Mercury Tool Manual
v6.0

MoDCS Research Group

[http://www.modcs.org]

CIn - Centro de Informatica UFPE

Cidade Universitaria - 50740-540 - Recife - Brazil

- Tel +55 81 2126.8430 -

December 30, 2024

Contents

1 Overview

This manual describes the **Mercury** tool: a software for supporting performance, dependability, and energy flow modeling in an easy and powerful way. The tool provides graphical 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), energy flow models (EFMs), and event trees (ETs).

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. Here we describe a comprehensive overview of the features as well as the steps to create, edit and evaluate the models supported by the tool. The following is an overview of Mercury's features:

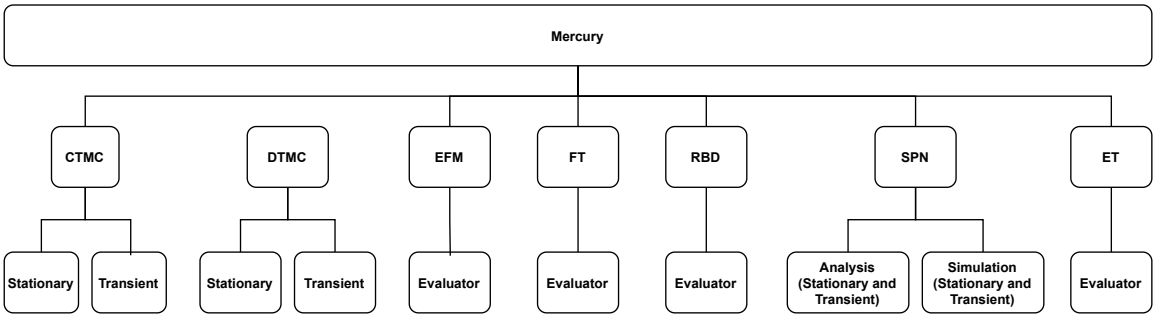


Figure 1: Mercury Tool - Features

Mercury has been developed in the Java language, which offers platform independence. The graphical interface allows modeling of systems using one or more views — RBD, FT, EFM, SPN, CTMC, or DTMC, and ET — while auxiliary modules (e.g., random variate generator and moment matching) are also available. In this way, users can choose the view best suited to their needs.

In addition, Mercury provides a feature that allows you to import models created in other programs that use the ". TN" standard format (the one used in tools like TimeNet [1]). In addition, there is also an option to export models created with Mercury to a ". TN" file that conforms to this standard. All projects developed with Mercury are saved in a ".xml" file, which contains all information about the created models.

1.1 How to Install the Tool

The first step to installing the latest version of Mercury is to access the URL `https://www.modcs.org/?page_id=2392`. There is a license agreement that must be signed and sent to the Mercury developers before the user is granted access to the download page.

Mercury is available on the MoDCS site in many flavors. There is the Mercury version with the Java runtime environment (JRE) already configured and there is the version without the JRE. In the first case, the user simply extracts the files into a folder and runs Mercury. In the version without the JRE, the user has to install and configure the JRE on their machine. It is important to note that Mercury is not compatible with newer versions of Java. From version 9 onwards, Java began to adopt a modular architecture called Java Platform Modular System (JPMS) [1]. JPMS radically changed the way systems are developed in Java, and many classes that were available in earlier versions no longer exist. Because of this, applications that have not yet been ported to this new architecture will not work properly on Java 9+ versions, which is the case with Mercury. The recommended version for running Mercury is JRE 1.8.

The Mercury installer contains executable files, a folder with third-party libraries, and a folder with example models. If the user selects the version with the JRE, there is also a folder with the JRE. Mercury's memory footprint is approximately 60 MB, but may increase depending on the size of the models and the type of analysis performed by the tool. When you start Mercury, the initialization screen shown in Figure 2 is displayed.

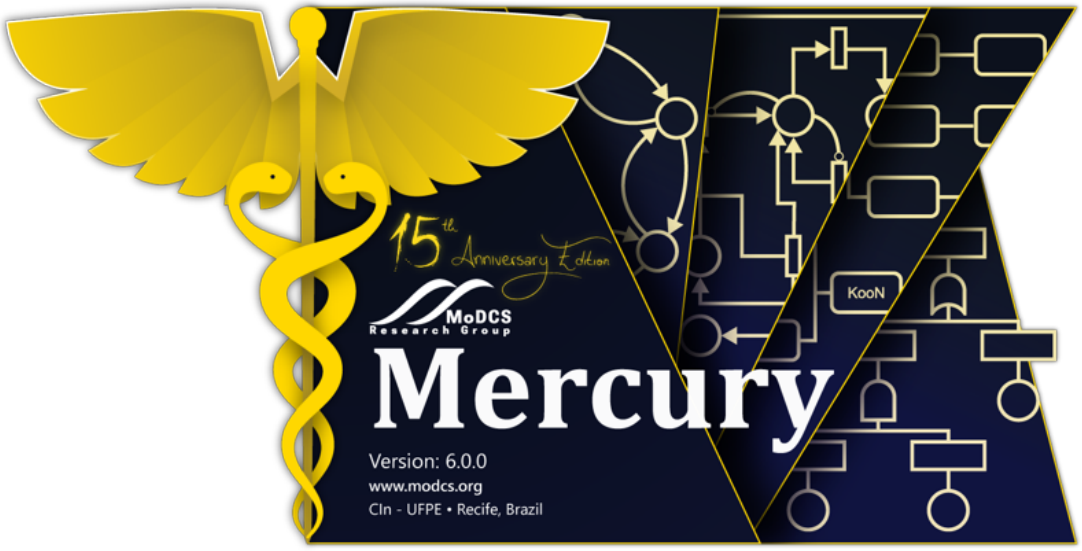


Figure 2: Initialization Screen

1.1.1 Linux System Requirements

This subsection describes the minimum Linux system configuration required to run Mercury. Make sure that the system meets these minimum requirements: 1) Java Runtime Environment (JRE) or Java Development Kit (JDK) version 1.8; and 2) OpenJFX package. The OpenJFX package is required to run the Fault Tree module. On

[1] https://www.oracle.com/br/corporate/features/understanding-java-9-modules.html

Ubuntu, the administrator can use the following command to install OpenJFX: *sudo apt-get install openjfx*. The Fault Tree module is not available if the last condition is not met.

1.1.2 Increasing JVM Memory Allocation

To increase the maximum memory allocated by the Java Virtual Machine (JVM) when running the tool, the parameter `-Xmx` must be added to the command used to start Java. This parameter defines the upper limit for the heap memory of the JVM so that Mercury can process larger models. For example, the command in the file `.bat` or `.sh` should be changed as follows:

```
java -Xmx1G -jar Mercury.jar
```

In this example, `-Xmx1G` configures the JVM to use up to 1 GB of heap memory. The value can be adjusted as required by replacing `1G` with a different size, e.g. `512M` (for 512 MB) or `2G` (for 2 GB), depending on the available system memory.

1.2 Graphical User Interface (GUI)

Mercury offers seven different views: (i) RBD, (ii) FT, (iii) EFM, (iv) SPN, (v) CTMC, (vi) DTMC, and (vii) ET. In this section, we briefly describe each of these views. Each formalism has its own section and more details about each view can be found in the respective section.

1.2.1 RBD View

The Reliability Block Diagram (RBD) is a success-oriented modeling approach and allows the creation of a visual representation of a system that shows how components contribute to the failure or success of a system. The RBD view (see Figure 3) provides features for performing reliability and availability analysis for large and complex systems using blocks. The types of block configurations supported by the tool are series, parallel, and k-out-of-n. It also provides solution by closed-form equations, so results are usually obtained faster than by simulation or numerical solutions of other models. In addition, users can add labels and run experiments for a specific component. When you create 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, indicating that its properties have not yet been defined. For more information about the support for RBDs provided by Mercury, see Section 3.

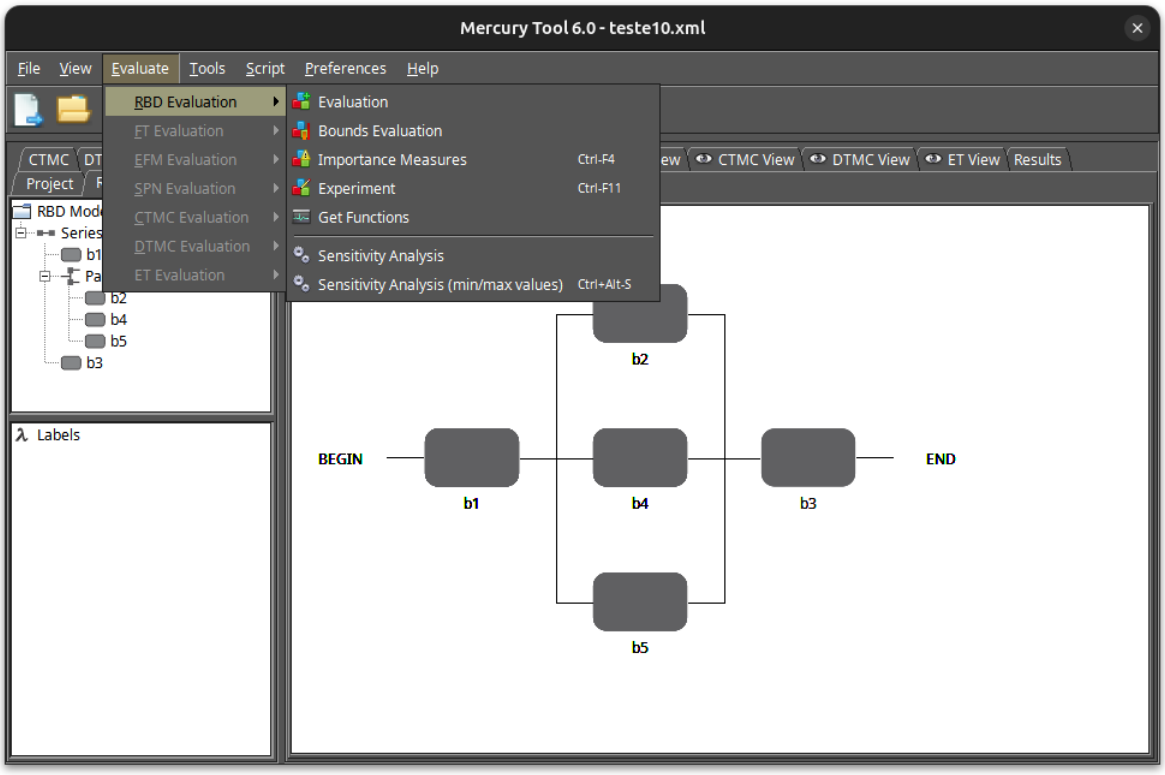


Figure 3: RBD View

1.2.2 FT View

Fault Trees (FTs) and RBDs differ in their purpose. FT is a top-down logical diagram that allows you to create a visual representation of a system that shows the logical relationships between the associated events and causes that lead to a system's failure. When you create 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 called "undefined" because no event leads to it. See Section 4 for more information about Mercury's support for FTs.

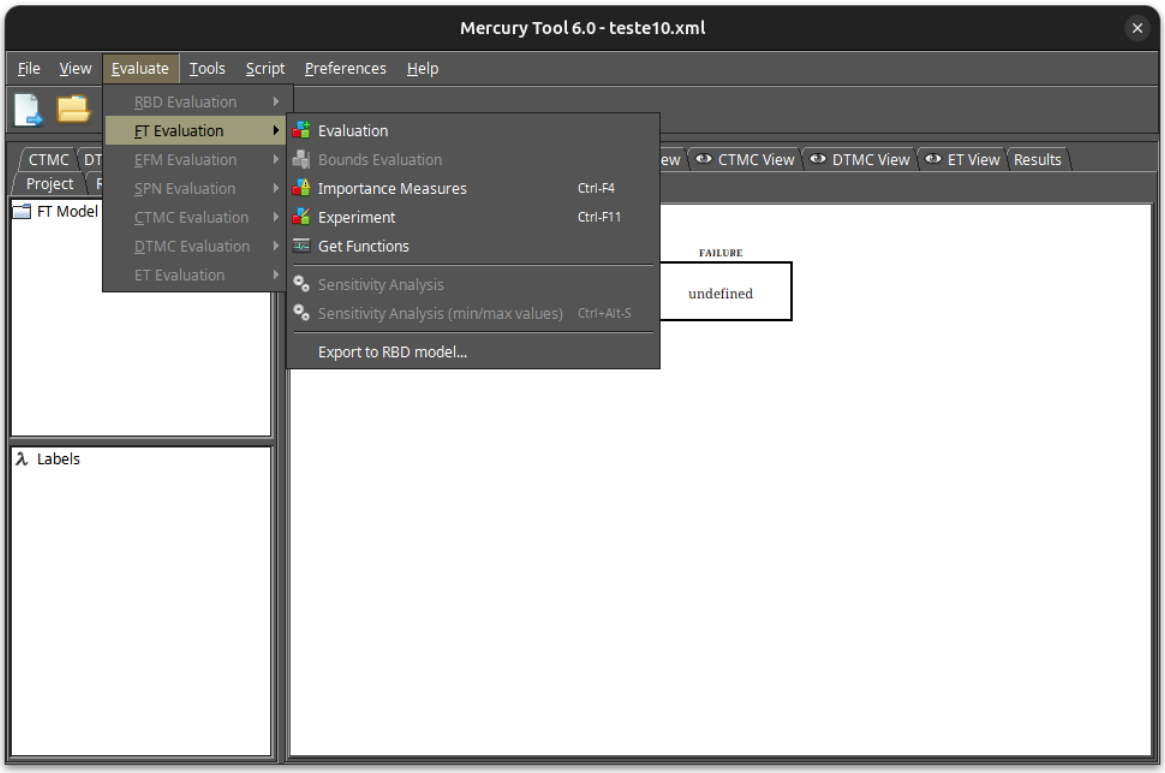


Figure 4: FT View

As we mentioned earlier, you need to install the JavaFX package to make the Fault Tree module available on Linux-based distributions. You can download it from the following URL `https://www.oracle.com/technetwork/pt/java/javafx/downloads/index.html` or install it from a Linux terminal. For Microsoft Windows systems, no additional packages need to be installed.

1.2.3 EFM View

The Energy Flow Model (EFM) view provides functionality to calculate sustainability and cost estimates for data center power and cooling infrastructures, taking into account the energy constraints of individual devices. EFM models represent the flow of energy between system components in terms of their respective efficiency and the maximum energy each component can deliver (for electrical devices) or the maximum cooling capacity (for cooling devices). Figure 5 represents an example of an EFM model. See Section 8 for more information about the support Mercury provides for EFMs.

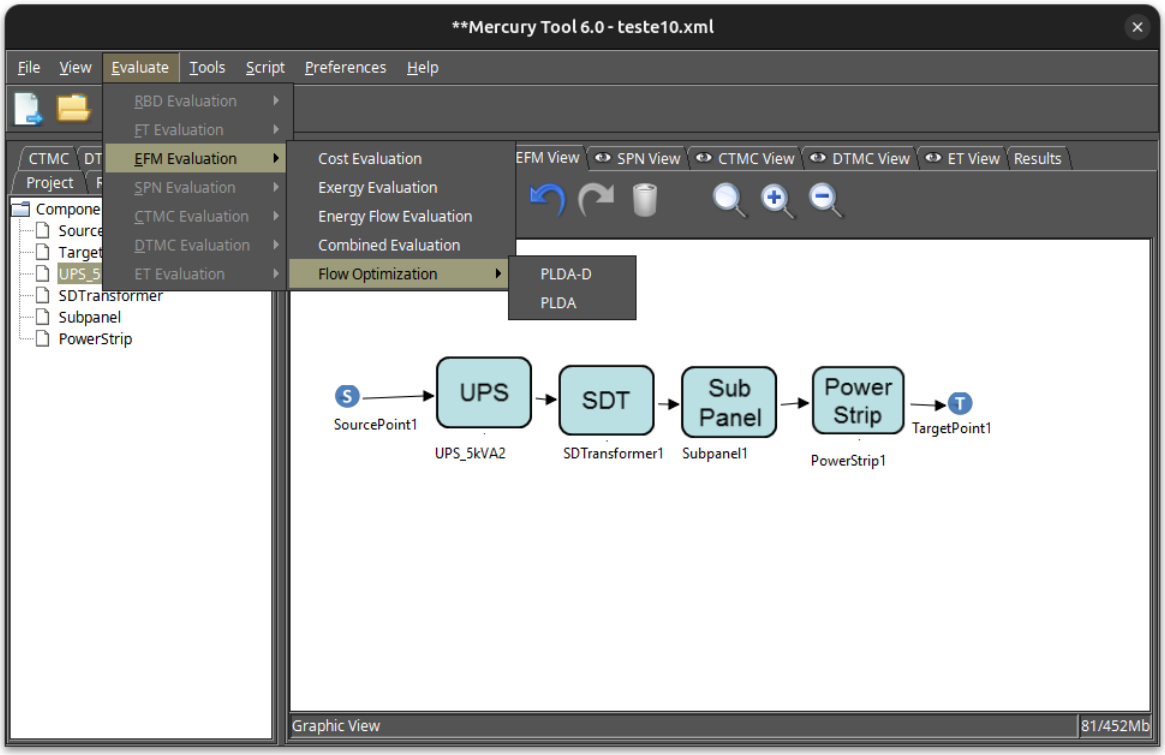


Figure 5: EFM View

1.2.4 SPN View

With respect to stochastic Petri nets, Mercury allows evaluations to be performed by simulation or numerical analysis (i.e., numerical solution of the underlying Markov chains). Both types of evaluations allow the computation of **transient** and **stationary** metrics. Time-dependent metrics are obtained by performing transient evaluations, while stationary metrics are obtained by performing stationary evaluations. Figure 6 shows the SPN view with an SPN model as an example. See Section 2 for more information about Mercury's support for SPNs.

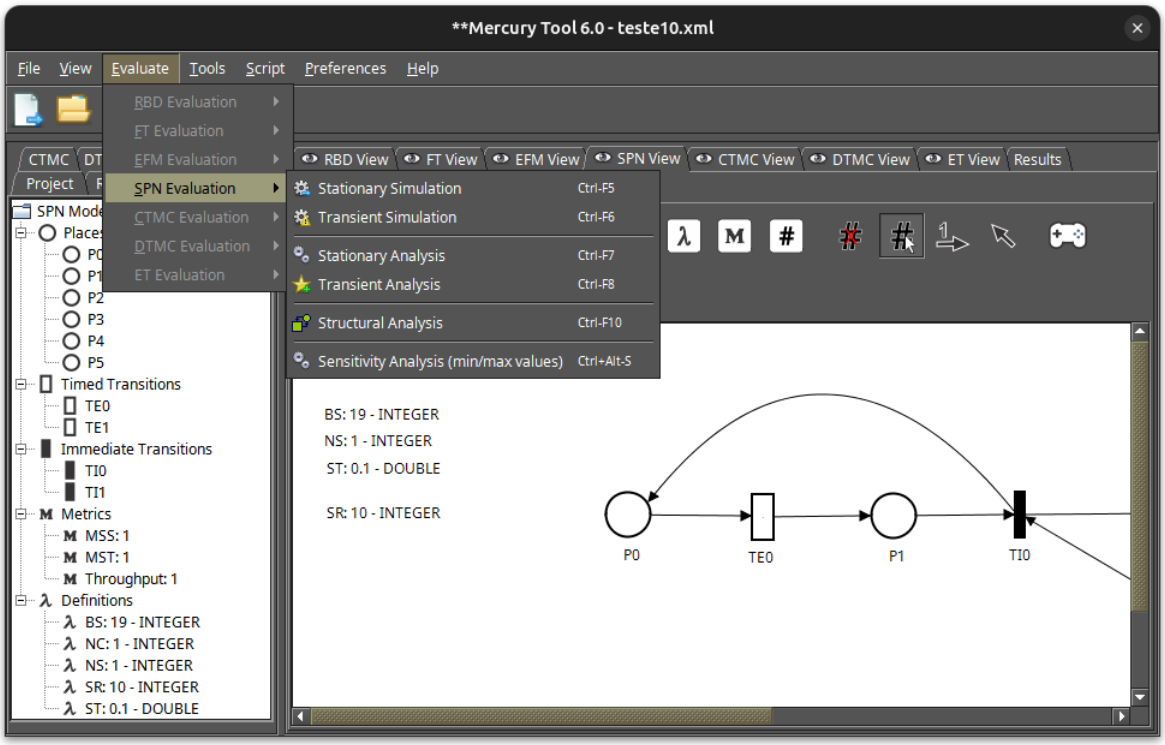


Figure 6: SPN View

1.2.5 CTMC View

The CTMC view provides features for drawing and evaluating continuous-time Markov chains (see Figure 7). Numerical solutions of CTMCs can be performed by stationary or transient analyzes. There are two methods for computing stationary metrics: GTH (Grassmann-Taksar-Heyman) and Gauss-Seidel. Transient metrics are calculated by default using the ”Uniformization” method (also known as Jensen method), but the user can also use the ”4th-order Runge Kutta” method. Sensitivity analysis is also available in the CTMC view. The rate of each state transition can be defined using polynomial expressions related to user-defined variables (referred to as parameters/definitions on Mercury). Parameter names may contain Greek letters. In addition to states and transitions, users can also define reward rates associated with states. In such a case, CTMCs become Markov reward models. For models with absorbing states, Mercury also allows users to calculate the probability of absorption and the mean time to absorption. The user can create custom metrics by formulating expressions that can contain state probabilities. Parameters and metrics can be easily viewed and modified in the CTMC editor. For more information on the support Mercury provides for CTMCs, see Section 5.

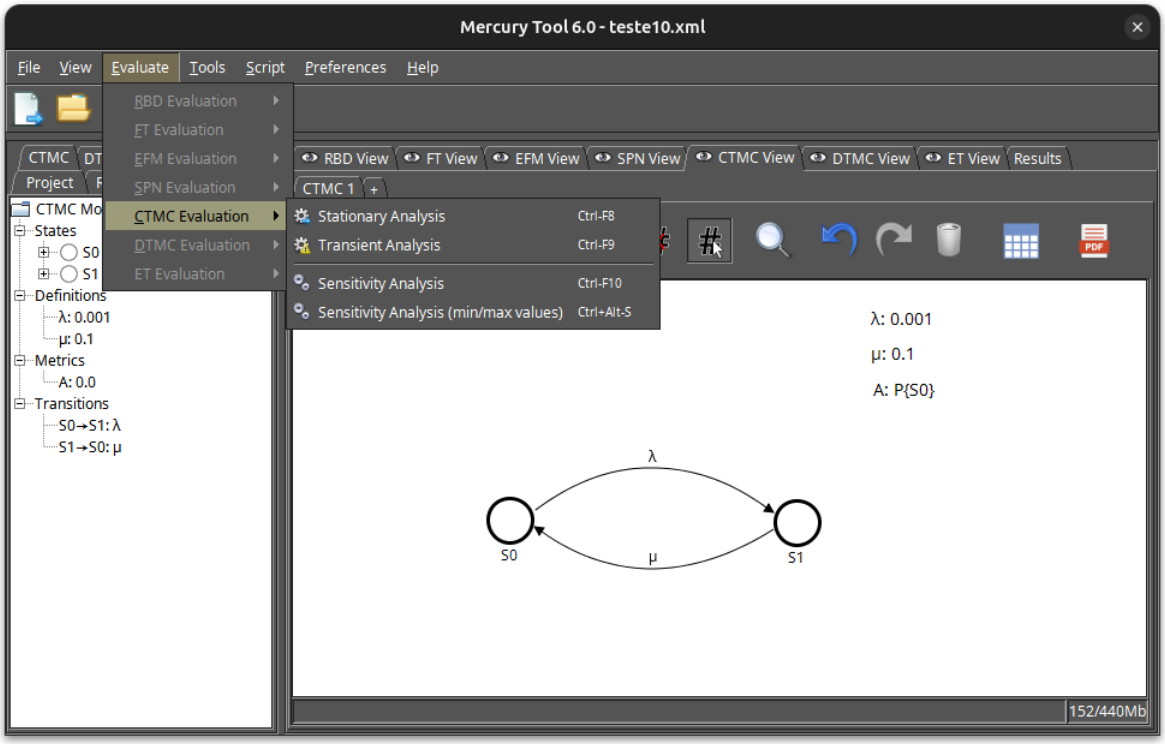


Figure 7: CTMC View

1.2.6 DTMC View

The DTMC view provides features for drawing and evaluating discrete-time Markov chains (see Figure 8). Numerical solutions of DTMCs can be performed by stationary or transient analyzes. There are two methods for computing stationary metrics: GTH (Grassmann-Taksar-Heyman) and Gauss-Seidel. The parameter names may contain Greek letters. For models with absorbing states, Mercury also allows the calculation of the absorption probability and the mean time to absorption. The user can create custom metrics by formulating expressions that refer to state probabilities. Parameters and metrics can be easily viewed and modified in the DTMC editor. For more information about the support Mercury provides for DTMCs, see Section 6.

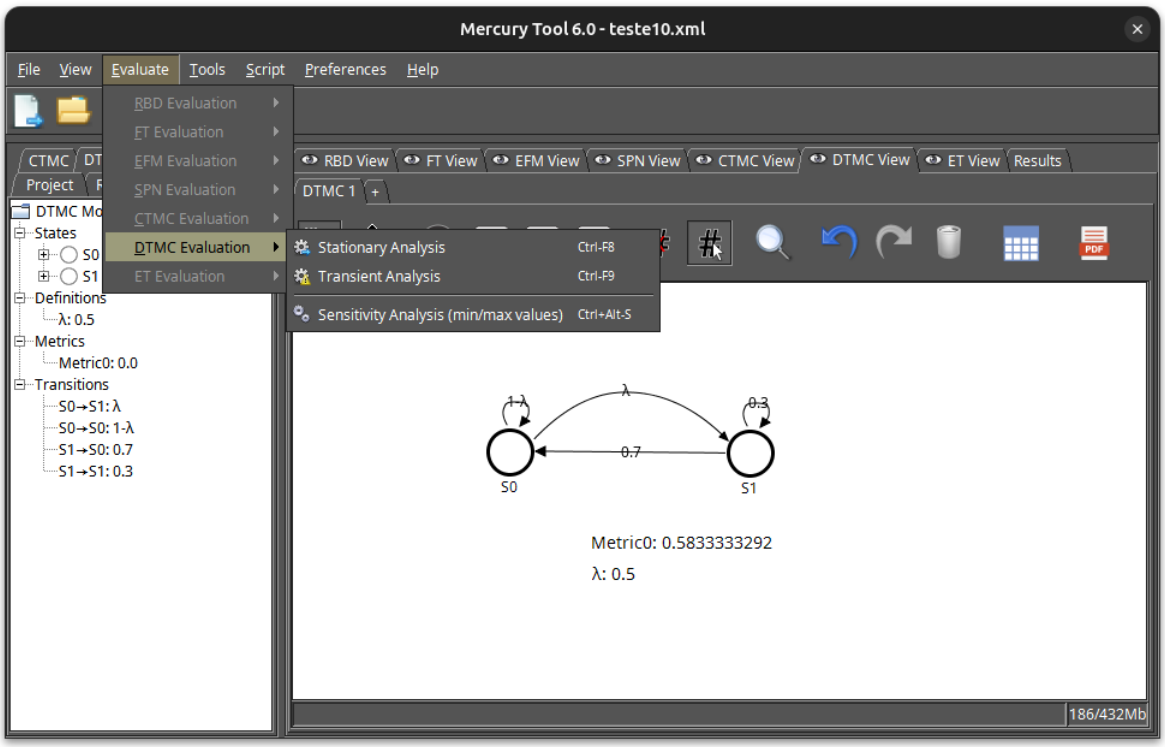


Figure 8: DTMC View

1.2.7 ET View

The ET view provides features for drawing and evaluating Event Trees (see Figure 9). Event Trees offer an easy way to draw event-based models. An ET is composed of nodes, which represent the events of the model, and transitions between nodes, which represent the probability of the corresponding event. Section 7 details ET modeling.

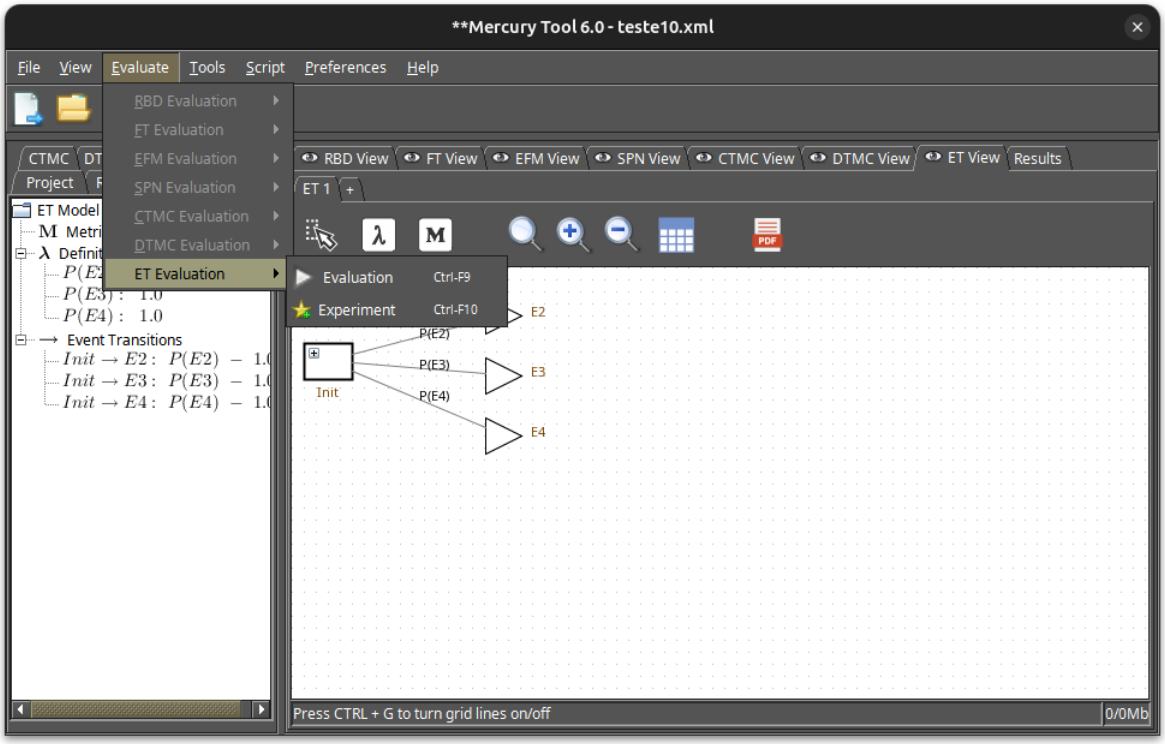


Figure 9: ET View

1.3 Main Menu

Mercury's main menu is shown in Figure 10. As we can see, Mercury has seven main menu items. Some menu items in each main menu item have keyboard shortcuts associated with them. To illustrate this, Figure 11 shows the options available in the **File** menu.

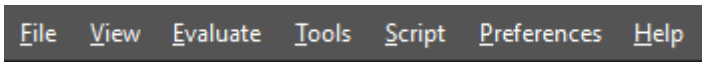


Figure 10: Main Menu

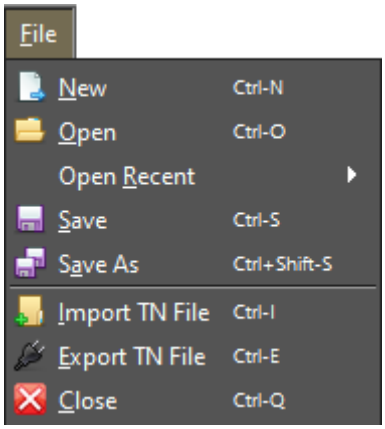


Figure 11: Menu File

Next, we describe the options available in the File menu.

- **New.** Create a project. When you create a project, all modeling views are made available and initialized empty, except for the RBD, FT and ET views, which each start with a default component with no probability assigned. *Shortcut: Ctrl + N*

- **Open.** Open a project. Mercury only allows you to open files in the Mercury project file format with the ".xml" extension. However, Mercury allows importing models created in other engines that use the ". TN" standard format. Select the option "Import TN File" to import files in this format. *Shortcut: Ctrl + O*

- **Open Recent.** Here you can see a list of the twenty-five latest projects.

- **Save.** Save the latest project changes to the current file or to a new file for a new project. When you save a project for the first time, a window appears where you can select 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 specifying a new location and name for the file. *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*

Let us now describe the View menu (see Figure 12). In this menu the user can show/hide the views provided by the tool: RBD, FT, EFM, SPN, CTMC, DTMC and ET. Figures 13 and 14 show the main window with the seven views visible and hidden respectively.

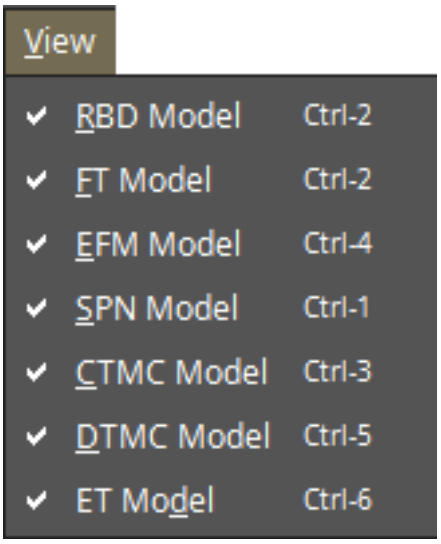


Figure 12: Menu View

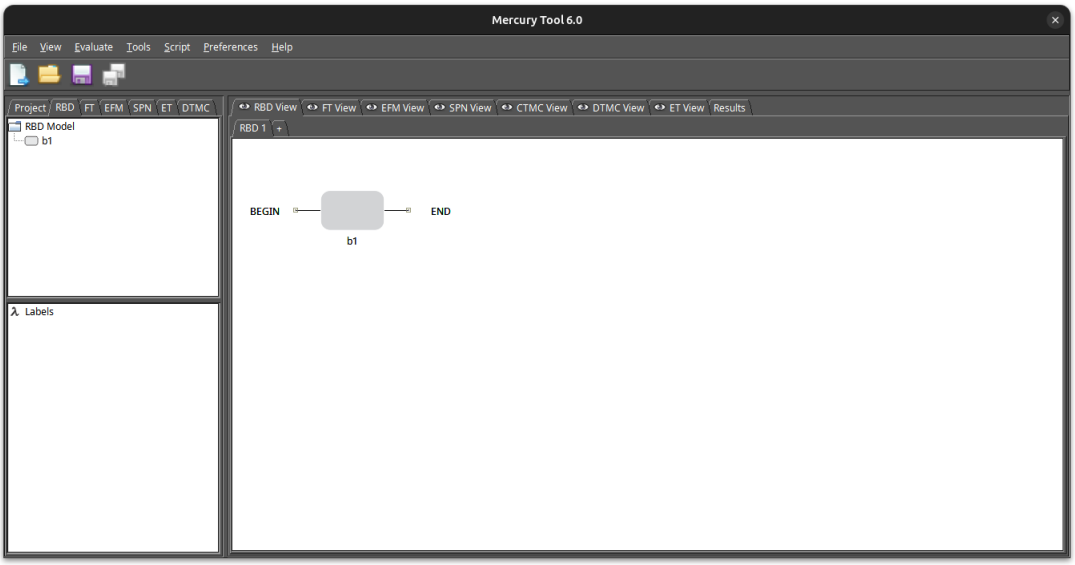


Figure 13: Mercury with the Seven Views Visible

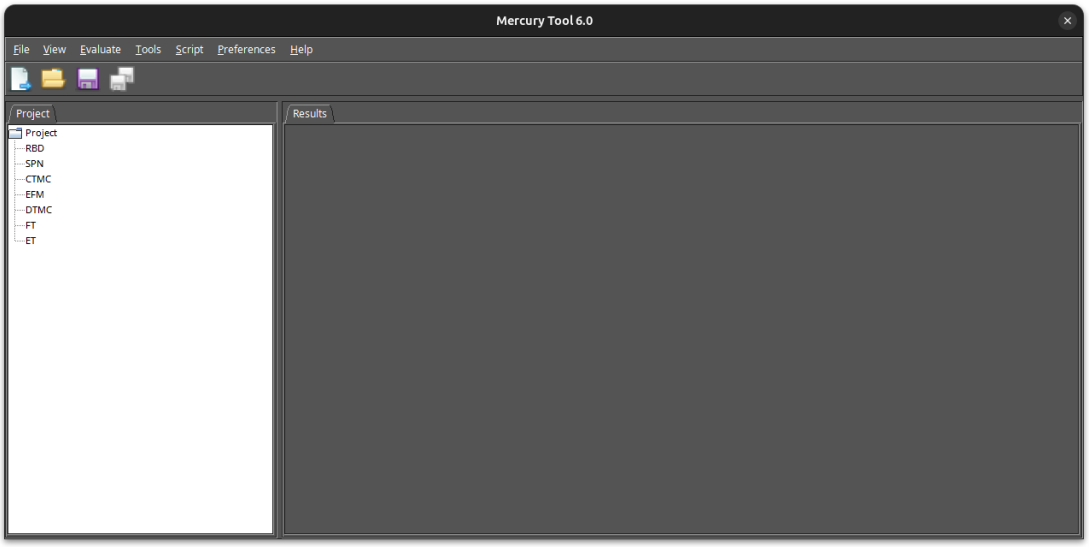


Figure 14: Mercury with the Seven Views Hidden

Let us now describe the Evaluate menu (see Figure 15). This menu contains a menu group for each formalism supported by the tool. A menu group is only active when the corresponding model view is active in the main window. The menu items available in each menu group for each formalism are described in the following sections.

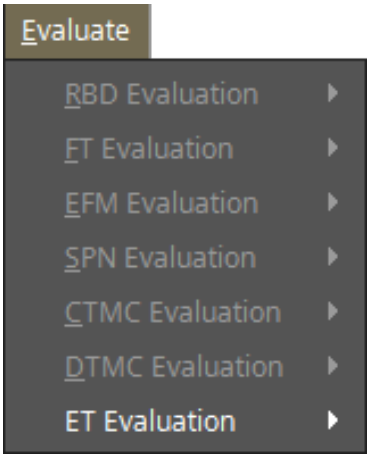


Figure 15: Menu Evaluate

Next, we describe the options available in the Tools menu (see Figure 16).

EMA Tool. An Expectation-Maximization Auto-fitting (EMA) algorithm is a fundamental technique for parameter estimation in statistical models when handling clustered data. The EMA tool implements an algorithm that provides a robust approach to parameter estimation in complex models. It is suitable for scenarios where data points have unknown probabilities of belonging to different clusters and these clusters follow known distributions with unknown parameters. The main goal of the EMA tool is to iteratively estimate the parameters for each cluster such that they maximize the likelihood of the observed data. This iterative process starts with the initial parameter values, computes the posterior probabilities, optimizes the

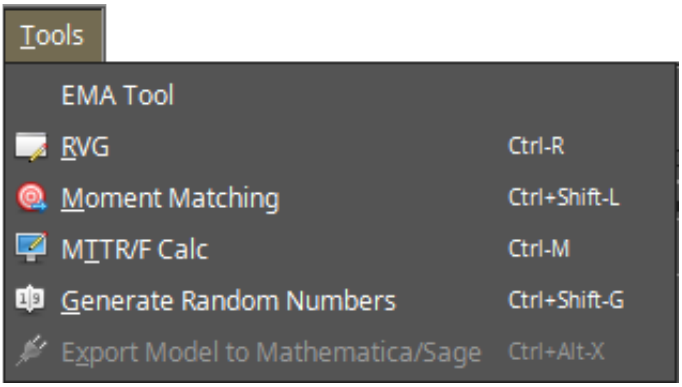


Figure 16: Menu Tools

parameters, and repeats until convergence, i.e., until the change in the incomplete log-likelihood is less than a predefined threshold. The EMA tool supports two types of evaluations: simple and random search. In simple evaluation, the user specifies the punctual values to be considered for the number of clusters and number of phases parameters (see Figure 18). In random search evaluation, the algorithm tries to find the number of clusters and phases that provide the best convergence considering the minimum and maximum acceptable values for each parameter (see Figure 19). The plot shows how the model fits the actual data (see Figure 20). This allows users to assess how well the model describes the behavior of the data. The tool can also extract expressions from the fitting (see Figure 21). The EMA tool has applications in clustering, density estimation, and probabilistic modeling. See Section C for more information.

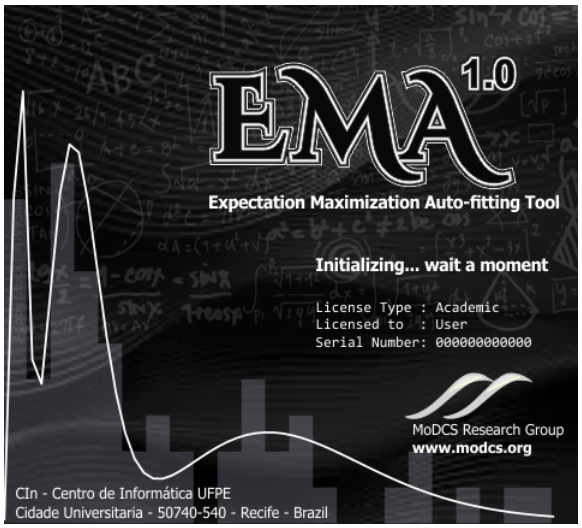


Figure 17: The Expectation Maximization Auto-fitting Tool

RVG stands for **Random Variate Generator**. A module to support the generation of random numbers, providing a large number of probability distributions. Statistical summaries considering the generated numbers are also provided, such as standard deviation, variance, mean, skewness, and kurtosis (see Figure 22). Results can be exported for supporting analyzes using other software. RVG is used by the SPN simulator, which supports the evaluation of models with non-exponential times.

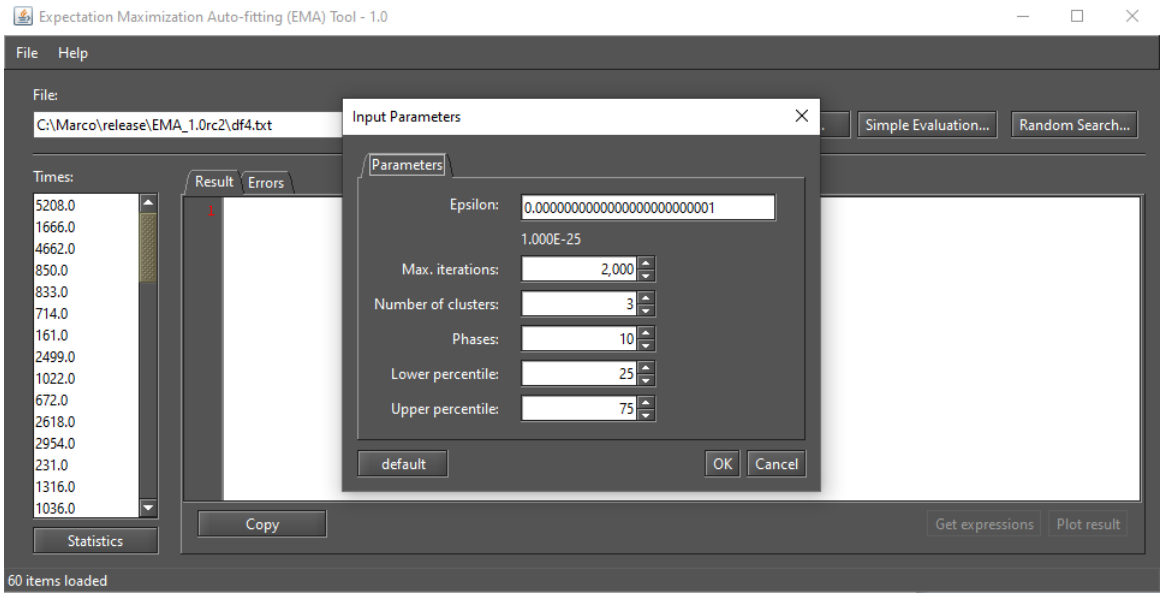


Figure 18: EMA - Simple Evaluation

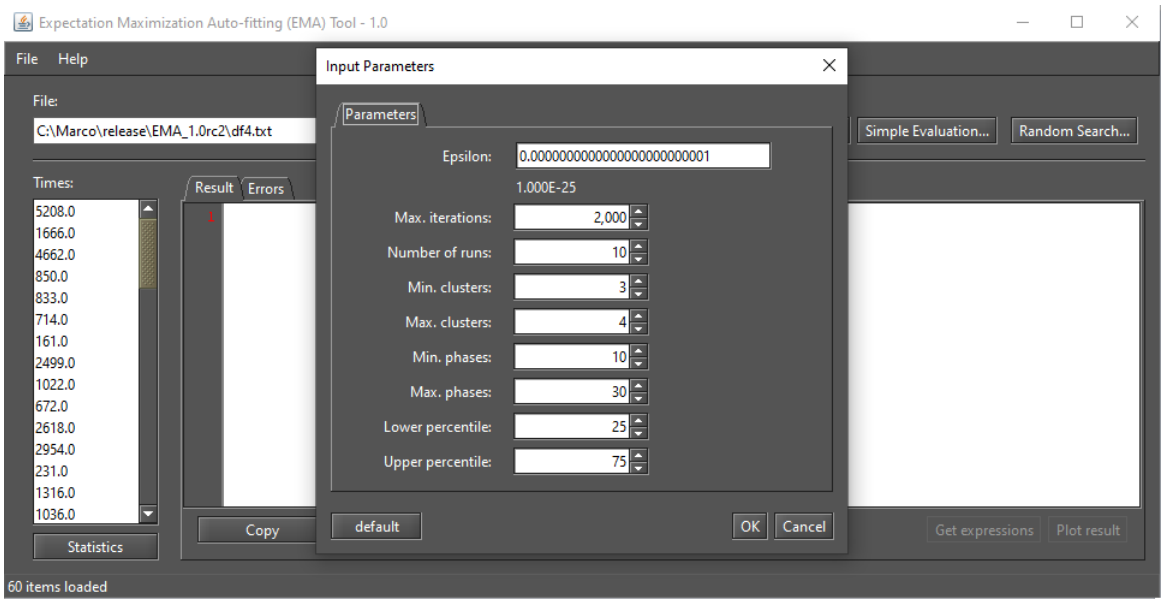


Figure 19: EMA - Random Search

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 23). By supporting numerical evaluations of metrics for models that have non-exponential times associated with them.

MTTR/F Calculator. Enables calculation of MTTR (mean time to repair) and MTTF (mean time to failure) using the availability and reliability curves as input parameters. Reliability is a time-dependent metric that indicates the probability of something working under certain conditions over a given period of time.

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

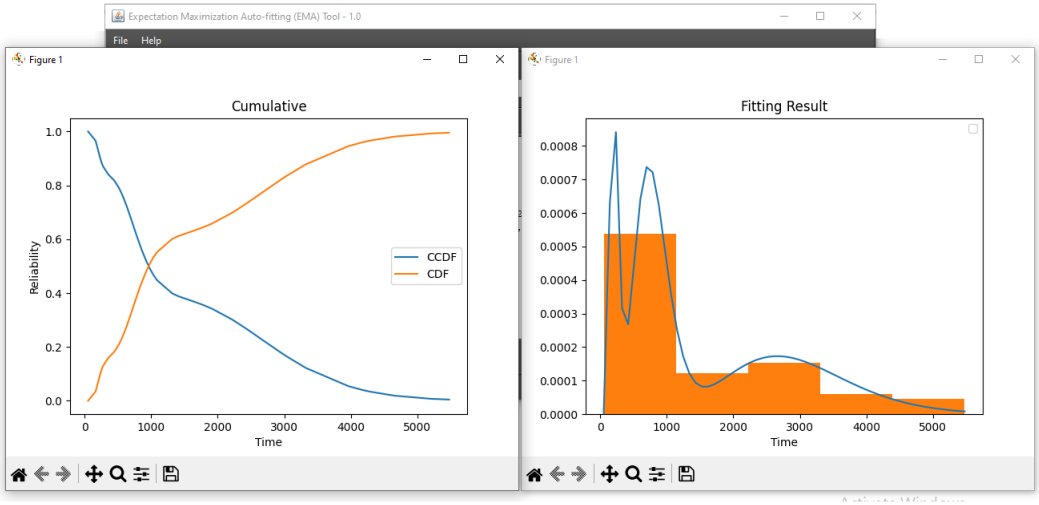


Figure 20: An EMA Fitting Result

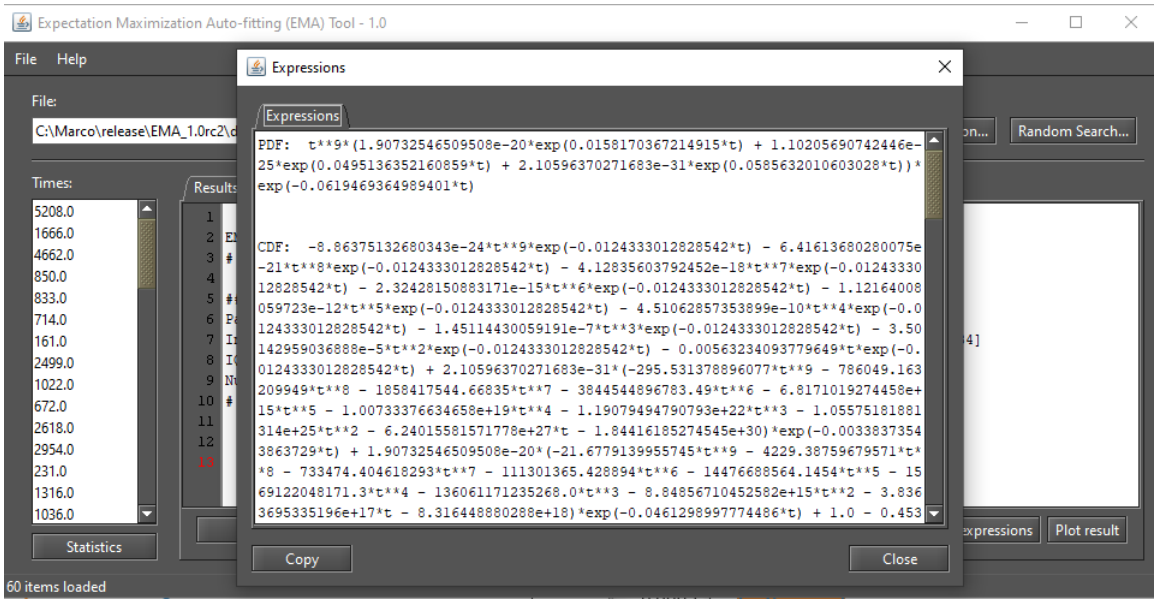


Figure 21: Expressions from an EMA Evaluation

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

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):

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

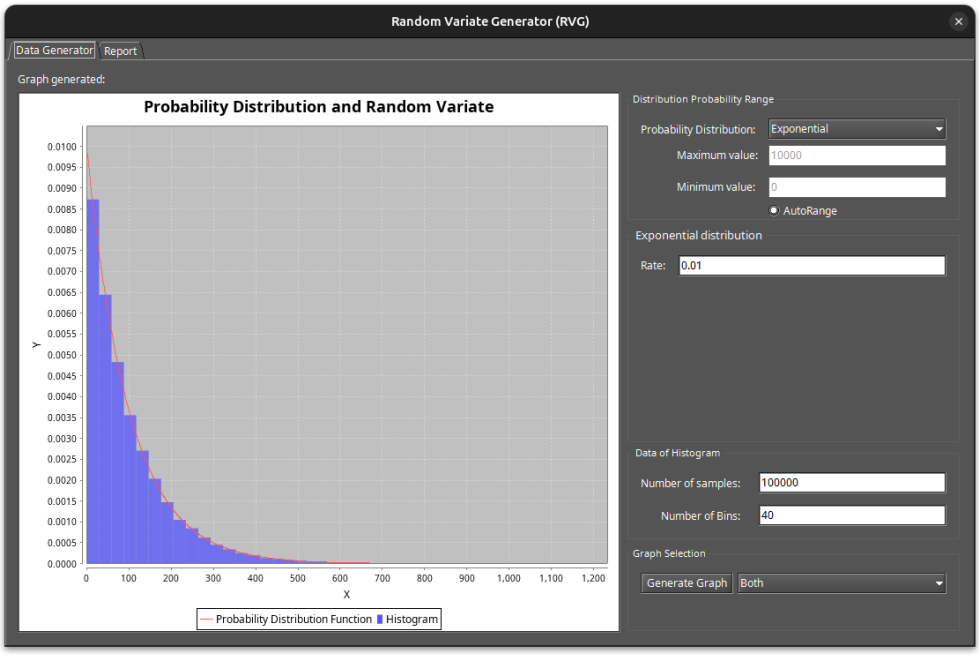


Figure 22: RVG Module

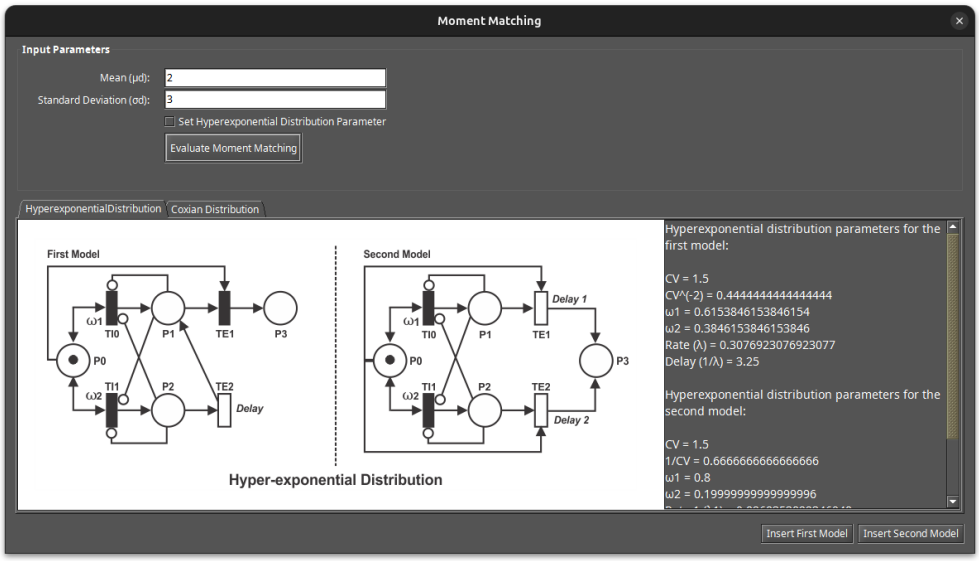


Figure 23: Moment Matching

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

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

Figure 25 shows the MTTR/F Calculator window where users should specify *Availability* and a CSV file without headers and with two columns to specify the evaluation time and reliability value. Using the *File* button we can refer to a file containing time and reliability values.

After specifying the parameters, the user should calculate the results by pressing the *Calculate* button.

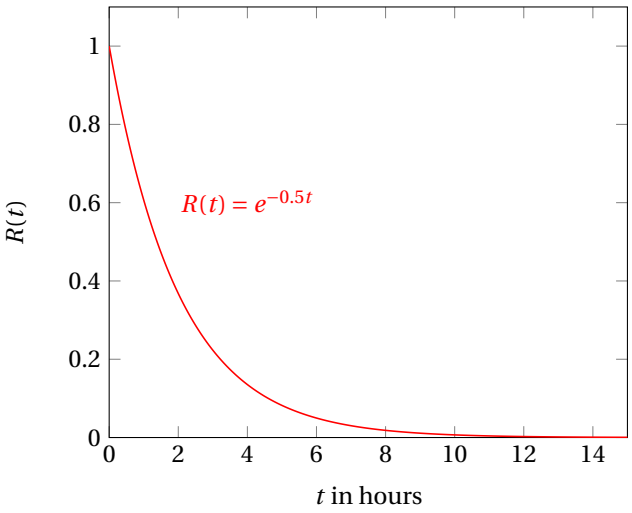


Figure 24: Reliability Over Time

Figure 26 shows an example of a result obtained by calculating MTTR and MTTF values. For a more detailed example, see the following video `https://youtu.be/Hmu5DX3CJCg`.

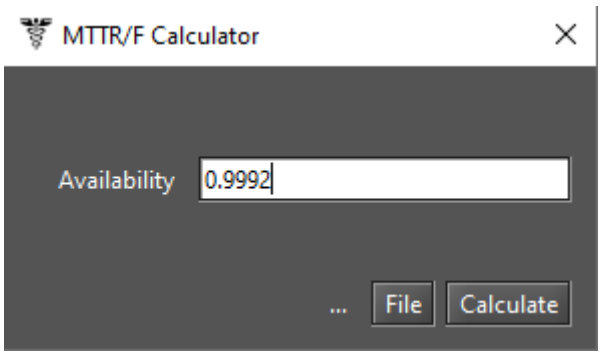


Figure 25: MTTR/F Calculator

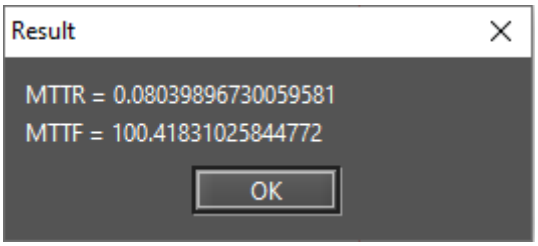


Figure 26: MTTR/F Calculator Result

Evaluate External RBDs. Module that allows us to calculate availability metrics for external RBD files created according to a specific format.

Generate Random Numbers. This module allows us to generate random numbers that follow a certain probability distribution. It supports a large number of probability distributions. First, the user must enter the size of the sample and select the probability distribution that will guide the generation of the numbers. After that, you need to specify the value for each parameter of the selected distribution. Before you start

generating numbers, you need to select the location where the file with the generated sample will be saved.

Export Model to Mathematica/Sage. This feature exports SPN/CTMC models to Wolfram Mathematica language/SageMath format (see Figure 27). When exporting to Mathematica, Mercury creates the nb file to be opened in Mathematica.

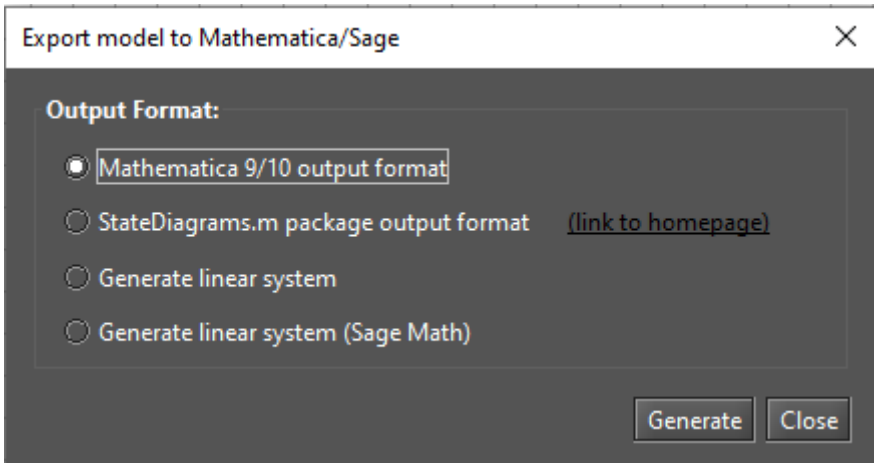


Figure 27: Export model to Mathematica/Sage

Let us now describe the menu “Script” (Figure 28). In this menu, the user can generate the representation of the active model in the scripting language format or create a script from scratch. Clicking the ”Generate script” menu converts the active model into a script so that it can be evaluated and modified in the script editor. Figure 29 shows the script editor with a script representation of an SPN model. Mercury also provides scripts as examples. To open them, simply click on the menu item representing the script and it will open in the script editor. See Chapter 10 for more information about the Mercury scripting language and its grammar.

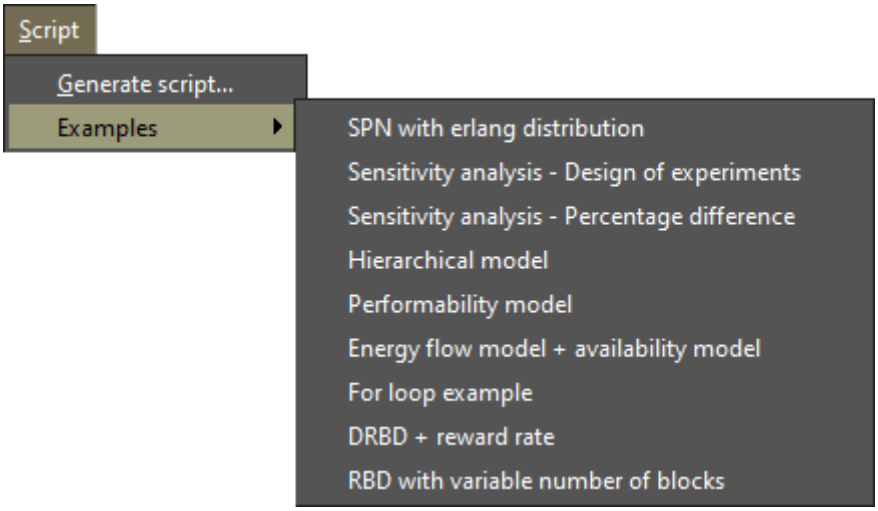


Figure 28: Menu Script

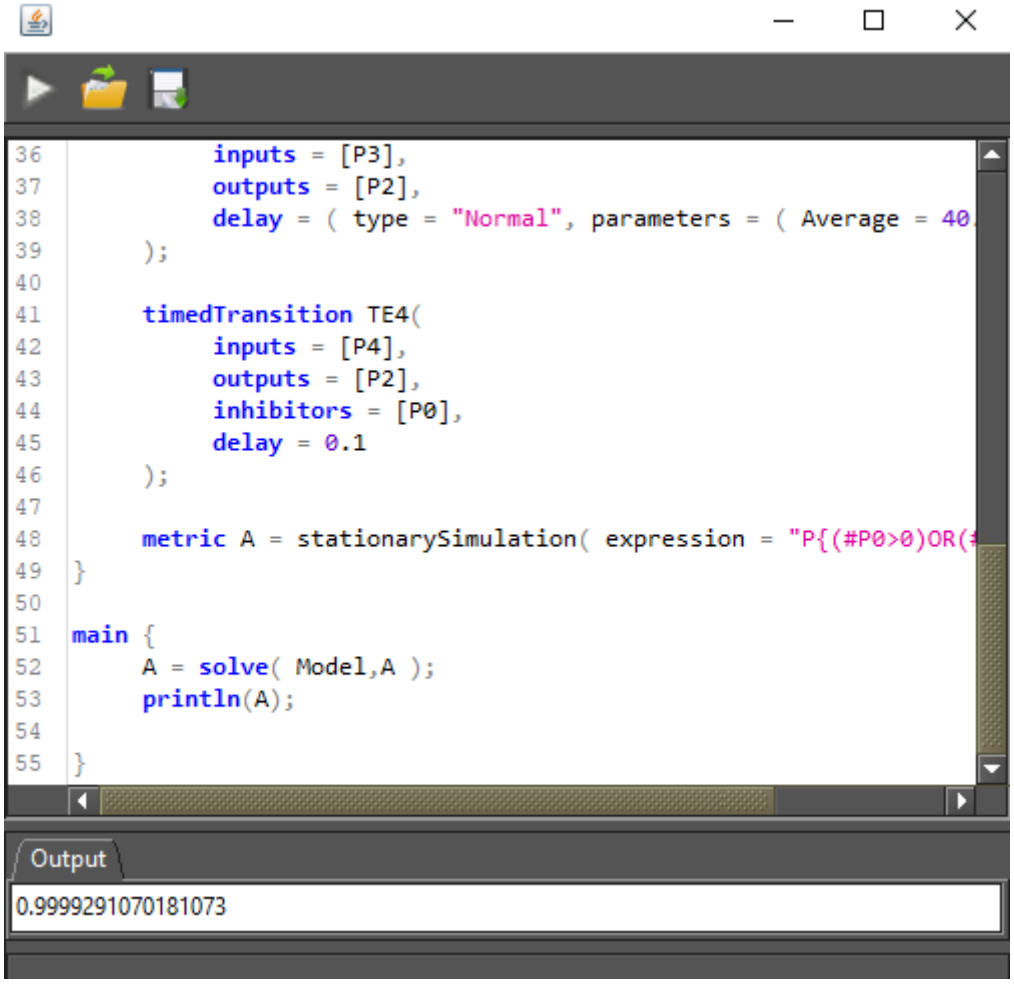


Figure 29: Script Editor

1.4 Main Toolbar

The main toolbar provides access to some of the most commonly used features in Mercury, such as creating or saving a project. It appears at the top of the main window, just below the menu bar. Figure 30 shows the command buttons on this toolbar, each represented by an icon. The following items describe each of the buttons.

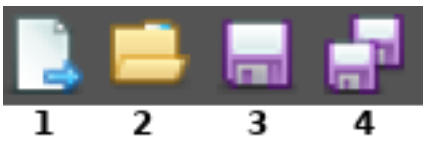


Figure 30: Main Toolbar

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*

1.5 Drawing Area

Mercury supports seven formalisms and provides a modeling view for each of them — RBD, FT, EFM, SPN, CTMC, DTMC and ET. It also provides another view to display the results produced by the simulators. Click on the View tab in the main window to activate the respective visualization. Views can be made visible and invisible. To do this, the view must be checked or unchecked on the View menu.

The drawing area is an empty area where you can add components for a formalism. To add a component, you usually have to click on a button that represents the component in the toolbar. After that, you must click on the desired location within the canvas to insert the selected component there — except for RBDs, ETs and Fault Trees. Figure 31 shows the drawing area of the SPN view.

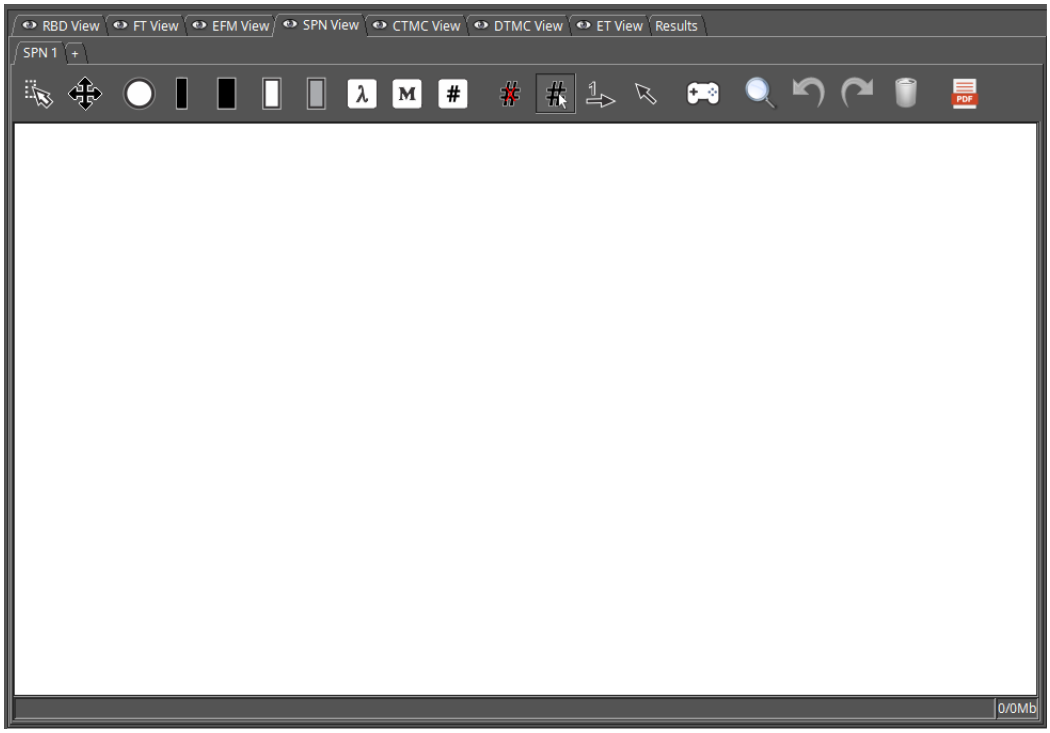


Figure 31: SPN Drawing Area

On the left side of the main window, there is a tab for each formalism, as we can see in Figure 32. When you access these tabs, all the components that make up the current model are displayed. You can access a component's properties by double-clicking on it. In this example, the SPN tab is active. Each time a component is inserted into the drawing area or one of its properties is updated, the panel on the left is also updated.

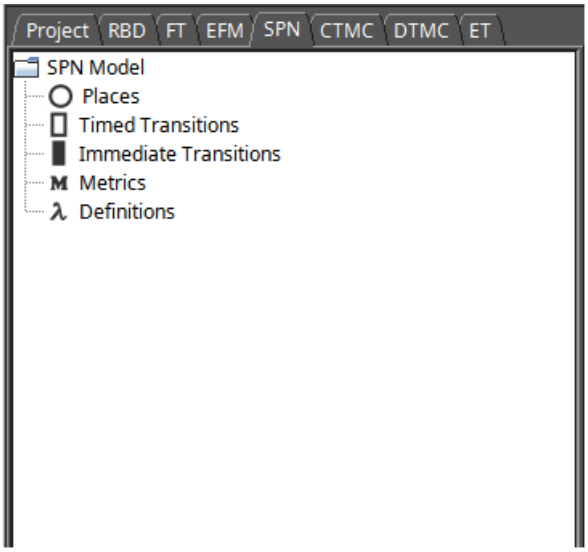


Figure 32: Left-Side Panel

The editor provides smoother handling of components on the canvas, allowing users to move and position elements effortlessly. It provides visual assistance through vertical and horizontal alignment lines when adding, resizing, or moving components on the canvas, ensuring precise alignment (Figure 33). In addition, users can fine-tune component positioning with the keyboard arrow keys for precise control over element placement.

Mercury also allows the user to right-click and select “Enable Grid” option or press the CTRL + G keys to activate and deactivate grid mode. When active, this mode draws several lines on the canvas to help align components, as shown in Figure 34. These features give the user complete control over layout and positioning, improving the presentation of the model. In addition to that, the user can also right-click and select “Enable Snap to Grid” or press CTRL + H keys to activate and deactivate snap to grid mode. When active, this mode allows objects to be be moved in steps based of the grid.

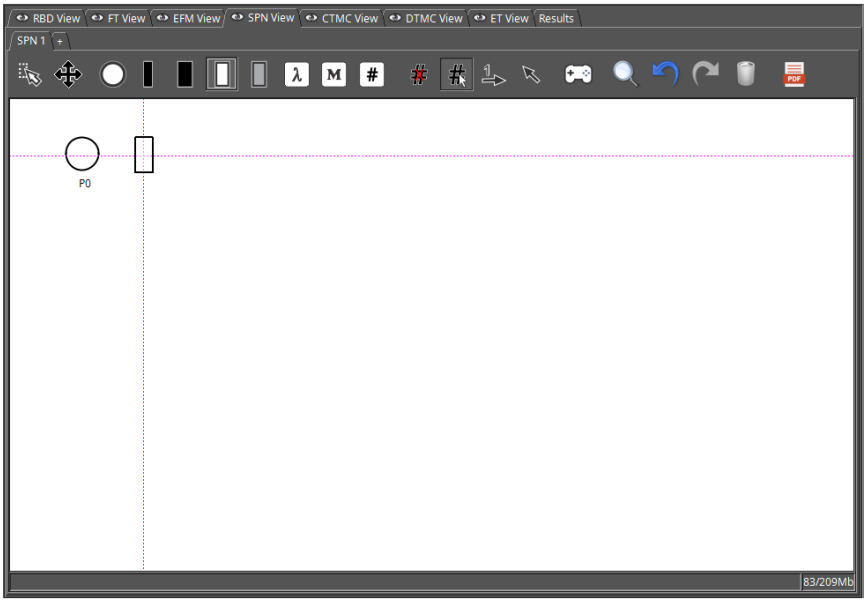


Figure 33: Editor Showing Alignment Lines

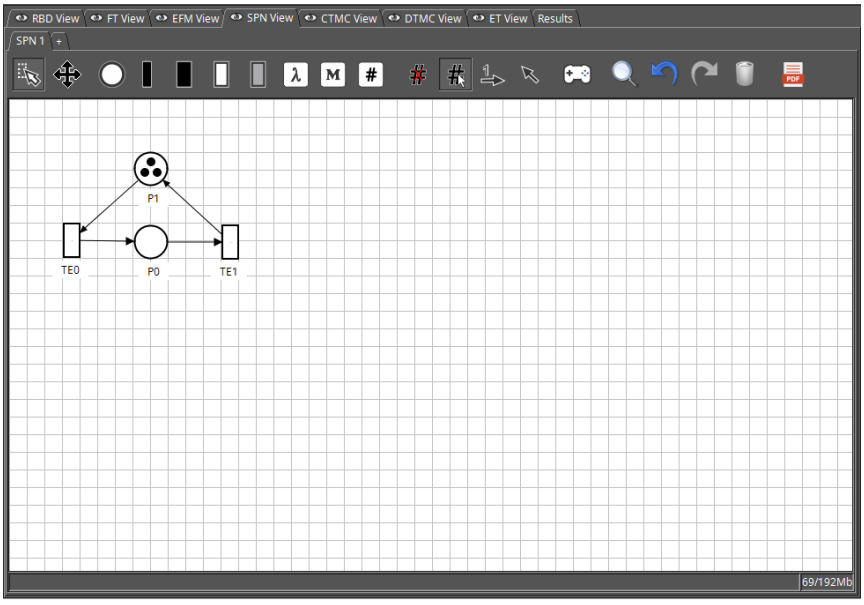


Figure 34: Grid Lines

Mercury supports the creation of projects that allow 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 shown in Figure 35. A new tab will then be created (see Figure 36). The user also has the option to rename,

remove or duplicate the model. These options are available from the popup menu that appears when the model tab is right-clicked (see Figure 37). If the user chooses the ”Rename” or ”Duplicate” option, the new name of the model can be entered in the dialog (see Figure 38).

Figure 35: Adding a New Tab

Figure 36: SPN with Two Models

Figure 37: Popup Menu for Tabs

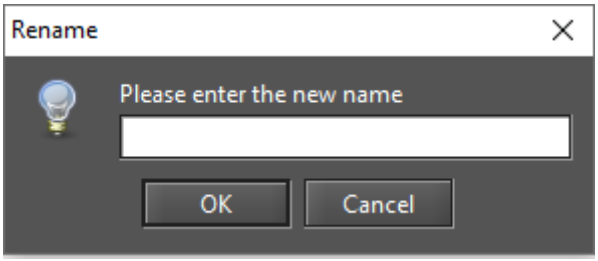
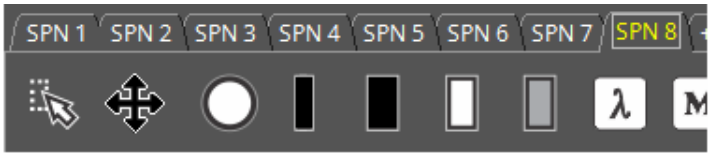
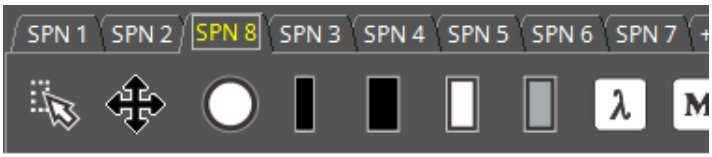


Figure 38: Renaming a Model

Mercury now supports drag-and-drop rearrangement of model tabs. This allows you to customize the layout of your workspace by simply clicking on a tab and moving it to the desired location (see Figure 39). By allowing tabs to be rearranged at will, the feature improves control over the workspace so that it is better tailored to the user’s specific needs and improves the overall user experience. This feature is especially valuable for projects where users frequently work with multiple models and need a flexible way to keep everything well organized and easily accessible.



(a) Before



(b) After

Figure 39: Tab Reordering with Drag and Drop

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 40 shows a model with two transitions — one timed and one immediate —, and two places. Below we describe the process of modeling SPNs with Mercury.

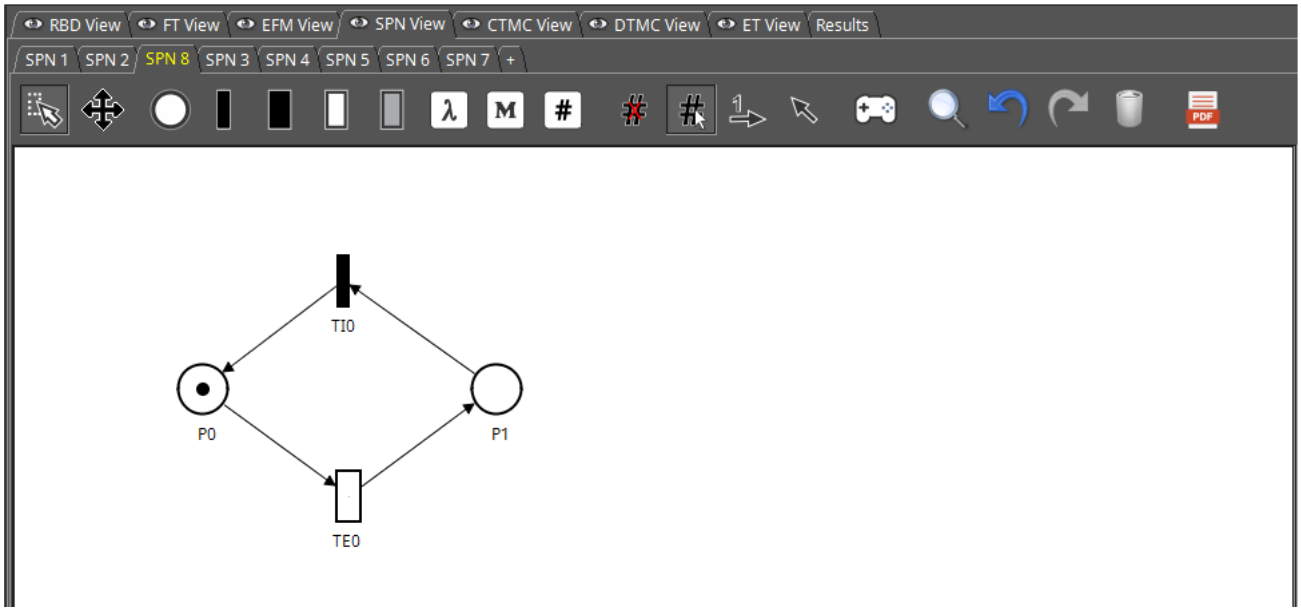


Figure 40: SPN Model

Figure 41 shows the SPN toolbar. Some buttons on the SPN toolbar are used to model SPNs. This toolbar is visible when the SPN view is active. In the following we describe each button.

Figure 41: SPN Toolbar

1. **Selection.**

Turns on the selection mode. This mode allows you to select components on the drawing area. When this mode is enabled, you can select more than one component in the drawing area by holding down the SHIFT key while clicking on the components. Another way to select a group of components is to create a selection area. A selection area is created by holding down the left mouse button while moving the mouse. All components that are in this area will be selected. Also in this mode, arcs can be created between places and transitions when pointing at the center of the place/transition and moving to a transition/place.

2. **Just move elements.**

Turns on the just move elements mode. This mode allows you to only select and move components on the drawing area. When this mode is enabled, you can select and move more than one component in the drawing area by holding down the SHIFT key while clicking on the components. Another way to select a group of components is to create a selection area. A selection area is created by holding down the left mouse button while moving the mouse. All components that are in this area will be selected. Creating arcs is not possible in this mode.

3. **Place.**

Adds places to the model.

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

By default, new places do not have tokens.

4. **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.

5. **Deterministic Transition.**

Inserts deterministic transitions into the model.

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

6. **Exponential Transition.**

Adds exponential transitions to the model.

Click the “Exponential Transition” button and then click the desired location in the drawing area.

7. **Non-Exponential Transition.**

Adds non-exponential transitions to the model.

Users should click on the “Non-Exponential Transition” button and then click on

8. **Definition.**

Adds definitions to the model. Click the ”Definition” button, and then click the desired location in the drawing area. A definition is a variable that stores a numeric value. It may be associated with some properties of other SPN components. In this case, Mercury takes into account the current value of the definition for the property being referenced. References are made by entering the name of the definition as the value of the property or within an expression. Definitions can be associated with 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 can refer to the same definition. Definitions are useful to support experiments. In this case, by changing the value of a definition, you can evaluate the effect of that change on a metric.

9. **Metric.**

Adds metrics to the model. Click the “Metric” button and then click the desired location in the drawing area. Basically, a metric is an expression used to evaluate a property of the model. A metric can be useful for evaluating whether a certain state has been reached or how much time it took to perform a certain activity.

10. **Comment.**

Adds comments to the model. Click the “Comment” button and then click the desired location in the drawing area. Then, double-click the comment element to edit the comment, allowing you to add LaTeX expressions, or remove any text and keep the borders to create a dashed rectangle to highlight objects. This is detailed in Section 9.

11. **Hide Comments.**

Hides comments in the model. Click the “Hide Comments” button to hide all comments shown in the model. Clicking the button again shows the comments in the model. This is detailed in Section 9.3.

12. **Disable Comments Selection.**

Disables comments selection in the model. Click the “Disable Comments Selection” button to avoid comments from being selected, moved, or edited. Thus, only the model’s elements will be selectable, movable, or editable. This is detailed in Section 9.4.

13. **Show/Hide Arcs Labels.**

Hides/shows the labels above the arcs. This type of label indicates the multiplicity of the arc. The multiplicity of arcs indicates how many tokens are consumed or generated at certain places. In the case of inhibitor arcs, it indicates how many tokens a place must have for a transition not to activate.

14. **Turn Default/Inhibitor Arc Mode On/Off.**

This button allows you to select the type of arc to be used to connect transitions and places. You can choose between two types of arcs: standard and Inhibitor arcs. Inhibitor arcs may only be used to connect places with transitions. When you create a new project, the standard arc is active by default.

15. **Undo.**

Undo recent changes from the model.
All recent changes are stored and can be rolled back, one after another.
Shortcut: Ctrl+Z

16. **Redo.**

Redo recent changes undone to the model.
Shortcut: Ctrl+Y

17. **Remove.**

Removes selected components from the model. If you select a group of components, all components will be removed by clicking this button. You can also delete components by pressing DEL or right-clicking the selected component and choosing ”Remove”. If you remove a place or a transition, its arcs will also be removed.

18. **Default Scale.**

Apply standard scale to the drawing area.

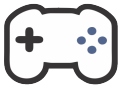
19. **Scale Up.**

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

20. **Scale Down.**

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

21. **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.

22. **Export to PDF.**

This feature allows users to export their models to PDF files.

Now let us look at the interaction within the drawing area. When you right-click on an SPN component, a popup menu appears. All components have a popup menu associated with them that contains at least two items:

- **Remove.** Removes the selected component from the model.
- **Properties.** Displays the ”Properties” dialog box, where the user can change the properties of the selected component.

Figure 42 shows the pop-up menu for transitions.

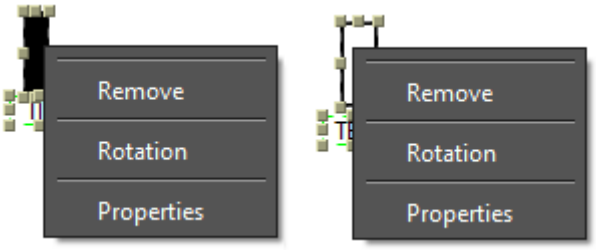


Figure 42: Popup Menu of Transitions

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

- **Rotation.** Rotates the selected transition. Transitions can be positioned horizontally or vertically. Figure 43 shows an immediate transition in horizontal position. All arcs of the transition are readjusted when it is rotated.

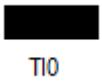


Figure 43: An Immediate Transition Positioned Horizontally

Let us now describe the properties of timed transitions. To view them, you should right-click on the respective transition, as shown in Figure 42, and then click on the ”Properties” item. Another option is to double-click on the timed transition. A third way is to double-click on the transition's representation on the left side of the window. All roads lead to Rome. It is important to emphasize that this last option is available for all components of the model. Figure 44 shows the properties of a timed transition.

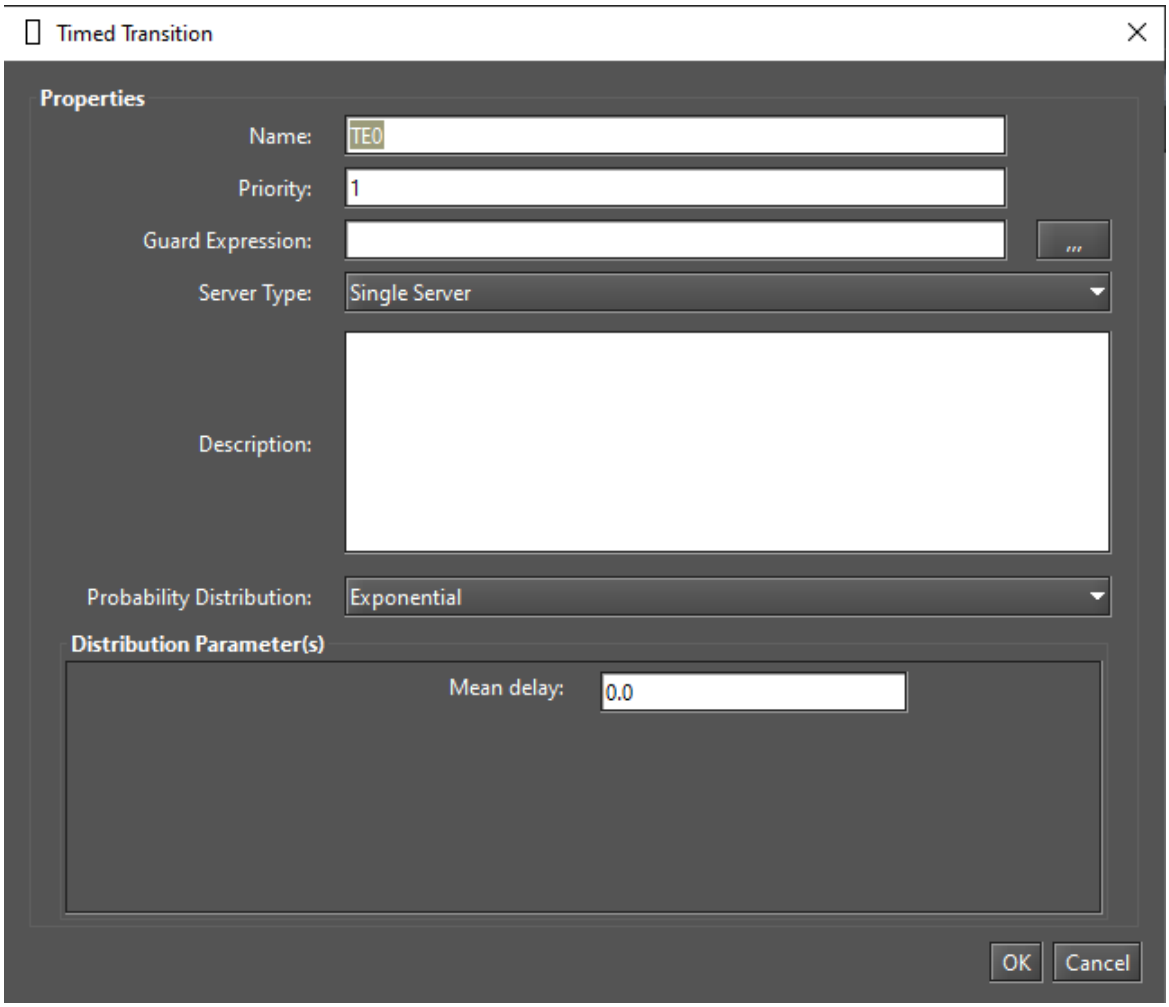


Figure 44: Properties of Timed Transitions

Next, we describe each one.

- **Name.** Name for the transition. It is used to identify the component in the model. Mercury accepts only alphanumeric characters and underscores. Also, the name must start with an alpha character. If this rule is not followed, an error occurs. Also, it is not possible to assign a name that is already used by another component of the same type.

- **Priority.** Firing priority assigned to the transition. The higher the priority, the higher the priority in firing. It is important to emphasize that immediate transitions always have priority over timed transitions.

- **Guard Expression.** A Boolean expression that allows a transition to be activated and fired. Apart from the fact that the current marking allows it, a transition is activated and can be fired only if the guard expression assigned to it evaluates to true.

  The current version of Mercury does not support floating-point literal in the guard expression. Figure 45 shows an example of how this occurs. To overcome this limitation, it is necessary to insert a definition/-variable with the floating point value. In our example, we have defined a double definition named **X**. Once created, the definition can be referenced in a guard expression, as shown in Figure 46. We are working on solving this issue and will release a new version once this issue is solved.

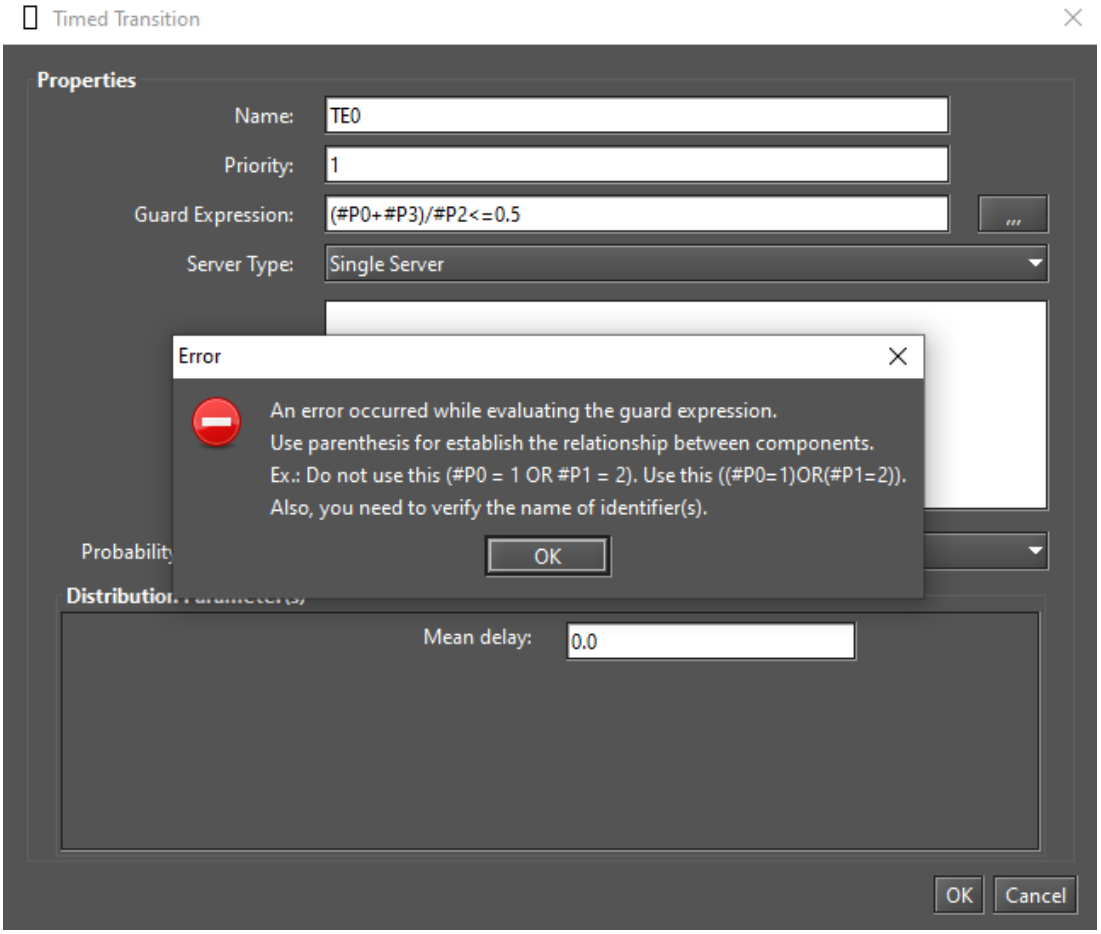


Figure 45: Error when Using Floating-Point Literal in Guard Expressions

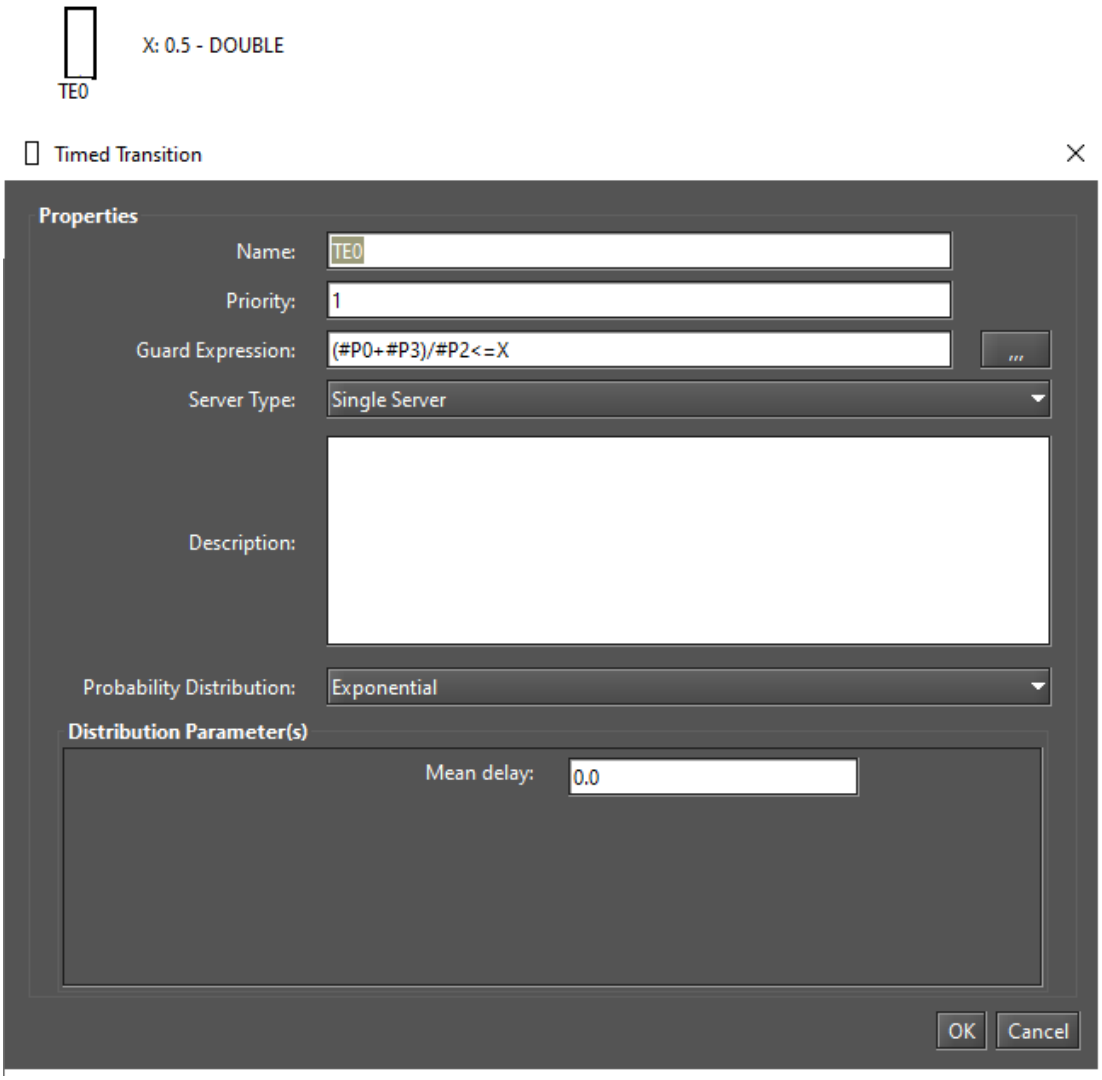


Figure 46: Solution to Refer Floating Values in Guard Expressions

- **Server Type.** The firing semantic assigned to the timed transition. The user can choose one of the two available options. 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.** Each component of the model can be assigned a description. It contains additional information about the component or about the real component/subsystem/action represented by the component. The description aims to improve the understanding about the model being created. It has no semantic value in evaluating the model. It is just plain text attached to a component.

- **Probability Distribution.** Mercury supports a large number of probability distributions. If all timed transitions are exponential, the model can be evaluated by numerical analysis or simulation. On the other hand, if there is at least one non-exponential timed transition, the model can only be evaluated by

simulations. Depending on the selected distribution, fields appear for the parameters of this distribution, in which the user can enter the values. Only non-negative real values may be entered for 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 one parameter called ”Delay” which corresponds to the delay in triggering the transition. In addition to the delay, other parameters may be required depending on the distribution chosen. For example, the exponential distribution requires only the mean delay. The Erlang distribution, on the other hand, requires two parameters: mean delay and shape. The normal distribution requires two parameters: mean and standard deviation. Each probability distribution has its own parameters that must be entered by the user before performing any evaluations.

Now, we describe the properties of immediate transitions (see Figure 47).

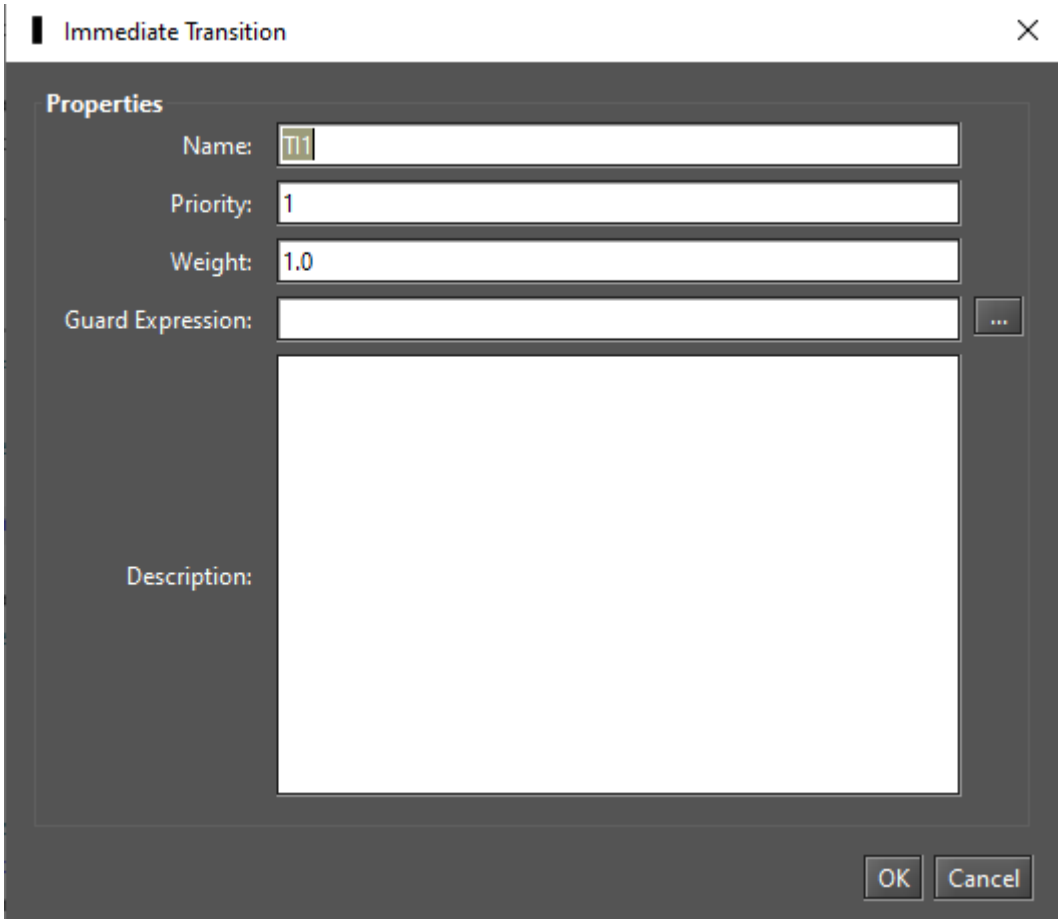


Figure 47: Properties of Immediate Transitions

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 48).

- **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 append an integer definition to the marking property once the definition has been created. To do this, the user only needs to enter the name of the definition in this field.
- **Description:** See page 26.

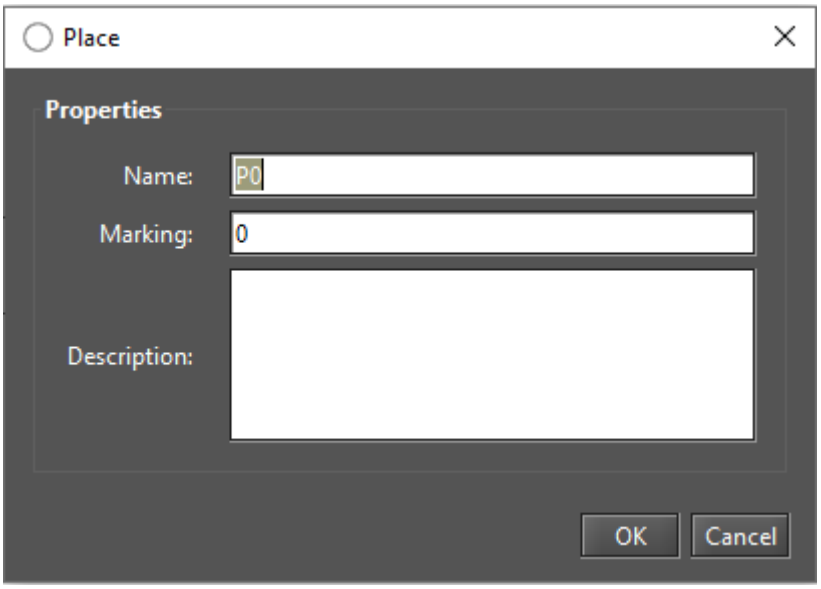


Figure 48: Properties of Places

Figure 49 shows the pop-up menu of arcs.

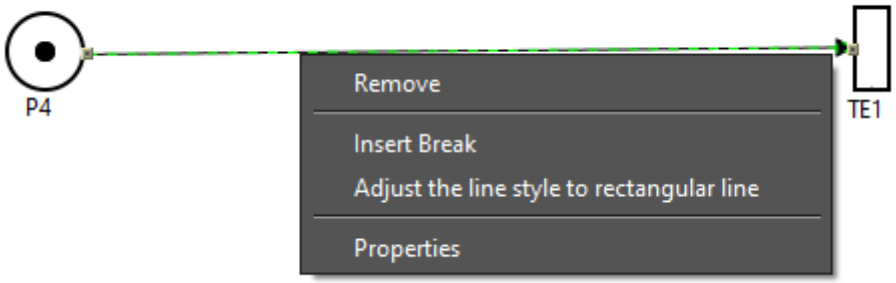


Figure 49: Menu of Arcs

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 at the clicked location. If you click on the break point and keep the mouse button pressed, you can move this point to the desired position. This way you can change the shape of the arc.

- **Adjust the line style to rectangular/curved line.** Mercury supports two line styles for arcs: rectangular and curved. The curved line style is the default style. To switch to the rectangular style, simply click the appropriate menu item for the selected arc. Figure 50 shows an SPN with a rectangular arc and Figure 51 shows an SPN with a curved arc.

Figure 50: SPN with a Rectangular Arc

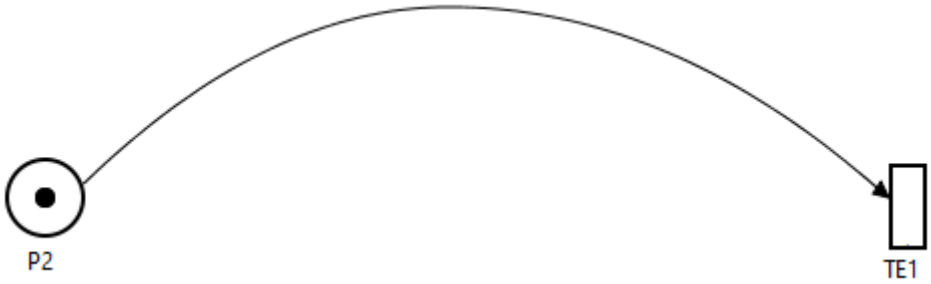


Figure 51: SPN with a Curved Arc

Figure 52 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 if the arc is an output arc of a place, or it represents the number of tokens generated in a place if the arc is an input arc to that place.

In some dialog boxes we see a button with an ellipsis as its description. This button is always next to a text box, as highlighted in Figure 53.

Clicking this button opens the Expression Editor (see Figure 54).

The expression editor is a text editor that allows you to easily create expressions to define guard expressions and metrics. It is a simple editor that highlights parentheses, brackets, and braces and some keywords. The editor has a button to reduce the font of the text and another one to enlarge it. This makes it easy to define even large and complex expressions.

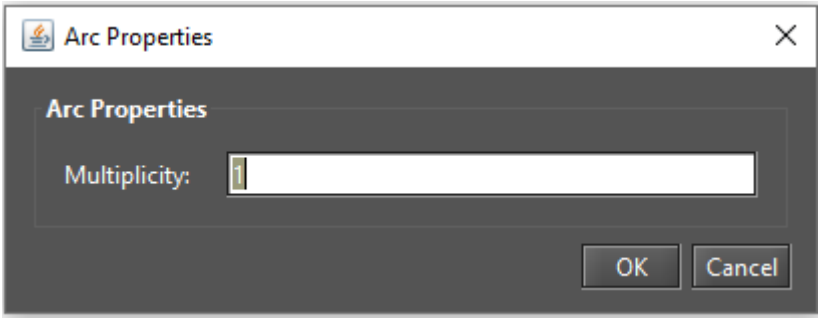


Figure 52: Properties of Arcs

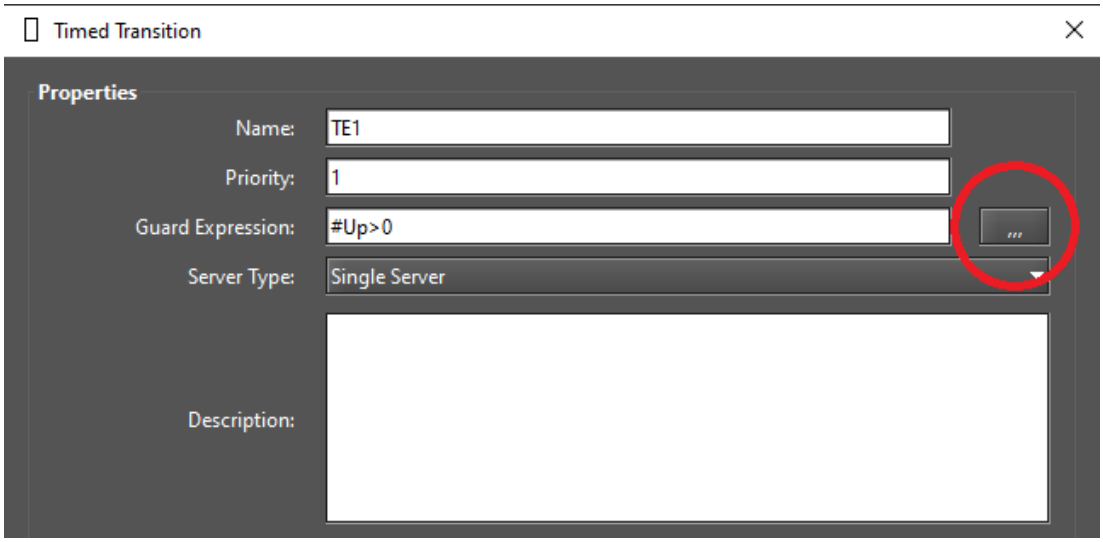


Figure 53: Accessing the Expression Editor

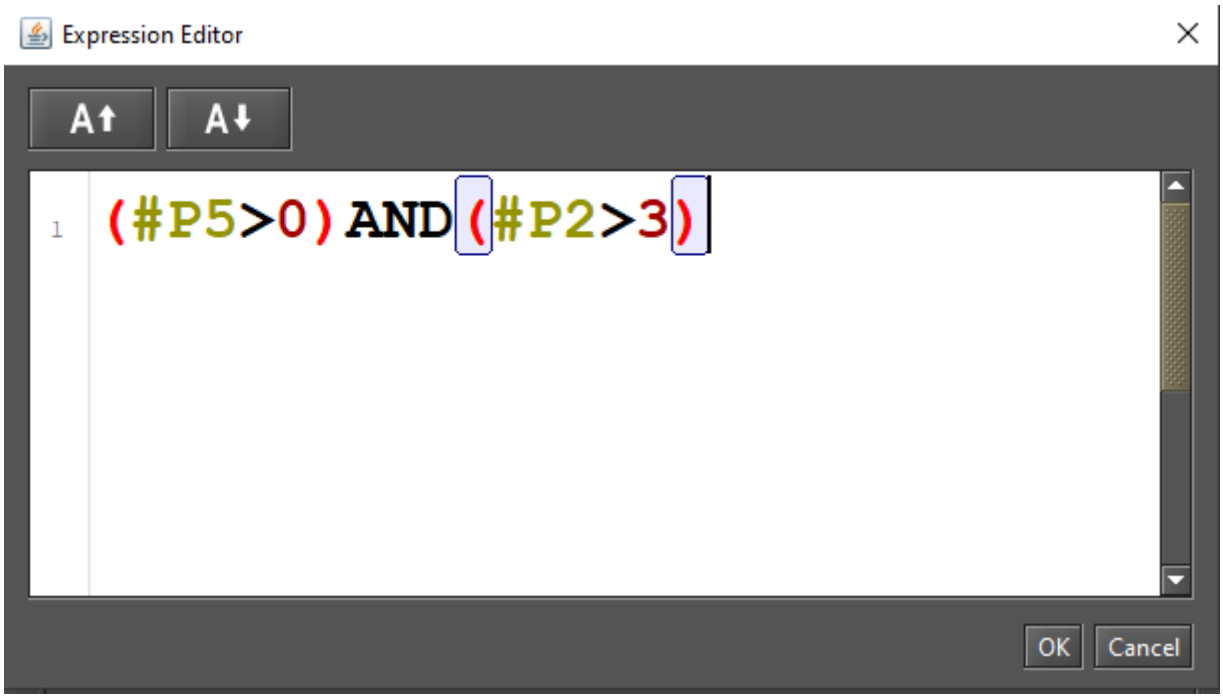


Figure 54: Expression Editor

Now let us take a look at the properties of the ”Definition” component. We have already described this component on page 21. When you access the properties of a definition, the dialog box shown in Figure 55 is displayed. Below we describe each property of this component.

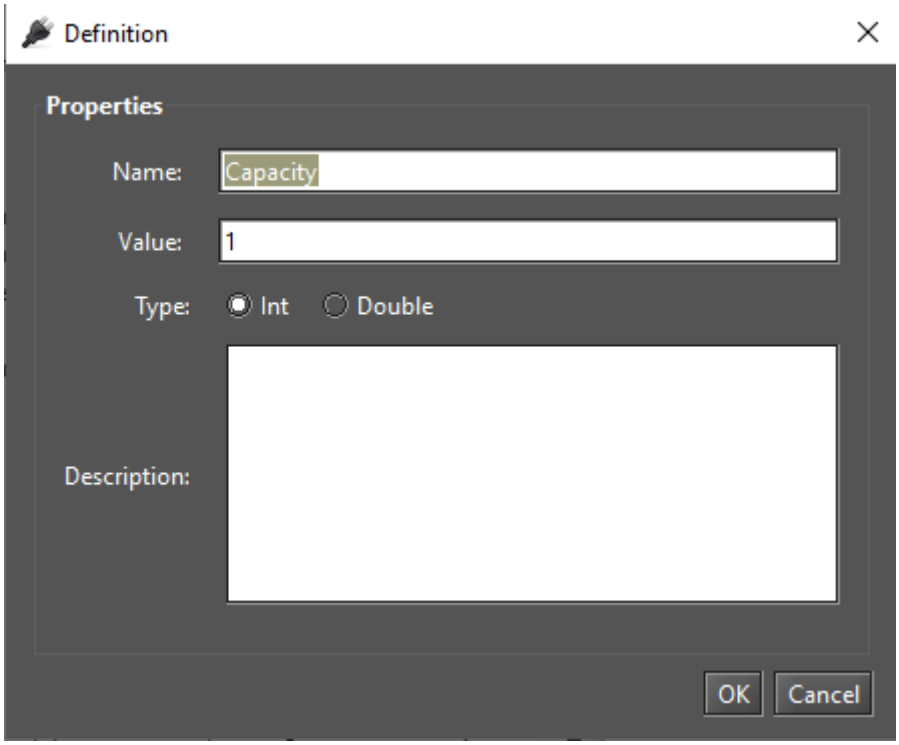


Figure 55: Properties of Definitions

- **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. When you set the properties of a definition, you must select the appropriate type for the value entered. If you select the INT type and enter a Double value, an error occurs.
- **Description.** See page 26.

An important point to note is that definitions are variables that store numeric values, as we mentioned earlier in this section. A definition cannot reference another definition, but it can be referenced by other components of other types. When you update the properties of a definition, a confirmation dialog appears if it is referenced by other components. If the definition is referenced and the values entered in the fields are valid, the corresponding properties will be updated accordingly. Figure 56 shows what happens when the user tries to update a referenced definition.

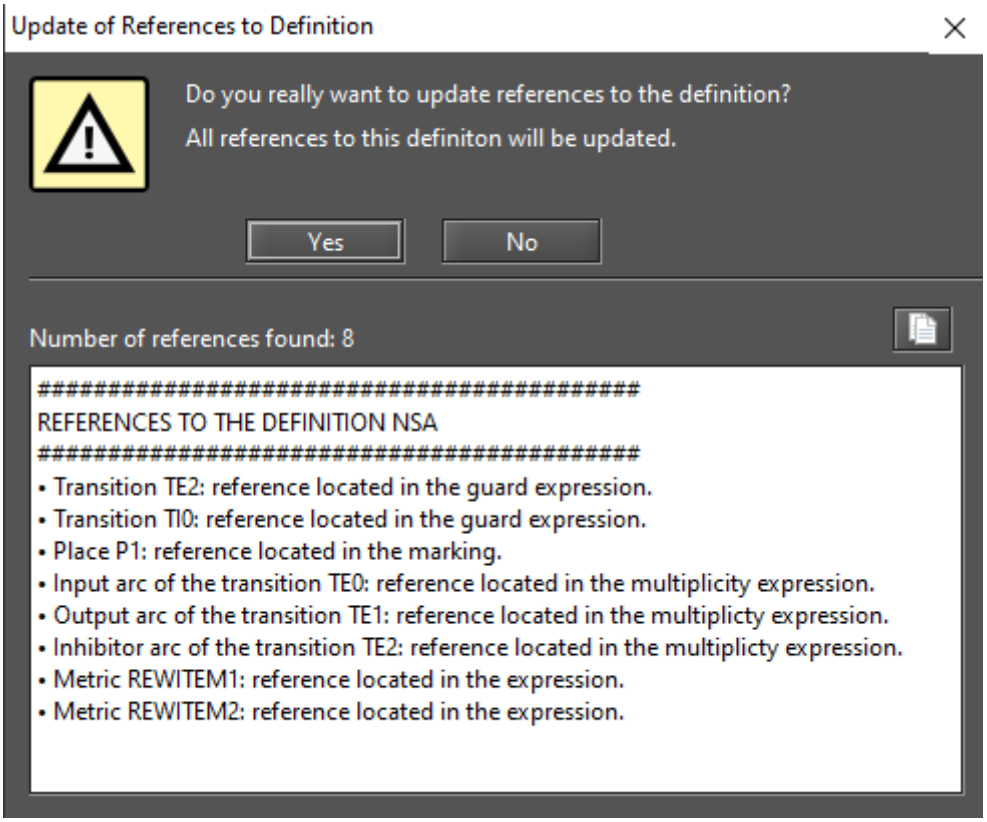


Figure 56: Confirmation to Update the References to a Definition

As we can see, all found references to the definition are displayed. The user must confirm to update the definition and all properties of other components that reference it. If you cancel the operation, the definition will not be updated. This behavior is intended to prevent the model from becoming corrupted. If you reference a definition that does not exist or store an invalid value for the property, no evaluations can be performed. Figure 57 shows the update log that is displayed after the update operation finishes successfully.

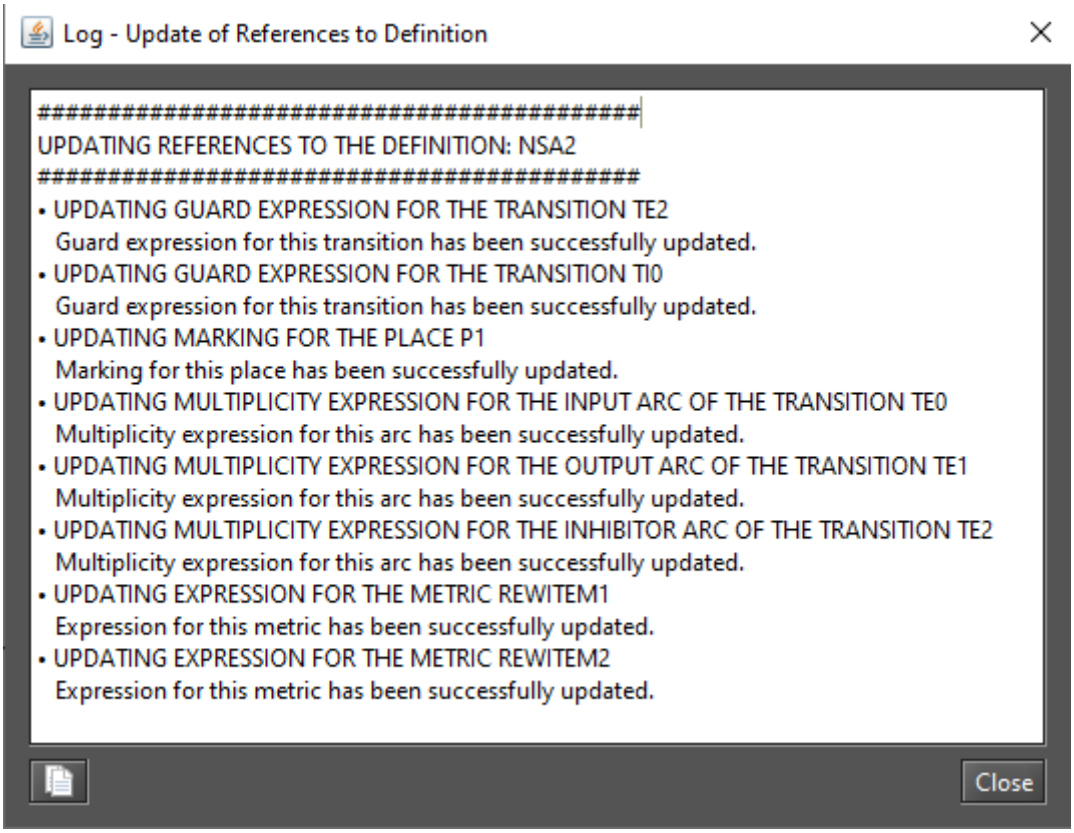


Figure 57: Updating References to a Definition

In this dialog the user can see if all references have been updated successfully. If errors occur during this process, they are displayed in the log. An example of an error occurring is when a definition of INT has been defined with a positive integer value and is referenced by the marking property of a place. If you change the type of the definition to DOUBLE or enter a negative value, the new value of the definition will not be included in the marking property. As is known, it is not possible to define a negative integer value as a marking, so the new value will not be accepted. In such cases, the properties that are referenced are set to their default values, as we can see in Figure 58.

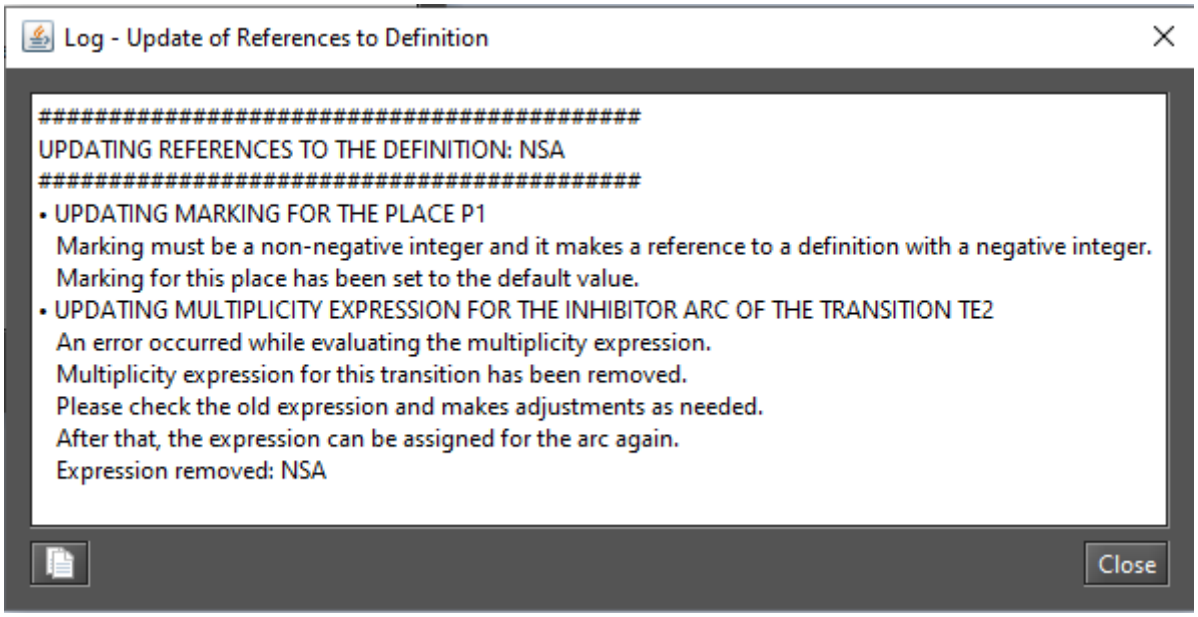


Figure 58: References Updating with Errors

When removing a definition, the same confirmation dialog is displayed as we can see in Figure 59. When the user confirms the removal, all properties referencing the definition are set to their default values (see Figure 60).

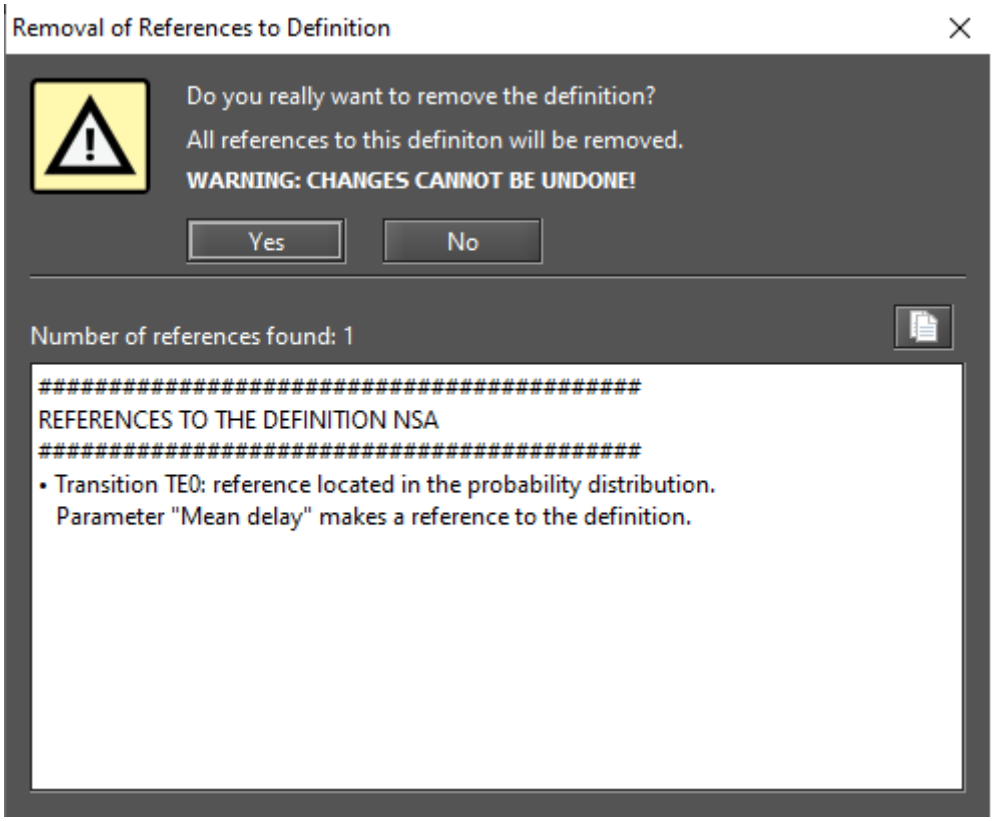


Figure 59: Confirmation to Remove a Definition

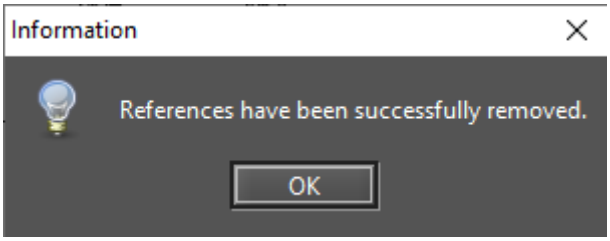


Figure 60: Invalid References are Removed

Now let us take a look at the properties of the component ”Metric”. We have already described this component on page 21. When you access the properties of a metric, the dialog box shown in Figure 61 opens. Below we describe each property of the metric.

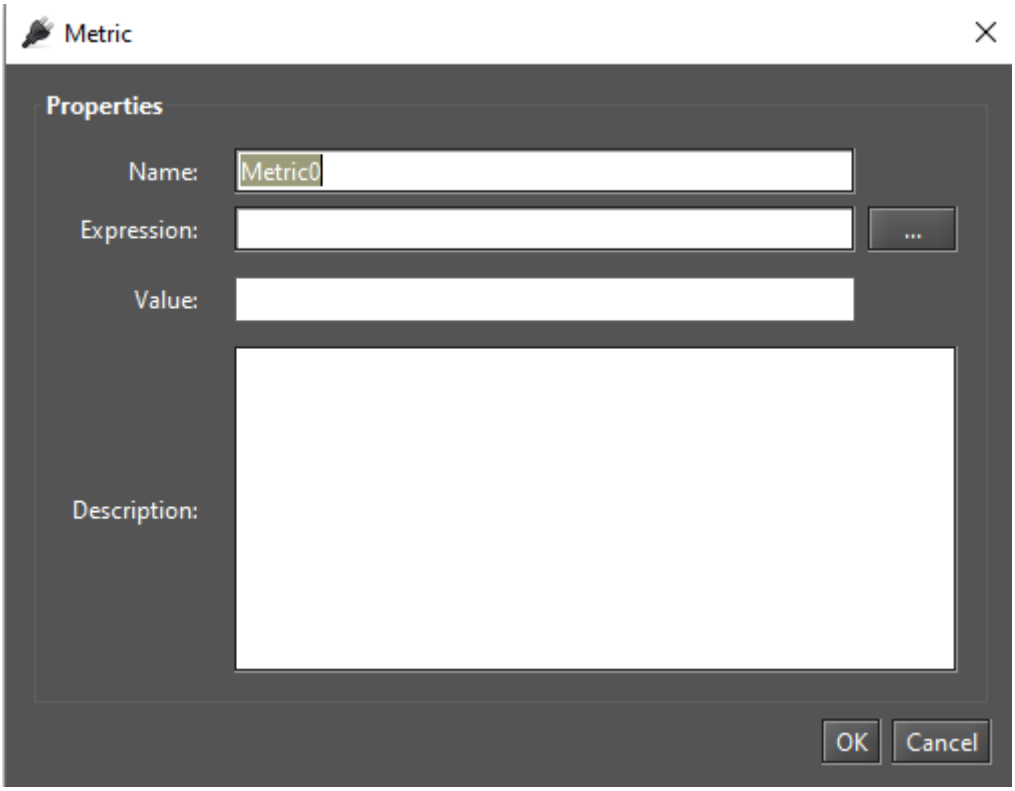


Figure 61: Properties of Metrics

- **Name.** Name for the metric. It is used to identify the metric in the model.
- **Expression.** Expression evaluated by performing analyzes or simulations. The expression can be used, for example, to obtain the state of the model at a given time or the time to perform an activity. In the appendices you can find the syntax for creating 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 enhances both the usability of the tool and the readability of the models. Once an SPN component is inserted, you can read its properties on the drawing area by placing the mouse pointer over

it. A tooltip will then appear showing all the properties of the component. As we can see in Figure 62, all the properties of a transition appear in the tooltip. Mercury provides this feature for all components of all supported formalisms.

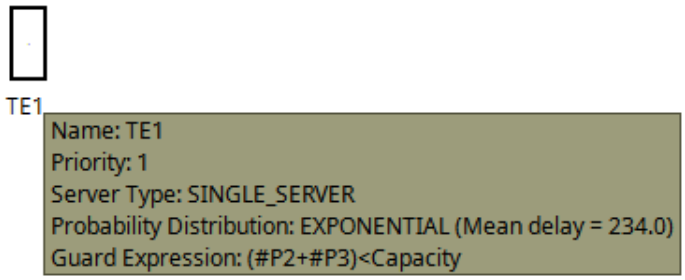


Figure 62: Tooltip for a Transition

Mercury provides for evaluations of SPN models, analyzes and simulators. Steady-state and transient metrics can be evaluated for both. These features can be found in the ”Evaluate” menu under the ”SPN Evaluation” option. They can also be accessed by clicking on the command buttons on the main toolbar (see Figure 63). Next, we will introduce the simulators and then the analyzes.

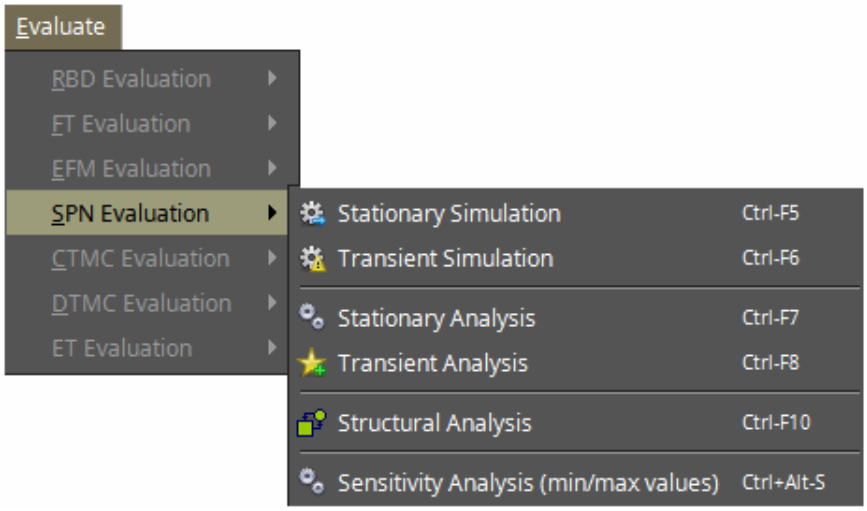


Figure 63: SPN Menu

2.1 SPN Simulation

Models with non-exponential transitions can only be evaluated by simulations. Mercury provides two types of simulators. The stationary simulator provides steady-state metrics and the transient simulator provides time-dependent metrics. We will introduce the stationary simulator below and describe the transient simulator in the next section.

2.1.1 Stationary Simulation

Figure 64 shows the input parameters for the stationary simulator. These parameters are detailed below.

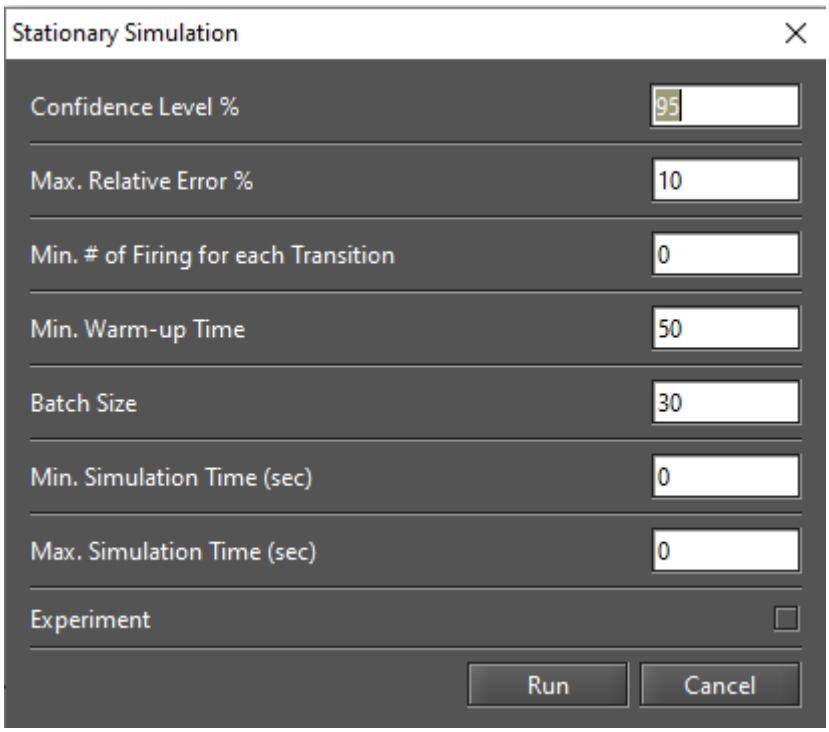


Figure 64: Stationary Simulator

- **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 to stop the simulation. If you enter a value greater than 0, the simulation will not stop until 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 has been reached, if defined. If you enter the value “0” for this input parameter, the simulator will 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 in steady state and no metrics are collected during this period. There are some methods to support evaluation of when the model enters steady state, but Mercury requires the user to define the

period of the transient phase. We plan to implement some estimation methods in future versions to detect the end of the transition phase. Since we are evaluating stochastic models, it is expected that the warm-up period is not a deterministic value when a series of simulations are performed. Therefore, the user defines a minimum warm-up time. Once the global simulation time is equal to or greater than the user-defined warm-up time, the simulation starts generating batches, collecting metrics, and calculating statistics.

- **Batch Size.** Defines the number of samples that will compose each batch in the simulation.
- **Min. Simulation Time (sec).** This time corresponds to physical time and must be expressed in seconds. This time can help us perform simulations in cases where the model may have 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 is no variation in the simulation error. Therefore, the simulator cannot proceed with the simulation by estimating the required number of batches to achieve the desired relative error. Entering a minimum simulation time prevents the simulator from stopping the simulation if the initial number of batches has no variation in the error. If you enter a value greater than 0, the simulation will stop if the global time is greater than this time and the simulation error is less than or equal to the relative error, or if other stop conditions are met. If there is no change in error until the minimum time is reached, the simulation will stop. If you set the value 0 for this parameter, this stop criterion will not be 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 specified 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 will stop when this time is reached. If you assign the value 0 to this parameter, this stopping criterion will not be taken into account.
- **Experiment.** Experiment allows us to run a series of simulations by changing the value of a particular parameter in each simulation. The change of parameters can be linear or logarithmic. The value of the parameter is changed considering a step size and a minimum and maximum value. At the end of an experiment, Mercury presents a graph showing the impact of each value change on the selected metric. In this graph the user can see the average value and confidence intervals for each point.

Figure 65 shows the stationary simulator in action.

The information displayed in this window is self-describing. The stationary simulator has two tabs. The "*Batches and Errors*" tab displays the logs of the processed batches and the relative error of the simulation up to that point (see Figure 66). The "*Transitions Firings*" tab shows the number of firings for each fired transition, as well as the percentage of firings relative to the other fired transitions (see Figure 77).

The simulation finishes when one of the following is reached: maximum relative error, minimum 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, whichever comes first. At the top of the window there is

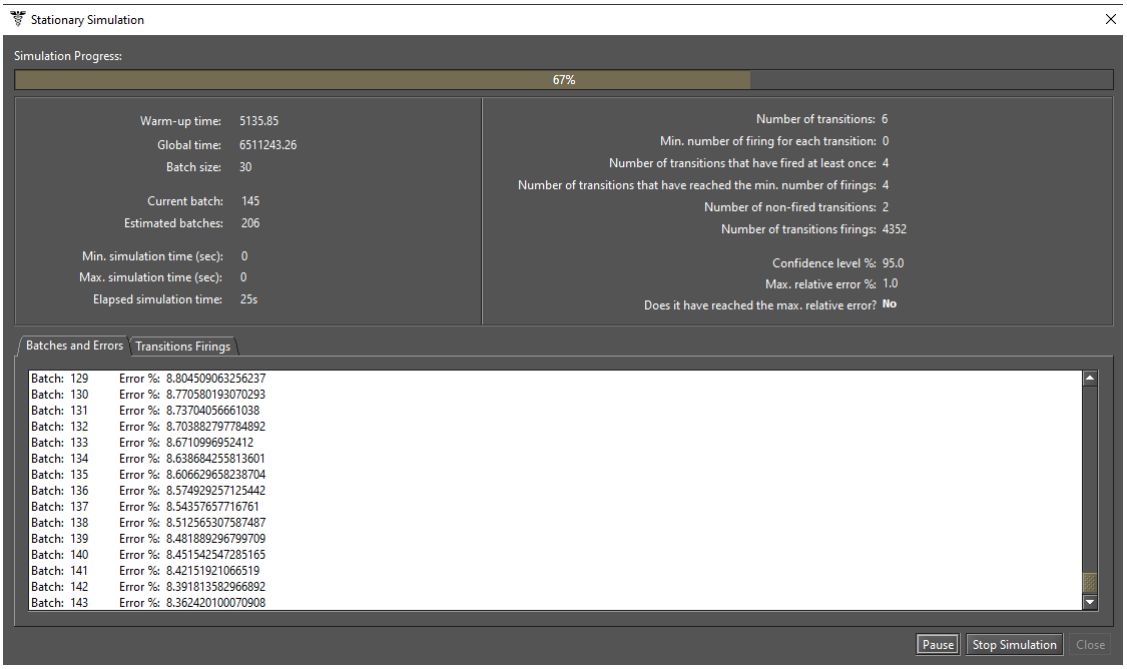


Figure 65: Stationary Simulator

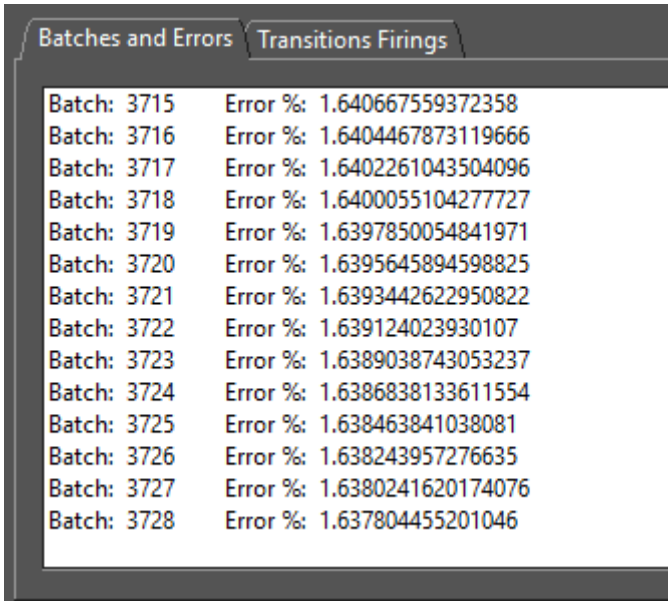


Figure 66: Batches and Errors

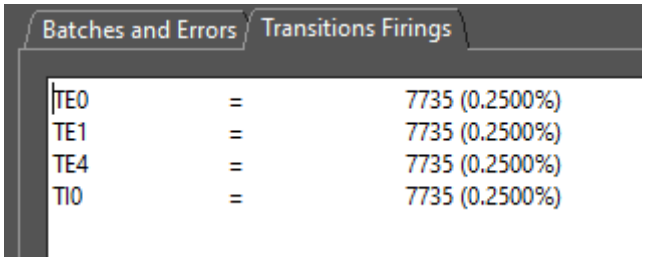


Figure 67: Transitions Firings

a progress bar that shows the progress of the simulation. This simulation progress is subject to adjustments depending on the simulation, as the previously estimated number of batches to reach the relative error can be

re-estimated, changing the overall progress of the simulation. In addition, the user can pause, resume, and stop the simulation at any time (see Figure 78).

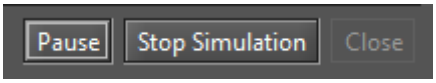


Figure 68: Buttons on the Stationary Simulator

When a simulation is complete, the user can export the result as plain text or as a spreadsheet (MS Excel) (see Figure 69). Considering the result of a simulation, a variety of statistics are calculated.

Figure 69: 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 displayed on the “Result” tab of the main window. Figure 70 shows the example model we used in the simulator. Listing 2 shows an example of the output generated by the simulator. In this example, only one metric was evaluated.

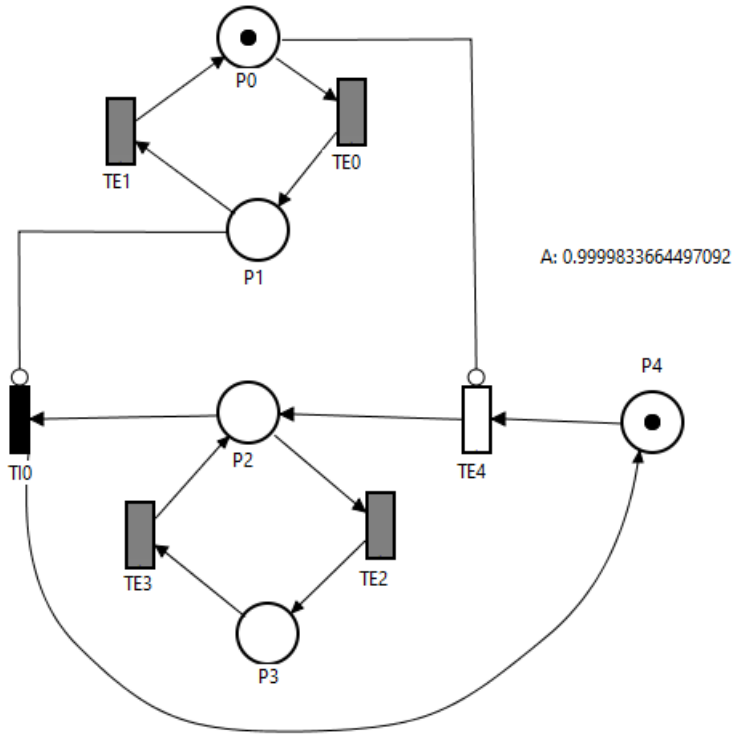


Figure 70: SPN Model for Stationary Simulation

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 show you how to perform experiments in the stationary simulator. Figure 71 shows the dialog box that appears when you confirm the simulation input parameters and enable the “Experiment” option.

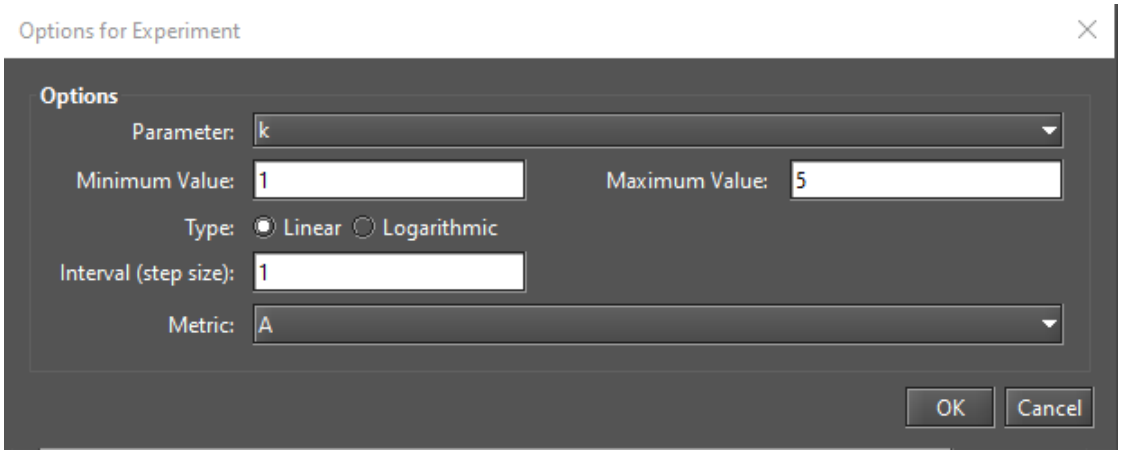


Figure 71: Executing an experiment in the Stationary Simulation

In this window, the user must select the parameter to be changed, its minimum and maximum values, whether the value should be changed linearly or logarithmically, and the step size if the value is changed linearly. If the change is logarithmic, it is considered as a base-10 logarithmic function. Also, the user must select the metric to be evaluated. At the end of the simulation, a graph is created that takes into account the value of the metric for each change in the parameter value. As can be seen in Figure 72, the mean value and its confidence

interval are plotted for each point. When you move the mouse pointer over the point representing the mean, the values of the confidence intervals are displayed as a tooltip. Figure 73 shows the result of a stationary simulation experiment with a base-10 logarithmic variable.

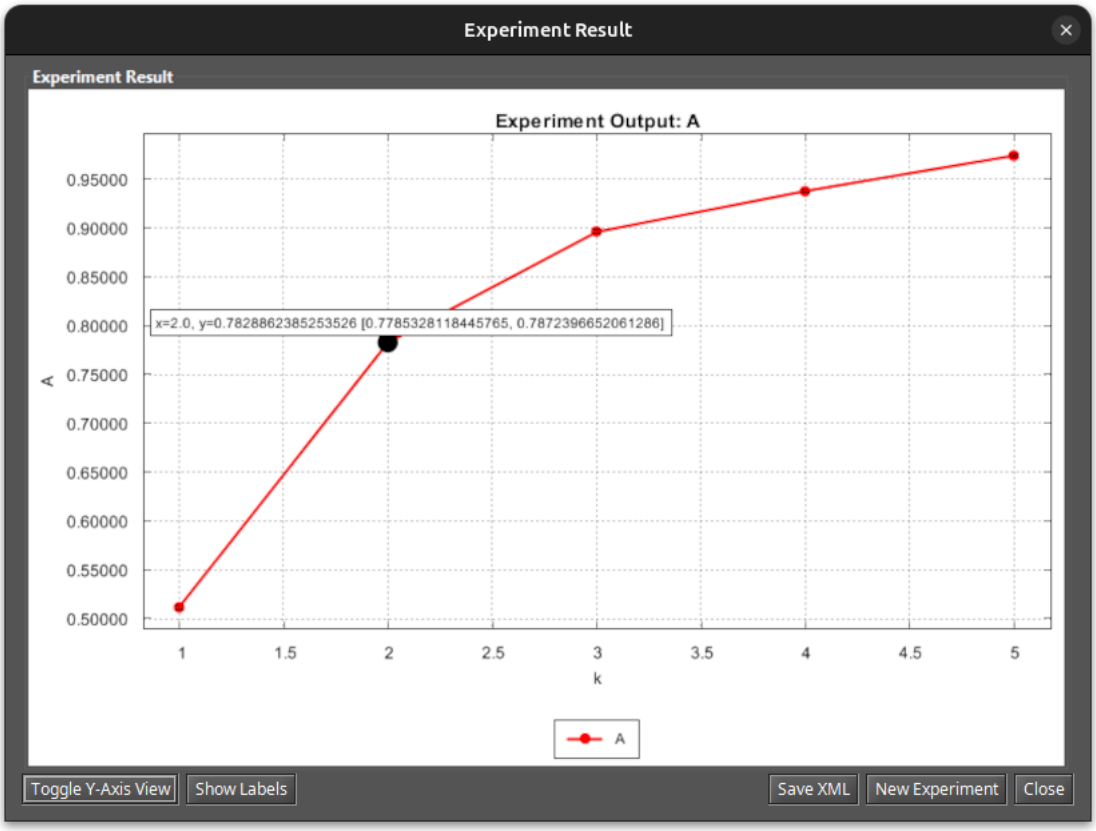


Figure 72: Output of a Stationary Simulation Experiment for an SPN Model - Variable k Linearly Changed

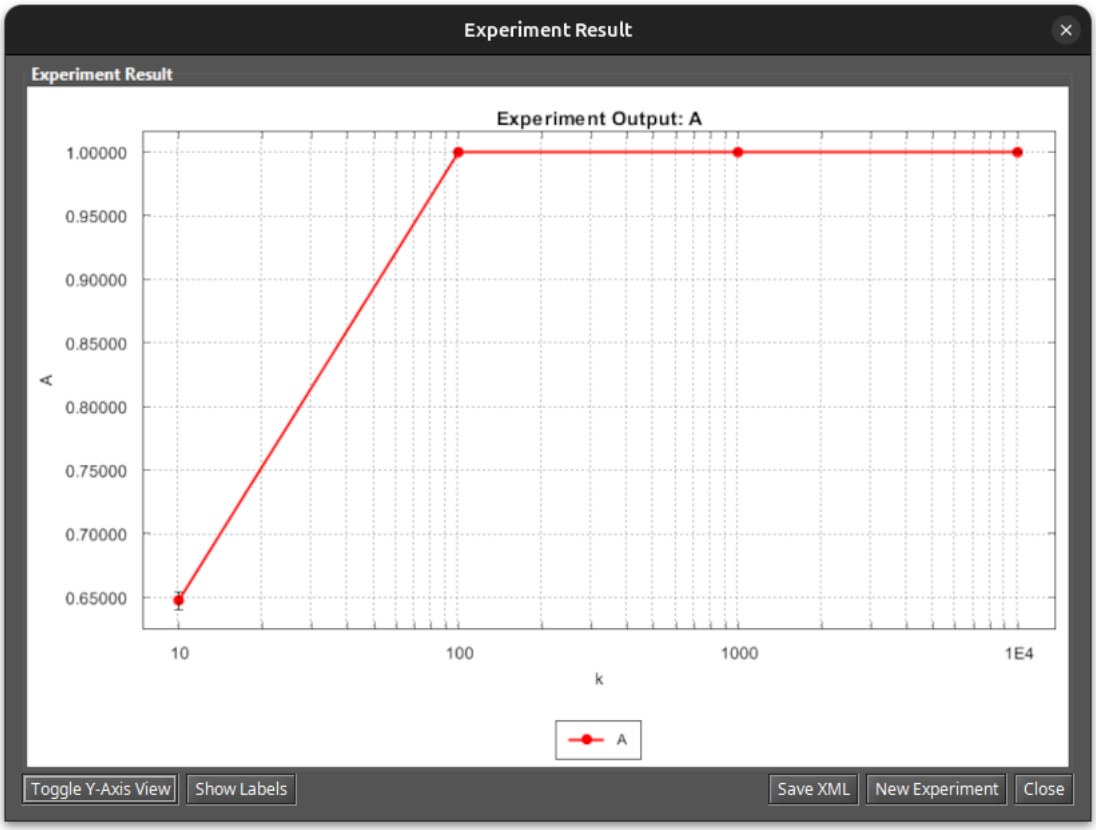


Figure 73: Output of a Stationary Simulation Experiment for an SPN Model - Logarithmic Variable

2.1.2 Transient Simulation

Models with non-exponential transitions can only be evaluated by simulations. Transient simulations can be used when the user is interested in evaluating metrics at a particular point in time. A transient simulation consists of a series of replications and each replication consists of a series of runs. Each run runs from time 0 until time t', specified by the user in the "time" parameter, is reached. When the current set of runs is finished, the value of each sampling point of the current replication is calculated. The replication represents the mean values of the points in its set of runs.

The transient simulator can be accessed from the menu *Evaluate -> SPN Evaluation -> Transient Simulation*. Figure 74 shows the input parameters for the transient simulator. Each parameter is described below.

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

  "DES + Linear Regression 1" calculates the value of each point at the end of each run by linear interpolation between two known points. When the number of runs of a replication has been performed, the values of each point in the current set of runs are collected and its average value is assigned to the same point in the current replication.

  "DES + Linear Regression 2" calculates the value of each sampling point of the current replication when the set of runs has been performed. Unlike the first method, this method calculates the value of each point

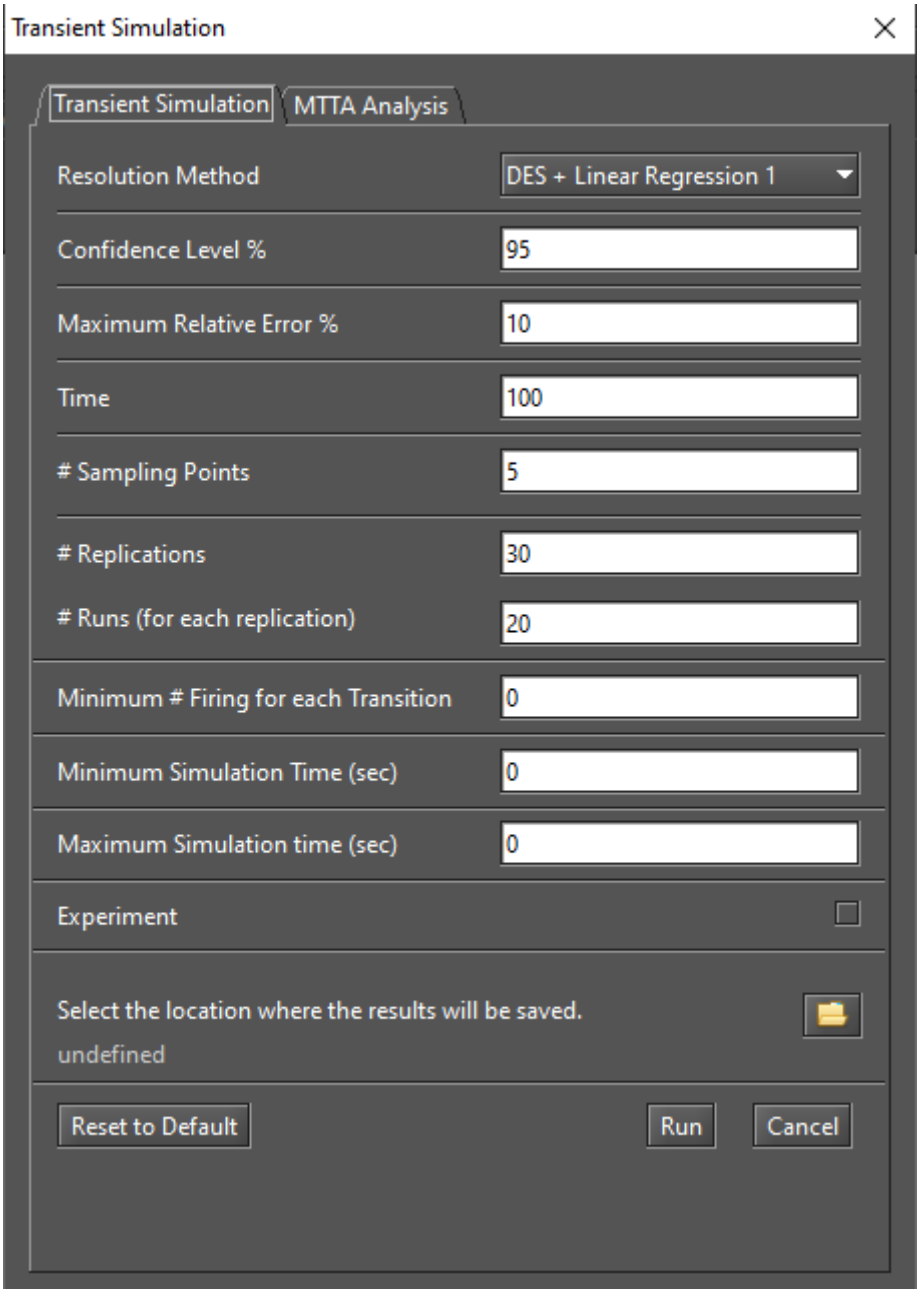


Figure 74: Input Parameters for the Transient Simulator

of the current replication considering its entire set of runs. This method applies linear regression between several known points. For each sampling point, this method considers two sets of events. The first one comprises the set of the last events that occurred before the evaluated point. The second set consists of the first events that occurred at the evaluated point or after it. In each run, the events that occurred before the evaluated point and the events that occurred at the evaluated point or after are collected. For each set of points, the mean value of the metric and the mean time of occurrence of the events are calculated. Then the value of the metric for the current sampling point is estimated.

- **Confidence Level.** The confidence interval for obtaining the metrics.
- **Max. Relative Error %.** Defines the maximum relative error in order to stop the simulation.

- **Time.** Sets the evaluation time (t'). Each run starts with time 0 until time t' is reached. This time can be divided into different intermediate points and each metric is evaluated for each point. The intermediate points are defined by the number of sampling points.

- **Sampling Points.** Specifies the number of sampling points that will be evaluated during the simulation. The time interval from time 0 to time t' is divided into intermediate points considering this number of sampling points. If the user chooses to evaluate only one sampling point, only the value of the metric at time t' will be considered.

- **Replications.** Sets the initial number of replications of the simulation. If the initial number of replications is reached and the simulation error has not yet been reached, then the simulator estimates a new number of replications to reach the desired error, taking into account the current state of the simulation. The simulator re-estimates the number of expected replications to reach the simulation error whenever the number from the last estimation is reached and the error is not yet reached.

- **Runs.** Specifies the number of runs for each replication. As we mentioned earlier, each replication consists of a series of runs. Each run starts at time 0 until time t' is reached. When the number of runs for the current replication is reached, the values for time t' and any intermediate times are calculated and assigned to the current replication. A new replication is then initiated unless the stop criteria are met.

- **Min. firing for each Transition.** Sets the minimum number of firings for each transition. This is another condition to stop the simulation. If you enter a value greater than 0, the simulation will not stop until the number of firings for each transition is equal to or greater than the defined value and the error criterion has been reached or the maximum elapsed time has been reached, if defined. If you enter the value "0" for this input parameter, the simulator will 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 specified in seconds. This criterion can help us run simulations when the model may have rare events. Rare events occur when the difference between the delays associated with the transitions is huge. Rare events may be the reason why there is no variation in the simulation error. Therefore, the simulator cannot proceed with the simulation by estimating the necessary number of replications in order to reach the desired relative error, since the relative error has not changed since the beginning (it is 0). Entering a minimum time avoids stopping the simulation in this case. If the minimum time is reached and there has been no change in error, the simulation will stop. Otherwise, the simulation continues until the error or another stop criterion is reached. If you enter the value "0" for this input parameter, the simulator will 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 specified in seconds. If the stop criteria (minimum simulation time, maximum relative error, and number of firings for each transition) are not

met before this time is reached, the simulation will stop when this time is reached. If you set the value “0” for this input parameter, the simulator will not consider this stopping criterion.

- **Experiment.** Experiment allows us to run a series of simulations by changing the value of a particular parameter in each simulation. The change of parameters can be linear or logarithmic. The value of the parameter is changed considering a step size and a minimum and maximum value. At the end of an experiment, Mercury presents a graph showing the impact of each value change on the selected metric. In this graph, the user can see the average value and confidence intervals for each point.

- **Location.** Before you start the simulation, you must specify the location where you want to save the results.

Figure 75 shows the transient simulator in action.

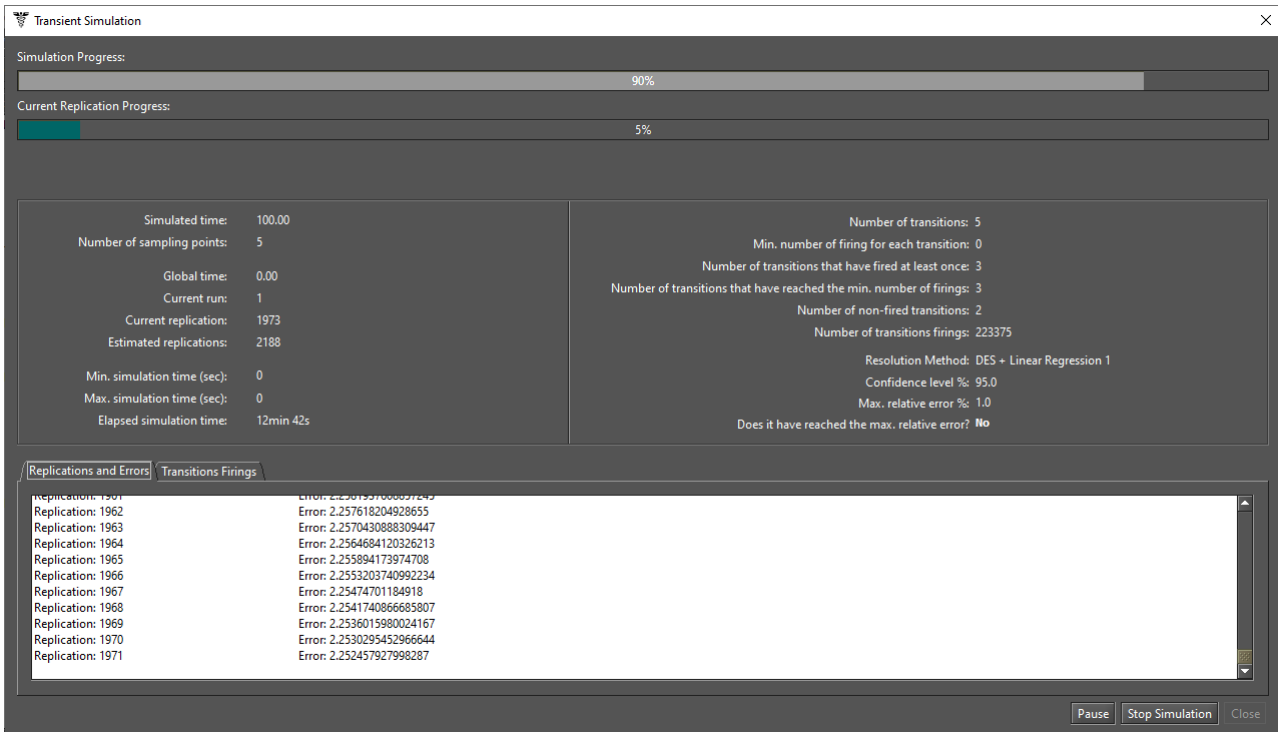


Figure 75: Transient Simulator

The information displayed in this window is self-describing. The transient simulator has two tabs. The ”*Replications and Errors*” tab displays the logs of replications processed and the relative error up to that point (see Figure 76). The ”*Transitions Firings*” tab shows the number of firings for each fired transition, as well as the percentage of firings relative to other fired transitions (see Figure 77).

The simulation finishes when one of the following is reached: maximum relative error, minimum 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, whichever comes first. At the top of the window there is a progress bar that shows the progress of the simulation. This simulation progress is subject to adjustments

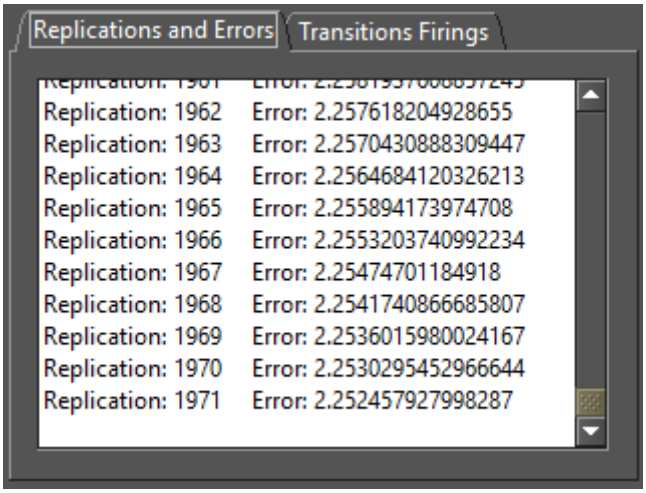


Figure 76: Replications and Errors

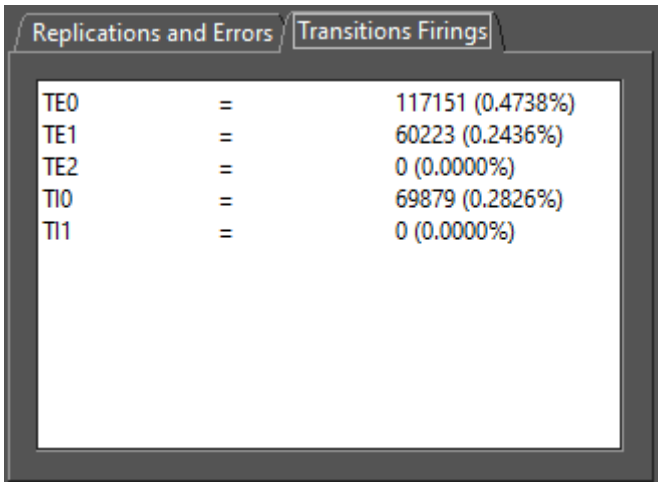


Figure 77: Transitions Firings

depending on the simulation, as the previously estimated number of replications to reach the relative error may be re-estimated, changing the overall progress of the simulation. In addition, the user can pause, resume, and stop the simulation at any time (see Figure 78).

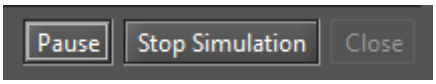


Figure 78: Buttons on the Transient Simulator

A large number of statistics is computed by considering the result of a simulation. 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 a simulation, the result is displayed on the “Result” tab of the main window. Listing 2 shows an example of output generated by the transient simulator. In this example, only one metric was 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/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 evaluates the behavior of absorbing models and generates a large number of results from them. Figure 79 shows an example of an absorbing model.

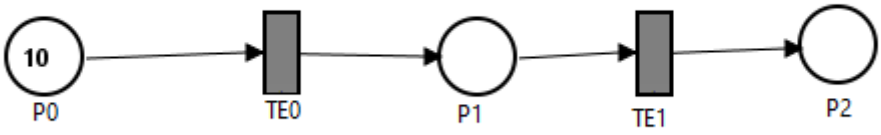


Figure 79: Absorbing SPN Model

Mean time to absorption (MTTA) simulation is accessed by following the menu depicted in Figure 80.

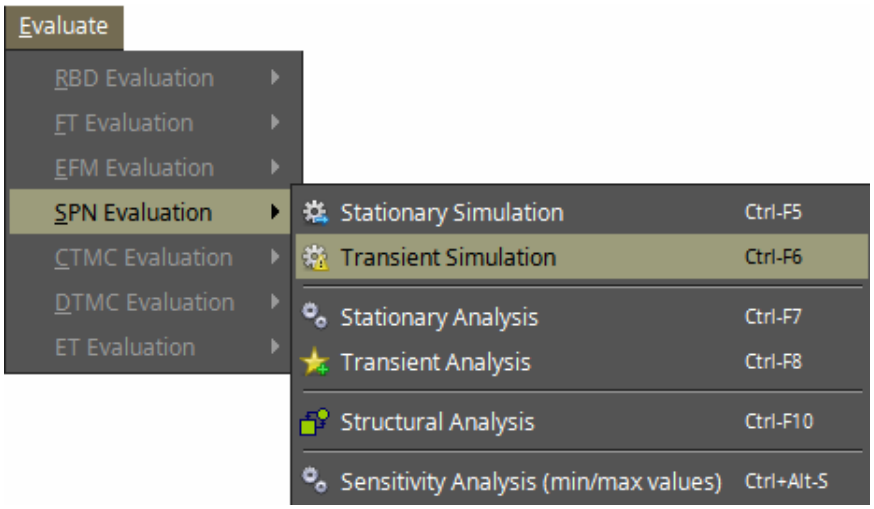


Figure 80: Accessing the Transient Simulator

By accessing this menu, a window with two tabs is displayed (see Figure 81). The first tab contains the input parameters for the default transient simulator (see previous section). The second tab ("MTTA Analysis") contains the input parameters for the MTTA simulation, which are described below:

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

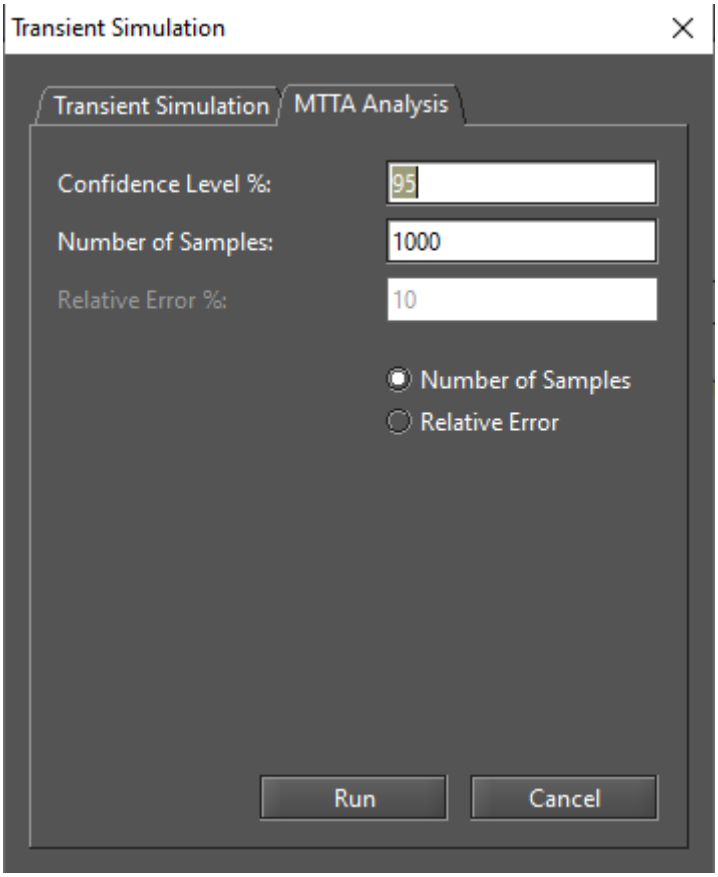


Figure 81: MTTA Analysis Dialog

At the end, a window displays the results of the transient simulation for the absorbing model being evaluated. The “Summary” tab provides statistics about the simulation. As we can see in Figure 82, a large number of statistics are calculated. Listing 3 shows the output of an MTTA simulation in detail.

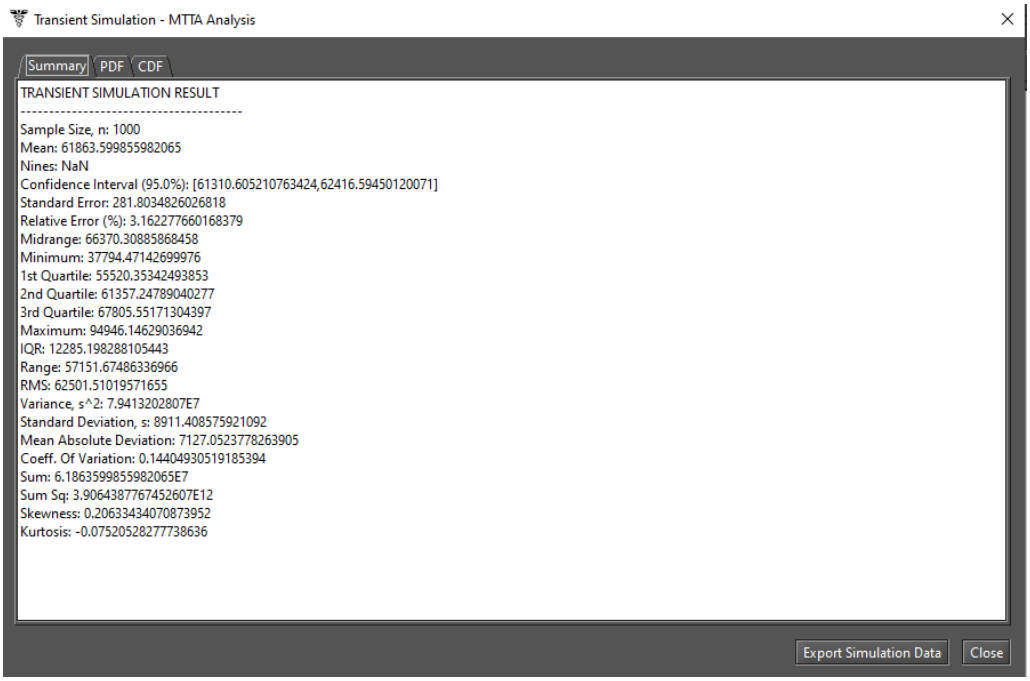


Figure 82: MTTA Result - Summary

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
```

The “PDF” tab displays the probability density function of the generated data (see Figure 83). The cumulative distribution function of this data is displayed on the “CDF” tab (see Figure 84). By placing the cursor on any blue point of the plotted curve, the tool will display the x-axis and y-axis values as a tooltip (see Figure 85). The user also has the option to export the result to a MS Excel spreadsheet (an .xls file).

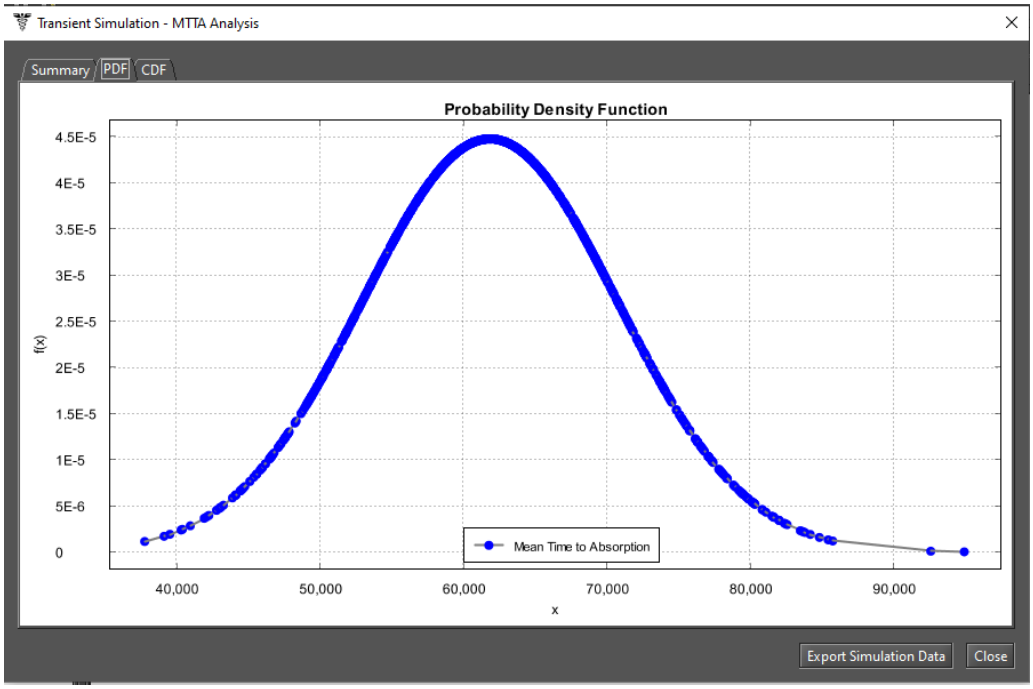


Figure 83: MTTA Result - PDF

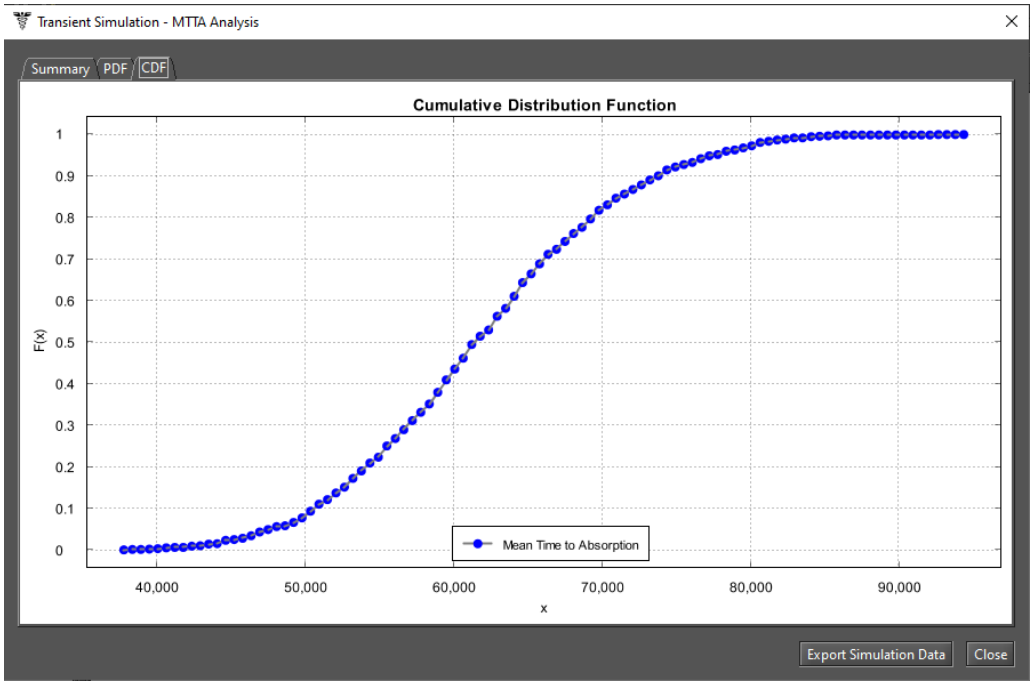


Figure 84: MTTA Result - CDF

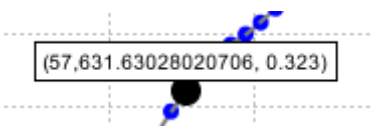


Figure 85: X and Y Axis Values

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 being evaluated. Stationary analysis computes steady-state probabilities, useful for evaluating the long-term average behavior of modeled systems. Transient analysis, on the other hand, computes time-dependent probabilities, useful for evaluating the behavior of modeled systems at a particular point in time.

As of Mercury version 5.2, the tool provides two methods for storing the CTMC states underlying the SPN models evaluated during the state space generation process for the application of analytical solutions. The first method (*"memory"*) is the traditional method, where the state space of the model is stored only at RAM. In the second method (*"disk"*), the CTMC states are stored on disk during state space generation, making it possible to generate large CTMCs on computers where the amount of RAM is limited. Both methods are available for stationary and transient analysis.

2.2.1 Stationary Analysis

Figure 86 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**.

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 ($10^{-}7$). Clicking the "Run" button will trigger the solution algorithm and once it is finished, the results will be displayed 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 CTMC...
CTMC generated... (1s)
Executing GTH numerical method...
Done! (elapsed time: 1s)
S0=0.9903691816162996
S1=0.009630818383700439
```

When solving the model by Gauss-Seidel, the user can change not only the maximum error but also the maximum number of iterations. The default value for such a parameter is "-1", which means that the algorithm will not stop until the convergence of the results is reached, taking into account the error entered in the input dialog (see Figure 87).

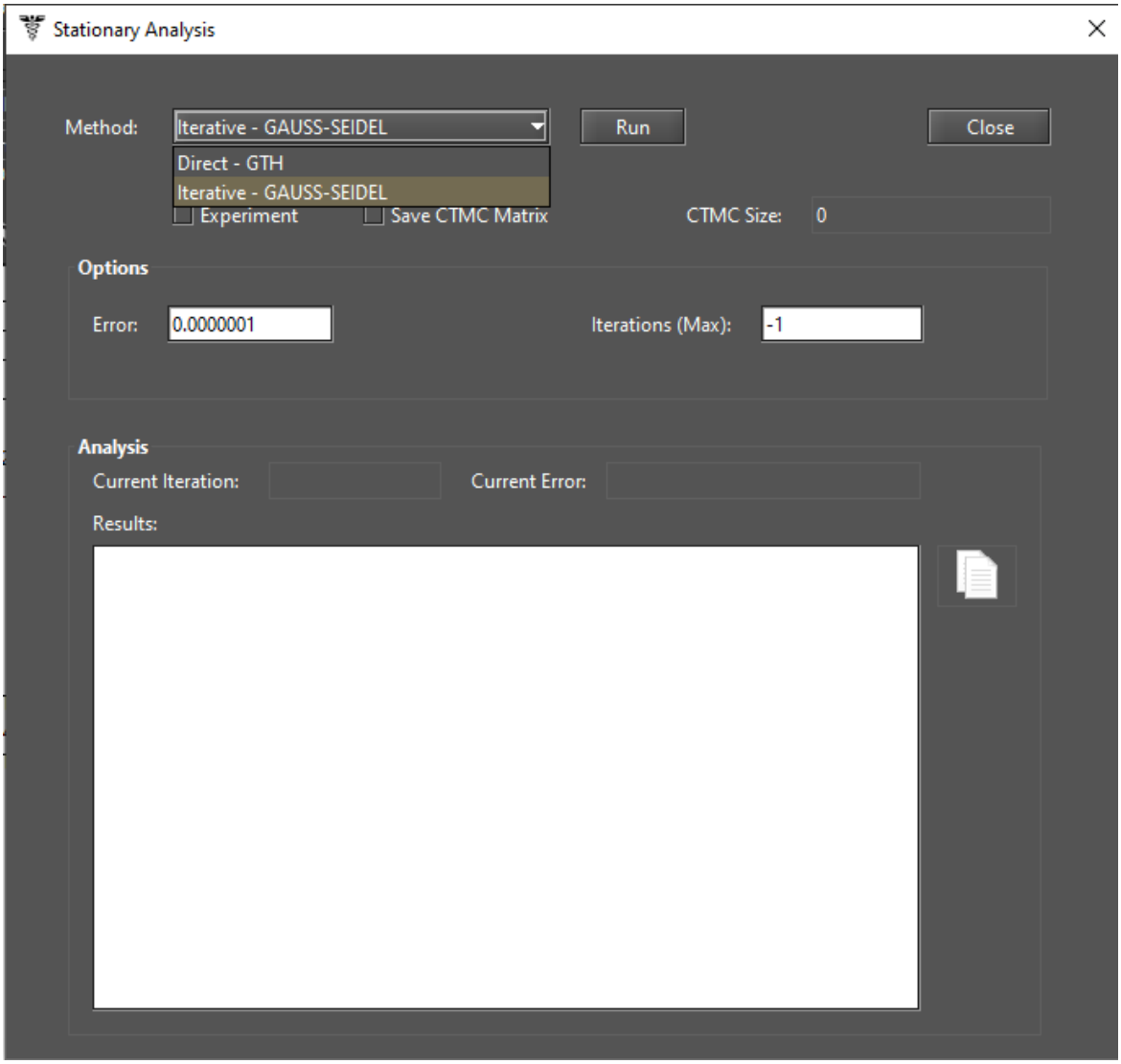


Figure 86: Stationary Analysis Window

Metrics are updated as soon as the analysis is complete, regardless of the method chosen. Their values are updated in the drawing area, as shown in Figure 88, where a metric called “Availability” has been defined.

SPN models can also be solved for a range of values of the user-defined parameters. To do this, check the “Experiment” box in the “Stationary Analysis” window and then click the “Run” button. A new dialog box will appear where the user can specify the input parameters for the experiment to be run (see Figure 89).

Below, we describe each of them.

- **Parameter.** Parameter (definition) that will have its value changed.
- **Minimum Value.** Initial value to be assigned to the selected parameter.
- **Maximum Value.** Final value to be assigned to the selected parameter.
- **Type.** Determines whether the value of the parameter is changed linearly or logarithmically. If it is logarithmic, the parameter value is changed by a base-10 logarithmic function, taking into account the minimum and maximum values.

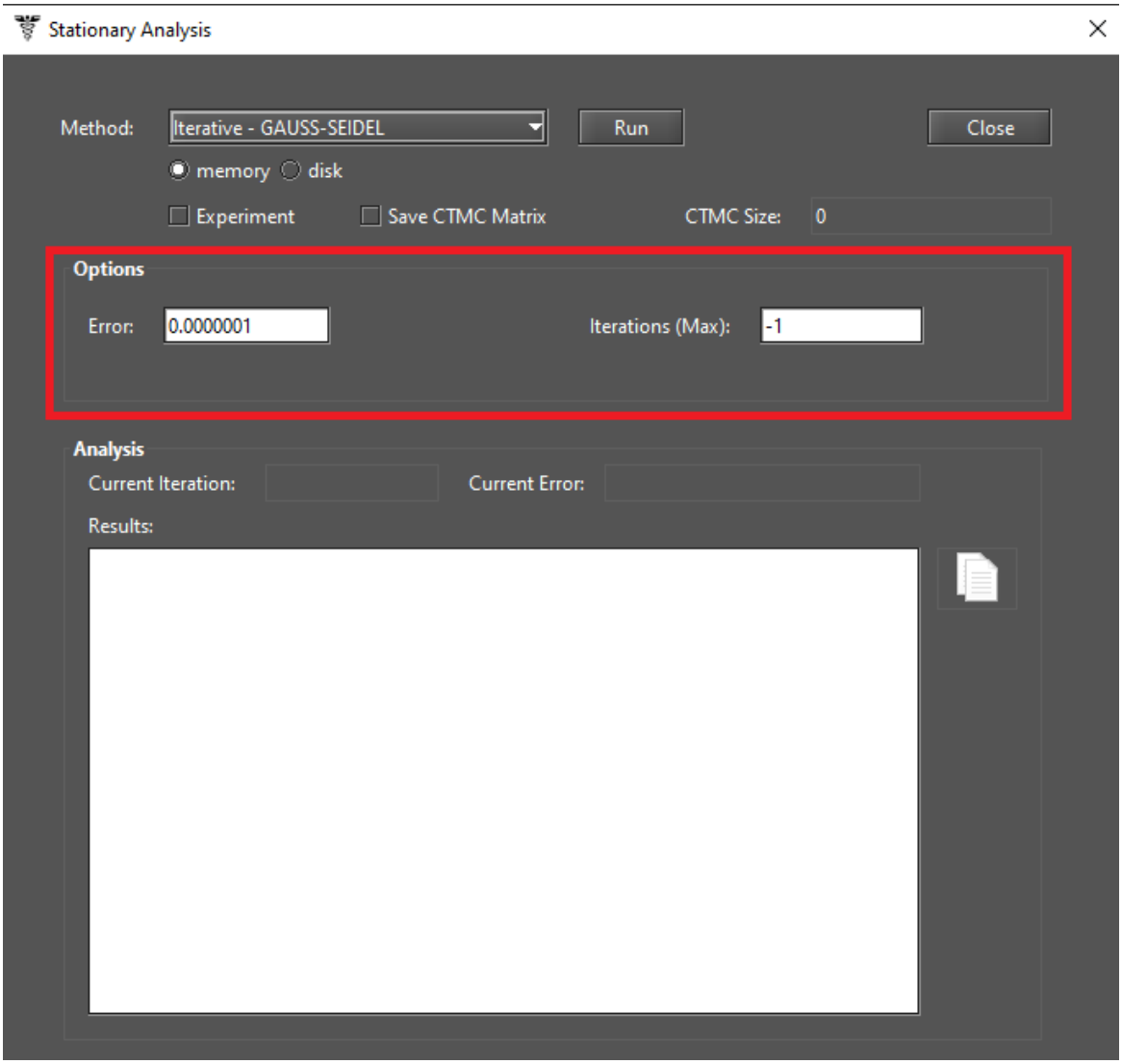


Figure 87: Stationary Analysis Window - Gauss-Seidel Method

- **Interval (step size).** This is the step size for changing the value of the parameter. The parameter starts with the minimum value and its value is increased considering the step size. At each change, the selected metric is evaluated. The experiment ends when the maximum value for the parameter is reached.
- **Metric.** Metric to be evaluated.

At the end of the experiment, the results are displayed and a graph is plotted, as we can see in Figures 90 and 91.

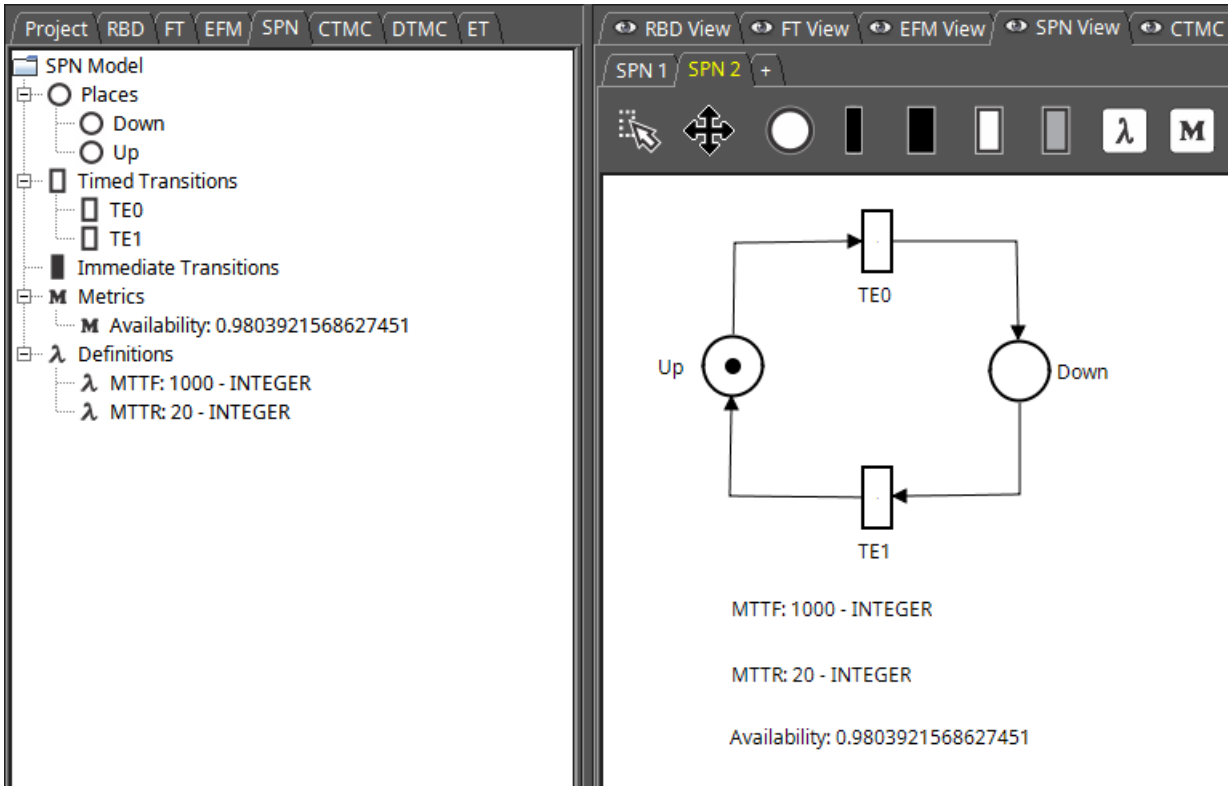


Figure 88: An SPN Model

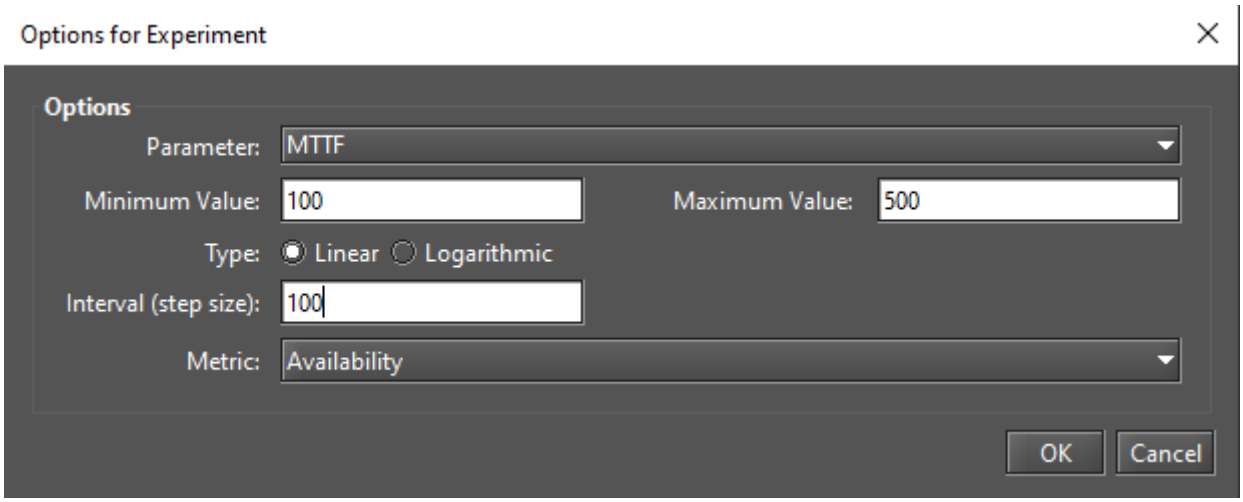


Figure 89: An SPN Experiment

2.2.2 Transient Analysis

Figure 92 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)**.

When solving a model, the user can define:

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

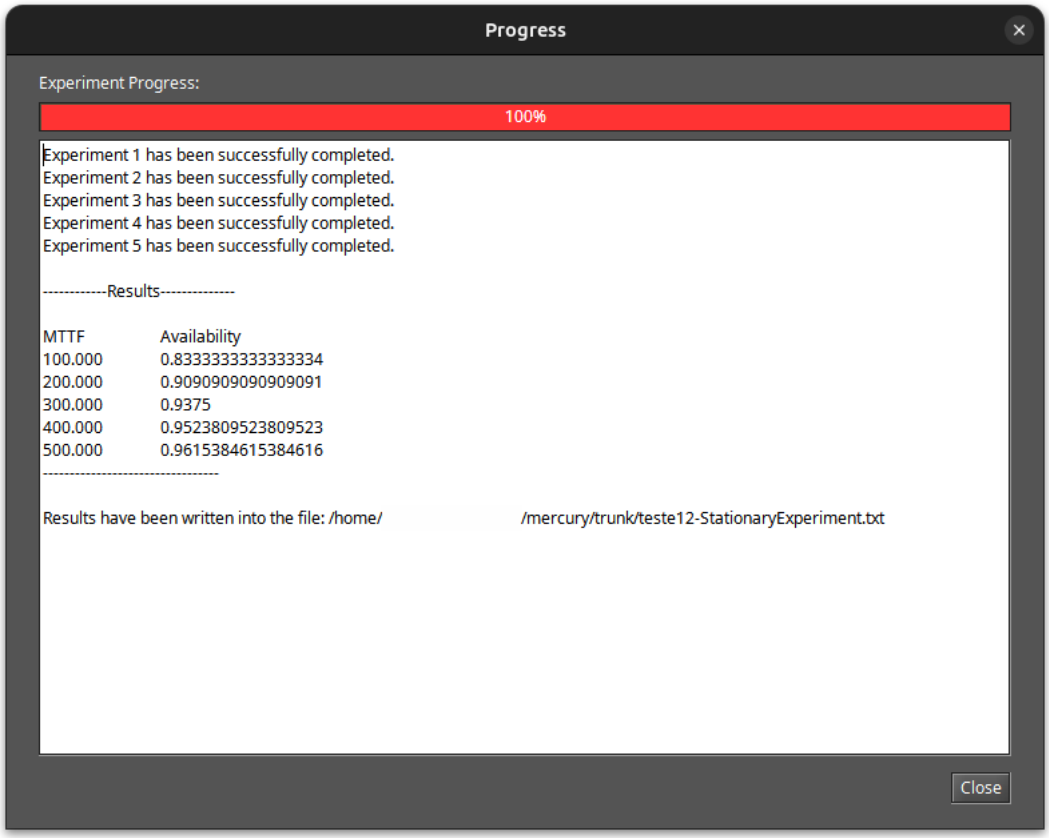


Figure 90: Results from an SPN Experiment

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.

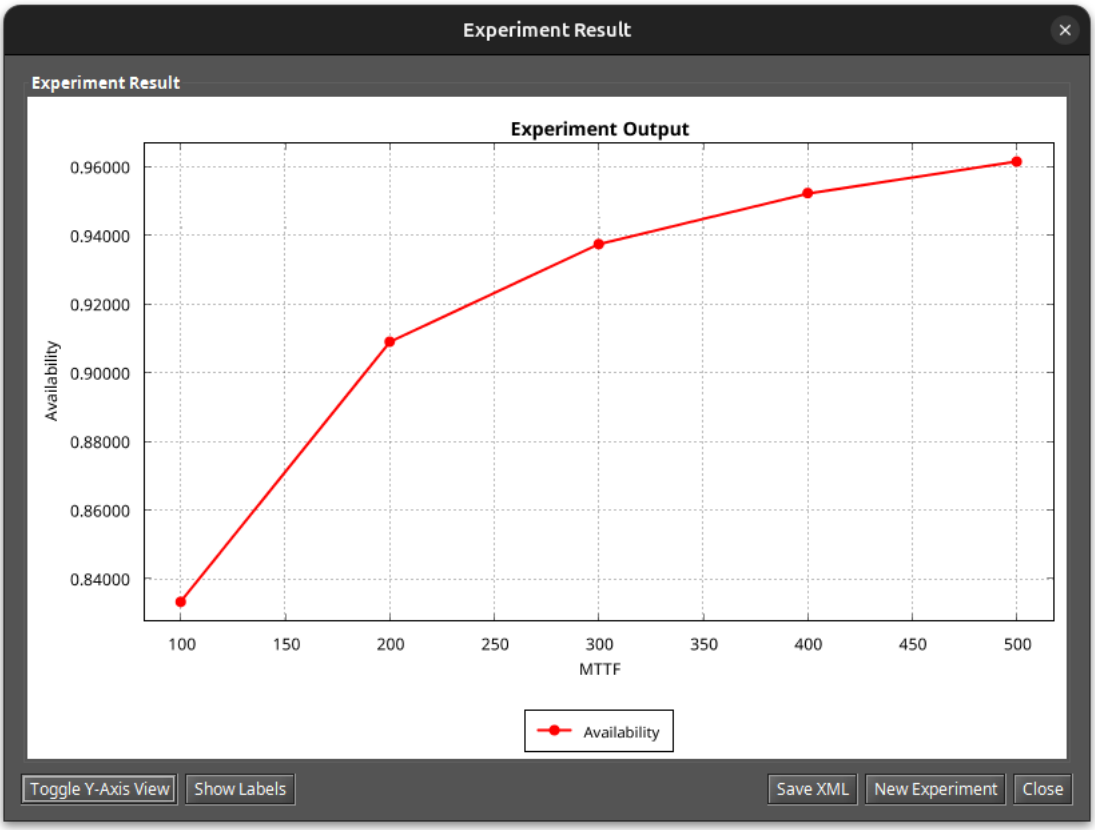


Figure 91: Graph from an SPN Experiment

2.3 SPN Structural Analysis

Mercury provides a feature to analyze SPNs without generating the reachability graph, but by considering the structure of the model. The "Structural Analysis" allows to prove some properties of a SPN model using **invariants** and **traps** techniques. It is accessible from the menu *Evaluate -> SPN Evaluation -> Structural Analysis* (see Figure 93).

When the structural analysis is complete, the "Structural Analysis" window is displayed with various tabs, each containing information about the structural properties of the SPN: **Matrix O (output matrix)**, **Matrix I (input matrix)**, **Matrix C (incidence matrix)**, **Matrix H (inhibitor matrix)**, **Classification**, **Invariant Analysis**, and **Siphons/Traps**. In the same window, you can export the result to a plain text file by clicking the "Save to file" button.

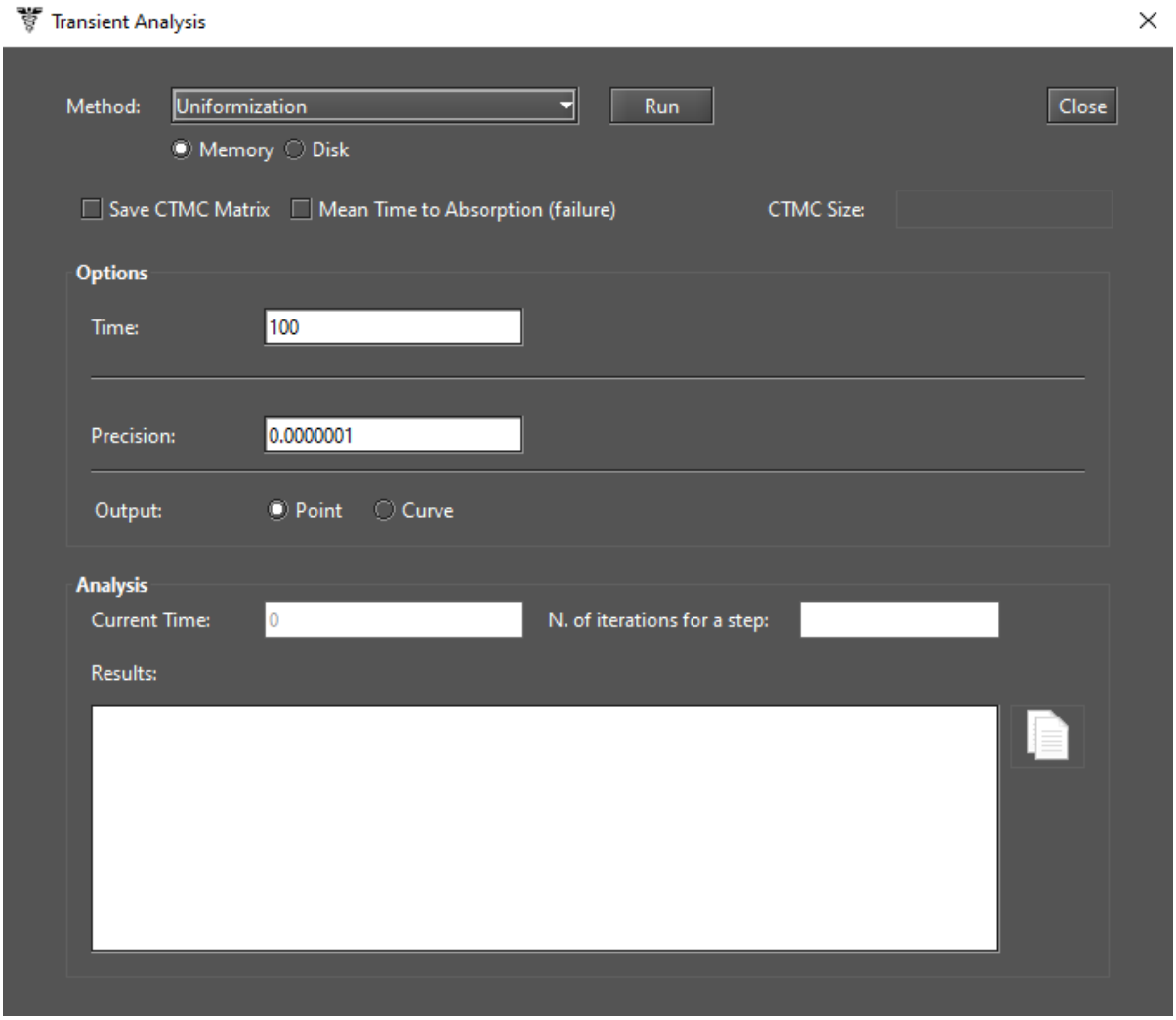


Figure 92: Transient Analysis Window

2.4 Token Game

Token Game allows us to simulate the behavior of SPN models. In other words, users can debug the model. In this way, errors in the construction phase of the model can be easily detected and improvements can be made to fix them. Figure 95 shows the model we took as an example.

Considering the guard expression defined below, there is a transition called ALL_F that represents the behavior of a system. In other words, when it fires, the mode of that system changes from operational to faulty. The guard expression associated with this transition (ALL_F) is represented as follows.

$$((\#Termination9\_ON = 0)OR(\#CircuitBreaker8\_ON = 0)$$
$$\text{OR } (((\#UPS0\_ON = 0)OR(\#SDTransformer3\_ON = 0))$$
$$\text{AND } ((\#UPS7\_ON = 0)OR(\#SDTransformer6\_ON = 0))))$$

Considering our SPN model and using Token Game, users can simulate device failures as well as the corresponding consequences on the availability of a system. To turn Token Game on or off, the user must click on the "Token Game" button highlighted in Figure 96. Also, we can see in this figure that any component can fail by

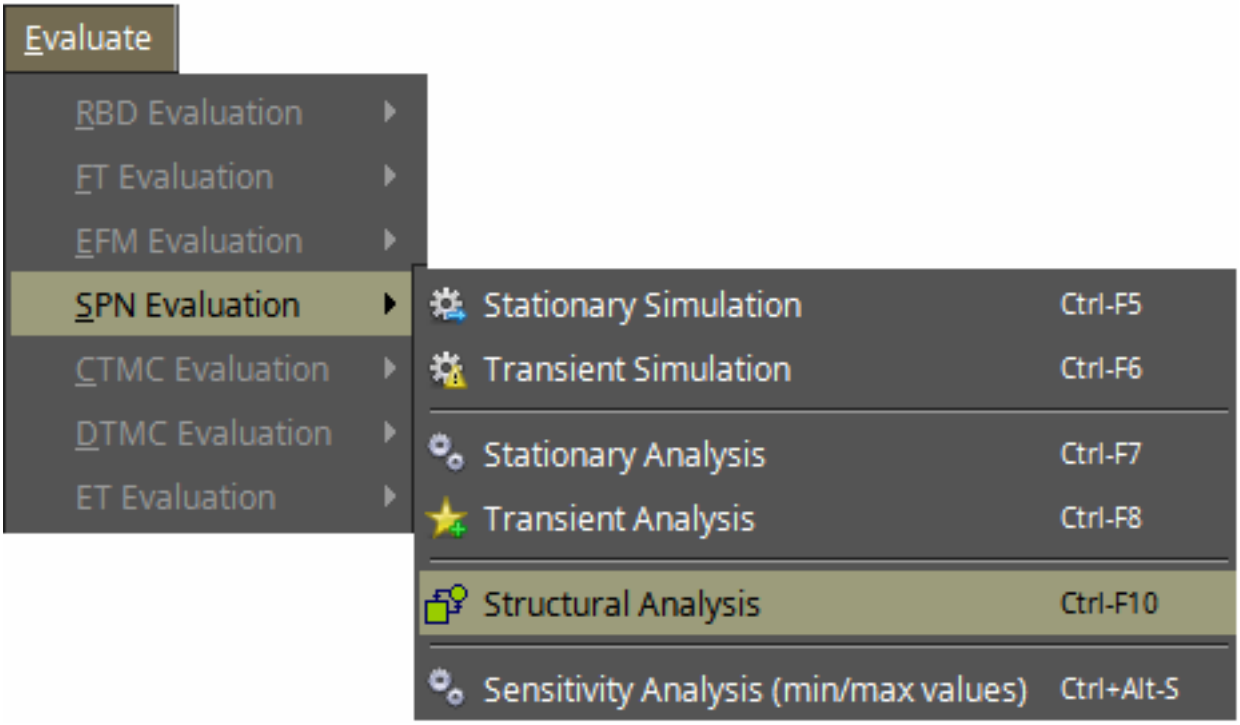


Figure 93: Structural Analysis Function

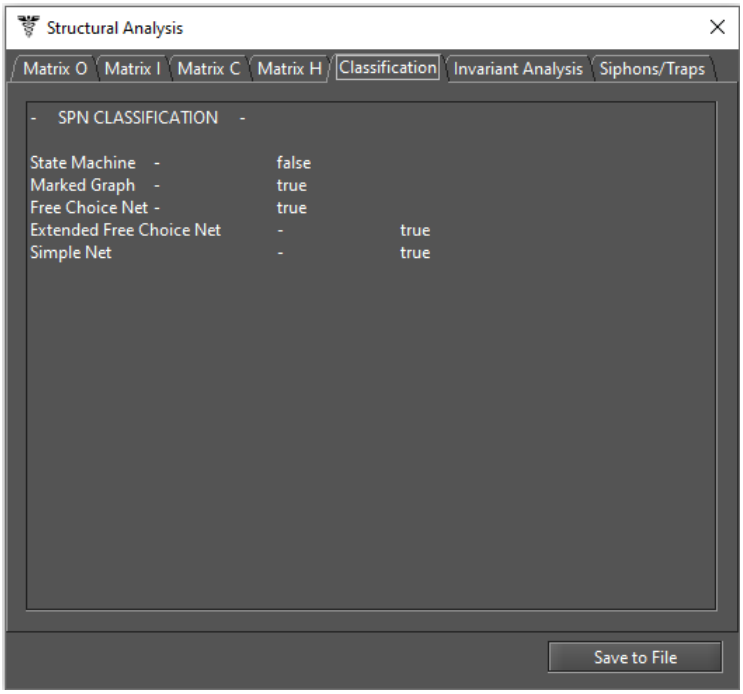


Figure 94: Structural Analysis Window

triggering an enabled transition (the ones highlighted in green).

If we assume that the transition “Termination9_F” was fired (see Figure 97), this means that the termination failed. So we see that only one transition is ready to be triggered (*ALL_F*), which means that the system has switched to failure mode as expected.

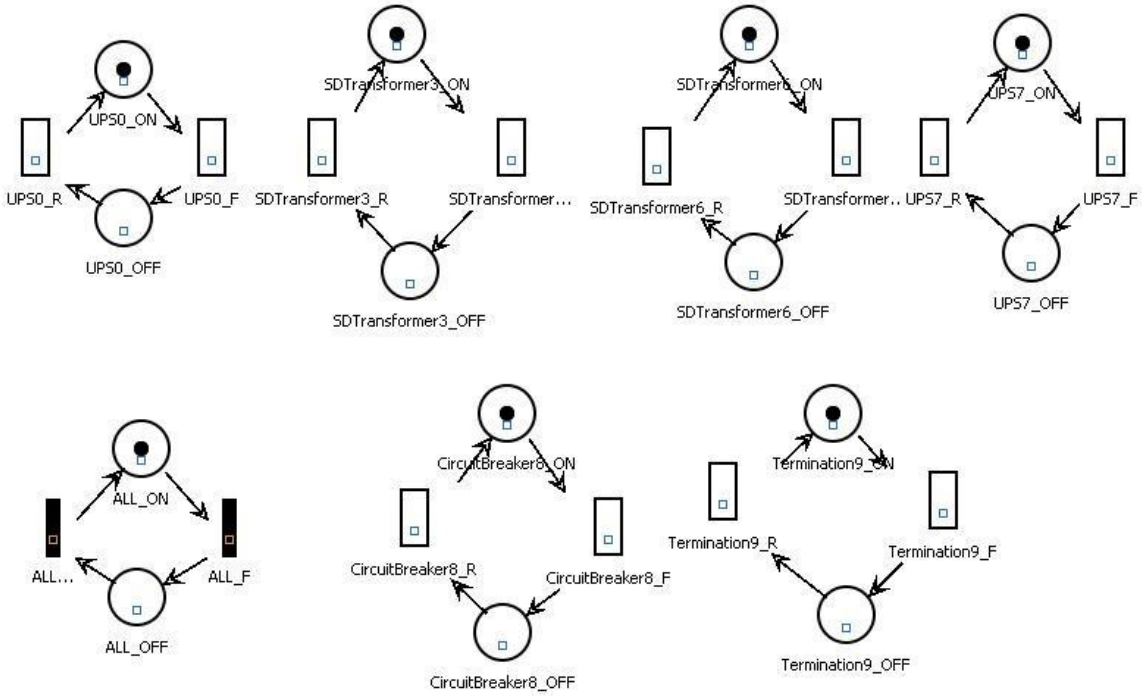


Figure 95: SPN Model

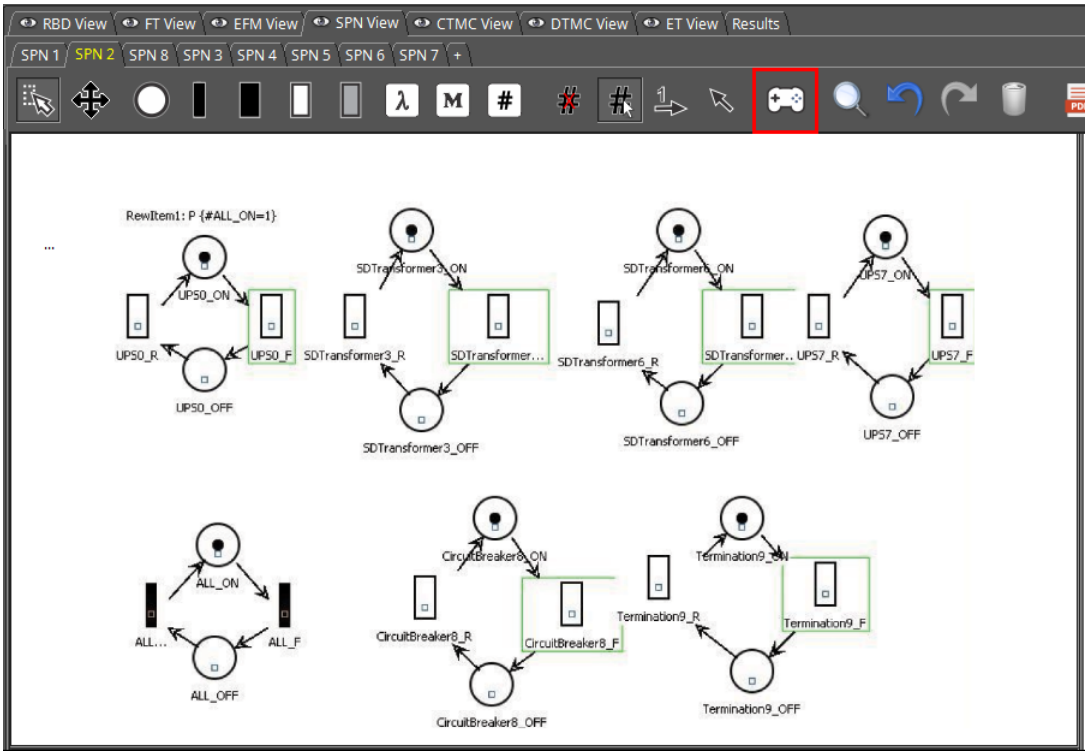


Figure 96: Turning Token Game On

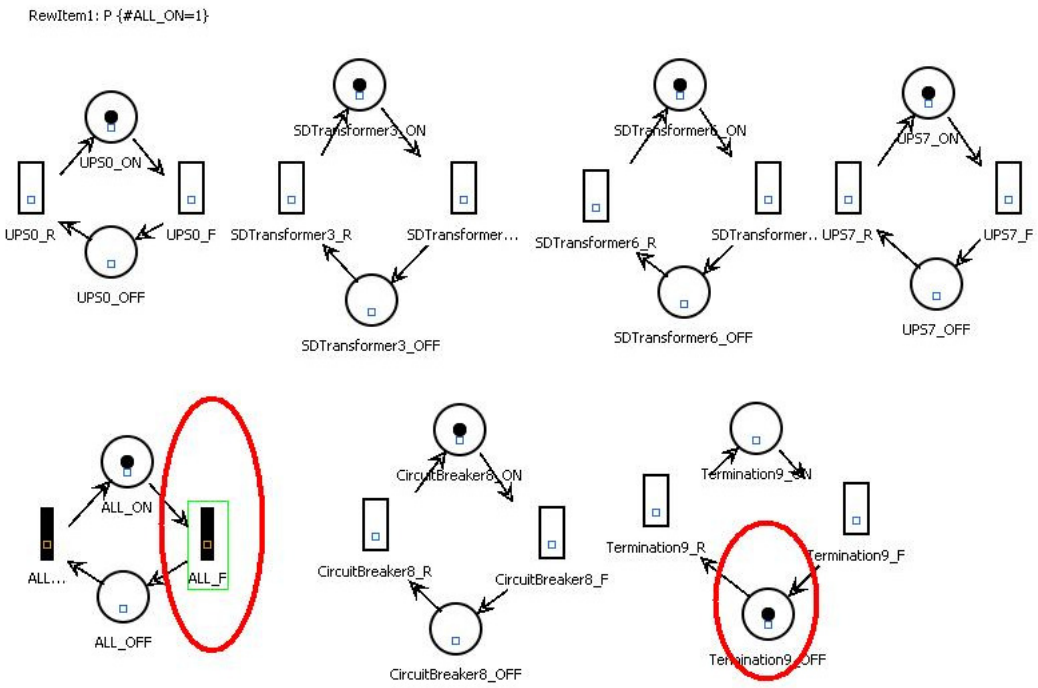


Figure 97: Example of a Token Game

2.5 Sensitivity Analysis

Mercury allows us to perform a sensitivity analysis for SPNs and calculate partial derivative sensitivity indices for them. This analysis is accessible from the *Evaluate -> SPN Evaluation -> Sensitivity Analysis (min/max values)* menu. Figure 98 shows the "Sensitivity Analysis" window.

Users must choose between two methods of sensitivity analysis: "Design of Experiments" and "Sensitivity Indices". The former uses the standard method of analysis of variance to determine the effect of each factor on the results. The latter uses the technique of percentage difference, thus requires a minimum and a maximum value for each parameter to calculate the corresponding percentage variation on the selected metric.

Mercury is able to calculate the sensitivity to an SPN measure with respect to any SPN delay parameter.

The results of the sensitivity analysis are presented in three possible output formats: "None", "JFreeChart", and "R". Users must select one of these outputs, as shown in Figure 99. The default option is "None", which outputs the results to the text area at the bottom of the window.

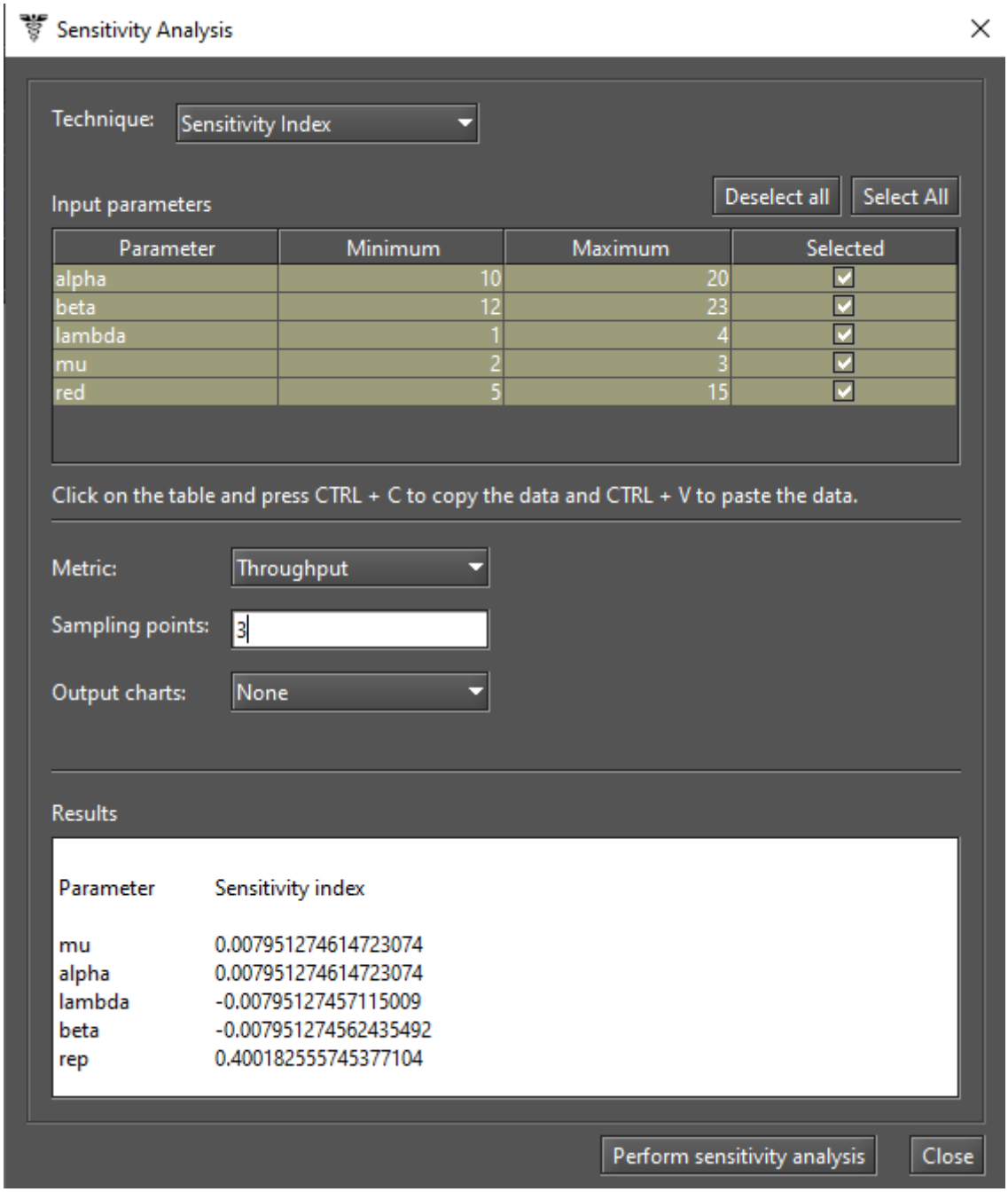


Figure 98: Sensitivity Analysis for an SPN Model

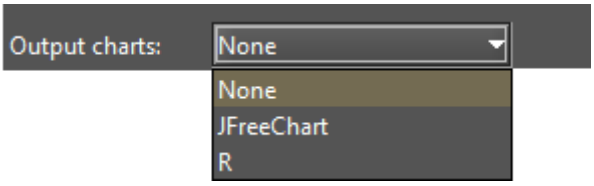


Figure 99: Output Options for Sensitivity Analysis

3 RBD Modeling and Evaluation

Reliability Block Diagram (RBD) is a success-oriented modeling approach to support dependability assessments. By evaluating RBDs, users can see how the failure or success of individual components contributes to overall reliability and availability. When you create a project, the default RBD model contains only one empty block. When you access the RBD view, you can see that this default model contains a light gray block named b1 (see Figure 100). This color indicates that the properties of this block have not yet been defined. RBD is evaluated from left to right.

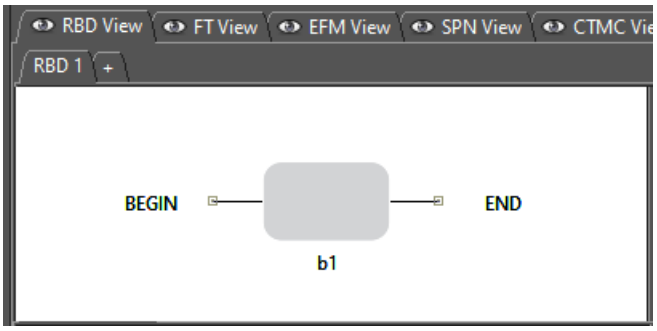


Figure 100: Default RBD Model

Unlike the other formalisms, the RBD view does not have a toolbar that allows the user to select components and make changes to the model. All operations to change the model are performed by selecting menu items with the mouse. For example, the user must right-click on the first block to create another block. As you can see in Figure 101, there are some options in the popup menu. You make changes to the model by selecting the appropriate action in the respective menu item. Among the available options you can find the basic operations: insert, edit and remove blocks.

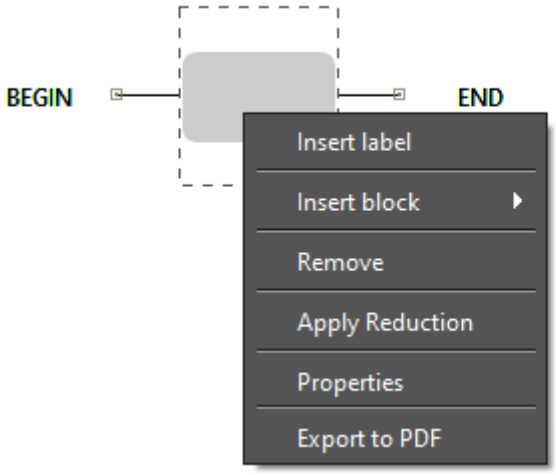


Figure 101: Popup Menu for RBD Blocks

To insert a block, the “Insert block” menu must be selected. Depending on how the new block is to be connected to the other blocks, you must select the “Series” or “Parallel” menu. For each of these blocks there are

two types of blocks that can be created: “Simple Block” and “k-out-of-n Block” (see Figure 102).

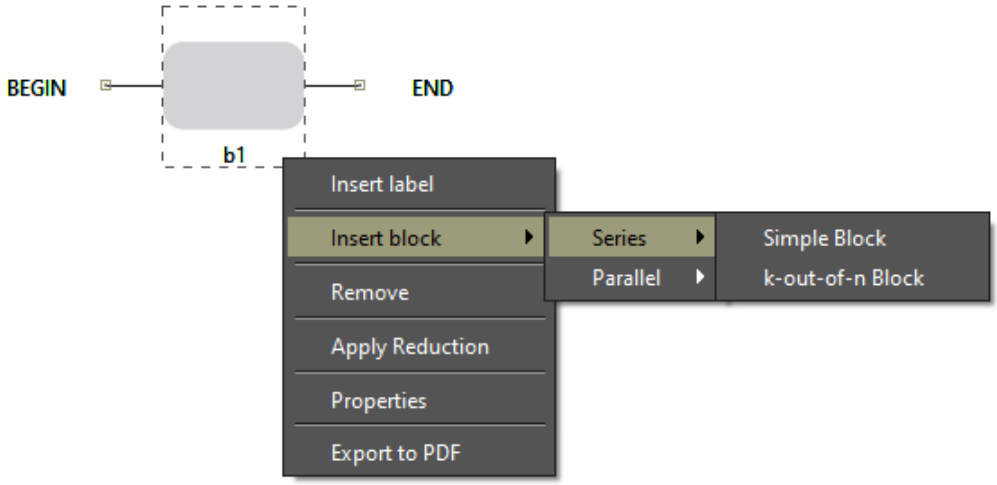


Figure 102: Inserting an RBD Block

When you select the type of block to insert, a window appears where you can specify the properties of the new block(s). Figure 103 shows the dialog that appears when inserting a simple block, and Figure 104 shows the dialog that appears when inserting a k-out-of-n block. As you can see, the only difference between the two dialogs is that two new fields (K and N) are displayed when the “k-out-of-n Block” option is selected. In addition to the properties of the block, you must also specify how many blocks should be inserted. In this case all new blocks will have the same properties.

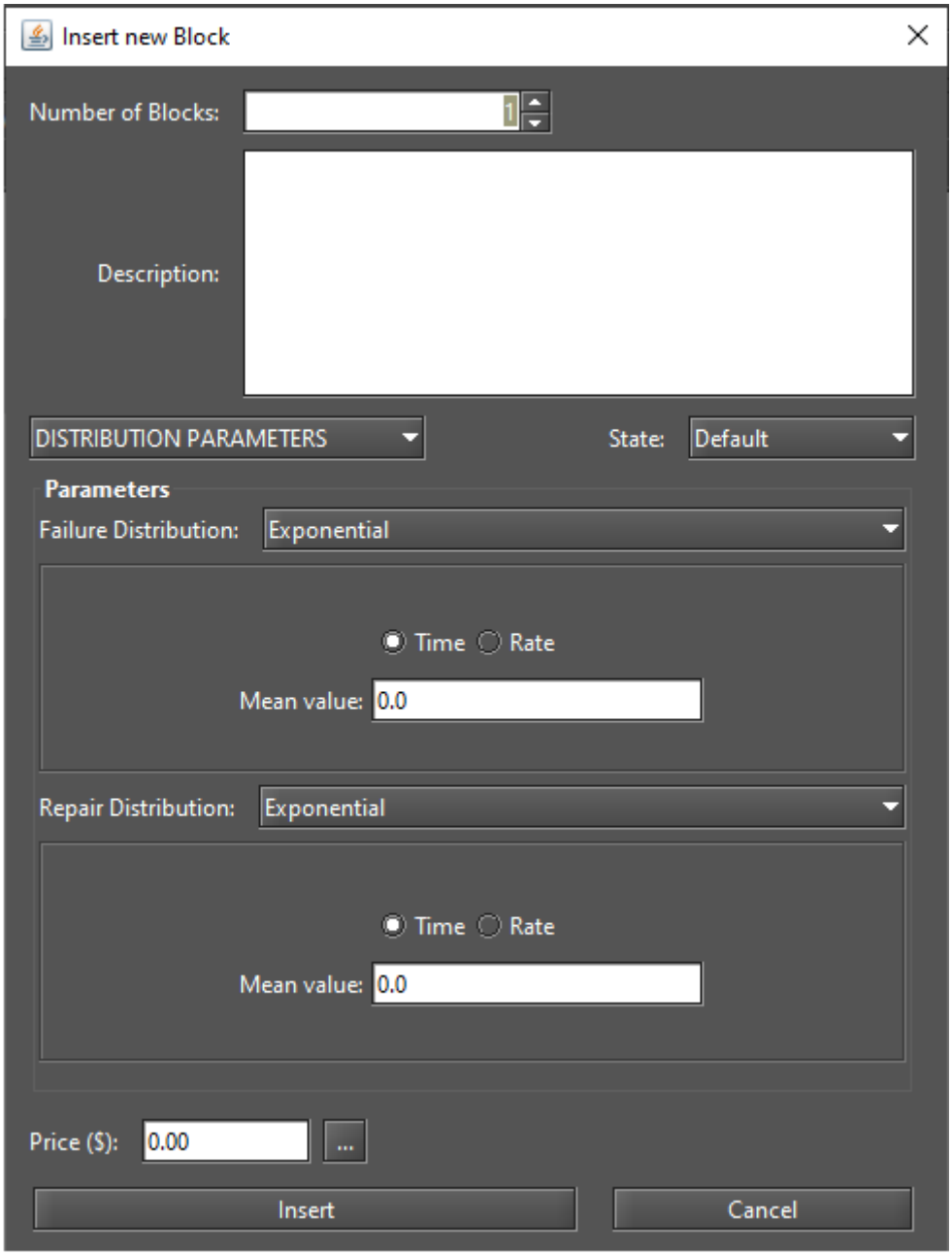


Figure 103: Inserting a Simple Block

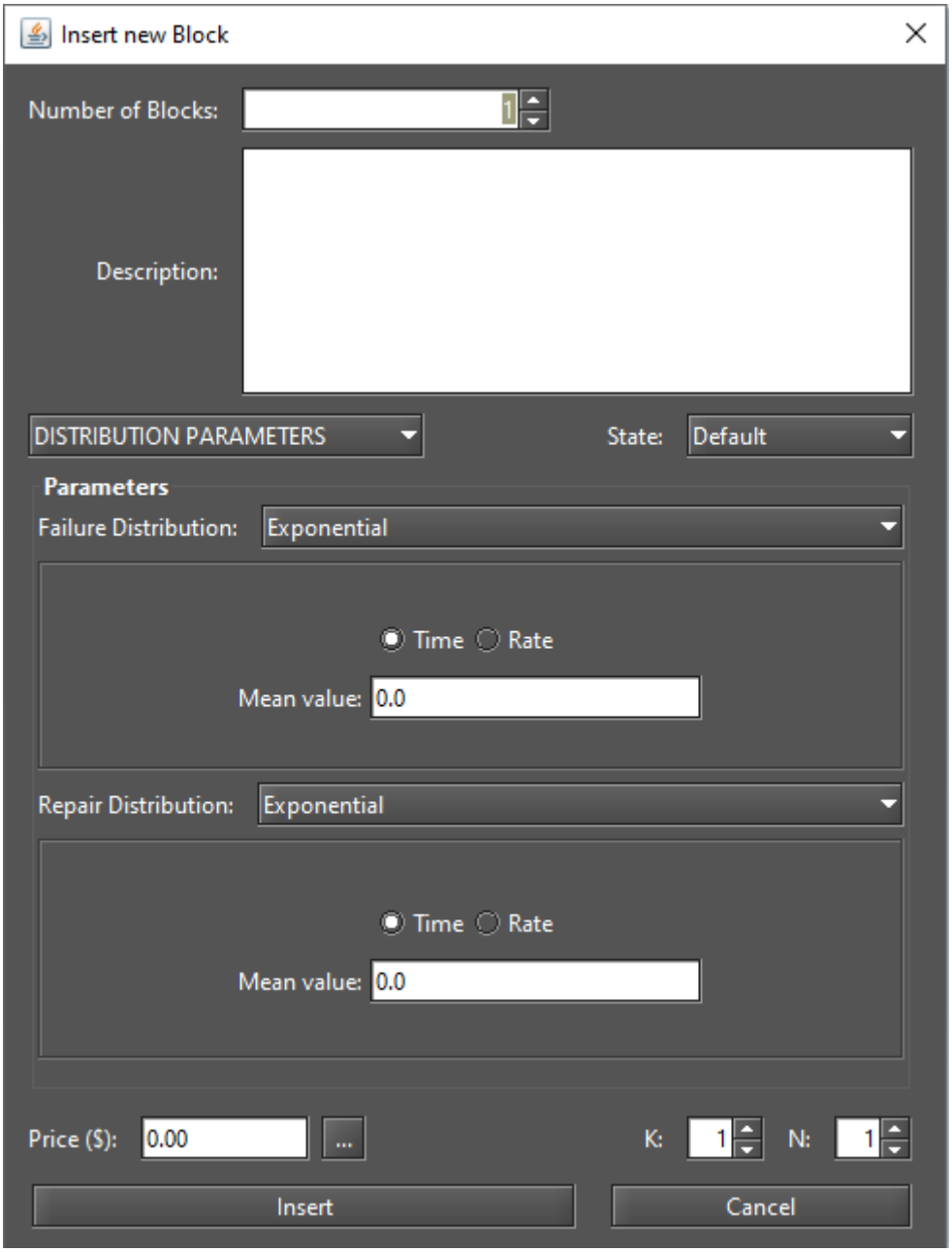


Figure 104: Inserting a K-out-of-N Block

Once a block is created, you can edit its properties by right-clicking on it and selecting the “Properties” menu. Another option is to double-click on the block. A third option is to double-click on the component view in the upper left panel under the “RBD Model” icon (see Figure 105). Figure 106 shows the properties of an already created simple block. The moment a block is inserted, the “Block Name” field is not displayed to the user because the name for that block is automatically set by Mercury (see Figure 103). Once the block is created, you can change its name by accessing its properties.

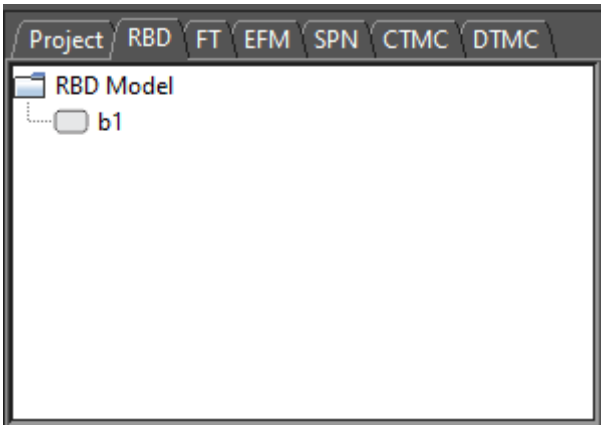


Figure 105: Top-Left RBD panel

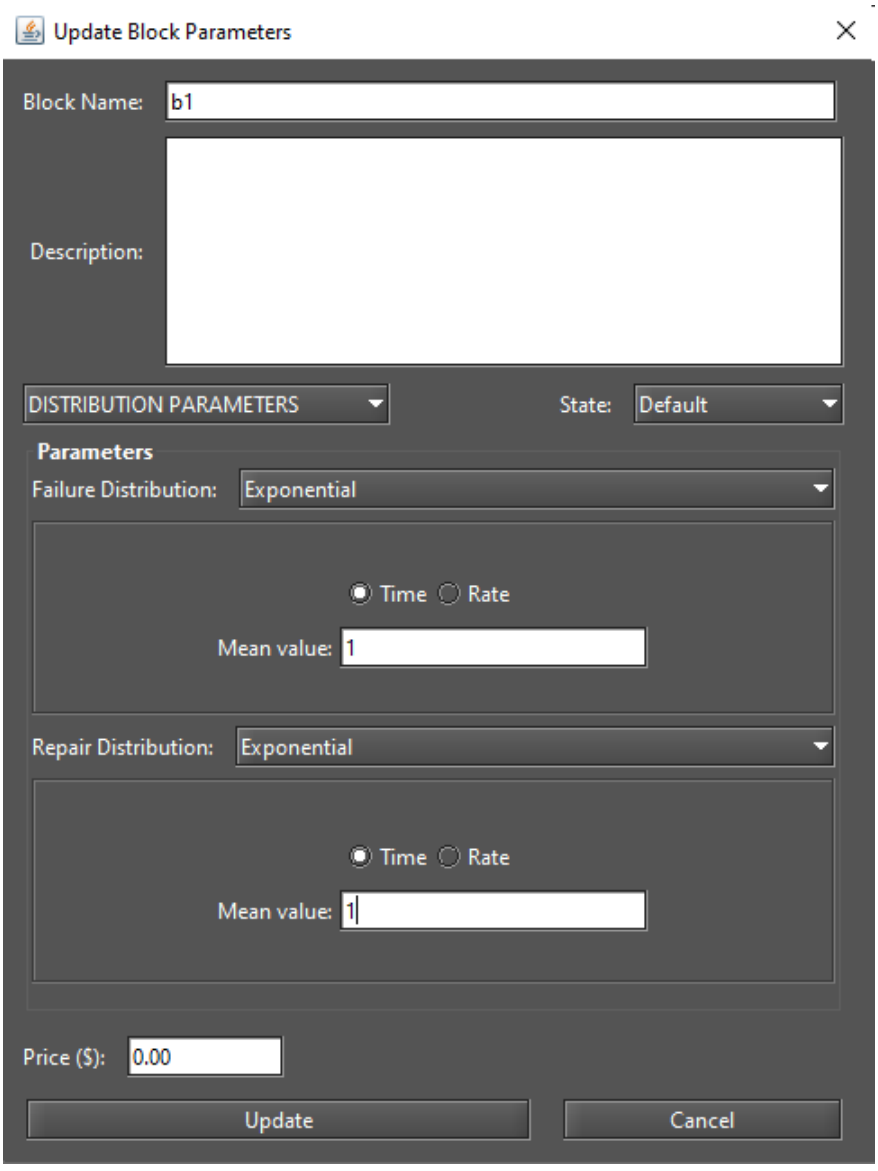


Figure 106: Properties of Simple Blocks

Next, let us get an overview of block properties.

- **Number of Blocks.** The moment a block is created, the number of blocks to be created is queried. If you enter a value greater than 1, all blocks will be assigned the same parameter values.
- **Block Name.** Name for the block. At the time a block is created, Mercury determines its name. Once created, the name can be changed by the user by accessing the block's properties.
- **Description.** A description is additional information about the block or the component/subsystem it represents. It is intended to improve understanding of the model and has no semantic value in evaluating the model. It is just plain text attached to the block.
- **Parameters Type.** Blocks accept three types of parameters: DISTRIBUTION PARAMETERS, AVAILABILITY, and RELIABILITY. At any given time, only one of them can be selected. The default type is DISTRIBUTION PARAMETERS. If the parameter type is DISTRIBUTION PARAMETERS, the user can enter the appropriate values for the failure and repair parameters (see Figures 103, 104, and 106). If the type is AVAILABILITY or RELIABILITY, the user can enter the appropriate value considering the selected type, as we can see in Figure 107. In the context of the last figure, the user must enter the availability of the component represented by the block.
- **State.** State of the block. Two states are available: DEFAULT or FAILED. The default state is DEFAULT, which means that the block is working properly.
- **Failure Parameters.** Mercury supports a large number of probability distributions. Fields appear representing the parameters of the selected distribution so that the user can enter their values. Each failure parameter may be assigned a label. Using the **"..."** button we can select an already declared label.
- **Repair Parameters.** Fields appear for the parameters of the selected distribution, where the user can enter the appropriate values. Each repair parameter can be provided with a label. Using the button **"..."** we can select an already declared label.
- **Price.** Cost in terms of the component represented by the block. The cost of blocks is considered by evaluating the model using the "Component Importance and Total Cost of Acquisition" method. See Section 3.2.4 for more information on this type of evaluation.
- **K and N.** Two fields appear when k-out-of-n blocks are edited. A KooN block represents a set of N identical components in a single block. All components in this set have the same failure and repair parameters. This type of block allows the user to specify the minimum number of components (K) that must be functioning in order for failure not to occur. Figure 108 shows how a KooN block is represented. As we can see, the values of the K and N parameters are shown next to the block name in the diagram.

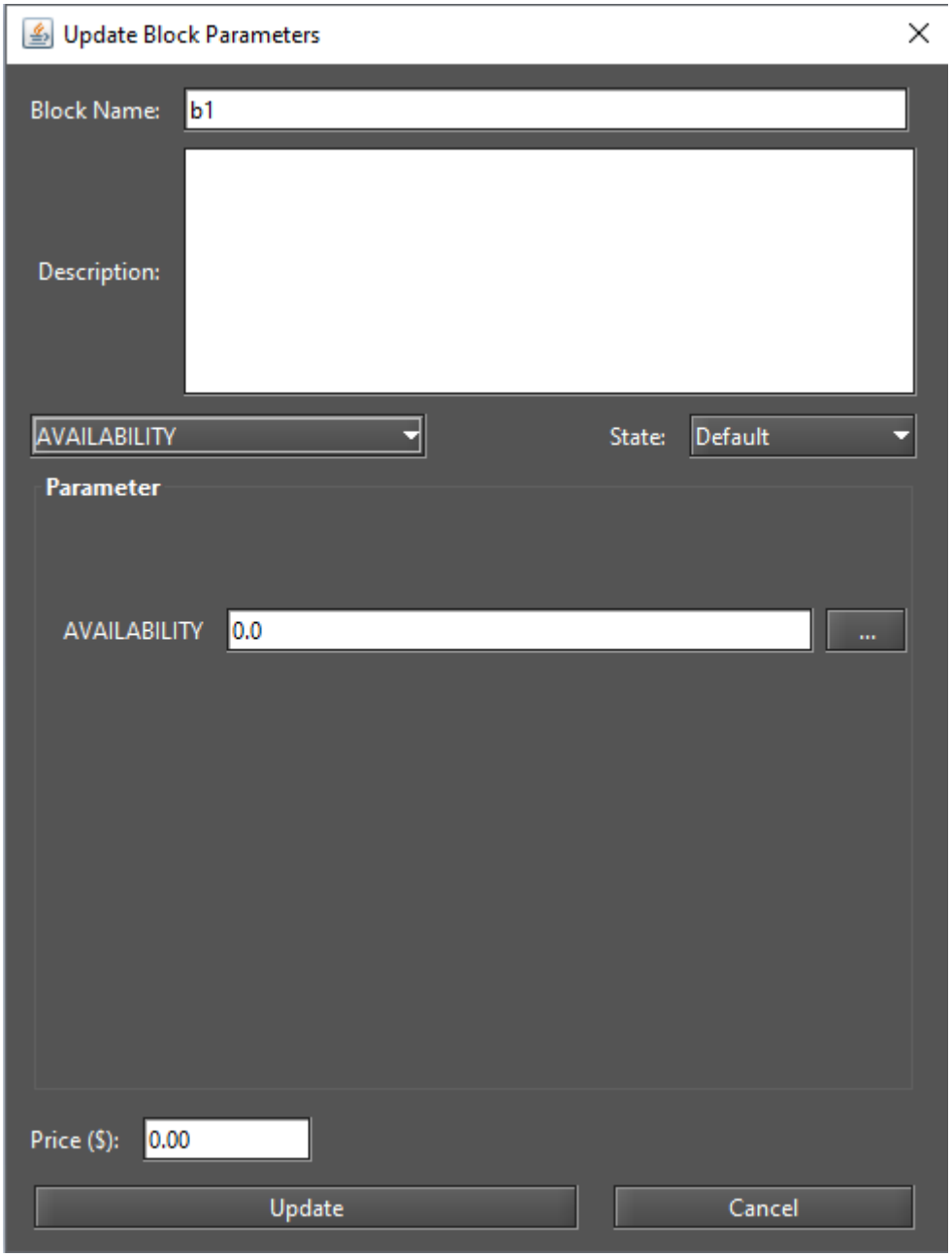


Figure 107: Defining the Availability of a Block

Figure 108: An Exponential KooN block

Figure 109 shows an RBD with two blocks in series and Figure 110 shows an RBD with two blocks in parallel. The color of the block changes by assigning all required parameters. Evaluations can only be performed if all required parameters of all blocks are entered.

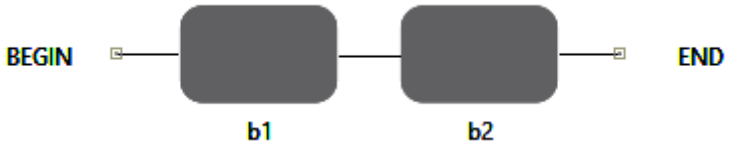


Figure 109: Two Blocks in Series

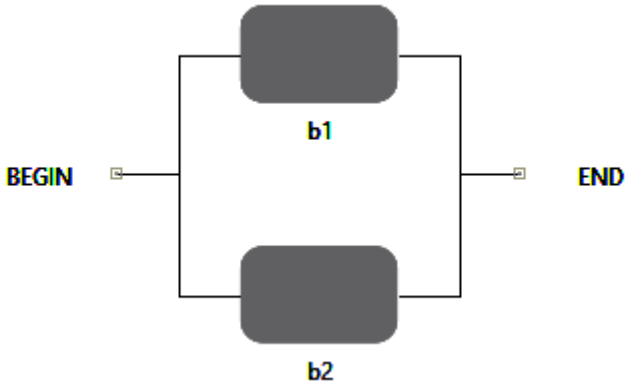


Figure 110: Two Blocks in Parallel

Mercury has a feature to improve the readability of models. Once the parameters of a block have been assigned, you can read them on the diagram by moving the mouse pointer over the block. A tooltip will then appear showing all the properties of that block. As we can see in Figure 111, all properties are displayed in the tooltip. All types of components of all formalisms supported by Mercury provide this feature.

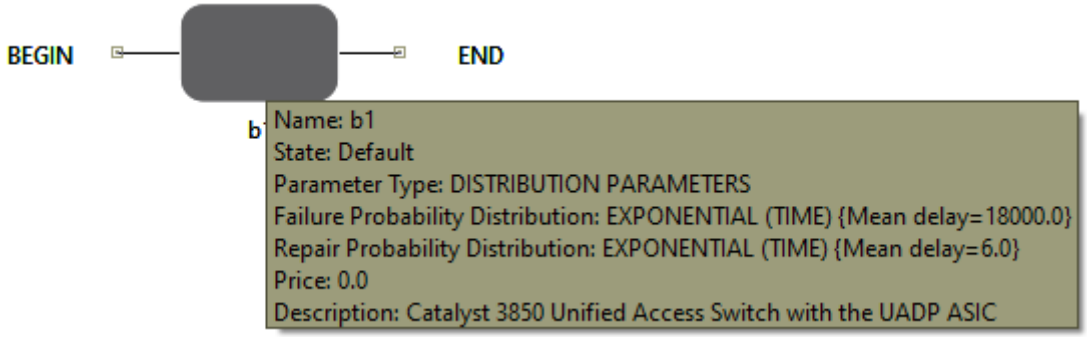


Figure 111: Tooltip for a Block

Finally, let us look at the types of blocks and how they are represented graphically. Figure 112 shows the six types of RBD blocks. Figures 112.a and 112.b show blocks that have no parameters associated with them. Figures 112.c and 112.d show exponential blocks, but in c) the state of the block is defined as “Default”, while in d) it is defined as “Failed”. Figures 112.e and 112.f show 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 without failure/repair parameters are represented by a light gray block. Exponential blocks are represented by a dark gray block. Finally, non-exponential blocks are represented by a blue block.

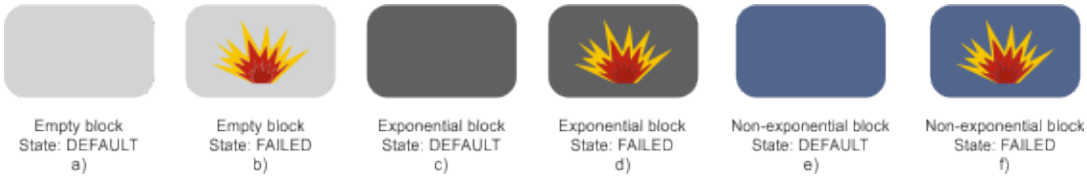


Figure 112: Types of RBD Blocks

3.1 RBD Reduction

To reduce the complexity of RBD models, a feature is available that supports a reduction process aimed at reducing the number of blocks. This function can be used until the model consists of only one block. However, the original parameters of the blocks can be lost and only the metrics and properties of the original model are preserved.

Figure 113 shows a model before applying a reduction step, in which there are four blocks. This feature is applied by right-clicking on the block and selecting “Apply Reduction” (see Figure 114). Figure 115 shows the model after the reduction has been applied. It can be seen that the number of blocks has been reduced to three (b2 and b3 have been reduced to only one block, ssb6).

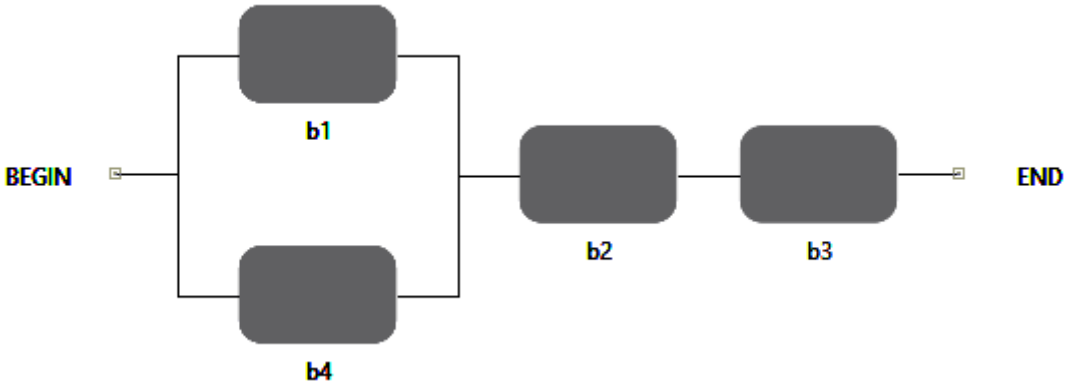


Figure 113: RBD Before a Reduction Step

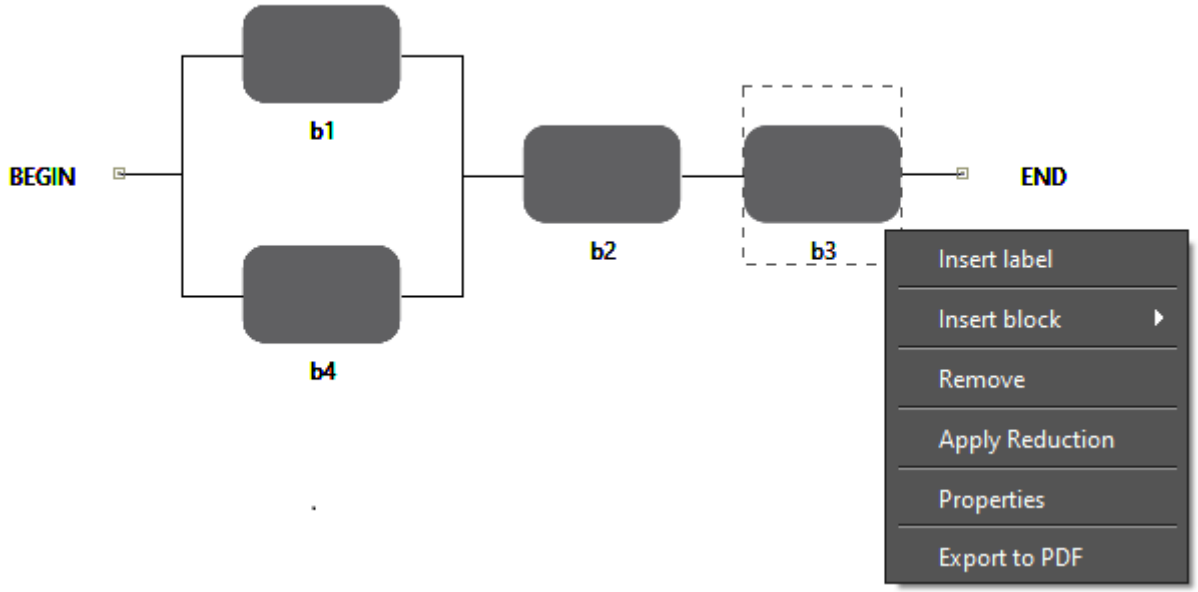


Figure 114: Applying Reduction to an RBD Model

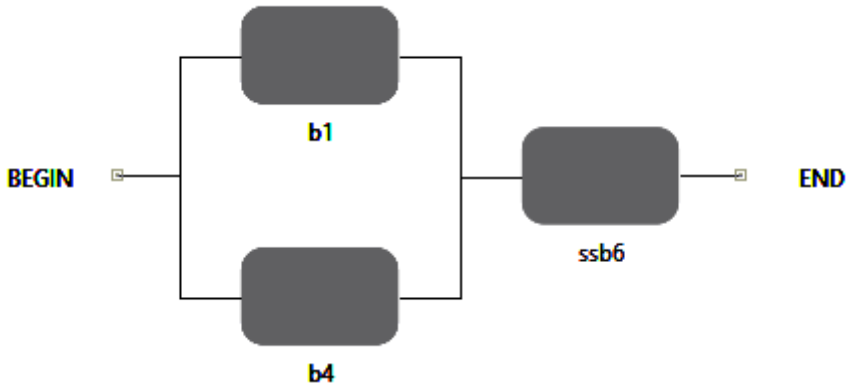


Figure 115: RBD After a Reduction Step

When one or more blocks to be reduced are connected in series, the original characteristics of the model are preserved. However, when applying reductions in blocks connected in parallel, if the TIME or RATE parameter have been edited, it is only possible to retain the characteristics at one point in time. Therefore, when applying a reduction in a block connected in parallel, new options are displayed, as shown in Figure 116. In this case, the user must select the metric to be evaluated. When reliability is selected, the time for reliability estimation is requested, as shown in Figure 117. Figure 118 shows the model after the blocks in parallel have been reduced.

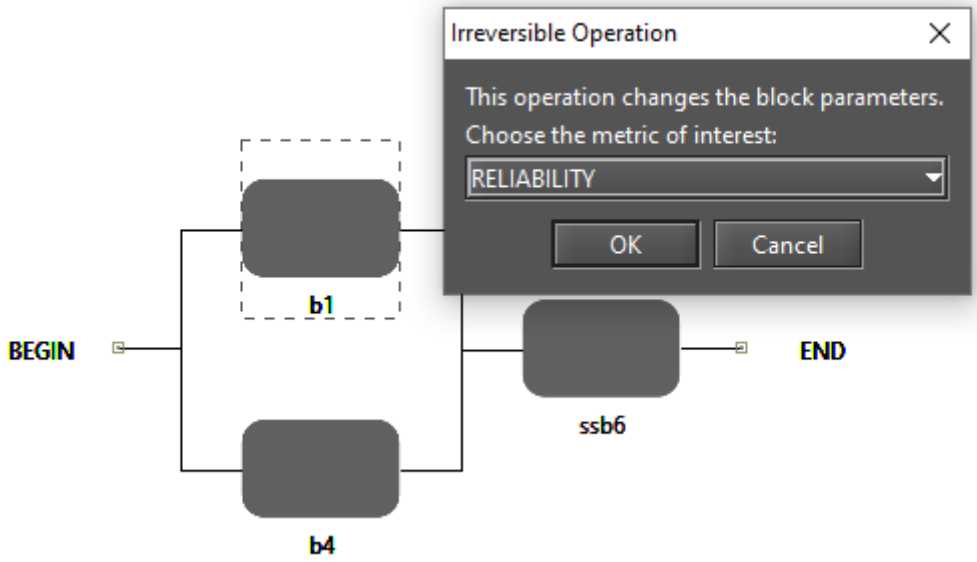


Figure 116: RBD Before the Blocks Connected in Parallel are Reduced

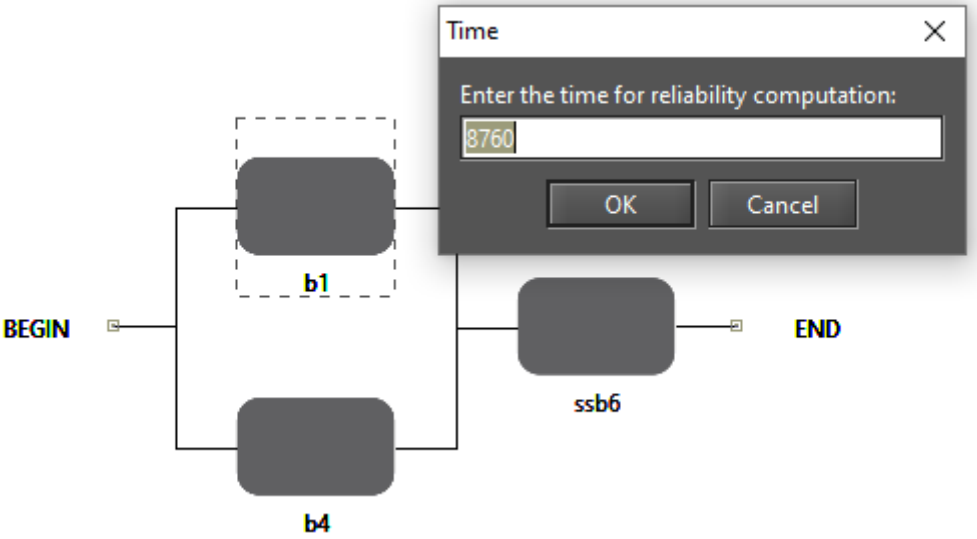


Figure 117: Entering the Time for Reliability Computation

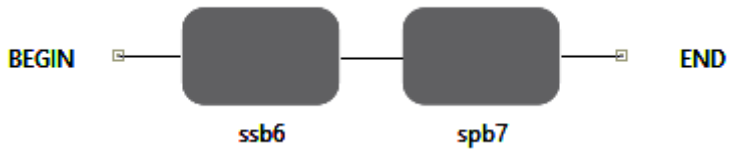


Figure 118: RBD After the Blocks Connected in Parallel Has Been Reduced

3.2 RBD Evaluation

Mercury offers a large number of evaluations for RBDs:

- Evaluation (SFM and SDP methods);
- Bounds Evaluation;
- Importance Measures;
- Experiment;
- Get Functions;
- Sensitivity Analysis; and
- Sensitivity Analysis (min/max values).

These evaluations are available from the *Evaluate -> RBD Evaluation* menu, as shown in Figure 119. In the next subsections, we present each evaluation.

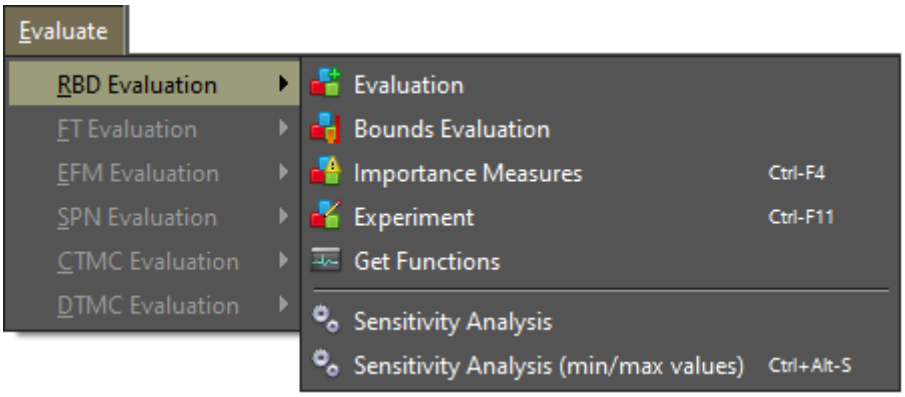


Figure 119: RBD Evaluation Menu

3.2.1 Evaluation

Evaluation can be used to perform a large number of dependability analyzes (see Figure 120). It can be accessed from the *Evaluate* -> *RBD Evaluation* -> *Evaluation* menu.

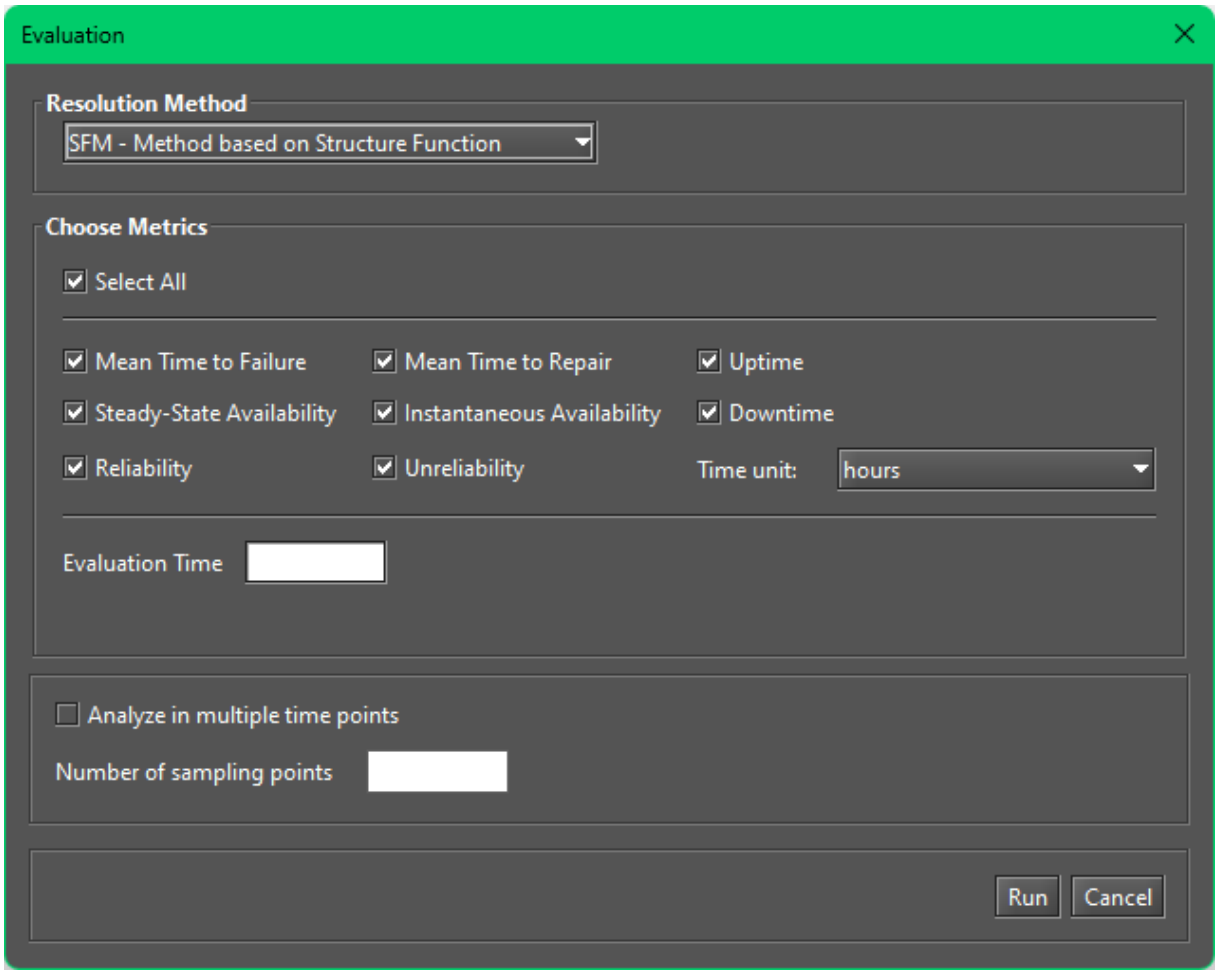


Figure 120: Evaluation

As you can see, you can evaluate eight metrics: mean time to failure, mean time to repair, steady-state availability, instantaneous availability, reliability, unreliability, uptime and downtime. Users can select or deselect all metrics by clicking the toggle checkbox labeled “Select All”. Users can also select the unit of time to be included in the calculation of uptime and downtime: seconds, minutes, hours, and days. If time-dependent metrics are selected — reliability, unreliability or instantaneous availability —-, the time parameter is required. There is also an option to analyze time-dependent metrics by considering multiple points in time. The metric is calculated for each point.

Mercury provides two methods for computing dependability measures. You can choose between SFM (structural function method) and SDP (sum of disjoint products), as shown in Figure 121. SFM computes measures considering the structural function of the model. The Boolean algebra-based SDP method, on the other hand, computes measures considering minimal cuts and paths.

After selecting the options and entering the evaluation time and number of sampling points, if required,

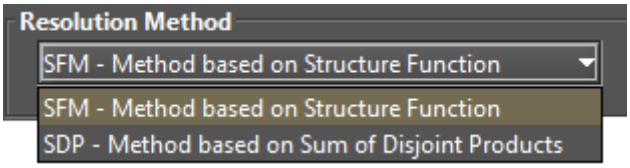


Figure 121: Resolution Methods

the user must click on the “Run” button. Once this is done, a window with the results will appear, as shown in Figure 122. 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 results obtained by evaluating an RBD.

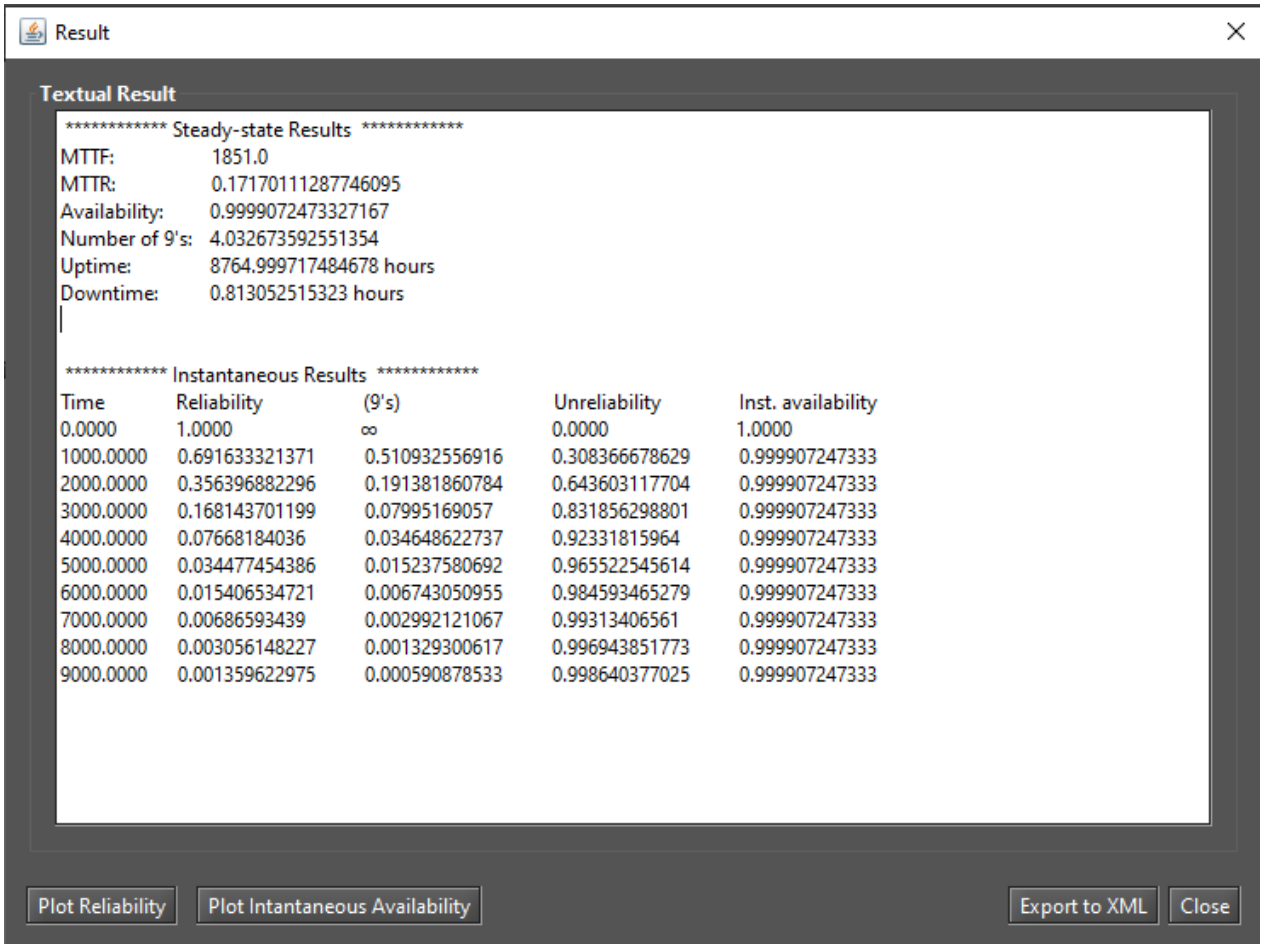


Figure 122: Results from Dependability Evaluations

Listing 5: Dependability Results for an RBD Model

```
MTTF:              1851.0
MTTR:              0.17170111287746095
Availability:      0.9999072473327167
Number of 9's:     4.032673592551354
Uptime:            8764.999717484678 hours
Downtime:          0.813052515323 hours
```

```
************ Instantaneous Results ************
Time            Reliability     (9's)           Unreliability
0.0000          1.0000          infinity        0.0000
1000.0000       0.691633321371  0.510932556916  0.308366678629
2000.0000       0.356396882296  0.191381860784  0.643603117704
3000.0000       0.168143701199  0.07995169057   0.831856298801
4000.0000       0.07668184036   0.034648622737  0.92331815964
5000.0000       0.034477454386  0.015237580692  0.965522545614
6000.0000       0.015406534721  0.006743050955  0.984593465279
7000.0000       0.00686593439   0.002992121067  0.99313406561
8000.0000       0.003056148227  0.001329300617  0.996943851773
9000.0000       0.001359622975  0.000590878533  0.998640377025
```

Figure 123 shows the "Reliability Chart" dialog box, which is displayed by clicking the "Plot Reliability" button. Figure 124 shows the "Instantaneous Availability Chart" dialog, which is displayed when you click the "Plot Instantaneous Availability" button. These buttons are only visible if the corresponding dependability metrics are selected in the input dialog. The number of points on the plotted lines is determined by the number of points entered by the user.

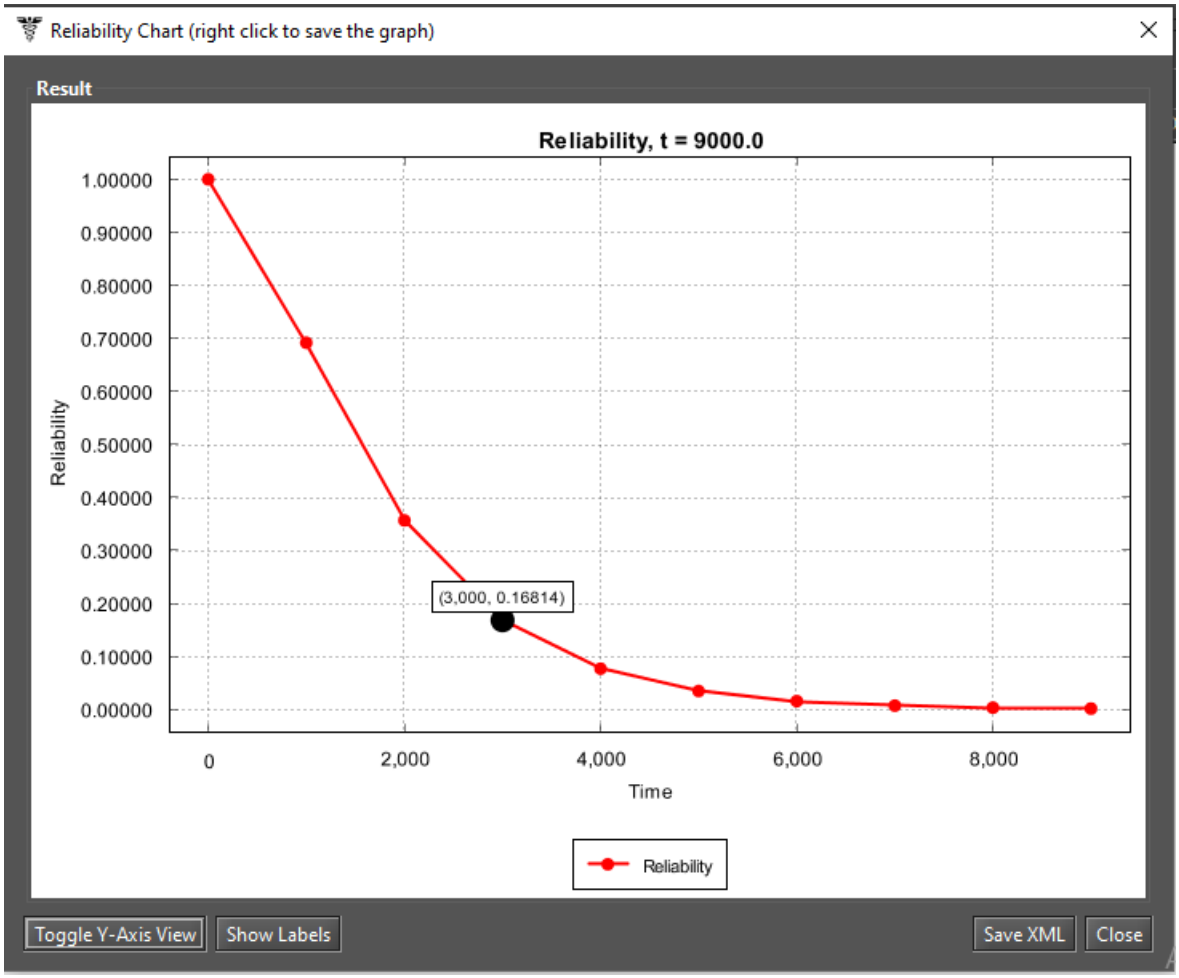


Figure 123: Reliability Chart

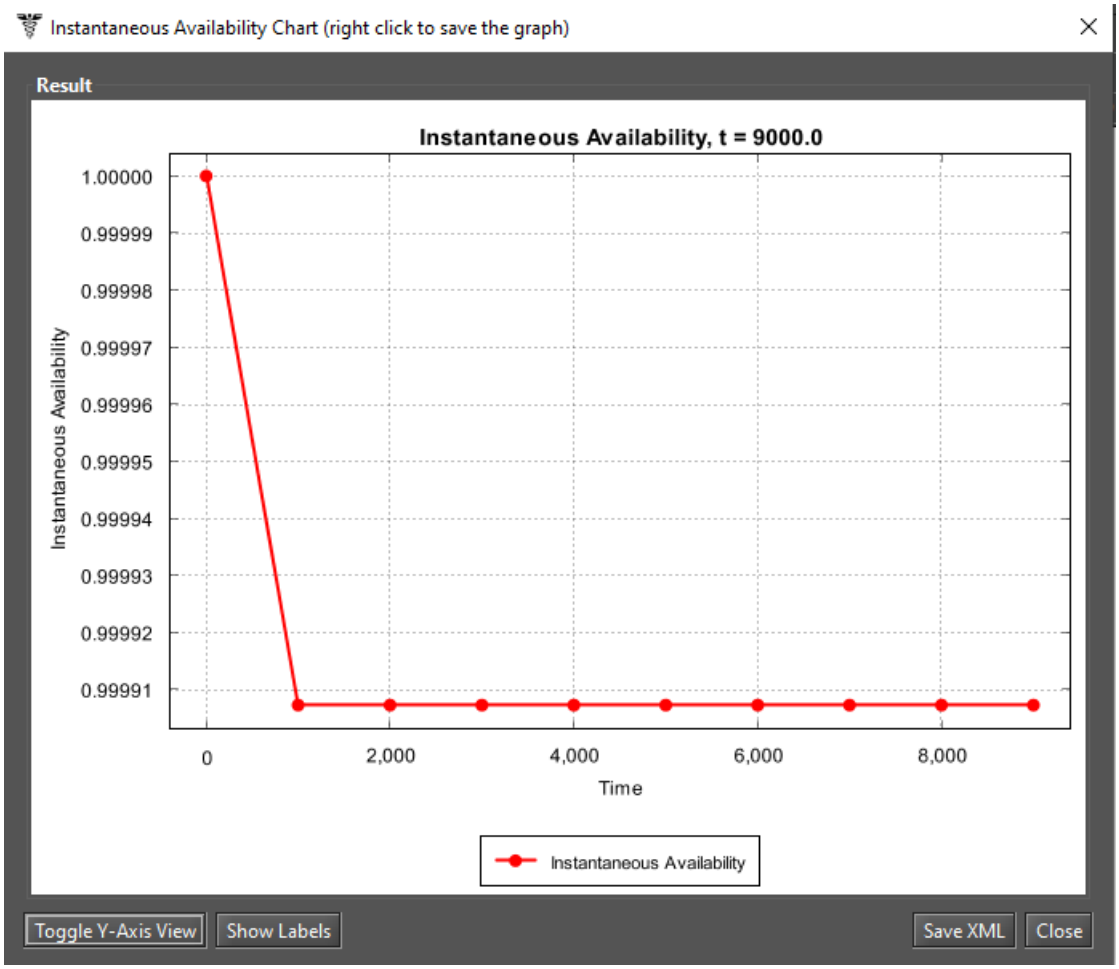


Figure 124: Instantaneous Availability Chart

RBD models can only be solved by simulation when non-exponential probability distributions are associated with the model. In this case, Mercury shows the message ”Non-exponential distributions detected” at the bottom of the “Evaluation” window (see Figure 125). Mercury calculates metrics from non-exponential models considering confidence intervals. The non-exponential blocks are converted into SPNs and these blocks are solved through simulations.

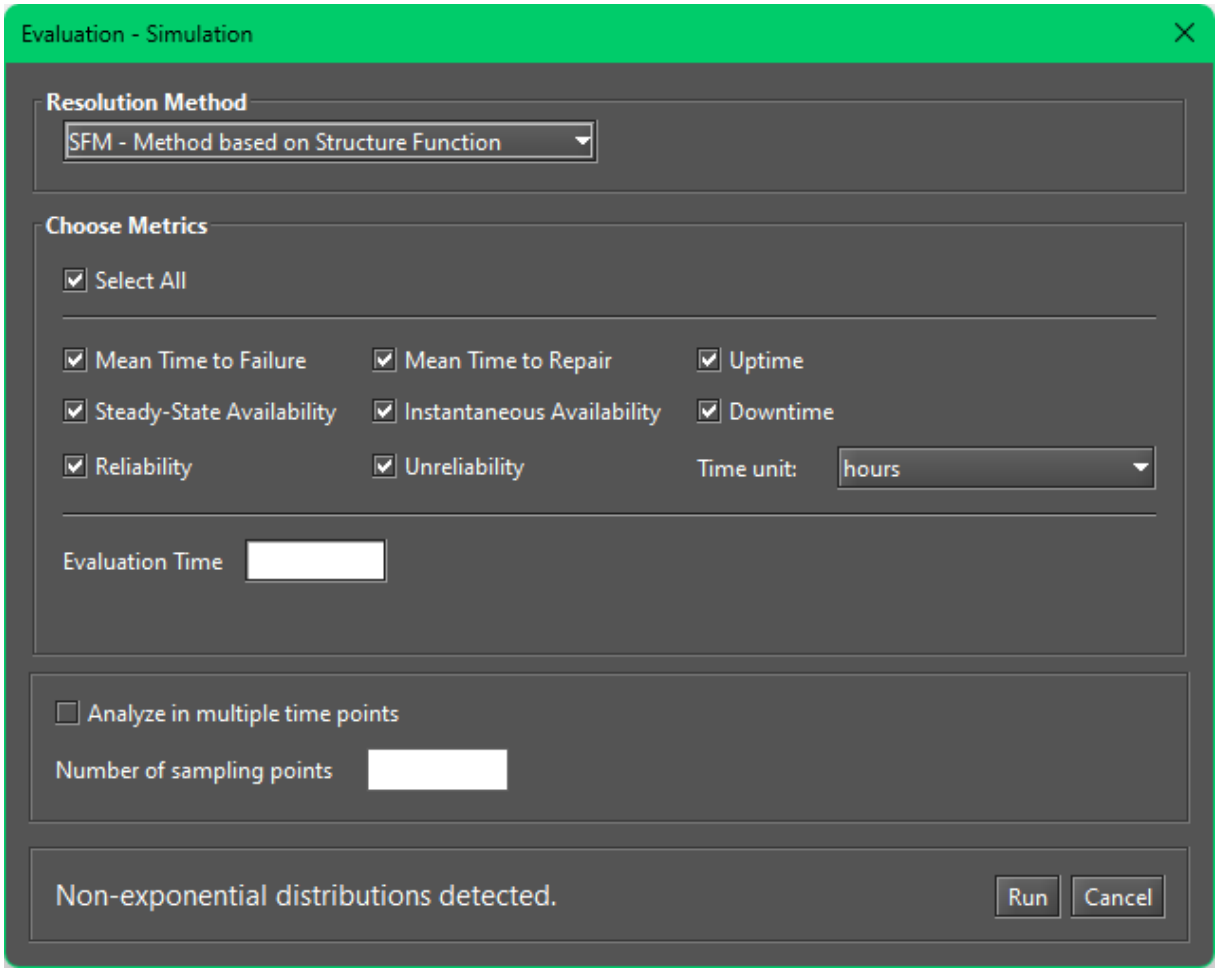


Figure 125: RBD Analysis by Simulation

When you click the “Run” button, some parameters must be entered to support the simulation. The parameters required depend on the metrics you choose. If you select only steady-state metrics, Mercury displays the window shown in Figure 126. On the other hand, if you select only time-dependent metrics, Mercury displays the window shown in Figure 127. If you select both transient and steady-state metrics, both tabs appear in the same window, as shown in Figure 128. In this case, when clicking the Run button in the dialog box shown in Figure 128, Mercury considers the parameters in both tabs. If no changes are made to the parameters, Mercury considers the default parameter values. The results are displayed once the parameters have been defined and the simulation is complete. Results are presented with confidence intervals, as shown in Figure 129. It is important to emphasize that the following metrics cannot be solved by running a simulation: MTTF, MTTR, and instantaneous availability. For more information about simulations, see Section 2.1.

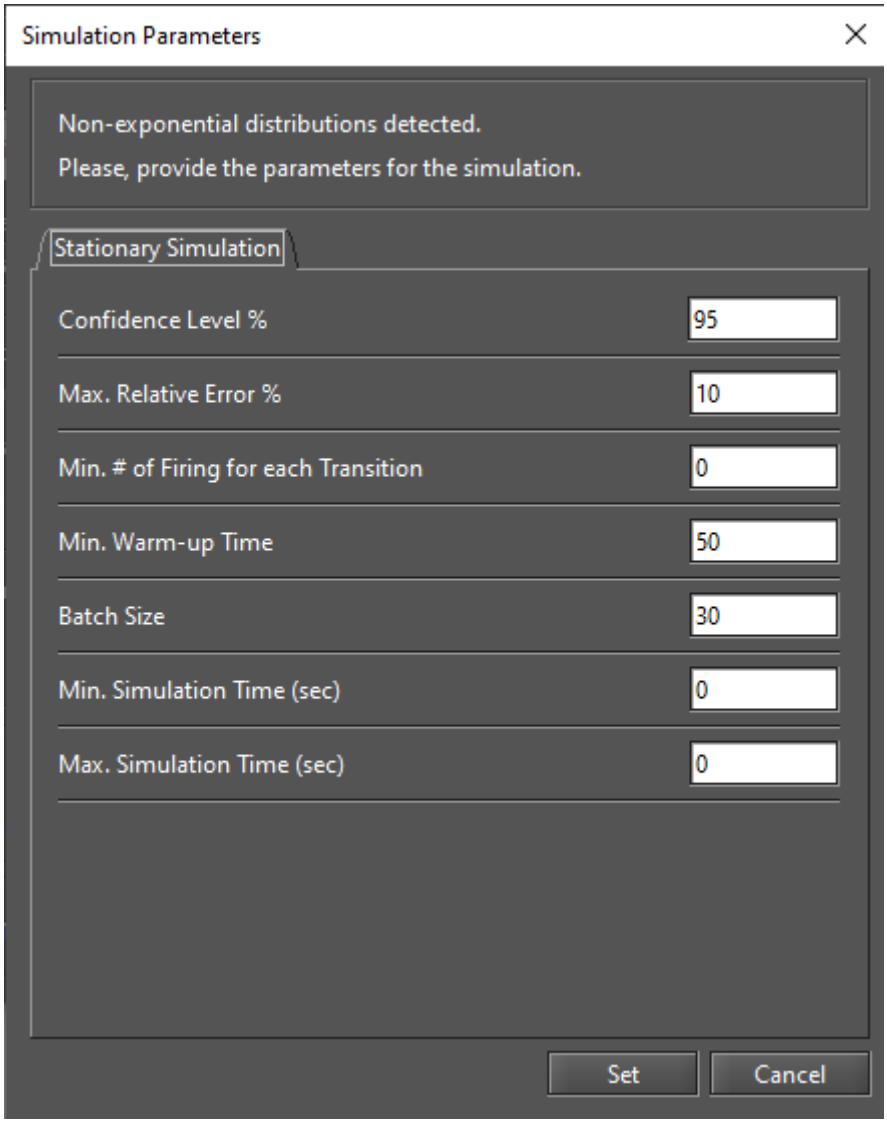


Figure 126: Simulation for Steady-State Metrics

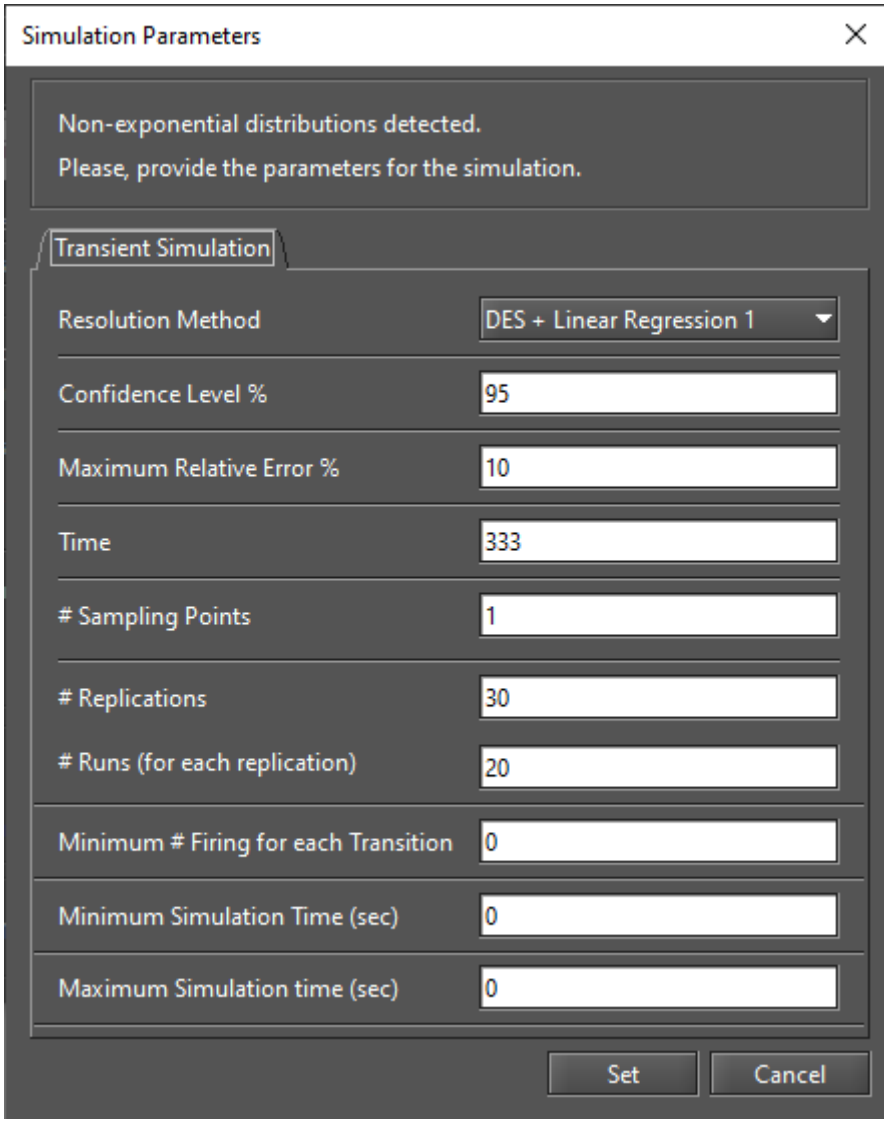


Figure 127: Simulation for Time-Dependent Metrics

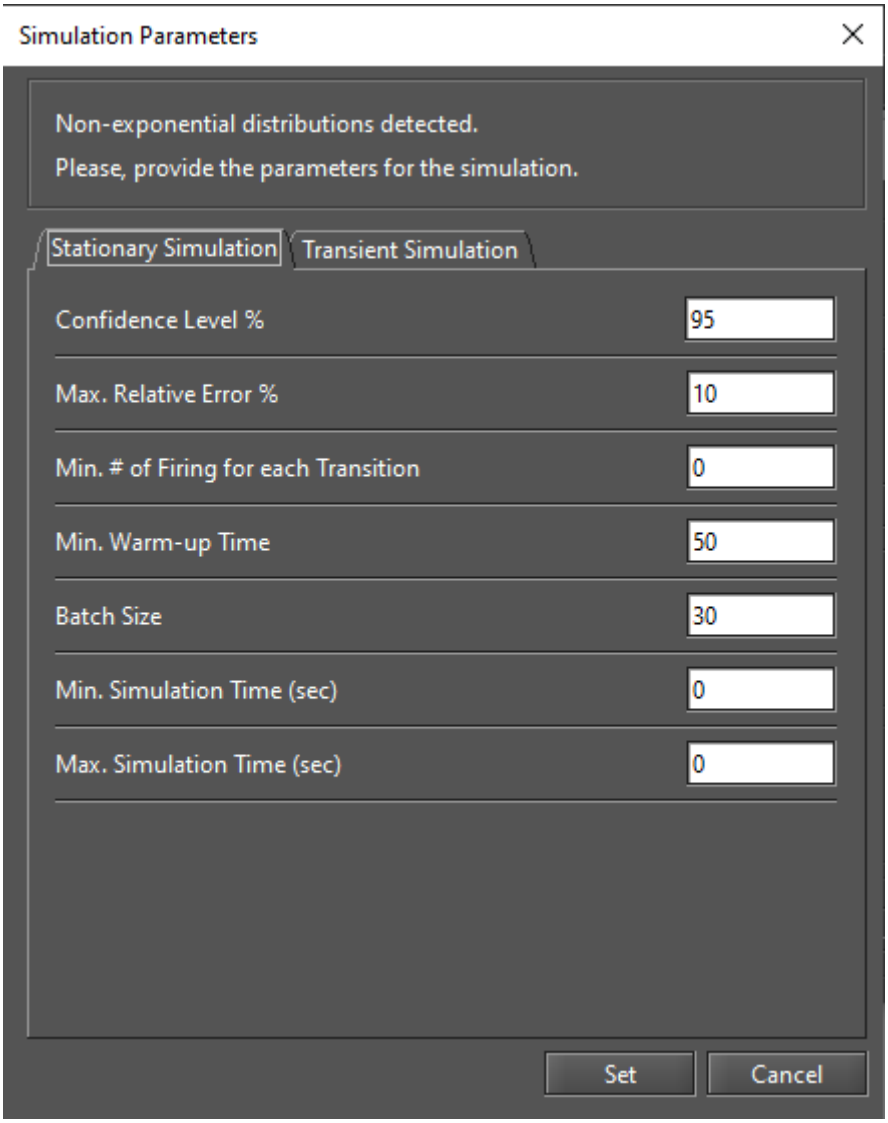


Figure 128: Simulation for Steady-State and Time-Dependent Metrics

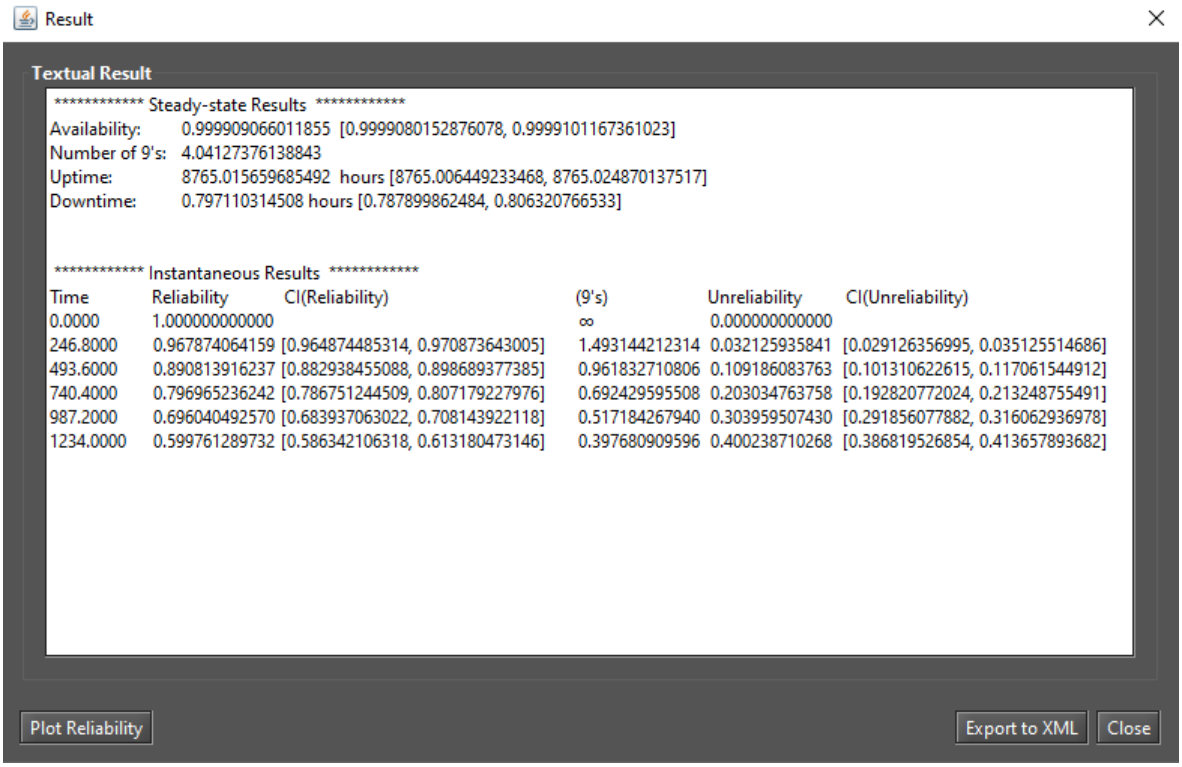


Figure 129: Result from a Non-Exponential RBD Model Evaluation with Confidence Intervals

3.2.2 RBD Experiment

Mercury allows us to evaluate the impact of varying some parameters on the model. Now we will show you how to use the experiment feature. The first step is to define one or more labels. Labels are variables that store numerical values and can be associated with the failure/repair parameters of blocks. The value of a label is changed taking into account a step size 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 shown in Figure 130. Another way is to right-click on the label area in the left pane and choose “Insert label.”. Once this is done, the “Label Properties” window is displayed (see Figure 131). There the user can set the properties of the label.

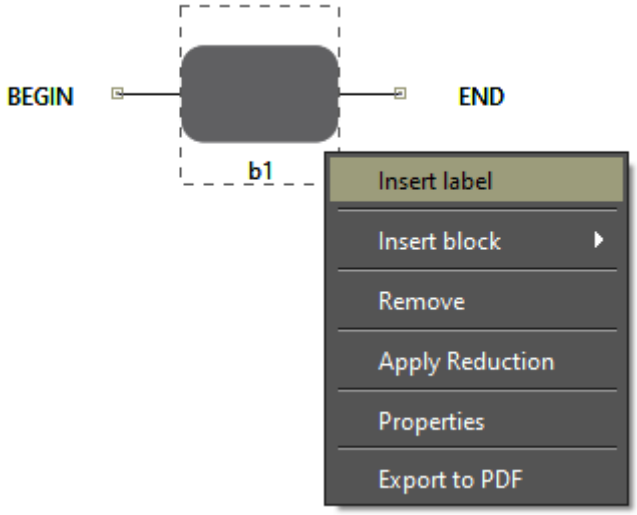


Figure 130: Inserting a Label into the RBD Model

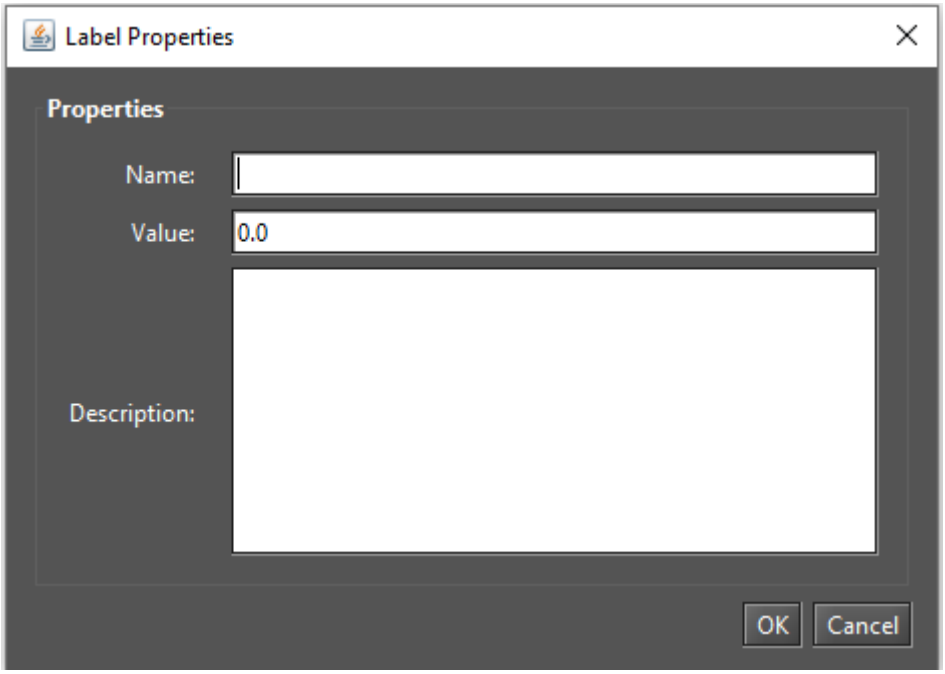


Figure 131: Properties of an RBD label

Once a label is inserted, it is available in the left window on the RBD tab. Now it can be linked to one or more block parameters (see Figure 132).

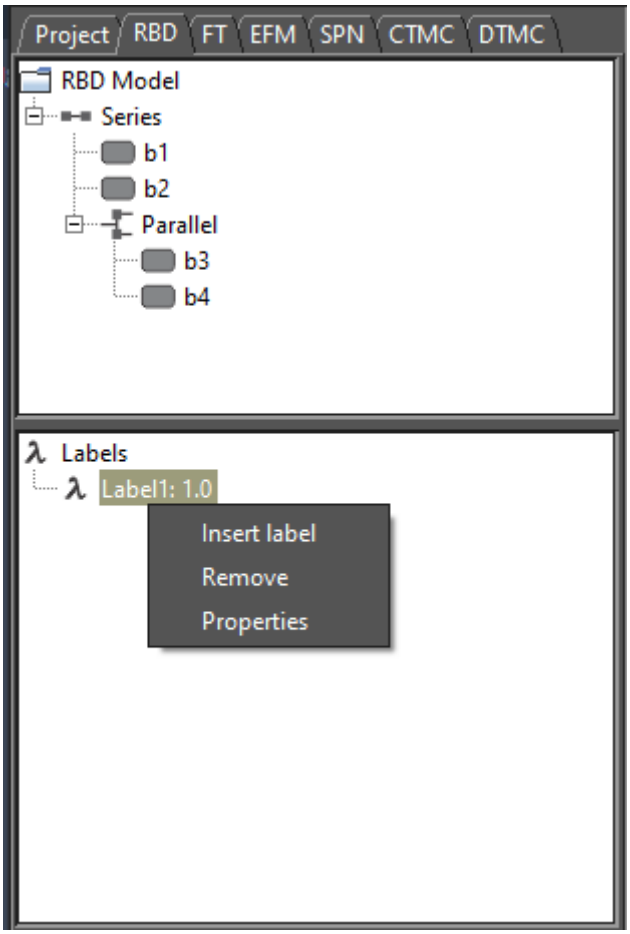


Figure 132: Left-Side RBD Panel

When you right-click on a label, a popup menu with three menu items appears (see Figure 132). We describe each of these items below.

- **Insert label.** Displays the “Label Properties” window where the user can insert a label.
- **Remove label.** Remove the selected label.
- **Properties.** Displays the “Label Properties” window where the user can 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 133 demonstrates how to attach a label to a block parameter. It is also possible to attach a label to the price parameter of a component in the RBD model, as shown in Figure 134.

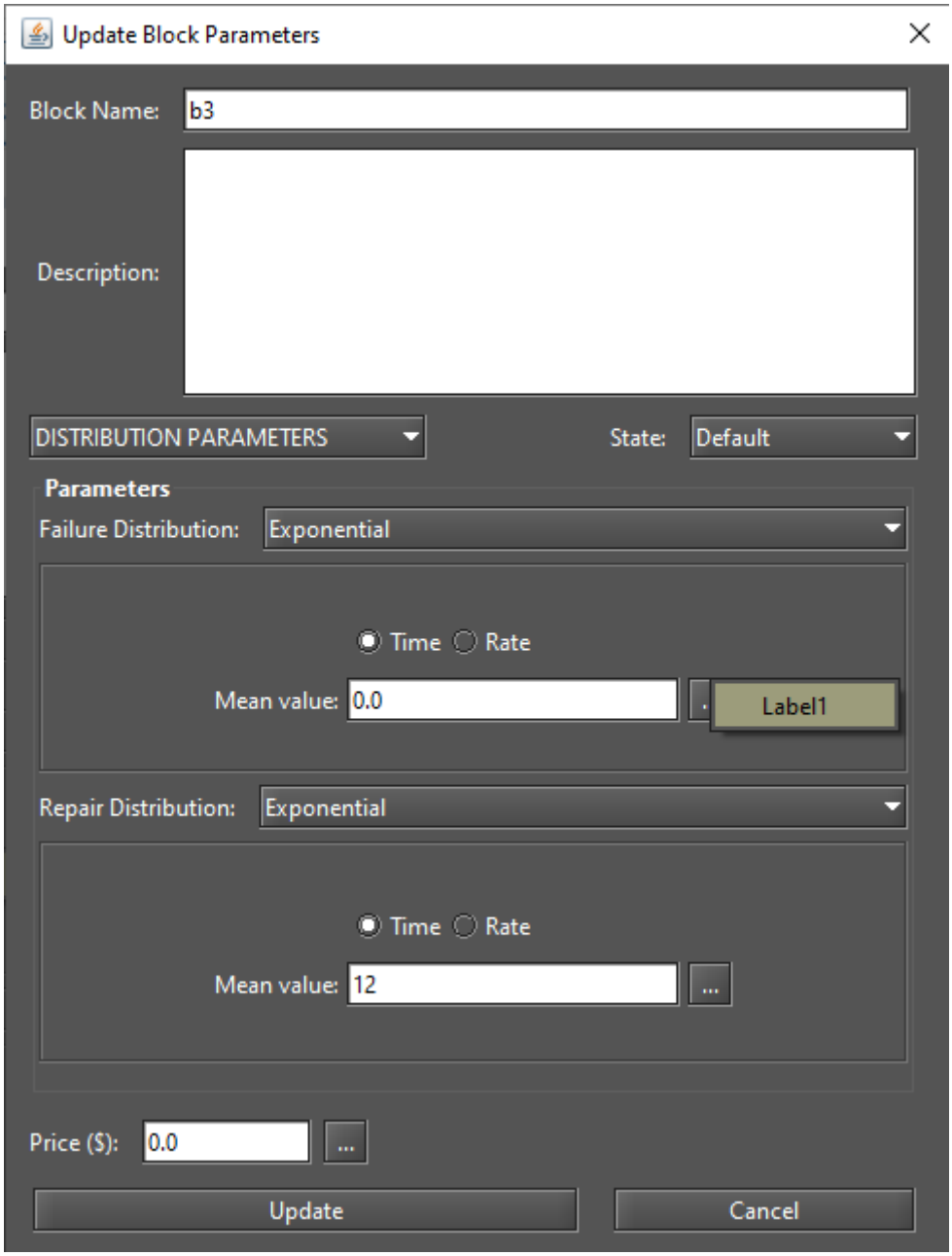


Figure 133: Attaching a Label to a Block Parameter (Failure Distribution - Mean value)

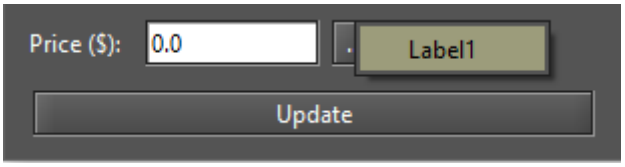


Figure 134: Attaching a Label to the Price Parameter of a Block

Experiment of RBDs can be accessed from the menu *Evaluate -> RBD Evaluation -> Experiment*. Figure 135 shows the "Experiment" window. To run experiments, the user must enter values for all required fields.

We describe each of these options below.

- **Parameter.** The label whose value will be changed at each iteration of the experiment.

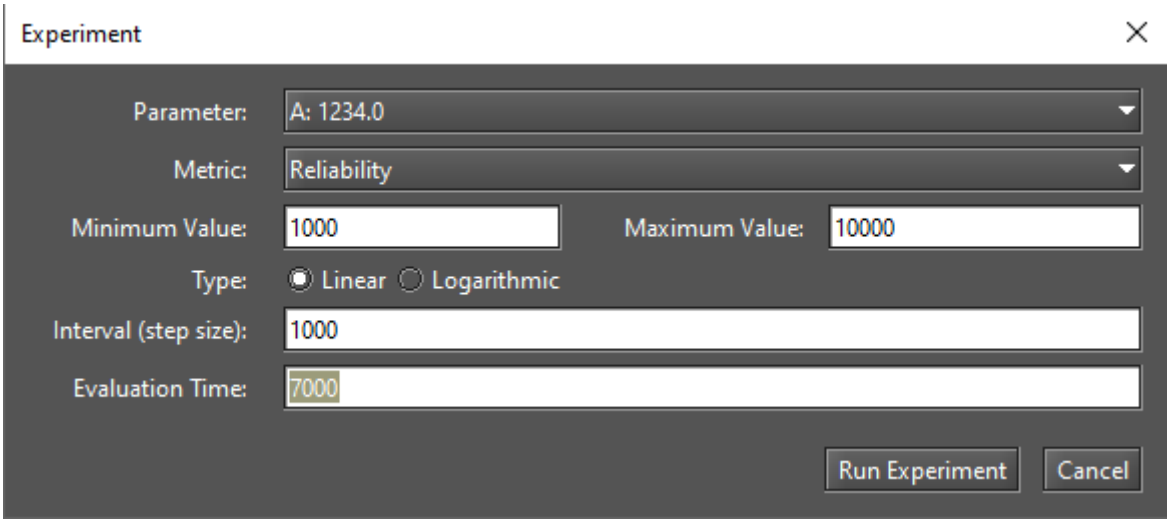


Figure 135: RBD Experiment

- **Metric.** The metric to be evaluated.
- **Minimum Value.** Initial value for the selected label.
- **Maximum Value.** Final value for the selected label.
- **Type.** Determines whether the value of the parameter is changed linearly or logarithmically. If it is logarithmic, the parameter value is changed by a base-10 logarithmic function, taking into account the minimum and maximum values.
- **Interval.** Step size that will be taken into account when changing the value of the label. The label starts with the minimum value and its value is incremented considering this interval. At each change, the selected metric is evaluated. The experiment is finished when the maximum value for the label is reached.
- **Evaluation Time.** Evaluation time considered in the calculation of time-dependent metrics. For time-dependent metrics — reliability, unreliability, instantaneous availability — it is necessary to enter the time parameter.

After defining the input parameters, the user must click on the “Run Experiment” button to start the experiment. If the model to be evaluated contains non-exponential blocks, the user must also enter the simulation parameters, as shown in Figure 136. Once the experiment is finished, the “Experiment Result” dialog is displayed (see Figure 137).

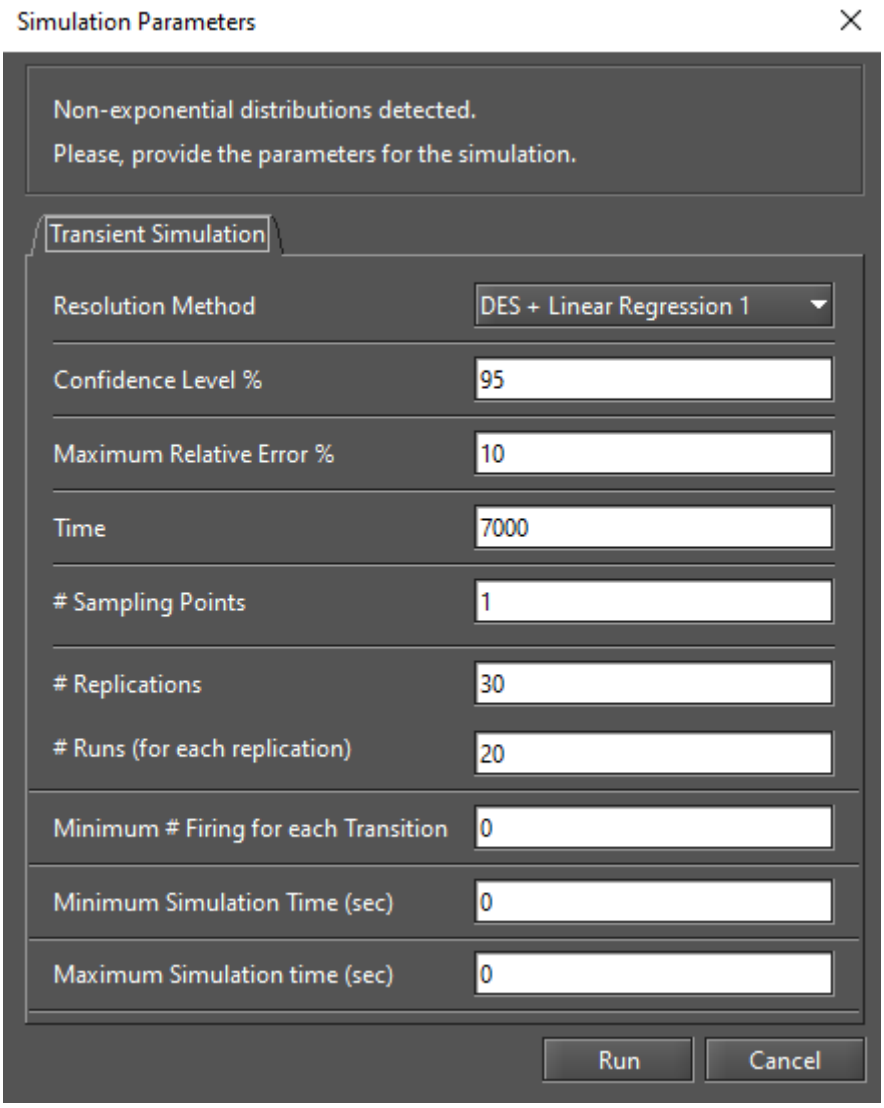


Figure 136: RBD Experiment - Simulation Parameters

Figure 137: Experiment Result Dialog

You can initiate a new experiment by clicking the button labeled “New Experiment”. Assume that variable “A” is a base-10 logarithmic variable, ranging from 1 to 4, as shown in Figure 138. Then, Mercury calculates the metric and displays the result on a logarithmic scale, as shown in Figure 139.

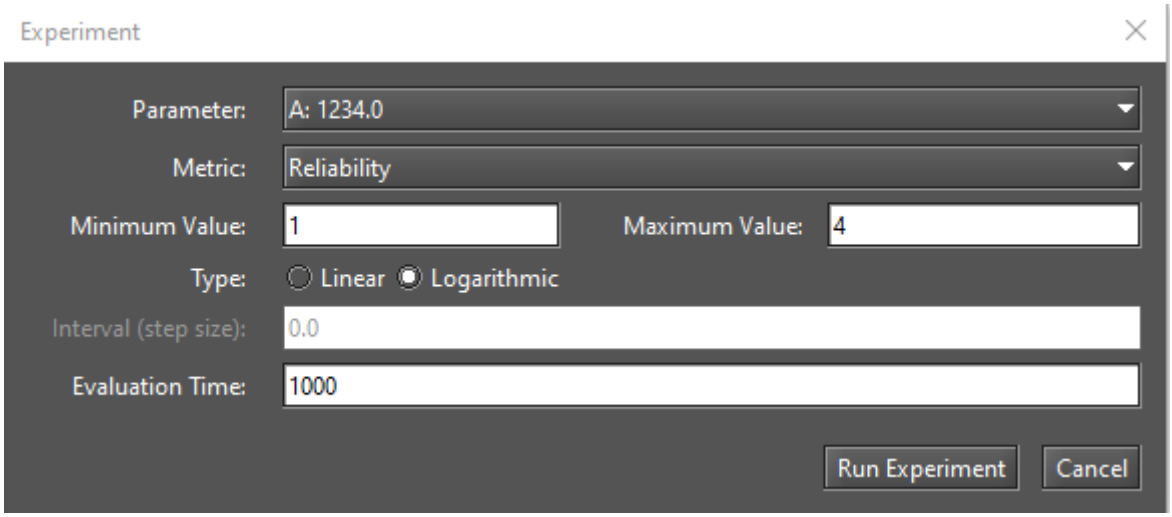


Figure 138: RBD Experiment - Logarithmic Variable

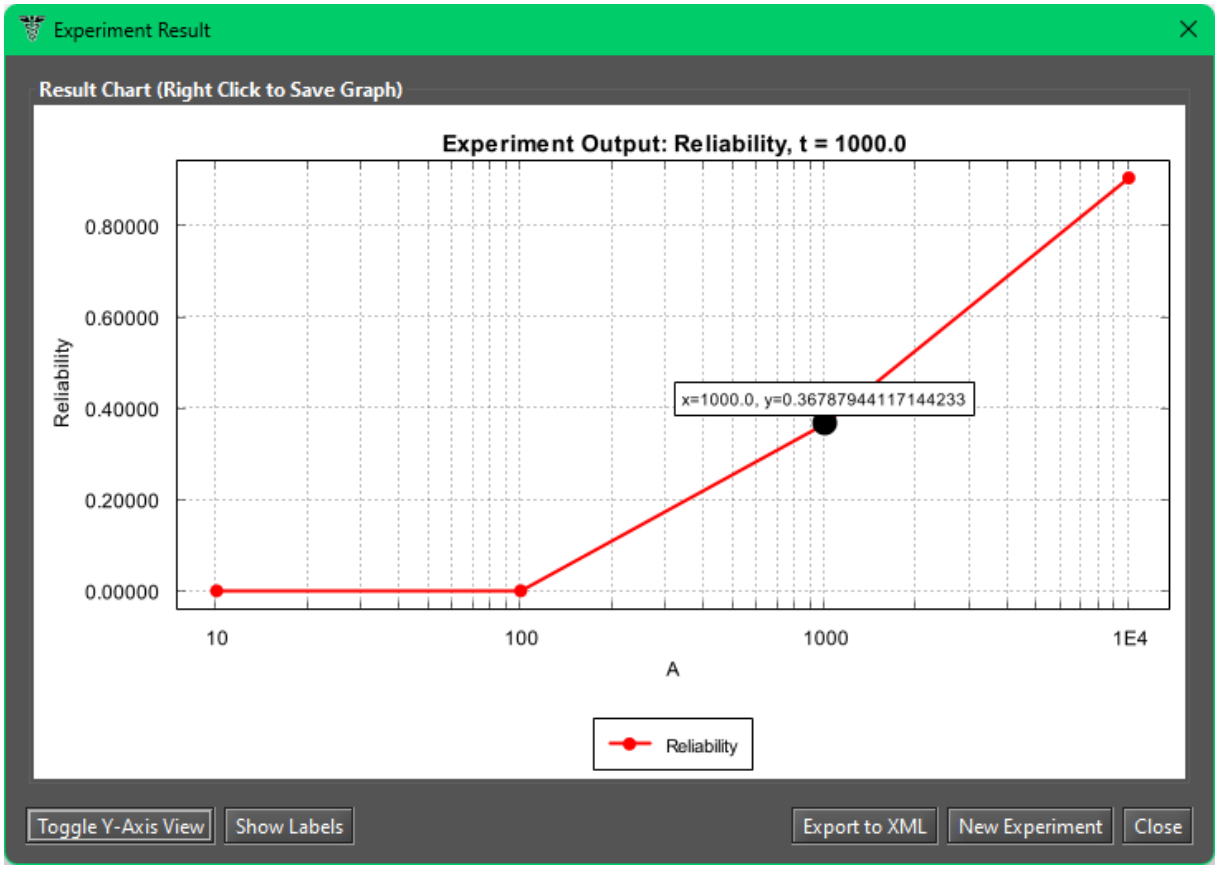


Figure 139: Experiment Result Dialog - Logarithmic Scale

3.2.3 Bounds for Dependability Analysis

Bounds for Dependability Analysis is used to estimate dependability metrics by calculating reliability, availability, or downtime. It is necessary to estimate the bounds (upper and lower limits) for the calculation of this analysis, where users get results quickly. This analysis should be performed when the model is huge. This analysis is divided into two parts: (i) the calculation of the limits and (ii) the use of the sum of disjoint points to determine the successive values and the number of iterations required.

Users can access this analysis by going to the *Evaluate -> RBD Evaluation -> Bounds Evaluation* menu. Figure 140 shows the “Bounds for Dependability Analysis” window. As shown in Figure 141, four metrics can be evaluated: Steady-State Availability, Instantaneous Availability, Reliability, and Downtime. The “time” parameter is required if you select “Instantaneous Availability” metric. Once you have selected a metric and entered the time, if applicable, you must click the “Get Start Values” button to start the evaluation.

First, the upper and lower values are calculated. The first path and the first cut are used to determine the upper and lower values of the selected metric. The paths refer to the lower bounds, where a minimal set of components is chosen to ensure the operational mode of the system. The cuts refer to the upper bounds where a minimal set of components is chosen to ensure the system in failure mode. After getting the upper and lower values, you can set the number of steps for the upper and lower values and click the ”Run” button (see Figure 142). Then you can see the result as shown in Figure 143. The user can plot a chart and export the result to a MS Excel

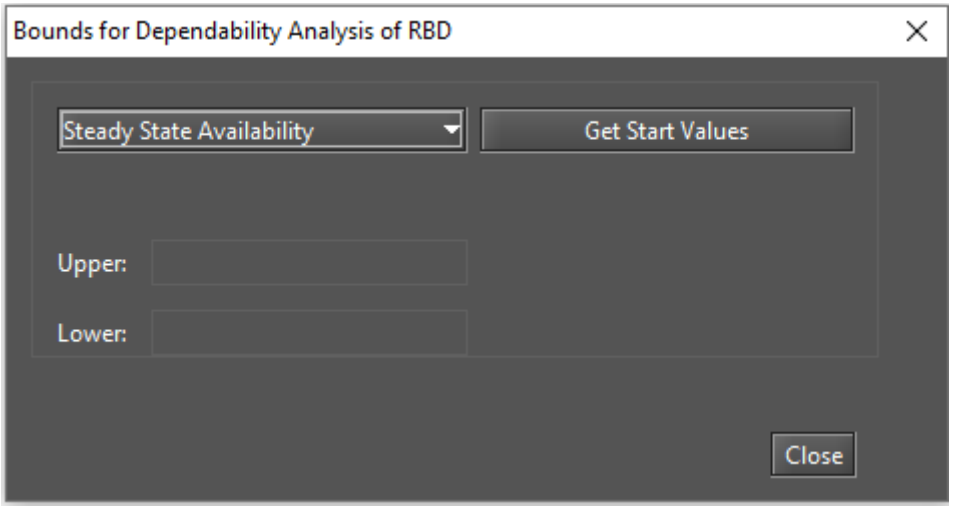


Figure 140: Bounds for Dependability Analysis

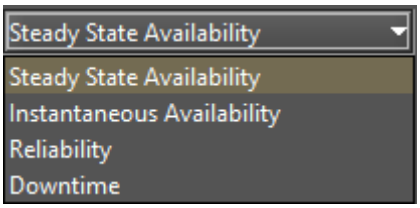


Figure 141: Metrics for Bounds Evaluation

file.

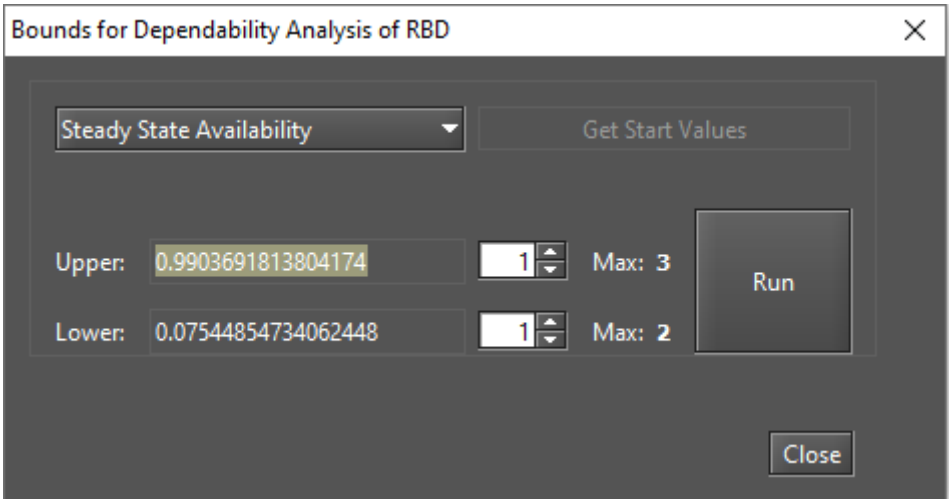


Figure 142: Bounds for Steady-State Availability

The method of determining successive values and the number of iterations is defined by the number of paths and cuts in the model. If you increase the number of iterations, the value found will be closer to the exact value. Once the calculation of the last path or cut is complete, the exact value of the metric can be found.

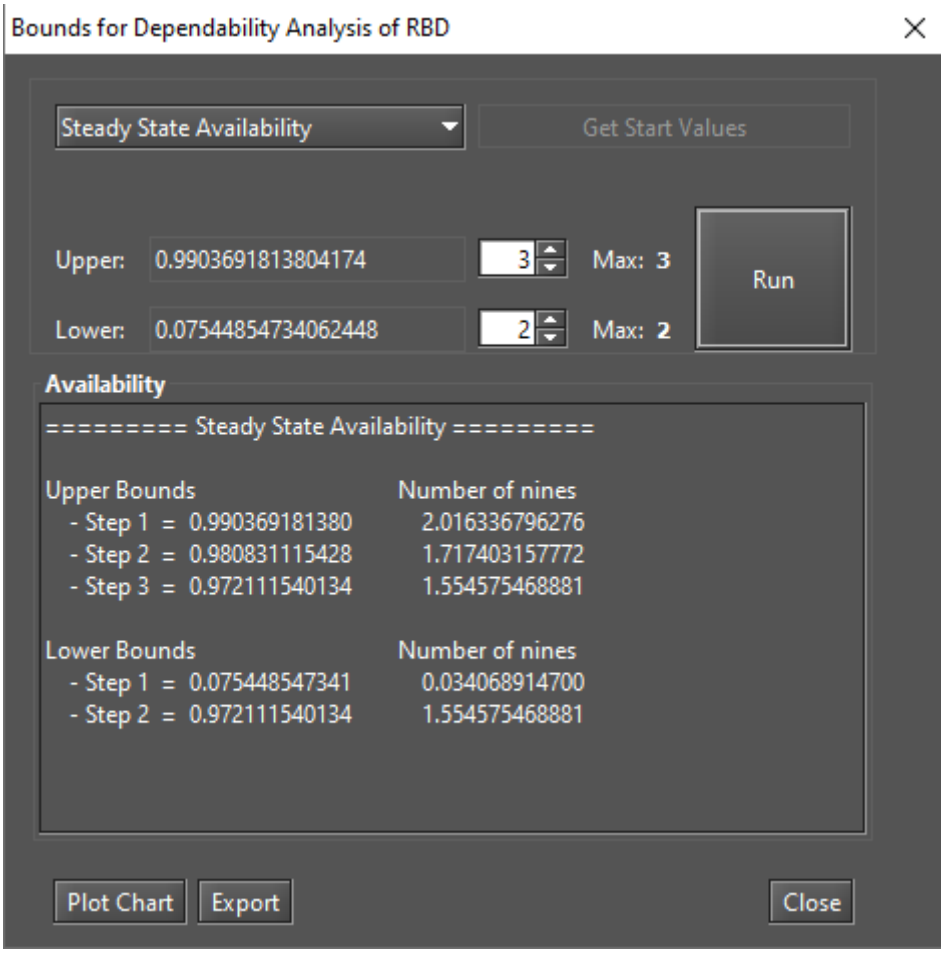


Figure 143: Bounds for Steady-State Availability - Steps Result

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. Considering the importance scores, the most important component (i.e., the component with the highest importance) should be improved to increase the reliability or availability of the system. This evaluation can be used, for example, to support maintenance activities.

You can use importance measures to determine the relative importance of each component with respect to the reliability or availability of the overall system. You can access this evaluation by selecting "Importance Measures" from the *Evaluate -> RBD Evaluation* menu. Then you need to select a metric in the "Component Importance Measures" window and then click the "Evaluate" button (see Figure 144). If the "Cost" parameter has been set for the blocks, it is also possible to evaluate the relationship between metrics and investment costs. The parameter "Time" is needed for the evaluation of the reliability metrics.

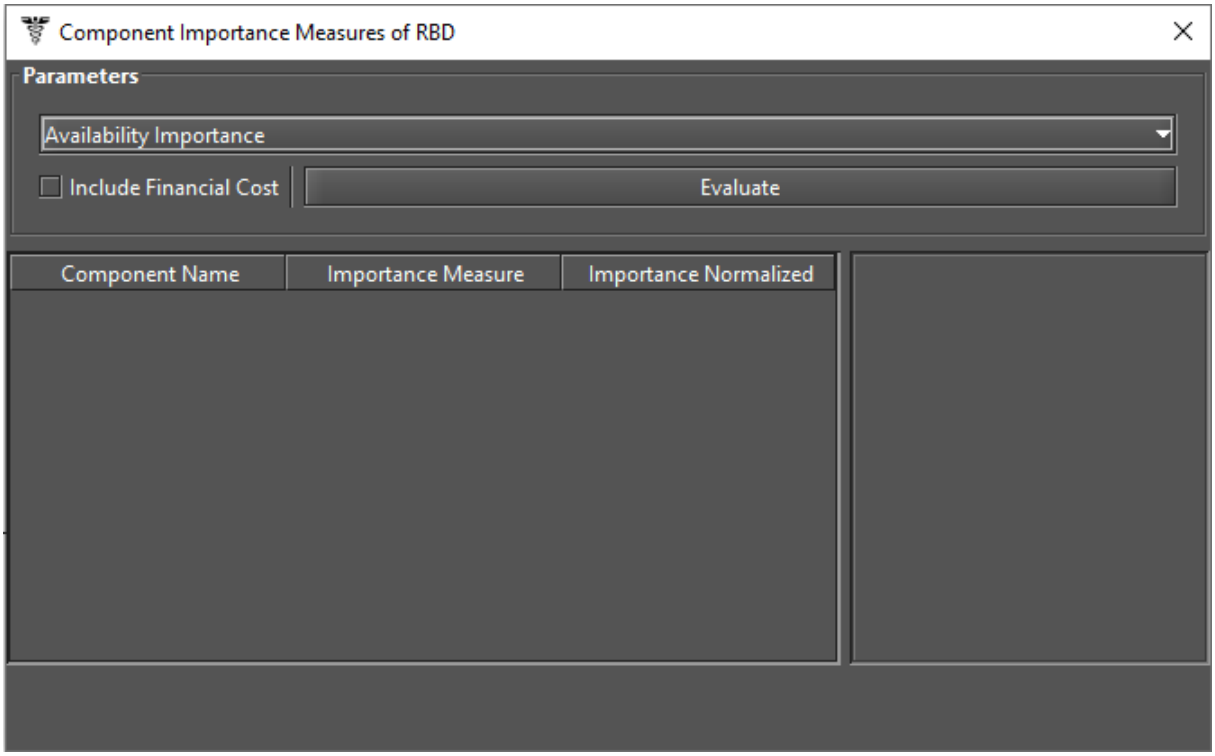


Figure 144: Component Importance Measures

As shown in Figure 145, you can choose between a few types of measures. The types are: “Availability Importance", “Reliability Importance (Birnbaum)", “Criticality Reliability Importance" and “Criticality Availability Importance." The “Criticality Importances” measures are obtained by considering the system in failure “(f)” or in operation.

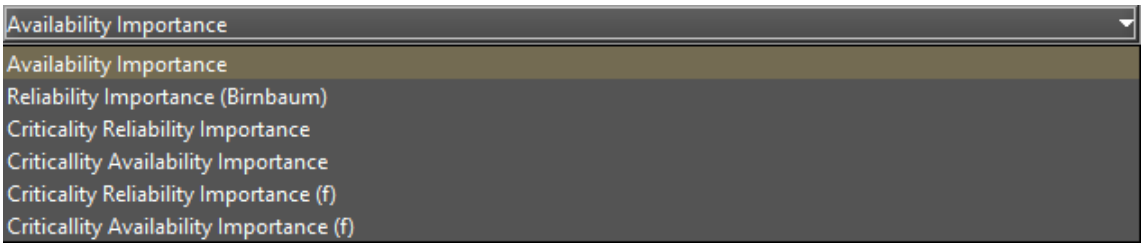


Figure 145: Metrics for Component Importance Measures

The results of such assessments are shown in Figure 146. The results show the importance score for each component and a graphical view as a ranked list highlighting the most important components in the analysis.

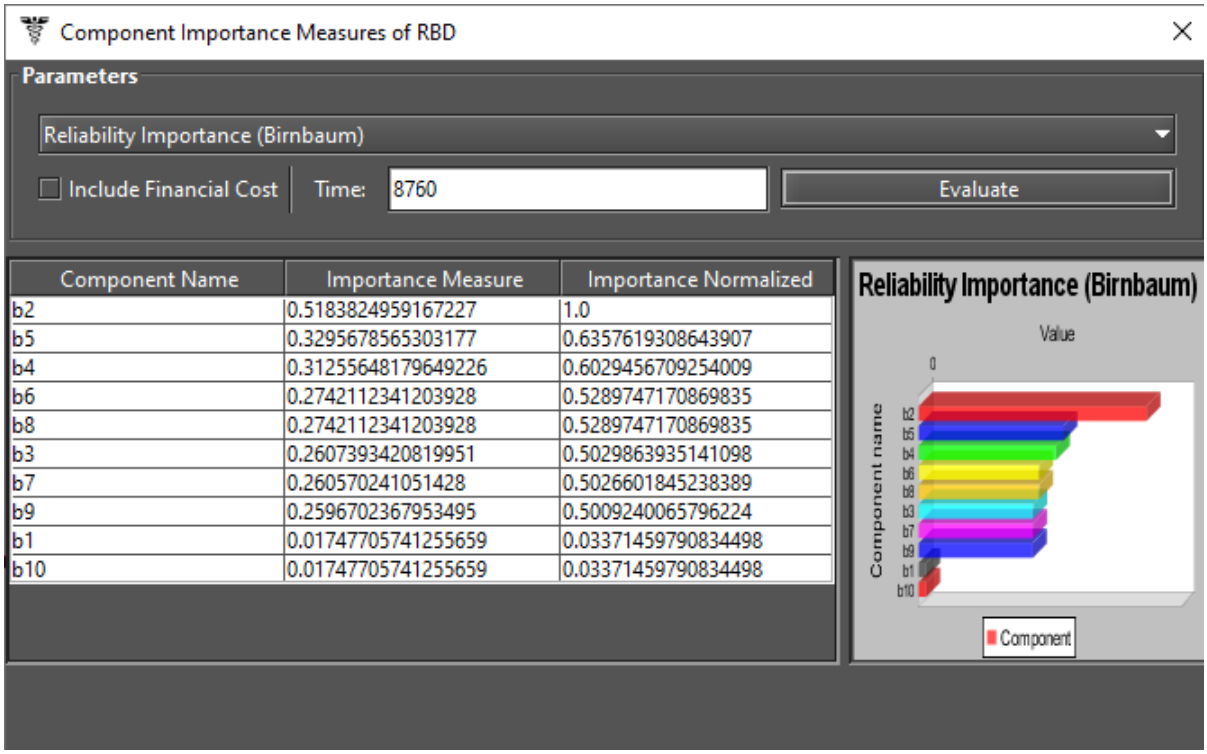


Figure 146: Result from Reliability Importance (Birnbaum) Evaluation

3.2.5 Structural and Logical Functions

Mercury generates structural and logical functions of RBD models. Both functions represent the system and refer to the states of the individual components. Also, it is possible to evaluate the impact on the system operation considering the faulty components. 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 block's properties (see Figure 147). If the state of a block is failed, the component is represented by a fire icon above the block, as mentioned earlier in this manual.

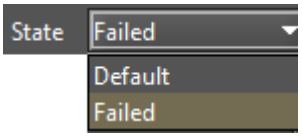


Figure 147: State of a RBD block

Let us now demonstrate how to obtain these functions using Mercury. Figure 148 shows an RBD model with blocks in series and parallel. As we can see, there is one failed block (block b5) in this model.

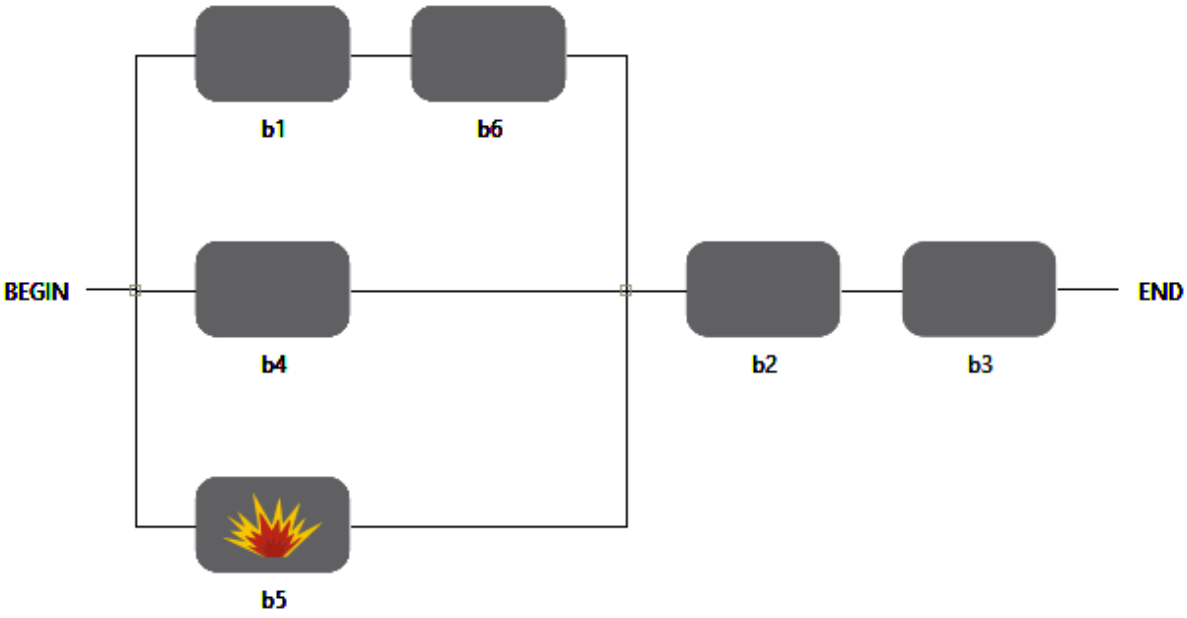


Figure 148: RBD Model for Structural and Logic Function Computation

Structural and logic functions can be accessed from the *Evaluate -> RBD Evaluation -> Get Functions* menu. Figures 149 and 150 show the structural and logic functions, respectively, of the RBD model shown above. In addition to the expressions, the tool shows the blocks marked as faulty (non-functional) and the current state of the system. In our example, the faulty block (b5) has no effect on the operating state of the system.

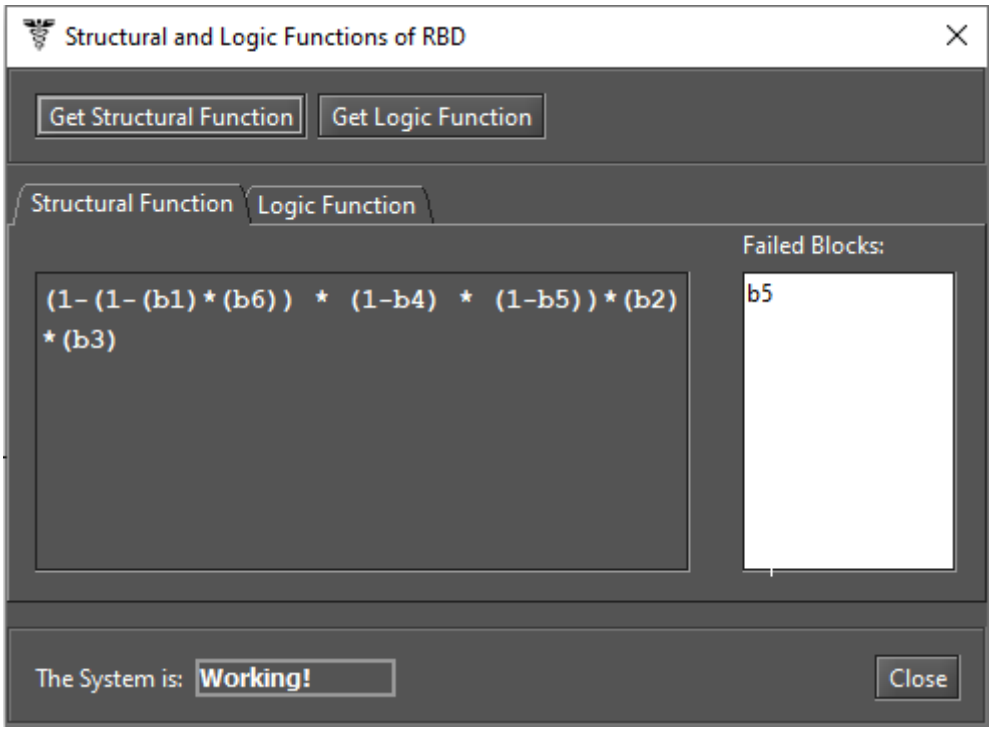


Figure 149: Structural Function

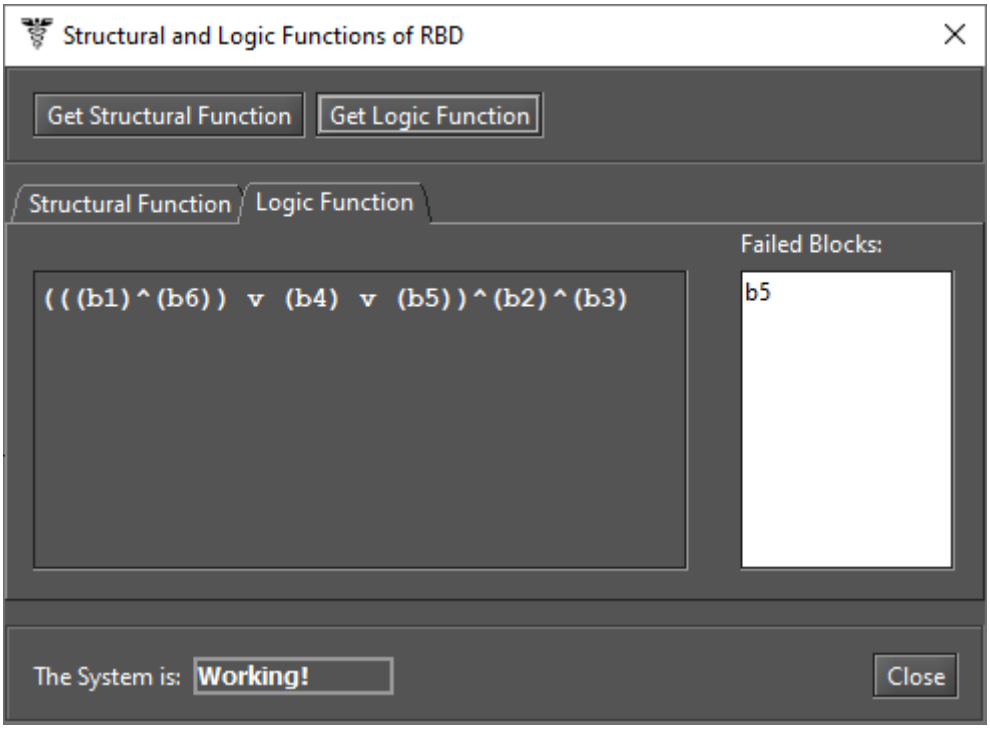


Figure 150: Logic Function

3.2.6 Sensitivity Analysis

Mercury calculates partial derivative sensitivity indices from RBDs through sensitivity analysis. These indices indicate the impact of each input parameter on the availability of the model. Mercury provides two types of sensitivity analysis for RBD models. The first type of sensitivity analysis considers the current values of the model's parameters and can be accessed from the *Evaluate -> RBD Evaluation -> Sensitivity Analysis* menu. The second type of analysis considers min/max values for each parameter and supports the "Design of Experiments" (DoE) method in addition to the "Sensitivity Indices" method. This second type of sensitivity analysis is shown in Section 2.5 and can be accessed from the menu *Evaluate -> RBD Evaluation -> Sensitivity Analysis (min/max values)*. Figure 151 shows the "Sensitivity Analysis" window to perform sensitivity analysis considering the current parameter values, displaying the partial derivative of the structural equation for each parameter and the sensitivity indices. It should be noted that both types of sensitivity analysis are only available when all event nodes of the model are exponential.

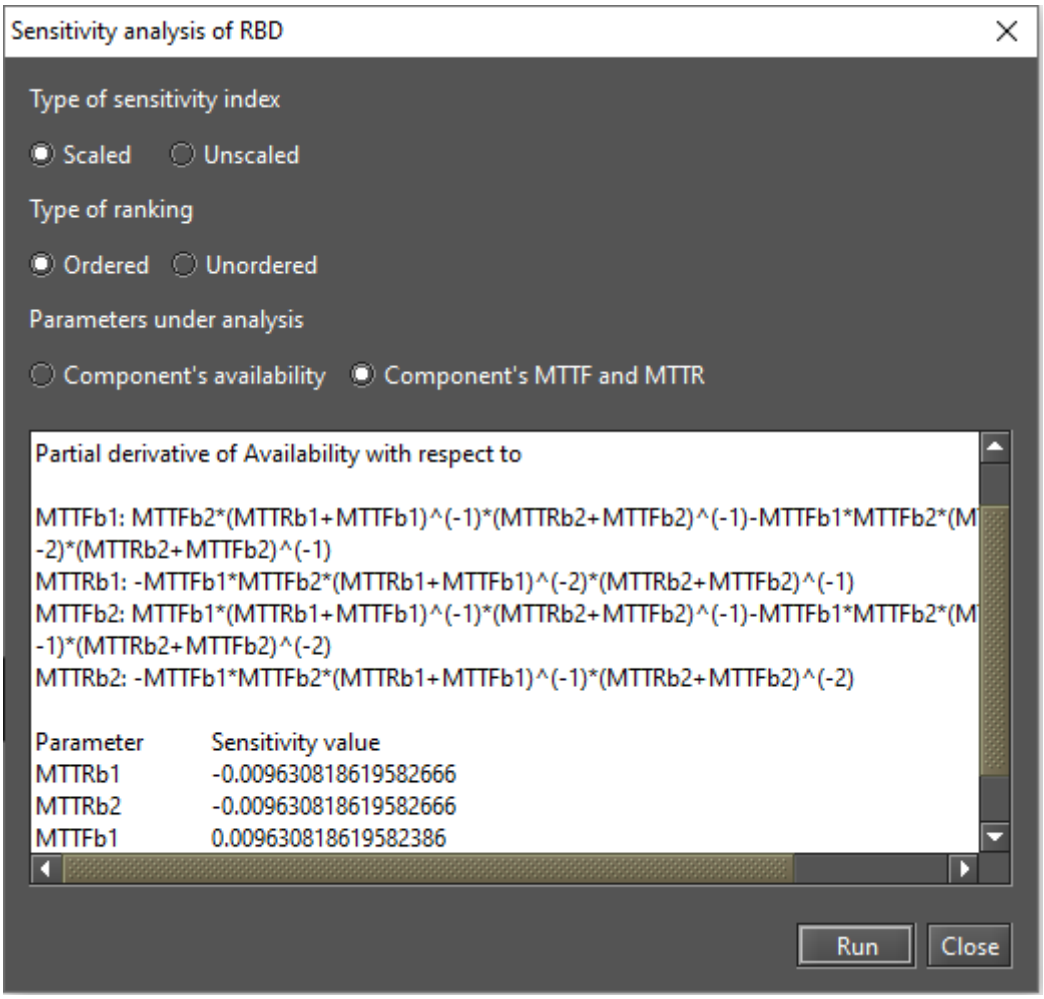


Figure 151: Sensitivity Analysis from an RBD Model

4 FT Modeling and Evaluation

Fault trees (FTs) and RBDs differ in their purpose. FT is a top-down logical diagram that allows you to create a visual representation of a system that shows the logical relationships between the associated events and causes that can lead to failure of the assessed system. When you create a project, the default model FT contains only a top-level event “FAILURE” referred to as “undefined," as shown in Figure 152. This means that no failure event leads to this top event. So from this point on, the user can define the model using components.

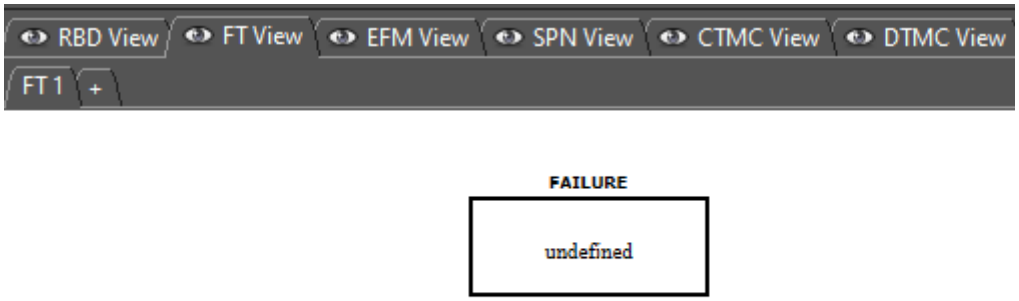


Figure 152: Default Fault Tree Model

With the Mercury tool, we can use two types of nodes: basic events and gates (logic ports). Basic events are represented as leaf nodes, as shown in Figure 153. On the other hand, each supported gate has its own graphical representation. As we can see in Figure 154, Mercury supports three types of gates: AND, OR, and K-out-of-N (KooN).

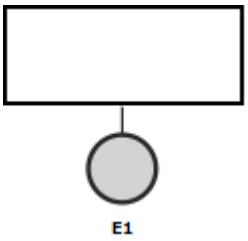


Figure 153: Basic Event

Figure 154: 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.

Unlike the other formalisms, the FT view does not provide a toolbar that allows the user to select components and make changes to the model. All operations to change the model are performed by selecting menu items with the mouse. For example, the user must right-click on the top event to create gates and event nodes. Changes to the model are made by selecting the appropriate action on the respective menu item. Among the available options, the user will find the basic operations: insert, edit (properties) and remove.

To create the first gate or single fault event, the user must right-click on the **FAILURE** event. In this popup menu there are only two menu items: "Add Gate" and "Add Single Event". The "Add Single Event" menu adds only a single fault event to the fault tree, as shown in Figure 155. Figure 156 shows how to add a gate to the top event. From there, you can choose between three different types of gates: *AND* , *OR* , and *KooN*.

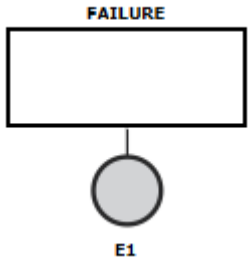


Figure 155: Fault Tree with a Single Event

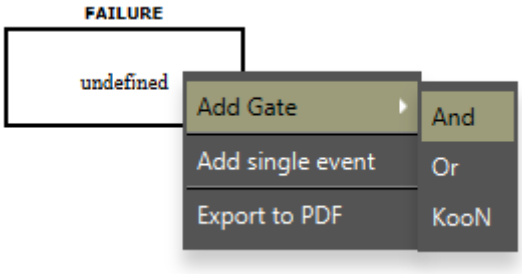


Figure 156: Adding the First Gate into the FT Model

Once you have selected the type of gate, the "Add Gate" dialog box appears (see Figure 157). The fields in this dialog box are described below.

- **ID.** ID for the gate to be inserted into 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 a ID, which uniquely identifies it.
- **Gate Type.** Type of the gate to be inserted. You can change the gate type by clicking the dropdown button.
- **K Value.** When you insert a KooN gate, this field is activated. 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. For this type of gate, the user should specify the minimum number of components (K) that must fail for the group of components to fail. Figure 158 shows how a KooN gate is represented. As we can see, the values of the parameters K and N are shown next to the gate ID in the diagram. In the current version of

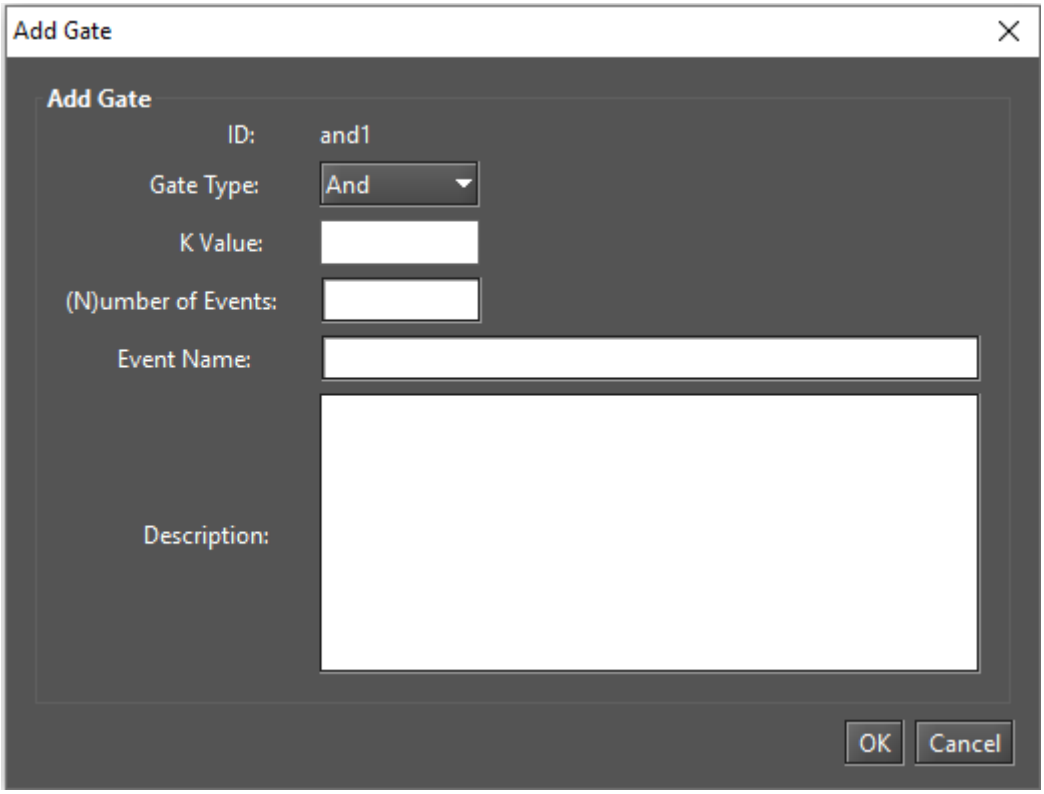


Figure 157: Add Gate Dialog

Mercury, it is not possible to add child gates to a KooN gate. The KooN gate can only have one basic child event representing the set of components. We intend to overcome this limitation as soon as possible.

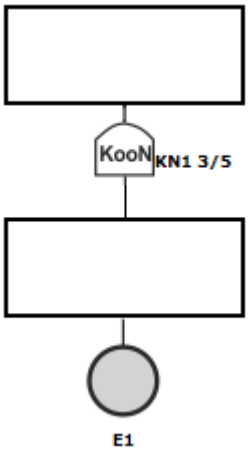


Figure 158: KooN Gate

- **(N)umber of Events.** When you add a gate, you must specify the number of basic events to be added as child nodes in the gate. Each gate must have at least two nodes. Once inserted, a basic event can be replaced by a gate. Thus, it is possible to define a FT model with a large number of levels and components.
- **Event Name.** Name of the component/event. It is displayed in the rectangular area above the node. The name can be changed at any time.

- **Description.** A description is additional information about the node or the component or subsystem it represents. It is intended to improve understanding of the model and has no semantic value in evaluating the model. It is simple text attached to the node.

After you enter the required fields and confirm by clicking the OK button, Mercury inserts the nodes and updates the FT diagram. Figure 159 shows a AND gate with two basic events.

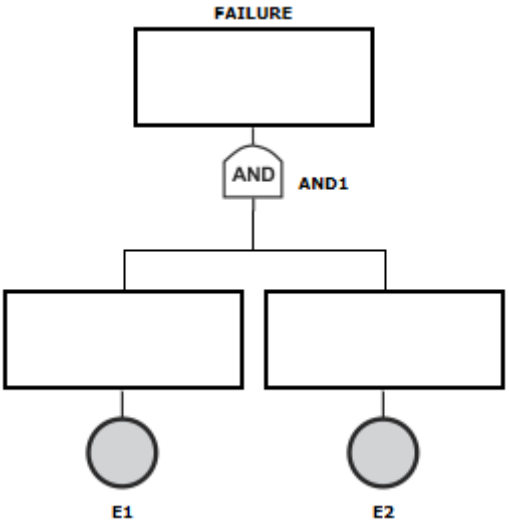


Figure 159: AND Gate with Two Basic Events

Double-clicking on a gate opens the “Gate Properties” dialog (see Figure 160).

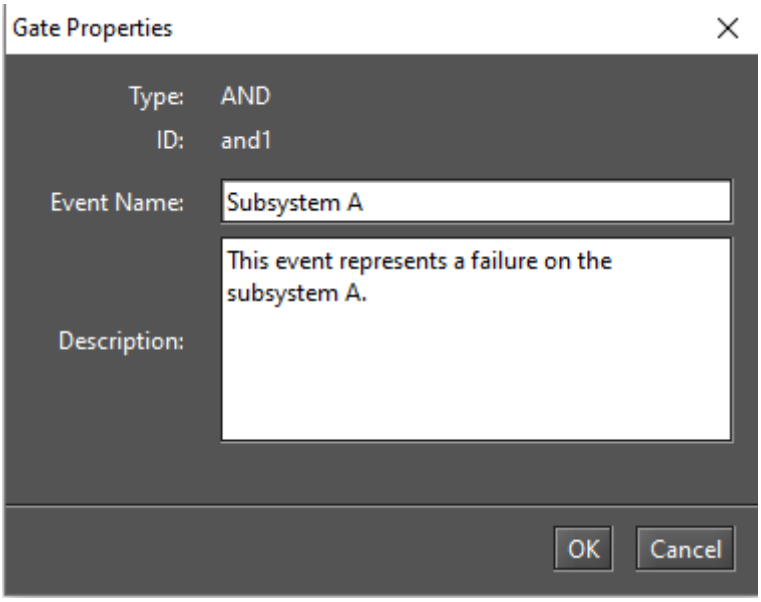


Figure 160: Gate Properties Dialog

If you change the name and description, the model will be updated accordingly. Figure 161 shows the updated model.

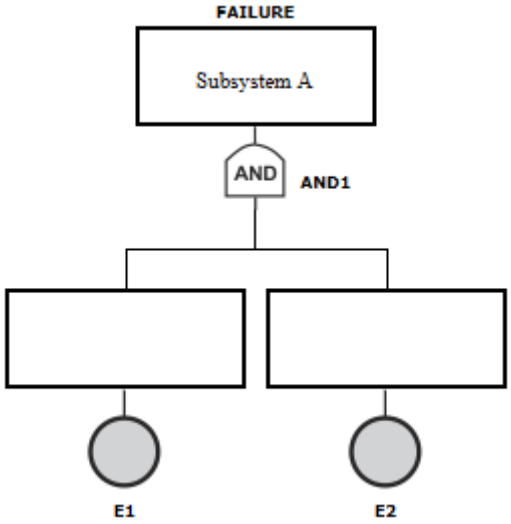


Figure 161: Description Updated

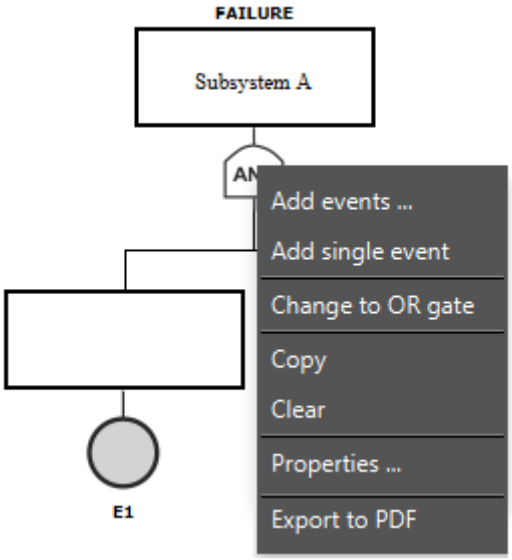


Figure 162: Gate Menu

Another way to edit the properties of a gate is to right-click on it and select “Properties...” from the menu that appears. As we can see in Figure 162, there are some options available in this menu.

Below we describe the options available in this popup menu.

- **Add events.** When you select it, a dialog appears (see Figure 163) where you must enter the number of events to be inserted into the selected gate. Then you have to set the properties (failure/repair parameters) for each new basic event.

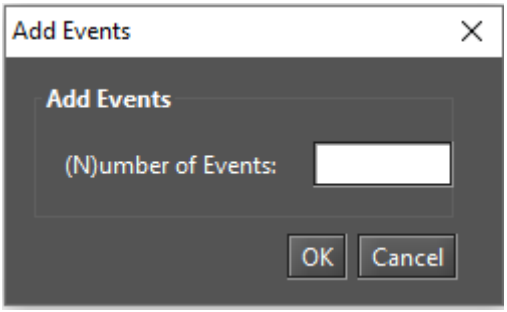


Figure 163: Add Events Dialog

- **Add single event.** Insert an empty event into the gate. In our example (see Figure 161), after choosing “Add single event” from the gate popup menu AND, a basic event is inserted into this gate, as highlighted in Figure 164.

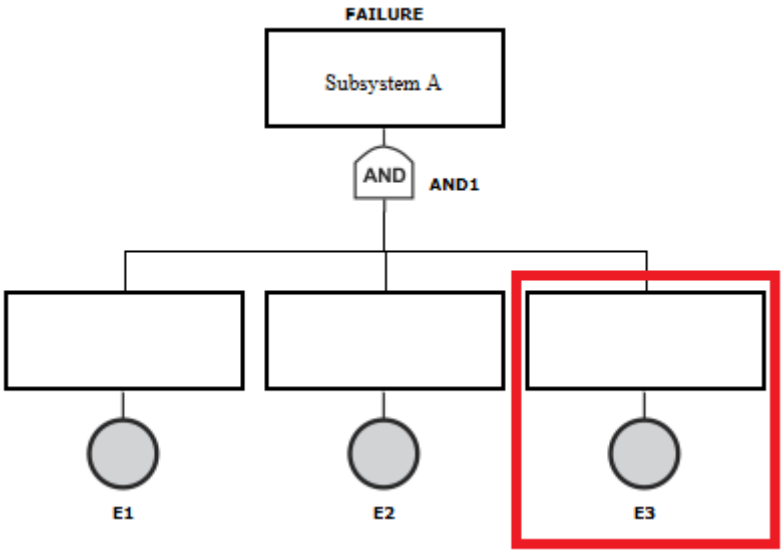


Figure 164: AND Gate with a New Event

- **Change to {OR, GATE} gate.** Change the type of the selected gate. If a port logic OR is selected, the only available option is to change it to a AND gate. The opposite is true for a AND gate. For example, if you look at our model, selecting this option will change the AND gate (see Figure 164) to a OR gate (see Figure 165).
- **Copy.** Copy the selected gate and all its child components to the clipboard.
- **Clear.** Empty the model. This option is only available when there is only one failure event in the model or if the gate selected is the top level gate.
- **Properties.** By selecting this option, the user can change the properties of the selected gate. Figure 160 shows the “Gate Properties” window for AND and OR gates. For a KooN gate, in addition to “Event Name" and “Description", you can also change the K and N parameters, as we can see in Figure 166.

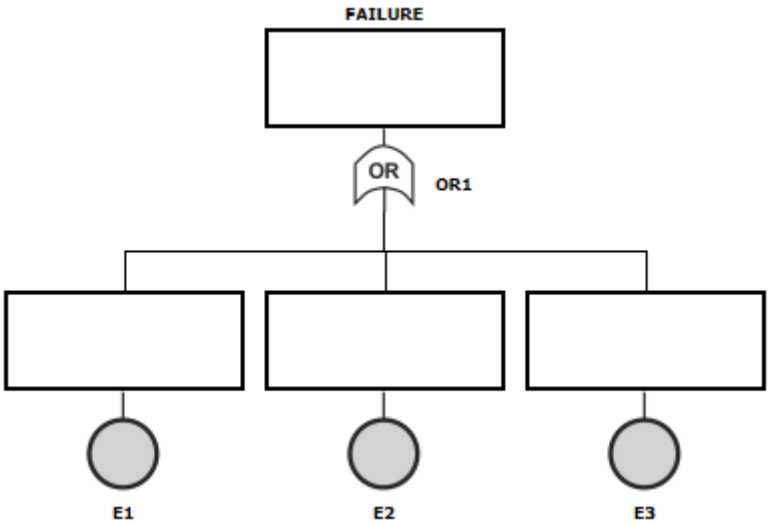


Figure 165: Changing the Type of the Gate

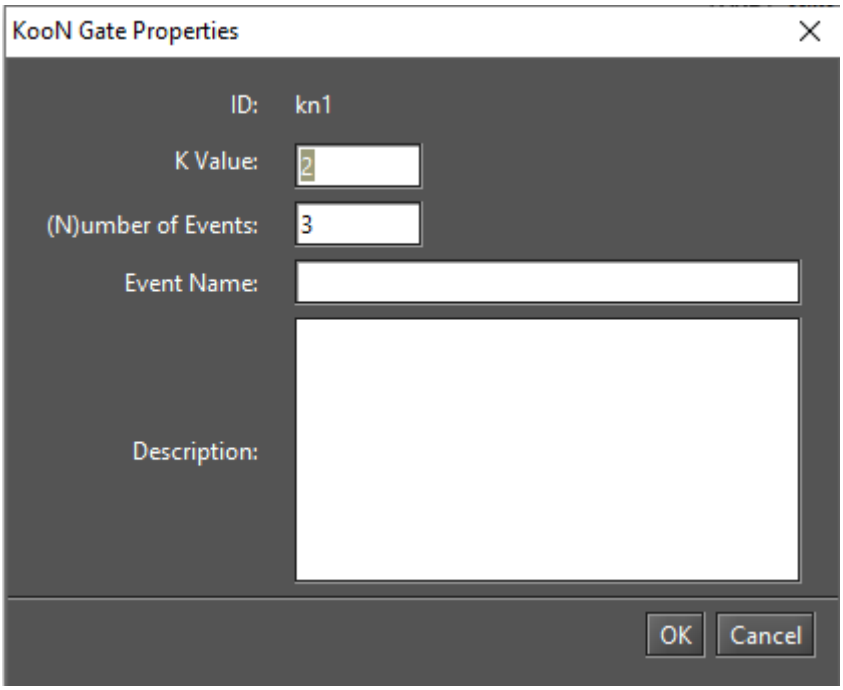


Figure 166: Properties of a KooN Gate

By right-clicking over a non-top gate, a slightly different popup menu is displayed, as we can see by looking at Figure 167.

Below we describe the options that we have not yet presented and that are available to 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 in the clipboard. This option is enabled only if components were copied to the clipboard.

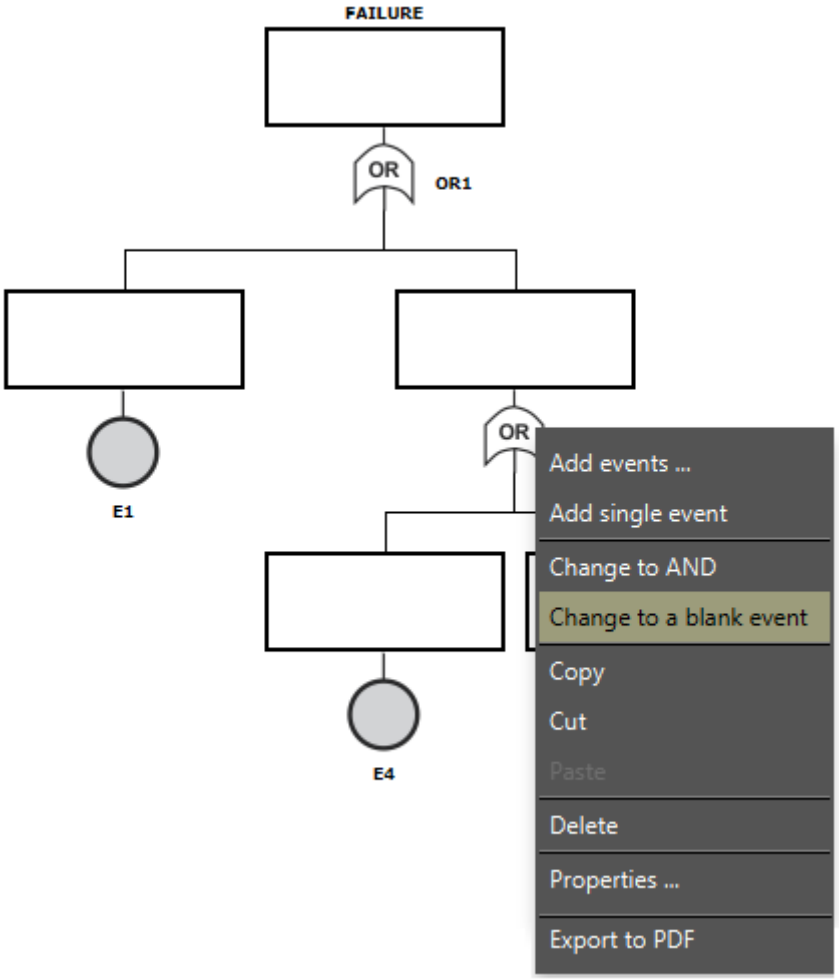


Figure 167: Popup Menu for Non-Top Level Gates

- **Delete.** Remove the selected gate and all its child nodes.

When you right-click on a basic event, a popup menu appears, as we can see in Figure 168. Below, we describe each menu item.

- **Change to a blank event.** Replace the selected node with an empty basic event.
- **Insert label.** Insert a label into the model. Labels are variables that store numerical values and can be associated with the failure/repair parameters of events.
- **Add gate.** Replace the selected basic event with a gate. After selecting the type of gate to replace the basic event from the submenu, the “Add Gate” dialog box appears (see Figure 157).
- **Copy.** Copy the selected event to clipboard.
- **Cut.** Cut the selected event to clipboard. This is only possible if the parent gate has at least three direct child nodes. Selecting this option will cause an error if there are only two direct nodes at the gate, as shown in Figure 169.

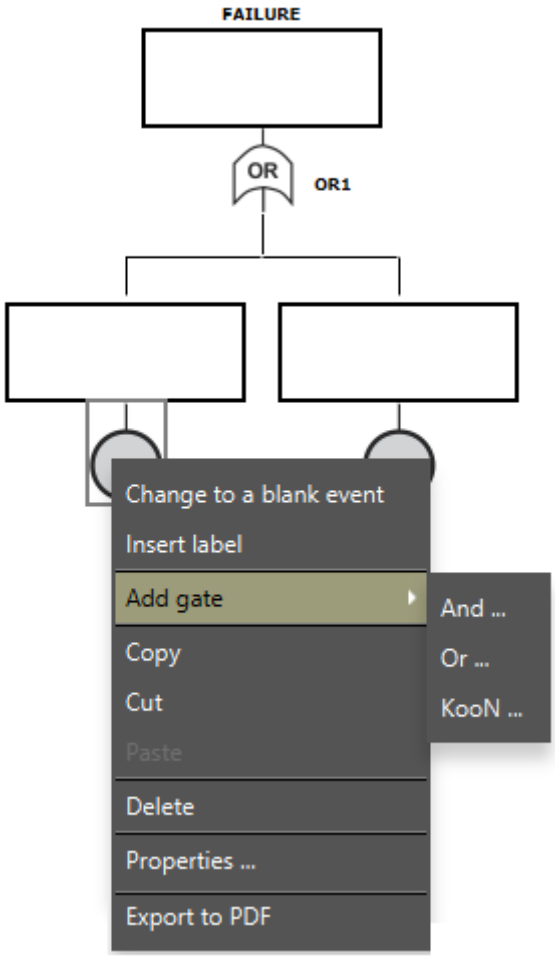


Figure 168: Popup Menu for Basic Events

- **Paste.** Replace the selected node with components from the clipboard. This function is enabled only if a component has been copied to the clipboard.
- **Delete.** Remove the selected node from the model. It is important to emphasize that each gate requires at least two direct nodes. Therefore, an error occurs if you try to remove a node when the model has only two nodes, as we can see in Figure 170.
- **Properties.** Display the dialog box for changing the properties of basic events (see Figure 171).
- **Export to PDF.** Export the FT model to a PDF file.

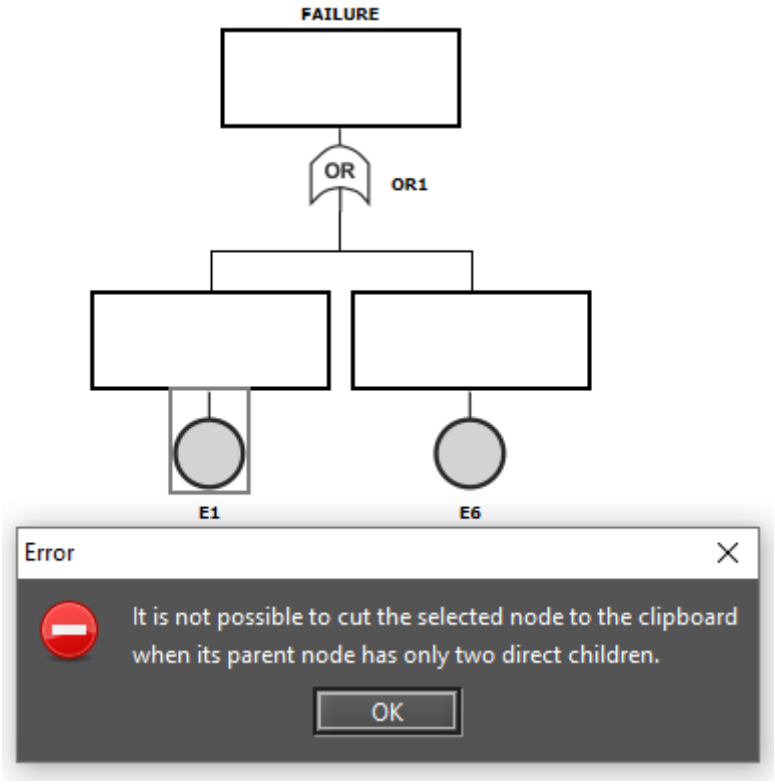


Figure 169: An Error Occurred While Trying to Cut a Node to Clipboard

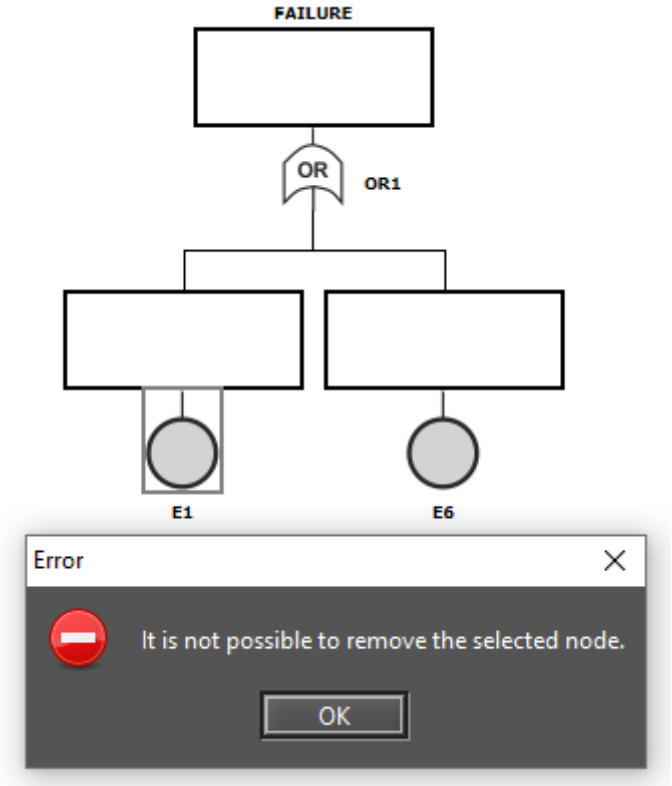


Figure 170: An Error Occurred While Trying to Delete a Basic Event

Let us next give an overview of the properties of basic events.

- **Parameters Type.** The basic events accept three types of parameters: DISTRIBUTION PARAMETERS,

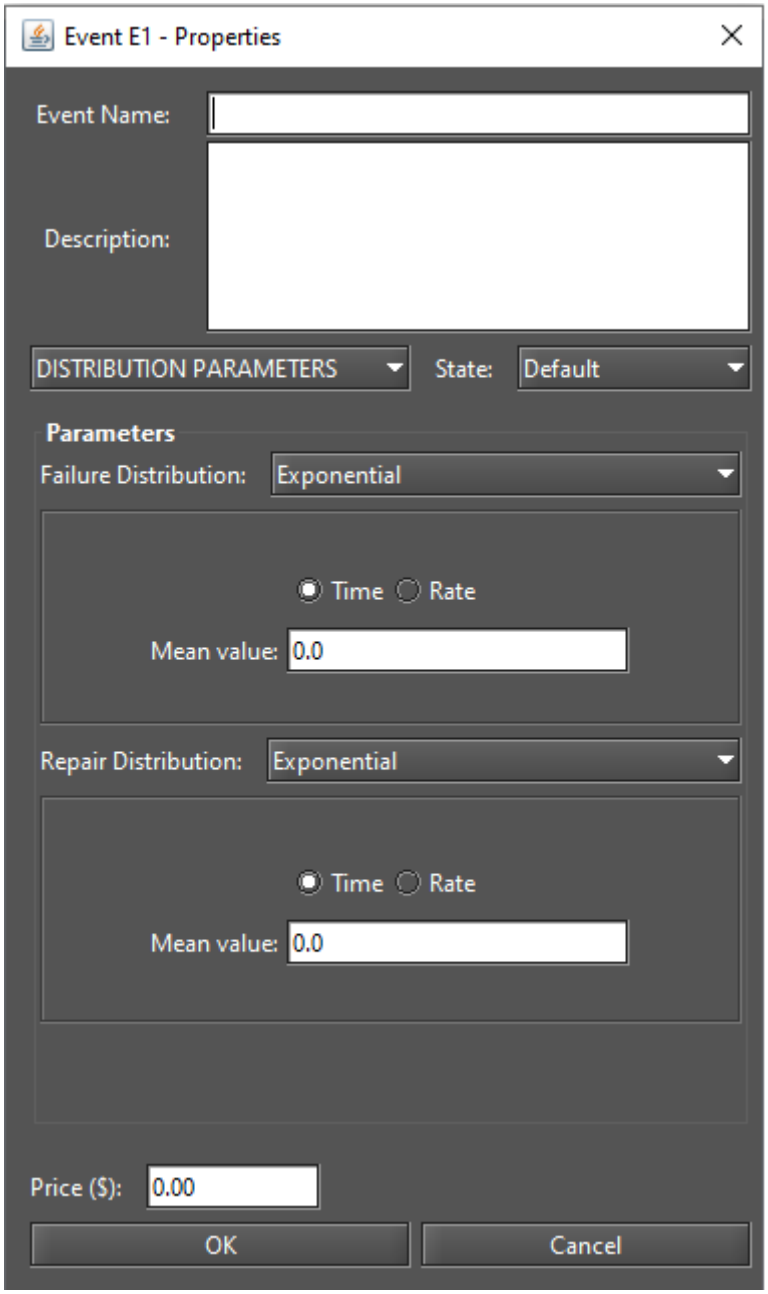


Figure 171: Properties Dialog for Basic Events

UNAVAILABILITY, and UNRELIABILITY. At any given time, only one of them can be selected. The default type is DISTRIBUTION PARAMETERS. If the parameter type is DISTRIBUTION PARAMETERS, the user can enter the appropriate values for the failure and repair parameters (see Figure 171). On the other hand, if the type is UNAVAILABILITY or UNRELIABILITY, the user can enter the corresponding value considering the selected type, as shown in Figure 172. In the context of the last figure, the user must enter the unavailability of the component represented by the basic event.

- **State.** State of the basic event. Two states are available: DEFAULT or FAILED. The default state is DEFAULT, which means that the component is working properly. On the other hand, if the state is FAILED, it means that the component has failed.

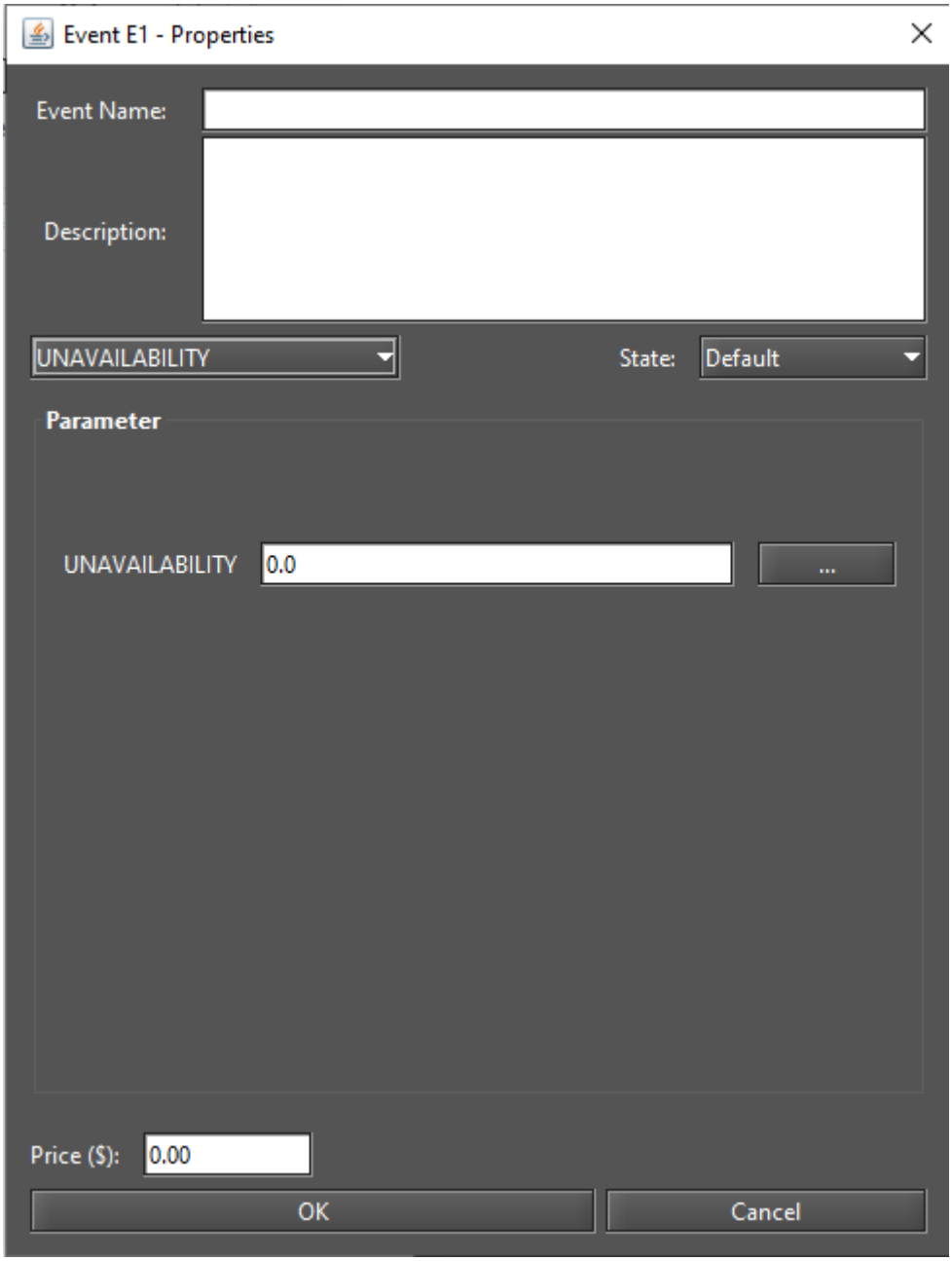


Figure 172: Defining the Unavailability for a Basic Event

- **Failure Parameters.** Mercury supports a large number of probability distributions. Depending on the distribution selected, fields appear representing the parameters of the selected distribution so that the user can enter their values. Each failure parameter can be assigned a label. Using the **"..."** button we can select an already declared label.

- **Repair Parameters.** Fields appear for the parameters of the selected distribution, where the user can enter the appropriate values. Each repair parameter can be provided with a label. Using the button **"..."** we can select an already declared label.

- **Price.** Cost related to the component represented by the basic event. The cost of the events is considered in the evaluation of the model by "Component Importance and Total Cost of Acquisition" method. See

Section 4.1.4 for more information on this type of evaluation.

Finally, let us look at the types of basic events and how they are represented graphically. Figure 173 shows the six types of basic events. Figures 173.a and 173.b show events that have no parameters associated with them. Figures 173.c and 173.d show exponential events, but in c) the state of the event is defined as “Default”, while in d) it is defined as “Failed”. Figures 173.e and 173.f show 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 without failure/repair parameters are represented by a light gray circle. Exponential events are represented by a dark gray circle. And non-exponential events are represented by a blue circle.

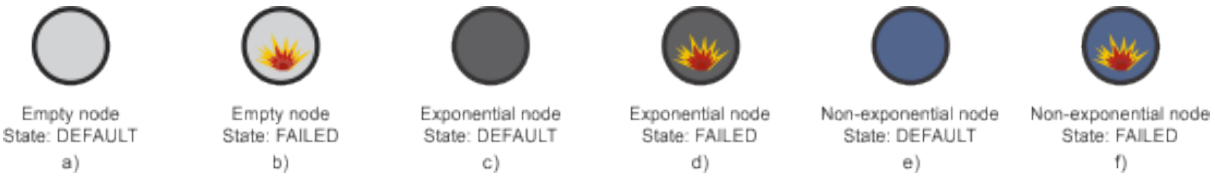


Figure 173: Types of Basic Events

When the required parameters of all basic events associated with a father gate are entered, the color for that gate changes to yellow, as shown in Figure 174.

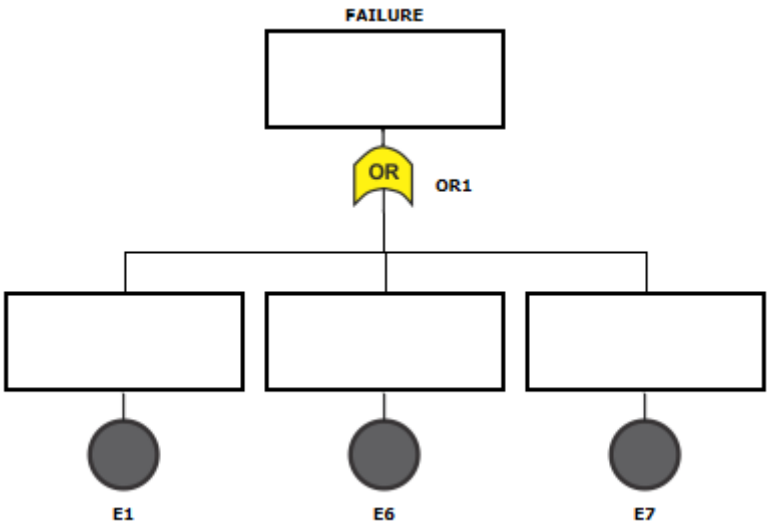


Figure 174: FT Model with All Event Parameter Values Defined

Mercury has a feature to improve the readability of models. Once the parameters of a node have been assigned, you can read them in the drawing area by positioning the mouse pointer over the node. A tooltip will then appear showing all the properties of that node. As we can see in Figure 175, all properties are displayed in the tooltip. All types of components of all formalisms supported by Mercury provide this feature.

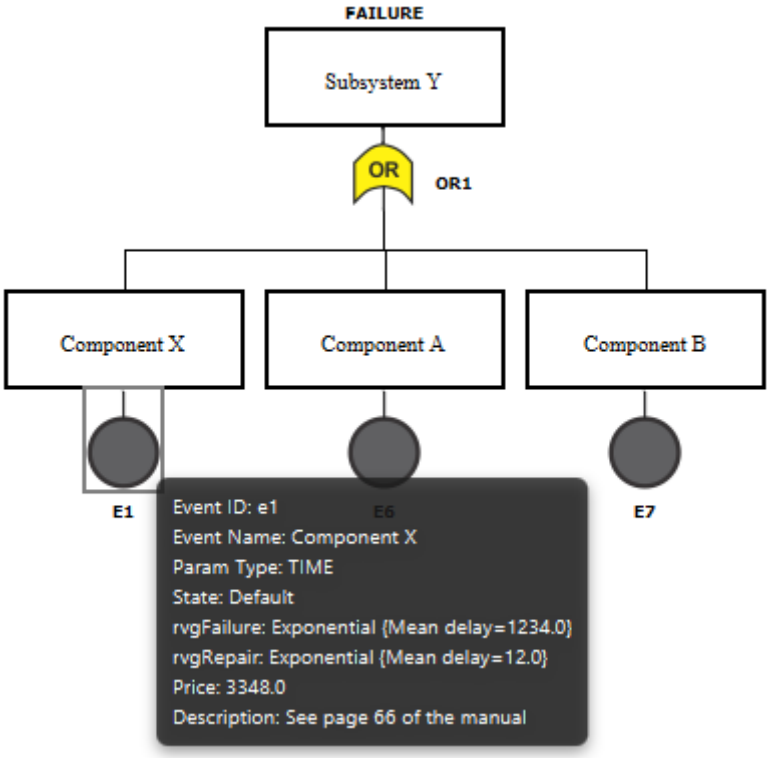


Figure 175: Tooltip for a FT Node

4.1 FT Evaluation

Mercury offers a large number of evaluations for FTs:

- Exact Evaluation;
- Bounds Evaluation;
- Importance Measures;
- Experiment;
- Get Functions;
- Sensitivity Analysis;
- Sensitivity Analysis (min/max values); and
- Export to RBD model.

These evaluations are available from the *Evaluate* -> *FT Evaluation* menu, as shown in Figure 176. We present these evaluations in the next subsections.

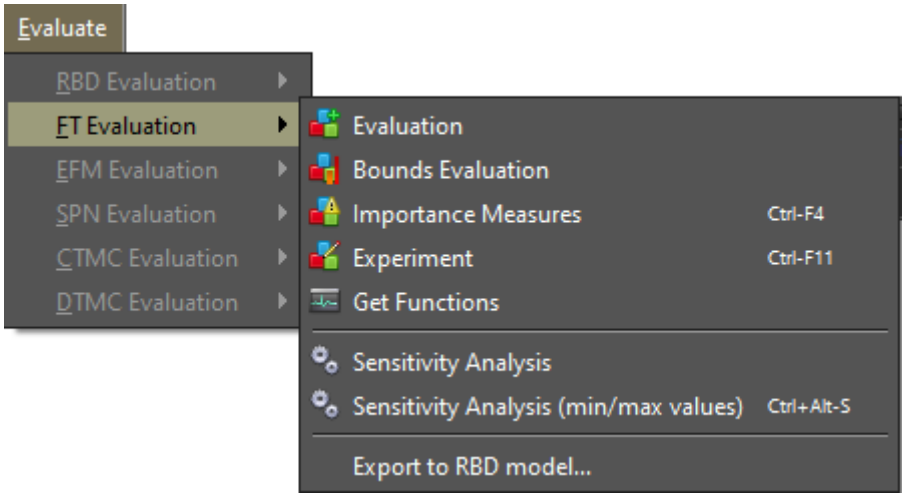


Figure 176: FT Evaluation Menu

4.1.1 Evaluation

A large number of dependability analyzes can be performed from the “Evaluation” menu. It can be accessed in the *Evaluate -> FT Evaluation* menu. Figure 177 shows the window for performing dependability evaluations. As we can see, we can evaluate eight metrics: mean time to failure, mean time to repair, steady-state unavailability, instantaneous unavailability, reliability, unreliability, uptime and downtime. Users can select or deselect all metrics by clicking the toggle checkbox labeled “Select All”. Users can also select the unit of time to be considered when calculating uptime and downtime: seconds, minutes, hours, and days. If time-dependent metrics are selected — reliability, unreliability or instantaneous unavailability —-, the time parameter is required. In addition, there is an option to analyze time-dependent metrics considering multiple points in time. The time interval goes from 0 to the evaluation time. The metric is calculated for each point.

Mercury provides two methods for calculating dependability metrics. We can choose between SFM (structural function method) and SDP (sum of disjoint products), as shown in Figure 178. SFM calculates measures considering the structural function of the model. The SDP method, on the other hand, which is based on Boolean algebra, calculates measures considering minimal cuts and minimal paths. After selecting the options and entering the evaluation time and the number of sampling points, the user must click on the “Run” button.

Let us now demonstrate how you can use the Mercury tool to perform evaluations on FTs. We have considered a model consisting of four events, each of which contributes to the overall failure of the evaluated system (see Figure 179). As we can see, the system fails when the events **e1** or **e6** occur, or both events occur in the AND gate — **e4** and **e5**. The node e6 represents a k-out-of-n component. For event e6 to occur, at least 3 of the 5 components must fail.

We performed an evaluation by considering 600 hours, ”days” as the time unit, and six sampling points (see Figure 180). At the end of the evaluation, a window with the results appears, as shown in Figure 181. 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 an example of a result obtained by evaluating the FT model

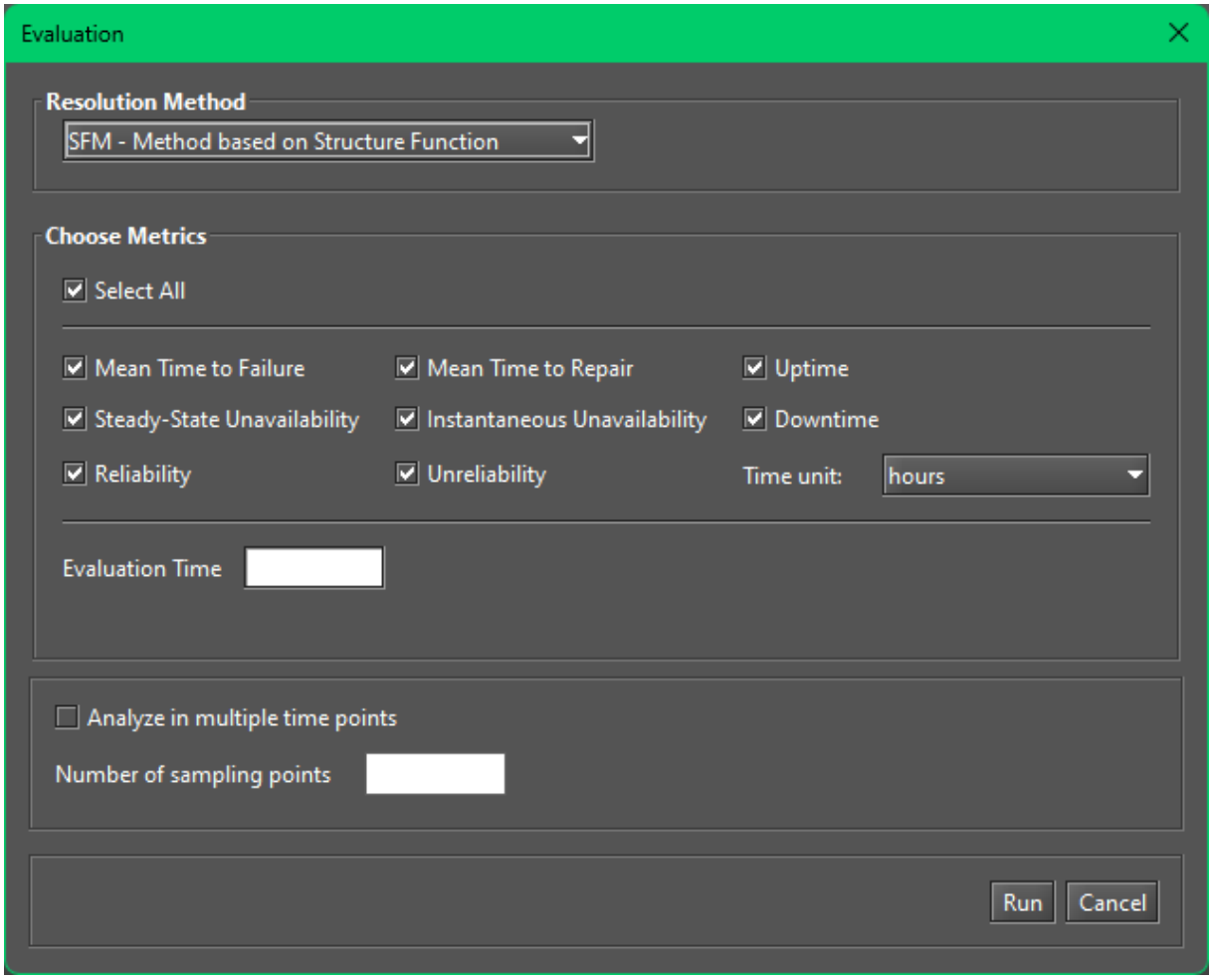


Figure 177: Evaluation for FTs

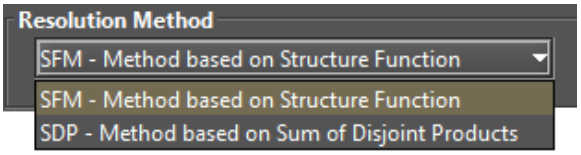


Figure 178: Resolution Methods

presented in Figure 179.

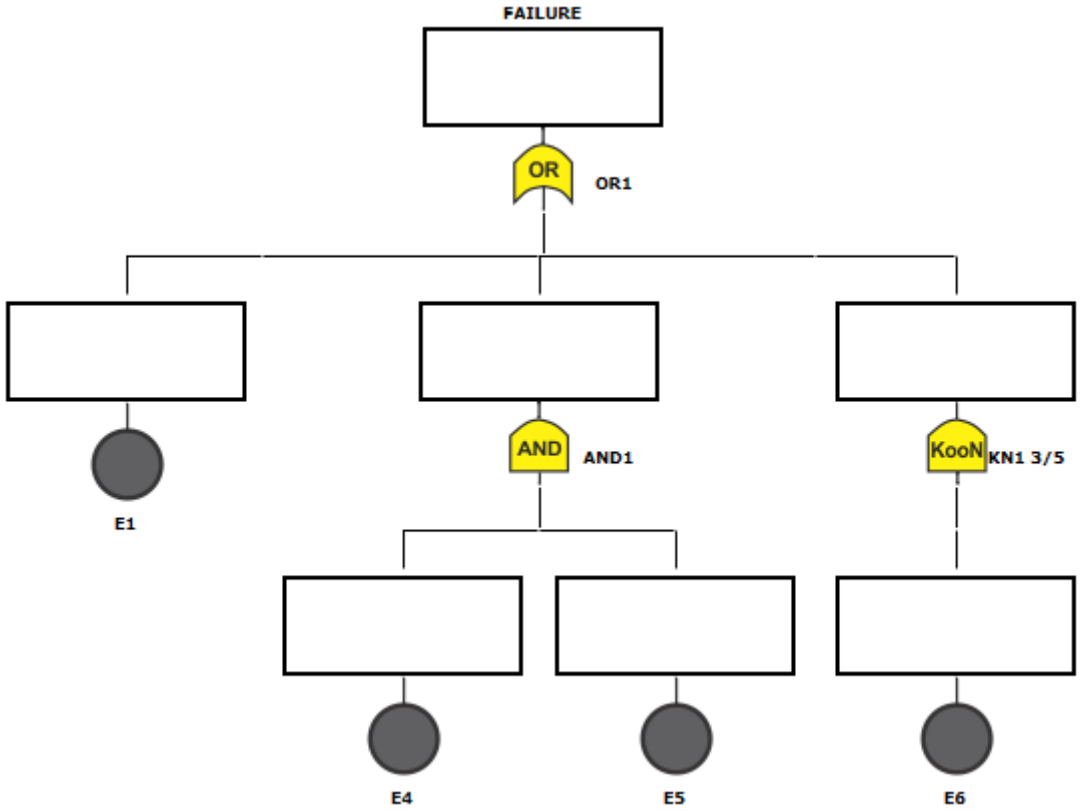


Figure 179: FT Model for Dependability Evaluation

Listing 6: Dependability Result

```
************ Steady-State Results ************
MTTF:              1048.7722944655652
MTTR:              40.804042901857514
Unvailability:     0.04
Number of 9's:     0.016576456949086937
Uptime:            351.564076736834 days
Downtime:          13.678122263166 days
************ Instantaneous Results ************
Time          Unreliability   (9's)          Reliability    Inst.unavail
0.0000        0.0000          0.0000         1.0000         0.0000
100.0000      0.078290535387  0.035405952916 0.921709464613 0.03314820574
200.0000      0.15143914031   0.071317004166 0.84856085969  0.036955533135
300.0000      0.219869505504  0.107832745644 0.780130494496 0.037392741526
400.0000      0.283901139697  0.145027017468 0.716098860303 0.037442945104
500.0000      0.343782929813  0.182952476492 0.656217070187 0.037448709832
600.0000      0.399717857929  0.221644576376 0.600282142071 0.037449371779
```

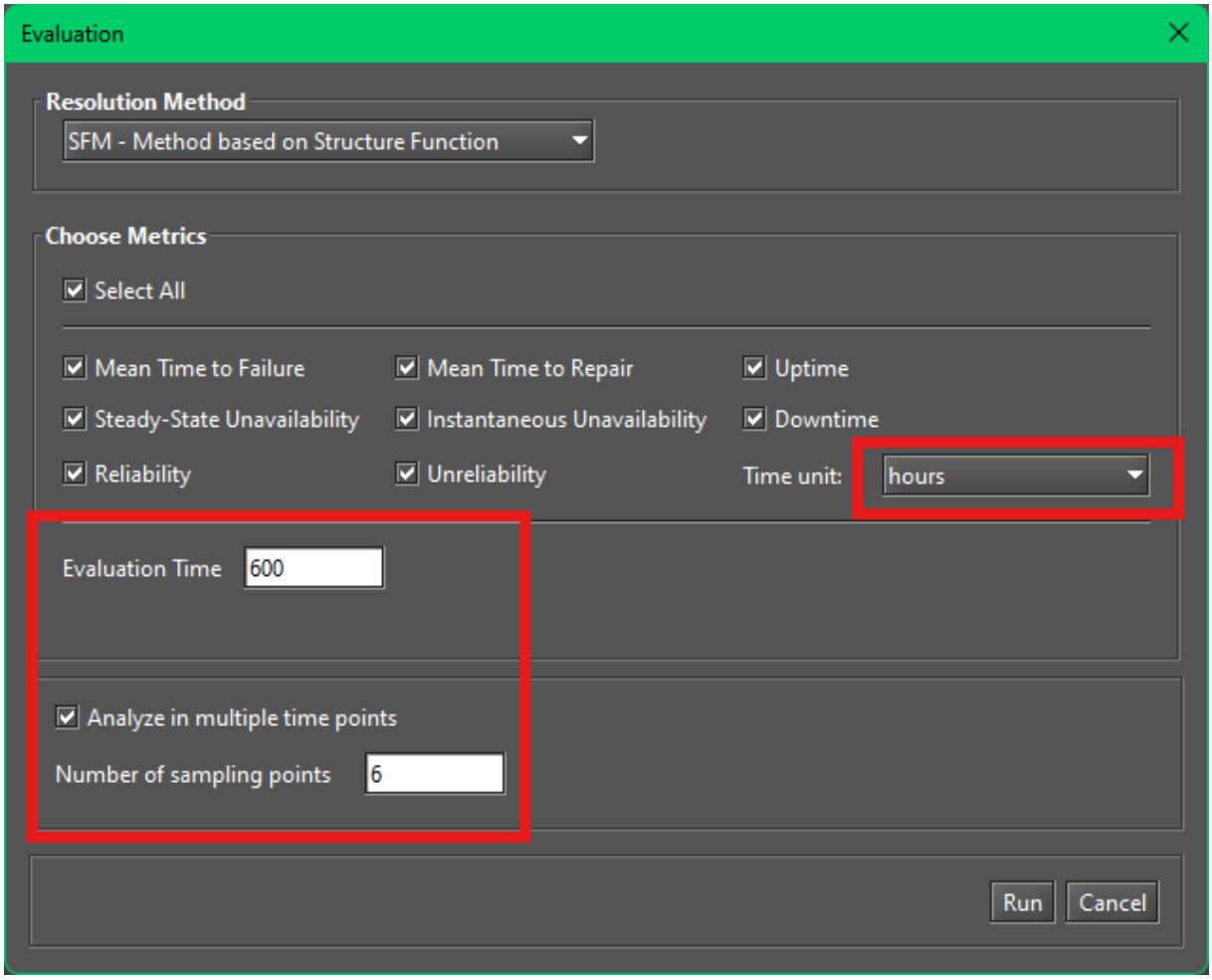


Figure 180: FT Analysis

Figure 182 shows the “Unreliability Chart” dialog, which is displayed by clicking the “Plot Unreliability” button. This button is only visible when reliability is selected in 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 by simulation are required when non-exponential probability distributions are associated with an event. Figure 183 shows a model with a non-exponential node (node e1). Non-exponential nodes are converted to SPNs and the model is solved by simulation. When Mercury detects this situation, it displays the message ”Non-exponential distributions detected" at the bottom of the Evaluation dialog (see Figure 184).

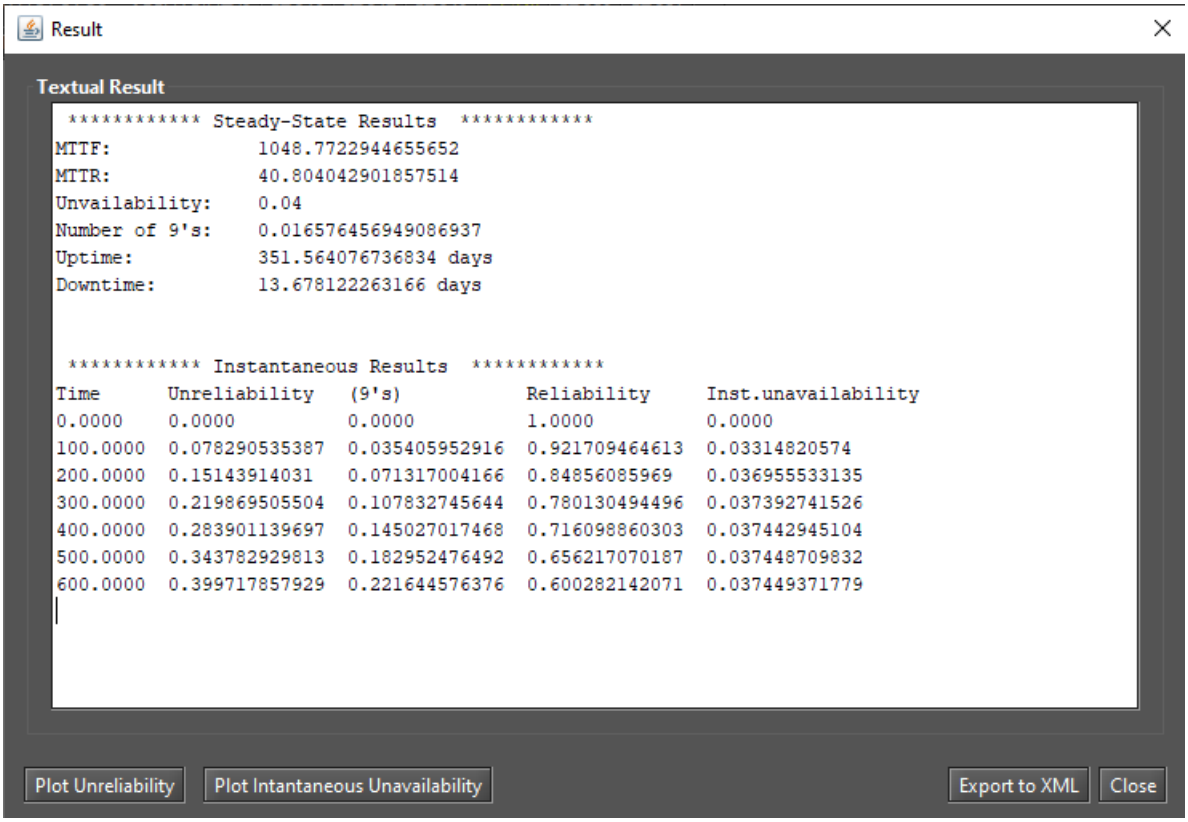


Figure 181: Dependability Result

When you click the “Run” button, some parameters must be entered to support the simulation. The parameters required depend on the metrics you choose. If you select only steady-state metrics, Mercury displays the window shown in Figure 185. On the other hand, if you select one or more time-dependent metrics, Mercury displays the window shown in Figure 186. If you select both transient and steady-state metrics, both tabs appear in the same window, as shown in Figure 187. In this case, when clicking the Run button in the dialog box shown in Figure 187, Mercury considers the parameters in both tabs. If no changes are made to the parameters, Mercury considers the default parameter values. The results are displayed once the parameters have been defined and the simulation is complete. Results are presented with confidence intervals, as shown in Figure 188. For more information about simulations, see Section 2.1.

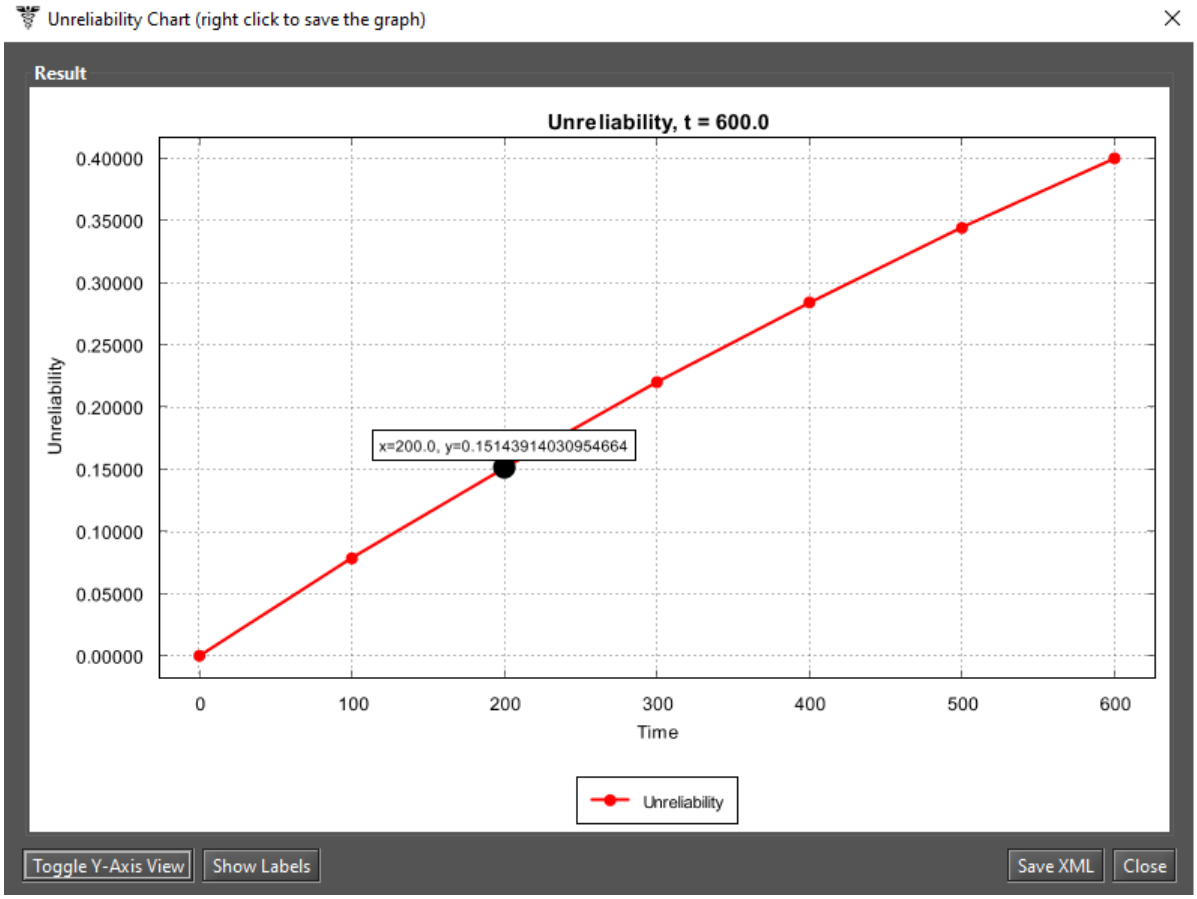


Figure 182: Unreliability Chart

4.1.2 FT Experiment

Mercury allows us to evaluate the impact of varying some parameters on the model. Now we will show you how to use the experiment feature. The first step is to define one or more labels. Labels are variables that store numerical values and can be associated with the failure/repair parameters of nodes. The value of a label is changed taking into account a step size and at each change the selected metric is evaluated. A label is inserted by right-clicking on an event and selecting “Insert label” as shown in Figure 189. Another way is to right-click on the label area in the left pane and choose “Insert label”, as depicted in Figure 190. Once this is done, the “Label Properties” window is displayed (see Figure 191). There the user can set the properties of the label.

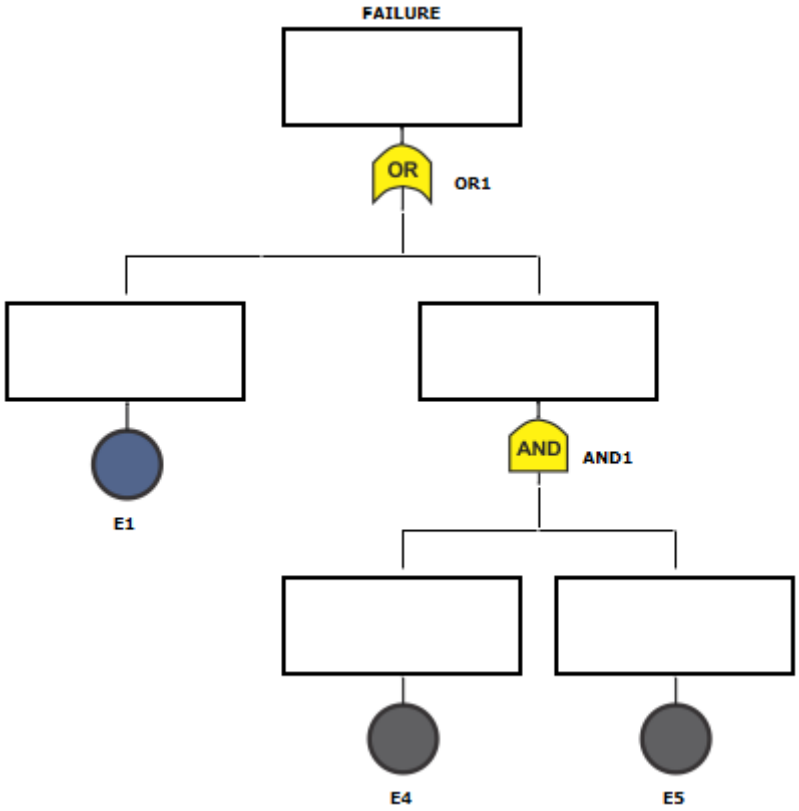


Figure 183: FT with a Non-Exponential Node

Once a label is inserted, it is available in the left window on the FT tab. Now it can be associated with one or more event parameters. When you right-click on a label, a popup menu appears with three menu items (see Figure 190). We describe each of these items below.

- **Insert label.** Displays the “Label Properties” window where the user can insert a label.
- **Remove label.** Remove the selected label.
- **Properties.** Displays the “Label Properties” window where the user can 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 192 demonstrates how to attach a label to an event parameter. It is also possible to attach a label to the price parameter of a component in the FT model, as shown in Figure 193.

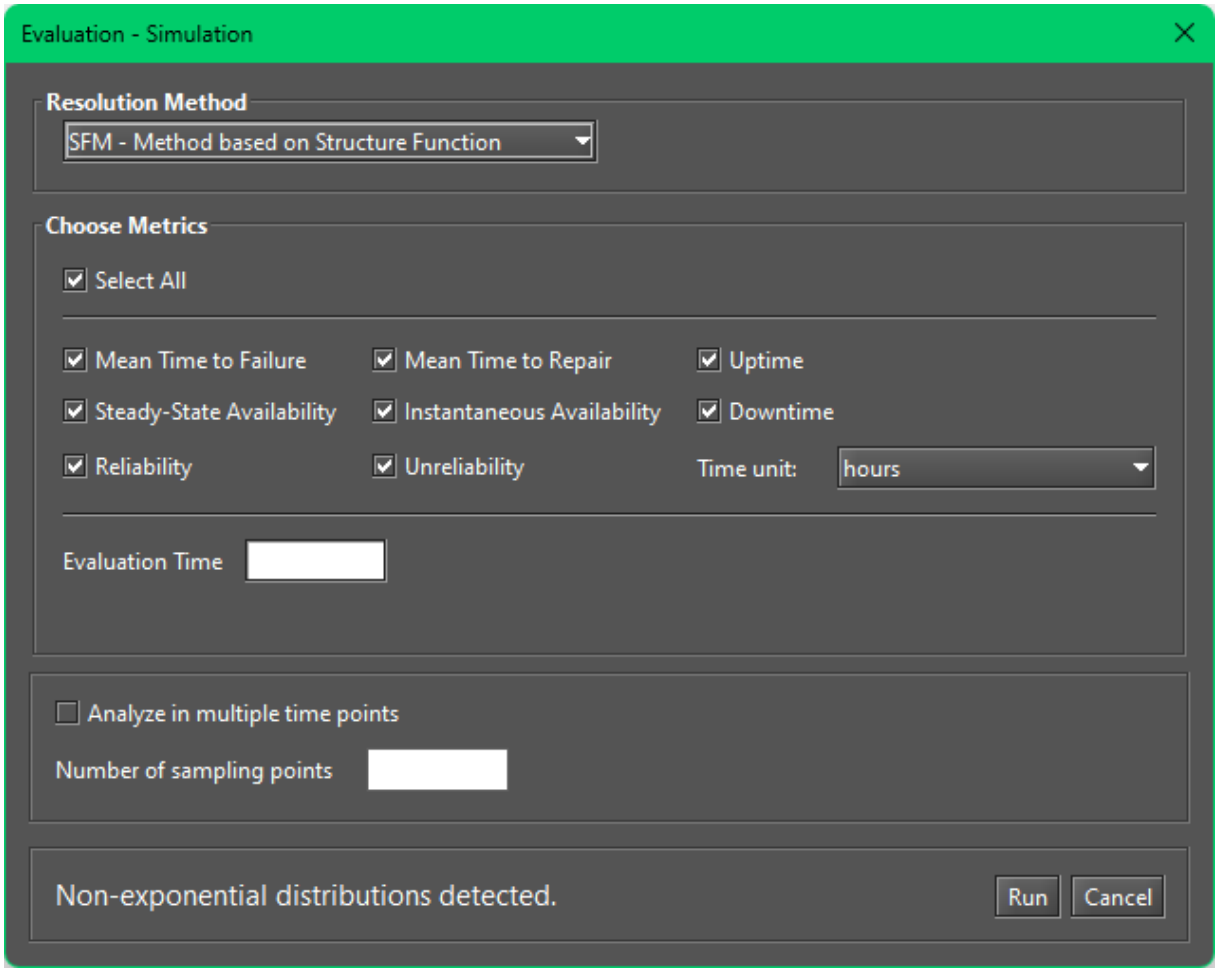


Figure 184: FT Evaluation by Simulation

"Experiment" can be accessed from the menu *Evaluate -> FT Evaluation -> Experiment*. Figure 194 shows the "Experiment" window. To run experiments, the user must enter values for all required fields. We describe each of these options below.

- **Parameter.** The label whose value will be changed at each iteration of the experiment.
- **Metric.** The metric to be evaluated.
- **Minimum Value.** Initial value for the selected label.
- **Maximum Value.** Final value for the selected label.
- **Type.** Determines whether the value of the parameter is changed linearly or logarithmically. If it is logarithmic, the parameter value is changed by a base-10 logarithmic function, taking into account the minimum and maximum values.
- **Interval.** Step size that will be taken into account when changing the value of the label. The label starts with the minimum value and its value is incremented considering this interval. At each change, the selected metric is evaluated. The experiment is finished when the maximum value for the label is reached.

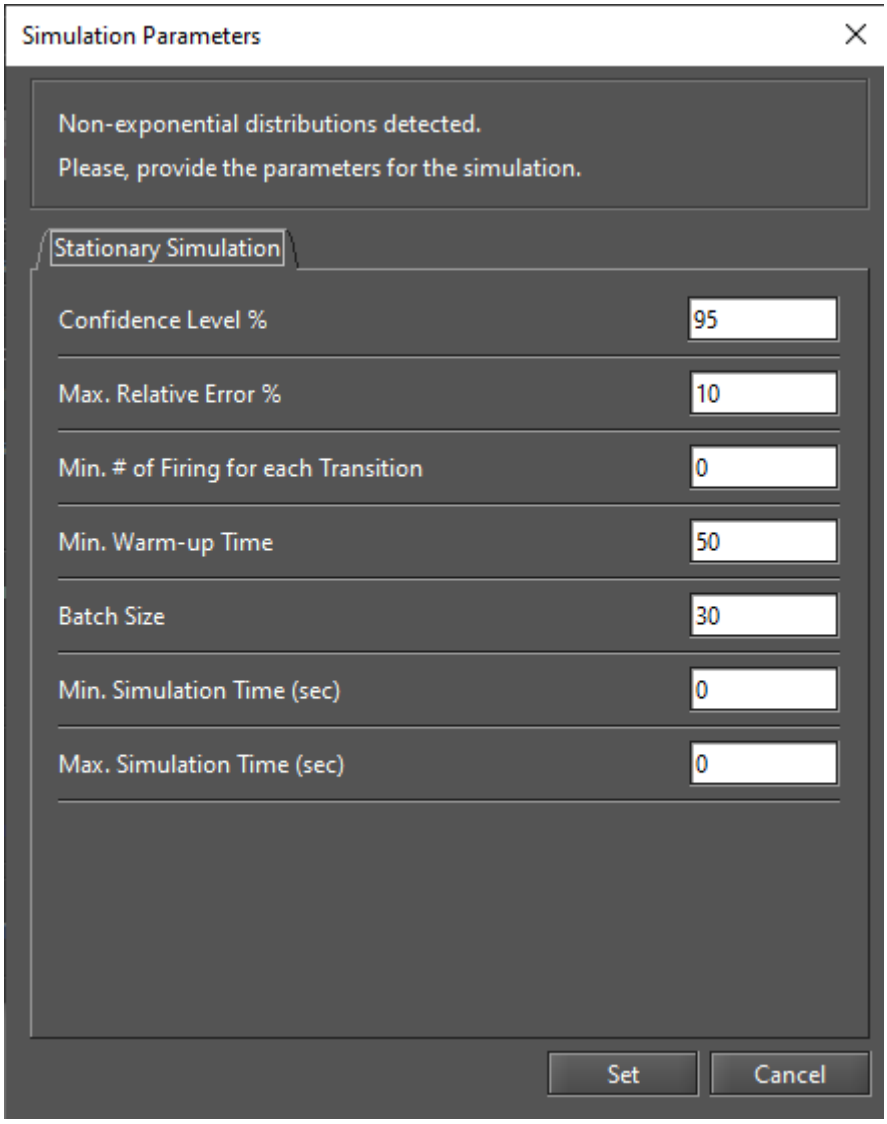


Figure 185: Simulation for Steady-State Metrics

- **Evaluation Time.** Evaluation time considered in the calculation of time-dependent metrics. For time-dependent metrics — reliability, unreliability, instantaneous unavailability — it is necessary to enter the time parameter.

After defining the input parameters, the user must click on the “Run Experiment” button to start the experiment. If the model to be evaluated contains non-exponential events, the user must also enter the simulation parameters, as shown in Figure 195. Once the experiment is finished, the “Experiment Result” dialog is displayed (see Figure 196).

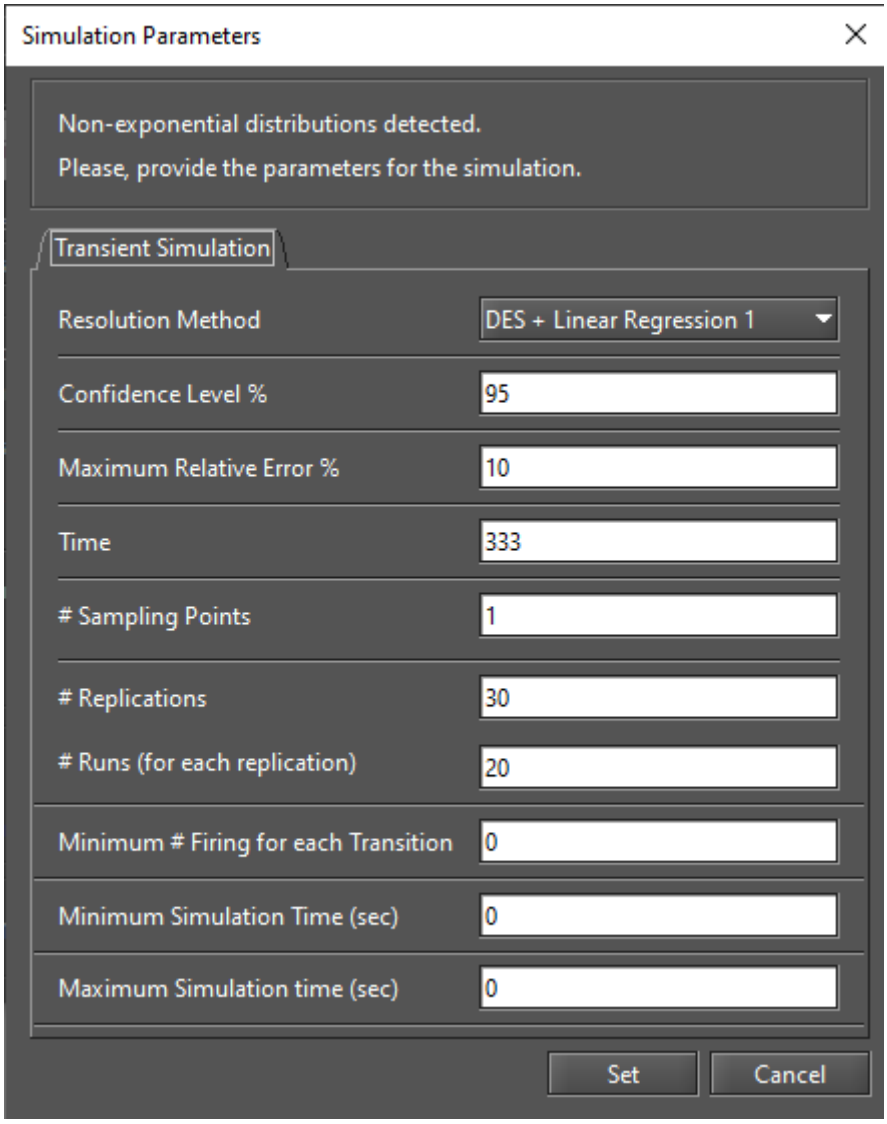


Figure 186: Simulation for Time-Dependent Metrics

4.1.3 Bounds for Dependability Analisys

"Bounds for Dependability Analysis" is used to estimate dependability metrics by calculating unreliability, unavailability, or downtime. It is necessary to estimate the bounds (upper and lower limits) for the calculation of this analysis, where users get results quickly. This analysis should be performed when the model is huge. This analysis is divided into two parts: (i) the calculation of the limits and (ii) the use of the sum of disjoint points to determine the successive values and the number of iterations required.

Users can access this analysis by going to the *Evaluate -> FT Evaluation -> Bounds Evaluation* menu. Figure 197 shows the "Bounds for Dependability Analysis" window. As shown in Figure 198, four metrics can be evaluated: Steady-State Unavailability, Instantaneous Unavailability, Unreliability, and Downtime. The "time" parameter is required if you select "Instantaneous Unavailability" metric. Once you have selected a metric and entered the time, if applicable, you must click the "Get Start Values" button to start the evaluation.

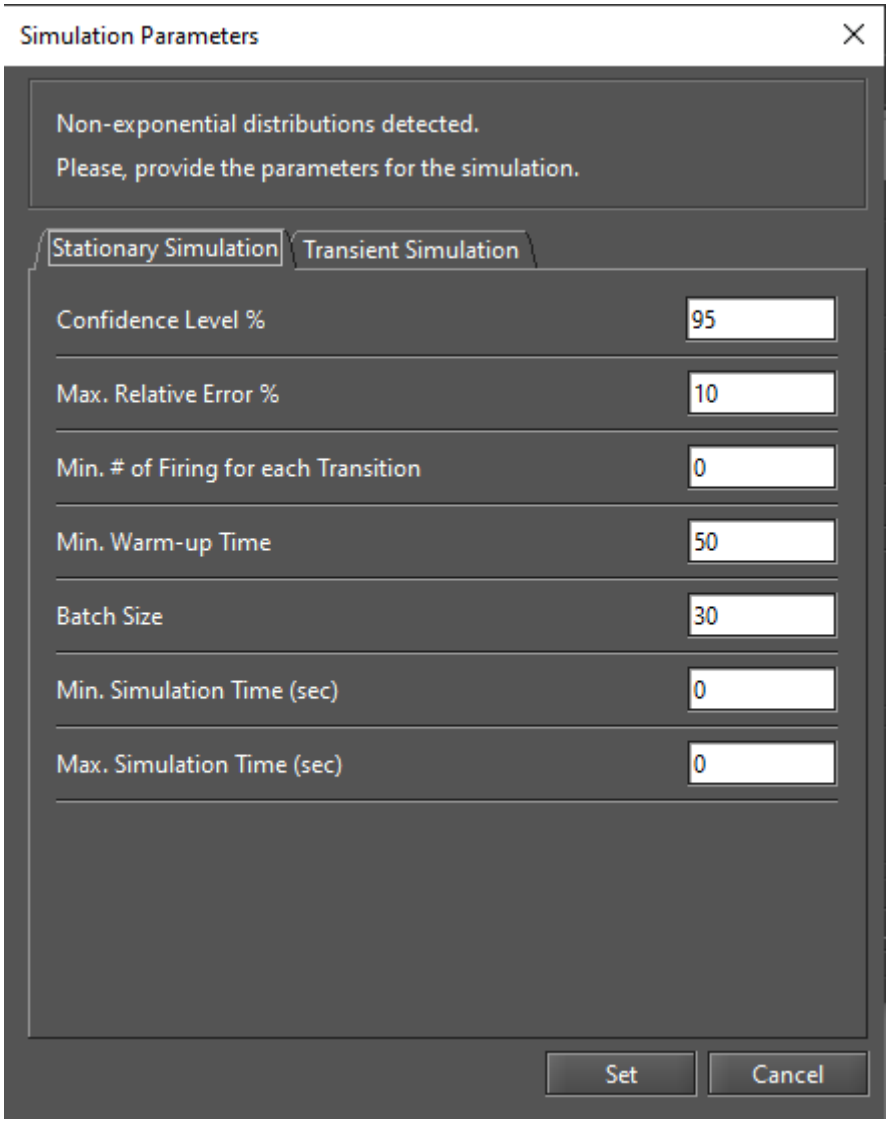


Figure 187: Simulation for Steady-State and Time-Dependent Metrics

Let us now demonstrate how to perform bounds evaluation using Mercury. We have considered a model consisting of five events, each of which contributes to the overall system failure (see Figure 199). As we can see, the system fails when the events **e1**, **e3**, and at least one of the following events occur: **e4**, **e5**, or **e6**. We evaluated the bounds for this model by considering the time parameter equal to 8760h (see Figure 200).

First, the upper and lower values are calculated. The first path and the first cut are used to determine the upper and lower values of the selected metric. The paths refer to the lower bounds, where a minimal set of components is chosen to ensure the operational mode of the system. The cuts refer to the upper bounds where a minimal set of components is chosen to ensure the system in failure mode. After determining the upper and lower values, we set the number of steps for the upper and lower values to three. Thus, we obtained the result shown in Figure 201. 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. If you click on the "Plot Chart" button, you can see how the lower and upper bounds for three steps converge to the exact value (see Figure 202).

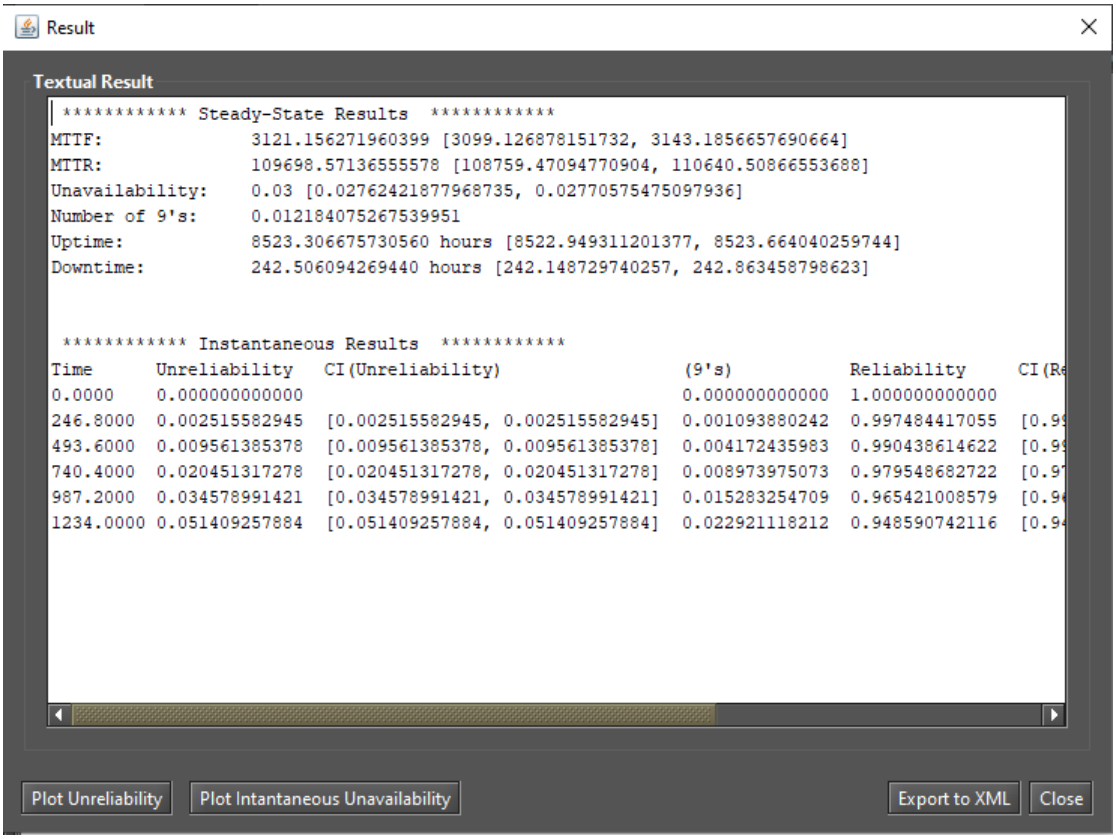


Figure 188: Result from a Non-Exponential FT Model Evaluation with Confidence Intervals

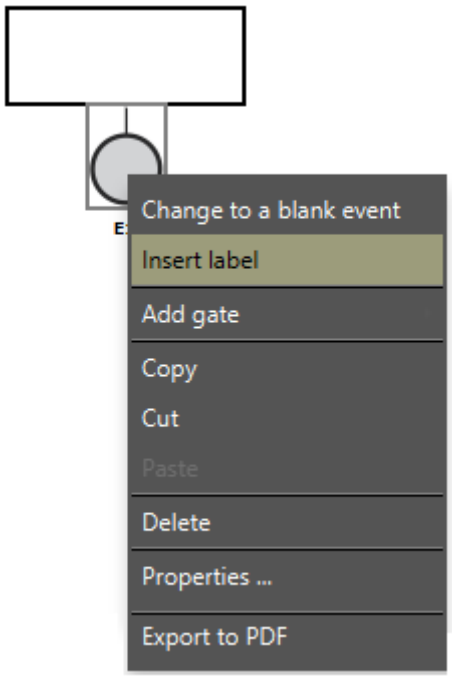


Figure 189: Inserting a Label into the FT Model

The method of determining successive values and the number of iterations is defined by the number of paths and cuts in the model. If you increase the number of iterations, the value found will be closer to the exact value. Once the calculation of the last path or cut is complete, the exact value of the metric can be found. The exact

Figure 190: Left-Side FT Pane

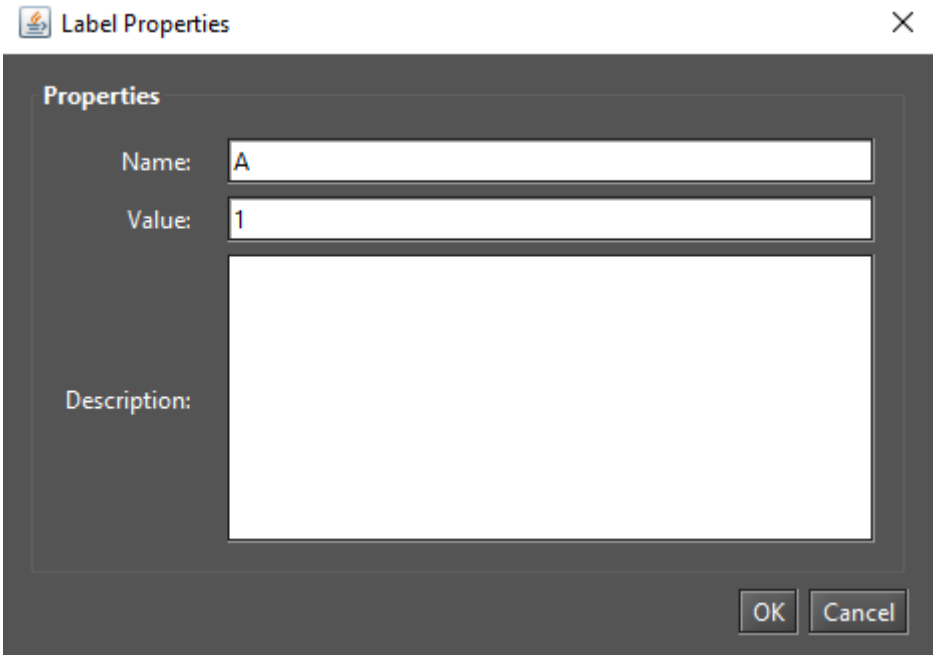


Figure 191: Inserting a Label in the FT Model

value will be the value found in the last step. We have performed the exact evaluation to obtain the reliability at 8760h. Figure 203 shows that reliability at 8760h is equal to the values obtained in the bounds evaluation for the maximum number of steps (see Figure 201).

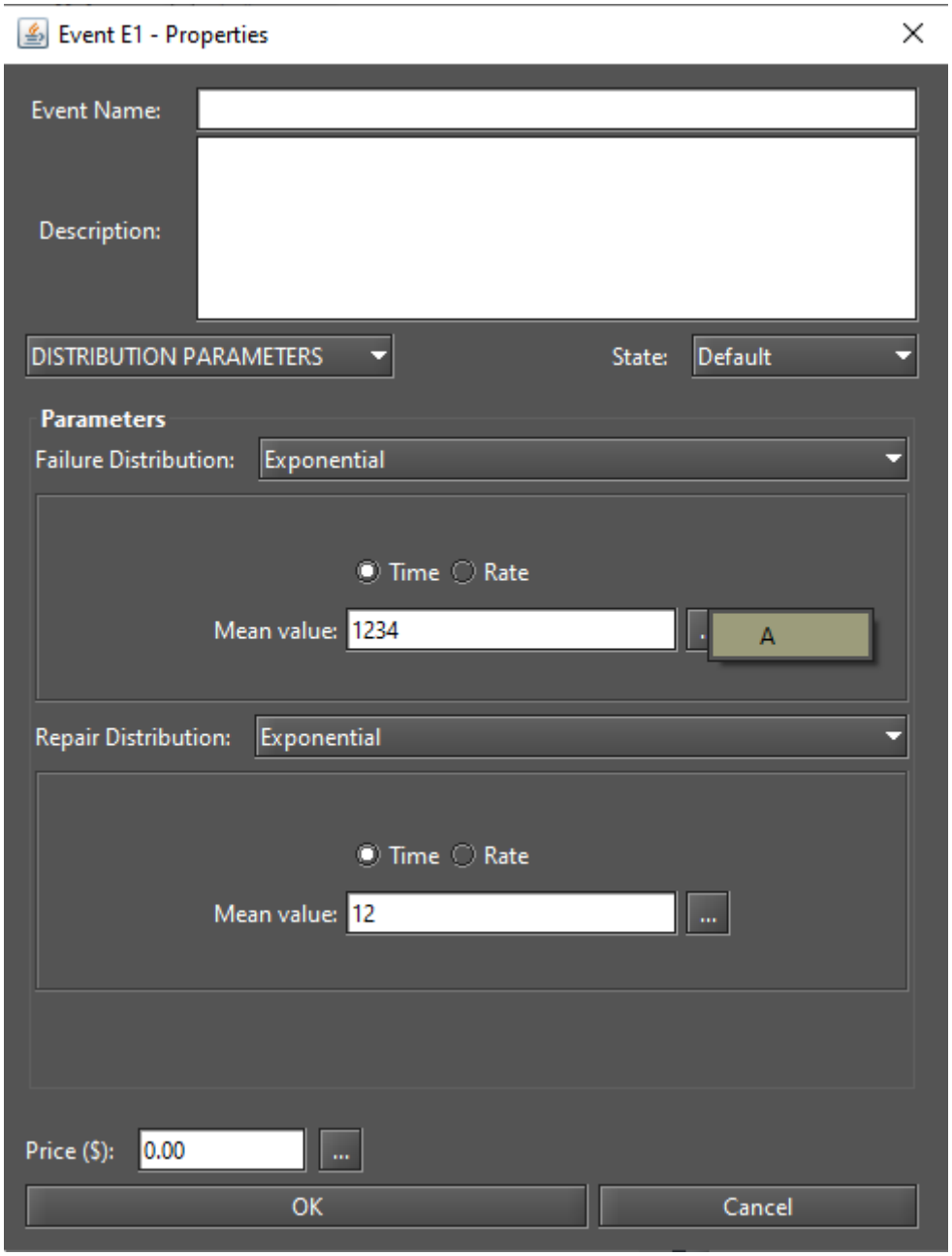


Figure 192: Attaching a Label to an Event Parameter

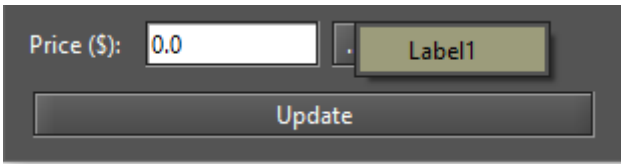


Figure 193: Attaching a Label to the Price Parameter of an Event

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. Considering the importance scores, the most important component (i.e., the component with the highest importance) should be improved to increase the reliability or availability of the system. This evaluation can be used, for

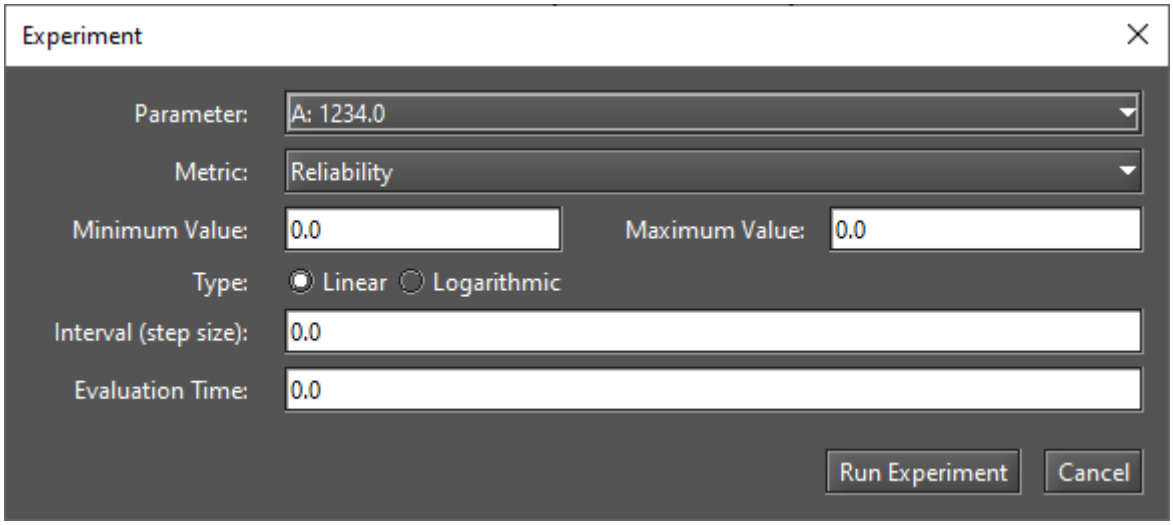


Figure 194: FT Experiment

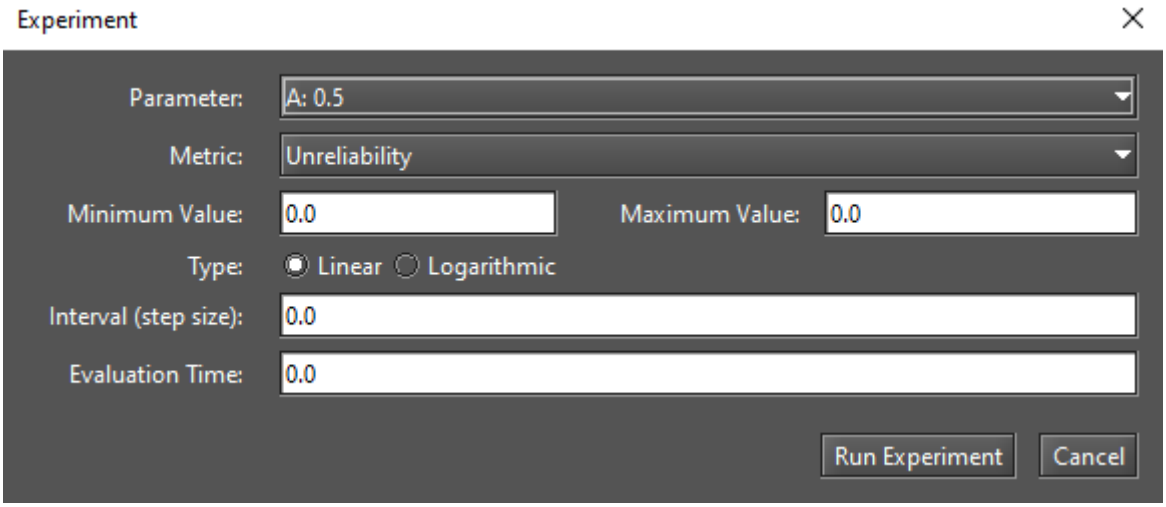


Figure 195: FT Experiment - Simulation Parameters

example, to support maintenance activities.

You can use importance measures to determine the relative importance of each component with respect to the reliability or availability of the overall system. You can access this evaluation by selecting “Importance Measures” from the *Evaluate -> FT Evaluation* menu. Then you need to select a metric in the “Component Importance Measures” window and then click the “Evaluate” button (see Figure 204). If the “Cost” parameter has been set for the nodes, it is also possible to evaluate the relationship between metrics and investment costs. The parameter “Time” is needed for the evaluation of the reliability metrics.

As shown in Figure 205, you can choose between a few types of measures. The types are: “Availability Importance", “Reliability Importance (Birnbaum)", “Criticality Reliability Importance" and “Criticality Availability Importance." The “Criticality Importances” measures are obtained by considering the system in failure “(f)” or in operation.

Below we show you how to perform “Component Importance” evaluations. We performed this evaluation

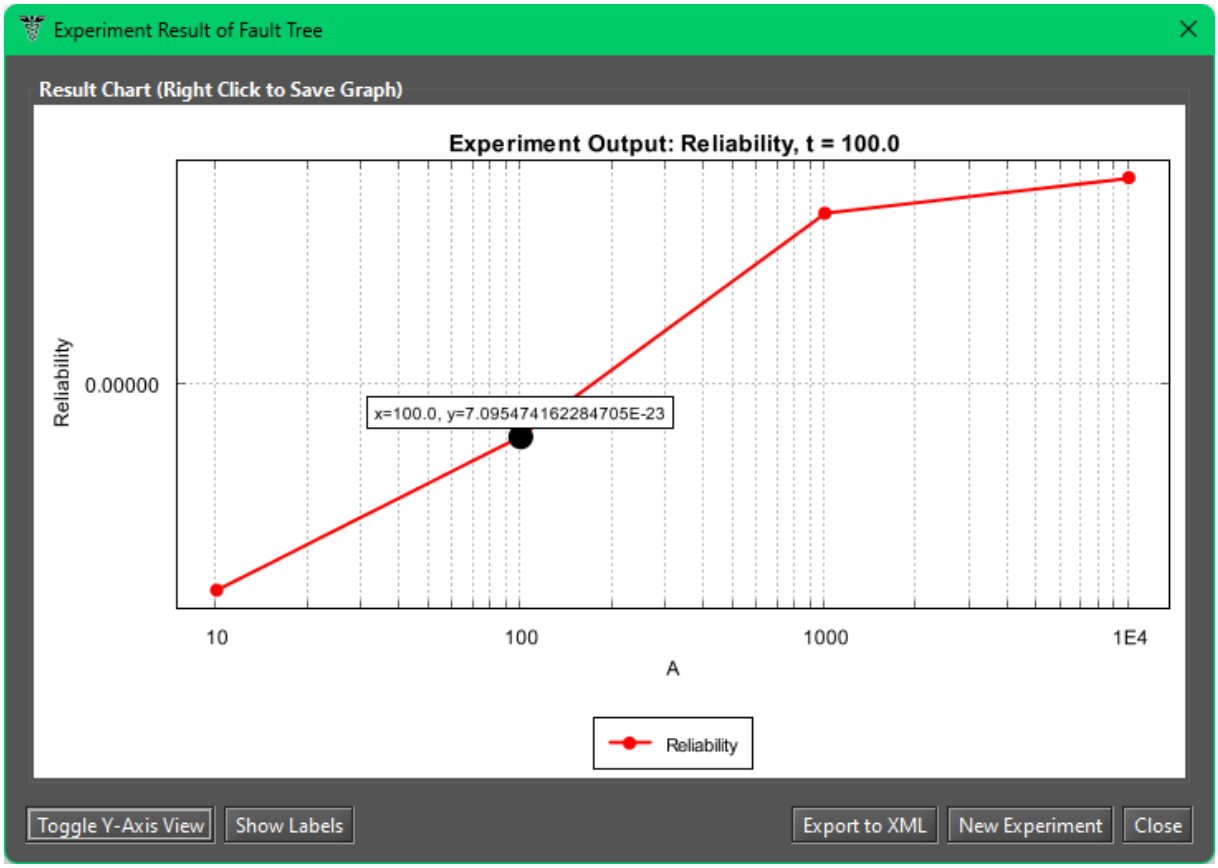


Figure 196: Experiment Result Dialog

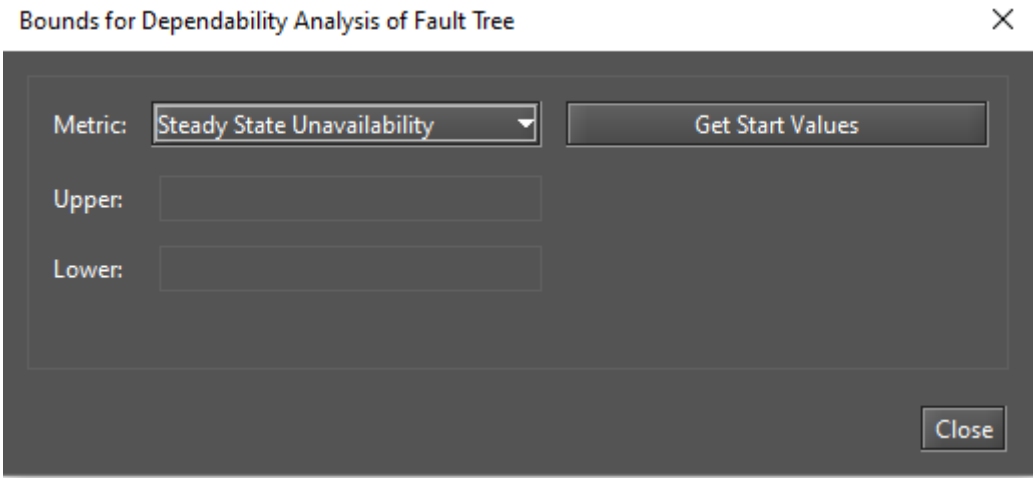


Figure 197: Bounds for Dependability Analysis

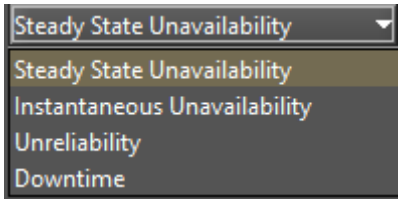


Figure 198: Metrics for Bounds Evaluation

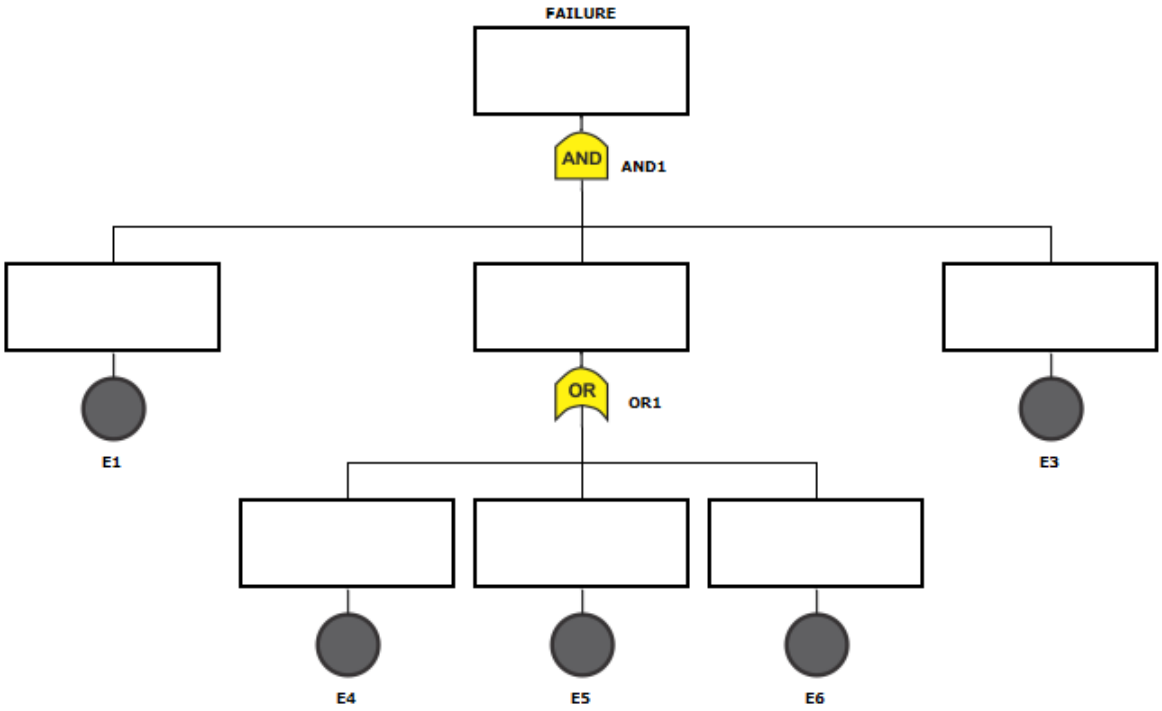


Figure 199: FT Model for Bounds Evaluation

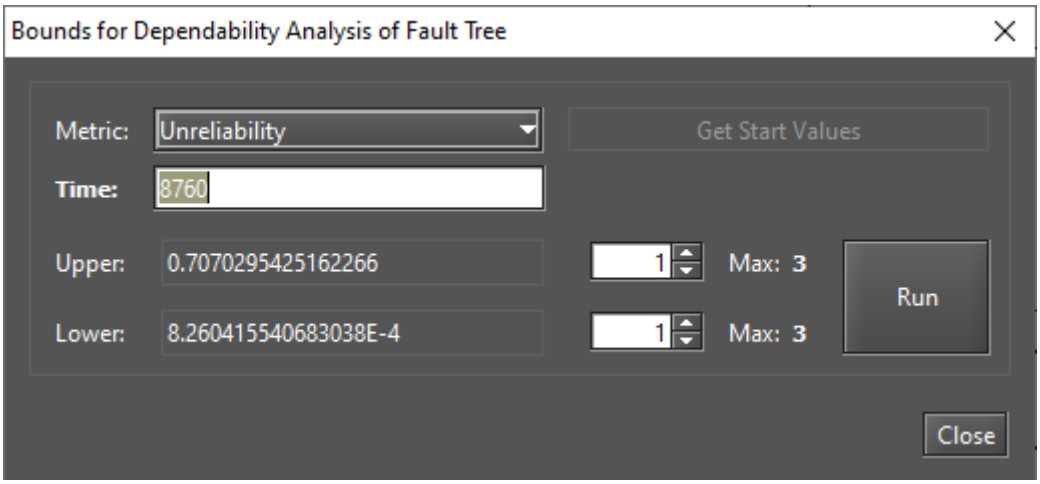


Figure 200: Bounds for Reliability

considering the ”Reliability Importance (Birnbaum)” metric for the model shown in Figure 206.

As we can see in Figure 206, the evaluated system has two sensors and each is connected to the single gate of the model. For the system to fail, at least one sensor must fail. “Sensor A” has an MTTF of 17520h and an MTTR of 72h. “Sensor B” has an MTTF of 6000h and an MTTR of 24h. All times are exponentially distributed. When we perform the “Component Importance” evaluation considering the time parameter of 8760h, we obtain the results shown in Figure 207.

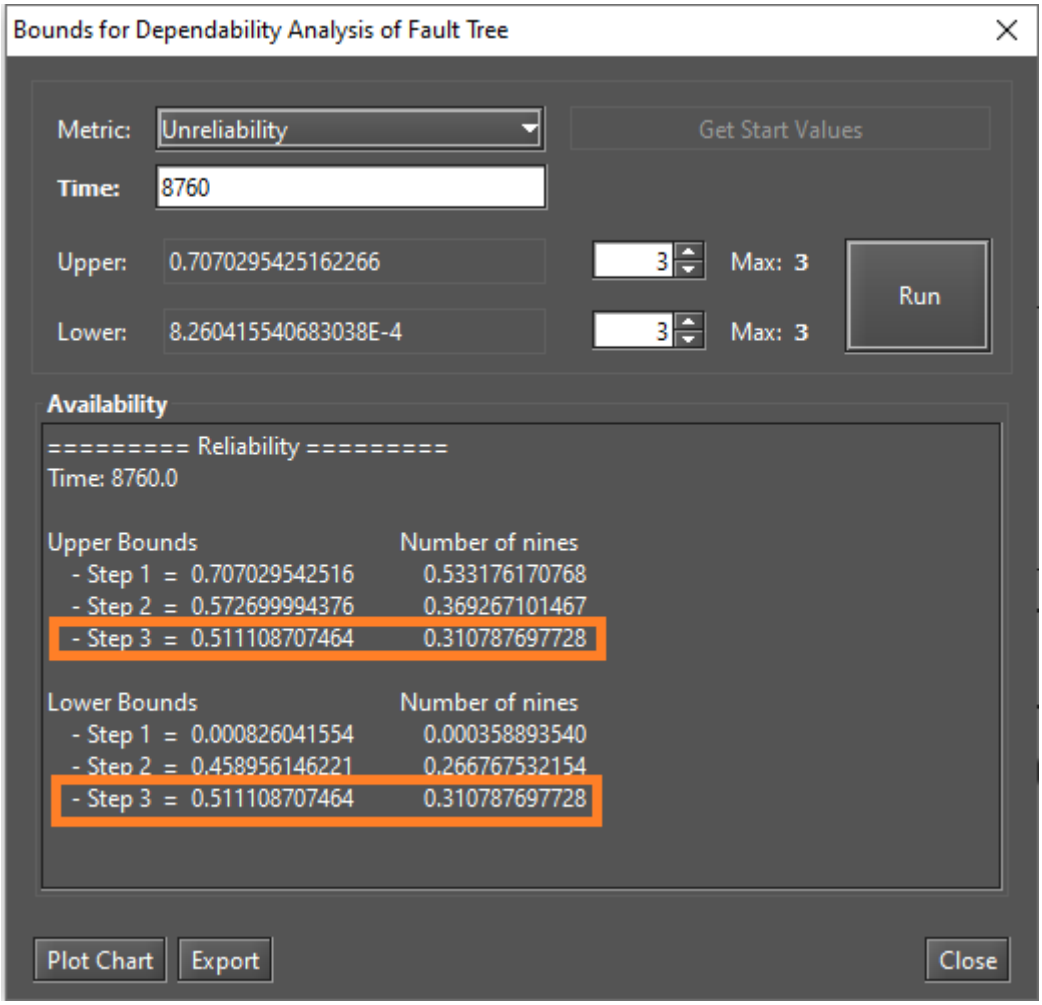


Figure 201: Bounds for Unreliability - Result

The results show the importance value for each component and a graphical representation in the form of a ranking that highlights the most important components. As we can see, the component “Sensor A” (e1) is the most important for the system reliability. If you replace “Sensor B” (e2) with another one with the same MTTR but an MTTF of 12000h, the result changes as shown in Figure 208.

We can see that the importance of “Sensor A” (e1) has decreased. However, considering a time interval of 8760h, “Sensor A” is still more important for overall reliability when compared to “Sensor B”.

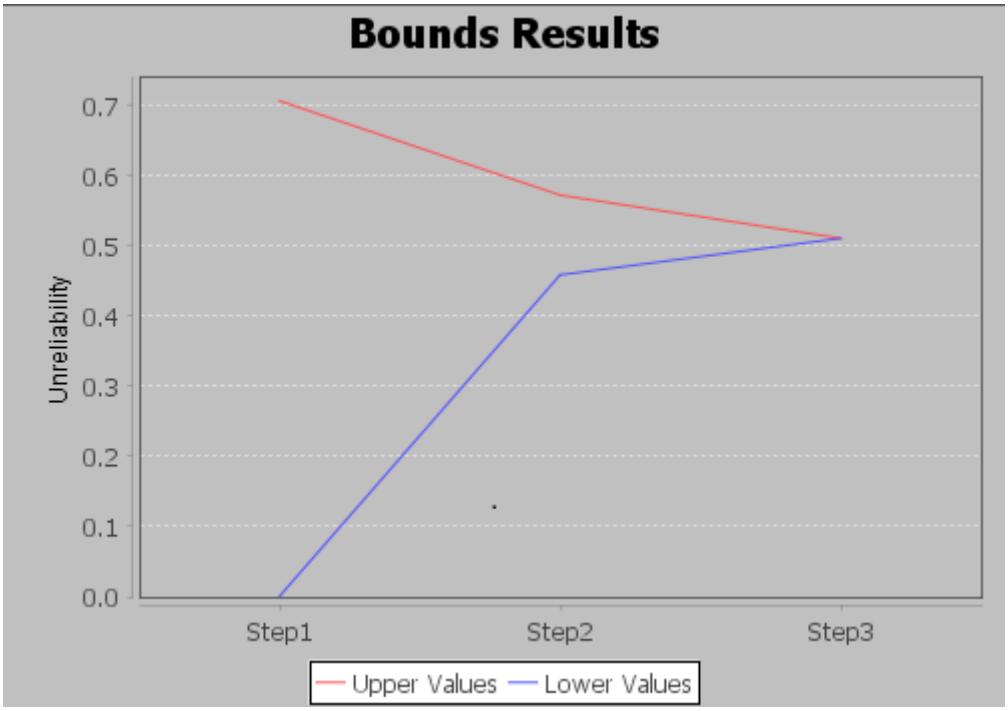


Figure 202: Plotting Bounds Result

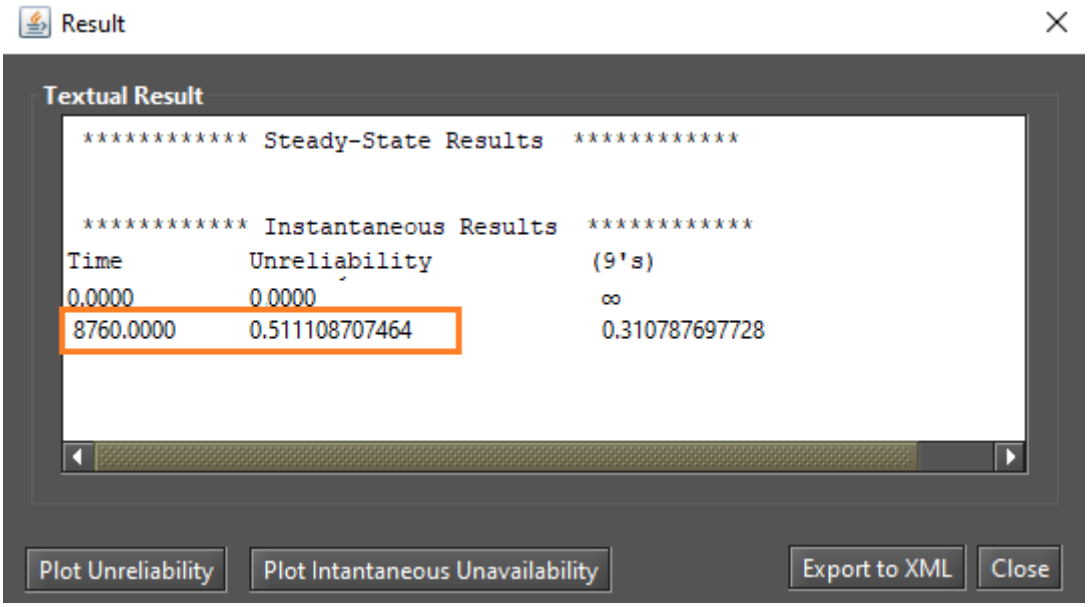


Figure 203: Unreliability at 8760h

4.1.5 Structural and Logical Functions

Mercury generates structural and logical functions of FT models. Both functions represent the system and refer to the states of the individual components. Also, it is possible to evaluate the impact on the system operation considering the faulty components. 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 event's properties (see Figure 209). If the state of an event is failed, the component is

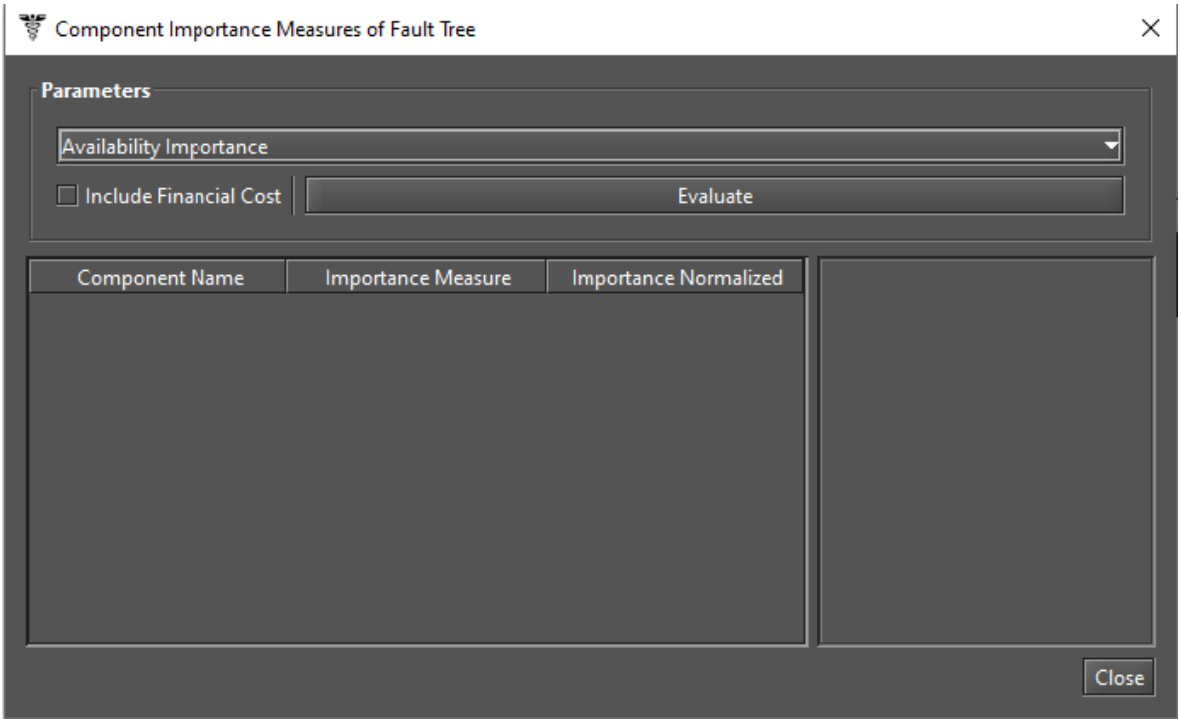


Figure 204: Component Importance Measures

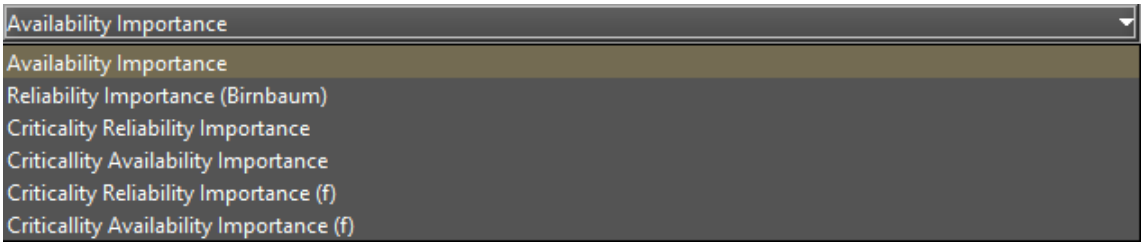


Figure 205: Metrics for Component Importance Measures

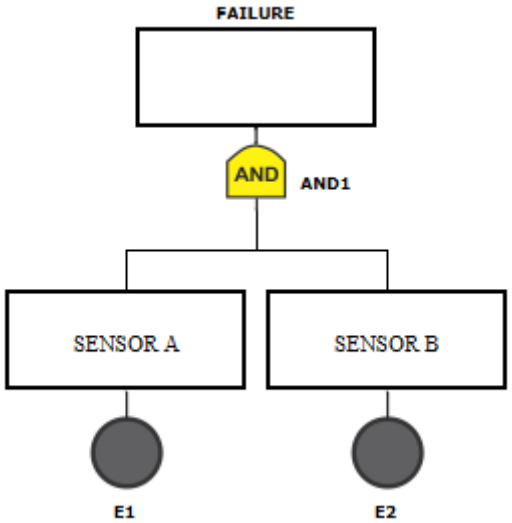


Figure 206: FT Model for Component Importance Evaluation

represented by a fire icon above the node, as mentioned earlier in this manual.

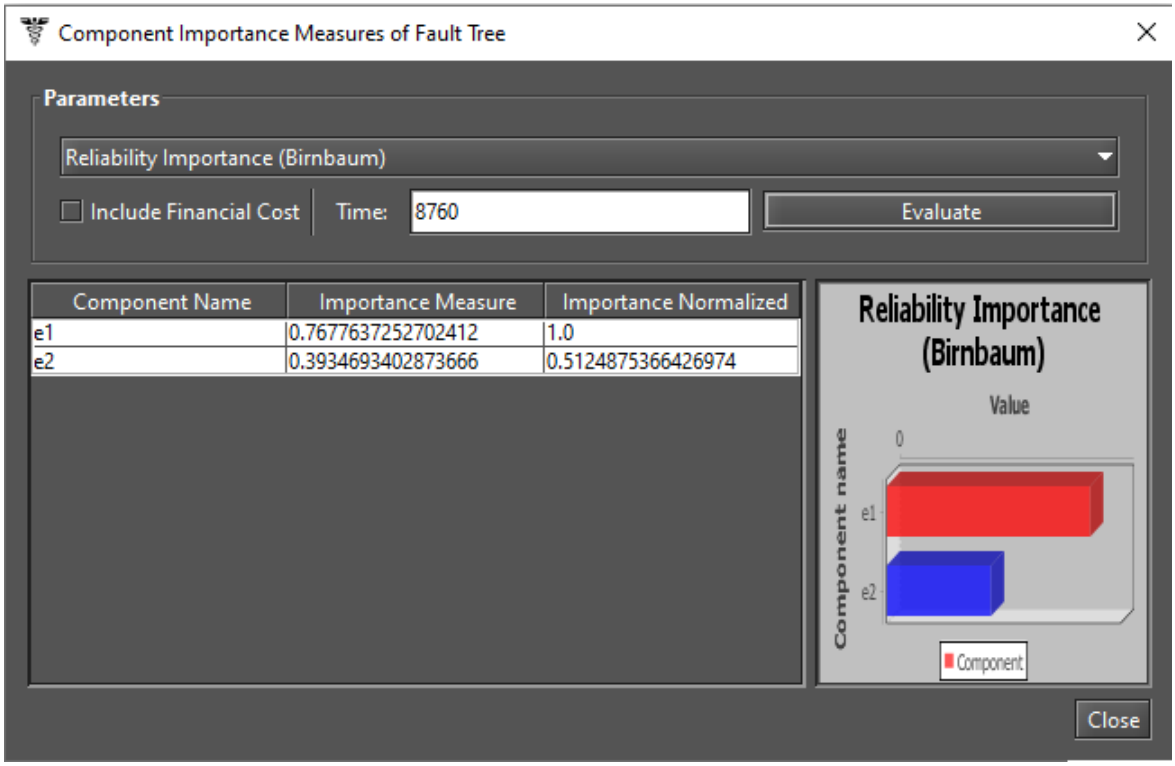


Figure 207: Reliability Importance Results

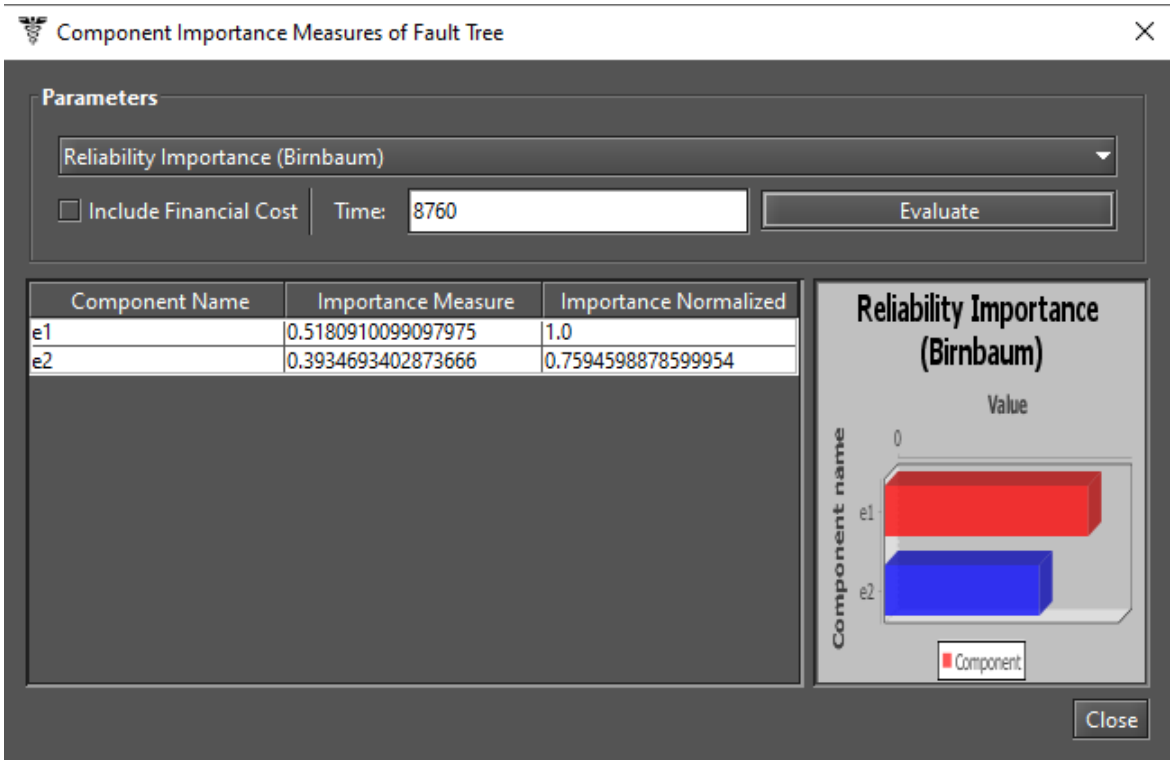


Figure 208: Reliability Importance when Replacing a Component

Let us now demonstrate how to obtain these functions using Mercury. Figure 210 shows a model with a AND gate and a OR gate. As we can see, there is a failed node in this model (event e3), and this node is a child of the OR gate.

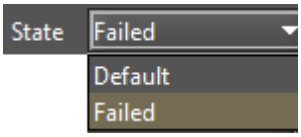


Figure 209: State of an Event

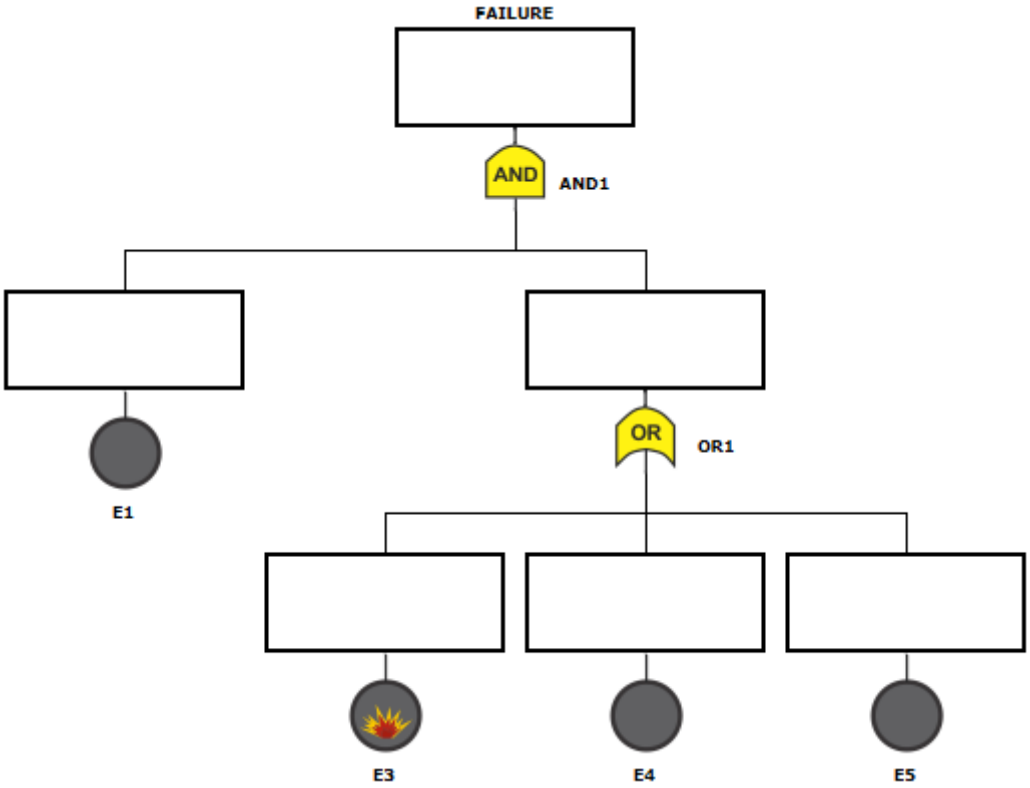


Figure 210: FT Model

Structural and logical functions can be accessed through the *Evaluate -> FT Evaluation -> Get Functions* menu. Figures 211 and 212 show the structural and logical functions, respectively, of the FT model shown above. In addition to the expressions, the tool displays the event nodes marked as faulty (non-functional) and the current state of the system. In our example, the faulty node (e3) has no effect on the state of the system.

On the other hand, we can see that changing the state of event node e1 to failed (see Figure 213) also changes the state of the system to failed (see Figure 214).

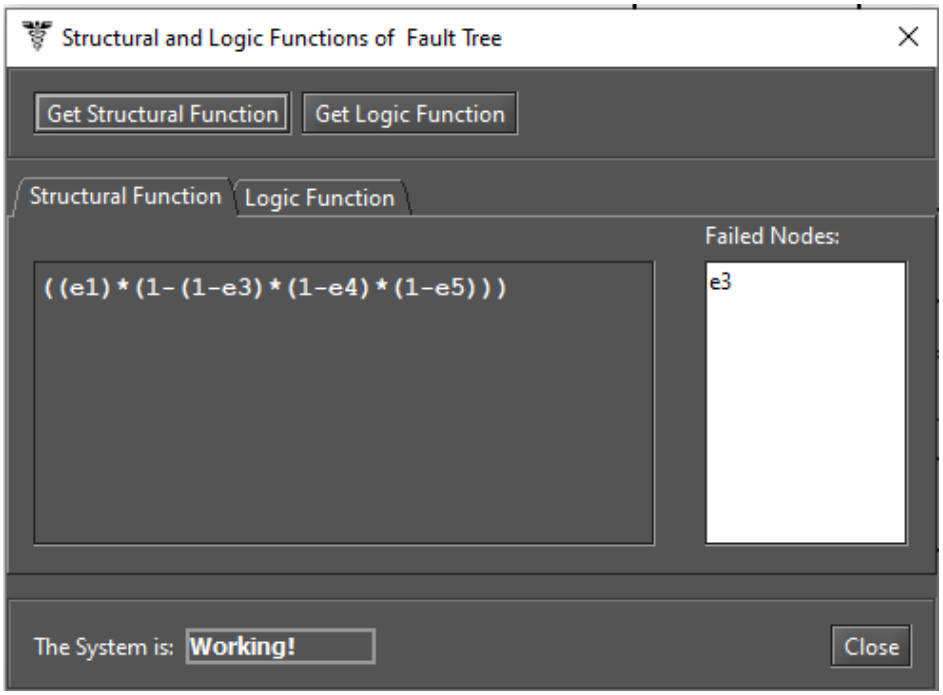


Figure 211: Structural Function

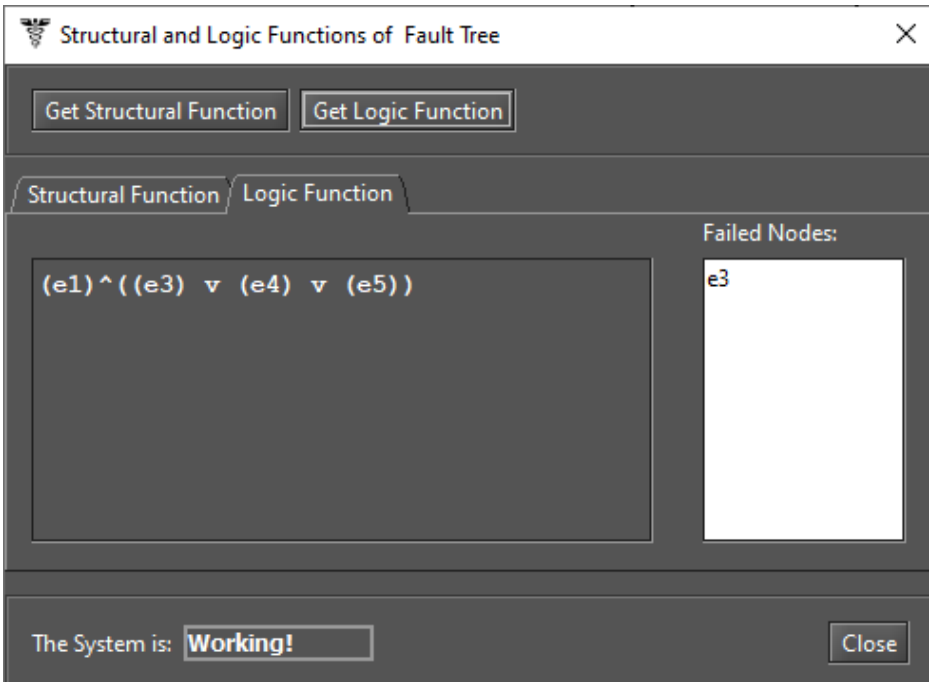


Figure 212: Logic Function

4.1.6 Sensitivity Analysis

Mercury calculates partial derivative sensitivity indices from FTs through sensitivity analysis. These indices indicate the impact of each input parameter on the availability of the model. Mercury provides two types of sensitivity analysis for FT models. The first type of sensitivity analysis considers the current values of the model's parameters and can be accessed from the *Evaluate -> FT Evaluation -> Sensitivity Analysis* menu. The second type of analysis considers min/max values for each parameter and supports the "Design of Experiments" (DoE)

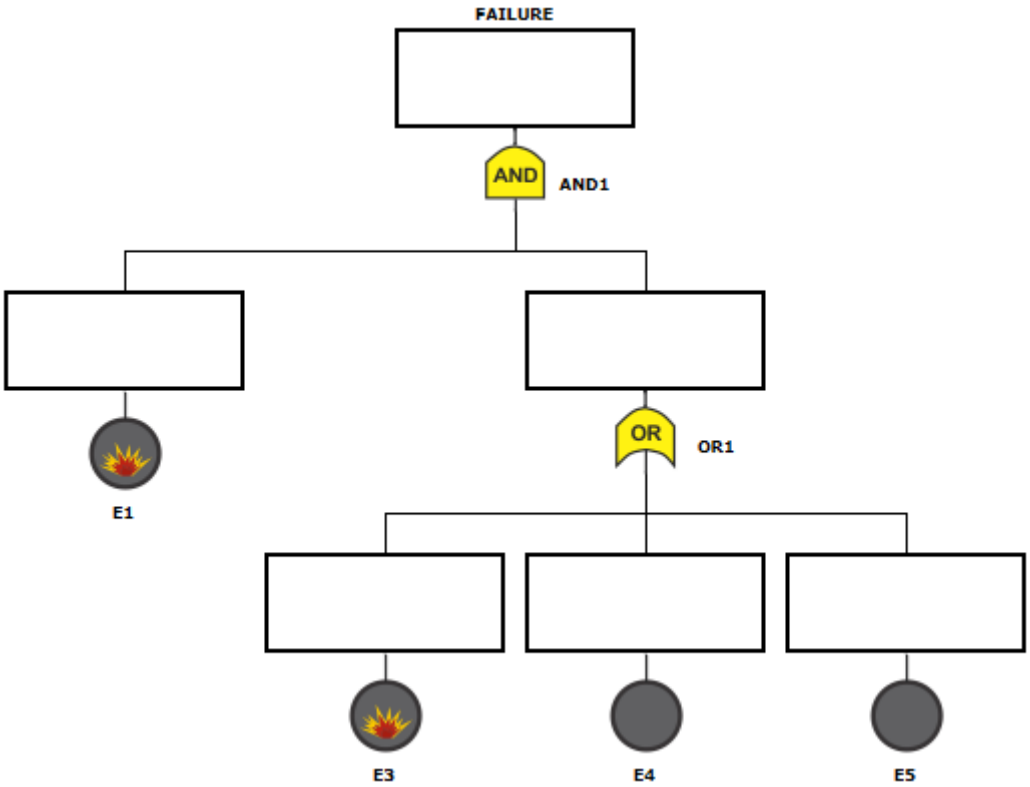


Figure 213: FT Model

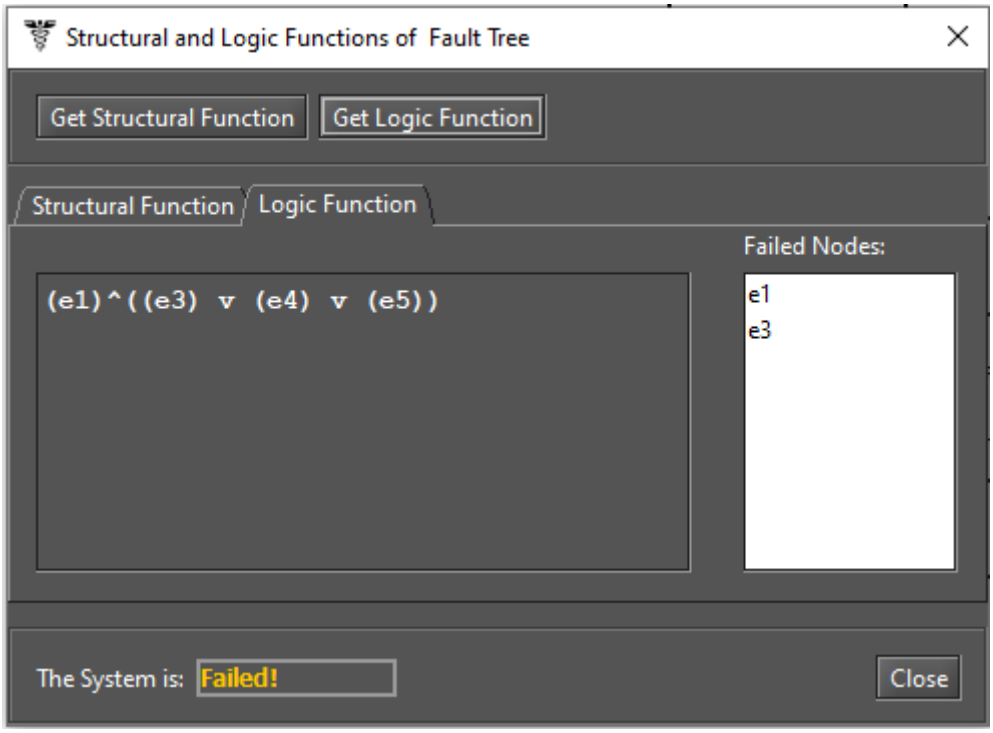


Figure 214: Logic Function and System State as Failed

method in addition to the “Sensitivity Indices” method. This second type of sensitivity analysis is shown in Section 2.5 and can be accessed from the menu *Evaluate -> FT Evaluation -> Sensitivity Analysis (min/max values)*. Figure 215 shows the “Sensitivity Analysis” window to perform sensitivity analysis considering the

current parameter values, displaying the partial derivative of the structural equation for each parameter and the sensitivity indices. It should be noted that both types of sensitivity analysis are only available when all event nodes of the model are exponential.

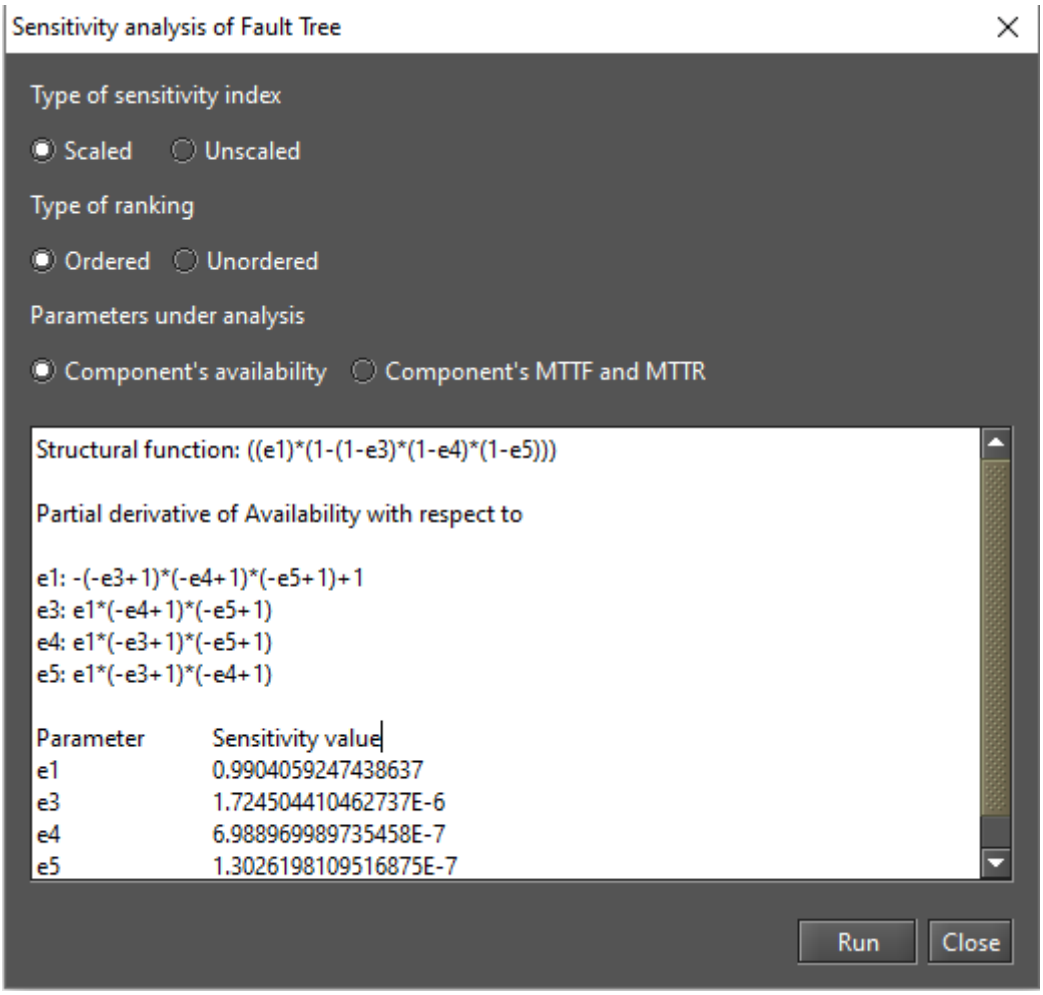


Figure 215: Sensitivity Analysis of Fault Tree

4.1.7 Export to RBD model

Users can convert FTs to RBDs. This is done via the menu *Evaluate -> FT Evaluation -> Export to RBD model*. The conversion process must be confirmed as shown in Figure 216. After that, the user must select the directory and enter the name of the file to be created.

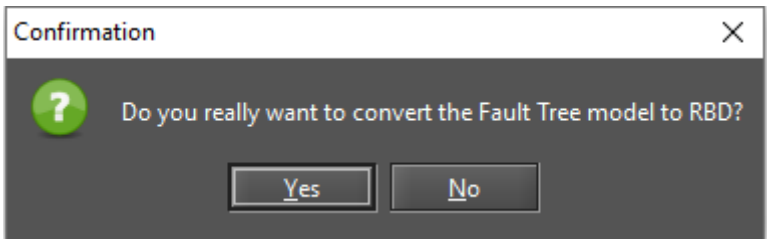


Figure 216: Converting FT to RBD

Figure 218 shows an RBD converted from the FT shown in Figure 217.

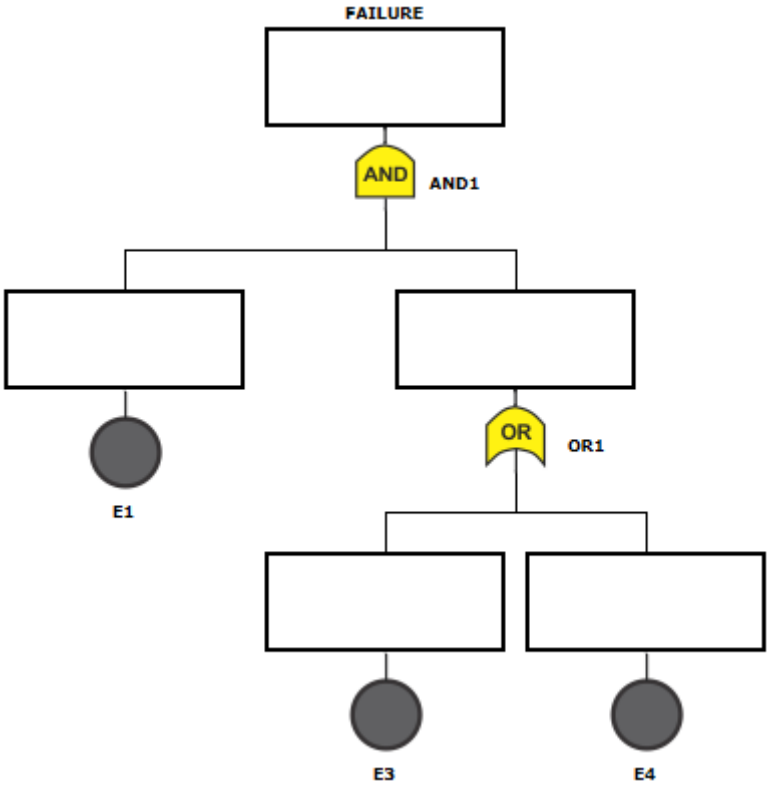


Figure 217: FT Model

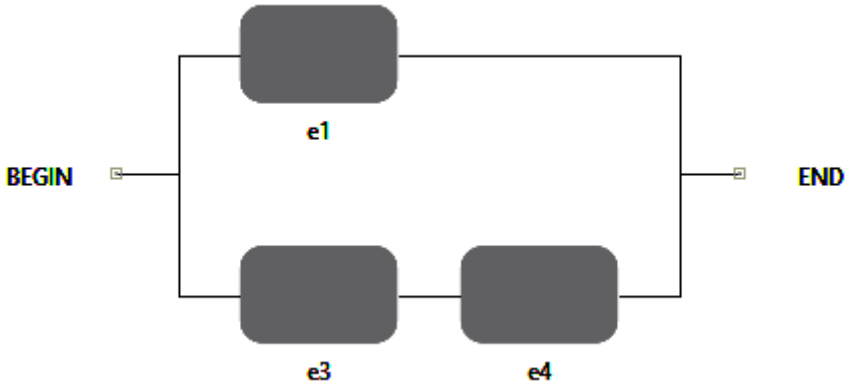


Figure 218: RBD Generated from a FT Model

5 CTMC Modeling and Evaluation

The first step to start modeling CTMC models on Mercury is to insert states into the graph. In the CTMC view, the user must click on the “State” button available in the toolbar (see Figure 219), and then click on the desired location in the drawing area to create a state there.

Figure 219: Adding a CTMC State

After adding states, the transitions between them are drawn by clicking on the center of the source state, only after the cursor turns into a hand symbol, and then drawing the line up to the target state, as shown in Figure 220. After that, a directed arc is created between the two states, as shown in Figure 221.

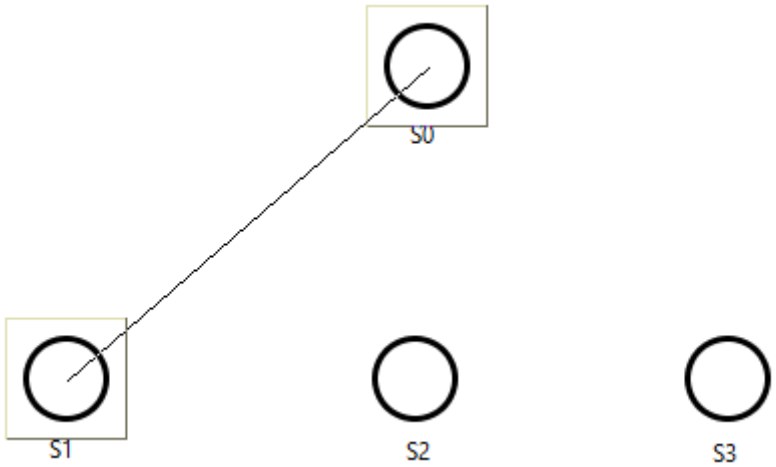


Figure 220: Adding a Transition Between States

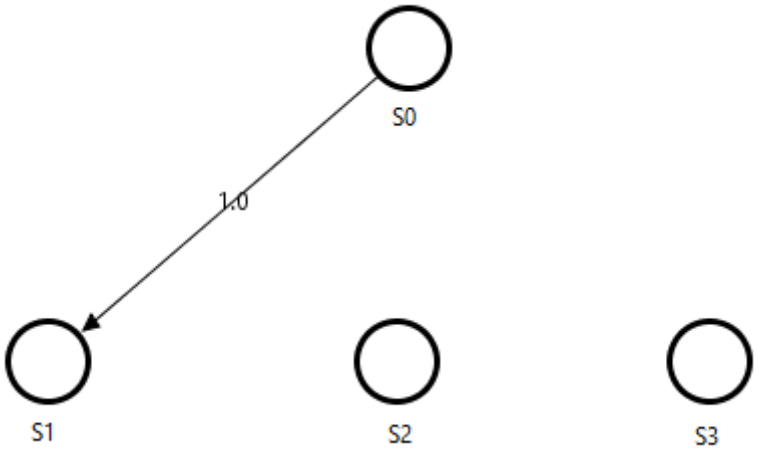


Figure 221: A Transition Between States

You can set the rate of each transition by double-clicking or right-clicking on the respective arc and selecting “Properties”. Figure 222 shows the window where the transition rate can be defined.

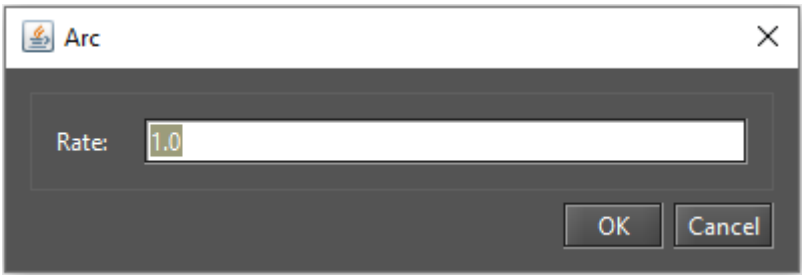


Figure 222: Defining a Rate for a State Transition

Mercury also allows us to assign reward rates to states. To do this, double-click or right-click on the selected state and select “Properties”. Figure 223 shows the “State” window where a reward rate can be assigned to a state. The default reward rate for each state is zero. You can enter any real value or an expression with user-defined parameters in this window. Also the name of the state can be changed.

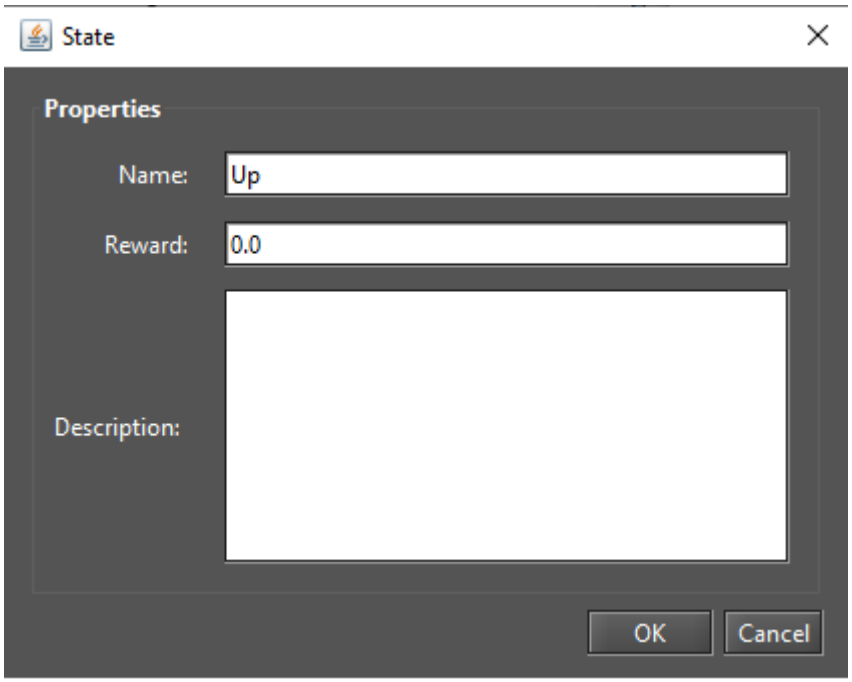


Figure 223: Properties of a State

After all states and transitions have been properly defined (see Figure 224), stationary and transient analyzes can be performed.

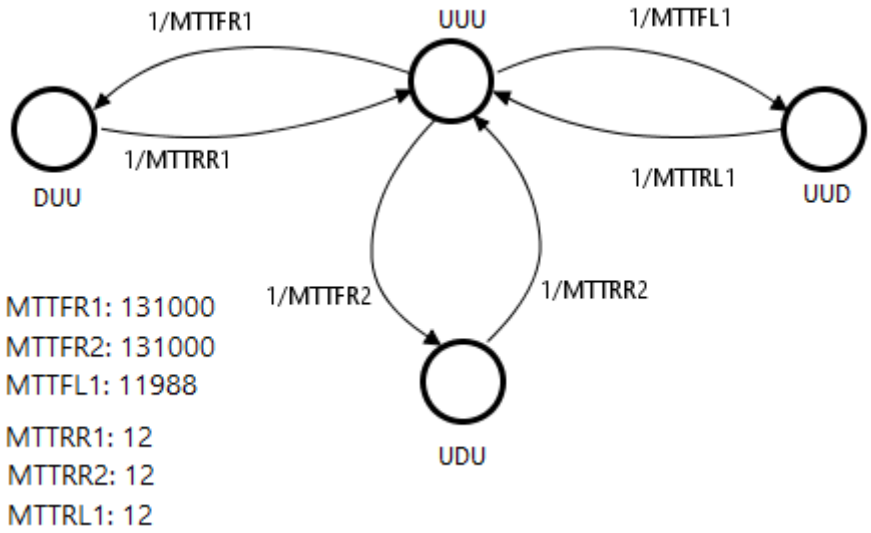


Figure 224: A CTMC Model

Users can export the infinitesimal generator matrix (transition rate matrix) of the model to a text file by clicking on the matrix icon in the toolbar, as shown in Figure 225.

Figure 225: Exporting the Transition Rate Matrix

Mercury has a feature to improve the usability of the tool. Once a CTMC component is inserted, you can read its properties on the drawing area by positioning the mouse pointer over it. A tooltip will then appear showing all the properties of the component. As you can see in Figure 226, a tooltip with the properties of a state is displayed.

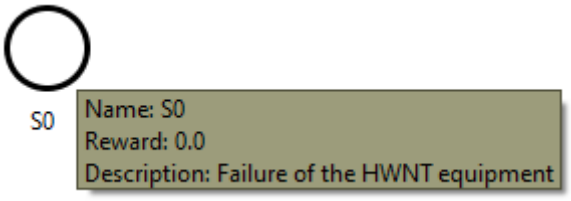


Figure 226: Tooltip for a CTMC State

5.1 Input Parameters/Definitions

Transition rates can be defined using expressions containing both numbers and user-defined parameters. The "Definition" button on the toolbar is represented by a λ symbol and creates a symbolic parameter (see Figure227).

Figure 227: Adding a CTMC Definition

After clicking on this button, the user must click on any point in the drawing area to place the definition there. This way a new parameter named **Param0** will be created (or **Param1**, and so on, if other parameters have already been created). By double-clicking the parameter or selecting "Properties" from the definition's popup menu, you can access its properties.

The name of the parameter can be defined by a combination of alphanumeric characters. Identifiers on Mercury must start with at least one alpha character. Special characters (e.g., a hyphen or an ampersand) are not allowed, except for underscores. If names with Greek letters are used, Mercury will convert them to the corresponding symbol of the lowercase Greek alphabet (see Figures 228 and 229). The value assigned to the parameter can be a numeric expression. Symbols or parameter names are not allowed in the value field.

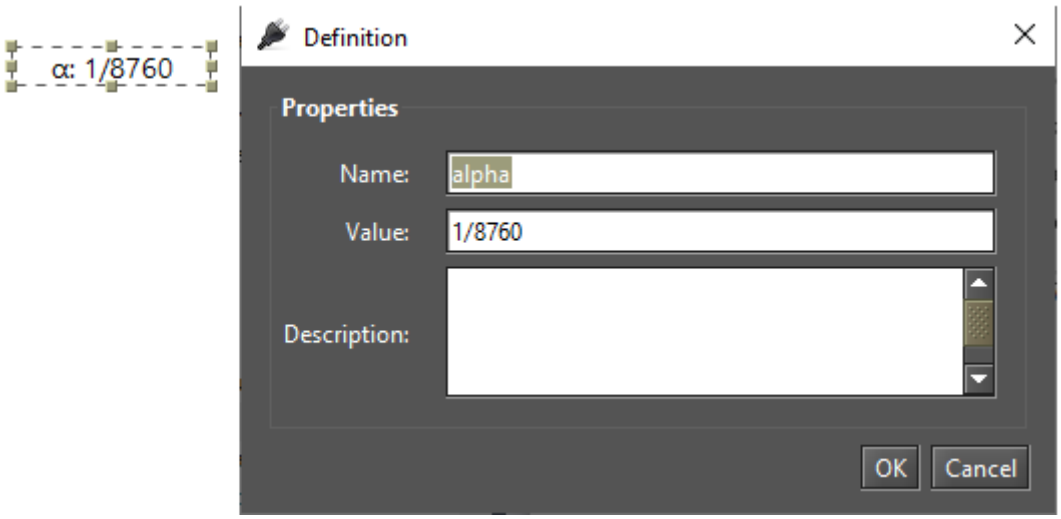


Figure 228: Modifying a CTMC Parameter

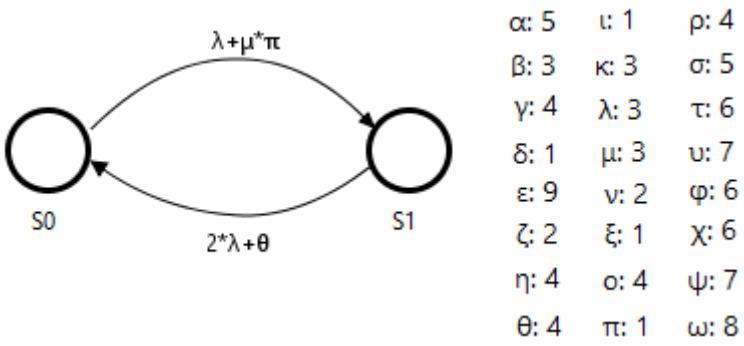


Figure 229: Parameters Named by Using Greek Letters

5.2 Metrics

Using the "Metric" button, we can define metrics to extract some characteristic of the model (see Figure 230).

Figure 230: Adding a CTMC Metric

Once a metric is inserted, users can change its name and description and define the expression used to calculate its value. The syntax for metric expressions is based on state probabilities (**P{*state name*}**), rewards (**R{*state name*}**), and base-10 logarithmic function (**LOG{*expression*}**). Expressions for probabilities, rewards, and logarithmic values for any states can be combined (i.e., added, subtracted, etc.). Using the example shown in Figure 231, the metric **AvB3** indicates the availability of the system (state "Up") represented by the two-state model. In this case, availability is calculated using the expression **P{Up}** — that is the probability of remaining in the state **Up**. Once the model has been evaluated using stationary or transient analysis, the metrics in the drawing area are updated accordingly (see Figure 232). Figure 233 shows an example of calculating the base-10 logarithm of the stationary probability of a given state. As we can see, by the expression $LOG\{1 - P\{Down\}\}$ we obtain the base 10 logarithm of the probability that the system is in the "Up" (operating) state, which gives $-7.232216190958877 \times 10^{-4}$.

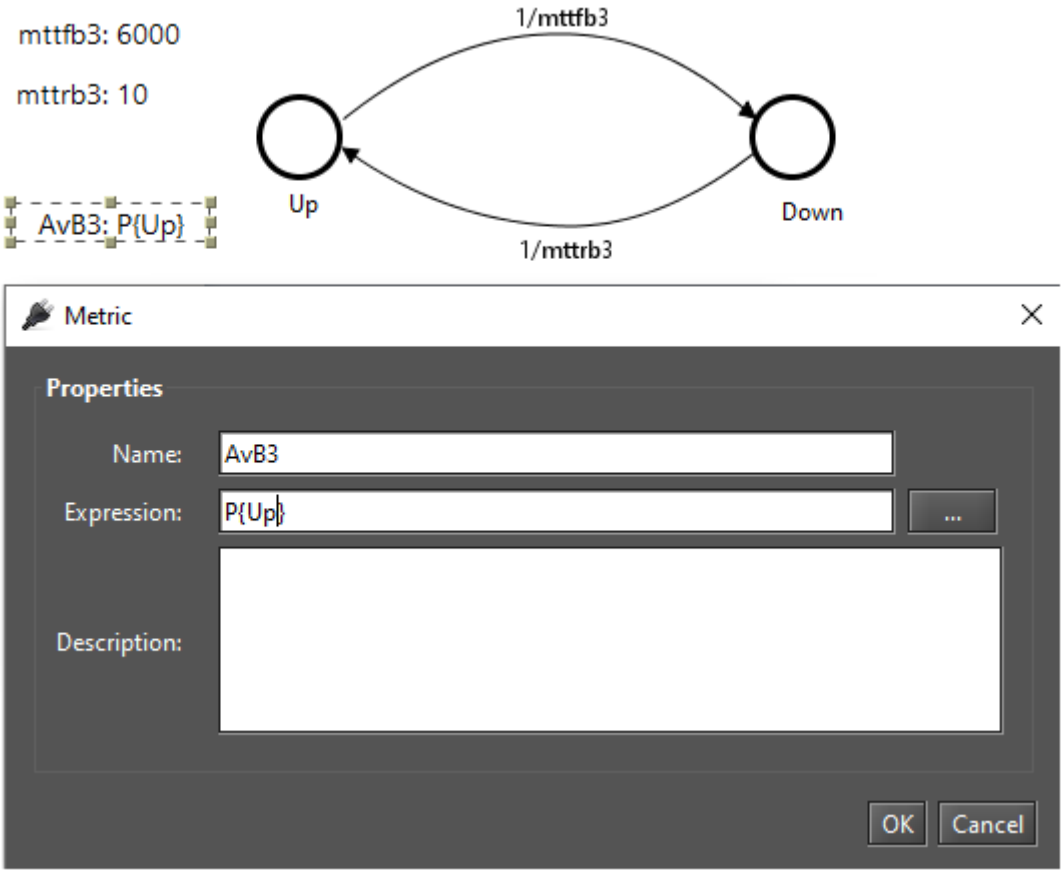


Figure 231: Defining Name and Expression for a Metric

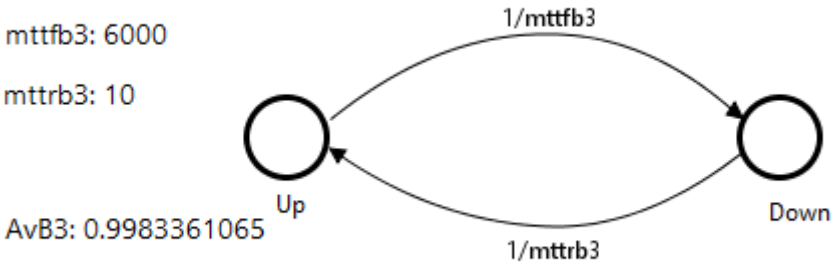


Figure 232: A CTMC Metric Solved

The reward rate can be determined using the metric expression **R{}**, and the mean time to absorption — if there is at least one absorbing state — can be calculated by defining a metric using the expression **MTTA**.

Metric expressions are still visible in the “Metrics” group on the left side of the CTMC tab (see Figure 234). This panel shows all the components that make up 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 associated with that transition.

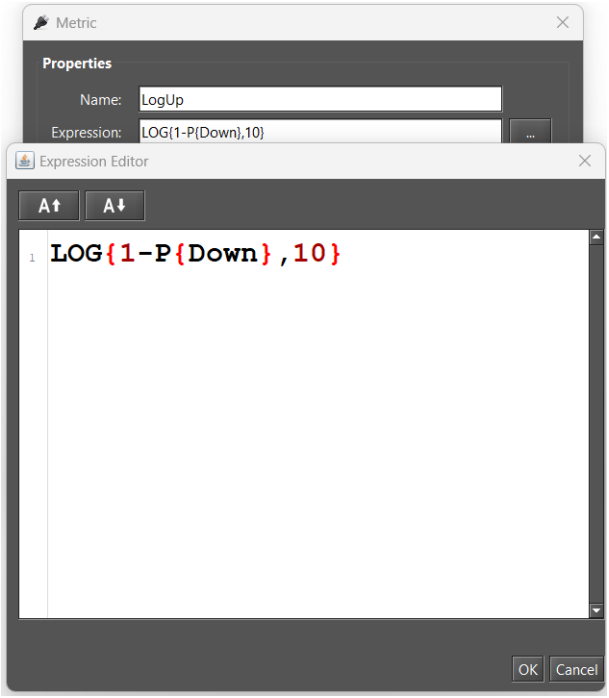


Figure 233: Using the LOG Function Expression

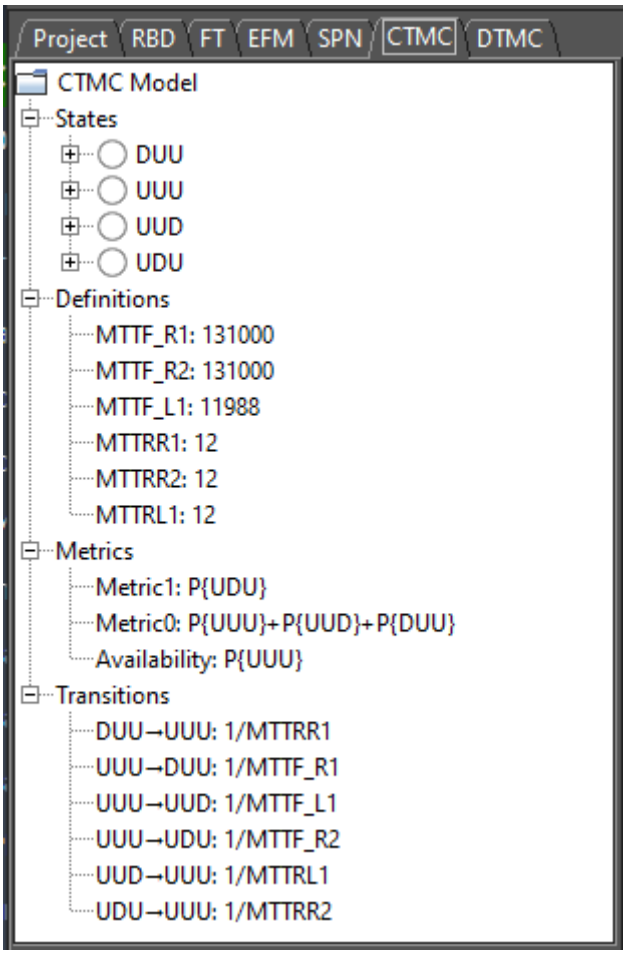


Figure 234: CTMC Panel on the Left Side of the Main Window

5.3 CTMC Evaluation

Mercury makes it possible to perform a large number of evaluations on CTMCs. The tool provides four functionalities for CTMC evaluations: “Stationary Analysis”, “Transient Analysis”, “Sensitivity Analysis”, and “Sensitivity Analysis (min/max values).” These evaluations are available from the *Evaluate -> CTMC Evaluation* menu (see Figure 235). In the next subsections, we will introduce each of them.

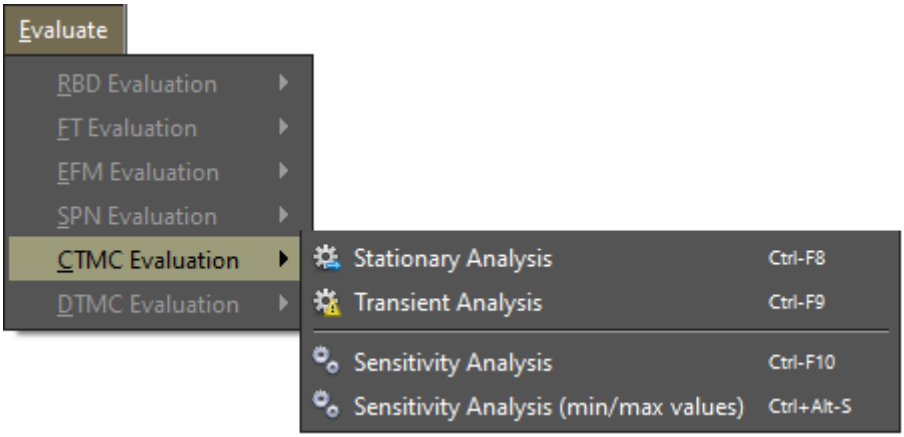


Figure 235: CTMC Evaluation Menu

5.3.1 CTMC Stationary Analysis

Stationary analysis calculates steady-state probabilities useful for evaluating the long-term average behavior of modeled systems. Figure 236 shows the “Stationary Analysis” window, which contains 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. The default value for the maximum error is 0.0000001 ($10^{-}7$). When you click the “Run” button, the solution algorithm is triggered. Once it is finished, the results are displayed in the text area at the bottom of the window (see Listing 7) and written to a plain text file, with the project file name appended with the suffix “-StationaryAnalysis.txt”.

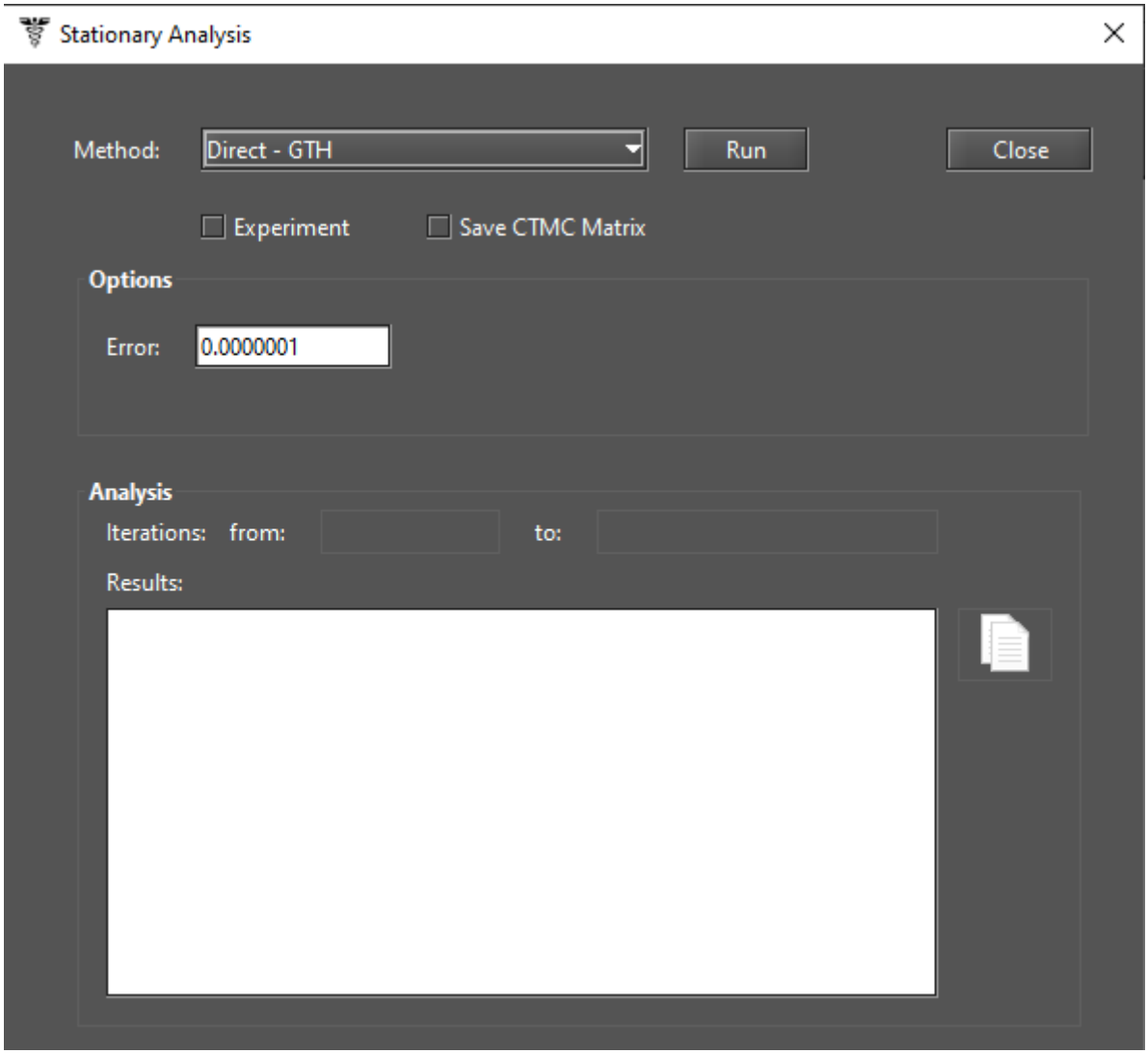


Figure 236: Stationary Analysis Window

Listing 7: Stationary Analysis for 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, it is possible to change not only the maximum error but also the maximum number of iterations. The default value for such a parameter is "-1", which means that the algorithm will not stop until the convergence of the results is reached taking into account the entered error (see Figure 237).

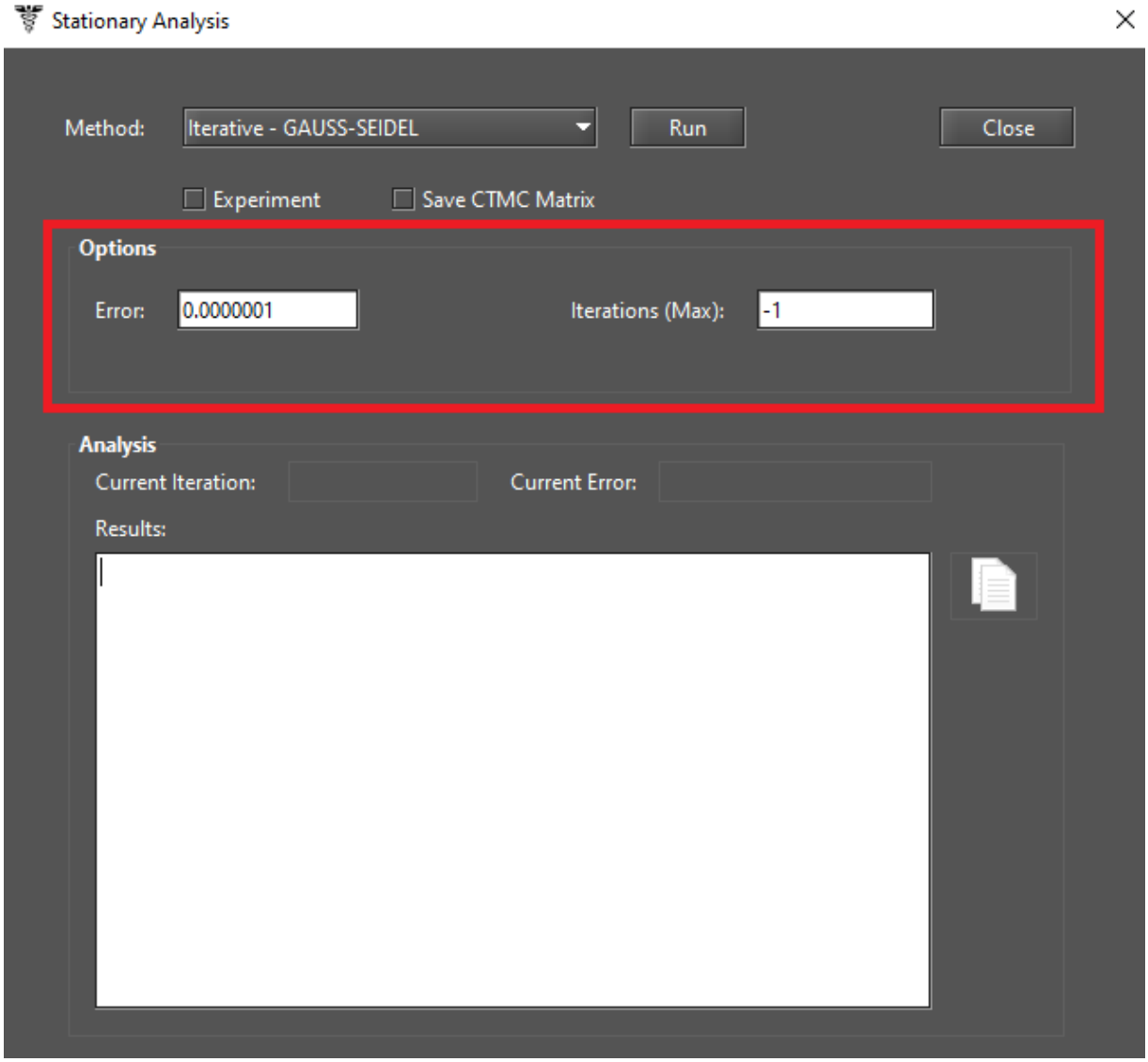


Figure 237: Stationary Analysis Window - Gauss-Seidel Method

Metrics are updated in the drawing area once the analysis is complete (see Figure 238, where the metric is located on the left side of the model: “Availability”).

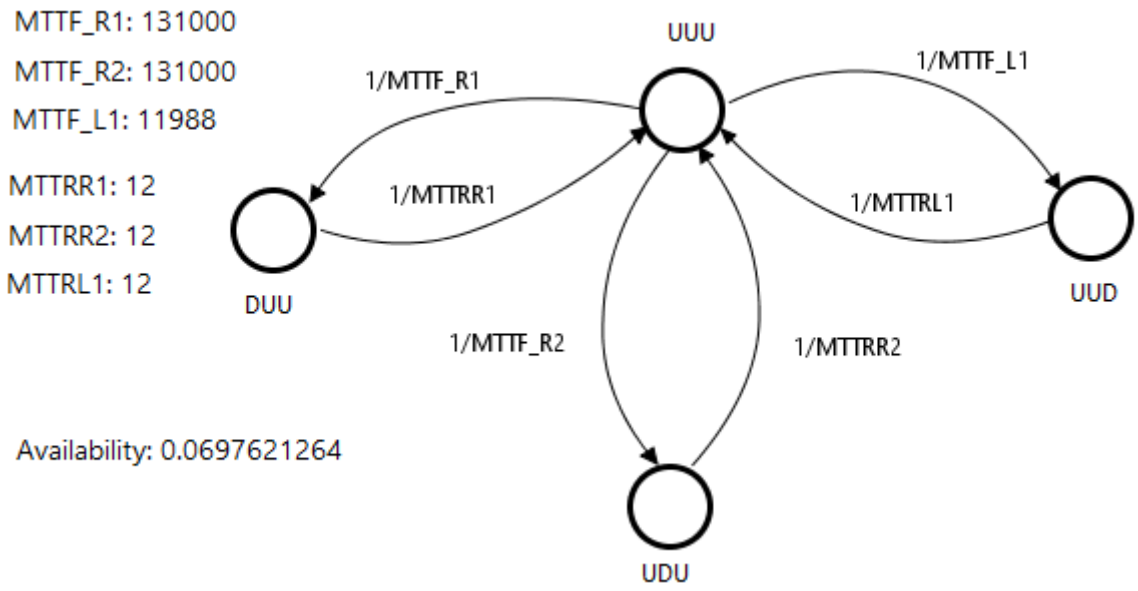


Figure 238: CTMC Metrics Updated

CTMC models can also be solved for a range of values of a user-defined parameter. To do this, check the "Experiment" option and click the "Run" button in the "Stationary Analysis" window. A new window will appear where the user can specify the input parameters for the experiment (see Figure 239).

Figure 239: CTMC Experiment

Below, we describe each field on this window.

- **Parameter.** Parameter to have its value changed.
- **Minimum Value.** Initial value to be assigned to the selected parameter.
- **Maximum Value.** Final value to be assigned to the selected parameter.

- **Type.** Determines whether the value of the parameter is changed linearly or logarithmically. If it is logarithmic, the parameter value is changed by a base-10 logarithmic function, taking into account the minimum and maximum values.
- **Interval.** This is the step size for changing the value of the parameter. The 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 is finished when the maximum value for the parameter is reached.
- **Metric.** Metric to be evaluated.

At the end of the experiment, the results are displayed and a graph is plotted, as we can see in the Figures 240 and 241. If you want to run another experiment in the same model, using the button “New Experiment" in 241 is an easier way to do this. You will return to the window "Options for Experiment" (Figure 239).

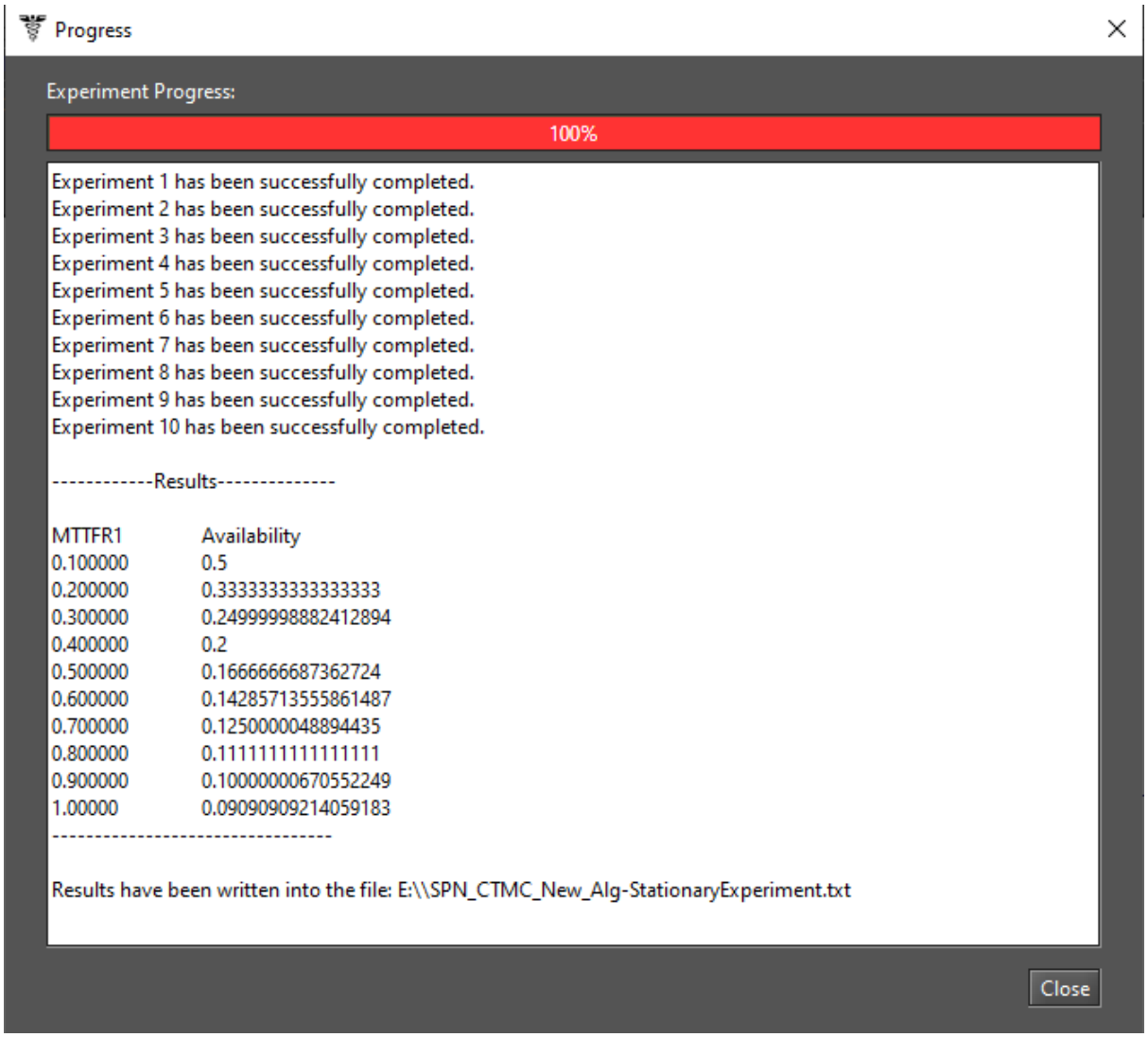


Figure 240: Results from a CTMC Experiment

Another option in the ”Stationary Analysis” window allows us to save the CTMC matrix to a file. It will be written to a plain text file, appending the name of the project file with the suffix ”-MatrixQ.txt”.

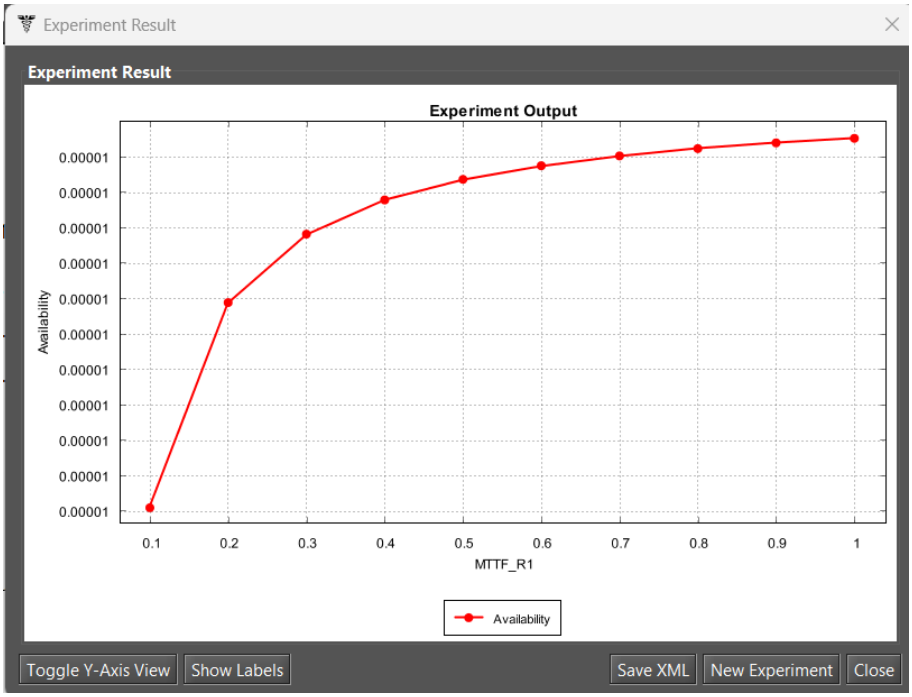


Figure 241: Graph from a CTMC Experiment

5.3.2 CTMC Transient Analysis

Transient analysis computes time-dependent probabilities useful for evaluating the behavior of modeled systems at a given time. Figure 242 shows the "Transient Analysis" window, which contains an input field for selecting one of the two available solution methods: **Uniformization** (also known as Jensen 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 the "Set Initial State Probability" button (see Figure 243).

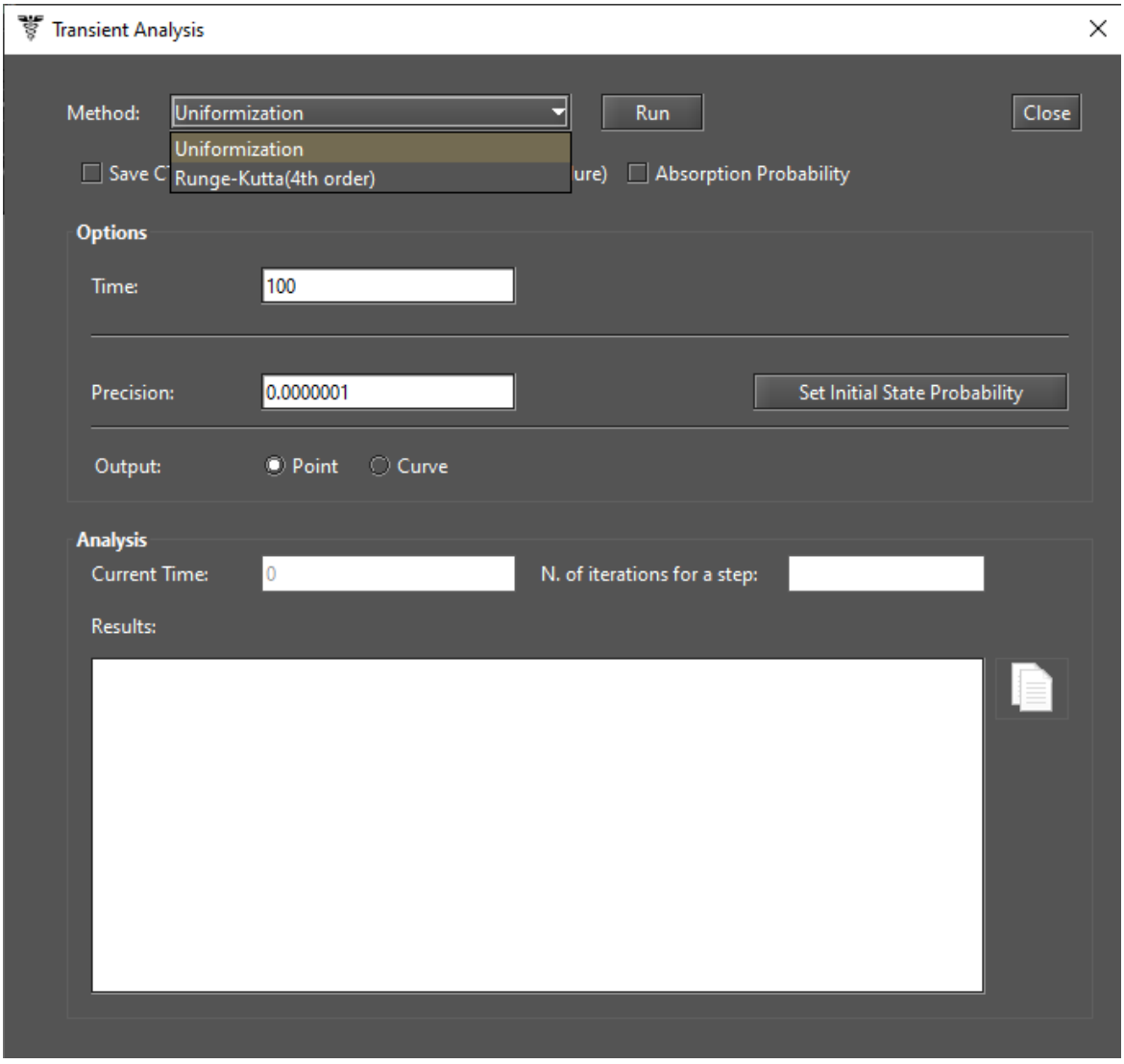


Figure 242: Transient Analysis Window

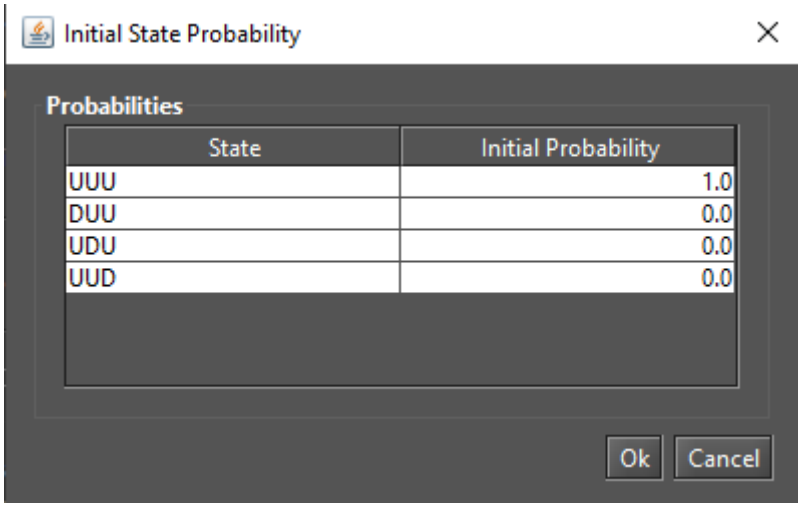


Figure 243: Initial State Probability Window

When selecting the Uniformization method, keep in mind that the time needed to obtain results is proportional to the time entered for the analysis, since Uniformization is an iterative algorithm.

When you click the “Run” button, the solution algorithm is started. Once it is completed, the results will be displayed in the text area at the bottom of the “Transient Analysis” window. Also, they will be written to a simple text file appending the project filename with the suffix “-TransientAnalysis.txt”.

In this window we can also choose between **Point** or **Curve** analysis. The **Point** analysis is the default and shows the state probabilities only for the given time. The **Curve** analysis writes all state probabilities calculated from time equal to zero to the specified time into a text file.

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

5.3.3 Sensitivity Analysis

Mercury calculates partial derivative sensitivity indices from CTMCs through sensitivity analysis. These indices indicate what effect each input parameter has on a metric. Mercury provides two types of sensitivity analysis for CTMC models. The first type of sensitivity analysis considers the current values of the model's parameters and can be accessed from the *Evaluate -> CTMC Evaluation -> Sensitivity Analysis* menu. The second type of analysis considers min/max values for each parameter and supports the "Design of Experiments" (DoE) method in addition to the "Sensitivity Indices" method. This second type of sensitivity analysis is shown in Section 2.5 and can be accessed from the menu *Evaluate -> CTMC Evaluation -> Sensitivity Analysis (min/max values)*. Next we demonstrate a sensitivity analysis considering the current parameter values. Figure 244 shows a CTMC representing the availability of a network with two routers and one link.

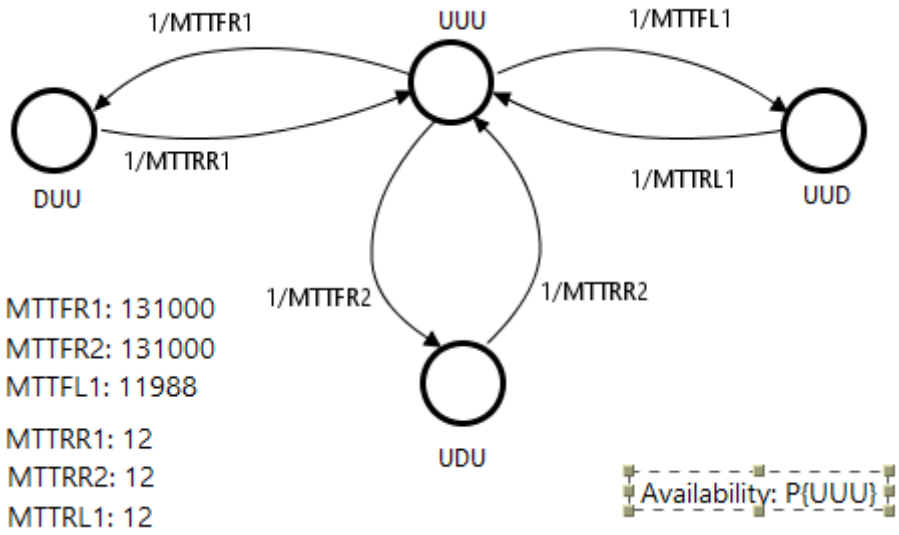


Figure 244: CTMC Model Representing a Computer Network

This model was proposed in [4] and has six parameters that affect system availability. These parameters are the 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 calculated using Mercury, as shown in Figure 245.

The "Sensitivity Analysis" window for the current parameter values has four options:

- **Type of sensitivity index** can be scaled or unscaled. If the user chooses scaled indices, each partial derivative is multiplied by the ratio between the respective parameter value and the metric value. This removes the influence of the parameter units and provides the sensitivity in a non-dimensional view. Unscaled indices are the raw results of the partial derivatives. For more details on scaled and unscaled indices, see [4] and [5].

- **Type of ranking** might be ordered or unordered. Typically, ordered rankings are preferred to quickly identify the most important parameters as well as those that 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 note that no sensitivity analysis can be performed if no measure has been defined. All measures can be evaluated at once.
- **Parameter of interest** can be any parameter in the model. The user can choose to see the sensitivity of the selected measure with respect to only one parameter or to all parameters.

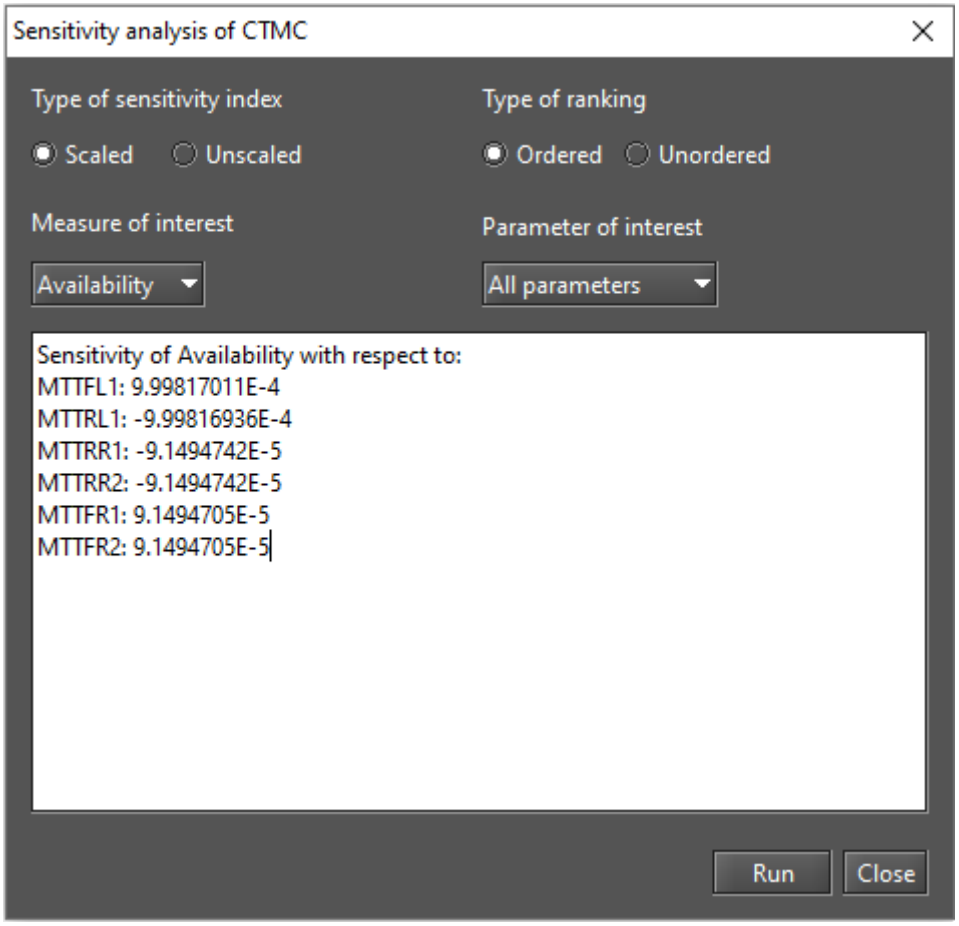


Figure 245: Results of Sensitivity Analysis for a CTMC

6 DTMC Modeling and Evaluation

The first step to start modeling DTMC models on Mercury is to insert states into the graph. In the DTMC view, the user must click on the “State” button available in the toolbar (see Figure 246), and then click on the desired location in the drawing area to create states there.

Figure 246: Adding a DTMC State

Transitions between them are drawn by clicking on the center of the source state after the cursor turns into a hand symbol, and then dragging the line to the target state, as shown in Figure 247. After that, a directed arc is created between the two states, as shown in Figure 248.

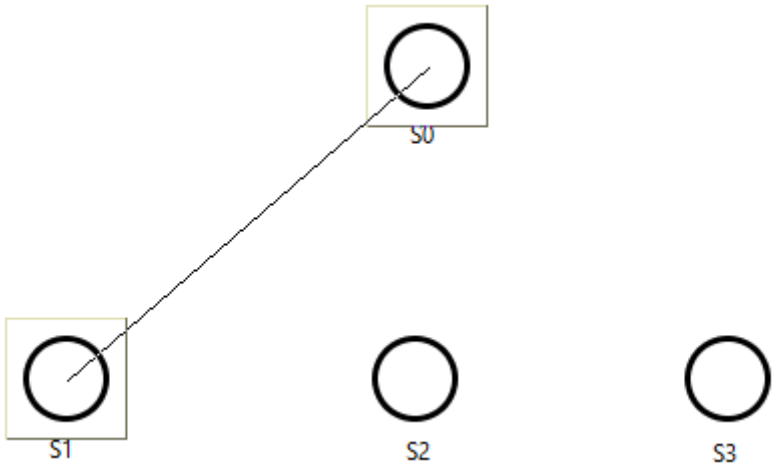


Figure 247: Adding a Transition Between States

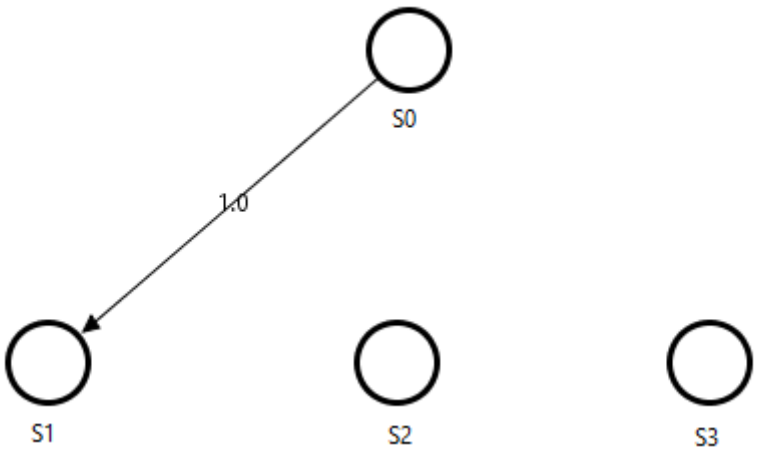


Figure 248: A Transition Between States

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 shown in Figure 249.

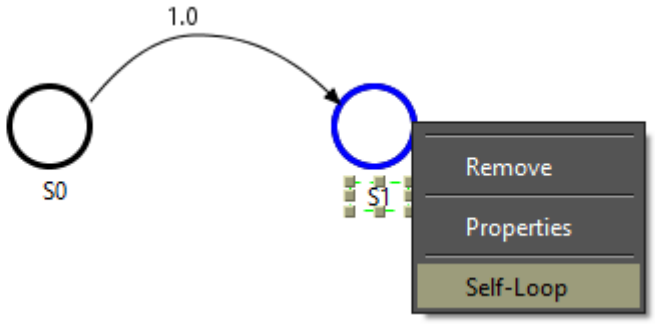


Figure 249: Defining a Self-Loop to a State

Then a self-loop arc is drawn for the state, as we can see in Figure 250. The left pane on the DTMC tab is updated accordingly.

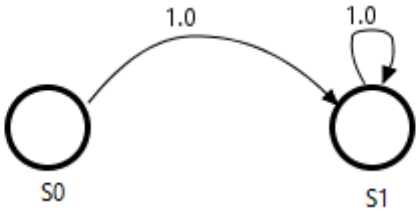


Figure 250: State with a Self-Loop Transition

The user can define the probability for each transition by double-clicking or right-clicking on the corresponding arc and selecting “Properties”. Figure 251 shows the window where a transition probability can be defined.

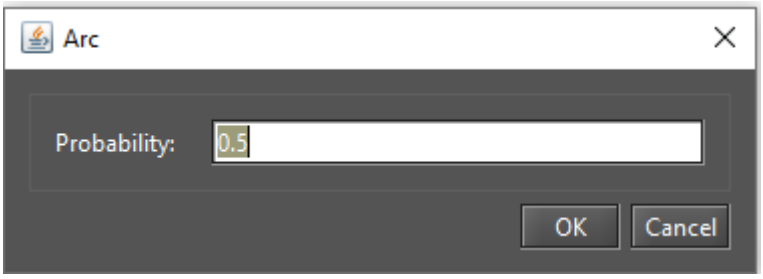


Figure 251: State Transition Probability

Figure 252 shows a DTMC model for which some parameters and a metric are defined. If you define all the states and transitions correctly, you can perform stationary and transient analyzes.

Users can export the probability matrix of a model to a text file by clicking the button represented by a matrix icon in the toolbar, as shown in Figure 253.

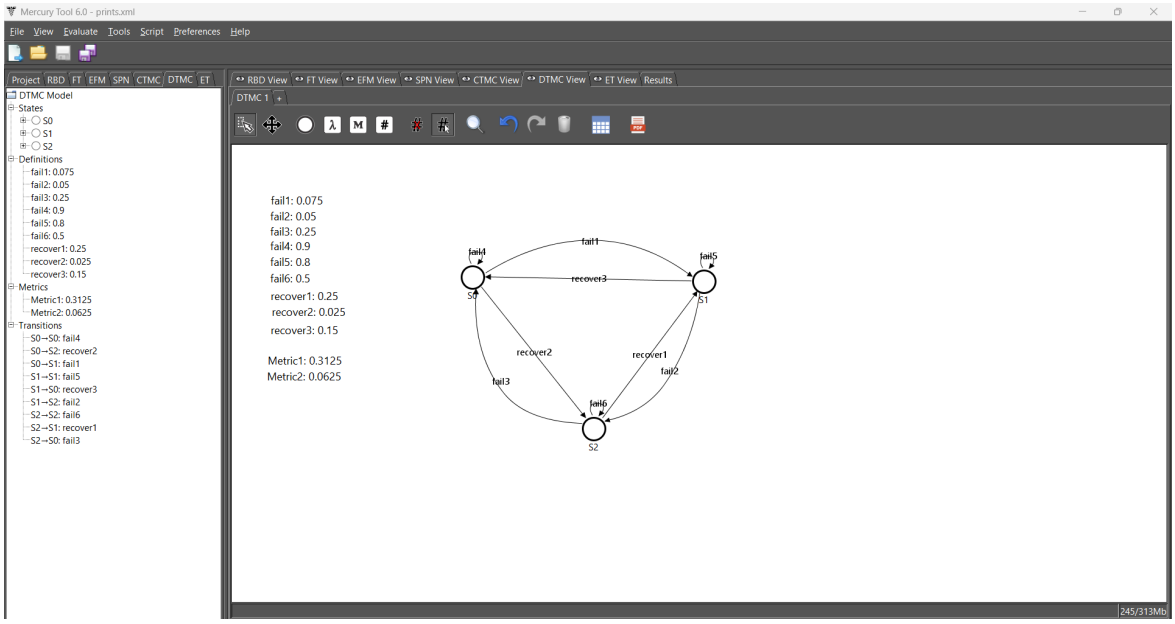


Figure 252: Example of DTMC Model

Figure 253: Exporting the DTMC Probability Matrix

It is worth noting that when modeling DTMCs, the sum of probabilities for all output arcs must equal one for each state. Otherwise, it is not possible to perform evaluations on the model. Figure 254 shows an error that occurs when this condition is not met.

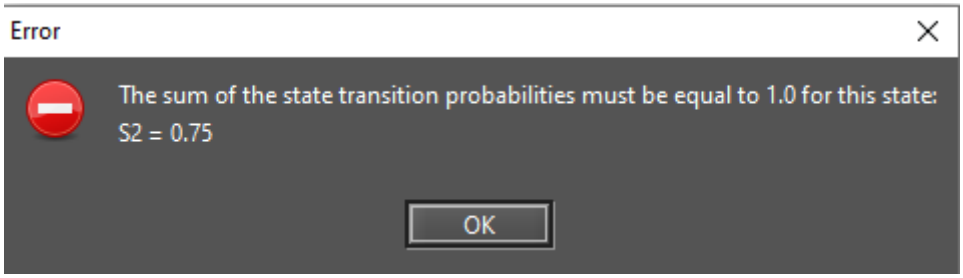


Figure 254: Transition Probabilities Error

Mercury has a feature to improve the usability of the tool itself. Once a DTMC component is inserted, you can read its properties on the drawing area by placing the mouse pointer over it. A tooltip will then appear showing all the properties of the component. As you can see in Figure 255, all properties of a state are displayed in the tooltip.

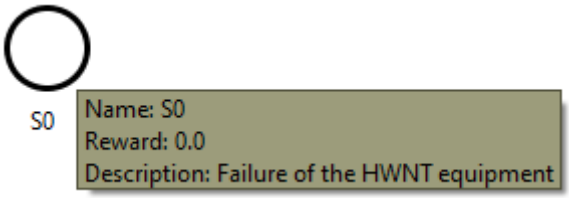


Figure 255: Tooltip for a DTMC State

6.1 Input Parameters

Transition probabilities can be defined using expressions containing both numbers and user-defined parameters. The "Definition" button on the toolbar is represented by a λ symbol and creates a symbolic parameter (see Figure256).

Figure 256: Adding a DTMC Definition

When we click this button, we must click on any point in the drawing area to put the definition there. This way a new parameter named **Param0** will be created (or **Param1**, and so on, if other parameters have already been created). By double-clicking the parameter or selecting "Properties" from the definition's popup menu, we can access its properties.

The name of the parameter can be defined by a combination of alphanumeric characters. Identifiers on Mercury must begin with at least one alpha character. Special characters (e.g. a hyphen or an ampersand) are not allowed, except underscores. If names with Greek letters are used, Mercury will convert them to the corresponding symbol of the lowercase Greek alphabet (see Figures 257 and 258). The value assigned to the parameter can be a numerical expression. Symbols or parameter names are not allowed in the value field.

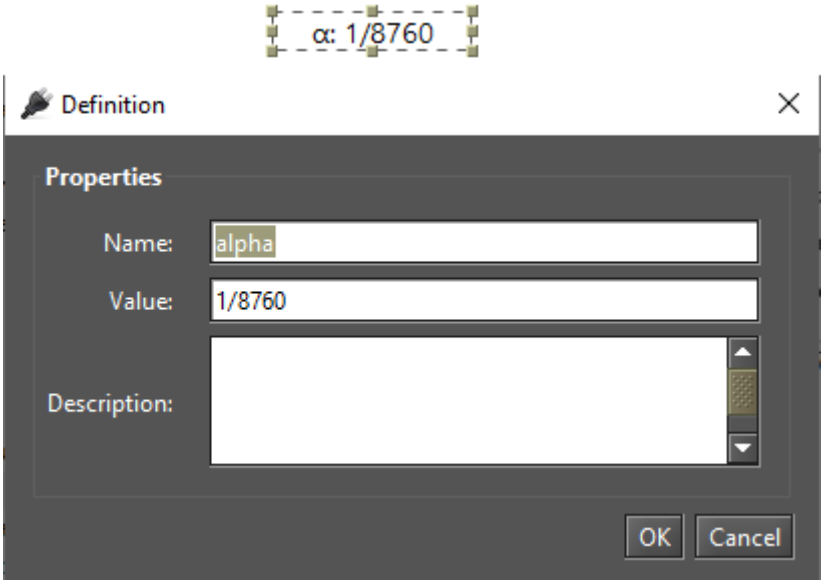


Figure 257: Modifying a DTMC Parameter

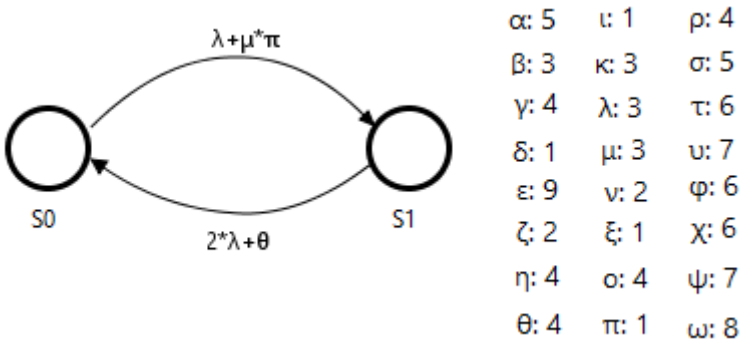


Figure 258: Parameters Named Using Greek Letters

6.2 Metrics

Using the “Metrics” button, we can define metrics to evaluate some characteristics of the model (see Figure 259).

Figure 259: Adding a DTMC Metric

Once you have inserted a metric, you can change its name and description and define the expression used to calculate its value. The syntax for metric expressions is based on state probabilities (**P{*state name*}**), rewards (**R{*state name*}**), and base-10 logarithmic function (**LOG{*expression,10*}**). Expressions for probabilities, rewards, and logarithmic values for any states can be combined (i.e., added, subtracted, etc.). Using the example shown in Figure 260, the metric **Prob** indicates the probability that the system remains in the states “S0”, “S1”, and “S2”, which is “1”.

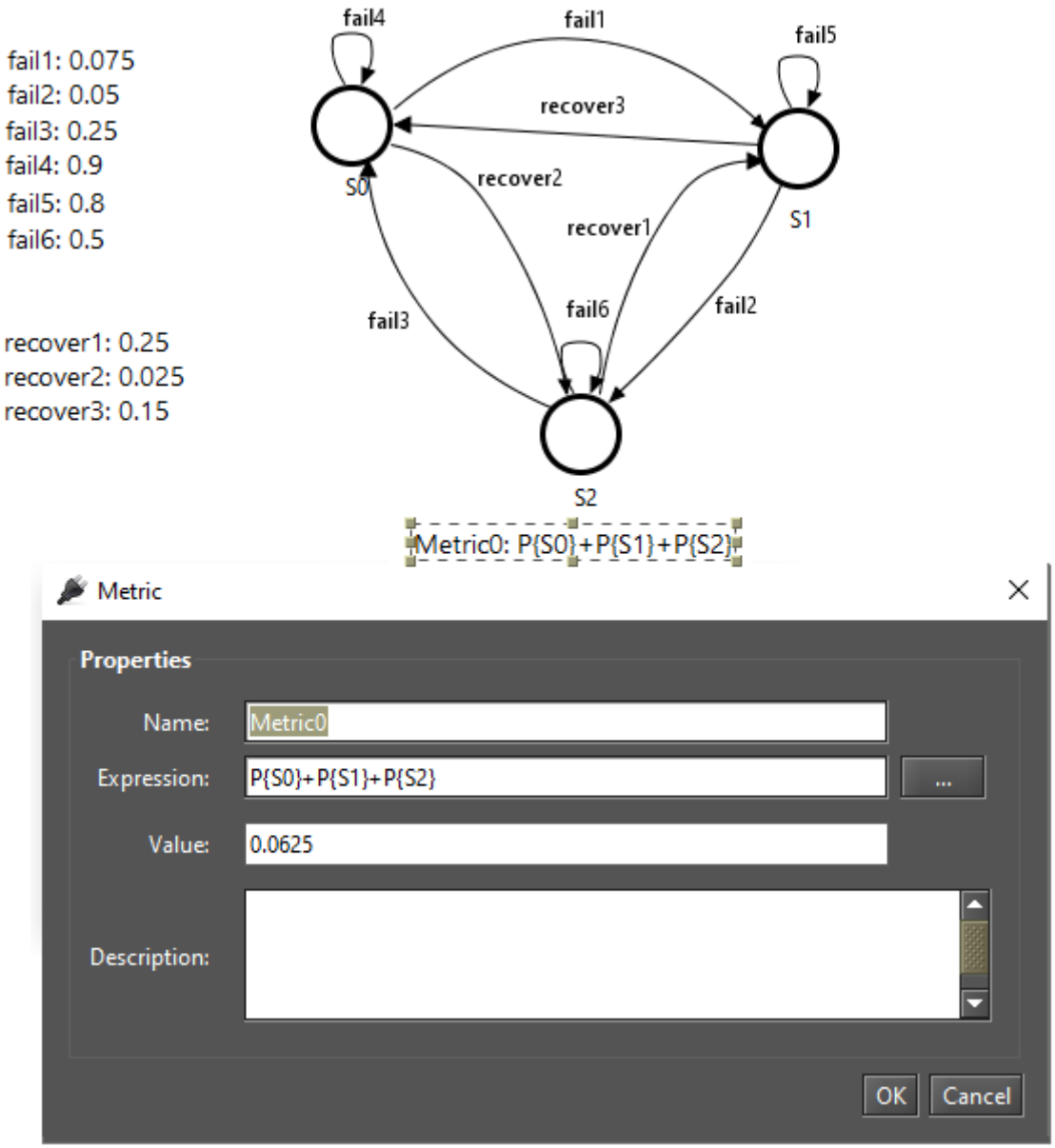


Figure 260: Defining Name and Expression for a Metric

Prob is calculated by the expression "**P{ SO }+P{S1}+P{S2}**" — it represents the probability of remaining in the states "SO", "S1" and "S2". Once the model has been evaluated using stationary or transient analysis, the metrics in the drawing area are updated accordingly (see Figure 261).

Metric expressions are still visible in the "Metrics" group in the left panel on the DTMC tab (see Figure 262). This panel shows all the components that make up the DTMC model: states, parameters, metrics, and transitions. State transitions are represented by the source and target states, followed by the expression or value associated with that transition.

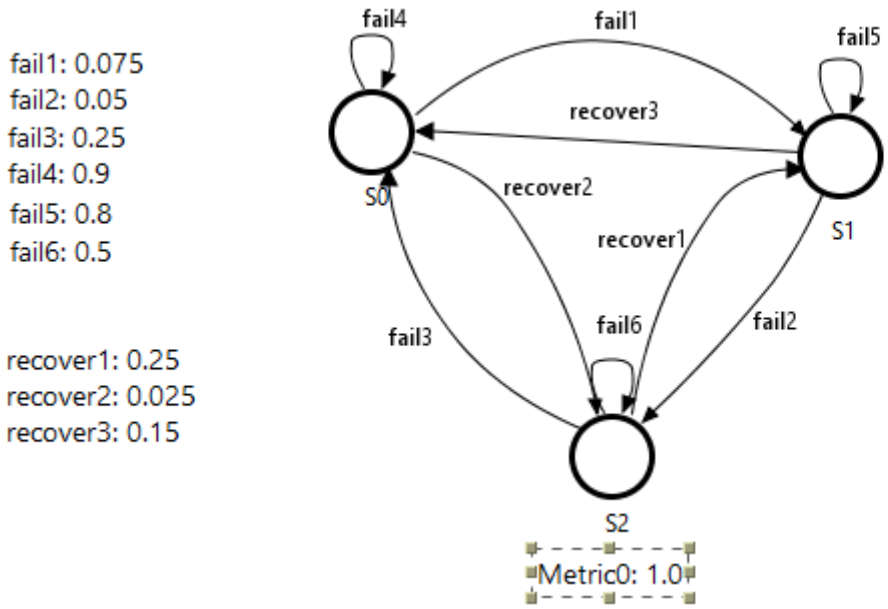


Figure 261: A Metric Solved

6.3 DTMC Evaluation

Mercury makes it possible to perform 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 from the *Evaluate -> DTMC Evaluation* menu (see Figure 263). In the next subsections we will introduce each evaluation.

6.3.1 DTMC Stationary Analysis

Stationary Analysis computes steady-state probabilities useful for evaluating the long-term average behavior of modeled systems. Figure 264 shows the “Stationary Analysis” window, which contains 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. The default value for the maximum error is 0.0000001 (10^-7). When you click the “Run” button, the solution algorithm is triggered. Once it is finished, the results are displayed in the text area at the bottom of the window (see Listing 8) and written to a plain text file, appending the project file name with the suffix “-StationaryAnalysis.txt”.

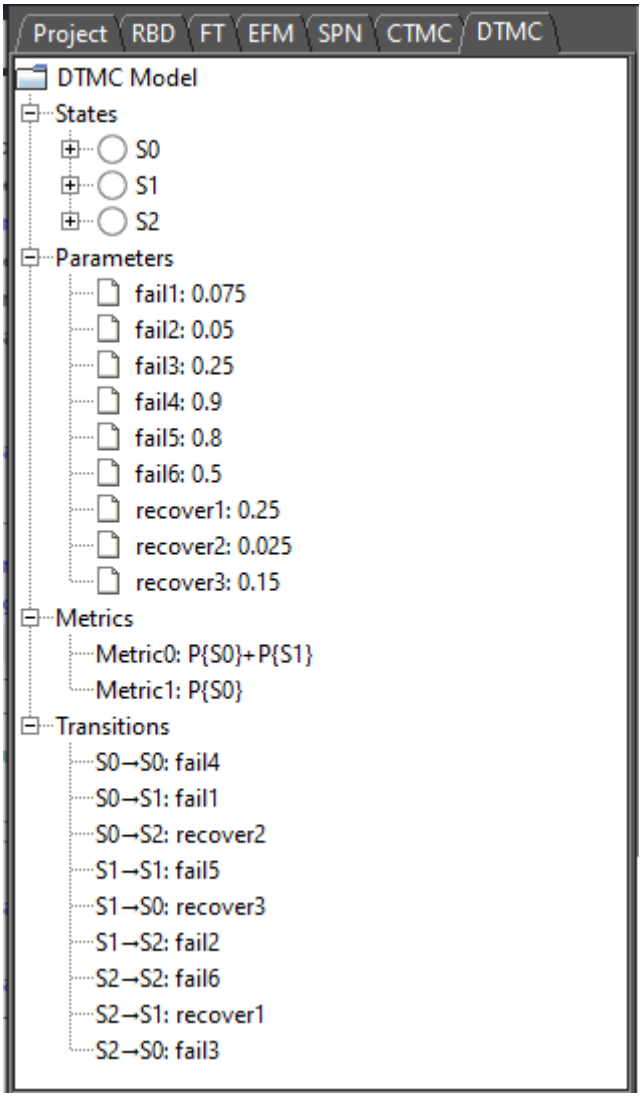


Figure 262: DTMC Panel on the Left Side of the Main Window

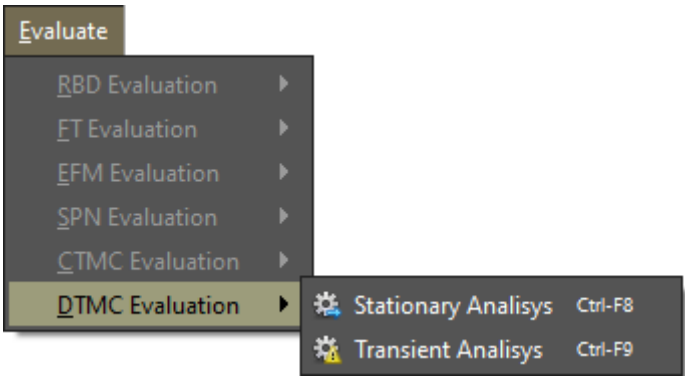


Figure 263: DTMC Evaluation Menu

Listing 8: Stationary Analysis of a DTMC

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

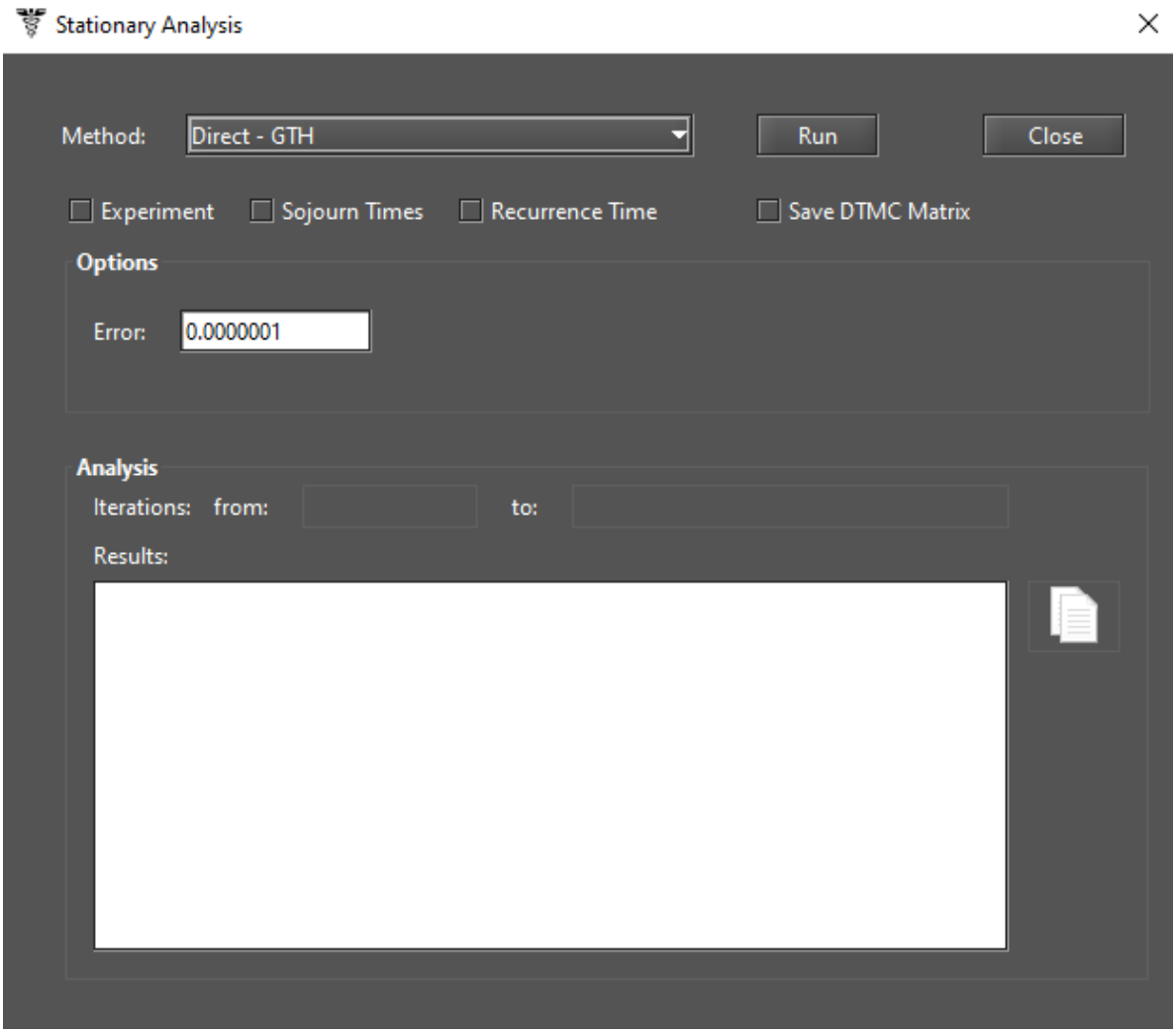


Figure 264: Stationary Analysis Window

```
##############################
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 by Gauss-Seidel, it is possible to change not only the maximum error but also the maximum number of iterations. The default value for such a parameter is "-1", which means that the algorithm will not stop until the convergence of the results is reached taking into account the entered error (see Figure 265).

Metrics are updated in the drawing area once the analysis is complete.

DTMC models can also be solved for a range of values of a user-defined parameter. To do this, check the

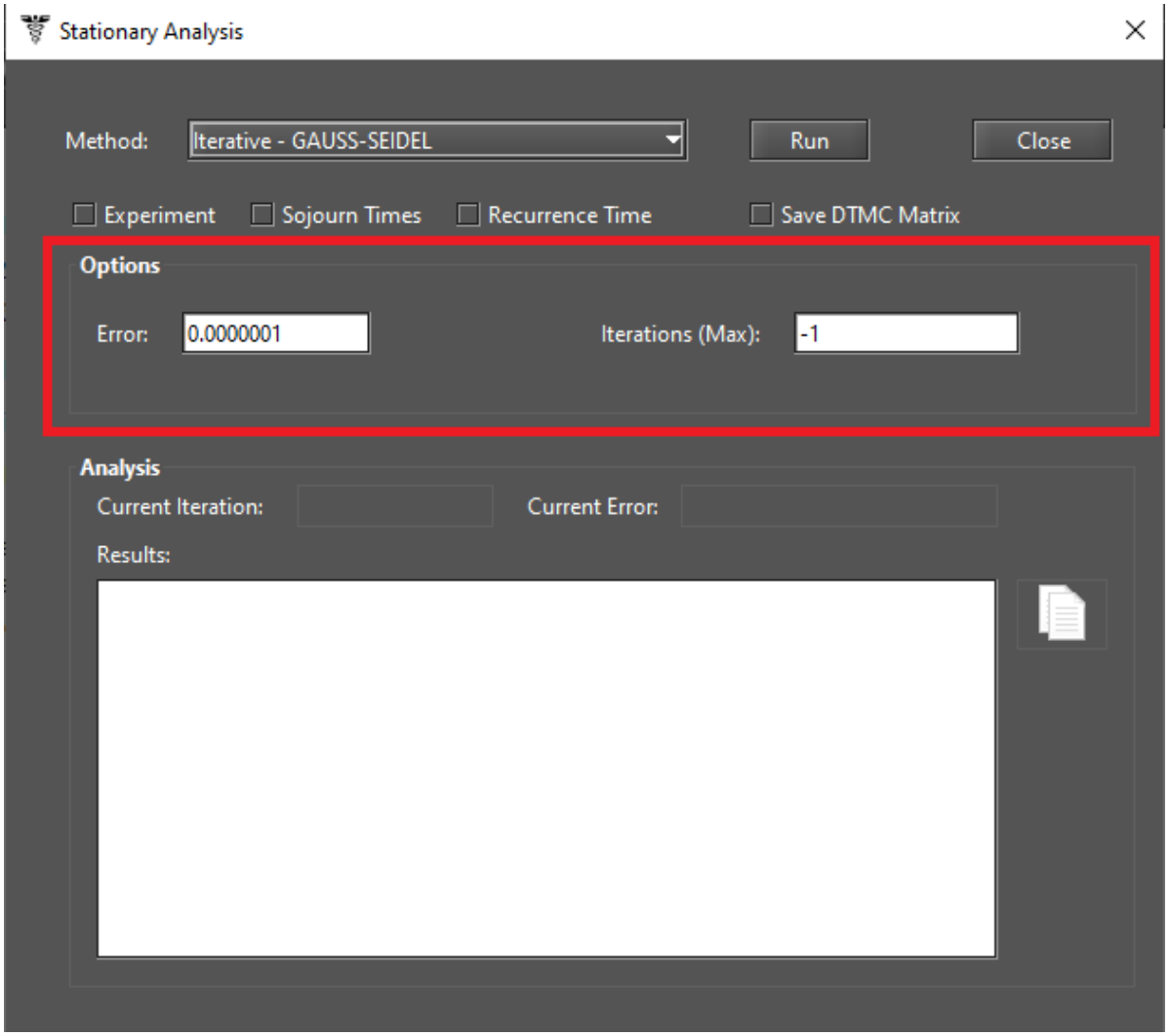


Figure 265: Stationary Analysis Window - Gauss-Seidel Method

"Experiment" option and click the "Run" button. A new window will appear where the user can specify the input parameters for the experiment (see Figure 266).

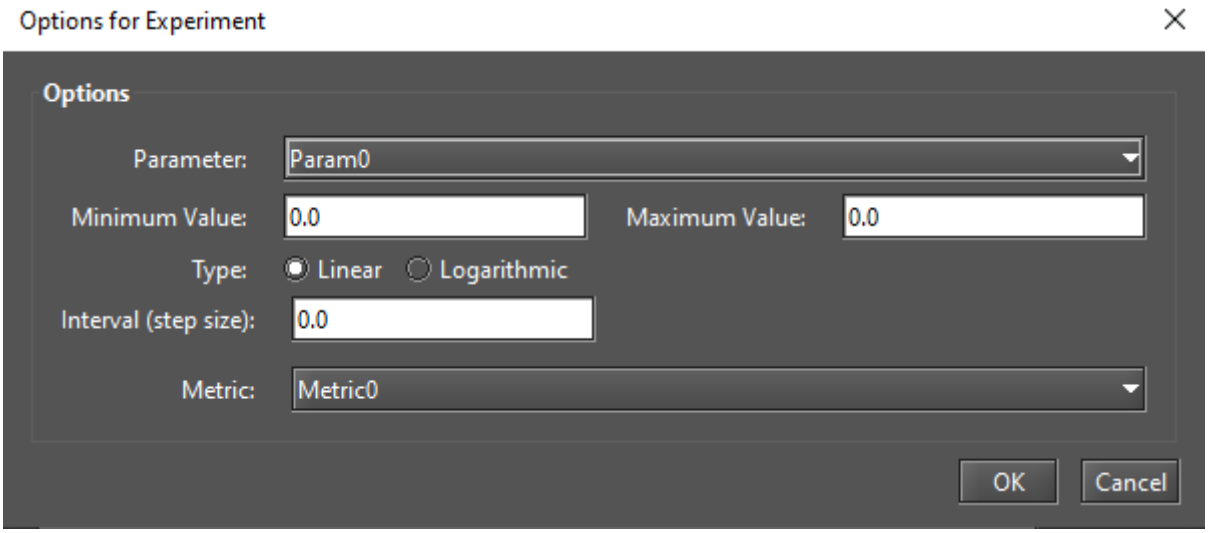


Figure 266: DTMC Experiment

Below, we describe each field on this window.

- **Parameter.** Parameter to have its value changed.
- **Minimum Value.** Initial value to be assigned to the selected parameter.
- **Maximum Value.** Final value to be assigned to the selected parameter.
- **Type.** Determines whether the value of the parameter is changed linearly or logarithmically. If it is logarithmic, the parameter value is changed by a base-10 logarithmic function, taking into account the minimum and maximum values.
- **Interval.** This is the step size for changing the value of the parameter. The 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 is finished when the maximum value for the parameter is reached.
- **Metric.** Metric to be evaluated.

At the end of an experiment, a graph is generated and the results are displayed, as we can see in the Figures 267 and 268. If you want to run another experiment in the same model, using the button “New Experiment" in 267 is an easier way to do this. You will return to the window "Options for Experiment" (Figure 266).

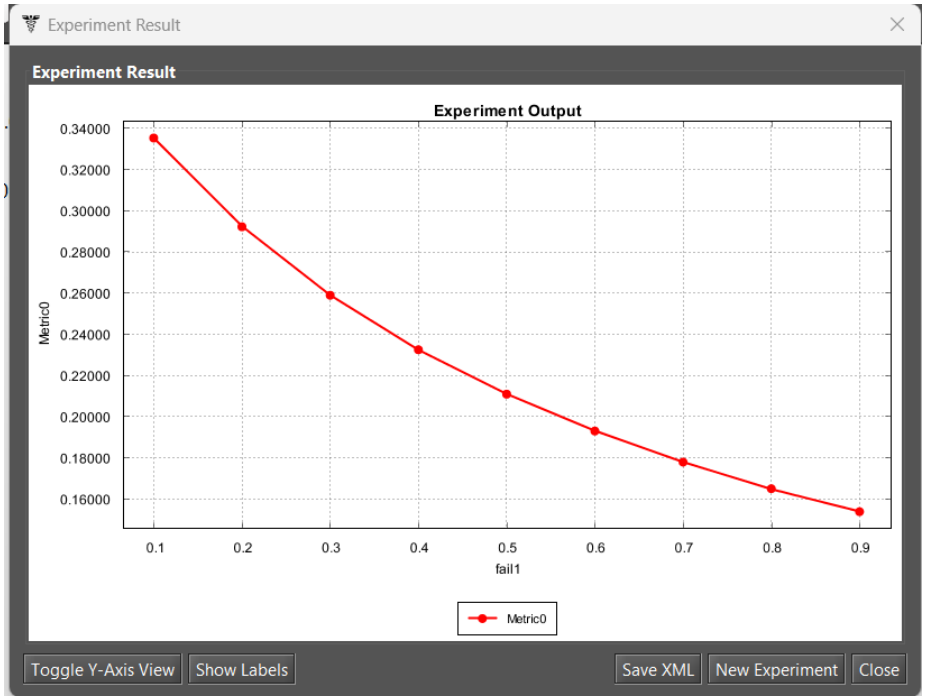


Figure 267: Graph from a DTMC Experiment

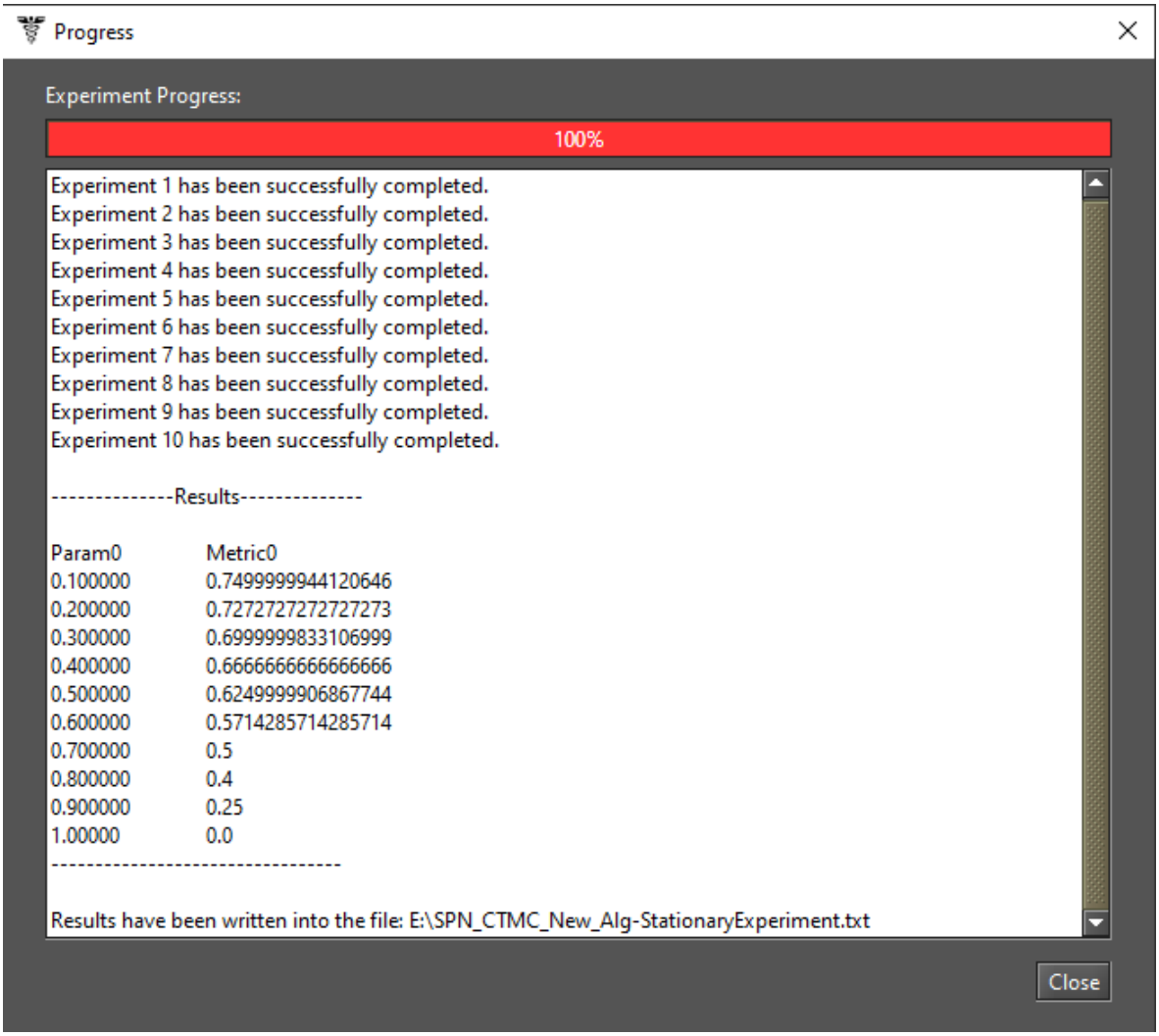


Figure 268: Results from a DTMC Experiment

Mercury can also calculate the following metrics when 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 required to return to each state. After reaching one state and moving to another, how long does it take to return to the previous state? It is determined by this metric. Listing 10 shows the recurrence times calculated 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=3.1999999999999993
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 given time. Figure 269 shows the window “Transient Analysis”.

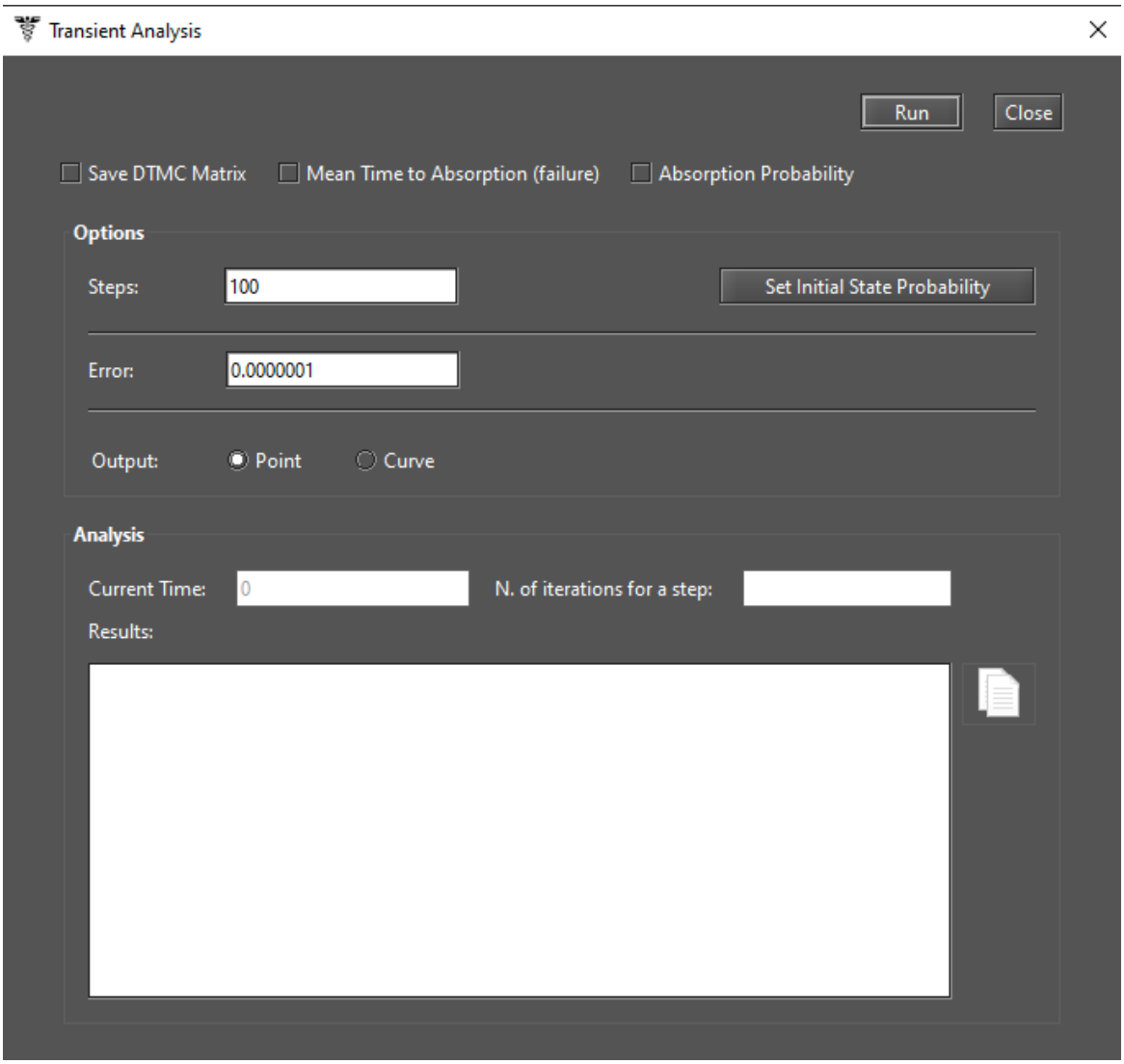


Figure 269: Transient Analysis Window

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

- **Steps** for which the analysis should be performed (default: 100).
- **Error** (default:10^-7) to be taken into account when calculating the results.
- **Initial state probabilities** (default value: 1 for the first inserted state, 0 for the remaining states). These probabilities are defined by clicking the “Set initial state probability” button (see Figure 270).

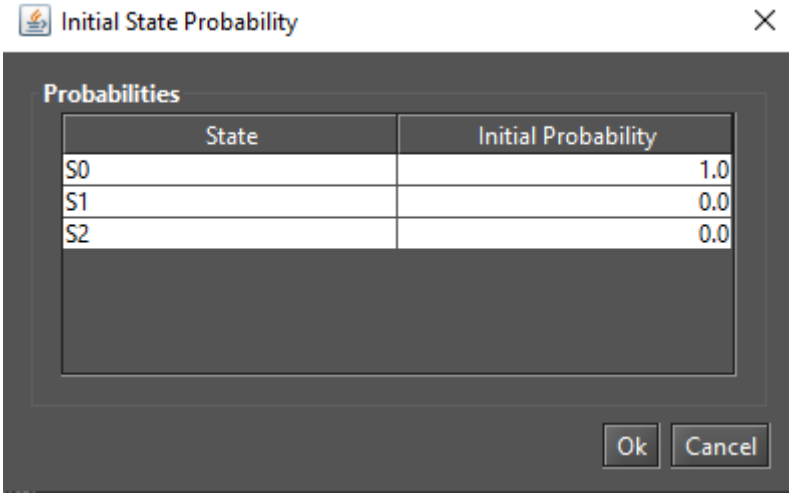


Figure 270: Initial State Probability Dialog

The internal step size affects the accuracy of the results and also the time needed to calculate the metrics. When you click the "Run" button, the solution algorithm is triggered. Once it is finished, the results are displayed in the text area at the bottom of the "Transient Analysis" window and written to a plain text file containing the project filename with the suffix "-TransientAnalysis.txt".

The "Transient Analysis" window also allows the user to choose between **Point** or **Curve** for the analysis. **Point** is the default option and displays probabilities for states only for the specific time point. **Curve** writes all state probabilities calculated from time equal to zero to the specified value to a simple text file.

The mean time to absorption (**MTTA**) can be calculated by checking "Mean Time to Absorption (failure)". MTTA is displayed after the state probabilities in the "Results" section. For MTTA calculation, the user can also define a metric by using the keyword **MTTA** as an expression. The absorption probability for each state is another available metric. Listing 11 shows MTTA and absorption probability using 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 (MTTA): 2.0
```

6.3.3 Sensitivity Analysis

Mercury calculates partial derivative sensitivity indices from DTMCs through sensitivity analysis. These indices indicate what effect each input parameter has on a metric. Mercury provides for DTMC models a sensitivity analysis considering min/max values for each parameter and this analysis supports the "Design of Experiments" (DoE) method in addition to the "Sensitivity Indices" method. This sensitivity analysis is shown in Section 2.5 and can be accessed from the menu *Evaluate -> DTMC Evaluation -> Sensitivity Analysis (min/max values).*

7 ET Modeling and Evaluation

The first step to start modeling ET models on Mercury is to insert leaf nodes into the graph. In the ET view (see Figure 271), the user must right-click on the “Init” root node present on the canvas, select the “Add node” button (see Figure 272), and specify the number of leaf nodes to be added (see Figure 273). Notice that, as the first node is the root, it cannot be deleted. Then, the specified number of events will be added to the model, as shown in Figure 274.

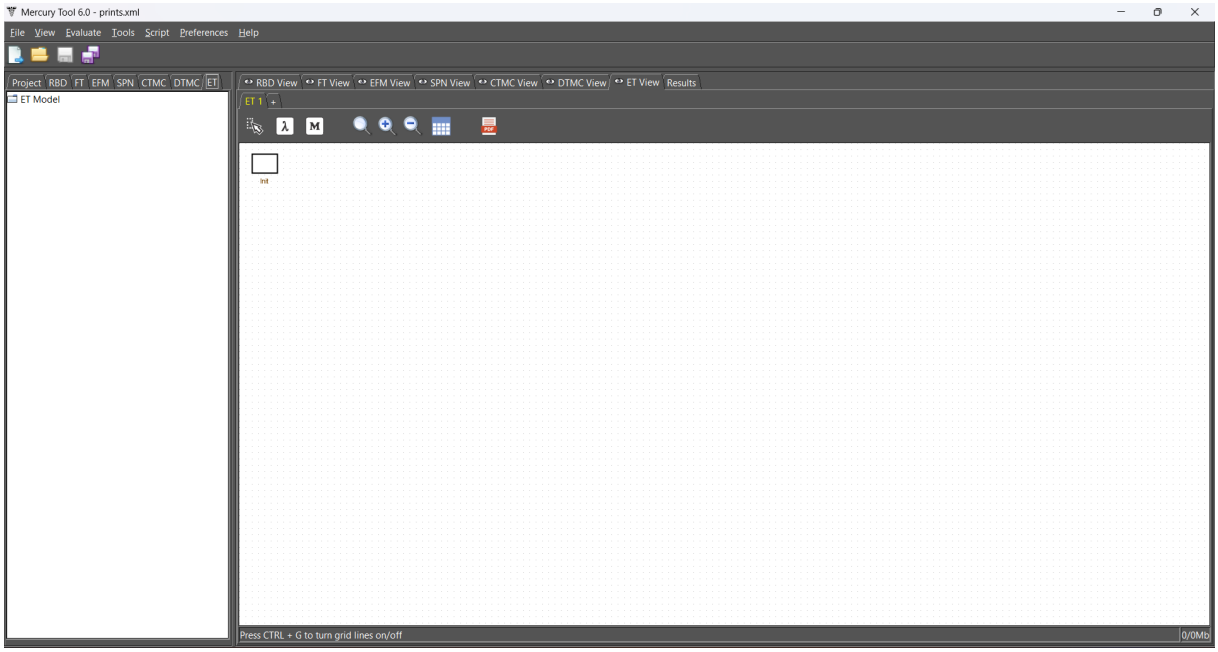


Figure 271: ET View

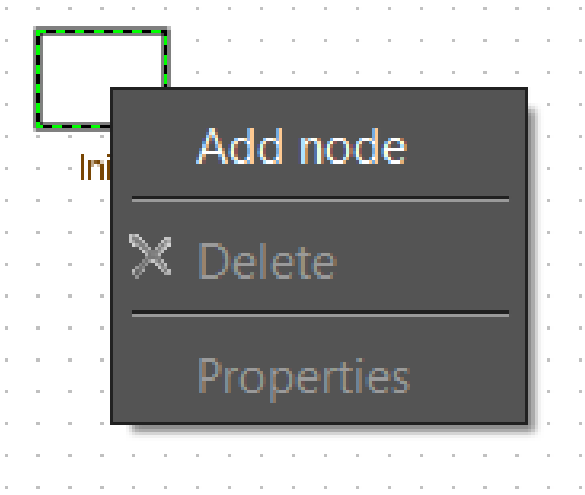


Figure 272: Floating Menu for Adding Events from a Node

Other leaf nodes can also be added by following the same procedure as stated before, i.e., clicking on an intermediary node with the right mouse button, selecting “Add node,” and specifying the number of new nodes to be added.

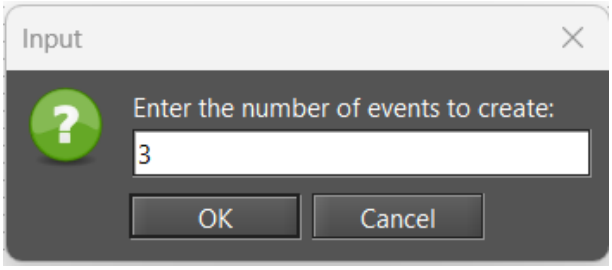


Figure 273: Dialog for Adding Events from a Node

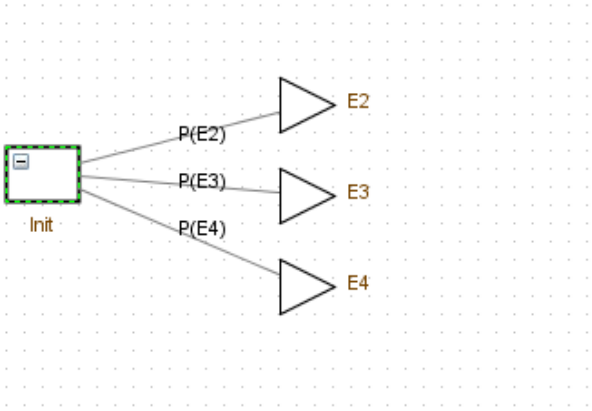


Figure 274: Added Events to the Model

After adding nodes, the hierarchical tree is updated in the left-side panel, as depicted in Figure 275.

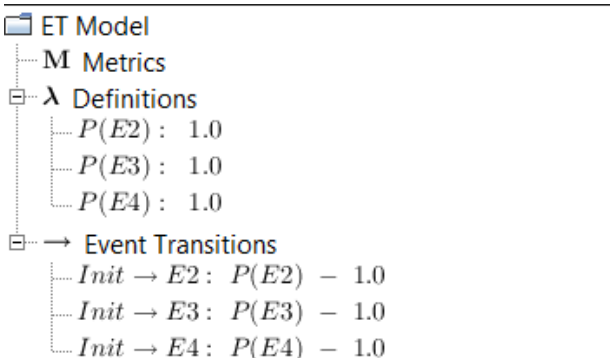


Figure 275: ET Hierarchical Tree

The initial probabilities associated with each connection between nodes are defined as 1.0, or 100%. The user can define the probability for each transition by double-clicking the arc connecting two nodes or by right-clicking on the ending node of the specific arc and selecting the “Properties”, then clicking the “Edit” button on the definition section (see Figures 276 and 277). Figure 278 shows the window where a transition probability can be defined. The user can edit the expression using the visual editor by selecting “Probability Expression” in the “Name Type” dropdown or by plain text by selecting “Plain Text” in the same dropdown. Notice that the dialog shown in Figure 277 can also be used to edit the name of an event and add a description to it. Event names can be defined using expressions containing LaTeX syntax.

It is worth noting that when modeling ETs, the sum of probabilities for all output arcs must equal one for

each node. Otherwise, it is not possible to perform evaluations on the model. Figure 279 shows an error that occurs when this condition is not met.

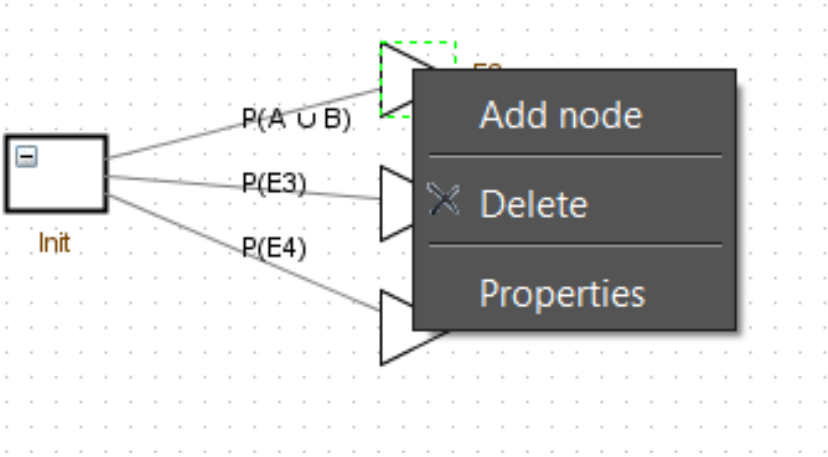


Figure 276: Properties of a Node

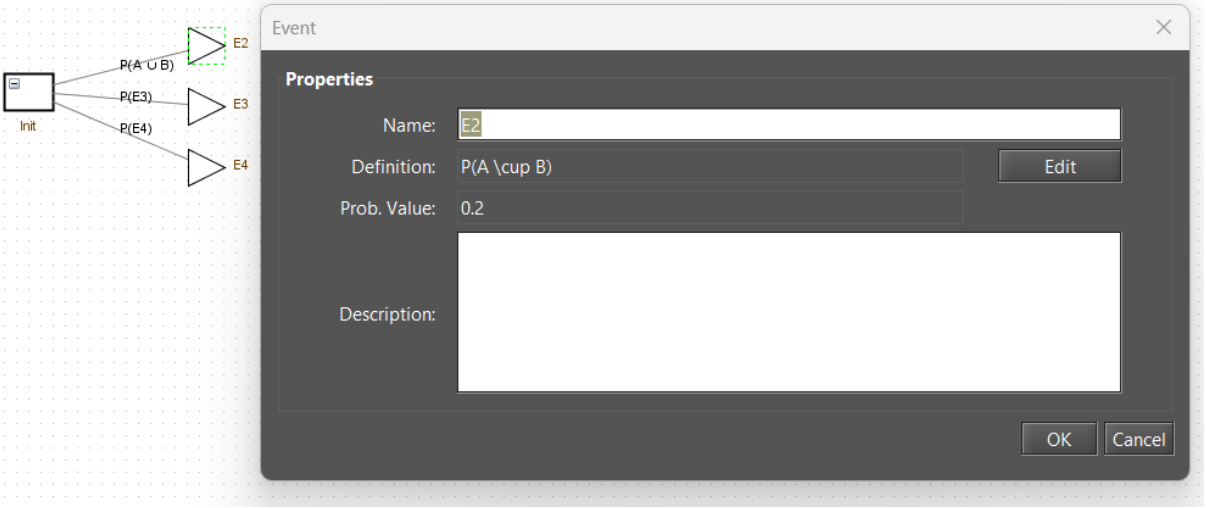


Figure 277: Edit an Event

The ET view has the toolbar shown in Figure 280. The parameter list is accessed through the button depicted in Figure 281. When adding nodes, each parameter that appears in each arc will be added to the parameter list, as shown on the panel on the left side (see Figure 282). The user can add a parameter by clicking the "+ Add definition" button, remove a parameter by selecting it and clicking the "- Remove definition" button, edit a definition by clicking the "..." button in the "Name" column at the end of the parameter's name, or edit its value by double-clicking the value in the "Value" column.

The metrics button, shown in Figure 283, displays the metrics defined by the user. Basically, a metric is an expression used to evaluate a property of the model. You can evaluate the probability of the events in the ET and perform some operations with the probabilities, either by operating with other events' probabilities or other variables added as parameters. Once the metrics button is clicked, you can see on the left panel the ET's metrics list, as shown in Figure 284. The "Name" column shows the metric's name, "Expression" shows the metric's LaTeX

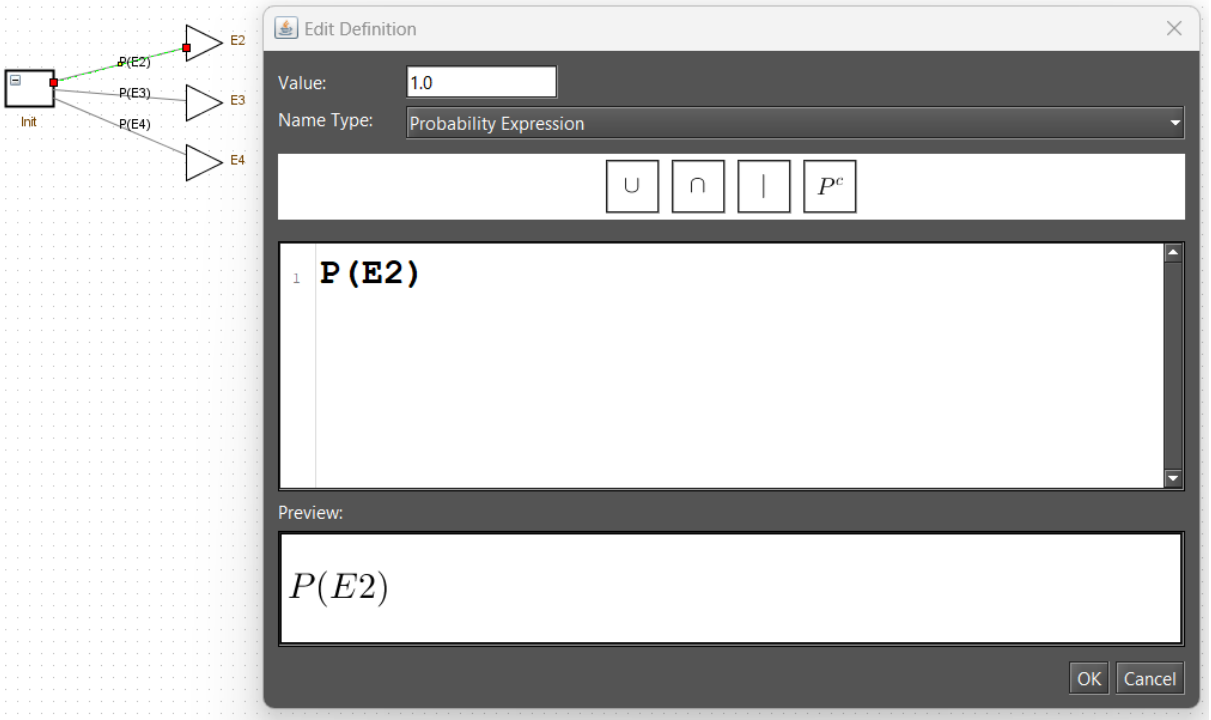


Figure 278: Edit the Associated Probability to an Arc

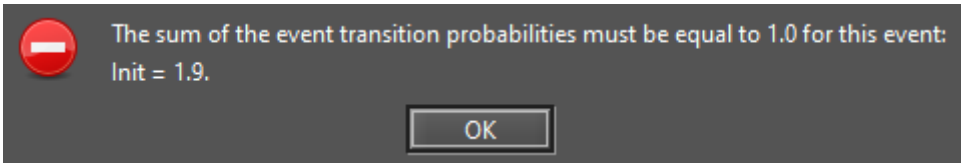


Figure 279: Event Transition Probabilities Error

Figure 280: Toolbar of the ET View

Figure 281: Definitions (Parameters) Button on the Toolbar of the ET View

expression to be evaluated, and the last column “Value” shows the resulting value when the metric’s expression is evaluated.

Metrics can be created by clicking the “+ Add metric” button on the left panel. Then, a new metric will be created in the metrics list. Clicking on any of the “...” buttons at the metric’s row will open the “Metric Editor” dialog, where the metric’s name and expression can be entered. The metric’s name must be a plain text. The

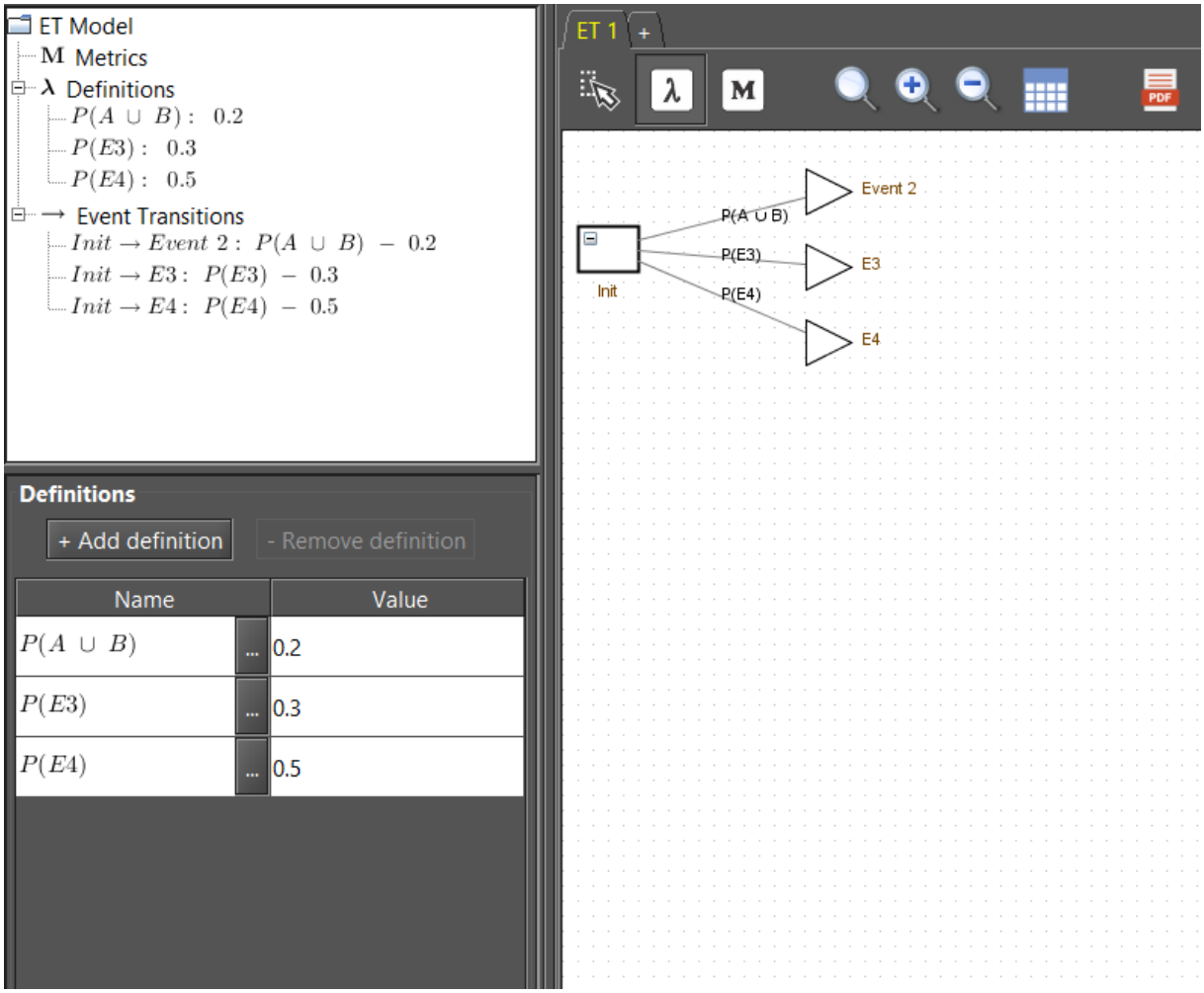


Figure 282: Parameters Panel of the ET View

"Value" field is disabled and will be populated when the model is evaluated. Finally, the "Expression Editor" field must be filled with expressions, as shown in Figure 285. The parameter's name must be wrapped in "[]" and "[/]". Basic math operations can be performed in the "Expression Editor".

The zoom buttons, shown in Figure 286, are used to enhance the visibility of the model, especially when it becomes too large to fit on the entire screen. The user can return to the original viewport scale by clicking the default magnifying glass, increase the zoom by clicking on the magnifying glass with a plus sign, or decrease the viewport scale by clicking on the magnifying glass with a minus sign.

The save matrix button, shown in Figure 287, is used to transform the entire model into a matrix that displays every associated probability, as discussed in the DTMC section, allowing the users to export the probability

Figure 283: Metrics Button on the Toolbar of the ET View

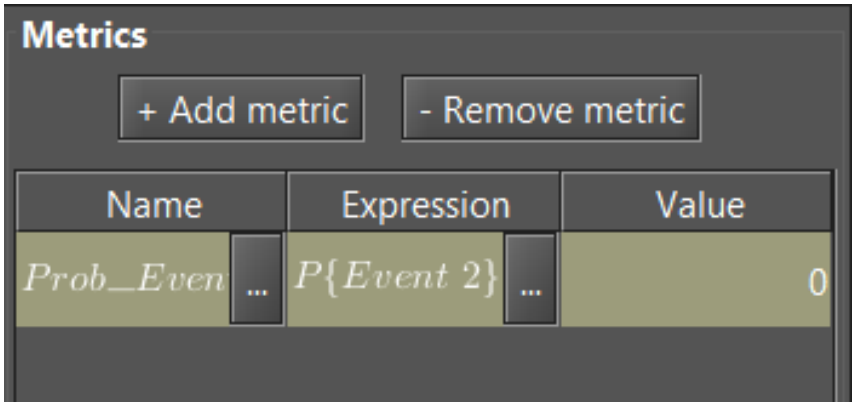


Figure 284: Metrics Panel of the ET View

Figure 285: Metric Editor dialog

Figure 286: Zoom Buttons on the Toolbar of the ET View

matrix of a model to a text file.

Figure 287: Save Matrix Button on the Toolbar of the ET View

7.1 ET Evaluation

Mercury allows us to evaluate ETs models. This evaluation gives us the probabilities of all nodes of our ET (Initial node, chances node, and leaf nodes) and all metrics defined in the model. It is available from the *Evaluate -> ET Evaluation* menu (see Figure 288).

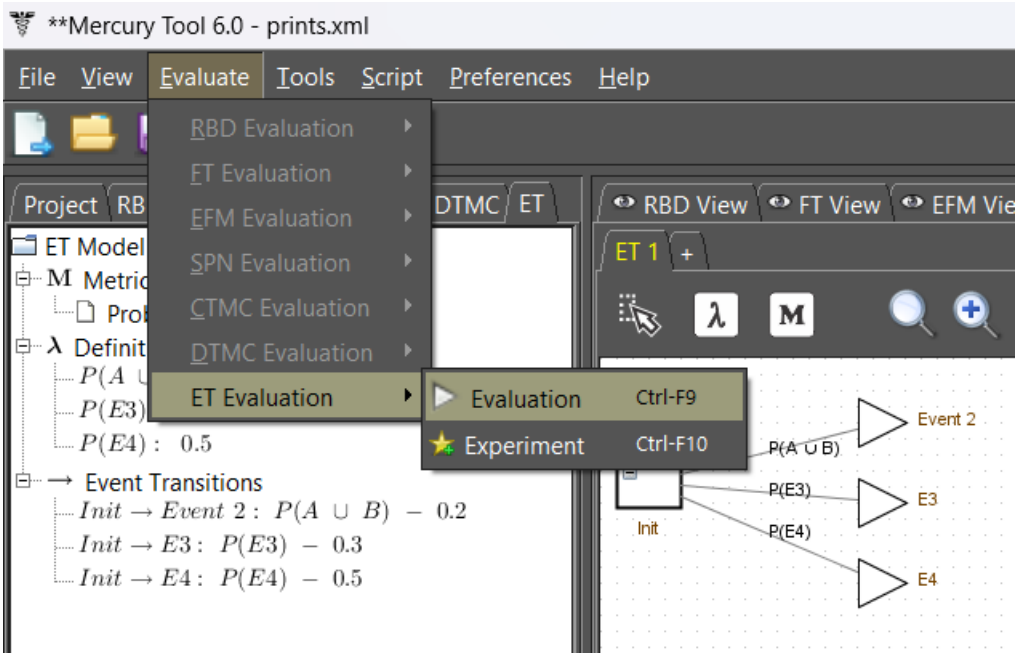


Figure 288: ET Evaluation Menu

The model is evaluated and the result is shown in a new window (Figure 289).

Notice that for the evaluation of this model type, we do not need to set anything to do this. To enable an evaluation, the only requirement is that the sum of the outgoing probabilities of each initial or chance node must equal 1.

7.1.1 ET Experiment

To repeat the evaluation many times by simply changing the probabilities, you can use the Experiment feature. This feature is accessible via the same menu as the evaluation (see Figure 288).

When we enable this option, the next window appears (Figure 290).

- The **parameter** option is the variable (definition) that you want to modify.
- The **minimum value** is the initial value that the selected parameter will have in the first evaluation.
- The **maximum value** is the last value that the specified parameter may assume.
- The **interval (step size)** is the increment that the parameter undergoes in each evaluation. Depending on this step, the maximum value may or may not be reached.
- The **metric** is the metric for which you want to obtain the result. In this mode, we cannot obtain results for every metric at the same time.

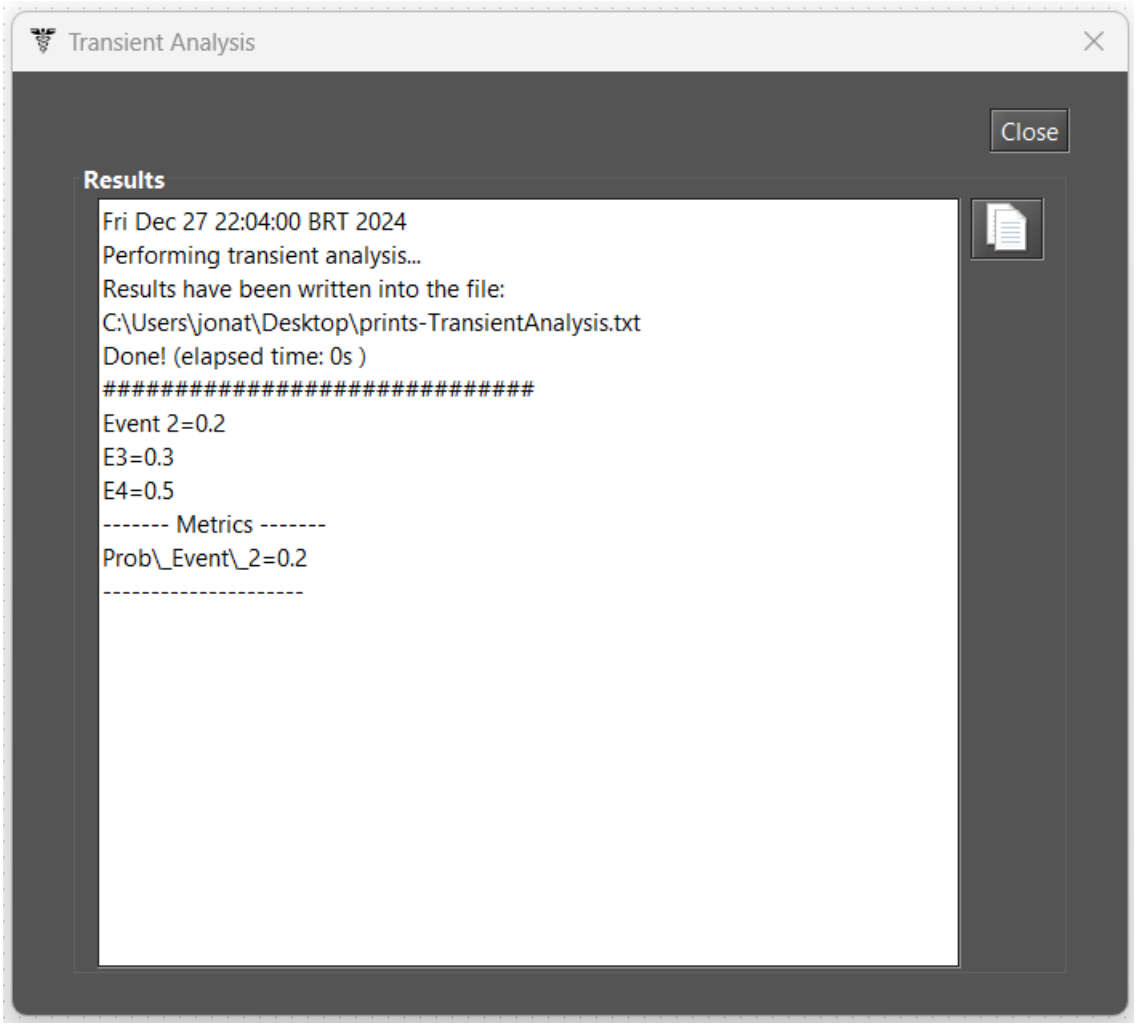


Figure 289: Evaluation Result Window

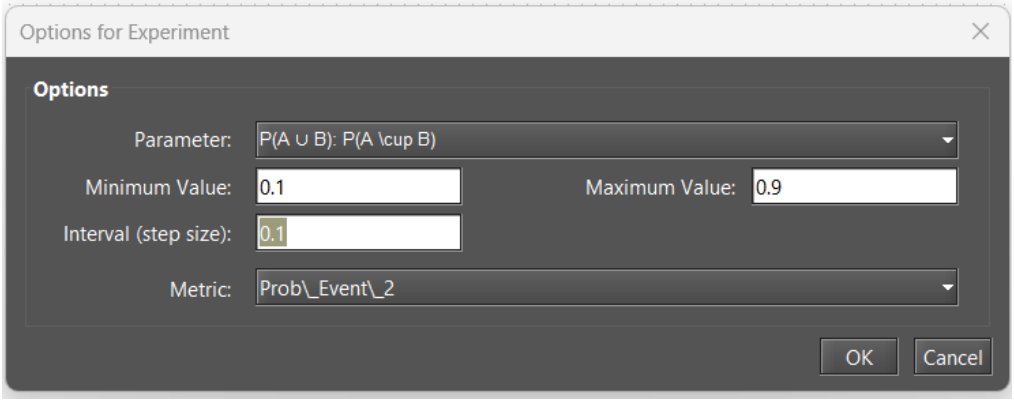


Figure 290: Options for Experiment

It is important to emphasize that for each source event the sum of its outgoing probabilities must equal 1. To ensure this condition, Mercury performs a proportional redistribution of the remaining probabilities when the selected parameter is a transition probability of an event. In this case, the adjusted transition probability of

any other outgoing event $P(E_i)$ associated with the same source event can be calculated using the following expression:

$$P(E_i) = (1 - P(\text{condition})) \cdot \frac{P(E_i)}{\sum_j P(E_j)} \tag{1}$$

where $P(\text{condition})$ is the selected parameter and $\sum_j P(E_j)$ is the sum of all initial transition probabilities of the other outgoing events associated with the source event linked to the selected transition probability. This expression ensures that the other adjusted transition probabilities $P(E_i)$ maintain their proportional relationships while satisfying the condition that the sum of the transition probabilities of the same source event is equal to 1.

For example, if there is a selected transition probability such as $P(A \cup B)$ and two other transition probabilities $P(E2)$ and $P(E3)$ associated with the same source event, the remaining transition probability becomes $1 - P(A \cup B)$ and $P(E2)$ and $P(E3)$ are adjusted as in the equations 2 and 3:

$$P(E2) = (1 - P(A \cup B)) \cdot \frac{P(E2)}{P(E2) + P(E3)} \tag{2}$$

$$P(E3) = (1 - P(A \cup B)) \cdot \frac{P(E3)}{P(E2) + P(E3)} \tag{3}$$

This approach ensures that the transition probabilities for the other outgoing events are redistributed proportionally, while the total sum of 1 for the transition probabilities of the source event is retained. This approach can be applied to any number of outgoing events, ensuring that the probabilities are adjusted proportionally.

Initially, the experiment result will be presented in a plot format (see Figure 291), but, if the plot is closed the result can be obtained in the format of a common evaluation (Figure 289). Here, each point on the curve represents an evaluation with different values for the selected parameter. Consequently, this also affects the other probabilities of the same node to which the selected parameter is linked.

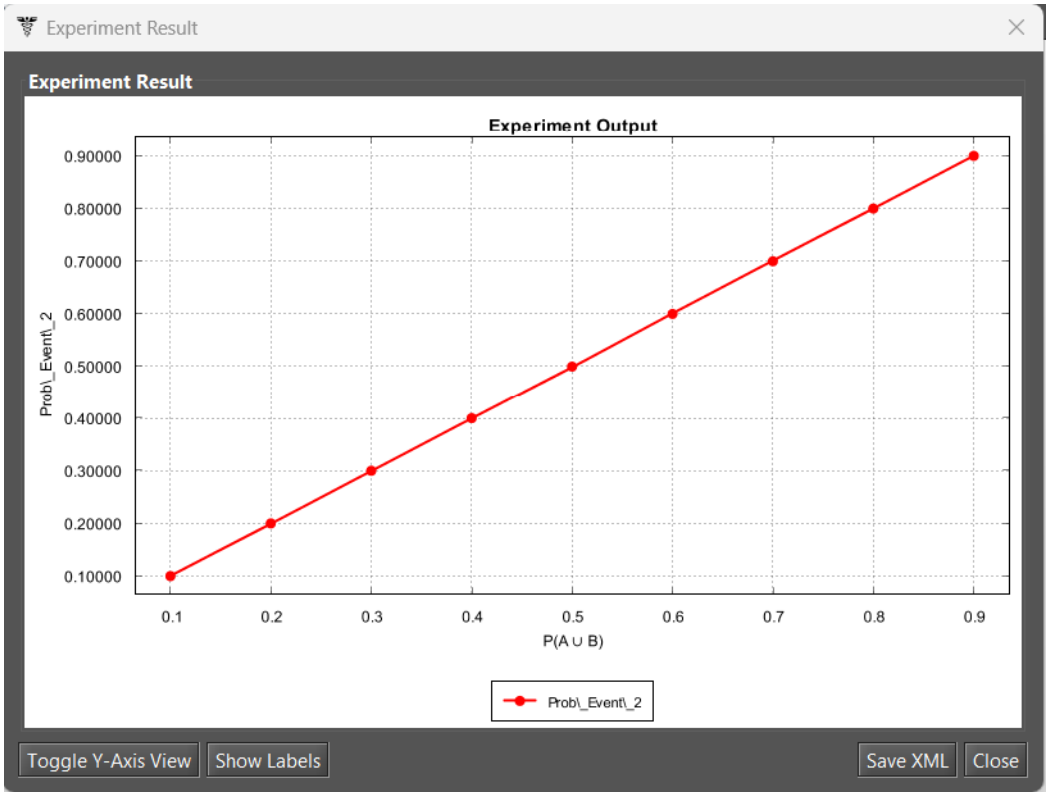


Figure 291: Experiment Result

8 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 budged needed to obtain the system components in order to build the system. The operational cost is the cost to maintain the system in the operational mode.

Figure 292 depicts an EFM model representing a system composed of four components and it demonstrates how the energy flow occurs between them.

Now we will introduce the EFM toolbar and its available resources. This toolbar is visible when the EFM view is selected. Figure 293 shows the buttons present on it and their descriptions are detailed below.

1. **Default Cursor.** Activates the selection mode. This mode makes it possible to select model components in the drawing area.

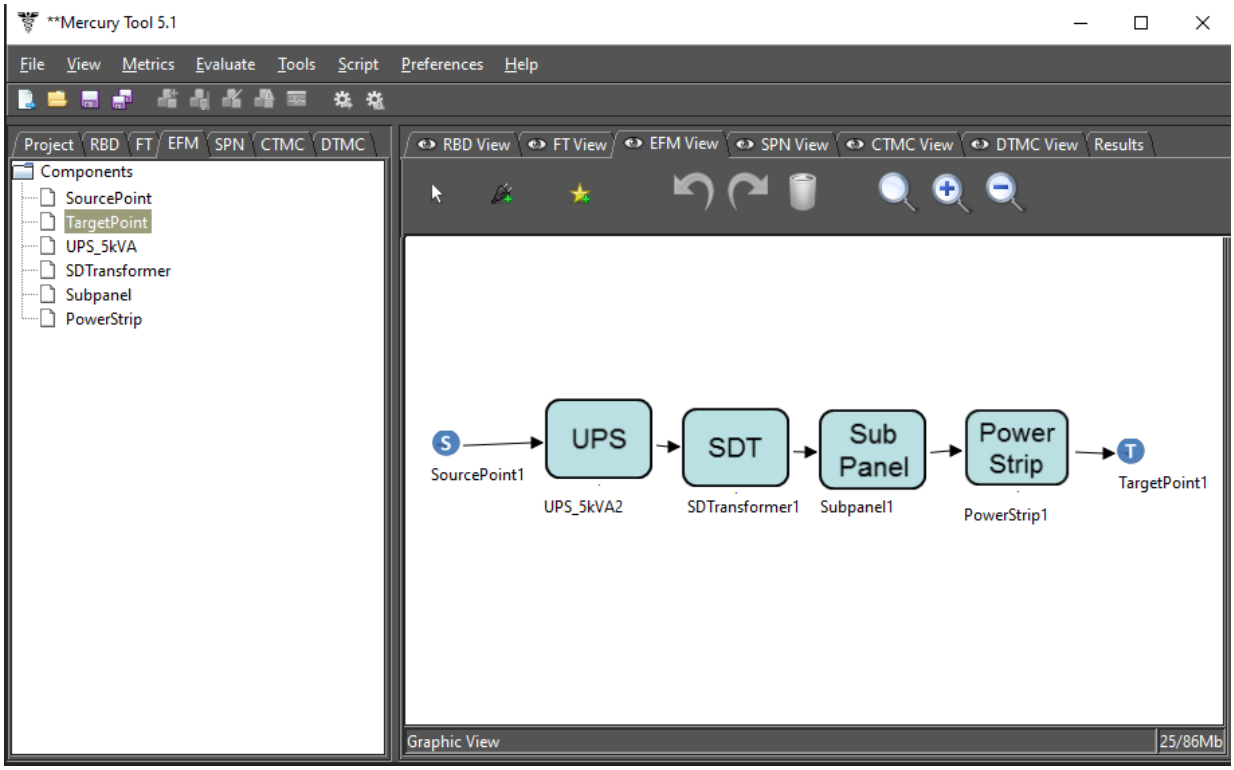


Figure 292: EFM Example

Figure 293: EFM Toolbar

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 294). 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.

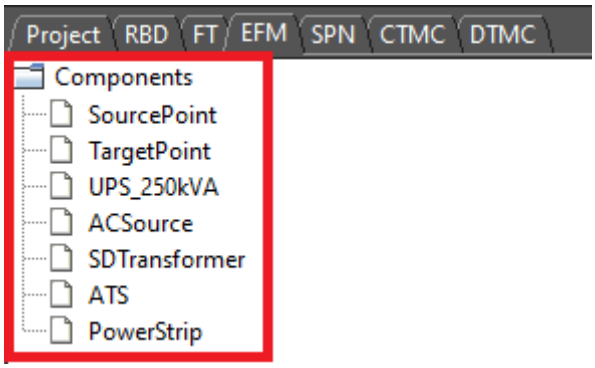


Figure 294: EFM Left-Side Panel

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 295). The new component will be available on the left side as shown by Figure 294. 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.

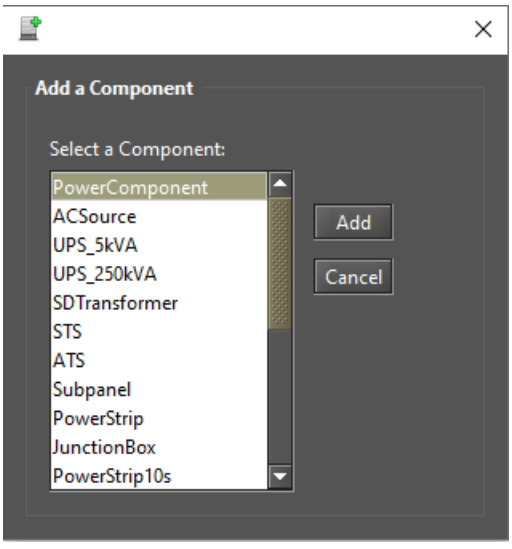


Figure 295: Inserting an EFM Component to the Project

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 295). 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 296. 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.

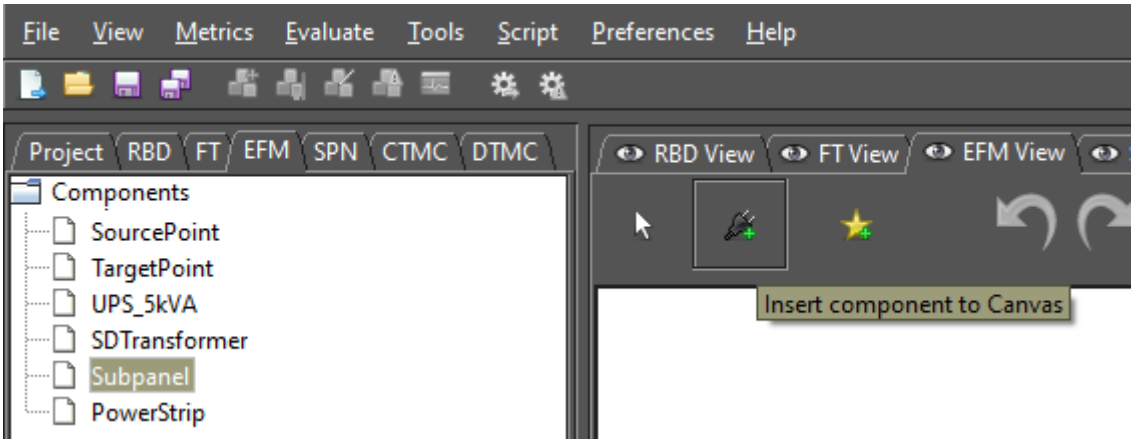


Figure 296: Inserting a Component into the Drawing Area

Once all components are inserted into the drawing area, the user can connect them including SourcePoints and TargetPoint as shown by Figure 297.

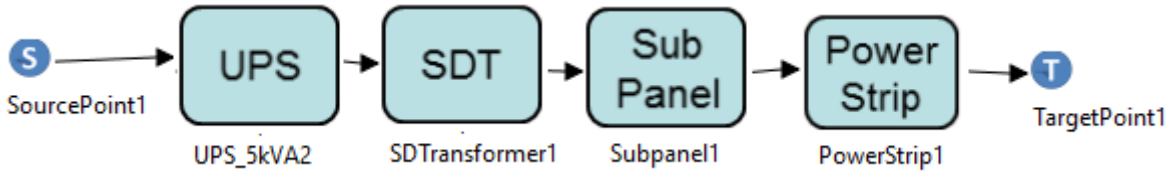


Figure 297: EFM Example

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 298.

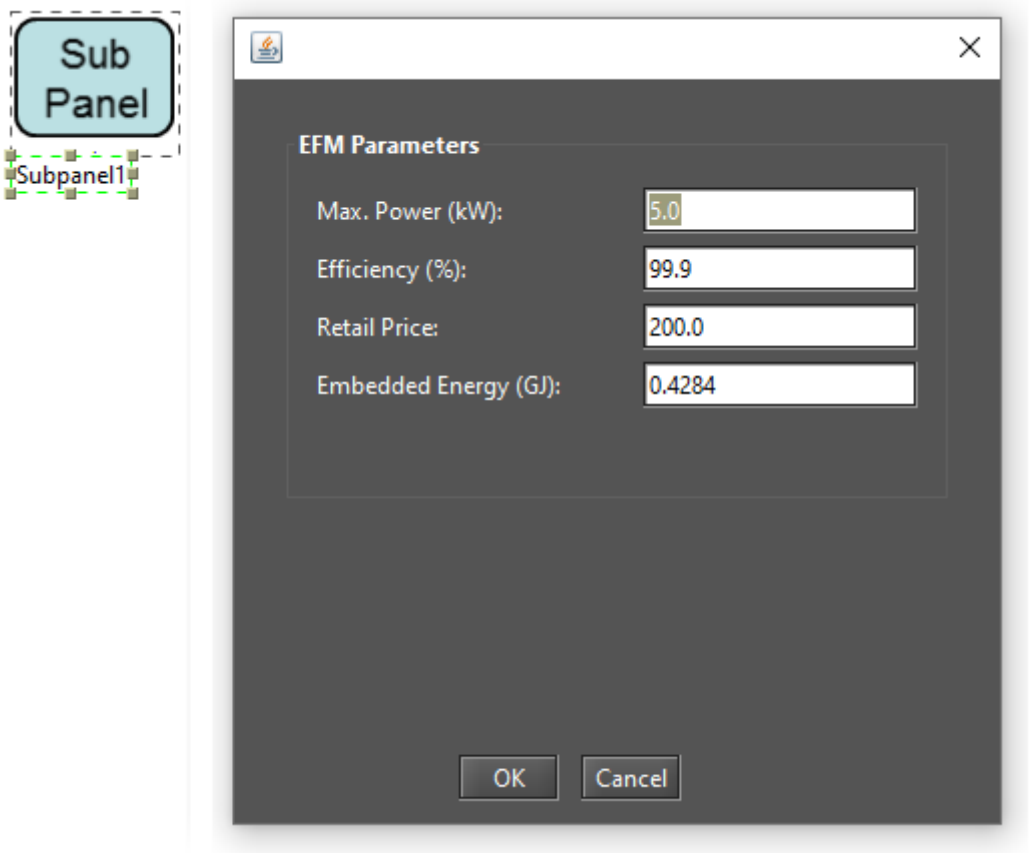


Figure 298: Component Properties

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 299).

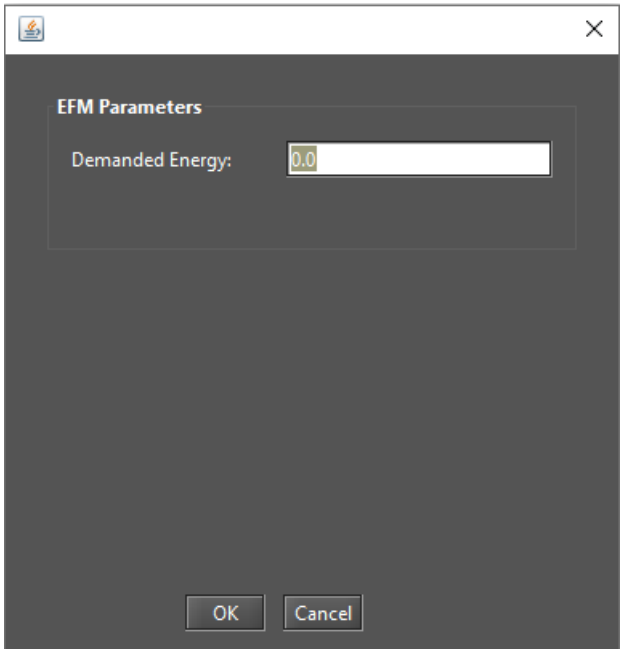


Figure 299: SourcePoint and TargetPoint Property

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 300. Five EFM evaluations are available. It is possible to evaluate cost, exergy, energy flow, the last ones combined, and flow optimization.

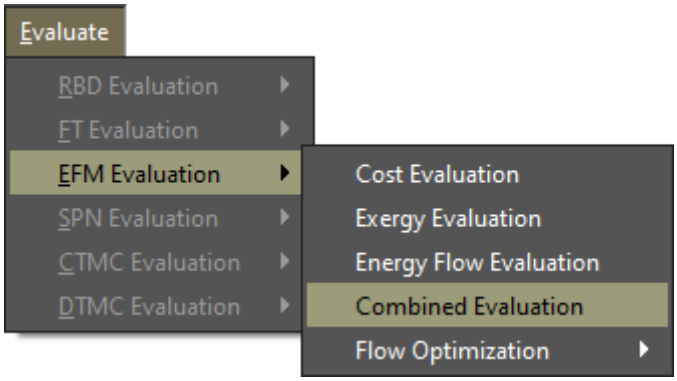


Figure 300: EFM Evaluation Menu

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

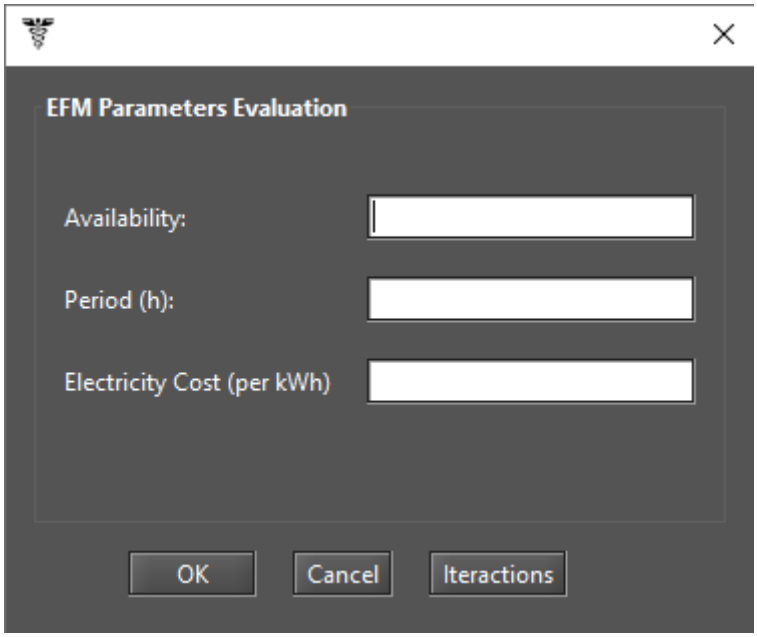


Figure 301: EFM Parameters Evaluation

Finally, the result is presented as demonstrated by Figure 302.

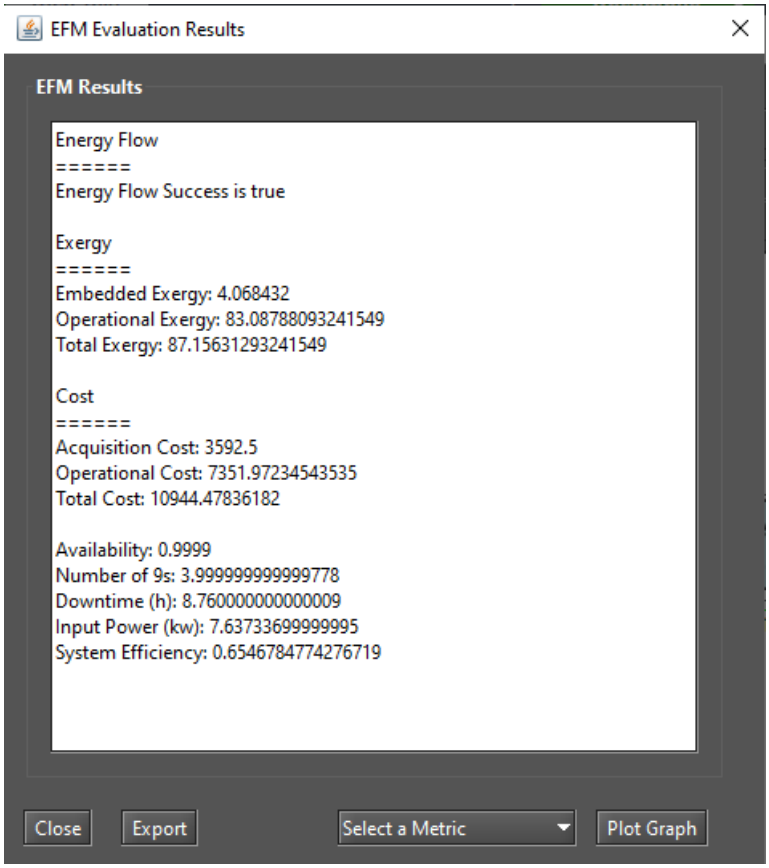


Figure 302: EFM Results

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 303). 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.

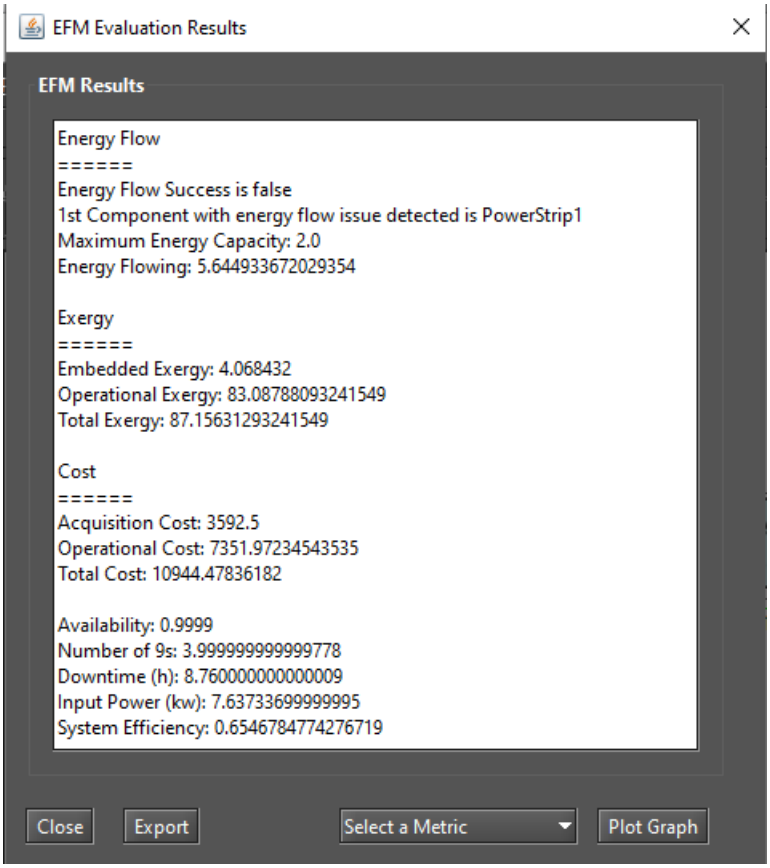


Figure 303: EFM Results with the Energy Flow Evaluation False

8.1 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 304 shows how to call the PLDA function.

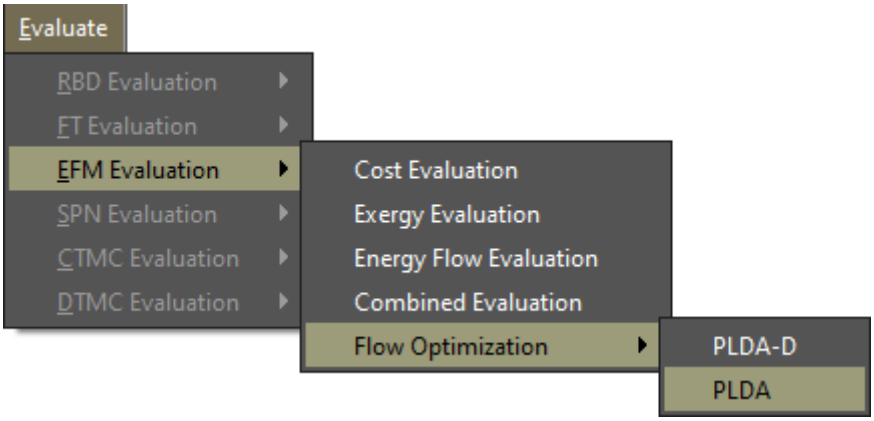


Figure 304: PLDA Optimization Function

Figure 305 depicts the results of the PLDA algorithm, with the minimum energy consumed, PUE, and DCiE highlighted.

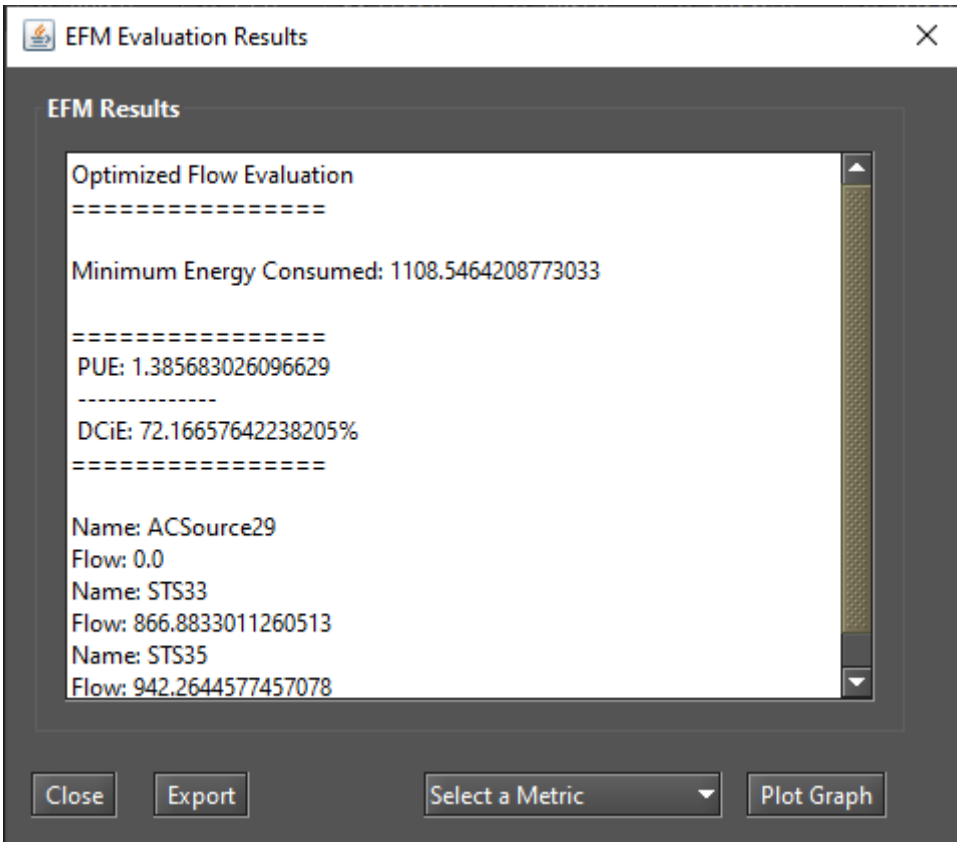


Figure 305: PLDA Optimization Results

8.1.1 Example of PLDA execution

Figure 306 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.

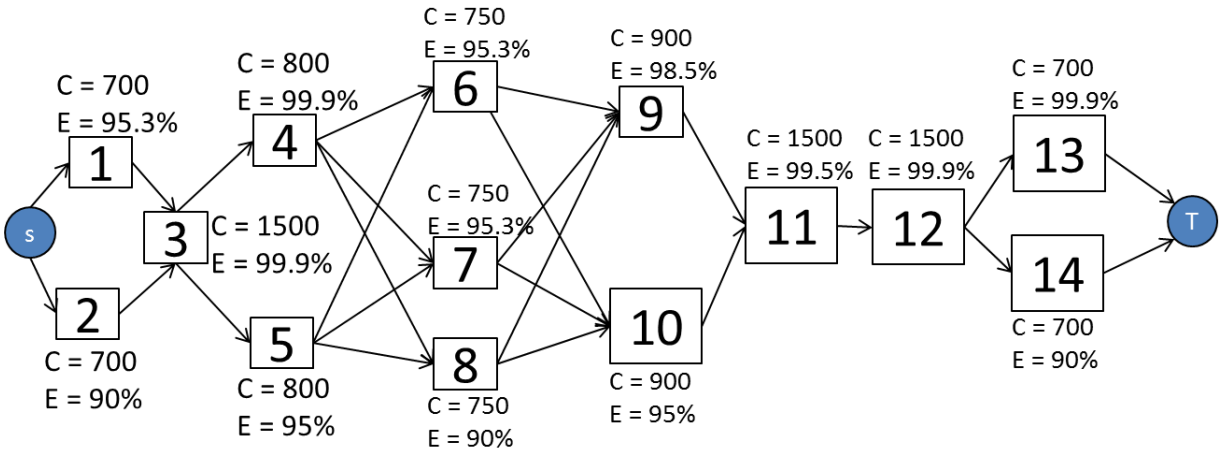


Figure 306: EFM Model

Figure 307 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.

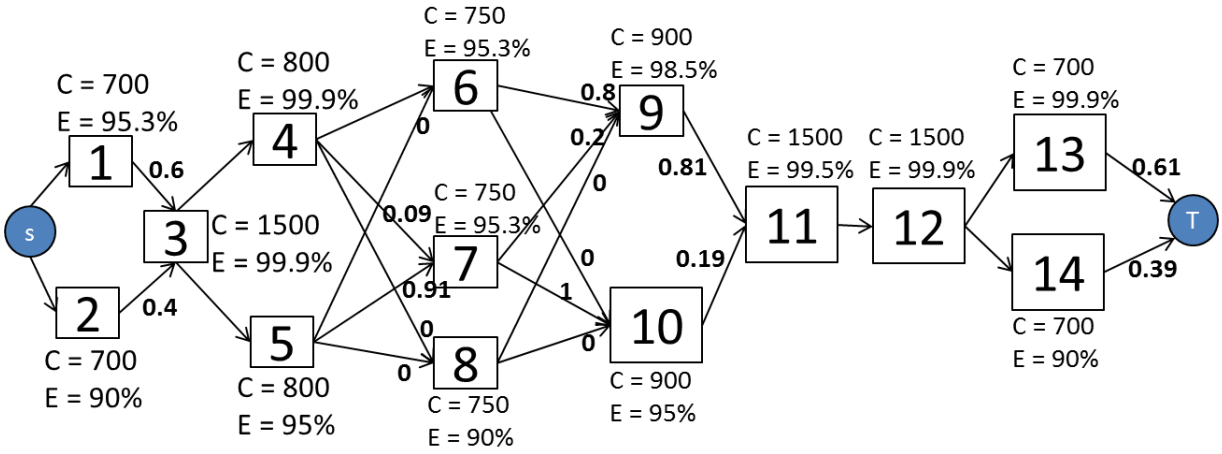


Figure 307: EFM Model After PLDA Execution

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

8.2 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 8.1), 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 308 demonstrates how to call the PLDA-D function. Figure 309 depicts the results generated by applying the PLDA-D algorithm, with the minimum energy consumed, PUE, and DCiE highlighted.

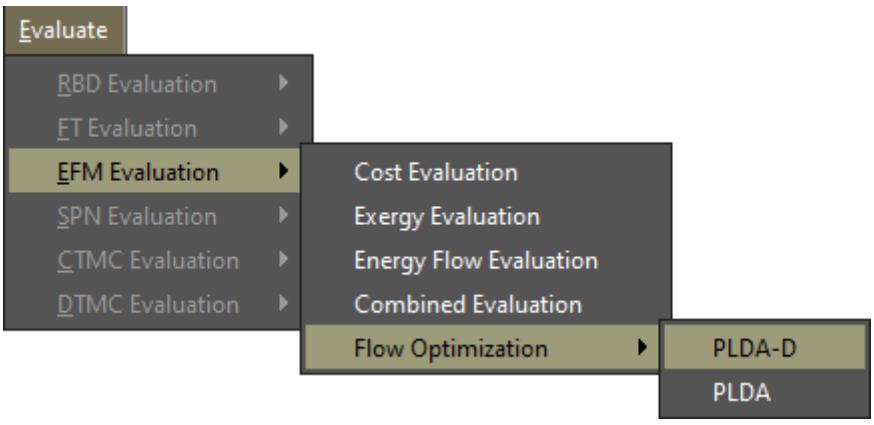


Figure 308: PLDA-D Optimization Function

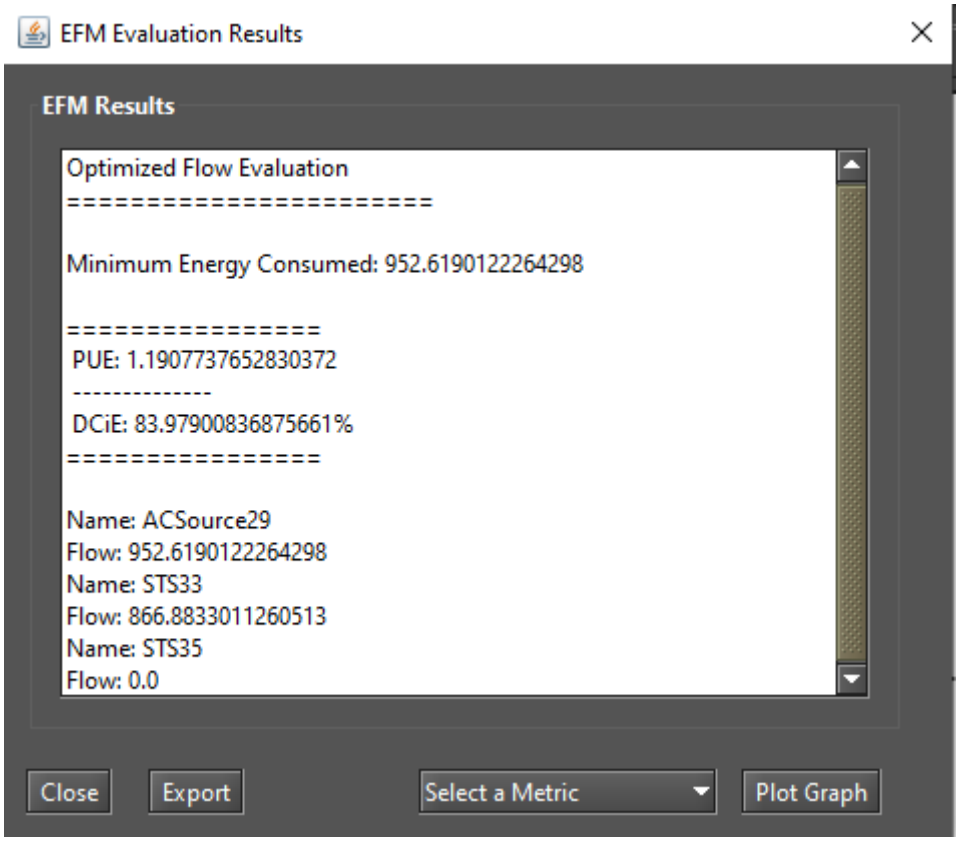


Figure 309: Results for Applying the PLDA-D Optimization

8.2.1 Example of PLDA-D Execution

Figure 310 illustrates the step-by-step of the PLDAD execution, highlighting some variables, edges, and weights. Lets consider the model represent by Figure 310.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 310.a, a perfect representation of an EFM model. The phase one of the PLDAD algorithm is represented by the Figure 310.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 310.c following to the Figure 310.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 310.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 310.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

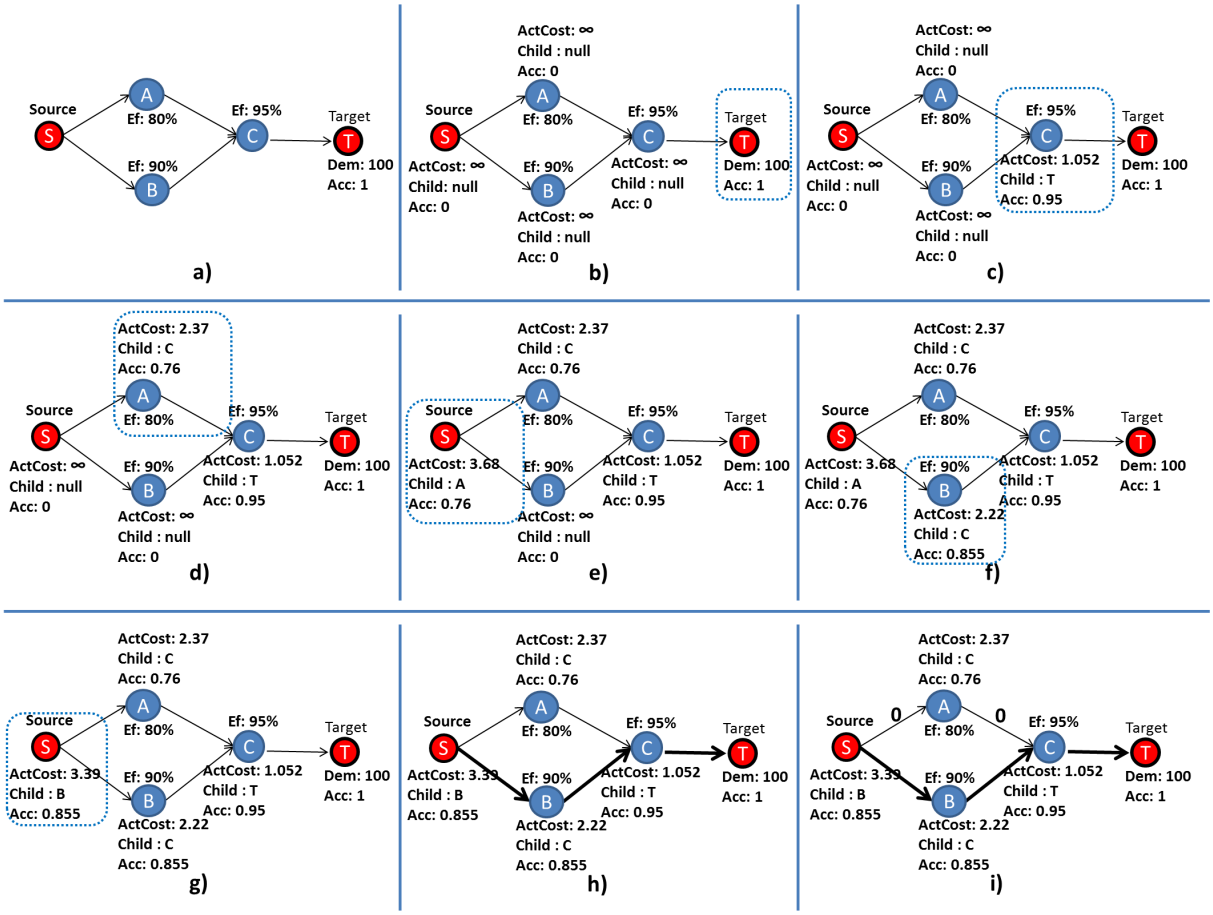


Figure 310: Exemple PLDAD Execution

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 310.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 310.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.

9 Comments

Along with the modeling features, Mercury allows you to annotate models and highlight model elements. The RBD and FT views provide adding comments and highlighting objects by right-clicking anywhere on the drawing area, then selecting the option “Add comment”, as shown in Figure 311. The comment will be placed at the cursor position, and can be moved anywhere in the drawing area. In the SPN, CTMC and DTMC views, comments and highlighting can be added to the drawing area by activating the “Comment” button on the toolbar, as shown in Figure 312. Then, move the cursor in the drawing area and click to place the comment. Comments are not available in the EFM and ET views.

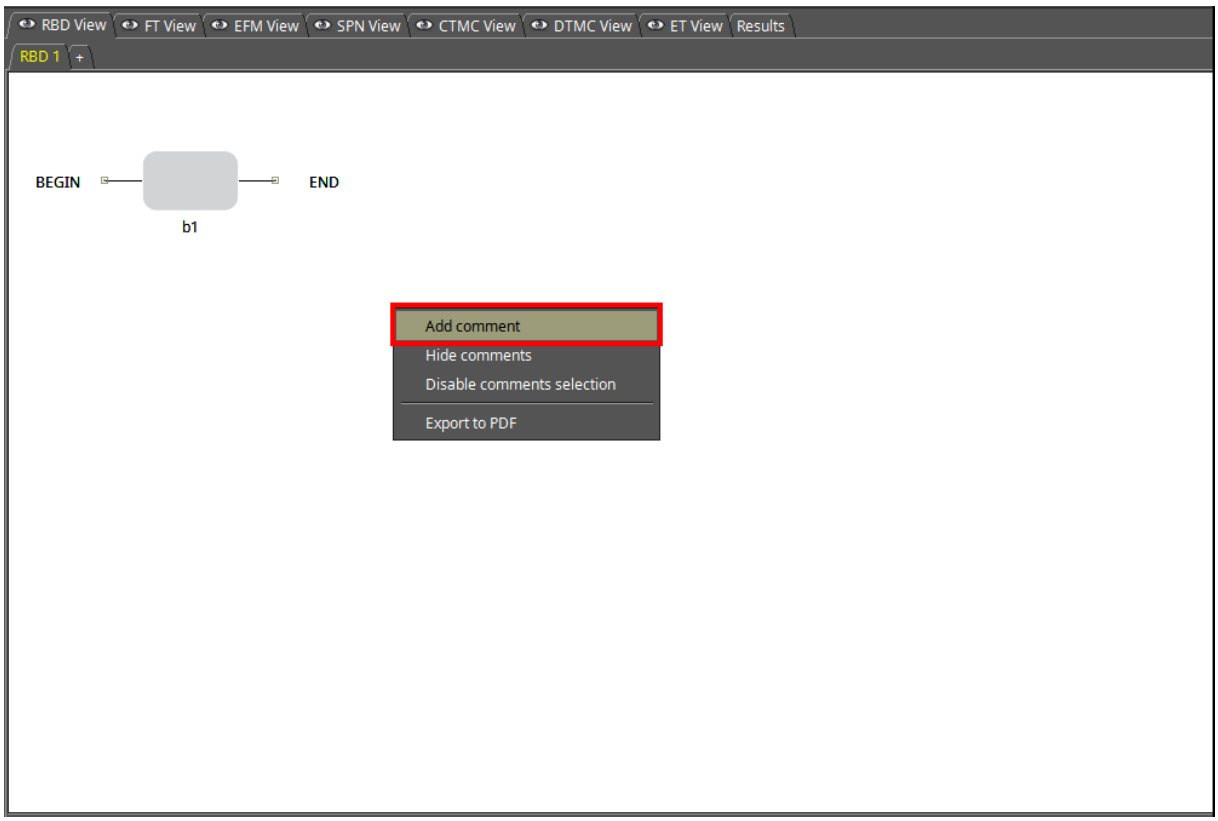


Figure 311: Adding comments on RBD view

Once a comment is created, an object with a “Comment” label will be available on the drawing area, as shown in Figure 313. The comments behave and are shown equally in the RBD, FT, SPN, CTMC and DTMC views.

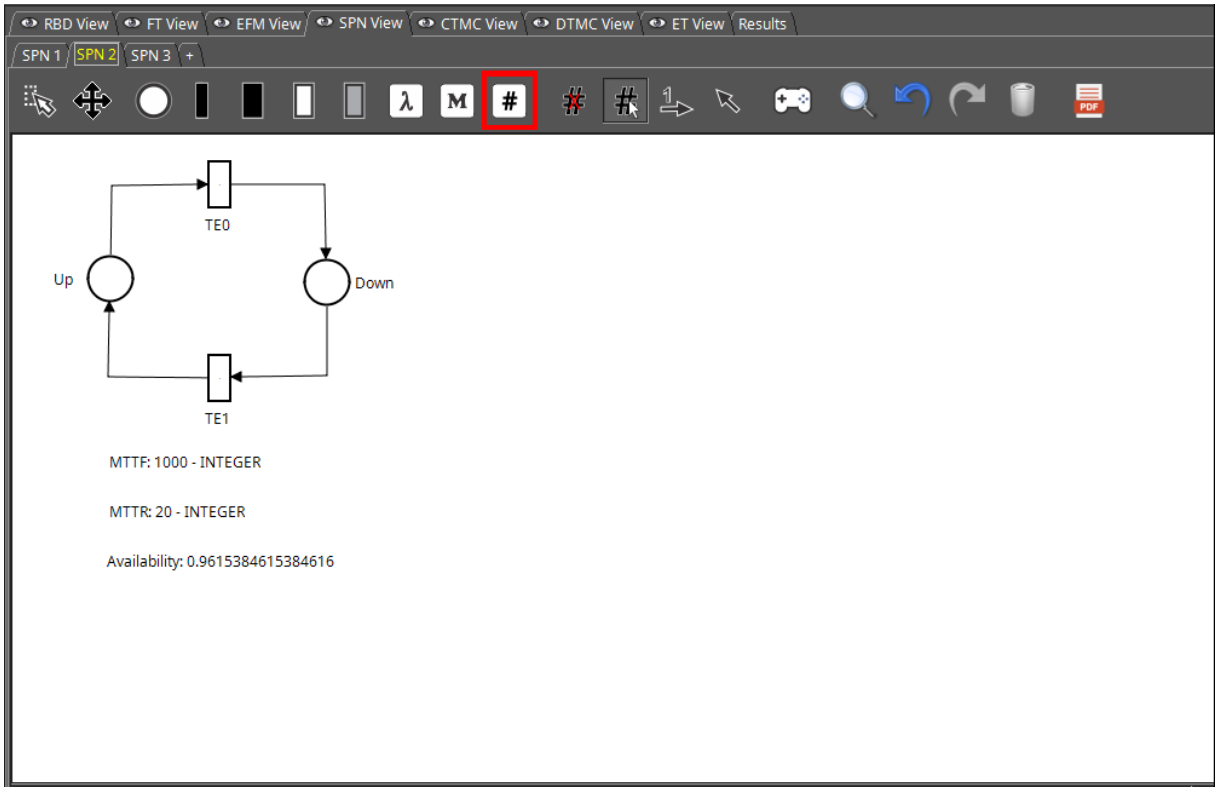


Figure 312: Adding comments on SPN view

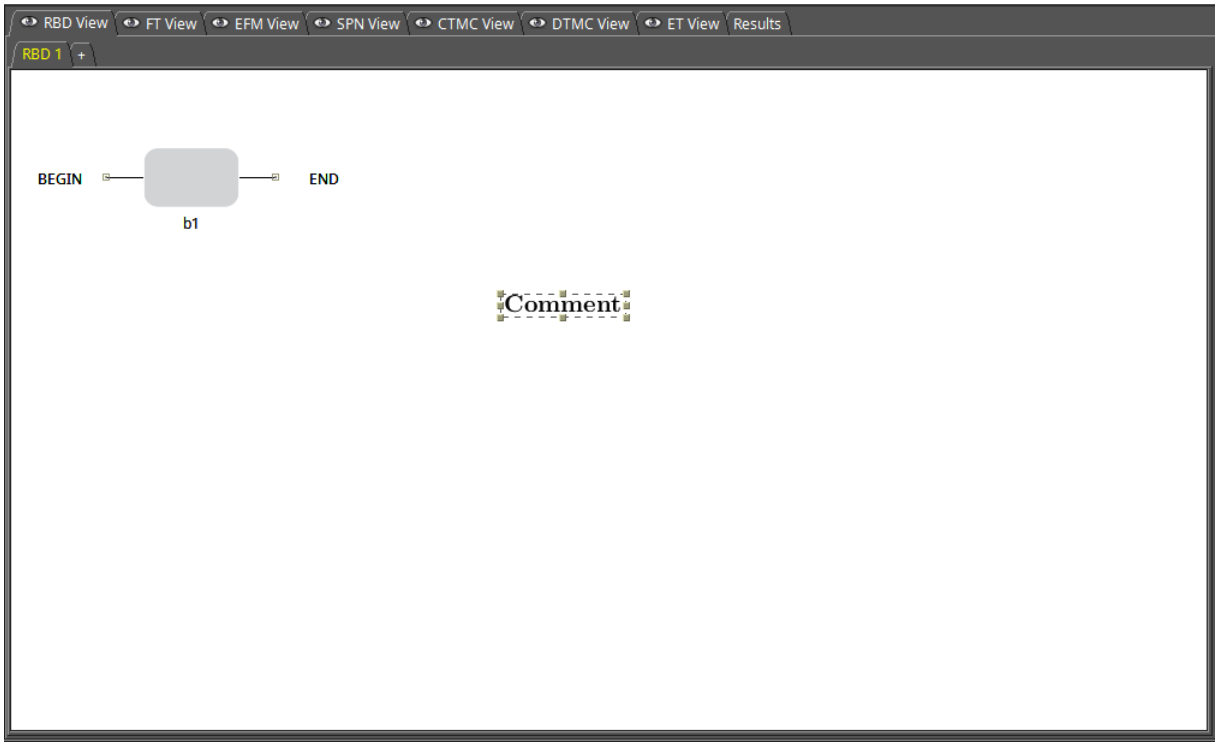


Figure 313: Added comment on RBD view

9.1 Editing a comment

Double-clicking the “Comment” object will open a dialog for editing the comment, as shown in Figure 314. The Edit Comment dialog has two checkboxes: “Enable border” and “Enable background color”. Checking the “Enable border” option enables a combobox to select the color of the border of the comment. Checking “Enable background color” enables a combobox to select the color of the background of the comment.

The Edit Comment dialog has a typing area and a resulting area. Typing LaTeX expressions in the typing area will render the expressions at the resulting area. The resulting area shows how the comment will become once editing is saved. Figure 315 shows how to edit a comment with LaTeX expression and background color, and Figure 316 shows the resulting comment in the CTMC view.

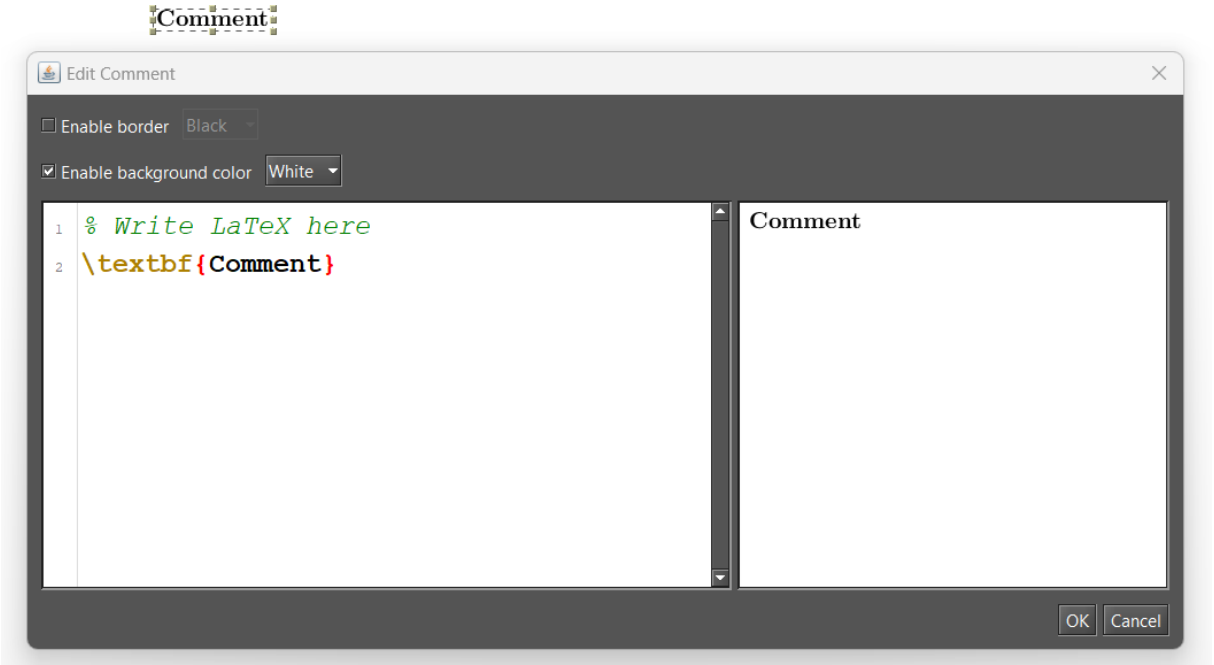


Figure 314: Editing a comment

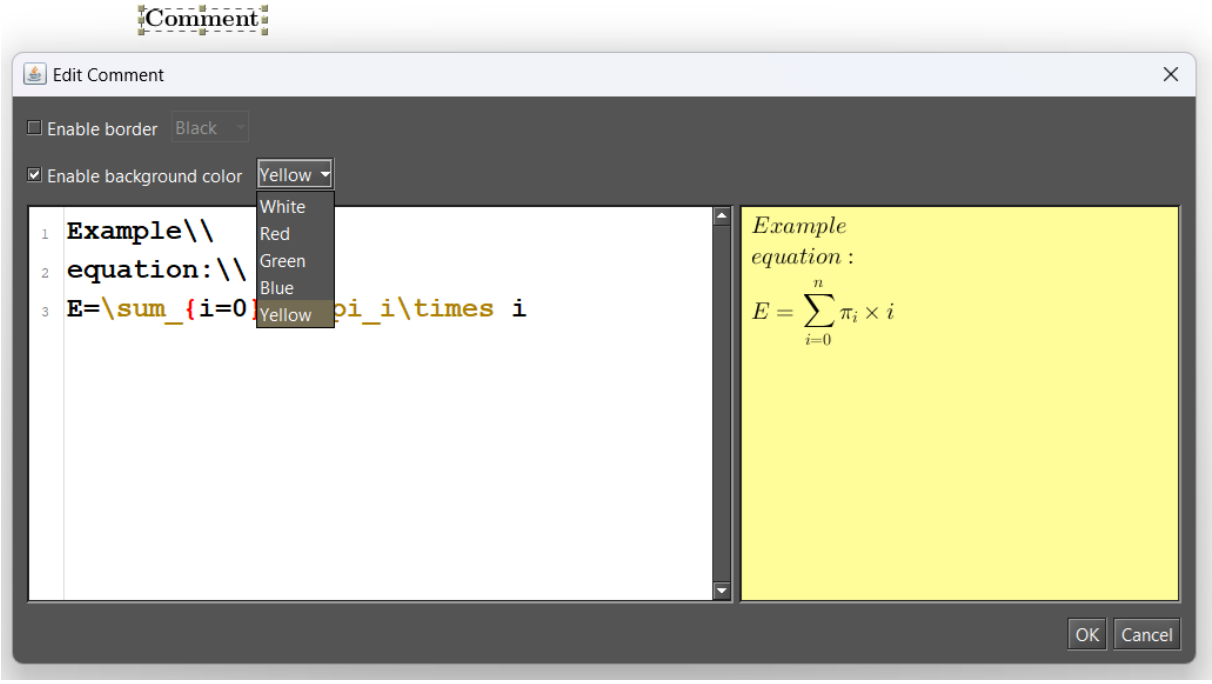


Figure 315: Editing comment with LaTeX expression and background color

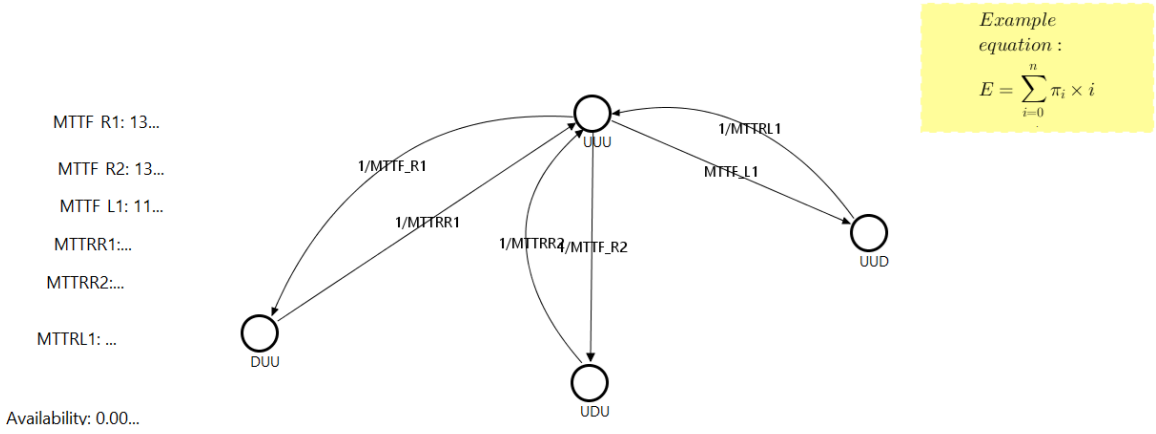


Figure 316: Edited comment with LaTeX expression and background color

9.2 Highlighting with comments

Highlighting elements in the drawing area is possible using comments in the RBD, FT, SPN, CTMC and DTMC views. This can be done by editing a comment, as shown in Figure 317, removing the text from the typing area, checking the “Enable border” option and disabling the “Enable background color” option. Then, you can select any of the available colors in the combobox next to the “Enable border” option.

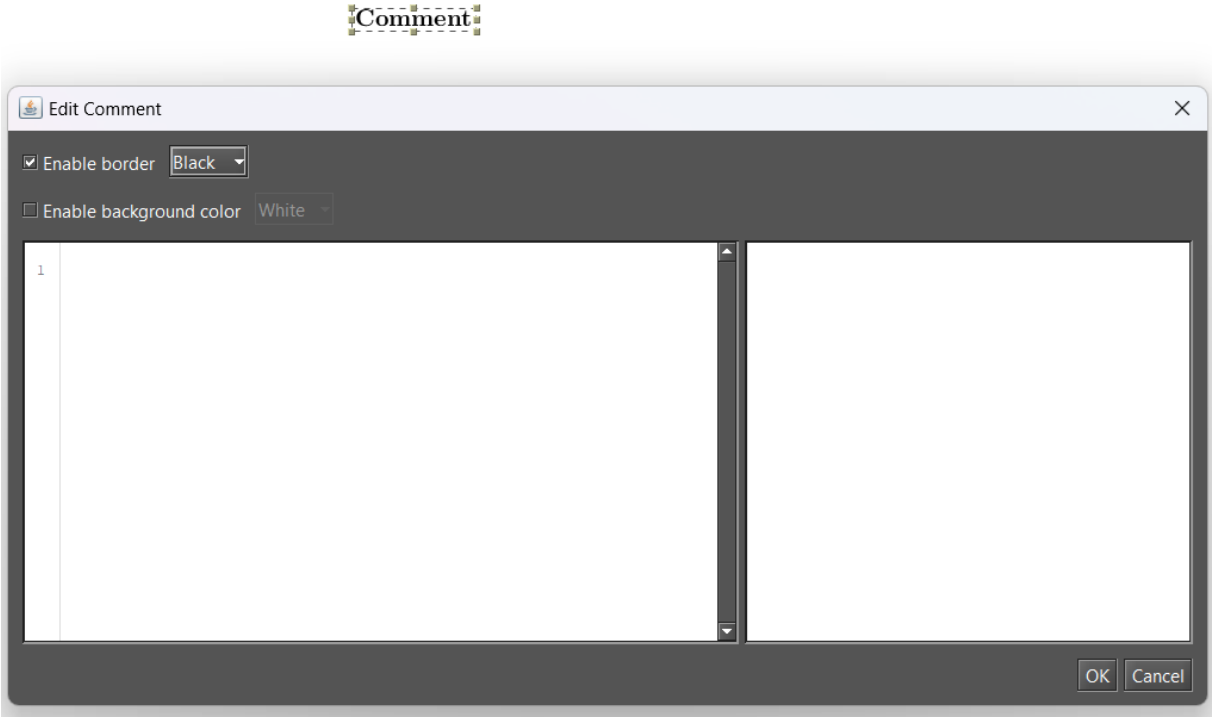


Figure 317: Highlighting with comments

The resulting highlight area is shown in Figure 318. It can be resized as shown in Figure 319 and 320. Finally, the highlight area can be moved anywhere in the drawing area to highlight any element. Figure 321 shows a SPN place highlighted.

Figure 318: Resulting highlight area

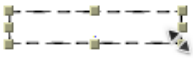


Figure 319: Resizing highlight area

Figure 320: Resizing highlight area result

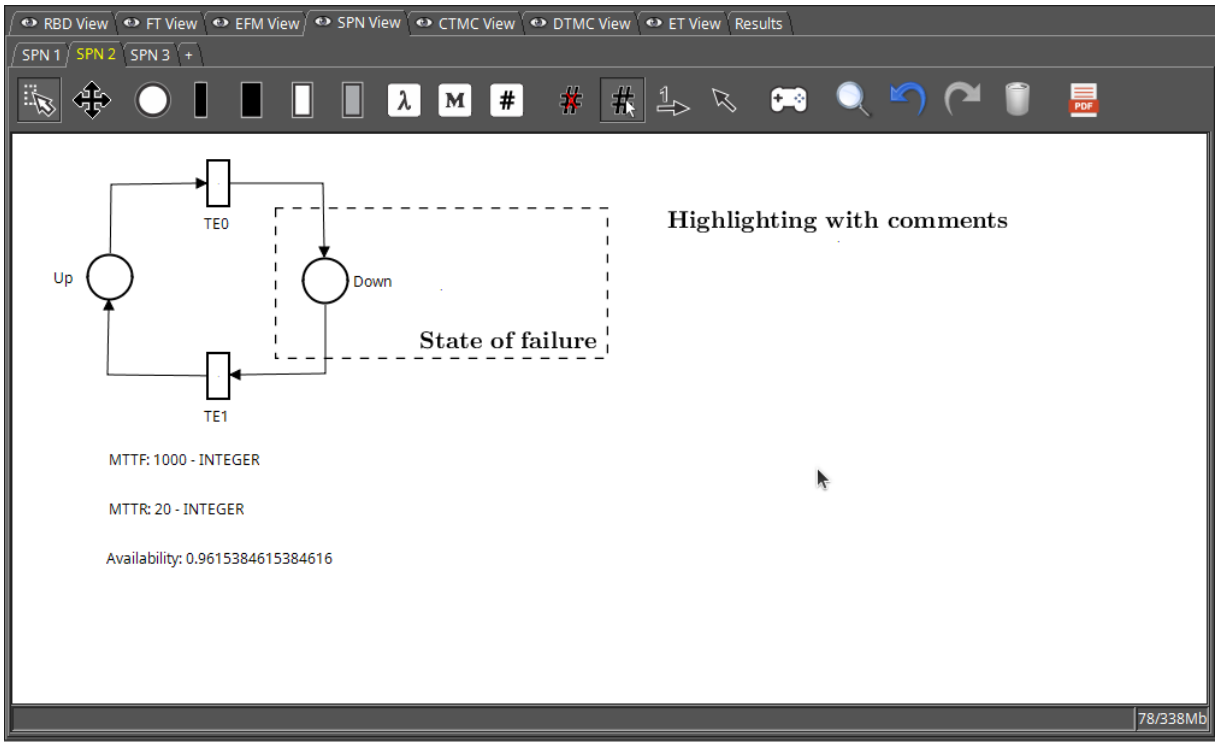


Figure 321: Highlighted element

9.3 Hiding comments

Hiding comments can be done in the RBD and FT views by right-clicking anywhere in the drawing area and selecting the “Hide Comments” option, as shown in Figure 322. This can be done in the SPN, CTMC and DTMC views by clicking the “Hide Comments” button in the toolbar, as shown in Figure 323. When comments are hidden, they can be shown by disabling the “Hide Comments” option available by right-clicking anywhere in the drawing area of the RBD or FT views, or by clicking the “Hide Comments” button in the SPN, CTCM or DTMC views’ toolbar.

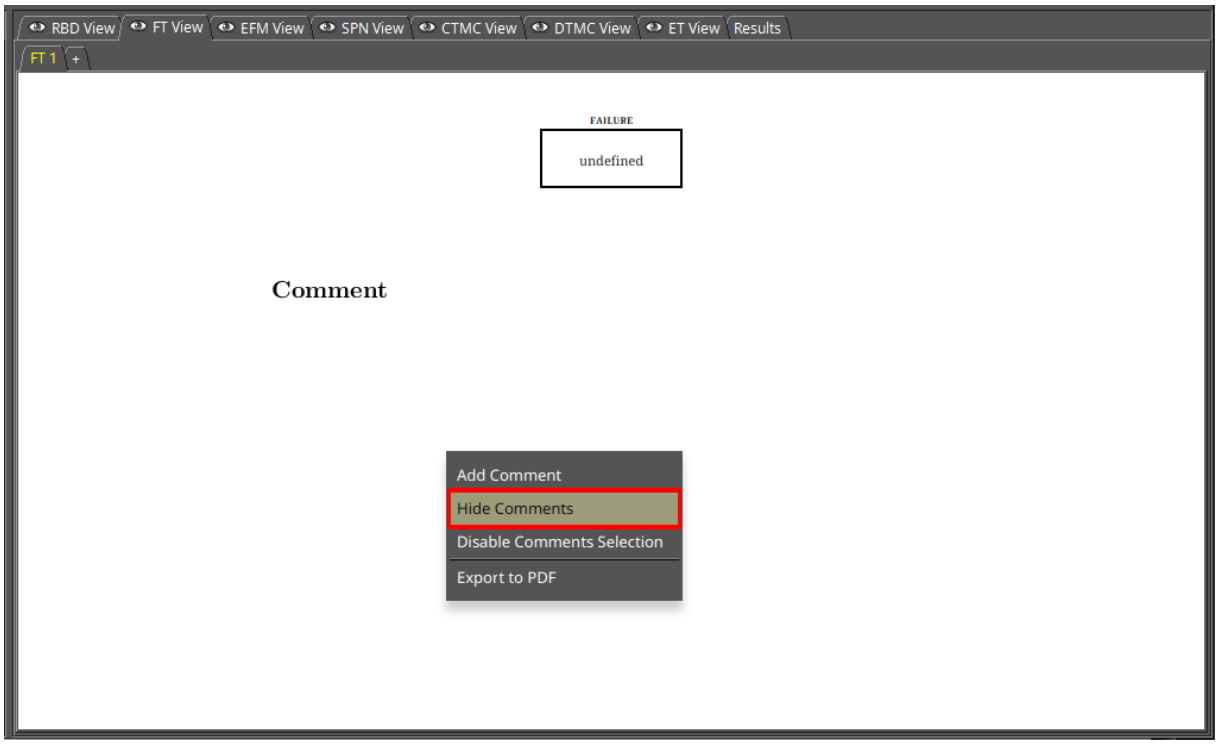


Figure 322: Hiding comments in FT view

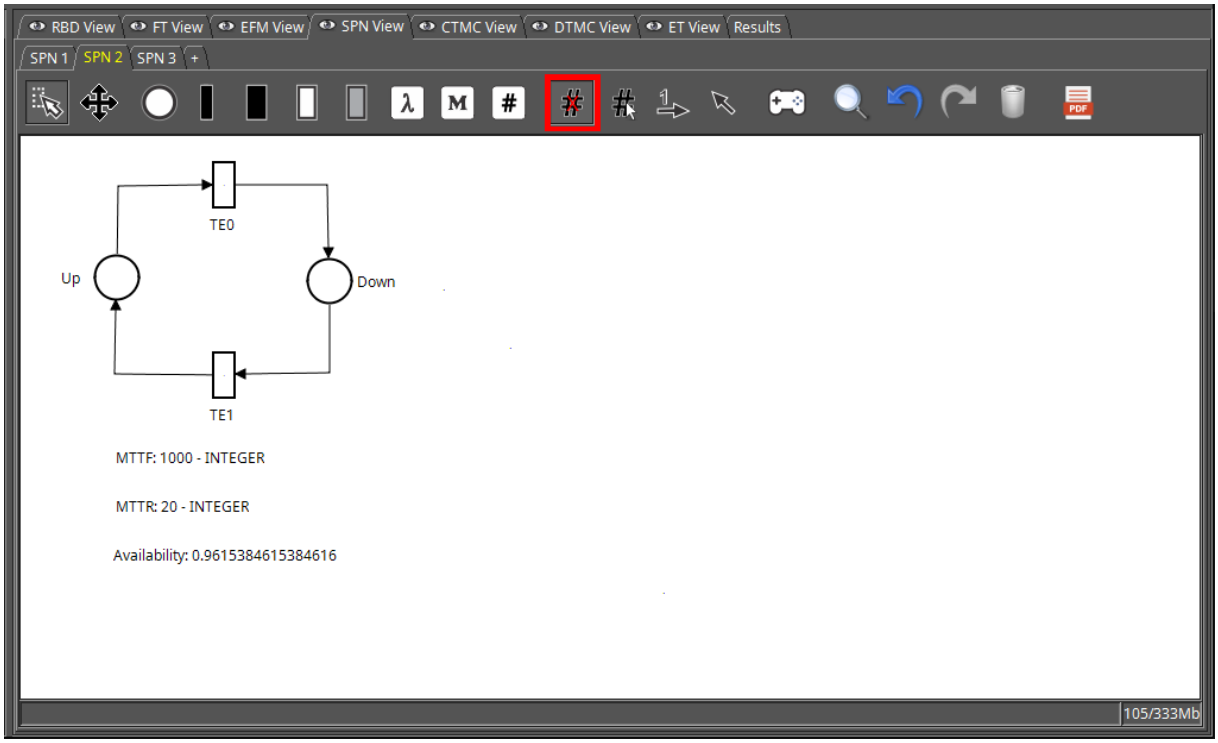


Figure 323: Hiding comments in SPN view

9.4 Disabling comments selection

Disabling comments selection is also possible in the Mercury tool. This option is available in the RBD and FT views by right-clicking in the drawing area, as shown in Figure 324. Additionally, this option is available in the SPN, CTMC and DTMC views through the toolbar button “Disable Comments Selection”, as shown in Figure 325. Re-enabling comments selection can be done following the same step.

Disabling comments selection can be useful to avoid moving comments when you try to move elements of the model. The selected elements of the CTMC as shown in Figure 326, are an example of this.

Disabling comments selection is specially useful in the FT view. Differently from the RBD, SPN, CTMC and DTMC views, in the FT view comments are always in front of the elements of the model, thus, disabling comments selection is essential to manipulate the FT elements.

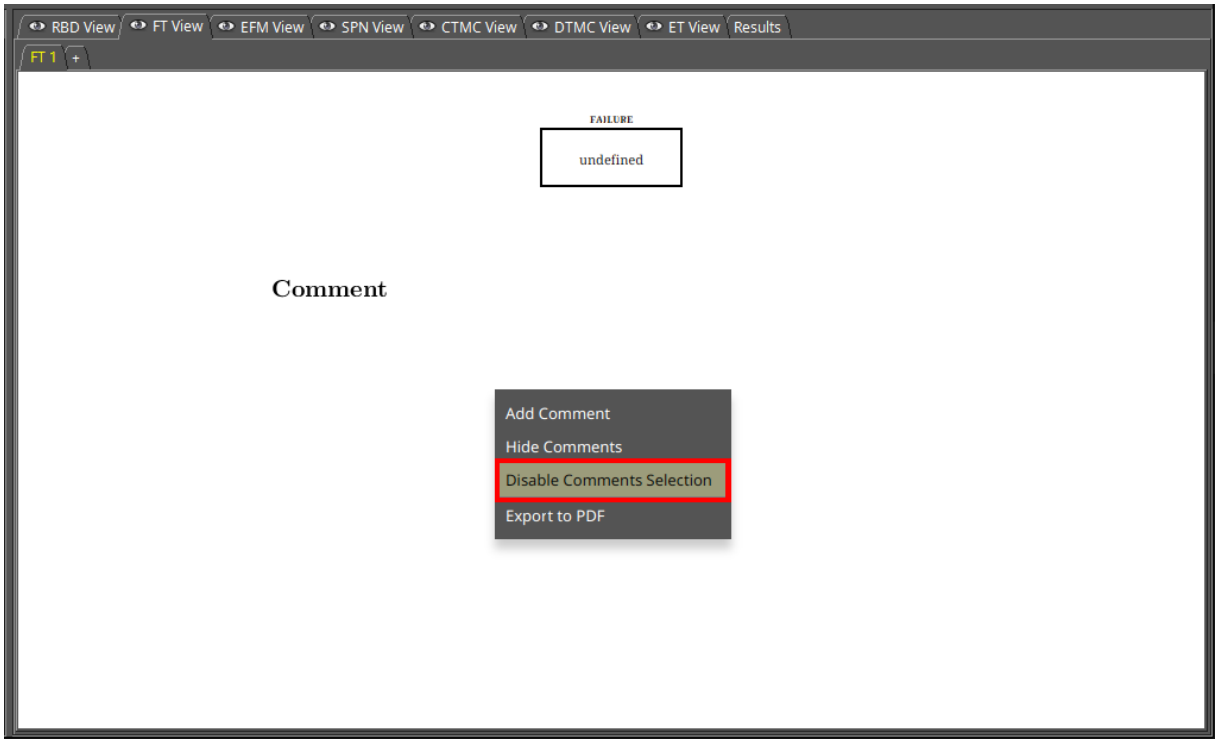


Figure 324: Disabling comments selection in FT view

Figure 325: Disabling comments selection in the toolbar

10 Mercury Scripting Language

10.1 Introduction

The Mercury scripting language was developed to allow more flexibility in evaluating models. To run Mercury scripts, we can use a command line interface (CLI) or access the "Script Editor" available within the Mercury GUI. The advantage of using this language in conjunction with the CLI tool is the ability to automate project workflow, evaluate models, extract metrics, and generate reports and graphs automatically. In addition, the language offers other advantages that are not supported when modeling via the graphical interface.

- Improved support for hierarchical modeling; each model can call another model and use its results as internal parameters.
- Improved support for symbolic evaluations and experiments. The parameters of a model can be defined as variables left open. We can change these variables and re-evaluate the model to measure the impact of these parameters on specific metrics.
- Support for Petri net transitions with a phase-type delay. This family of distributions can be used to approximate any distribution that is not an exponential distribution.

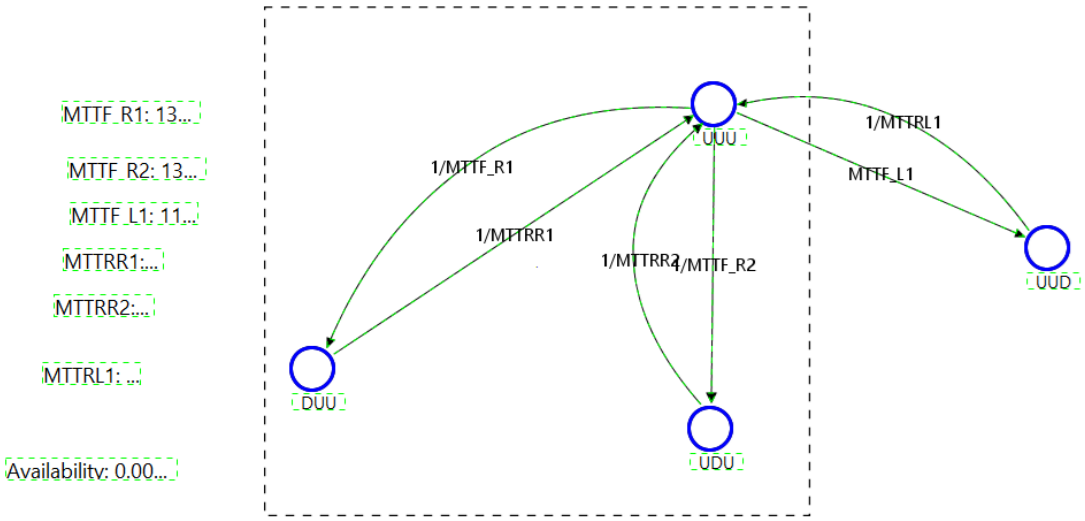


Figure 326: Selecting elements with comments selection disabled in CTMC view

- Support for hierarchical transitions in SPN models. This type of transition can be used to reduce model complexity or to express a recurring structure in the model that can be more easily reused. Some tools [1] [12] support hierarchical SPN models only for colored Petri nets.

10.2 Script Structure

We define the script syntax using BNF notation as follows:

Listing 12: Grammar for Mercury Scripts

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

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

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

A script consists of a model declaration section containing one or more models of the following types: SPN (stochastic Petri net), CTMC (continuous-time Markov chain), DTMC (discrete-time Markov chain), RBD (reliability block diagram), or ET (event tree). Support for the FT and EFM formalisms will be included in the next release. At the end of the model section 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 change variables, solve models and print the obtained results. We change variables to define parameters for a model and collect metric results using the function **solve**. The "for" command has been added to allow us to run experiments. With this command we can change a variable based on a list of values.

In Figure 327 we show a CTMC model, and in Listing 14 we present the corresponding Mercury script. First, we define a CTMC model named **CTMCModel** and declare its states, transitions, and metrics. The transition rates are defined as a function of the parameters **lambda** and **mu**. In the main block, we define values for these parameters and evaluate the "m1" metric of the CTMC model. The result is stored in the variable named **aval**. Finally, we print the content of this variable using the command **println**.

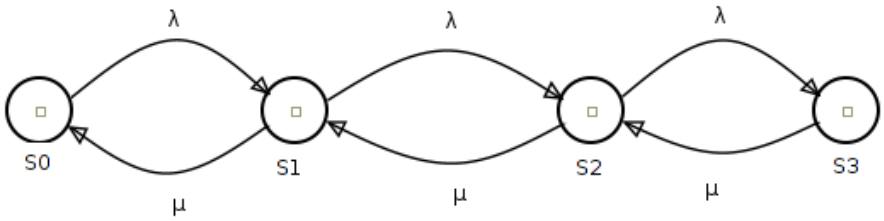


Figure 327: CTMC Model Example

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 m1 = stationaryProbability( st = S0 );
}

main{
    lambda = 0.00001;
    mu = 0.01;

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

10.2.1 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	
event	eventTransition	probability	solveMetric	

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 indicate the state for which we want to evaluate the probability. We cannot use the word "state" because this is a reserved word used to specify states in a CTMC model.

In the following sections, we describe the syntax for each supported formalism: CTMCs, RBDs, SPNs and ETs.

10.3 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. For availability models, a state may also receive an annotation *up* after the identifier. This annotation defines states in which the system is considered operational.

Listing 15: Grammar for CTMC Models

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

<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) steady-state probability; and iv) transient probability.

10.3.1 Availability

The availability metric does not require any parameters. This metric returns the sum of all stationary probabilities for states annotated with the keyword *up*. 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=lambdai_s2);
    transition uf -> ff(rate=lambda_s1);

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

    metric aval = availability;
}
```

10.3.2 Reward Metric

This metric calculates the sum of rates associated with each state. The parameters defined for this metric are a list of pairs: $< state\_name >=< value >$. The metric calculates the sum of the products of each rate and the stationary probability associated with the state. The states that do not receive a rate are implicitly associated with a zero rate. Below we list an example of a model with a reward metric. We recommend that the reader check that the metrics m1 and m3 give 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 );
}
```

10.3.3 Stationary and Transient Probabilities

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

In Mercury language, we use the metric **stationaryProbability(st = S)** to obtain the stationary probability associated with a state *S*. For the transition probability, we also need to specify the time *T* and the initial probability for each state. This corresponds to the probability that the model is in that state at time T = 0. In the script syntax, the states that are not specified in the list of initial probabilities are given an initial probability of 0. It is important to emphasize that the sum of all initial probabilities must equal 1, otherwise an exception will be thrown.

In the following, we will show how to obtain the metrics for stationary and transient probabilities using 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 )
    );
}
```

10.4 Reliability Block Diagram

An RBD model consists of:

- Exponential blocks representing components with an associated parameter for mean time to failure and mean time to repair;
- Hierarchical blocks evaluated by calling other external models;
- Series/parallel arrangements of other blocks;
- Declaration of top-level block; and
- 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 for RBD models:

- availability;
- mean time to failure (MTTF);
- mean time to repair (MTTR); and,

- reliability.

The first three metrics do not require parameters: Steady-State Availability, MTTF, and MTTR. The reliability and instantaneous availability metrics, on the other hand, require an *time* parameter. Considering the model shown in Figure 328, we created its script definition as shown in Listing 20.

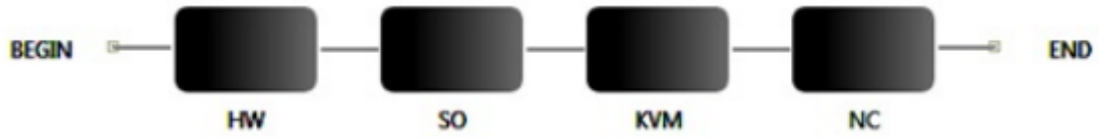


Figure 328: RBD Representing a Cloud Node [13]

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 );
}
```

10.5 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. There are three types of transitions: immediate, timed, and substitution. "Substitution" transitions allow us to create modular and reusable Petri nets. This functionality is only available in the scripting language. Another exclusive feature of the scripting language is support for *phase-type* distributions. In this section, we will show a simple SPN as an example that contains only exponential and immediate transitions.

Listing 21 shows the grammar in the notation BNF for describing SPN 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 as parameters within the transitions: *inputs*, *outputs*, and *inhibitors*. Timed transitions have delay and server type as parameters. If the "server type" parameter is omitted, it defaults to "SingleServer". Immediate transitions have as parameters (optional): weight, priority and an enabling function. Metrics are defined in the form of a string representing a reward metric, which is also 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 with the Mercury scripting language, we have proposed a model for an M/M/1/K queue based on the [14]. This model is shown in Figure 329.

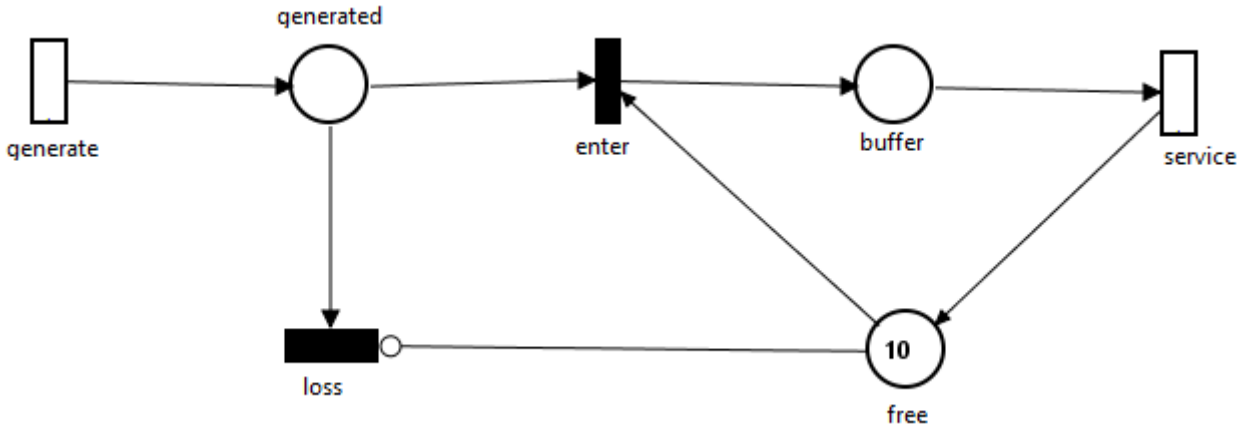


Figure 329: SPN Model Representing an M/M/1/k Queue[14]

The *generate* transition generates tokens corresponding to service requests. Each generated token is stored in the *generated* place and then a selection 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, which is represented by triggering the immediate ”loss” transition. The ”free” place controls the triggering of this transition by an inhibitor arc. The tokens waiting in the queue are placed in the ”buffer”. The transition ”service”

represents the processing of the requests. Since this is an M/M/1/K queue, we have only one server that can handle all requests. Therefore, the server semantics assigned to the transition ”service” is SINGLE SERVER.

Listing 22 shows the script for running the stationary analysis of the SPN model described earlier. The parameter **method** of the function **stationaryAnalysis** can only have 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 for this model. Below we describe each parameter of thefunction **stationarySimulation**.

- **confidenceLevel.** The confidence interval for determining the metrics.
- **maxRelativeError.** Defines the maximum relative error, which is one of the stop conditions of the simulation.
- **minFiringTransitions.** Defines the minimum number of firings for each transition in the model. This number of firings is another condition for stopping the simulation. If you enter a value of 0, the simulator does not consider the number of firings to stop the simulation. If you enter a value greater than 0, the simulation will stop when the number of firings for each transition is equal to the specified value.
- **warmup.** Sets the minimum warm-up period. The warm-up phase is the period when the model is not in steady state and the metrics are not collected during this period. There are a few methods to estimate whether the model has entered a steady state phase, but Mercury requires the user to set a value for the warm-up phase. Since we are evaluating stochastic models, it is expected that the warm-up time will not be an accurate value for every simulation run. Therefore, the user defines a minimum warm-up time. Once the global simulation time is equal to or greater than the user-defined warm-up time, the simulation begins collecting metrics, generating batches, and calculating statistics.
- **batchsize.** Sets 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 specified in seconds. If it is set to 0, the simulator will 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), the simulation will stop when this time is reached.

Listing 23: Script for Stationary Simulation

```
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");

        m1 = solve( Model,m1 );
        println(m1);
}
```

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

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

Listing 24: transientSimulation Function

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

10.6 Event Tree

In the Mercury scripting language, an Event Tree is described by nodes, which represent the events of the model, and by transitions between the nodes, which are described in the form of event probabilities. There are three types of nodes: initial, intermediate, and leaf.

An Event Tree script can be created by defining an ET object, with a name and a scope. In the ET object scope, *event* and *eventTransition* must be defined to create the event and the transitions between events, as shown in Listing 25. The script's syntax does not differ between the different types of nodes. Metrics can also be defined in the script.

Listing 25: Grammar for ET Models

```
<et_model>  ::= "ET" <identifier> "{"
<et_statements>
"}"

<et_statements> ::= <et_statement> ";" <et_statements> |
                    <et_statement> ";"
```

```
<et_statemnt> ::= <et_statement> |
                        <event_statement> |
                <event_transition_statement> |
                        <metric_statement>

<event_statement> ::= "event" <identifier> ";"

<event_transition_statement> ::= "eventTransition" <identifier>
    "->" <identifier> "("
        "probability" "=" <numeric_expression> ")" ";"

<metric_expression> ::= "metric" <identifier> = "solveMetric" "("
    "expression" "=" <numeric_expression> ")" ";"
```

To illustrate the syntax for modeling ET with the Mercury scripting language, we have proposed the following model, which is shown in Figure 330.

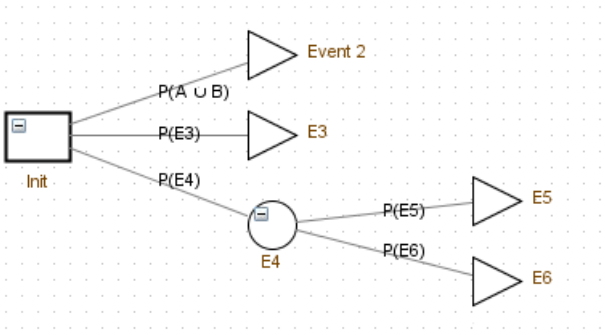


Figure 330: Example ET Model

This model has four terminal events (leaf nodes), *Event 2, E3, E5* and *E6*; one initial event (initial node), *Init*; and one intermediate event (intermediate node), *E4*. The arcs that connect the events are the event transitions, where each probability of the event transition is a definition (variable), and their values must be less than 1.

Listing 26 shows the generated script for ET model described earlier.

Listing 26: Script for Evaluate an Event Tree

```
ET Model{

        event Init;
        event Event 2;
        event E3;
```

```
        event E4;
        event E5;
        event E6;

        eventTransition Init -> Event 2 ( probability = 0.2);
        eventTransition Init -> E3 ( probability = 0.3);
        eventTransition Init -> E4 ( probability = 0.5);
        eventTransition E4 -> E5 ( probability = 0.4);
        eventTransition E4 -> E6 ( probability = 0.6);

        metric m0 = solveMetric (expression="P{Event 2}" );

}
main{

        m0 = solve( Model, m0 );
        println( "m0: " .. m0 );

}
```

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 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-standy 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{""}" = probability of being in the declared state;

"R{""}" = reward rate of being in the declared state;

"R{}" = steady-state reward of the system; and

"LOG{""}" = base-10 logarithmic function.

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

Listing 27: Syntax of Components for CTMC Models

```
<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>} ''}''

<logarithmic_measure> ::= ''LOG{''<expression>''}''

<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.

In this section, we present the specification in terms of SPN metrics, guard expressions, and arc multiplicity dependent on marking. We present a formal syntax description using the Backus-Naur form (BNF).

Three different expressions can be used in the Mercury tool (see Listing 28). SPN expressions are represented as “Metrics”, “GuardExpressions”, and “MarkingDependentMultiplicites”. “Metrics” are used to represent the evaluated metrics and can be a probability, an expectation, or a base-10 logarithmic function taking the value of a given expression as input. “GuardExpressions” are used to represent logical expressions to enable/disable the firing of transitions. “MarkingDependentMultiplicites” are numeric expressions that are evaluated as a function of the current marking to a particular arc multiplicity.

Listing 28: Syntax of Components for SPN Models

```
<Metric> ::=      ''P{''<logic_condition>''}''
               | ''E{''<marking_function>''}''
               | ''LOG{''<expression>'', base}''
               | ''POWER{<expression>, exponent}''
               | ''MIN{<expressionList>}''
               | ''MAX{<expressionList>}''

<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 elements of a given expression are separated by parentheses. For example, suppose we want to evaluate the probability that there are more than two tokens in place P1 and zero tokens in place P2. The corresponding expression is:

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

IMPORTANT. Spaces within expressions are not allowed. Therefore, the following expression is not allowed.

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

In general, guard expressions consist of various comparisons composed of ANDs, ORs, and NOTs. For example, let us look at the following expression:

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

This expression can be used as an activation function to trigger a transition only if the place P1 has one token and P2 has two tokens. Again, spaces are not allowed within the expressions and all subexpressions must be joined by parentheses.

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

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 the language, these expressions can change the place marking based on other place markings. For example, 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 the marking of P1 should be

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

It is also possible to have nested if-else expressions. To explain this, we extend the previous example and assume that the model has 4 places (P1, ..., P4) and the marking of P1 is 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 defined as follows:

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

This expression is similar to the nested if, elseif, ..., else expressions in standard programming languages such as C or Java.

C EMA Tool.

The Expectation-Maximization (EM) algorithm is an iterative technique that allows estimation of parameters in statistical models with incomplete or hidden data. In the context of the EM algorithm:

- **Expectation (E-step)**: The conditional probability of the hidden data is estimated based on the current parameters.
- **Maximization (M-step)**: The parameters are updated to maximize the expected value of the log-likelihood.

Each point in the set has a certain probability of belonging to a certain cluster. However, these probabilities are initially unknown. Moreover, we face another challenge: the parameters that define the distribution of each cluster are also unknown. Amidst these uncertainties, we introduce the Erlang-r distribution, a relevant choice where the value of "r" represents the number of phases of the distribution. Given this complexity, we could compute the maximum log likelihood, which is ideally the probability of the data. However, due to the hidden or unknown nature of clusters, computing this likelihood directly becomes complicated. As a solution, we work with the expectation of the incomplete log likelihood, which is maximized to find appropriate estimates of the unknown parameters.

Key formulas:

- Probability of point X_i belonging to cluster Z_i:

$$Z_i \sim \text{Categorical}\,(\pi_1, \pi_2, \ldots, \pi_n)$$

$$X_i \sim \text{Dist}\left(\mu_i\right)$$

$$P\left(Z_i = j\right) = \pi_j$$

- Erlang-r distribution:

$$f_X\,(x) = \frac{\mu\left(\mu x\right)^{r-1} e^{-\mu x}}{(r-1)!}$$

- Maximum log-likelihood expectation:

$$Q\left(\theta, \theta^0\right) = \sum_{i=1}^{N} \sum_{k=1}^{K} A_{i,k} \left[\ log P\,(X_i | Z_i = k, \theta) + \log \pi_k\right]$$

- Posterior probability:

$$A_{i,k} = P\left(Z_i = k \middle| X_i, \theta^0\right) = \frac{P\left(X_i \middle| Z_i = k, \theta^0\right) \pi_k}{\sum_{k'=1}^{K} P\left(X_i \middle| Z_i = k', \theta^0\right) \pi_{k'}}$$

- Maximization:

  For π:

$$\pi_j = \frac{\sum_{i=1}^{N} A_{i,j}}{\sum_{k=1}^{N} (A_{k,1} + A_{k,2} + \ldots + A_{k,n})}$$

  - For θ:

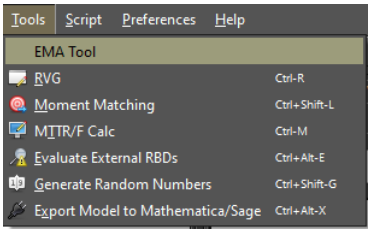
$$\theta_j = \frac{r \sum_{i=1}^{N} A_{i,k}}{\sum_{i=1}^{N} X_i A_{i,j}}$$

The EMA algorithm consists of the following steps:

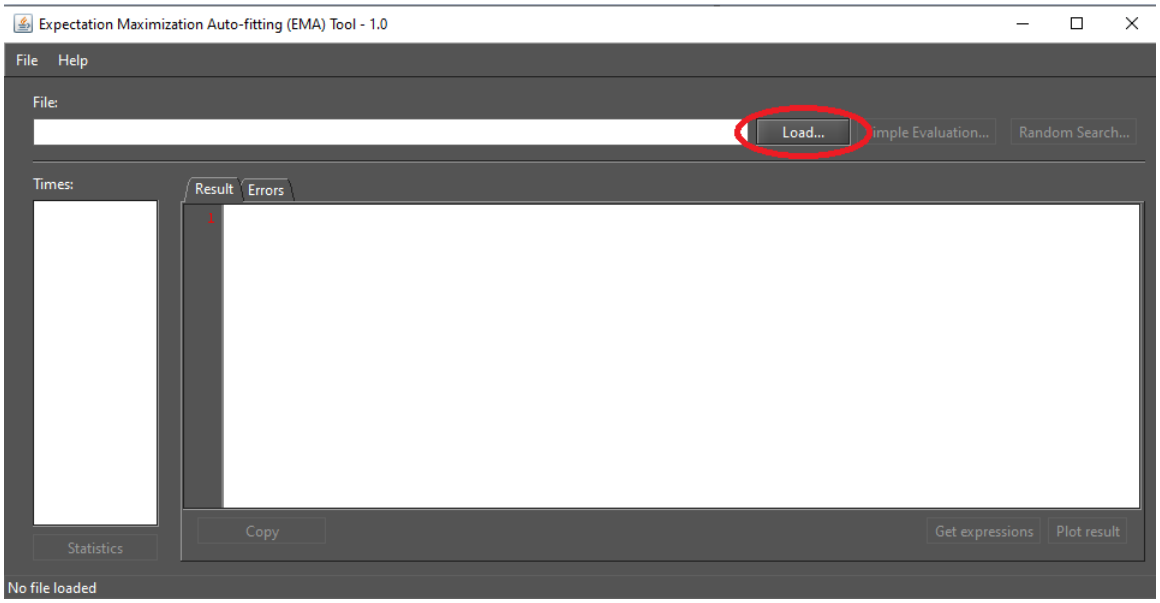
1. Initialize the parameters;
2. Derive the log likelihood expectation;
3. Calculate posterior probabilities;
4. Using posterior probabilities, find and update the optimal parameters; and
5. Repeat steps 2 to 4 until convergence.

To perform an evaluation using the EMA tool through Mercury:

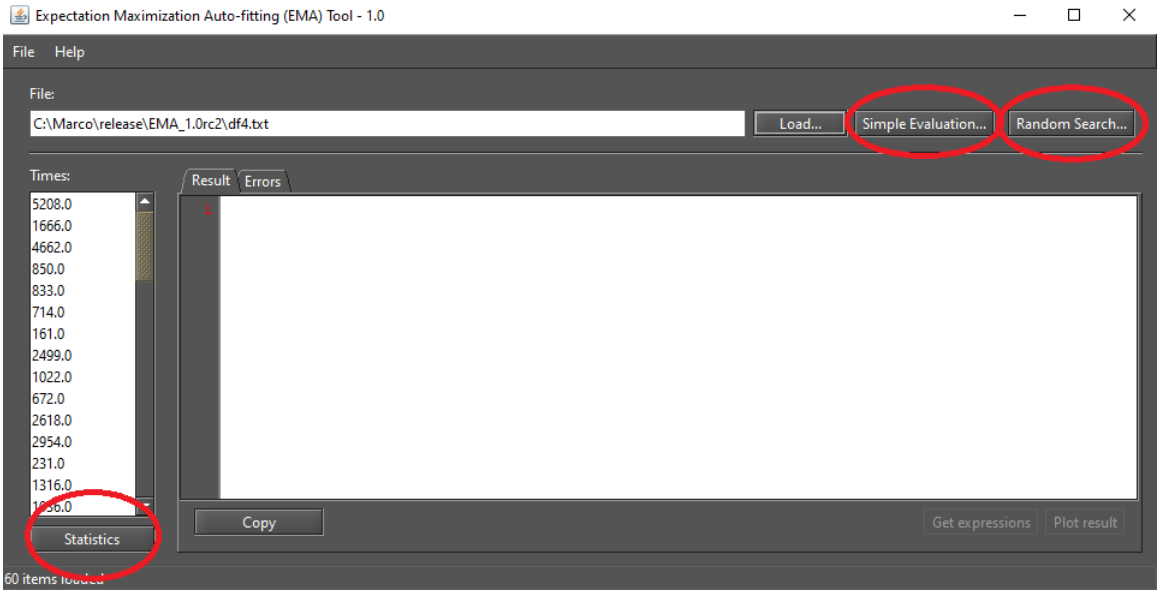
1. Open the EMA tool from the Tools menu.



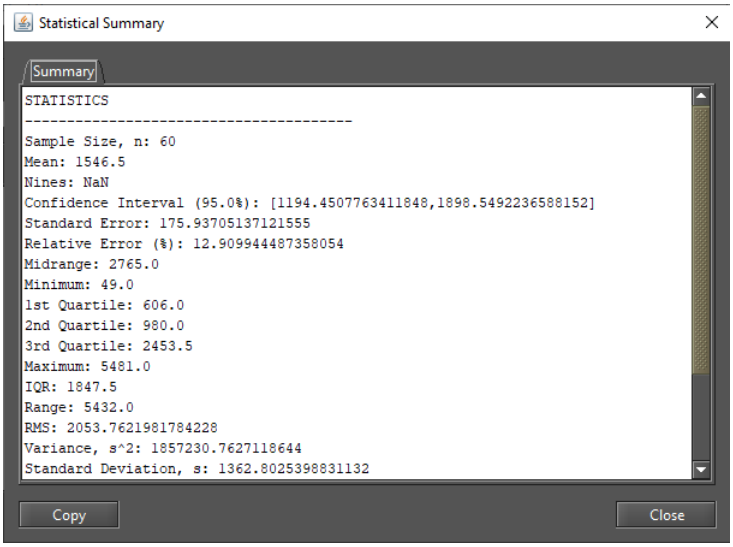
2. Load the dataset by clicking the Load button.



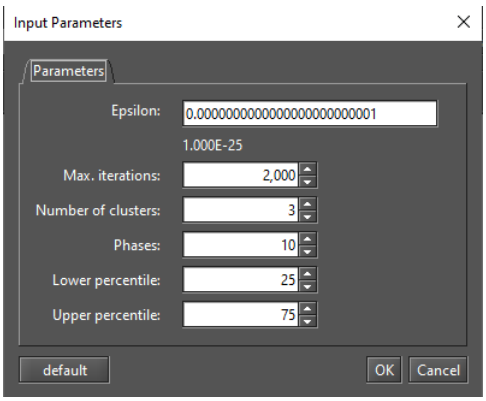
There are now three actions available:



3. **Statistics**. Selecting this option will display a summary of the statistics for the dataset, as follows:

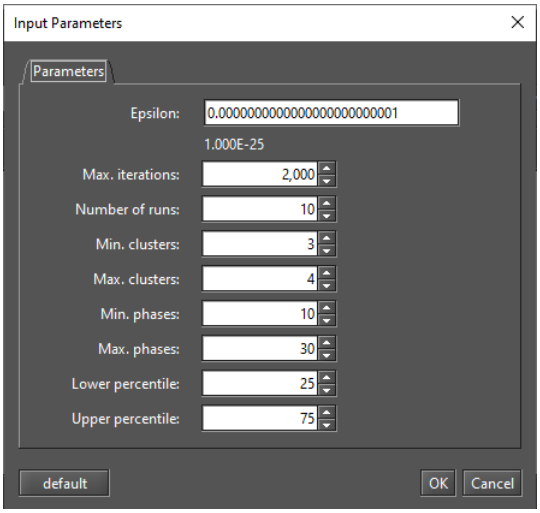


4. **Simple Evaluation**. A menu for the fitting will be displayed. The parameters are:



- **Epsilon**: Stopping criterion of the EM algorithm.
- **Max. iterations**: Maximum number of iterations (will be used if Epsilon is not reached).
- **Number of clusters**: Number of clusters to be used in the fitting process.
- **Phases**: Number of distribution phases in each Cluster.

5. **Random Search.** The process of determining the number of phases in the fitting is not always trivial. To assist this process, we have implemented an algorithm that randomly searches for the number of clusters and phases based on the Bayesian Information Criterion (BIC). BIC is a criterion for model selection from a finite set of models. Lower values for BIC are generally preferred. The parameters for the random search are:



- **Epsilon**: Stopping criterion of the EM algorithm.
- **Max. iterations**: Maximum number of iterations (will be used if Epsilon is not reached).
- **Number of runs**: Number of runs in which the algorithm determines random values for the parameters.
- **Min. cluster**: Minimum number of clusters used in the fitting process.
- **Max. clusters**: Maximum number of clusters used in the fitting process.
- **Min. phases**: Minimum number of distribution phases in each cluster.
- **Max. phases**: Maximum number of distribution phases in each cluster.

The Mercury Scripting Language Cookbook

Danilo Oliveira

Abstract This document presents a series of modeling problems, and shows how the Mercury Scripting Language (MSL) makes easy to solve them. Our objective is to show the strong points of the language in different pratical contexts.

1 Introduction

MSL (*Mercury Scripting language*) is a language provided by the Mercury tool [1] [1] for creating and evaluating models. The scripts can be executed by command line interface (CLI), via graphical interface, or inside a Java program (as we will show later in this document). The main objective of the scripting language is to allow the use of Mercury evaluation engine with greater flexibility than the GUI provides. The scripting language enables, for example, using shell scripts to automate an evaluation workflow. The scripting language offers an additional advantage that is the increased support to hierarchical evaluation [2] [3]. Input parameters of any model can be defined as function of an output metric defined by another model, independent of the modeling formalism. The Mercury scripting language currently supports SPN (Stochastic Petri Net), RBD (Reliability Block Diagram), EFM (Energy Flow Model) and CTMC (Continuous Time Markov Chain) models.

We structured this material as a "*cookbook*", that provides a series of "*recipes*". Each recipe contains a practical example that emphasizes some capability of the language:

Danilo Oliveira
Centro de Informática, UFPE, e-mail: dmo4@cin.ufpe.br

[1] Available at: http://www.modcs.org/?page_id=1630

- **Recipe #1 - Hierarchical modelling + composite metrics** — In this example we show how to structure a set of models hierarchically, and how to declare a metric that is function of another metric;
- **Recipe #2 - Experiments and sensitivity analysis** — In this example we show how to perform quick experiments with the *for* loop, and how to perform sensitivity analysis with the *percentage difference*, and *design of experiments* techniques;
- **Recipe #3 - Reliability block diagram with variable number of blocks** — This example shows how to define a RBD model that does not have a fixed number of blocks, and how to change this model;
- **Recipe #4 - Energy Flow model + availability model** — This example shows how to link an energy flow model with a dependability model in order to compute certain metrics. This linkage turns possible to see how the dependability parameters impacts the energy flow model metrics;
- **Recipe #5 - Using phase type distributions in a Stochastic Petri Net Model** — In this example we see how to use a phase-type distribution (e.g.: hypoexponential, hyperexponential, Erlang) as delay on a SPN model;
- **Recipe #6 - Sensitivity analysis on a performability model** — In this example we illustrates how to compose a performability model by composing a dependability and a performance model. Then, we show how to measure the impact of the dependability parameters on performance metrics;
- **Recipe #7 - Executing a script inside a Java program** — In this example we show how to run scripts from a Java program, and obtaining references to model and metrics programatically.

2 Recipes

2.1 Recipe #1 - Hierarchical modelling + composite metrics

In this recipe, we use as example the hierarchical model for the non-redundant cloud architecture presented in [4]. Figure 1 show the top level RBD model for the system. It is composed by a frontend server, a node server, a storage volume, and the video streaming service that runs on the VM deployed in the node. Except the storage volume components, all blocks are evaluated by another submodels. The frontend and the node servers are modeled by RBDs displayed on figures 2 and 3, respectively. The video streaming service is modeled by the CTMC presented in Figure 4.

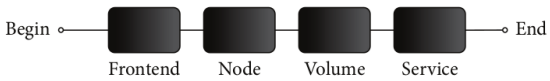


Fig. 1

Fig. 2

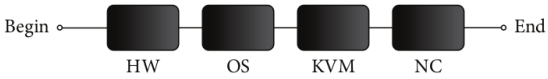


Fig. 3

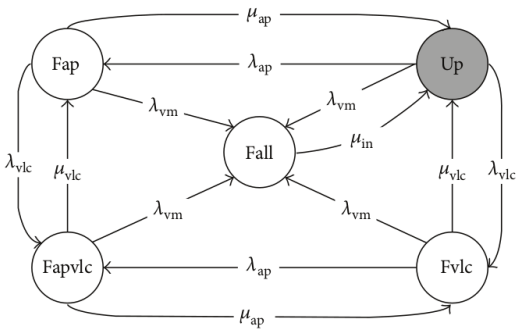


Fig. 4

The script for evaluating the system's availability is presented on Listing 1. On the preamble off the script, we set all the parameters for all models. After the preamble, we start declaring the models. Since the script is evaluated once, and there is no forward declarations, we must declare the submodels before the top level model. If a model A use the result of a metric defined by a model B as one of its parameters, the model B must be declared before the model A.

The models NRFrontend, and Node are simple RBDs composed by a set of blocks with exponential MTTF and MTTF. Each exponential block is declared using the "block" keyword followed by the MTTF and MTTR parameters enclosed by parenthesis. The value of a model parameter can be any numeric expression containing: parenthesis, arithmetic operators (*, +, /, -), number literals, variables, and functions. The availability metric is declared

using the “metric” keword, followed by the metric identifer, and the metric type. In this case, we use the “availability” metric, that takes no parameters.

The model NRService is represented by a CTMC with five states, and in only one state (the “Up” state) the system is available. For CTMC availability models, there is three ways to compute the availability. The first is to annotate the “up states” with the “up” keyword, and use the “availability” metric. Alternatively, we can use CTMC expressions following the syntax described in the Mercury manual, by using the “ctmcExpression” metrics, that takes as parameter a string containing the expression. The “up” annotation and the “availability” metric is a shorthand for availability models, while the “ctmcMetric” is a more general way to compute CTMC metrics.

On the top level model, we create a RBD with a series arrangement of blocks. But, instead of using simple exponential blocks, we use hierarchical blocks, by using the “hierarchy” keyword. For availability models, we must define the “availability” parameter for each hierarchical block. We can use any numeric expression for this parameter. Using the “solve” function allow us to solve a metric defined by another model and, in this way, composing models hierarchically. The NRArchitecture model represents the top level model depicted in Figure 1. It defines three metrics. The metric named “av” computes the steady state availability for the RBD. The metrics named “uav” and “downtime” represents composite metrics, i.e., metrics that are defined as function of another metrics. A composite metric is declared by using a numeric expression enclosed by parentheses.

Finally, we create the “main” block, that is the section of the script where we can set parameters, evaluate models, and print the computed results using the “print” and “println” functions. These functions accepts any numeric or string expression as argument, and outputs its values in the console. Using the “..” operator we can concatenate string with numeric values and presenting a more readable output.

```
lambda_ap = 1/788.4;
mu_ap = 1;
lambda_vlc = 1/336;
mu_vlc = 1;
lambda_vm = 1/2880;
mu_vm = 1;
mu_in = 1/0.019166;

mttf_hw = 8760;
mttr_hw = 100/60;
mttf_os = 2895;
mttr_os = 1;
mttf_kvm = 2990;
mttr_kvm = 1;
mttf_nc = 788.4;
mttr_nc = 1;
mttf_clc = 788.4;
mttr_clc = 1;
mttf_cc = 2788.4;
```

```
mttr_cc = 1;
mttf_walrus = 2788.4;
mttr_walrus = 1;

mttf_volume = 100000;
mttr_volume = 1;

RBD NRFrontend{
  block hw( MTTF = mttf_hw, MTTR = mttr_hw );
  block os( MTTF = mttf_os, MTTR = mttr_os );
  block cc( MTTF = mttf_cc, MTTR = mttr_cc );
  block clc( MTTF = mttf_clc, MTTR = mttr_clc );
  block walrus( MTTF = mttf_walrus, MTTR = mttr_walrus );

  series s1(hw, os, clc, cc, walrus);

  top s1;

  metric av = availability;
}

RBD Node{
  block hw( MTTF = mttf_hw, MTTR = mttr_hw );
  block os( MTTF = mttf_os, MTTR = mttr_os );
  block kvm( MTTF = mttf_kvm, MTTR = mttr_kvm );
  block nc( MTTF = mttf_nc, MTTR = mttr_nc );

  series s1(hw, os, kvm, nc);

  top s1;

  metric av = availability;
}

CTMC NRService{

  state fap;
  state Up up;
  state fapvlc;
  state fvlc;
  state fall;

  transition fap -> Up( rate = mu_ap);
  transition fap -> fapvlc( rate = lambda_vlc);
  transition fap -> fall( rate = lambda_vm);
  transition Up -> fap( rate = lambda_ap);
  transition Up -> fvlc( rate = lambda_vlc);
  transition Up -> fall( rate = lambda_vm);
  transition fapvlc -> fap( rate = mu_vlc);
  transition fapvlc -> fvlc( rate = mu_ap);
  transition fapvlc -> fall( rate = lambda_vm);
  transition fvlc -> Up( rate = mu_vlc);
  transition fvlc -> fapvlc( rate = lambda_ap);
  transition fvlc -> fall( rate = lambda_vm);
```

```
  transition fall -> Up( rate = mu_in);

  metric av = availability;
  metric av2 = ctmcExpression( expression = "P{Up}" );
}

RBD NRArchitecture{
  hierarchy frontend(
    availability=solve(NRFrontend, av)
  );

  hierarchy node(
    availability=solve(Node, av)
  );

  block volume( MTTF = mttf_volume, MTTR = mttr_volume );

  hierarchy service(
    availability=solve(NRService, av)
  );

  series s1( frontend, node, volume, service);

  top s1;

  metric av = availability;
  metric uav( 1 - av );
  metric downtime( uav * 365 * 24 );
}

main{
  a = solve( NRArchitecture, av );
  println("Availability: " .. a);

  u = solve( NRArchitecture, uav );
  println( "Unavailability: " .. u);

  d = solve( NRArchitecture, downtime );
  println("Annual downtime: " .. d);
}
```

Listing 1 Script for a cloud video streaming service model

2.2 Recipe #2 - Experiments and sensitivity analysis

In this recipe, we will use the model defined in the previous recipe. One of the greatest strengths of the scripting language is the ease to change a model parameter and observe how the metrics reacts by this change. By modifying the contents of a variable that is used as a model parameter (using the assign operator "="), and re-solving the metric of interest using the "solve" function,

we will get a new result that corresponds to the updated model. The relying on this feature, the scripting language provides a convenient way to perform experiments, i.e., outputting the result of a metric by changing a parameter over a list, and provide powerful functions to perform sensitive analysis.

2.2.1 Performing experiments using the "for" loop

To observe how a metrics reacts by changing a parameter over a list of values, we use the "for" loop available in the scripting language. Listing 2 shows an example. In this example, we change the "mttf_hw" parameter over the list enclosed by square brackets. For each iteration, we solve the "av" metric of the NRArchitecture model. Then, we print the parameter value, followed by a comma, and the metric result. By outputting the result this way, we can create a comma separated value (CSV) file that will be used by producing a chart by another tool (e.g. R, gnuplot, Excel, etc.)

In Figure 5 we show how to run the script in the command line interface. We run the Mercury program by typing "java -jar mercury.jar" on the terminal. Without any command line arguments, the graphical interface will be displayed. To evaluate the script, we must pass the "-cli" argument, followed by the file name of the script. Any "print" or "println" output will be sent to the console. To save the results in a file, we can use the redirect operator ">" of the shell.

```
main{
  for mttf_hw in [ 730, 1460, 2190, 2920, 3650, 4380, 5110, 5840,
     6570, 7300, 8030, 8760 ]{
    a = solve( NRArchitecture, av );
    println(mttf_hw .. ", " .. a);
  }
}
```

Listing 2 Performing an experiment with the for loop

2.2.2 Sensitivity analysis

The language has two predefined functions for sensitivity analysis: percentage difference, and design of experiments. The technique of sensitivity analysis by means of percentage difference consists in changing one parameter over a list of values, while holding the other parameters fixed, and calculating the percentage difference in the output metric considered. We perform this step for each parameter into our list, and sort them from the highest difference to the lowest. The formula for obtaining the percentage difference is [5]:

$$SI = \frac{D_{max} - D_{min}}{D_{max}}$$

```
$ java -jar mercury.jar -cli script.mry
730.0, 0.986992368139226
1460.0, 0.9892444891827497
2190.0, 0.9899969093793102
2920.0, 0.9903734414323411
3650.0, 0.9905994637837322
4380.0, 0.9907501883396075
5110.0, 0.9908578697990109
5840.0, 0.9909386424146657
6570.0, 0.9910014723886486
7300.0, 0.9910517406703346
8030.0, 0.9910928721093644
8760.0, 0.9911271502646058
$ java -jar mercury.jar -cli script.mry > output.csv
$
```

Fig. 5

, where SI is the sensitivity index for the selected parameter, D_{max} is the maximum value of the output metric, and D_{min} is the minimum value.

The Listing 3 shows the usage of the *percentageDifference* function. This function has three mandatory parameters: i) *model_* , which defines the model that will be used in the analysis; ii) *metric_* , which defines the metric from the model that will be used; iii) *parameters*, which represents the list of parameters that will be used in the analysis, and its respective values. Each parameter is set with a list of values enclosed by square brackets. Optionally, the user can define only two values: a minimum and a maximum value, and set the *samplingPoints* parameters, that specifies the number of intermediate points that will be generated for each list. The function prints in the console the list of parameters and respective sensitivity indexes, ranked from least to most influential parameter. Figure 6 shows the output for the script displayed on Listing 3.

```
main{
    av = solve( NRArchitecture, av );
    println(av);

    percentageDifference(
        model_ = "NRArchitecture",
        metric_ = "av",
        samplingPoints = 5,

        parameters =  (
            lambda_ap = [ 1/2000, 1/788.4],
            mu_ap = [ 1/5, 1 ],
            lambda_vlc = [ 1/500, 1/336 ],
            mu_vlc = [ 1/5, 1],
            lambda_vm = [ 1/4000, 1/2880 ],
```

```
            mu_vm = [ 1, 5 ],
            mu_in = [ 1/0.019166, 1/0.1 ],
            mttf_hw = [ 8760, 10000 ],
            mttr_hw = [ 100/60, 400/60 ],
            mttf_os = [ 2895, 4000 ],
            mttr_os = [ 1, 5 ],
            mttf_kvm = [ 2990 ],
            mttr_kvm = [ 1, 5 ],
            mttf_nc = [ 788.4, 2000 ],
            mttr_nc = [ 1, 5 ],
            mttf_clc = [ 788.4, 2000 ],
            mttr_clc = [ 1, 5 ],
            mttf_cc = [ 2788.4, 4000],
            mttr_cc = [ 1, 5 ],
            mttf_walrus = [ 2788.4, 4000 ],
            mttr_walrus = [ 1, 5 ],
            mttf_volume = [ 100000 ],
            mttr_volume = [ 1, 5 ]
        ),

        output = (
            type = "swing",
            yLabel = "Steady-state availability",
            baselineValue = av
        )
    );
}
```

Listing 3 Performing sensitivity analysis with the percentage difference technique

```
Sensitivity indexes:
mu_vm, 0.0
lambda_vm, 1.0888123523089612E-6
mttf_volume, 9.99980000378353E-6
mu_in, 2.6380992962909045E-5
mttr_volume, 3.999800010000601E-5
mttf_hw, 4.717463860337643E-5
mttf_kvm, 8.4419926446074 84E-5
mttf_cc, 8.811473386632772E-5
mttf_walrus, 8.811473386632772E-5
mttf_os, 1.9077128745120456E-4
lambda_ap, 7.671518620190045E-4
mttf_nc, 7.674182923738803E-4
mttf_clc, 7.674182923738803E-4
lambda_vlc, 9.729567248072722E-4
mttr_hw, 0.0011403591204155077
mttr_kvm, 0.0013355592654423127
mttr_cc, 0.0014319467315817298
mttr_walrus, 0.0014319467315817298
mttr_os, 0.002756718192627501
mu_ap, 0.005031158832812646
mttr_clc, 0.005041593143433287
mttr_nc, 0.005041593143433287
mu_vlc, 0.011706103871900606
```

Fig. 6 Output of the *percentageDifference* function

The *output* optional parameter is used to produce charts for each parameter. This parameter has three sub-parameters: i) *type*, which defines the type of chart that will be produced; ii) *yLabel*, which defines the label of the y axis in a chart; iii) *baselineValue*, for displaying a horizontal line in the chart for comparing each point with a baseline value. Currently, the only chart type available is *"swing"*, that displays all charts inside a GUI window. In the next release, we plan to include support for generating R and Gnuplot scripts for producing charts. Figure 7 displays the generated charts for the script shown above.

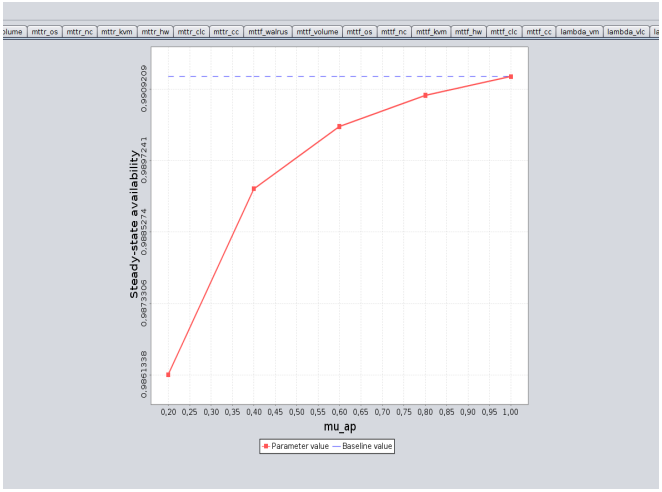


Fig. 7 Charts generated automatically by the *percentageDifference* function

The Design of Experiments (DOE) technique consists in taking a list of parameters and, for each parameter (called factor), a list of values (called levels), and performing a series of experiments with the possible combinations of factor and values. There are various possible designs for an experiment. One possible alternative is to run the experiment for all combination of levels using all factors. This is called a *full factorial design*. A drawback of this alternative is that, even for a small list of parameters, the number of experiments to be performed could be very large. One solution is to use only two levels - this is called a *two-level factorial design*. Even using only two levels, the number of experiments can be very large, if we have many parameters, and grows exponentially for each added parameter. To overcome this issue, we may perform a *fractional factorial design*, that uses only a subset of a factorial design. The Mercury tool allow us to perform two-level, full, and fractional designs, and compute the effects of each factor as described in [6].

Listing 4 shows the usage of the *designOfExperiment* function. It takes the same three mandatory parameters than the *percentageDifference* function: i)

*model_* , *metric_* , and iii) *parameters*. There is no *samplingPoints* parameter, and each parameter receives a list with only two values: a min, and a max value.

```
main{
    designOfExperiment(
        model_ = "NRArchitecture",
        metric_ = "av",
        parameters =  (
            lambda_ap = [ 1/2000, 1/788.4],
            mu_ap = [ 1/5, 1 ],
            lambda_vlc = [ 1/500, 1/336 ],
            mu_vlc = [ 1/5, 1],
            lambda_vm = [ 1/4000, 1/2880 ],
            mu_vm = [ 1, 5 ],
            mu_in = [ 1/0.019166, 1/0.1 ],
            mttf_hw = [ 8760, 10000 ],
            mttr_hw = [ 100/60, 400/60 ],
            mttf_os = [ 2895, 4000 ],
            mttr_os = [ 1, 5 ],
            mttf_kvm = [ 2990 ],
            mttr_kvm = [ 1, 5 ],
            mttf_nc = [ 788.4, 2000 ],
            mttr_nc = [ 1, 5 ],
            mttf_clc = [ 788.4, 2000 ],
            mttr_clc = [ 1, 5 ],
            mttf_cc = [ 2788.4, 4000],
            mttr_cc = [ 1, 5 ],
            mttf_walrus = [ 2788.4, 4000 ],
            mttr_walrus = [ 1, 5 ],
            mttf_volume = [ 100000 ],
            mttr_volume = [ 1, 5 ]
        )
    );
}
```

Listing 4 Performing sensitivity analysis with the design of experiments technique

2.3 Recipe #3 - Reliability block diagram with variable number of blocks

The MSL language provides a construct to define a RBD model with a variable number of blocks into a series or parallel arrangement. It may be useful to answer questions like "*How much the system's availability will be improved if we add one more redundant block?*". In this recipe, we will use the model of an Eucalyptus private cloud with a variable number of worker nodes. Figure 8 shows the top level model. The cloud is composed by a frontend node running the management services of the cloud, and by a set of worker nodes.

The models for the frontend and node were shown on previous section, in figures 2 and 3.

Lines 53 to 59 of Listing 2.3 shows an example of this construct. Using this construct, we can define a series/parallel arrangement with a variable number of similar blocks. After the block id ("*nodes*"), we pass the parameters of the grouping inside parentheses. The "*times*" parameter defines the number of blocks of the series/parallel arrangement. This value must have an initial value. If it uses a variable, the variable value must be previously inside the model (using the *set* keyword), or in the script preamble. For specifying the block that will be repeated, he have two notations. We can set the "*mttf*" and "*mttr*" parameters [2], or we can set the "*hierarchyBlock*" parameter, for hierarchical blocks. Figure 9 shows the output of the script of Listing 2.3.

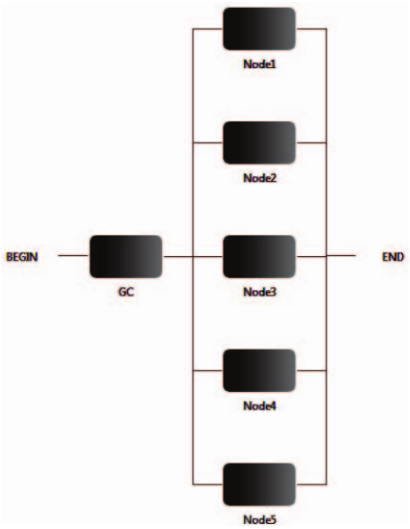


Fig. 8

```
mttf_hw = 8760;
mttr_hw = 100/60;
mttf_os = 2895;
mttr_os = 1;
mttf_kvm = 2990;
mttr_kvm = 1;
mttf_nc = 788.4;
mttr_nc = 1;
mttf_clc = 788.4;
```

[2] Notice that the parameters are in lowercase. This is due the fact that we now are using the dictionary syntax, therefore we can not use the *MTTF* and *MTTR* reserved words as keys

```
mttr_clc = 1;
mttf_cc = 2788.4;
mttr_cc = 1;
mttf_walrus = 2788.4;
mttr_walrus = 1;

mttf_volume = 100000;
mttr_volume = 1;

n_nodes = 1;

RBD NRFrontend{
  block hw( MTTF = mttf_hw, MTTR = mttr_hw );
  block os( MTTF = mttf_os, MTTR = mttr_os );
  block cc( MTTF = mttf_cc, MTTR = mttr_cc );
  block clc( MTTF = mttf_clc, MTTR = mttr_clc );
  block walrus( MTTF = mttf_walrus, MTTR = mttr_walrus );

  series s1(hw, os, clc, cc, walrus);

  top s1;

  metric rel = reliability( time = t );
}

RBD Node{
  block hw( MTTF = mttf_hw, MTTR = mttr_hw );
  block os( MTTF = mttf_os, MTTR = mttr_os );
  block kvm( MTTF = mttf_kvm, MTTR = mttr_kvm );
  block nc( MTTF = mttf_nc, MTTR = mttr_nc );

  series s1(hw, os, kvm, nc);

  top s1;

  metric rel = reliability( time = t );
}

RBD CloudModel{
  hierarchy frontend(
    reliability=solve(NRFrontend, rel)
  );

  parallel nodes(
    times = n_nodes,
    hierarchyBlock = (
      reliability= solve(model = Node, metric = rel)
    )
  );

  series s1( frontend, nodes );

  top s1;

```

```
  metric rel = reliability( time = t );
}

main{
  t = 100;

  for n_nodes in [1, 2, 3, 4, 5, 6, 7, 8] {
    r = solve_rm( CloudModel, rel );
    println( "Number of worker nodes: " .. n_nodes .. ",
    reliability: " .. r );
  }
}
```

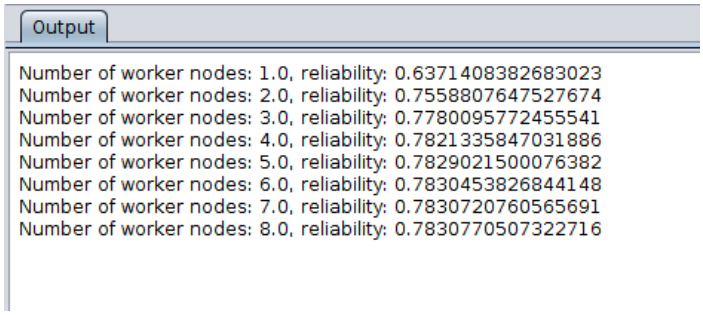


Fig. 9 Output of the script of Listing 2.3

2.4 Recipe #4 - Energy Flow model + availability model

The Energy Flow Model [7] formalism is used to represent the energy flow between the system components considering the respective efficiency and the maximum energy that each component can provide (considering electrical devices) or extract (assuming cooling devices). The system under evaluation can be correctly arranged, in the sense that the required components are properly connected, but they may not be able to meet system demand for electrical energy or thermal load.

Some metrics of an EFM model takes an *availability* parameter, that must inform the steady state availability of the datacenter being evaluated. This value must be computed by a separated availability model of the datacenter. Thanks to the hierarchical modeling capabilities of the scripting language, it is possible to link the EFM model and the availability model. This feature turns possible to see how the availability model parameters impact the EFM metric.

In this recipe, we will use as example the EFM model represented in Figure 10. Figure 11 shows the corresponding availability model. Listing 5 shows a script that implements these two models. In the line 48 of the script we declare a "*operationalExergy*" metric, that takes two parameters: *time* and *availability*. The *availability* parameter is obtained by the RBD model named *AvailModel*, using the *solve* function.

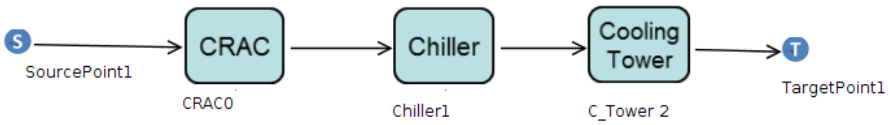


Fig. 10 EFM model

Fig. 11 Availability model of the EFM of Figure 10

```
RBD AvailModel{
    block crac( MTTF = mttf_crac, MTTR = mttr_crac );
    block chiller( MTTF = mttf_chiller, MTTR = mttr_chiller );
    block c_tower( MTTF = mttf_ctower, MTTR = mttr_ctower );

    series s1 ( crac, chiller, c_tower );

    top s1;

    metric aval = availability;
}


EFM EFM1{
    component source(
        type = "Source",
        parameters = (
            demandedEnergy = 10
        )
    );

    component target(
        type = "Target",
        parameters = (
```

```
            demandedEnergy = 10
        )
    );
    component crac(
        type = "CRAC",
        parameters = (
            efficiency = e,
            retailPrice = r
        )
    );
    component chiller(
        type = "Chiller"
    );
    component tower(
        type = "C_Tower"
    );

    arc source -> crac;
    arc crac -> chiller;
    arc chiller -> tower;
    arc tower -> target;


    metric m = operationalExergy( time = 1000, availability =
    solve( AvailModel, aval ) );
}


main {
    mttf_crac = 1000;
    mttr_crac = 1;
    mttf_chiller = 6000;
    mttr_chiller = 10;
    mttf_ctower = 10000;
    mttr_ctower = 20;

    m = solve( EFM1, m );
    println( "Operational exergy: " .. m );
}
```

Listing 5 Script for a EFM model

2.5 Recipe #5 - Using phase type distributions in a Stochastic Petri Net Model

When modelling a certain system of the real world using Stochastic Petri Nets, there are two options for evaluating the metrics: stationary/transient analysis and simulation. Stationary/transient analysis provides more accurate results, but the drawback is that the delay associated with the transi-

tions must be exponentially distributed. If an user collects real world data to parametrize his/her model, and the histogram indicates that the data is not even close to an exponential distribution, the assumption of an exponentially distributed delay makes the model deviates from the real system.

Consider, for instance, that the service time from the SPN from Figure 12 is measured from the real system, and the user obtained the data depicted in the histogram of Figure 13. As we can observe, the histogram curve is different from a exponential distribution. If the user assumes an exponential service time from these data, he/she can obtain different results from the real system.

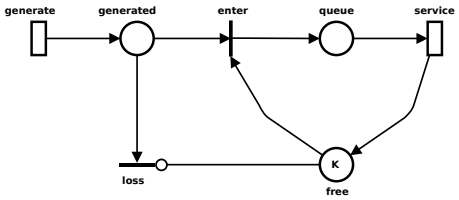


Fig. 12 Stochastic Petri Net model for a M/M/1/k queue [8]

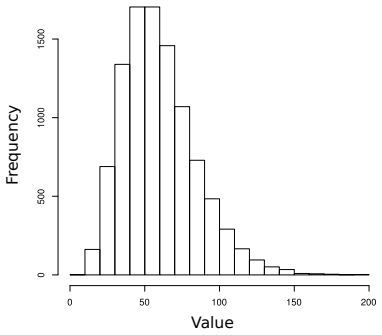


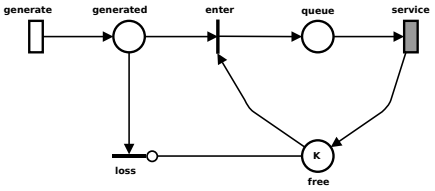
Fig. 13 Histogram plot from collected data

A existent solution to overcome this problem is to use *phase-type* distributions [9]. A phase-type distribution can be expressed as a composition of exponential distributions. A important characteristic of this class of probability distributions is that they can be used to approximate an empirical distribution [10].

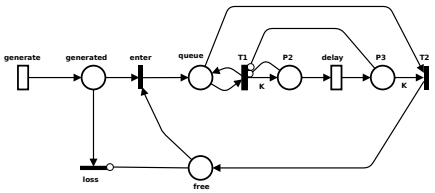
A tradeoff found in using phase-type distributions to approximate the time of firing of transitions is that the model can become more complex and difficult to understand. To simplify the use of this class of distributions in SPN

models, the MSL language defines a special syntax for expressing another distribution types than the exponential distribution. When the evaluation engine for the scripting language detects a phase-type distribution delay, it generates the structure for the phase-type transition as a subnet, and inserts this subnet on the actual Petri net by using the hierarchical transitions of the engine. As consequence, the model will be simpler than if the structure of the phase-type transition was mixed with the Petri net structure. In the graphical representation on the Mercury interface, exponential transitions are displayed with a white background, and non-exponential transitions are displayed with a shadowed background.

Figure 14 a) shows the SPN model with a transition with delay following an Erlang distribution. This transition is represented with a shaded background. Without this feature, our model should be depicted as in Figure 14 b), with the inclusion of additional places, arcs and transitions (displayed inside the dotted box) the shaded transition of the previous model.



(a)



(b)

Fig. 14 Stochastic Petri Net model with a Erlang transition

In the Listing 6, we show the code for the SPN model of Figure 14 a). The *service* transition is configured with a phase-type delay determined by a Erlang distribution. Instead of giving a numeric value for defining a simple timed transition, we have to pass the distribution type and its parameters. In this example, we use the *"Erlang"* string value to specify a Erlang distribution. This distribution have two parameters: *shape* - the number of phases, and *meanDelay* - the exponential delay of one phase.

```
k = 5;
arrivalTime = 1.5;
serviceTime = 0.1;
```

```
phases = 6;

SPN Model{

  place buffer( tokens = k );
  place generated;
  place queue;


  immediateTransition drop(
    inputs = [generated],
    inhibitors = [buffer]
  );

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

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

  timedTransition service(
    inputs = [queue],
    outputs = [buffer],
    delay = (
      type="Erlang",
      parameters = (
        meanDelay=serviceTime,
        shape=phases
      )
    )
  );

  // The expected number of tokens in the "queue" place
  metric equeue = stationaryAnalysis( expression = "E{#queue}" );

}

main {
  e = solve( Model, equeue );
  println( e );
}
```

Listing 6 Timed transition with phase-type delay

2.6 Recipe #6 - Sensitivity analysis on a performability model

In some performability studies, we are interested on finding how much the performance of a system is affected due the presence of failures. Thanks to the powerful *solve* function, we can link a dependability to a performance model, and measure how changing the dependability model parameters impacts the performance measures. Figure 15 shows an example of performability model. The performance model is a SPN model that represents a M/M/1/K queue. We put a place of *server_up*/*server_down* places that indicates the operational status of the server. If the server is down, the server is not able to process a request, as indicated by the inhibitor arc. The delay of the *fail*/*repair* transitions is computed by a separated RBD model. This RBD model is composed by three blocks: hardware, operating system and application. The MTTF and MTTR metrics of this model are used as delay for the *fail*/*repair* transitions on the SPN model.

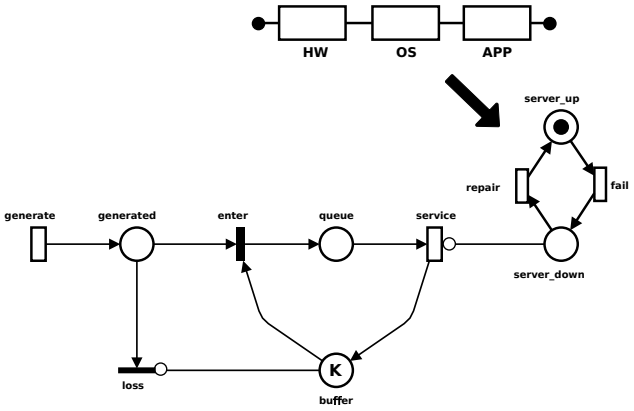


Fig. 15 Performability model

The script is exhibited on Listing 7. We create a RBD model named *DependModel*, that declares two metrics: the MTTF and the MTTR. The value of those metrics are used as input for the performance model, as shown in lines 62 and 66. On the SPN model, we declares two metrics: *m1*, that computes the stationary probability of all markings that turns the *service* transition enabled; *tp*, a composite metric that computes the throughput of the *service* transition. Finally, on the main block, we variate the *mttf_hw* parameter over a list, and show how the throughput of the server is affected by this variation.

```
arrivalTime = 1.5;
serviceTime = 1.2;
```

```
k= 5;

mttf_hw = 1000;
mttr_hw = 4;
mttf_os = 700;
mttr_os = 1;
mttf_app = 500;
mttr_app = 0.1;

RBD DependModel{
  block HW( MTTF = mttf_hw, MTTR = mttr_hw );
  block OS( MTTF = mttf_os, MTTR = mttr_os );
  block APP( MTTF = mttf_app, MTTR = mttr_app );

  series s1( HW, OS, APP );

  top s1;

  metric mttf_ = mttf;
  metric mttr_ = mttr;
}

SPN Model{

  place buffer( tokens = k );
  place generated;
  place queue;

  place server_up( tokens = 1 );
  place server_down;


  immediateTransition drop(
    inputs = [generated],
    inhibitors = [buffer]
  );

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

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

  timedTransition service(
    inhibitors = [server_down],
    inputs = [queue],
    outputs = [buffer],
    delay = serviceTime
  );

```

```
  timedTransition fail(
    inputs = [server_up],
    outputs = [server_down],
    delay = solve( DependModel, mttf_ )
  );

  timedTransition repair(
    inputs = [server_down],
    outputs = [server_up],
    delay = solve( DependModel, mttr_ )
  );

  metric m1 = stationaryAnalysis( expression = "P{(#queue>0)AND(#
    server_up=1)}" );
  metric tp( m1 / serviceTime );

}

main {
  for mttf_hw in [1000, 1200, 1300, 1400, 1500] {
    tp = solve( Model, tp );
    println( tp );
  }
}
```

Listing 7 Script for a performability model

2.7 Recipe #7 - Executing a script inside a Java program

Suppose that a programmer wants to create a specific tool that will be used to model cloud infrastructures. With this tool, an user will be able to create clusters, specify frontend and worker nodes, and compose them into a cloud infrastructure. By using the provided high level cloud model, the user will be able to extract performance and dependability metrics of a cloud. This can be achieved by converting the high level model into a SPN/RBD/CTMC model, and solving its metrics. To achieve this goal, the programmer must be able to create SPN/RBD/CTMC models, and solving it inside his/her program.

The Mercury tool exports the same API used by the scripting evaluation runtime. By using the Mercury executable archive (.jar) as a library inside an external Java program, it is possible to:

- Run scripts, i.e., parsing the script, loading the models in the runtime, and running the main block;
- Evaluate scripts, i.e., parsing the script, loading the models in the runtime, but not running the main block;

- Obtaining reference to models and metrics;
- Modifying parameters;
- Solving metrics.

The scripts can be created "on the fly" and stored into a *String* variable, or they can be stored in the disk. The Listing 8 shows a Java program that performs the above mentioned steps. To compile this class, we must to create a Java project using any IDE (Eclipse, Netbeans), and add the Mercury as a library in the dependencies. We supply a single .jar file that contains the Mercury API and also the Mercury dependencies into a single package, as shown in Figure 16.

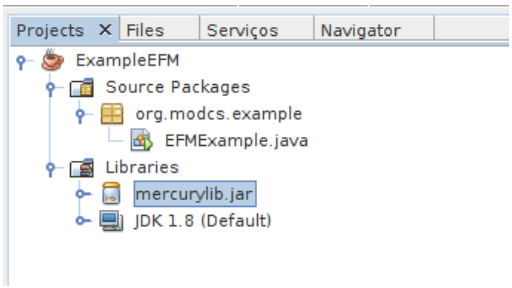


Fig. 16 Netbeans project with the Mercury API as library

In the Listing 8, we evaluate a script located in the file named *"efm_ model.mry"*, that must be in the same path of the program, or inside the root of the Netbeans/Eclipse project folder. The Script class has two constructors:

- public Script(java.io.File file);
- public Script(String script);

The first constructor is used to evaluate a script located inside a file, and the second is used to evaluate a script that is stored into a java.lang.String object. For evaluating this script, we call the *"evaluateScript"* method of the runtime object. This method sets the variables defined in the preamble (if this section exists), loads all models in the runtime, but does not execute the main block. If we want to run the main block after loading the models, we call the *"runScript"* method.

```
1 package org.modcs.example;
2
3 import java.io.File;
4 import org.modcs.tools.parser.model.ExecutionRuntime;
5 import org.modcs.tools.parser.model.Model;
6 import org.modcs.tools.parser.model.Script;
7 import org.modcs.tools.parser.model.metrics.Metric;
8
9
```

```
public class EFMExample {
    public static void main(String[] args) {

        //creating the scripting evaluatin runtime
        ExecutionRuntime runtime = new ExecutionRuntime();

        //creating an evaluating the script, without
        //running the main block
        Script scrpt = new Script( new File("emf_model.mry"));
        runtime.evaluateScript(scrpt);


        //modifying some variables and changing the model
        //parameters
        runtime.getVariableTable().setValue("mttf", 1000);
        runtime.getVariableTable().setValue("mttr", 1);
        runtime.getVariableTable().setValue("e", 0.8);
        runtime.getVariableTable().setValue("r", 5000);

        //obtaining reference to a model by passing
        //its identifier
        Model model = runtime.getModel("EFM1");

        //obtaining reference to a metric
        Metric m = model.getMetric("m2");

        //solving and printing the metric
        double val = m.solve();
        System.out.println("Metric value: " + val);


        //performing a experiment
        double[] mttfs = { 500, 1000, 1500, 2000, 2500 };

        for(double mttf: mttfs){
            runtime.getVariableTable().setValue("mttf", mttf);

            System.out.println(m.solve());
        }
    }
}
```

Listing 8 Executing a script inside a Java program

References

1. B. Silva, R. Matos, G. Callou, J. Figueiredo, D. Oliveira, J. Ferreira, J. Dantas, A. L. Junior, V. Alves, and P. Maciel, "Mercury: An integrated environment for performance and dependability evaluation of general systems," *Proceedings of Industrial Track at 45th Dependable Systems and Networks Conference (DSN)*, 2015.

2. J. Dantas, R. Matos, J. Araujo, and P. Maciel, "Eucalyptus-based private clouds: availability modeling and comparison to the cost of a public cloud," *Computing*, pp. 1–20, 2015.
3. R. Matos, J. Araujo, D. Oliveira, P. Maciel, and K. Trivedi, "Sensitivity analysis of a hierarchical model of mobile cloud computing," *Simulation Modelling Practice and Theory*, vol. 50, no. 0, pp. 151 – 164, 2015, special Issue on Resource Management in Mobile Clouds.
4. R. M. De Melo, M. C. Bezerra, J. Dantas, R. Matos, I. J. De Melo Filho, and P. Maciel, "Redundant vod streaming service in a private cloud: Availability modeling and sensitivity analysis," *Mathematical Problems in Engineering*, vol. 2014, 2014.
5. F. Hoffman and R. Gardner, "Evaluation of uncertainties in environmental radiological assessment models," in *Radiological Assessments: a Textbook on Environmental Dose Assessment*, J. Till and H. Meyer, Eds. Washington, DC: U.S. Nuclear Regulatory Commission, 1983, Report No. NUREG/CR-3332.
6. R. Jain, *The art of computer systems performance analysis.* John Wiley & Sons, 2008.
7. G. Callou, P. Maciel, D. Tutsch, J. Ferreira, J. Araújo, and R. Souza, "Estimating sustainability impact of high dependable data centers: A comparative study between brazilian and us energy mixes," *Computing*, vol. 95, no. 12, pp. 1137–1170, 2013.
8. 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.
9. L. Breuer and D. Baum, "Phase-type distributions," *An Introduction to Queueing Theory and Matrix-Analytic Methods*, pp. 169–184, 2005.
10. H.-B. Mor, "Performance modeling and design of computer systems," 2013.