232 shows an error occurred when this condition is not met.

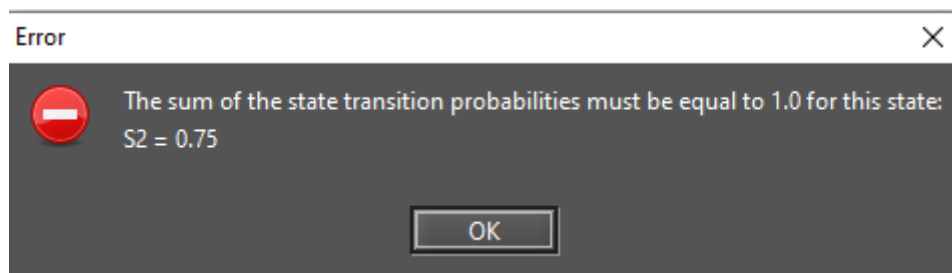


Figure 232: Transition Probabilities Error

Mercury has a feature for increasing the usability of the tool itself. Once a DTMC component has been inserted, it is possible to read its properties in the drawing area by positioning the mouse cursor over it. After that, a tooltip appears showing all properties of the component. As we can see by looking at Figure 233, all properties of a state appears in the tooltip.

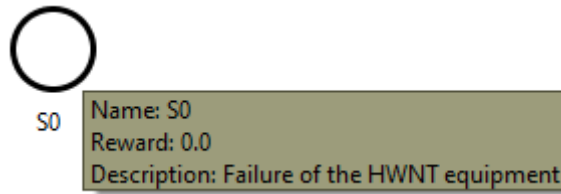


Figure 233: Tooltip for a DTMC State

6.1 Input Parameters

Transition probabilities can be defined by using expressions containing both numbers and user-defined parameters. Button “Definition” on the toolbar is represented by a λ icon, and it creates a symbolic parameter (see Figure 234).



Figure 234: Adding a DTMC Definition

By clicking on that button, the user must click on any point into the drawing area to put the definition there. Thus, a new parameter is created named **Param0** (or **Param1**, and so on, in case other parameters have been already created). By double-clicking on it or selecting “Properties” on the pop-up menu of the definition, it is possible to access its properties.

The name of the parameter can be defined by means of a combination of alphanumerical characters. Identifiers on Mercury must start with at least one alpha character. Special characters (e.g., a hyphen, or ampersand) are not allowed, except for underscores. In case names of Greek letters are used, Mercury converts them to the proper symbol on the Greek lowercase alphabet (see Figures 235 and 236). The value assigned to the parameter can be a numerical expression. Any symbols or parameter names are not allowed in the value field.

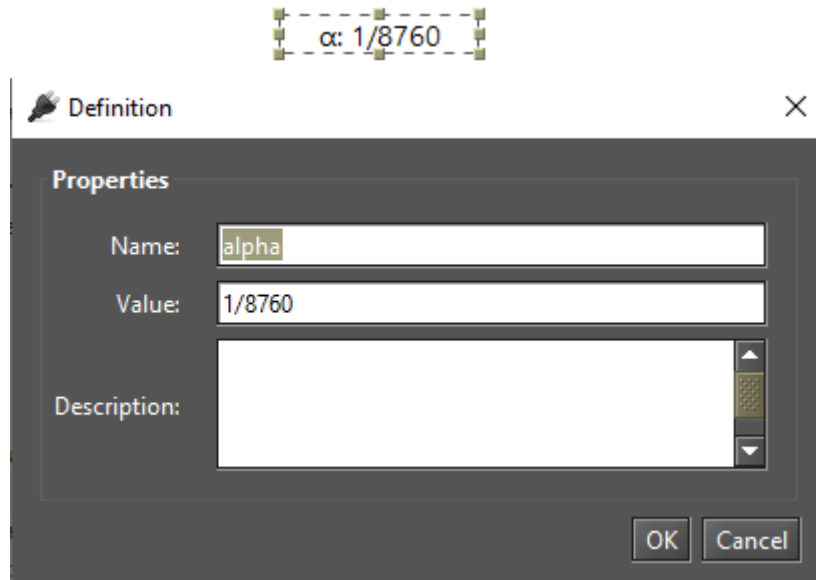


Figure 235: Modifying a DTMC Parameter

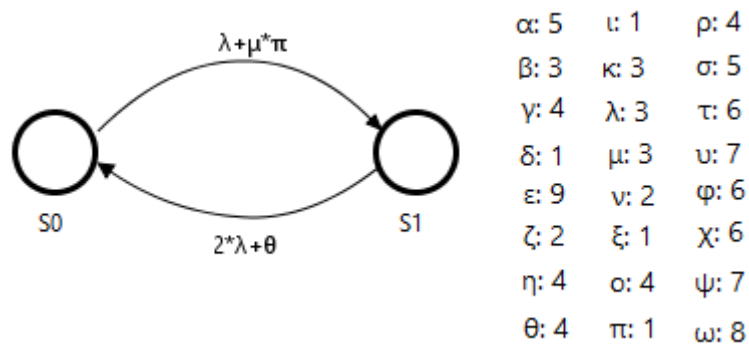


Figure 236: Parameters Named by Using Greek Letters

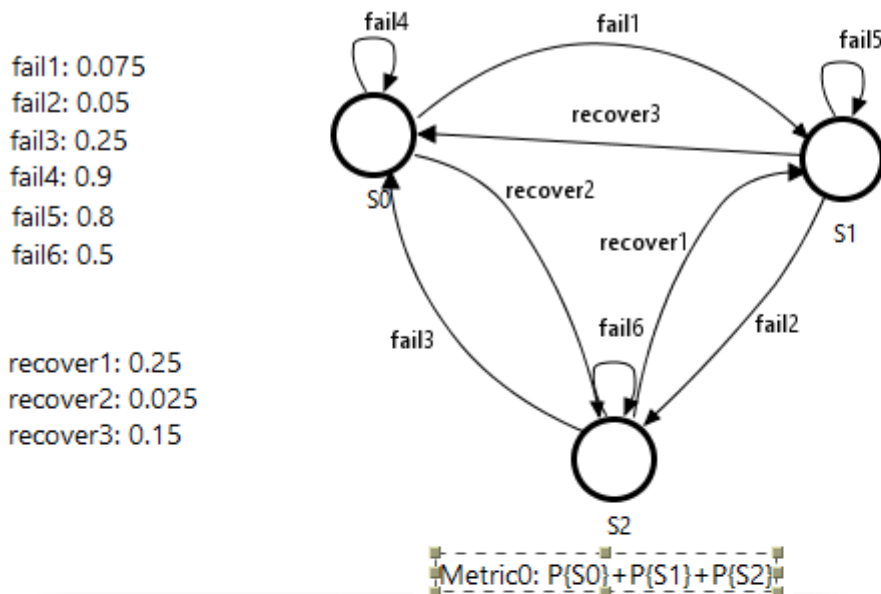
6.2 Metrics

Button “Metrics” allows us to define metrics in order to evaluate some characteristics of the model (see Figure 237).



Figure 237: Adding a DTMC Metric

Once inserting a metric, users can change its name, description, and define the expression to compute its value. Syntax for metric expressions is based on state probabilities ($P\{\text{state-name}\}$). Probabilities for any states can be combined in the expression (i.e., added, subtracted, and so on). By considering the example depicted in Figure 238, metric **Prob** indicates the probability for the system to remain in states “S0”, “S1”, and “S2” which is 1.



Metric

Properties

Name:

Expression: ...

Value:

Description:

OK Cancel

Figure 238: Defining Name and Expression for a Metric

Prob will be computed by the expression “ $P\{S0\}+P\{S1\}+P\{S2\}$ ” — it represents the probability of staying in states “S0”, “S1”, and “S2”. Once the model is evaluated by using stationary or transient analysis, metrics are updated in the drawing area accordingly (see Figure 239).

Metric expressions are still visible in the group “Metrics” in the left-side panel on the DTMC tab (see Figure 240). That panel shows all components composing the DTMC model: states, parameters, metrics, and transitions. State transitions are represented by the source and target states followed by the expression or value assigned for that transition.

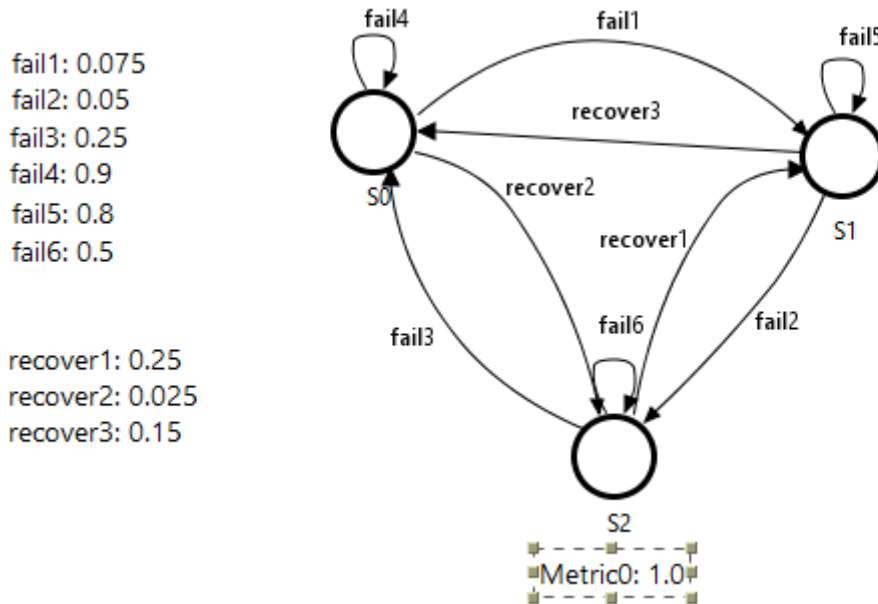


Figure 239: A Metric Solved

6.3 DTMC Evaluation

Mercury makes it possible to carry out a large number of evaluations on DTMCs. The tool provides two types of evaluations for DTMCs: “Stationary Analysis” and “Transient Analysis”. These evaluations are available in the menu Evaluate -> DTMC Evaluation (see Figure 241). In the next subsections, we will introduce each of them.

6.3.1 DTMC Stationary Analysis

Stationary Analysis computes steady-state probabilities, useful for evaluating the long-term average behavior of modeled systems. Figure 242 shows the window “Stationary Analysis”, which has a combo box for selecting one of two supported solution methods: **Direct - GTH** (Grassmann-Taksar-Heyman) and **Iterative - Gauss-Seidel**.

When solving DTMCs through GTH, it is possible to change the **maximum error** used in the algorithm. Default value for the maximum error is 0.0000001 (10^{-7}). By clicking on button “Run”, the solution algorithm is triggered and as soon as it finishes, results are presented in the text area at the bottom of the window (see Listing 8), and written in a plain text file with the name of the project file appended with the “-StationaryAnalysis.txt” suffix.

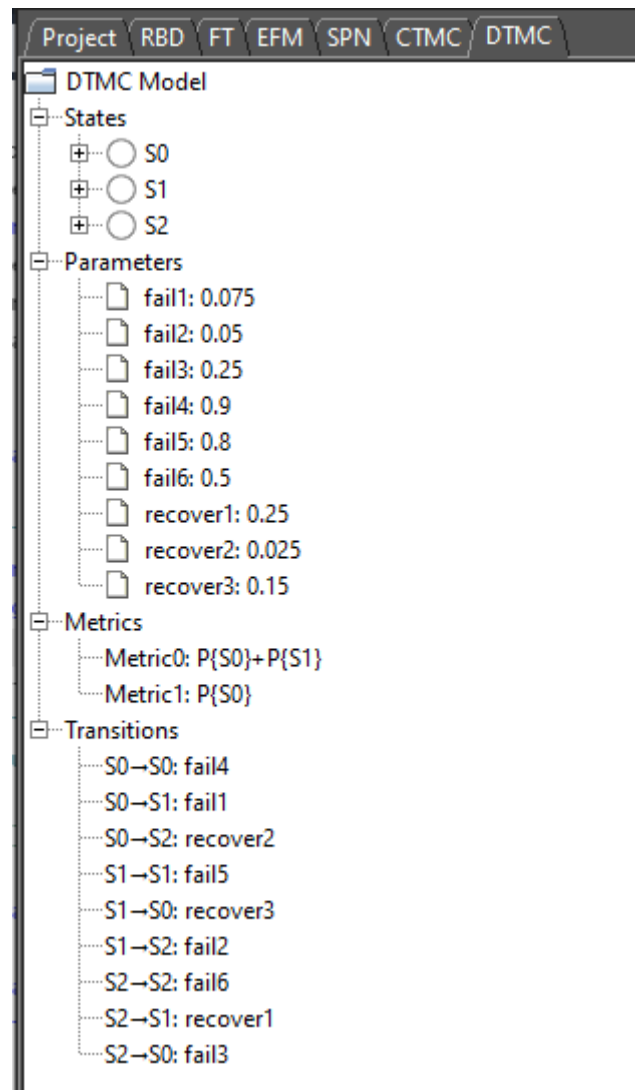


Figure 240: DTMC Panel on the Left Side of the Main Window

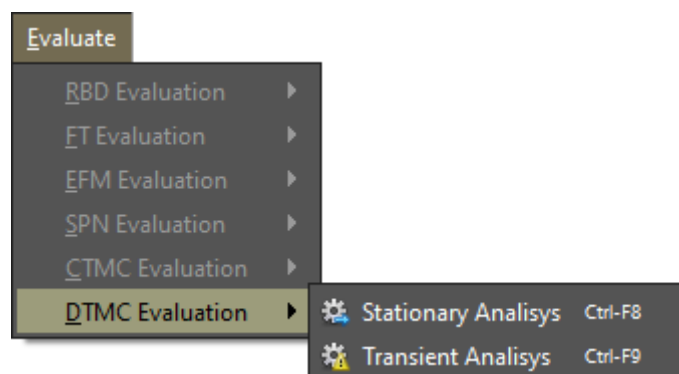


Figure 241: DTMC Evaluation Menu

Listing 8: Stationary Analysis of a DTMC

```
Tue Feb 05 07:01:25 BRT 2020
Performing stationary analysis...
Done! (elapsed time: 0)
```

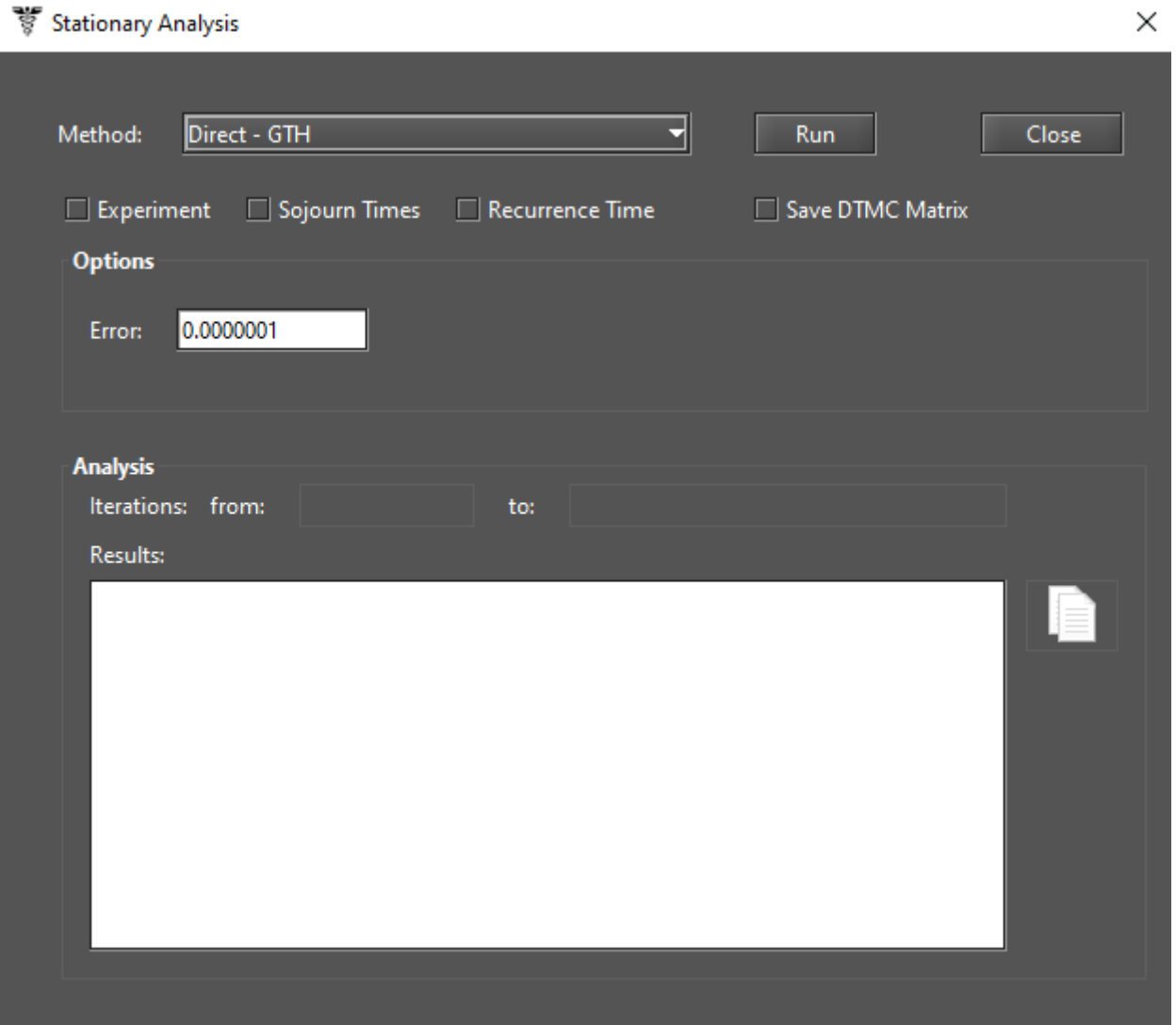


Figure 242: Stationary Analysis Window

```
#####
S0=0.625
S1=0.3125
S2=0.0625
----- Metrics -----
Metric0=1.0
-----

Results have been written into the file :
C:\Users\Thiago\Chapter_DTMC_Modelos-StationaryAnalysis.txt
```

When solving DTMCs through Gauss-Seidel, besides the maximum error, it is possible to change the maximum number of iterations. Default value for such a parameter is “-1”, indicating the algorithm only stops when the convergence of results is reached, considering the entered error (see Figure 243).

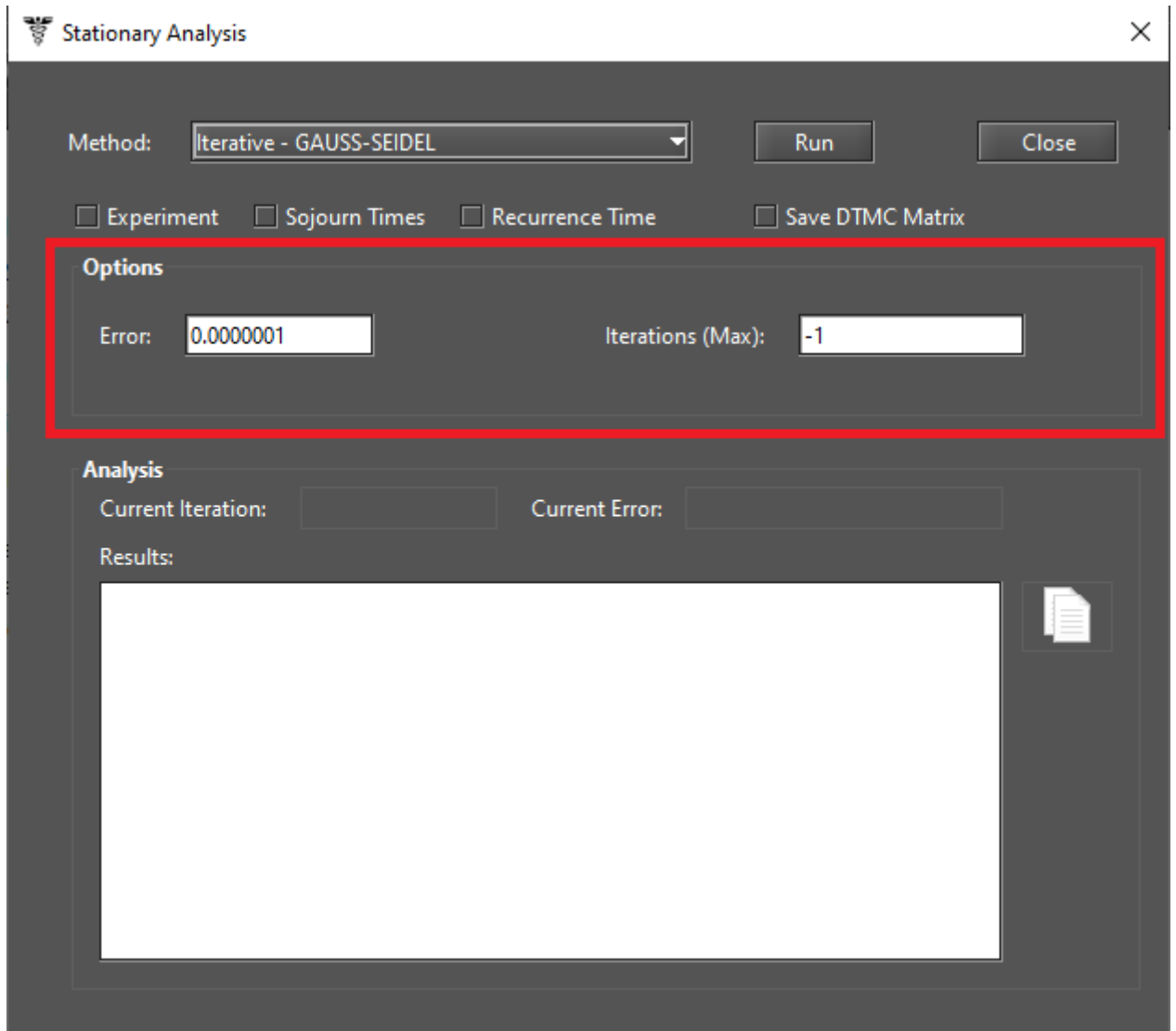


Figure 243: Stationary Analysis Window - Gauss-Seidel Method

Metrics are updated in the drawing area as soon as the analysis is completed.

DTMC models can also be solved for a range of values of an user-defined parameter. This is accomplished by checking “Experiment” and clicking on button “Run”. A new window is displayed in order for the user to define the input parameters for the experiment (see Figure 244).

Below, we describe each field on this window.

- **Varying Parameter.** Parameter to have its value changed.
- **Output Measure.** Metric to be evaluated.
- **Range Minimum Value.** Initial value to be assigned to the selected parameter.
- **Range Maximum Value.** Final value to be assigned to the selected parameter.
- **Interval.** It is the step-size for changing the value of the parameter. Parameter starts with the minimum

Options for Experiment

Options

Parameter:

Minimum Value: Maximum Value:

Interval (step size):

Metric:

OK Cancel

Figure 244: DTMC Experiment

value and its value is increased considering the entered interval. At each change, the selected metric is evaluated. Experiment finishes when the maximum value for the parameter has been reached.

At the end of an experiment, a graph is plotted and results are shown, as we can see by looking at Figures 245 and 246, respectively.

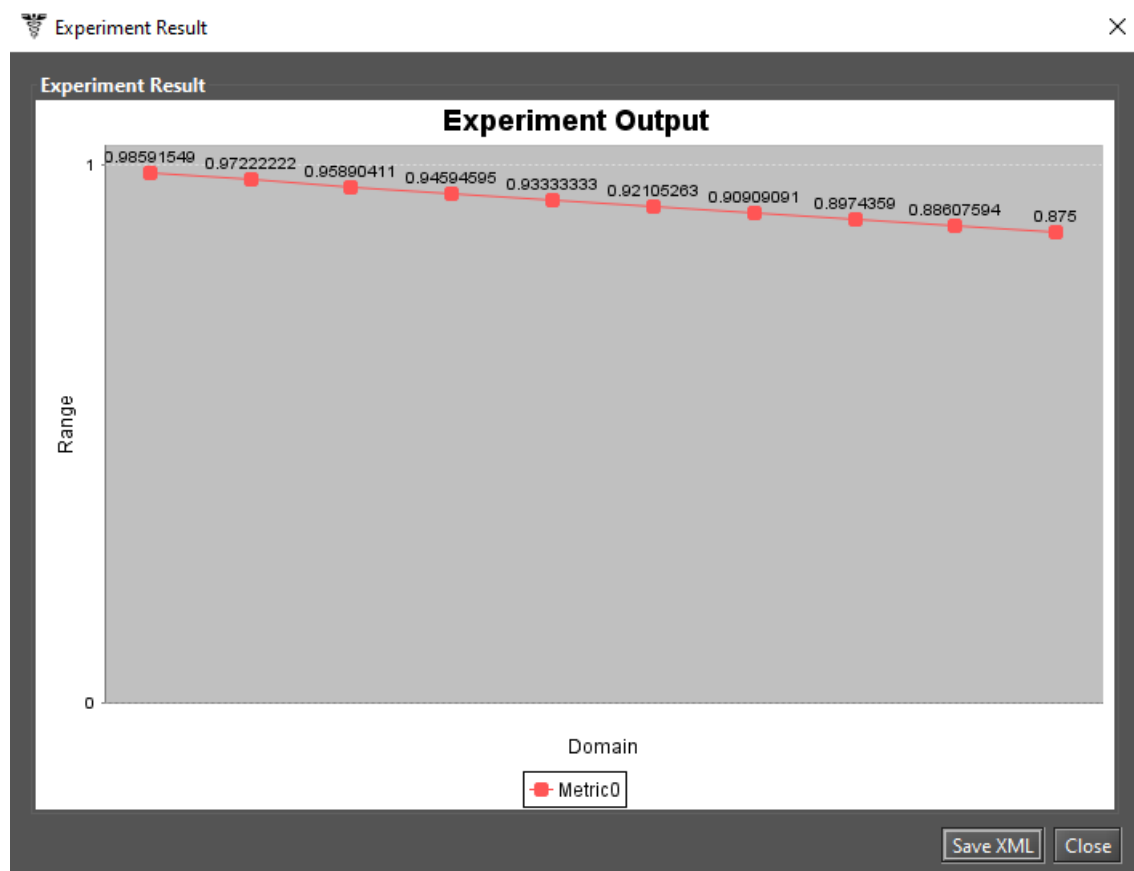


Figure 245: Graph of a DTMC Experiment

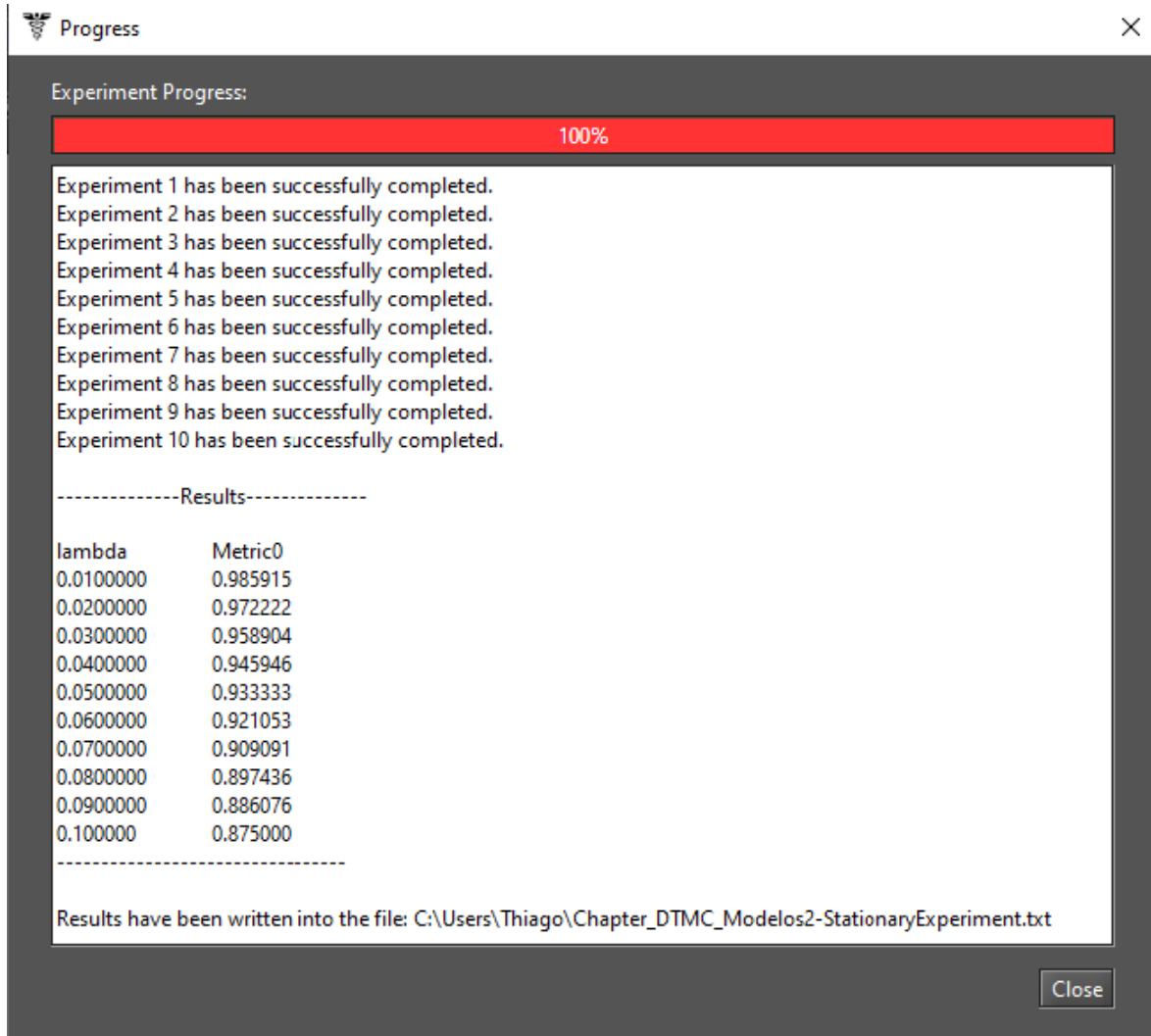


Figure 246: Results of a DTMC Experiment

Mercury can also compute the following metrics when they are selected.

- **Sojourn times.** Time spent in each state. Listing 9 shows sojourn times computed for each state of our DTMC (see Listing 9).
- **Recurrence time.** Time spent to return to each state. Once the state has been reached and a transition occurs to another state, what is the time to reach the former state again? It is obtained by this metric. Listing 10 shows recurrence times computed for each state of our DTMC (see Listing 10).

Listing 9: DTMC Stationary Result - Sojourn Times

```

Mon Aug 15 10:13:01 BRT 2020

Performing stationary analysis...

Done! (elapsed time: 0s)

#####

S0=0.6586375715496784

S1=0.280391409717778

S2=0.0609710187325436

Hold time: S0=11.494252873563209

Hold time: S1=5.000000000000001

Hold time: S2=2.0

----- Metrics -----

Metric0=1.0

-----

Results have been written into the file :

C:\Users\Thiago\Chapter_DTMC_Modelos-StationaryAnalysis.txt

```

Listing 10: DTMC Stationary Result - Recurrence Time

```

Mon Aug 15 10:14:50 BRT 2020

Performing stationary analysis...

Done! (elapsed time: 0)

#####

S0=0.625

S1=0.3125

S2=0.0625

Recurrence time: S0=1.6

Recurrence time: S1=3.199999999999993

Recurrence time: S2=15.999999999999996

----- Metrics -----

Metric0=1.0

-----

Results have been written into the file :

C:\Users\Thiago\Chapter_DTMC_Modelos-StationaryAnalysis.txt

```

6.3.2 DTMC Transient Analysis

Transient analysis computes time-dependent probabilities, useful for evaluating the behavior of modeled systems at a particular point in time. Figure 247 shows the window “Transient Analysis”.

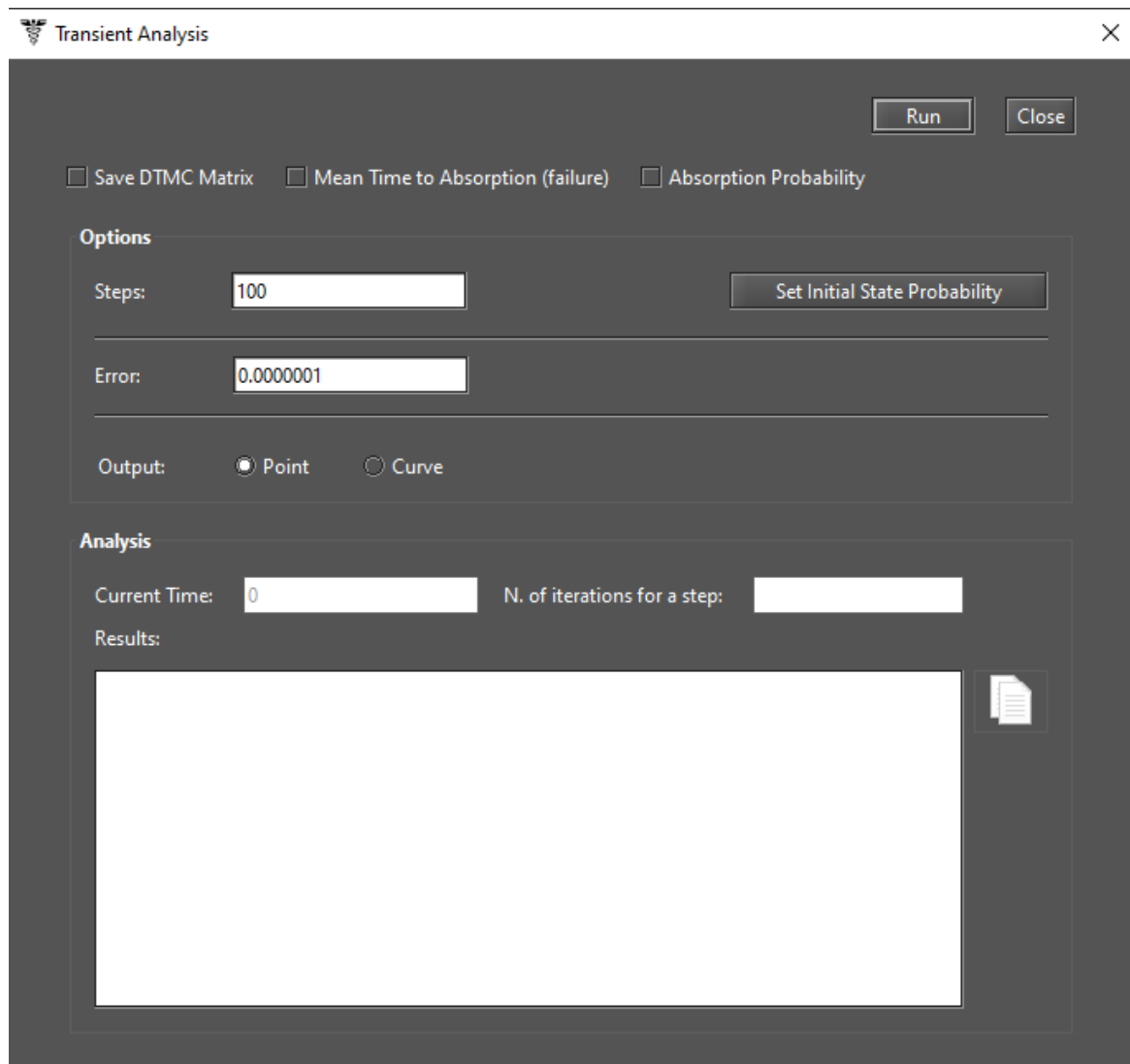


Figure 247: Transient Analysis Window

When solving transient metrics of DTMCs, the user can define:

- **Steps** for which the analysis will be carried out (default: 100).
- **Error** (default: 10^{-7}) to be considered when calculating results.
- **Initial state probabilities** (default: 1 for the first inserted state, 0 for the remaining states). These probabilities are defined by clicking on button “Set Initial State Probability” (see Figure 248).

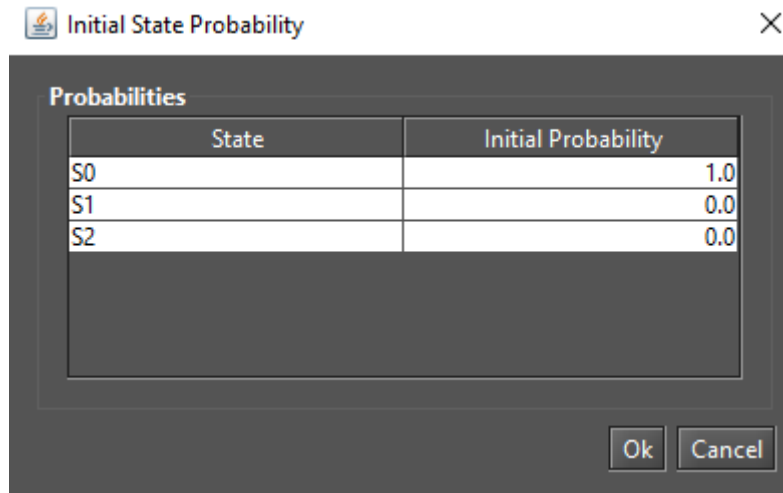


Figure 248: Initial State Probability Dialog

The internal step size will affect the accuracy of results and also the time required for computing the metrics. By clicking on button “Run”, the solution algorithm is triggered. As soon as it finishes, results are presented in the text area at the bottom of window “Transient Analysis”, also they are written in a plain text file having the filename of the project appended with the “-TransientAnalysis.txt” suffix.

“Transient Analysis” window also allows the user to choose between **Point** or **Curve** on the analysis. **Point** is the default option, and it shows probabilities for states only for the specific point in time. **Curve** writes in a plain text file all state probabilities computed from time equals zero until the specified value.

Mean time to absorption (**MTTA**) can be computed by checking “Mean Time to Absorption (failure)”. MTTA is presented after state probabilities in the “Results” area. For MTTA calculation, the user might also define a metric by using the keyword **MTTA** as expression. Absorption probability for each state is another metric that is available. Listing 11 shows MTTA and absorption probability by considering an absorbing DTMC as an example.

Listing 11: DTMC Transient Result - MTTA and Absorption Probability

```
Mon Aug 15 11:22:58 BRT 2020
Performing transient analysis...
Results have been written into the file :
C:\Users\Thiago\Chapter_DTMC_Modelos2-TransientAnalysis.txt
Matrix P has been written into the file :
C:\Users\Thiago\Chapter_DTMC_Modelos2-MatrixP.txt
Done! (elapsed time: 1s )
#####
S3=7.888609052210118E-31
S4=1.0
----- Metrics -----
Metric0=7.888609052210118E-31
```

Absorption probability to state S4: 1.0

Mean Time to Absorption (MTA): 2.0

7 EFM Modeling and Evaluation

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 the 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 budget 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.

Figure 249 depicts an EFM model representing a system composed of four components and it demonstrates how the energy flow occurs between them.

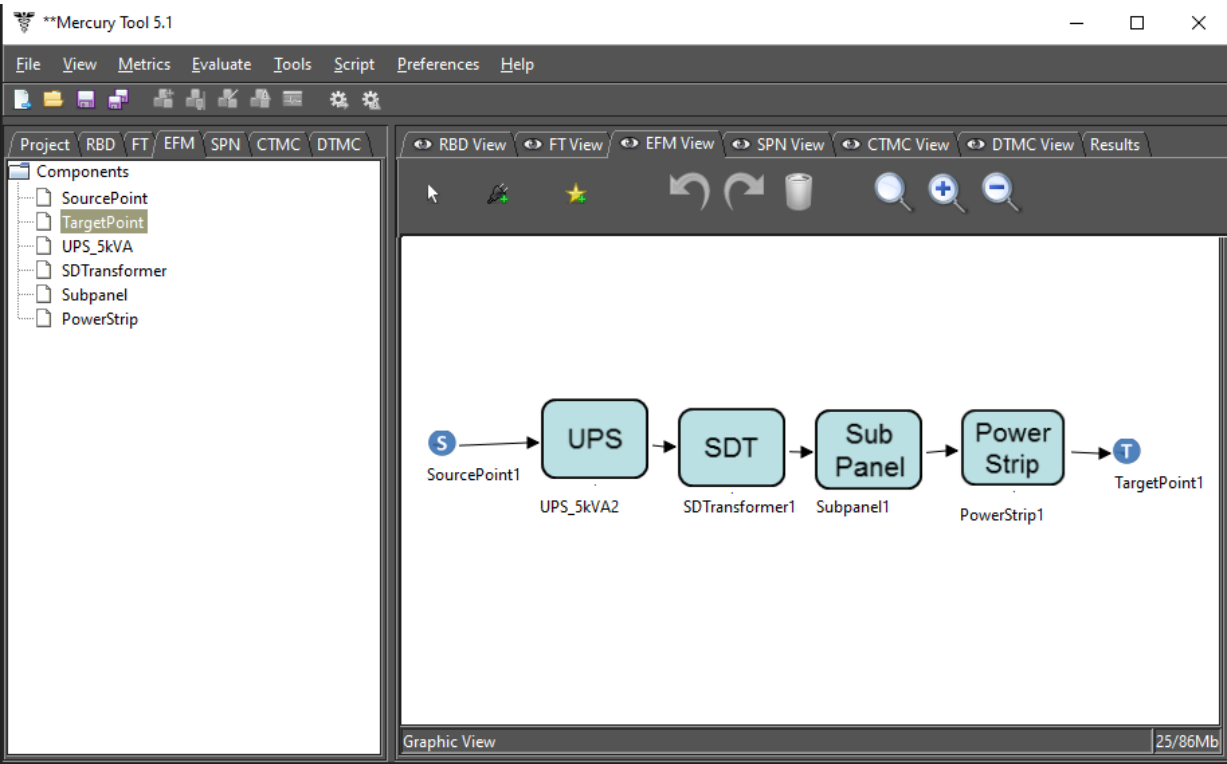


Figure 249: EFM Example

Now we will introduce the EFM toolbar and its available resources. This toolbar is visible when the EFM view is selected. Figure 250 shows the buttons present on it and their descriptions are detailed below.



Figure 250: EFM Toolbar

1. **Default Cursor.** Activates the selection mode. This mode makes it possible to select model components in the drawing area.
2. **Insert Component to Canvas.** Add datacenter components to the canvas. The first step to use this functionality is to select the EFM component on the left side panel (see Figure 251). After the selection, the user needs to click on this button and, after that, she needs to click in the drawing area on the location desired to put the new component.

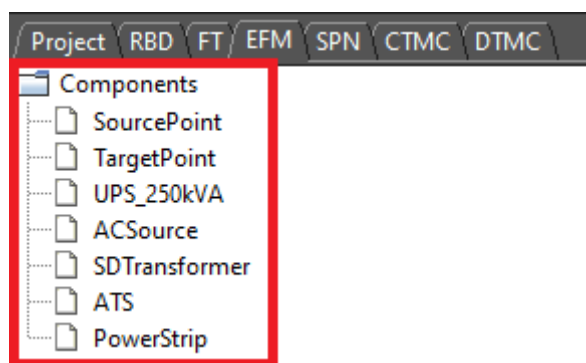


Figure 251: EFM Left-Side Panel

3. **Add a New Component to the EFM Project.** Add new data center components to the project. By clicking on it a dialog appears allowing the modeler to add a component to the project. In order to accomplish this, the user should select the component in this new dialog and confirm the selection by clicking on the “Add” button (see Figure 252). The new component will be available on the left side as shown by Figure 251. It is important to highlight that this function only adds components to the EFM project. The new component is not inserted into the drawing area at this moment. So, in order to do that, it is necessary to click on the “Insert Component to Canvas” button (2), after selecting the desired component on the left side panel.

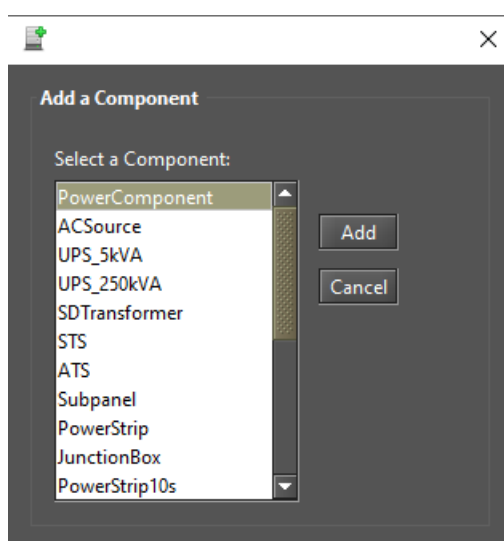


Figure 252: Inserting an EFM Component to the Project

4. **Undo.** Undo the last changes in the drawing area.
5. **Redo.** Redo the last changes in the drawing area.
6. **Delete.** Remove the selected components from the drawing area.
7. **Standard Scale.** Apply the standard scale to the drawing area.
8. **Scale Up Image.** Each click scales up the drawing image by 10% percent (zoom in).
9. **Scale Down Image.** Each click scales down the drawing image by 10% percent (zoom out).

Now, we will describe how to perform evaluations on an EFM project by using the Mercury tool. The first step in order to perform the sustainability evaluation is to create an EFM model. To add power or cooling components in the model, users should click on the start icon (“Add New Component to the Project” button). Once clicked on that button, a window appears in which it is possible to select the component to be added to the EFM project (see Figure 252). The list of components on the left side panel is updated after the addition of a component. The next step is to add the selected component on the left side panel to the drawing area as depicted in Figure 253. The user needs to click on the power plug icon on the toolbar (“Add Component to Canvas” button), after that she must click into the drawing area. These last steps need to be performed for each component that composes the evaluated system.

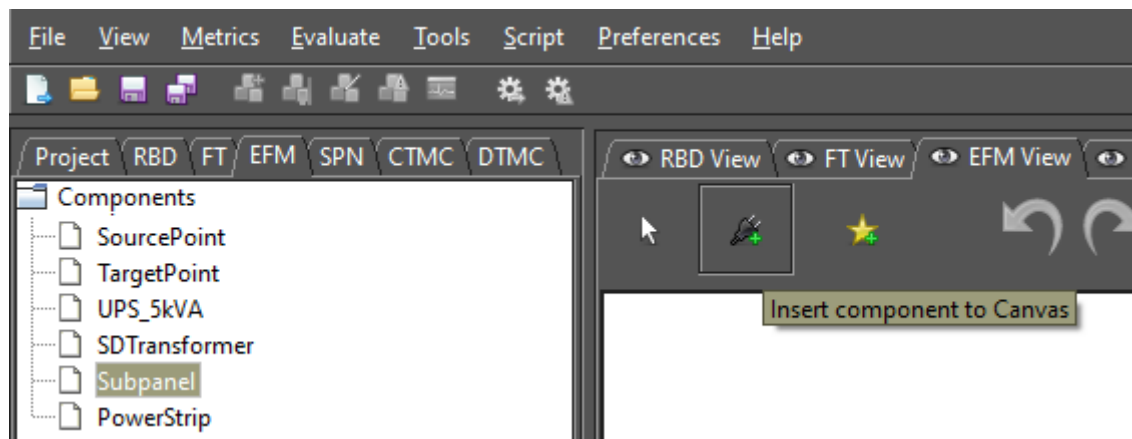


Figure 253: Inserting a Component into the Drawing Area

Once all components are inserted into the drawing area, the user can connect them including SourcePoints and TargetPoint as shown by Figure 254.

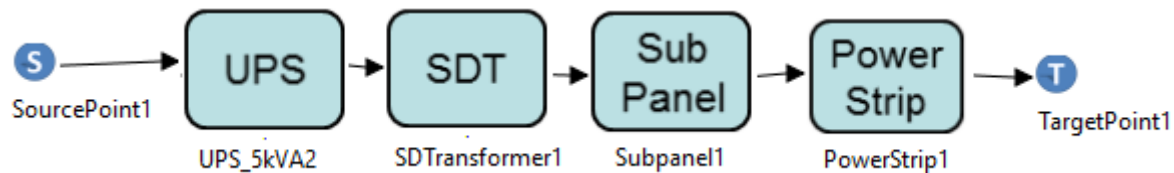


Figure 254: EFM Example

By right-clicking on a component and left-clicking on the “Properties” menu that appears provides a way to edit its properties as shown by Figure 255.

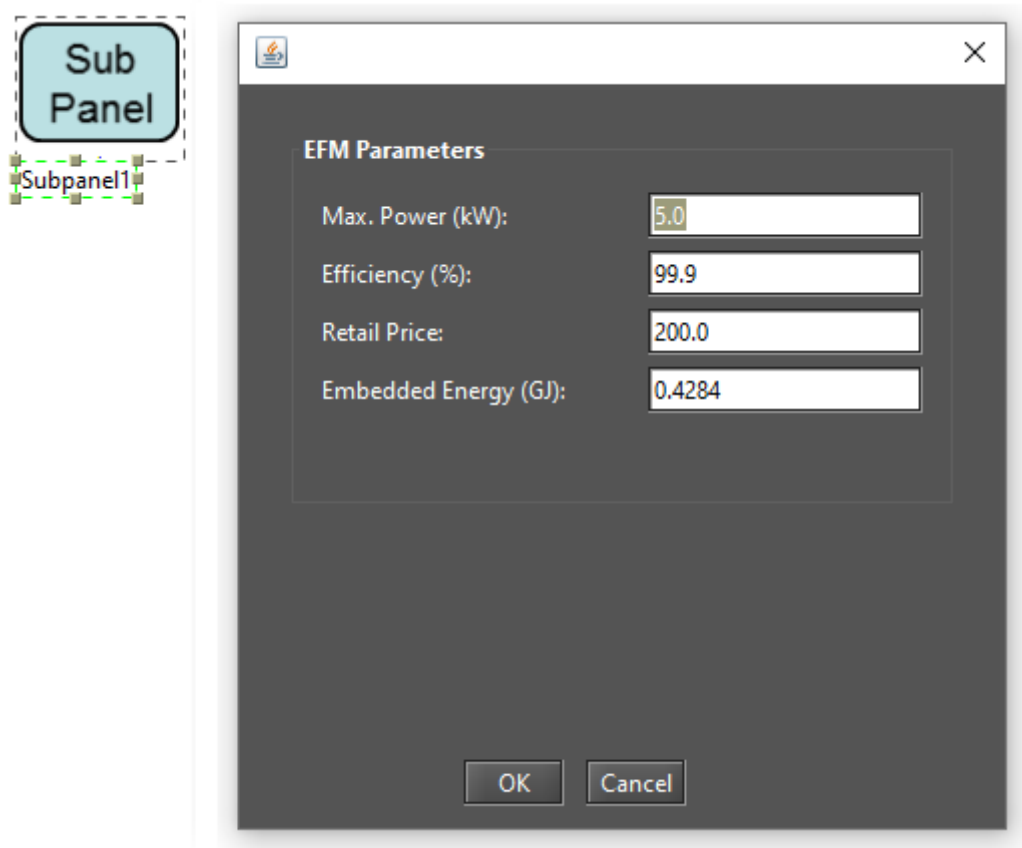


Figure 255: Component Properties

Additionally, it is important to stress that users have to set the demanded power on the TargetPoint (for power system) or on the SourcePoint (for cooling system). This is done by with a right-click on the Source or Target points and selecting the option “Properties”. After that, a dialog appears and the demanded power may be entered (see Figure 256).

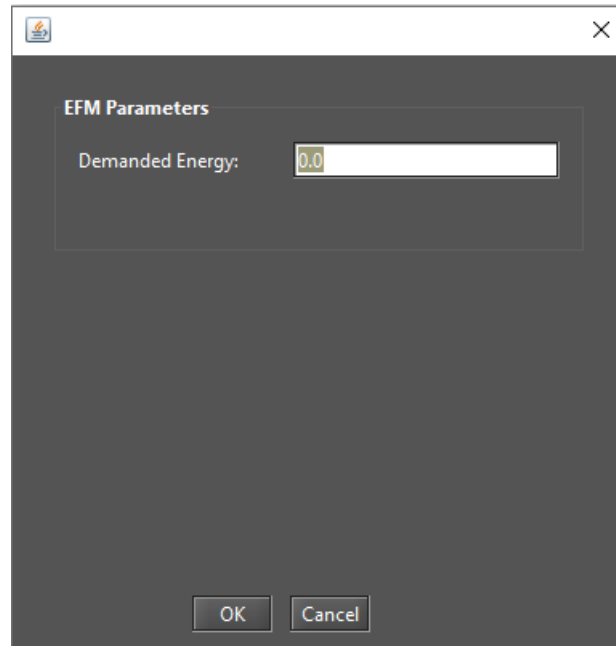


Figure 256: SourcePoint and TargetPoint Property

Once the EFM model is completed, evaluations can be conducted in order to extract some metrics. Evaluations are performed by selecting the desired option on the “EFM Evaluation” menu group available in the “Evaluate” menu on the main menu, as depicted by Figure 257. Five EFM evaluations are available. It is possible to evaluate cost, exergy, energy flow, the last ones combined, and flow optimization.

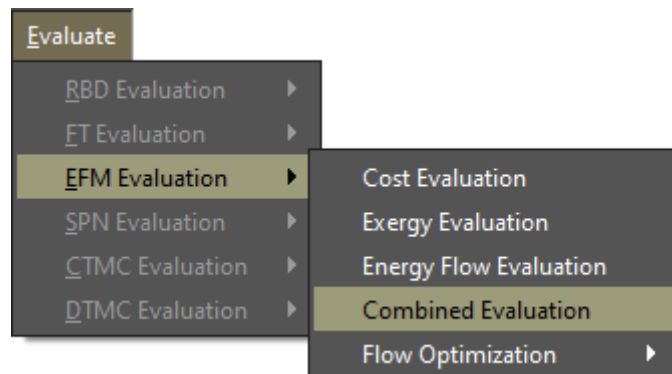
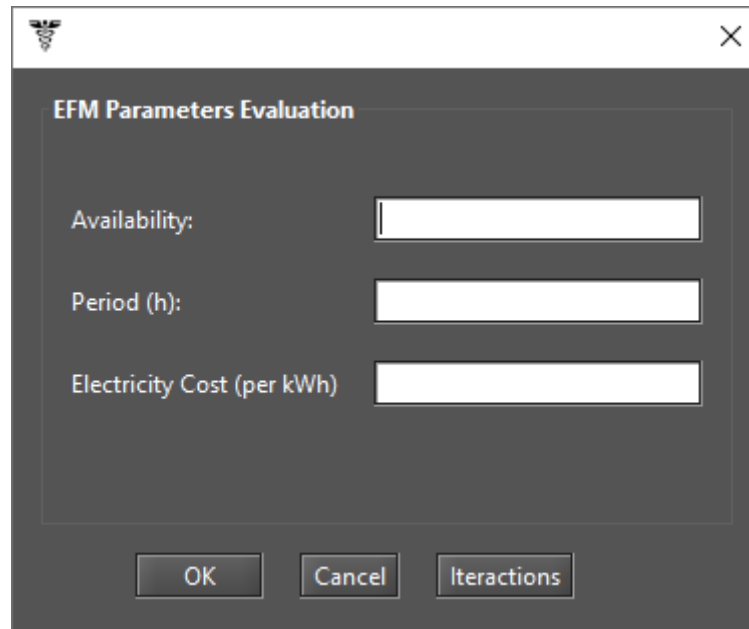


Figure 257: EFM Evaluation Menu

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 258. The parameters that the user may provide are availability, period to be considered (in hours), and electricity cost (per kWh).

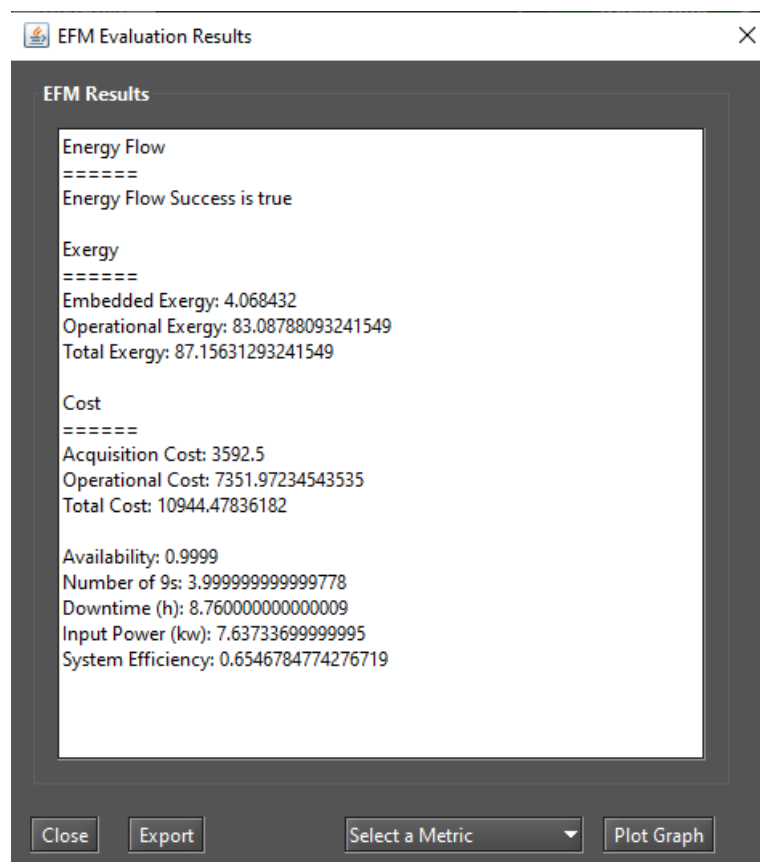


The dialog box is titled "EFM Parameters Evaluation" and contains three input fields: "Availability:", "Period (h):", and "Electricity Cost (per kWh)". At the bottom, there are three buttons: "OK", "Cancel", and "Iterations".

| Parameter | Value |
|----------------------------|-------|
| Availability: | |
| Period (h): | |
| Electricity Cost (per kWh) | |

Figure 258: EFM Parameters Evaluation

Finally, the result is presented as demonstrated by Figure 259.



The dialog box is titled "EFM Evaluation Results" and displays the following results:

```
EFM Results

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: 7351.97234543535
Total Cost: 10944.47836182

Availability: 0.9999
Number of 9s: 3.999999999999778
Downtime (h): 8.760000000000009
Input Power (kw): 7.63733699999995
System Efficiency: 0.6546784774276719
```

At the bottom, there are four buttons: "Close", "Export", "Select a Metric" (with a dropdown arrow), and "Plot Graph".

Figure 259: EFM Results

In this example, the result indicates that the energy flow evaluation returns true meaning that the power constraints present on each device were respected. However, in case this result is false, the Mercury tool shows the component in which the constraint was crossed (see Figure 260). In this window of results, the user has also an option to export the results to a spreadsheet (e.g., a file .xls), or plot a selected metric.

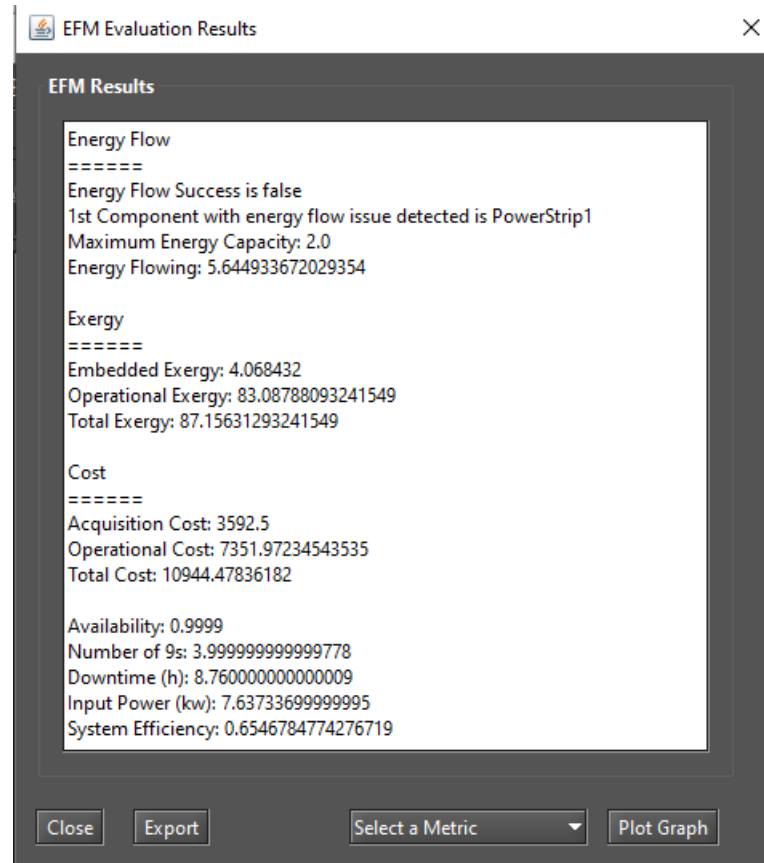


Figure 260: EFM Results with the Energy Flow Evaluation False

7.1 Power Load Distribution Algorithm - PLDA

A Power Load Distribution Algorithm (PLDA) is proposed to minimize the electrical energy consumption of the EFM models [6]. The PLDA is based on the Ford-Fulkerson algorithm, which computes the maximum possible flow in a flow network [7]. 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 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 [8, 9]. The PFS chooses the path according to the highest electrical capacities of nodes in the graph [10]. Figure 261 shows how to call the PLDA function.

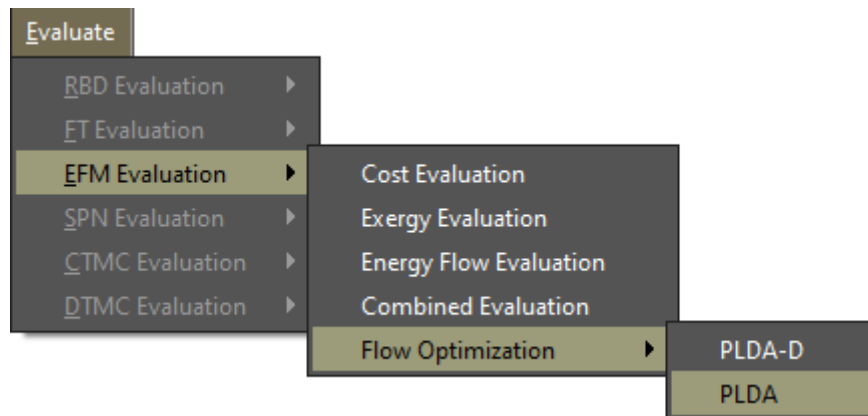


Figure 261: PLDA Optimization Function

Figure 262 depicts the results of the PLDA algorithm, with the minimum energy consumed, PUE, and DCiE highlighted.

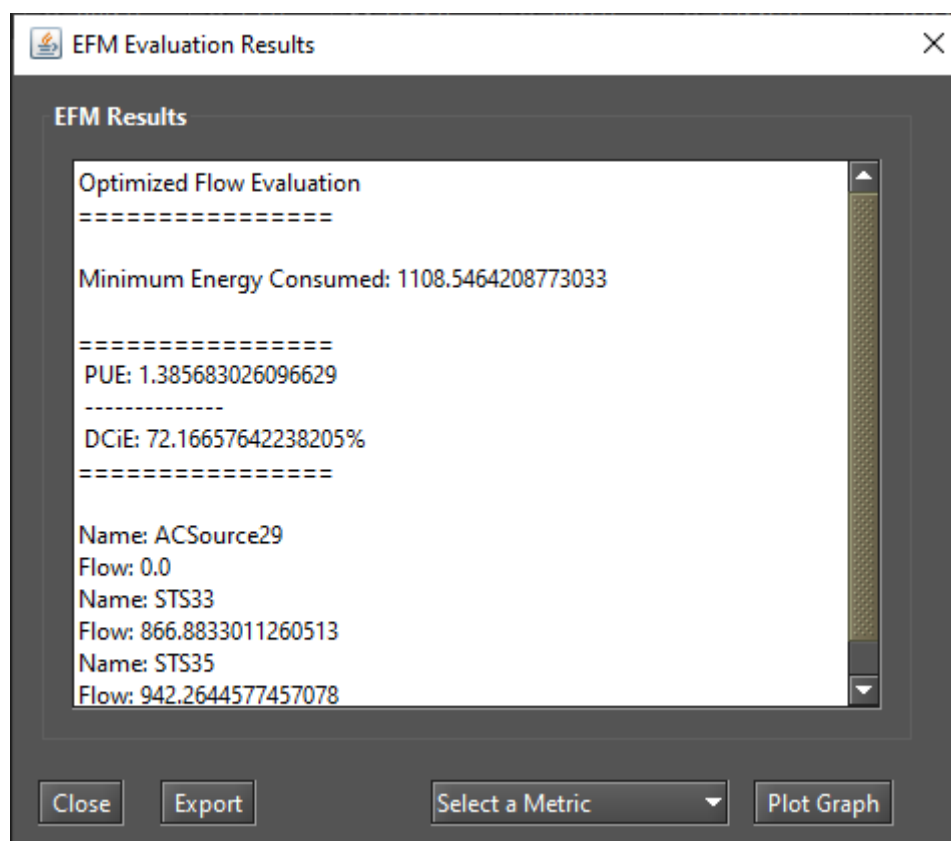


Figure 262: PLDA Optimization Results

7.1.1 Example of PLDA execution

Figure 263 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.

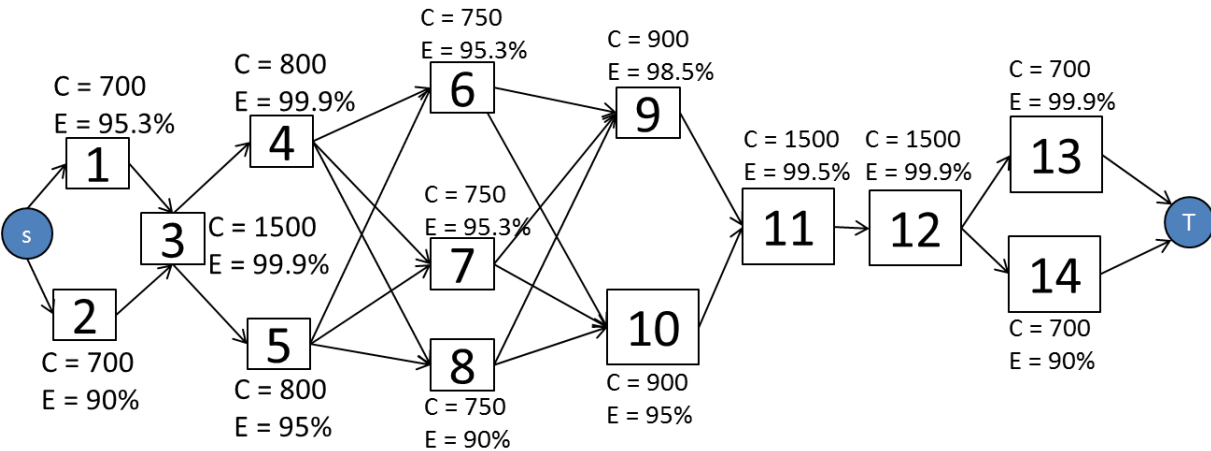


Figure 263: EFM Model

Figure 264 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.

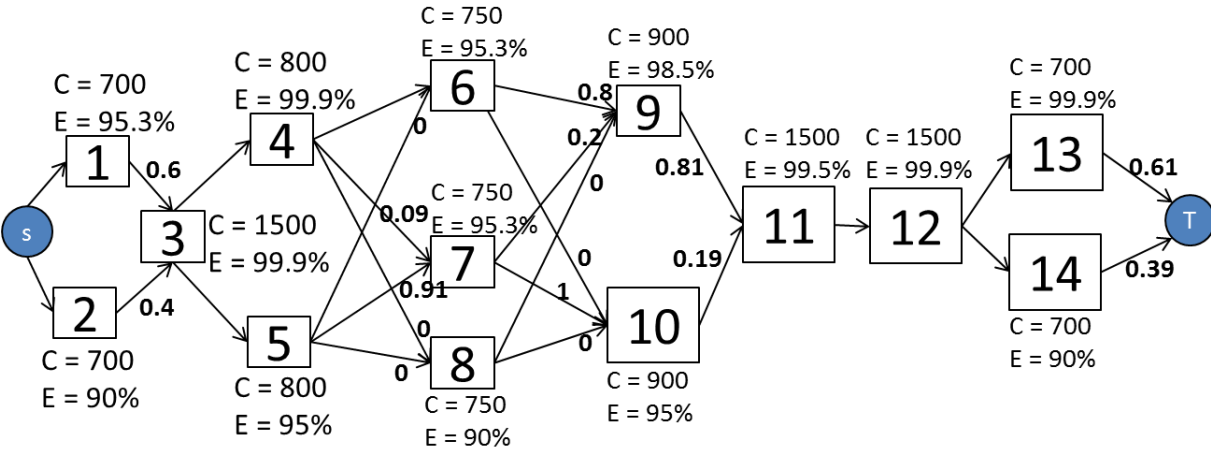


Figure 264: EFM Model After PLDA Execution

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 |

7.2 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 [6]. It is an evolution of the PLDA algorithm (see Section 7.1), applies for the same problem but with a big difference in the 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 [11] and Ford and Fulkerson [7] flow algorithm, but with many adaptations. The PLDA-D is divided into three phases: initialize, kernel, and the search for the best path. Figure 265 demonstrates how to call the PLDA-D function. Figure 266 depicts the results generated by applying the PLDA-D algorithm, with the minimum energy consumed, PUE, and DCiE highlighted.

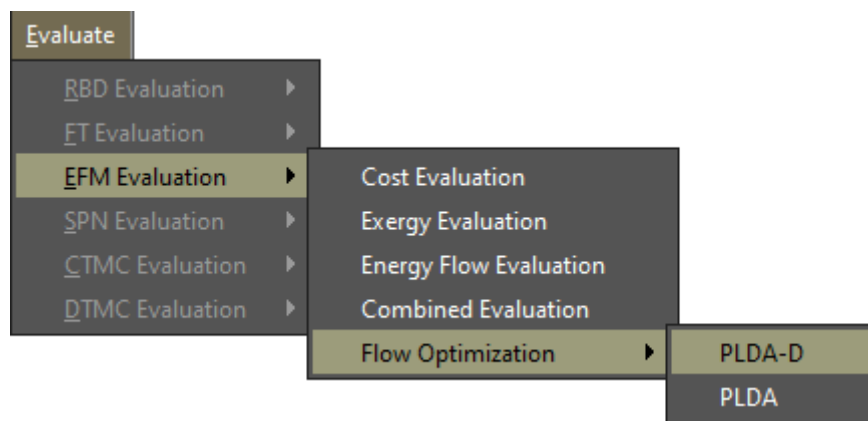


Figure 265: PLDA-D Optimization Function

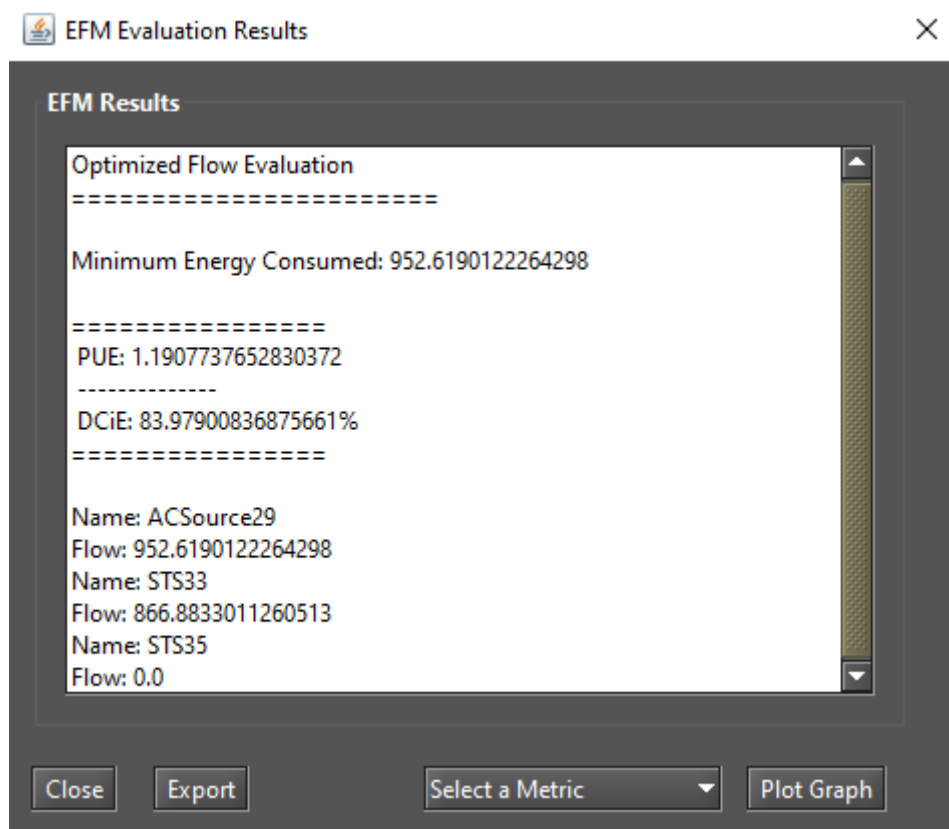


Figure 266: Results for Applying the PLDA-D Optimization

7.2.1 Example of PLDA-D Execution

Figure 267 illustrates the step-by-step of the PLDAD execution, highlighting some variables, edges, and weights. Lets consider the model represent by Figure 267.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 other 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 efficiency (*Ef*) values are known. The value of the *Target* node *Acc* is set to one. The others accumulated costs (*Acc*) are set to zero and the edge weights are set to the default value one, respectively, as depicted in Figure 267.a, a perfect representation of an EFM model. The phase one of the PLDAD algorithm is represented by the Figure 267.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 267.c following to the Figure 267.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 267.c the values of the *ActualCost* and *AccumulatedCost* are computed and the best child is chosen, according to 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 267.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

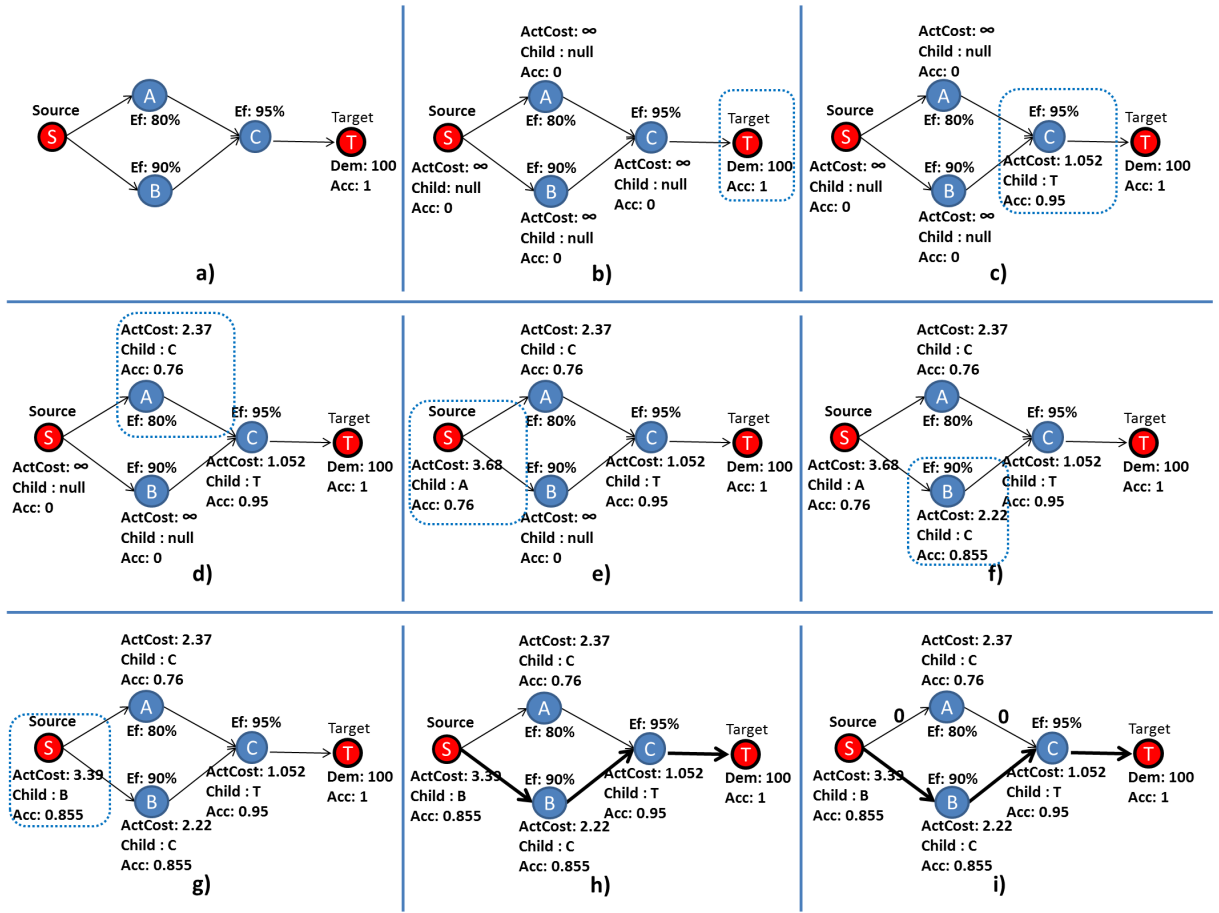


Figure 267: Example PLDAD Execution

previous path (3.68) to get to the *Source* node. Thus, the *Source* node has a 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 267.h represents the end of phase three. For this example, the best path from note Target to Source is: *Target, C, B, Source*. In Figure 267.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.

8 Mercury Scripting Language

8.1 Introduction

Mercury scripting language was designed to allow greater flexibility in model evaluations. To run Mercury scripts, we can use a command-line interface (CLI) tool or by accessing the “Script Editor” available inside the Mercury GUI. The advantage of using this language in conjunction with the CLI tool is the possibility to automate the project workflow, evaluate models, extract metrics and generate reports and graphs automatically.

In addition, there are other advantages offered by the language, which are not supported by modelling through the graphical interface.

- Increased support to hierarchical modeling: any model can call another model and use its results as an internal parameter;
- Increased support for symbolic evaluation and experiments. Parameters of a model can be defined as variables left open. We can change these variables and reevaluate the model to measure the impact of these parameters on certain metrics;
- Support for Petri net transitions with a phase-type delay. This family of distributions can be used to approximate any distribution that does not fit an exponential distribution;
- Support for hierarchical transitions in SPN models. This type of transition 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] [12] support hierarchical SPN models only for coloured Petri nets.

8.2 Script Structure

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

Listing 12: Grammar for a Mercury Script

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

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

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

A script consists of a section for declaring models, which contains one or more models of the type: CTMC (continuous-time Markov chain), RBD (reliability block diagram), or SPN (stochastic Petri net). Support for FT and EFM formalisms will be included in the next version. At the end of the model section, there is the main block, which has the following syntax:

Listing 13: 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 results obtained. We modify variables to define parameters for a model and collect metric results by using the **solve** function. The “for” command has been included to allow the execution of experiments. With this command, we can modify a variable according to a list of values.

In Figure 268, we show a CTMC model, and in Listing 14, we present the corresponding Mercury script. First, we define a CTMC model named **CTMCModel**, declare its states, transitions, and metrics. Transition rates are defined as function of the parameters **lambda** and **mu**. In the main block, we define values for these parameters and evaluate the metric “m1” of the CTMC model. The result is stored in the variable named **aval**. Finally, we print the content of this variable by using the command **println**.

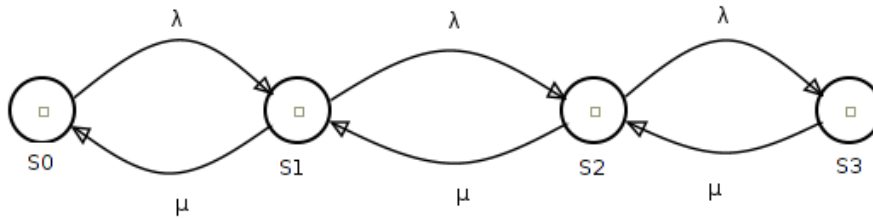


Figure 268: CTMC Model Example

Listing 14: CTMC Model

```

CTMC 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 ml = stationaryProbability( st = S0 );
}

main{
    lambda = 0.00001;
    mu = 0.01;

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

```

8.2.1 Reserved Words

In 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

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

In the following sections, we describe the syntax for each supported formalism: continuous-time Markov chains, reliability block diagrams, and stochastic Petri nets.

8.3 Continuous Time Markov Chain

In Listing 15, we describe the syntax for declaring CTMC models. A CTMC model contains definitions of states — with the reserved word *state*—, transitions —with the reserved word *transition*—, and metrics. A state can also receive an annotation *up* after the identifier, for availability models. This annotation defines states where the system is considered operational.

Listing 15: 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; and 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.

Listing 16: 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=lambda_s2);
    transition uf -> ff(rate=lambda_s1);

    transition fw -> ff(rate=lambda_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: $\langle state_name \rangle = \langle value \rangle$. The metric computes the sum of products of each rate and the stationary probability associated with the state. The states that do not receive any rate, are associated with a rate zero, implicitly.

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

Listing 17: 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 states. The stationary probability of a state S corresponds to the proportion of time that the model remains in this state. The transient probability of a state S within a time T , corresponds to the probability to be in this state S , after T units of time from initial time ($t = 0$).

In 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 probability for each state. That corresponds to the probability that at time $T = 0$, the model will be in that state. In the scripting syntax, the states that are not declared in the list of initial probabilities, receive an initial probability equal to 0. It is important to highlight that the sum of all initial probabilities must be equal to 1, otherwise, an exception will be raised.

The following we illustrate how to get the metrics for stationary and transient probabilities taking into account a CTMC model as an example.

Listing 18: 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 of:

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

Listing 19 shows the grammar for RBD models.

Listing 19: RBD 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> ";"

```

We have four metrics available to RBD models:

- Availability;
- Mean time to failure (MTTF);
- Mean time to repair (MTTR);

- Reliability.

The three first metrics do not take any parameters: steady-state availability, MTTF and MTTR. On the other hand, reliability and instantaneous availability metrics require a *time* parameter.

By considering the model depicted in Figure 269, we generated its script definition as shown in Listing 20.



Figure 269: RBD Representing a Cloud Node [13]

Listing 20: RBD Script

```

t = 100;

RBD Model{
    block HW( MTTF = 4000.0, MTTR = 72.0);
    block SO( MTTF = 2500.0, MTTR = 12.0);
    block KVM( MTTF = 4000.0, MTTR = 24.0);
    block NC( MTTF = 4000.0, MTTR = 24.0);
    series s0(HW, SO, KVM, NC );

    top s0;

    metric av = availability;
    metric rel = reliability( time = t );
    metric mttf = mttf;
    metric mttr = mttr;
}

main{
    av = solve(Model, av);
    rel = solve(Model, rel);
    mttf = solve(Model, mttf);
    mttr = solve(Model, mttr);

    println(" Availability: " .. av );
    println(" Reliability: " .. rel );
}

```

```

println("Mean time to failure: " .. mttf );
println("Mean time to repair: " .. mttr );
}

```

8.5 Stochastic Petri Nets

In the Mercury scripting language, a Petri net is described in terms of places and transitions. Places can be defined with an (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 available only in the scripting language. Another exclusive feature of the scripting language is the support for *phase-type* distributions. In this section, we will show a simple SPN as an example containing only exponential and immediate transitions.

Listing 21 shows the grammar in BNF notation for describing SPNs models in the Mercury 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 have as parameters the delay and the server type. If omitted, the “server type” parameter will be “SingleServer” by default. Immediate transitions have as parameters (optional): weight, priority, and an enabling function. The metrics are defined in terms of a string representing a reward metric, the same used in the graphical interface.

Listing 21: 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 modeling SPNs using the Mercury scripting language, we have proposed a model for an M/M/1/K queue, based on [14]. That model is depicted by Figure 270.

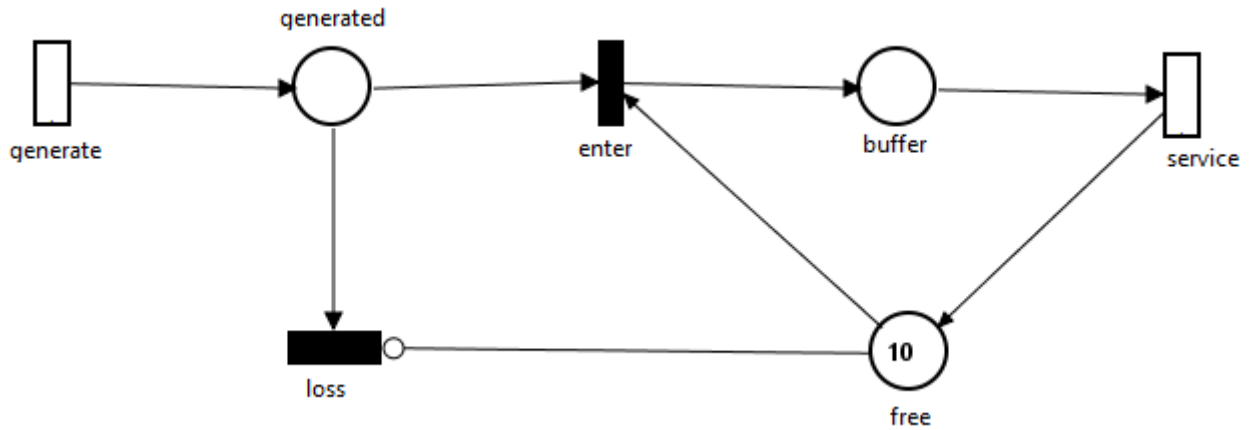


Figure 270: SPN Model Representing an M/M/1/k Queue[14]

Transition *generate* creates tokens that correspond to service requests. Each generated token is placed in the place *generated* and, thereafter, a choice is made. The token can be queued for processing by the server, if there is a free slot in the queue (tokens in the “free” place). Otherwise, the token is discarded, being represented by the firing of the immediate transition “loss”. The place “free” controls the firing of this transition through an inhibitor arc. The tokens waiting in the queue are placed in the “buffer”. The transition “service” represents

the processing of the requests. Considering that it is an M/M/1/K queue, we only have one server to handle all requests. Therefore, the server semantic assigned to the “service” transition is SINGLE SERVER.

Listing 22 shows the script for running the stationary analysis on the SPN model previously described. The parameter **method** of the **stationaryAnalysis** function can have only the values “direct” or “iterative”. “Direct” corresponds to the **Direct - GTH (Grassmann-Taksar-Heyman)** method. “Iterative” corresponds to the **Gauss-Seidel** method.

Listing 22: Script for Stationary Analysis

```
k = 10;
mu = 2;
lambda = 1;

SPN Model{
    place buffer;
    place free( tokens= 10 );
    place generated;

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

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

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

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

```

        metric m1 = stationaryAnalysis( method = "direct",
            expression = "P{#buffer>0}" );
    }

    main {

        setIntegerParameters("k", "mu", "lambda");

        m1 = solve( Model,m1 );

        println (m1);
    }

```

Listing 23 shows the script for running the stationary simulation on that model. Below, we describe each parameter of the **stationarySimulation** function.

- **confidenceLevel.** The confidence interval for obtaining the metrics.
- **maxRelativeError.** Defines the maximum relative error that is one of the stop conditions of the simulation.
- **minFiringTransitions.** Sets the minimum number of firings for each transition in the model. This number of firings is another condition for stopping the simulation. When set to 0, the simulator does not consider the number of firings in order to stop the simulation. Entering a value greater than 0, when the number of firings for each transition is equal to the defined value the simulation stops.
- **warmup.** Sets the minimum warm-up period. The warm-up phase is the period when the model is not considered to be in a steady-state, and the metrics are not collected in that period. There are some methods for estimating whether the model has entered a stationary phase, but Mercury requires the user to define a value for the warm-up time. As we are evaluating stochastic models, it is expected that the warm-up time is not an exact value for each simulation performed. Therefore, the user defines a minimum warm-up time and, once the global simulation time is equal to or greater than the warm-up time defined by the user, the simulation begins to collect the metrics, generate the batches, and calculate statistics.
- **batchsize.** Defines the number of samples that will constitute each batch in the simulation.
- **maxTimeMilliseconds.** It is used to define the maximum simulation time. This time corresponds to the physical time and must be defined in seconds. When set to 0, the simulator does not consider this parameter. If one of the stop conditions is not met before this maximum time is reached (maximum relative error or number of firings for each transition), then the simulation stops when this time is reached.

```

k = 10;
mu = 2;
lambda = 1;

SPN Model{

    place buffer;
    place free( tokens= 10 );
    place generated;

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

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

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

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

    metric m1 = stationarySimulation( parameters = ( warmup=0,
        confidenceLevel=90,
        maxRelativeError=0.05,
        minFiringTransitions = 0,
        maxTimeMilliseconds=0,

```

```

        batchSize=30
    ), expression = "P{#buffer>0}" );
}

main {
    setIntegerParameters("k", "mu", "lambda");

    ml = solve( Model,ml );
    println (ml);
}

```

Listing 24 shows how to run a transient simulation by using the scripting language. Below we describe each parameter of the **transientSimulation** function.

- **time.** The evaluation time.
- **expression.** The expression to be evaluated.

Listing 24: transientSimulation function

```
metric [name] = transientSimulation( time=[time], expression = "[exp]" );
```

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] 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
- [5] 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.
- [6] 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.

- [7] L. Ford and D. R. Fulkerson, *Flows in networks*. Princeton University Press, 1962, vol. 1962.
- [8] 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.
- [9] 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.
- [10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *et al.*, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 2.
- [11] R. Bellman, "On a routing problem," DTIC Document, Tech. Rep., 1956.
- [12] 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.
- [13] J. Dantas, R. Matos, J. Araujo, and P. Maciel, "An availability model for eucalyptus platform: An analysis of warm-standby replication mechanism," in *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1664–1669.
- [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 named <state_name>

“R{”<state_name>“}” = Reward rate of the state named <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 25: 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 occurrences of symbol

symbol1–symbol2 : range of values between symbol1 and symbol2

B Syntax of SPN Metrics, Guard Expressions, and Arc Multiplicity Dependent on Marking.

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

Three different expressions can be used in the Mercury tool (see Listing 26). SPN expressions are represented as “Metrics”, “GuardExpressions”, and “MarkingDependentMultiplicities”. “Metrics” can be a probability or an expectation and are used to represent the evaluated metrics. “GuardExpressions” are adopted to represent logic expressions in order to enable/disable the firing of transitions. “MarkingDependentMultiplicities” are numeric expressions that are evaluated depending on the current marking to a specific arc multiplicity.

Listing 26: 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 ABOUT SPN 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 > 2 AND #P2 = 0} // WRONG SYNTAX

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

(#P1 = 1) AND (#P2 = 2) // CORRECT SYNTAX

This expression can be adopted as an enabling 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 = 1 AND #P2 = 2 // WRONG SYNTAX

Regarding “if-else” expressions. The language supports if-else expressions to represent MarkingDependent-Multiplicity. This component is used to represent the 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 model 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 model 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, elseif, ..., else expressions in standard programming languages like C or Java.