User Manual
(SAFEST Tool)

Table of Contents

System failure models

System failure models are in fact failure scenarios that may arise during the life cycle of a system. They are modeled as fault trees and their probability of occurrence is then computed using efficient algorithms which are based on the theory of probabilistic model-checking and BDD techniques.

A mathematical logic is used to define several dependability metrics of failure scenarios. For this, we employ continuous stochastic logic (CSL). The important metrics are e.g. (conditional) dependability, mean-time-to-failure, average failure probability per hour, etc.

We can utilize parameters to specify the failure distributions of the basic elements of fault trees, creating parametric fault trees. Changing the values of basic elements allows us to produce several fault tree versions and their metrics results can then be compared .

System modeling with SAFEST

In the SAFEST tool, failure scenarios of a system can be grouped in a project.

A SAFEST project comprises of multiple:

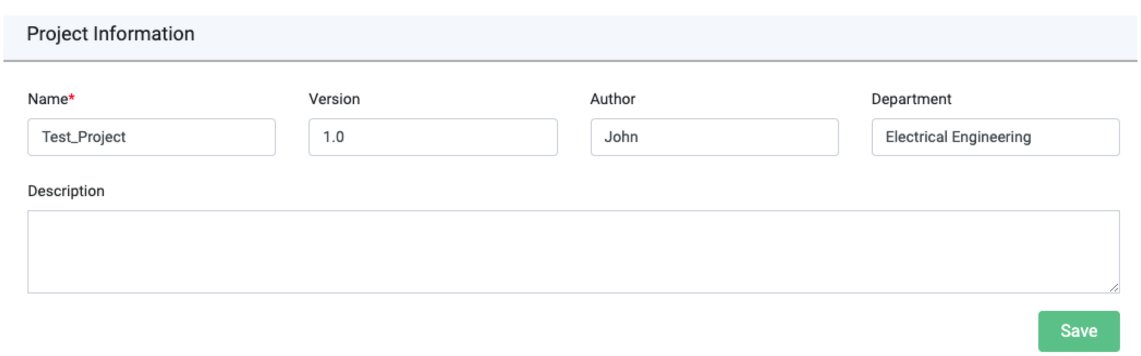
1. Failure models (scenarios). Each failure model comprises of:
    a. A hierarchical fault tree – static or dynamic

    b. Quantifiable states – in the form of boolean equations on elements of the fault tree.
2. Parameter Sets – to generate several variants of fault trees. Each parameter set comprises of:
    a. Constants
    b. Real-valued expressions
    c. Probability distributions (Exponential, Erlang, Weibull, Log-normal)
    d. Empirical probability distributions – failure distributions generated from data sets.
3. Metrics. There is a list of predefined metrics (classified into basic, complex and Importance metrics). For advanced users, it is possible to define custom metrics using continuous stochastic logic (CSL).

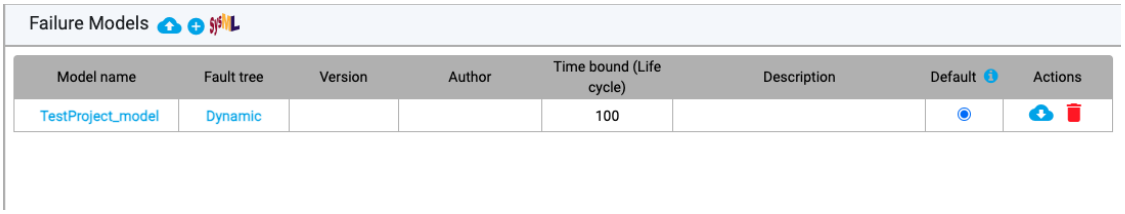
Steps to build system failure models:

1. Click on “New Project” in the File menu. The following window will appear.

2. Fill up mandatory fields, and click the “Create” button. The following page will appear, where users can make any changes if required.



3. Click on the “Failure Models” in the left panel to display all failure models in a table. For a new project, one failure model is created automatically as a dynamic fault tree.

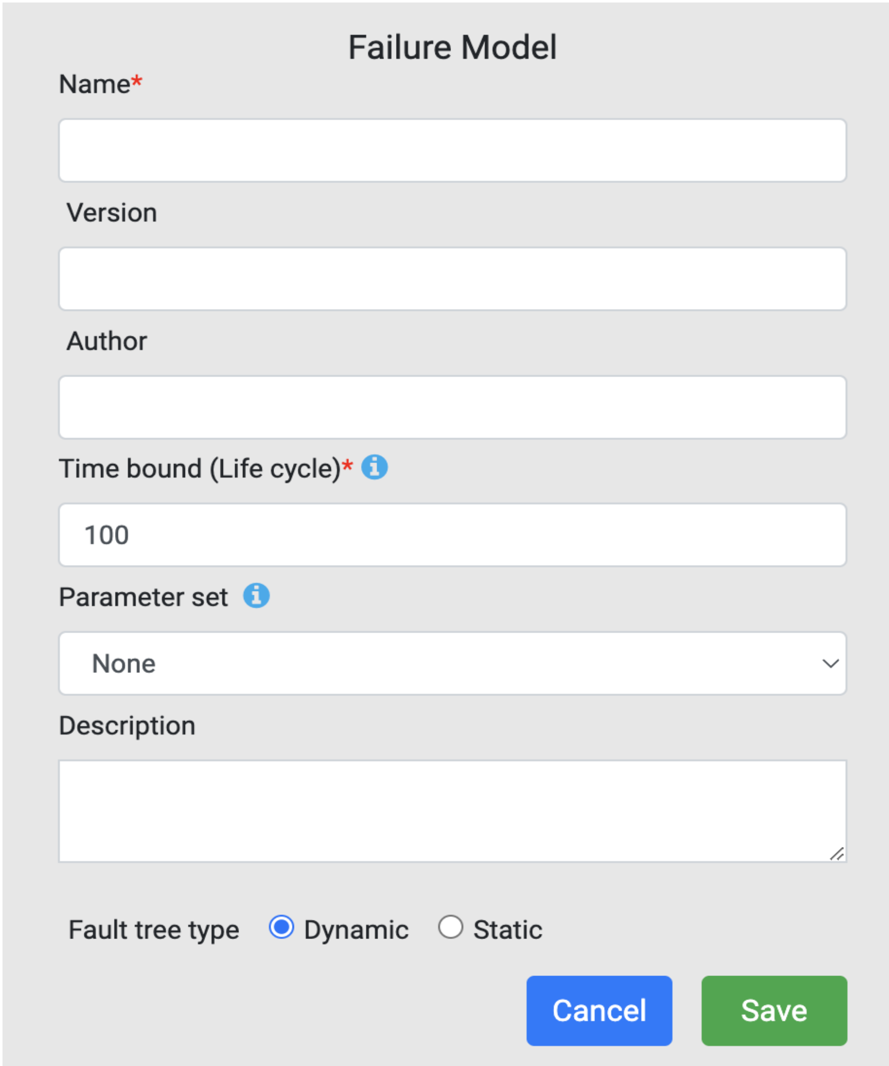


Failure Models

Model name	Fault tree	Version	Author	Time bound (Life cycle)	Description	Default	Actions
TestProject_model	Dynamic			100			

4. A failure model which is worked upon the most can be selected as a default model by selecting the corresponding radio button.
5. Failure models can be uploaded by clicking on the  near the “Failure Models” title. For downloading individual models, click the download icon in the respective row.

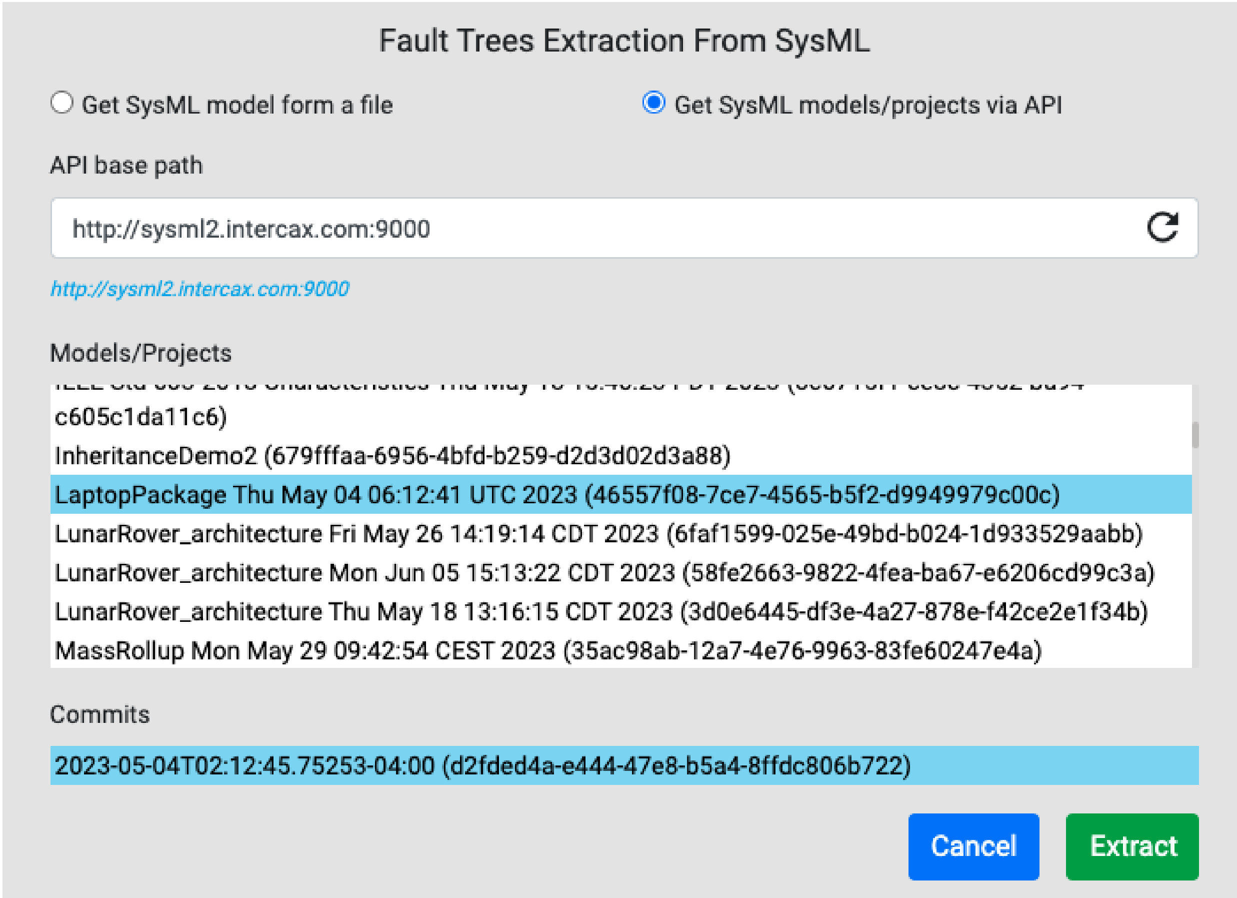
6. Click on the plus icons  near the “Failure Models” title to add a new failure model.



- A time bound for which model is to be analyzed may be inserted. This value can be changed at the time of analysis as well.
- A parameter set in which parameters to be used in the model are defined may be selected. It can also be changed at the time of analysis.
- Type of fault tree can be selected as “Static” or “Dynamic”.
- Click on the “Save” button to save a failure model.

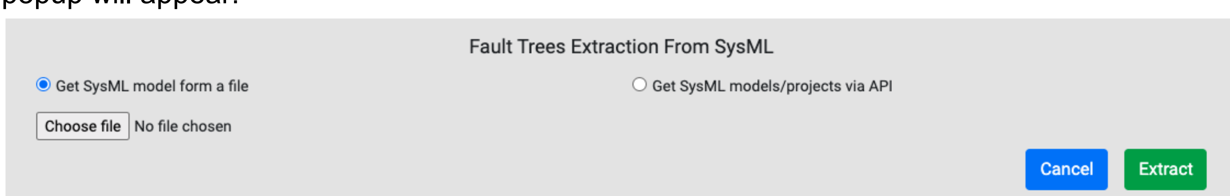
7. Users have the option to extract failure models from SysML 2.0 models. The SysML model has to be annotated with safety information in order to generate DFTs

automatically out of it. Click on the icon, the following popup will appear:



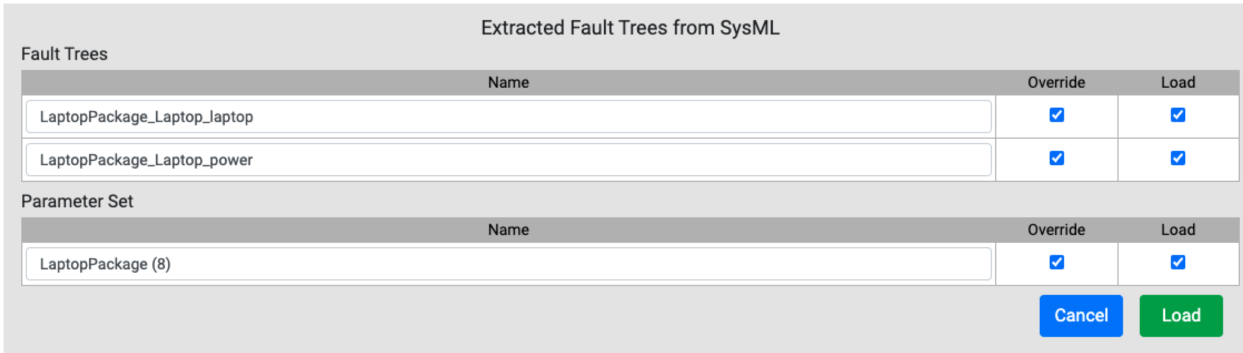
User has two options to extract DFTs out of SysML models

- **Get SysML model from a file:** Compile a SysML model, which is annotated with safety information – please read **Annotate SysML Models with Safety Information section** below for further details on how to annotate SysML models with safety information in order to prepare them for automatic extraction of DFTs out of them, in a jupyter notebook. And run the following command to export the model in JSON format. **Currently we support the latest version – v0.33.0 – of SysML 2.0:**
  %export <package_name>
  After downloading, it can be uploaded inside Failure Models inside the project. To do this click on the “Get SysML model from a file” radio button. The following popup will appear:

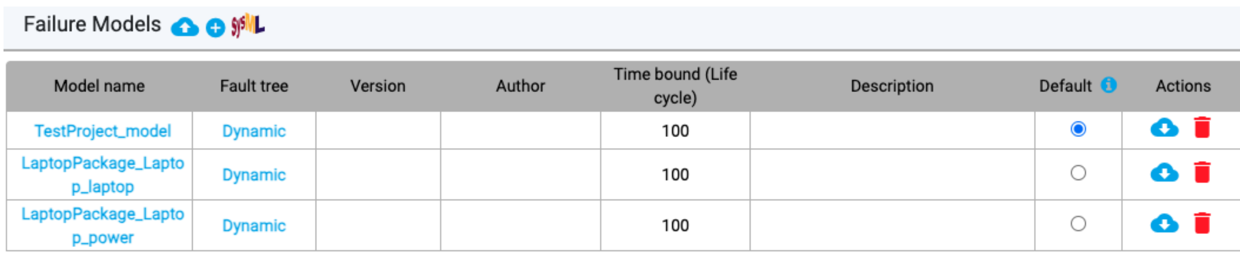
  

  Select the JSON file that you downloaded, and click the create button.
  The following popup will appear that contains fault trees of all failure scenarios

that have been mentioned in the SysML model. It also contains parameter that have been given in the SysML mode as e.g. failure rates of different components inside the SysML.

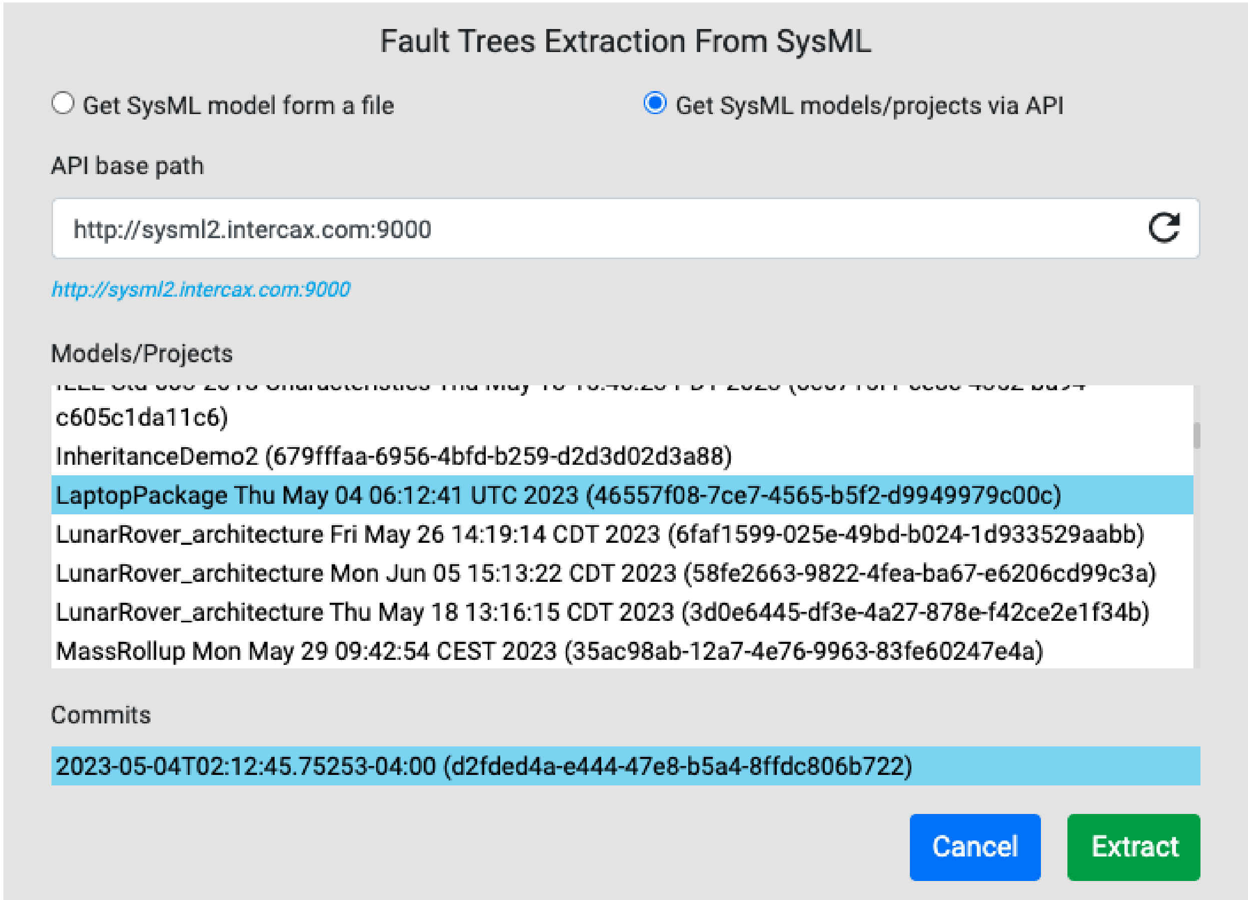


The user can select the failure models as well as parameters to be included inside the project as:



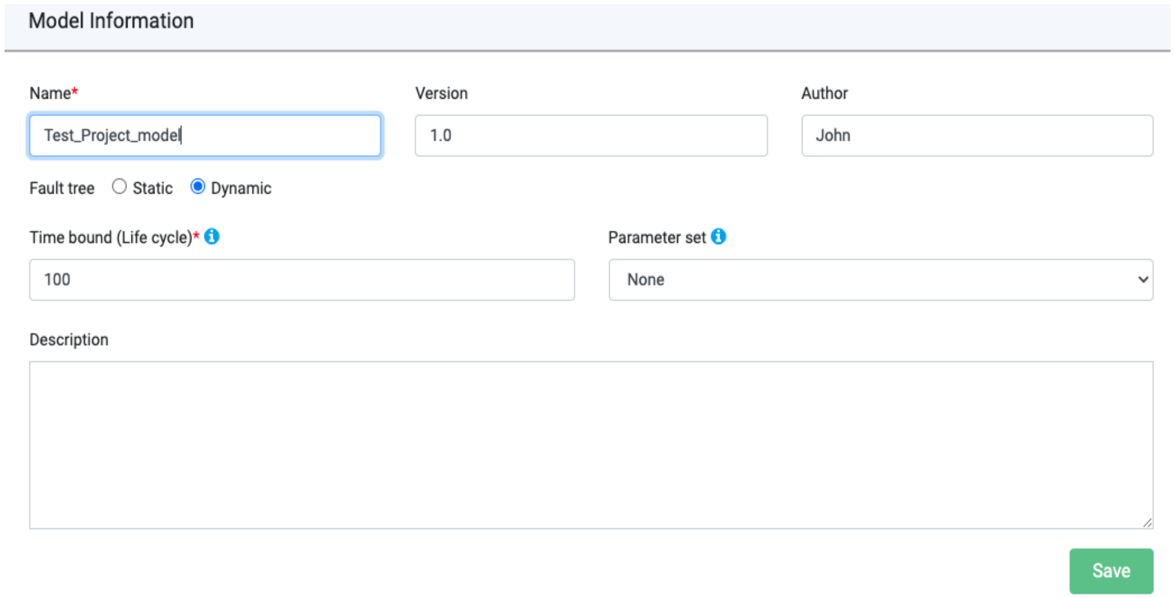
- **<u>Get SysML models/projects via API:</u>** SysML projects are normally uploaded at some central repository. One has the option to connect to that repository using its API and upload the model from there. The following popup appears when “Get

SysML models/projects via API” is selected:



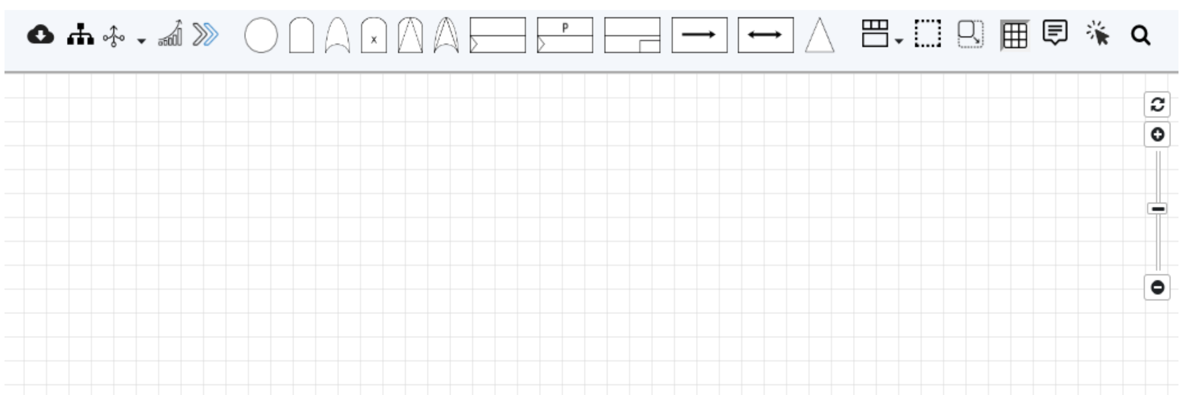
By default projects are extracted from the repository with API “http://sysml2.intercax.com:9000”.
Select a project and its commit, and then click the create button to upload the SysML model. Rest of the steps are the same as above. For testing purposes we have uploaded our “LaptopPackage” project in the above repository.

8. The algorithm will automatically generate fault trees of these scenarios and upload them to the project “Failure Models”. Please read **Annotate SysML Models with Safety Information section** below for further details.
9. Click on the failure model name in the left panel to see details about the model.

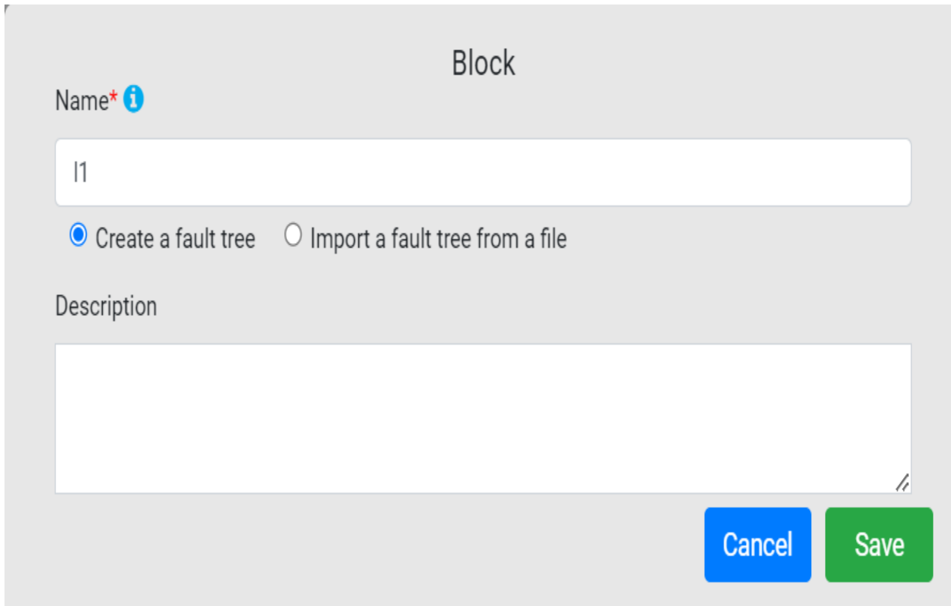


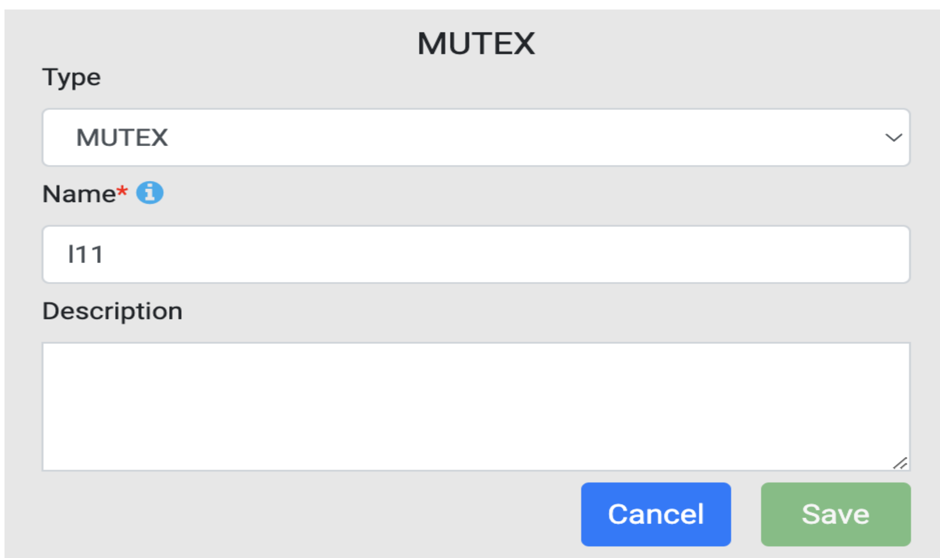
- Failure model fields can be changed on this screen and saved.

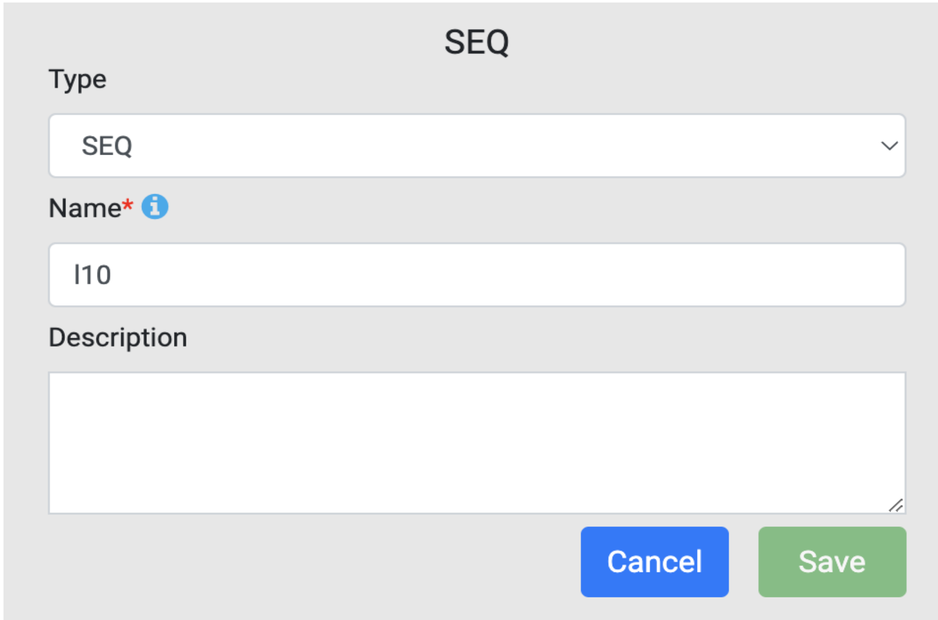
10. Click on the “Fault Tree” of the failure model to open a drag-and-drop grid.

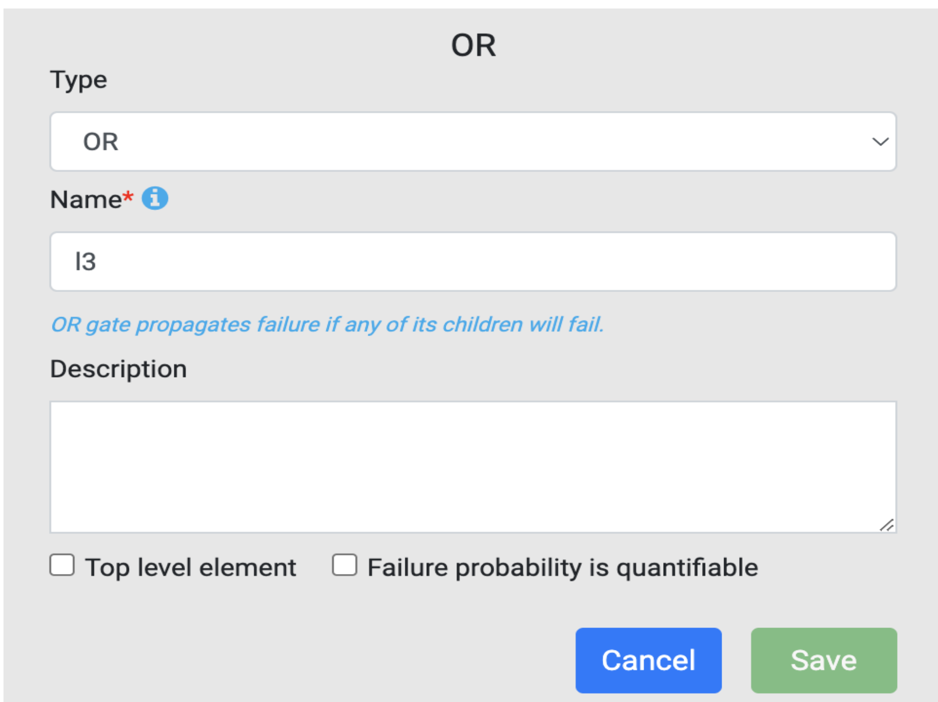


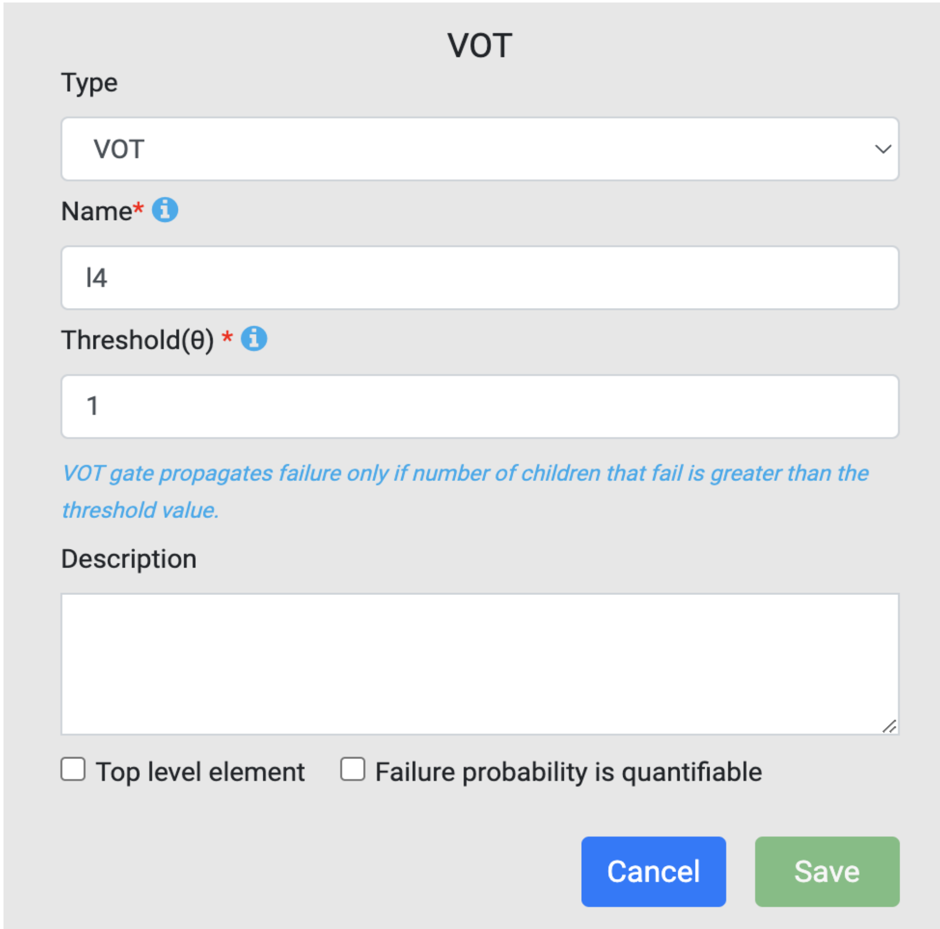
- Draw your tree by dragging different elements from the toolbar.
- Click on an element to update its information in the corresponding popup window

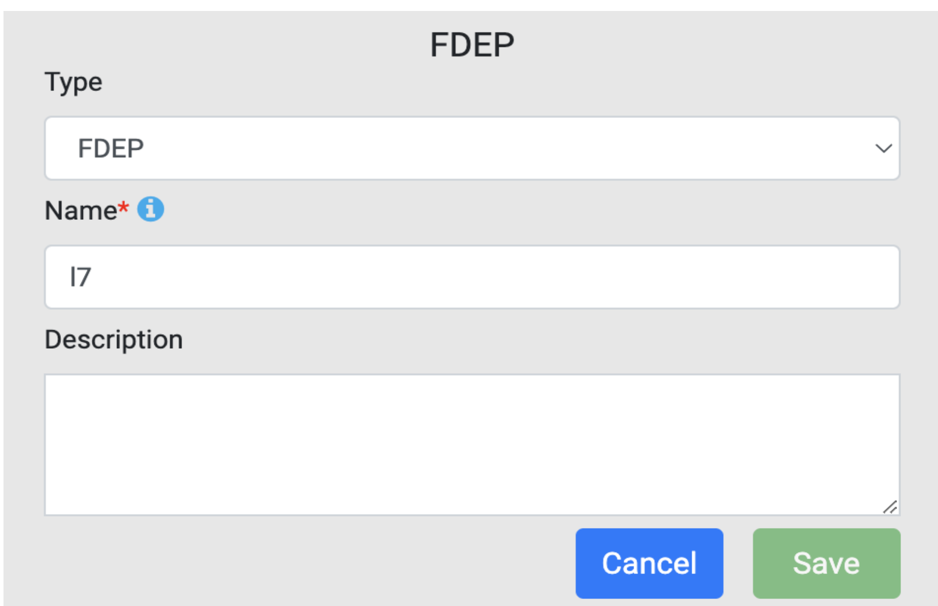

Block
Name*
I1
Create a fault tree
Import a fault tree from a file
Description
Cancel
Save

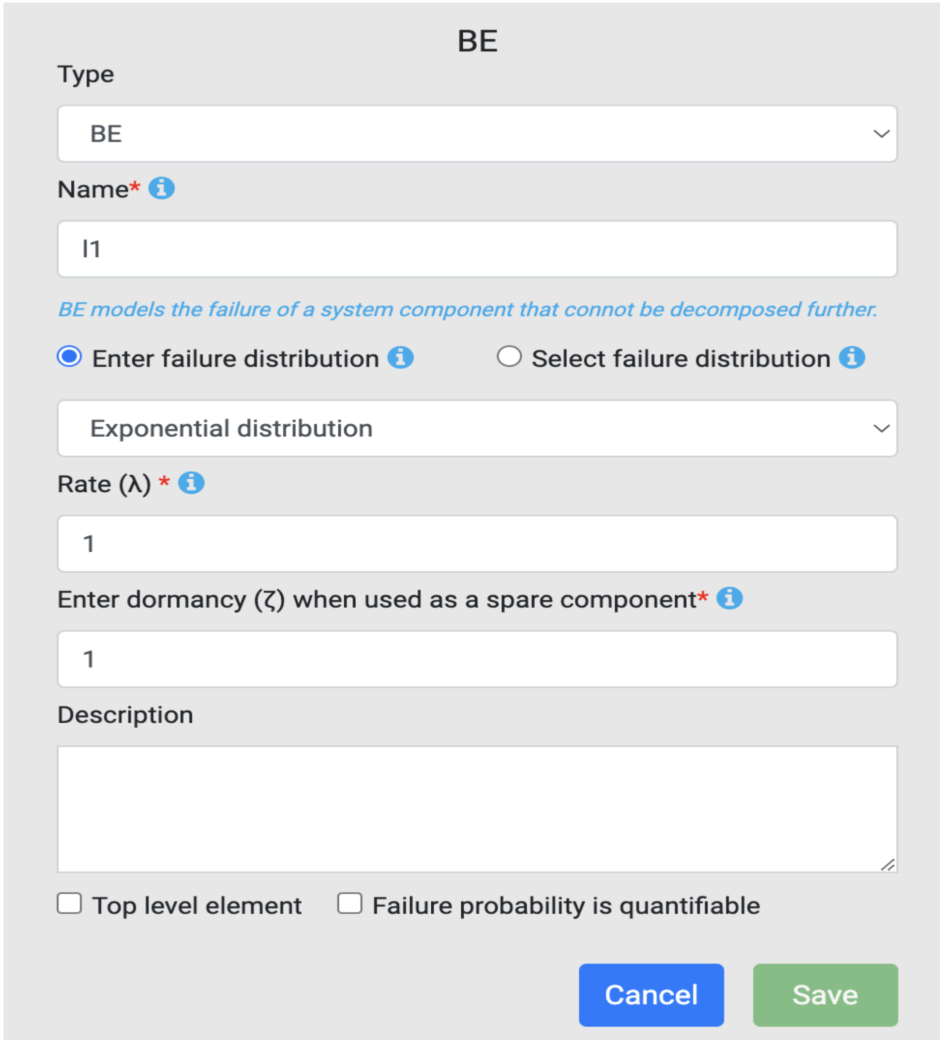

MUTEX
Type
MUTEX
Name*
I11
Description
Cancel
Save


SEQ
Type
SEQ
Name*
I10
Description
Cancel
Save

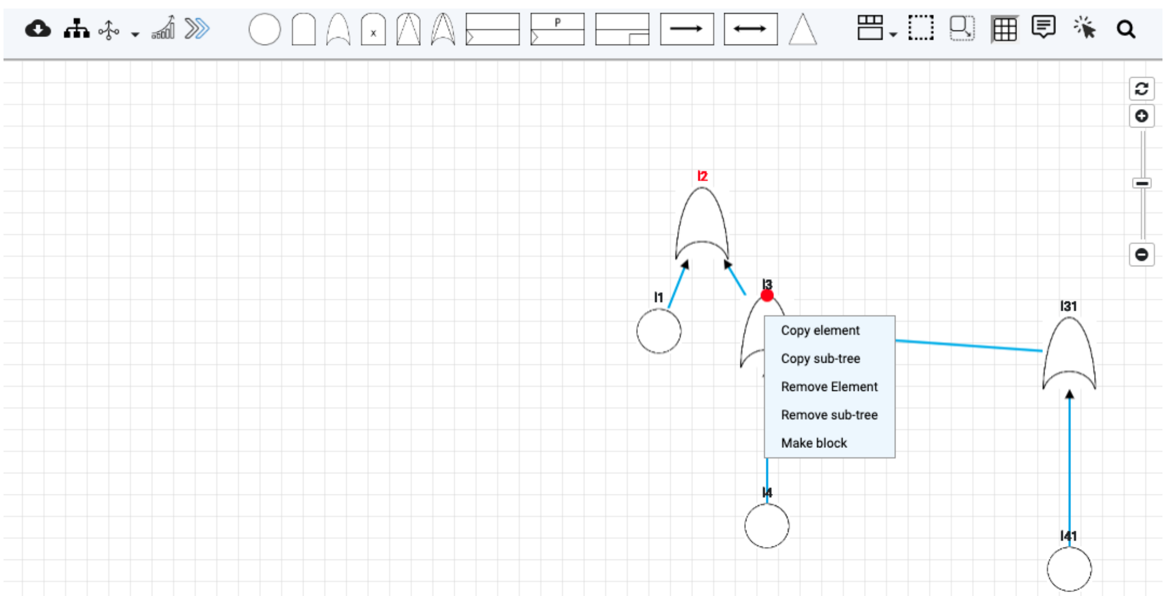

OR
Type
OR
Name*
I3
OR gate propagates failure if any of its children will fail.
Description
Top level element
Failure probability is quantifiable
Cancel
Save


VOT
Type
VOT
Name*
I4
Threshold(θ) *
1
VOT gate propagates failure only if number of children that fail is greater than the threshold value.
Description
Top level element
Failure probability is quantifiable
Cancel
Save

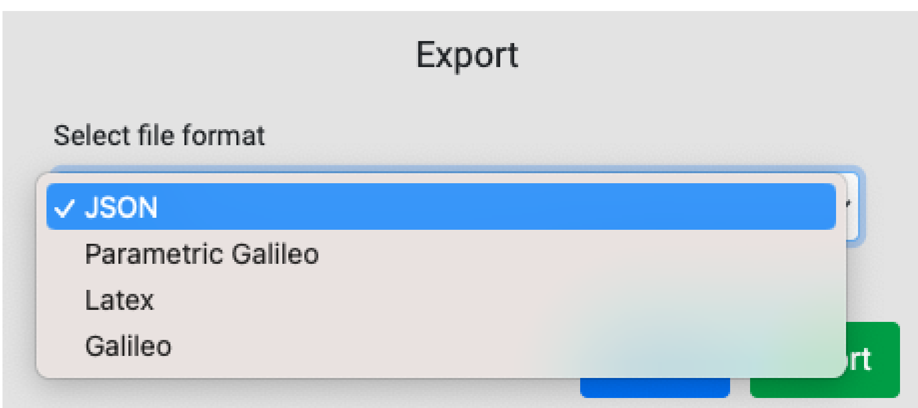

FDEP
Type
FDEP
Name*
I7
Description
Cancel
Save


BE
Type
BE
Name*
I1
BE models the failure of a system component that connot be decomposed further.
Enter failure distribution
Select failure distribution
Exponential distribution
Rate (λ) *
1
Enter dormancy (ζ) when used as a spare component*
1
Description
Top level element
Failure probability is quantifiable
Cancel
Save

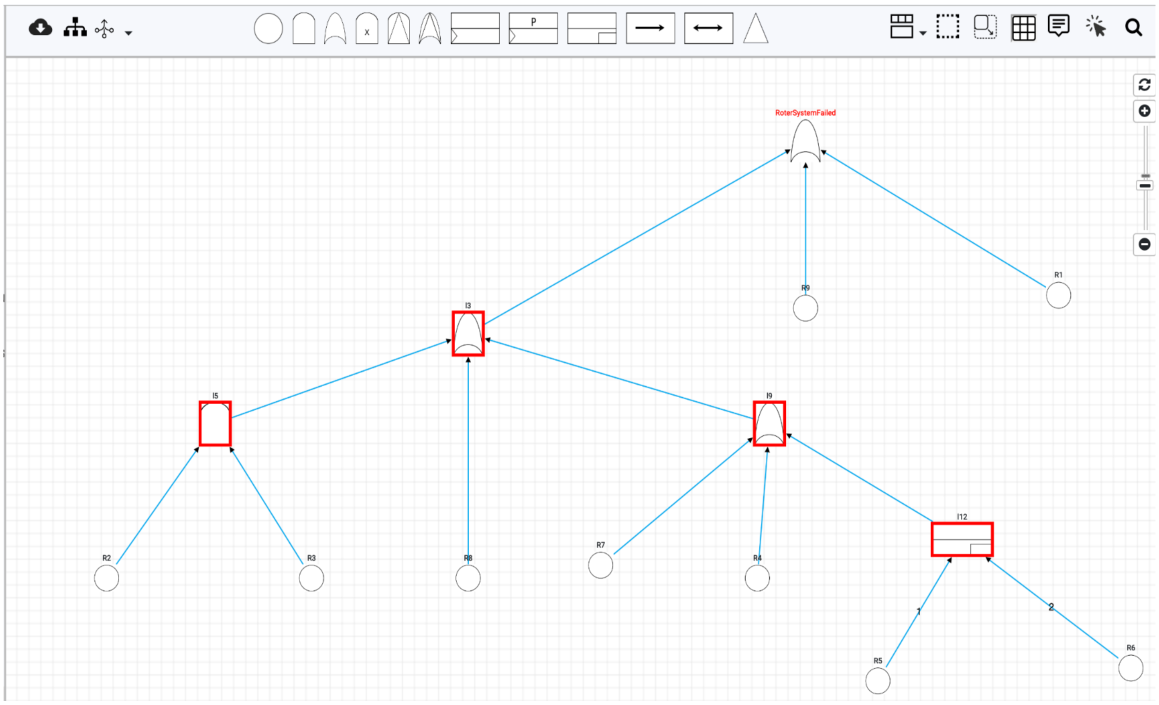
- Only one element can be selected as a top level element in a tree.
- “Failure probability is quantifiable” checkbox is only visible in advance view.
- Elements can be connected with each other by drawing edges between them.
- Those elements which propagate failure to their parents can be marked (by selecting their “Failure probability is quantifiable” checkbox) to create sub-scenarios. This feature is only available in “Advance View”.
- On right clicking on an element, a popup comes up that allows to copy the element, copy the sub-tree under it, delete the element, delete the sub-tree under it or convert the sub-tree under it into a block (if possible)



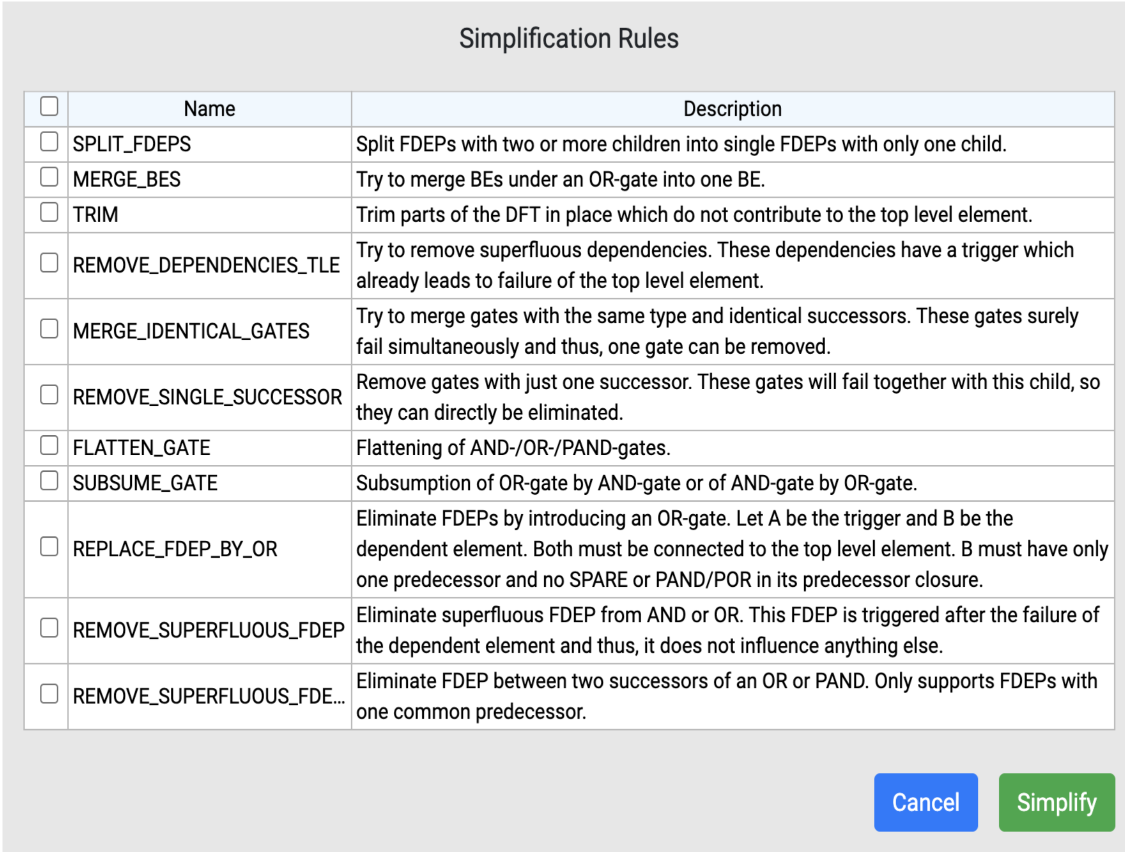
11. Cick on the download icon , a popup comes up to download the tree in JSON, Galileo, Parametric Galelio and Latex formats.



12. Click on the icon  (not visible when sub-trees are in focus) to highlight the elements that, along with their children, can be converted into modules (independent sub-trees). In order to convert an element and its children into a module, right click on it and then click “Make block”.



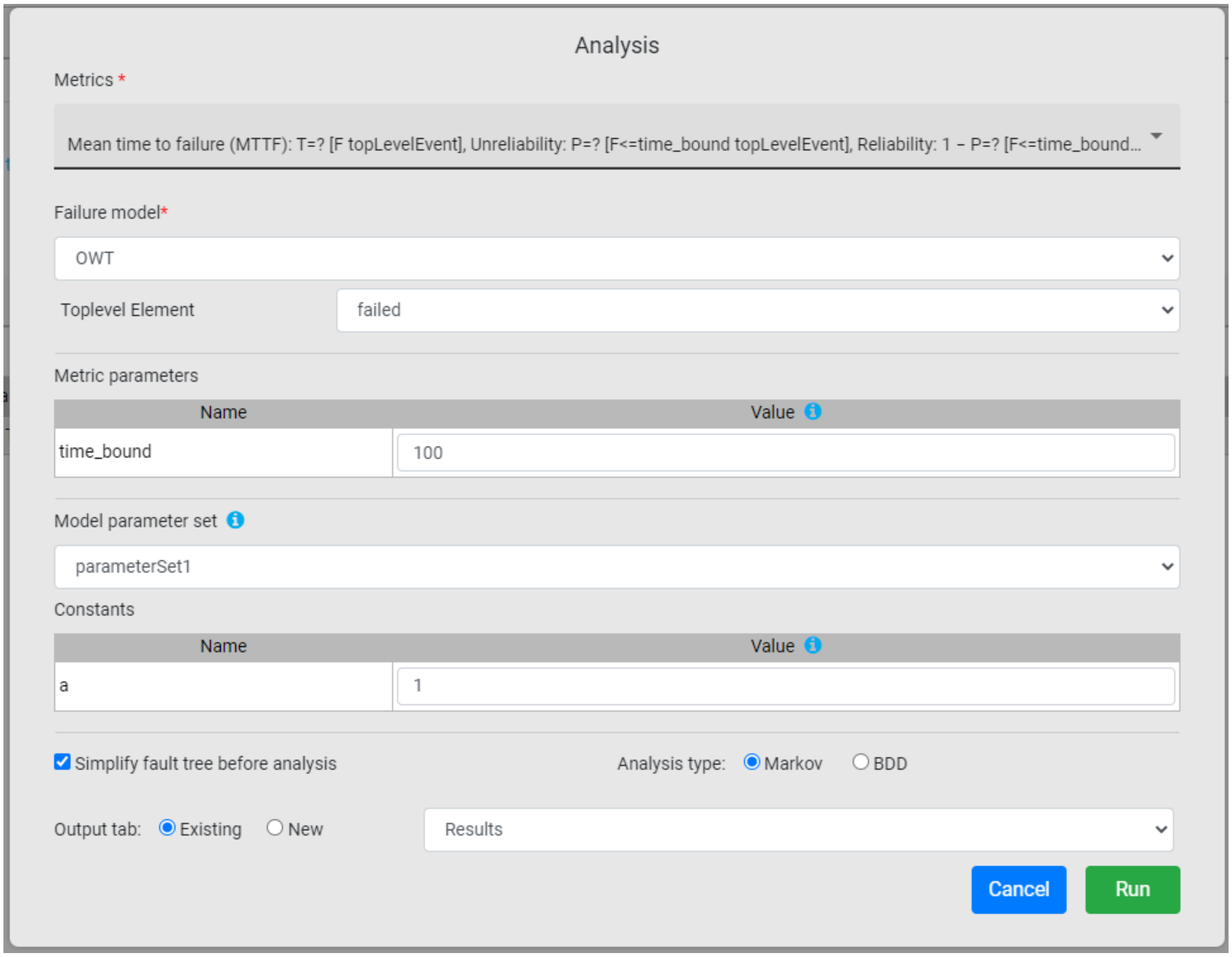
13. In order to simplify a fault tree click on down arrow along the icon . It will give three options for tree simplification:
    a. Simplify by all rules. It will apply all rules recursively for simplification.
    b. Default rules. It will apply a selected set of rules (most commonly used) for simplification. They are:
        i. SPLIT_FDEPS – Split FDEPs with two or more children into single FDEPs with only one child.
        ii. MERGE_BES – Try to merge BEs under an OR-gate into one BE.
        iii. TRIM – Trim parts of the DFT in place which do not contribute to the top level element.
        iv. REMOVE_SINGLE_SUCCESSOR – Remove gates with just one successor. These gates will fail together with this child, so they can directly be eliminated.
        v. FLATTEN_GATE – Flattening of AND-/OR-/PAND-gates.
    c. Custom rules. It allows users to select rules that are to be applied for simplification.



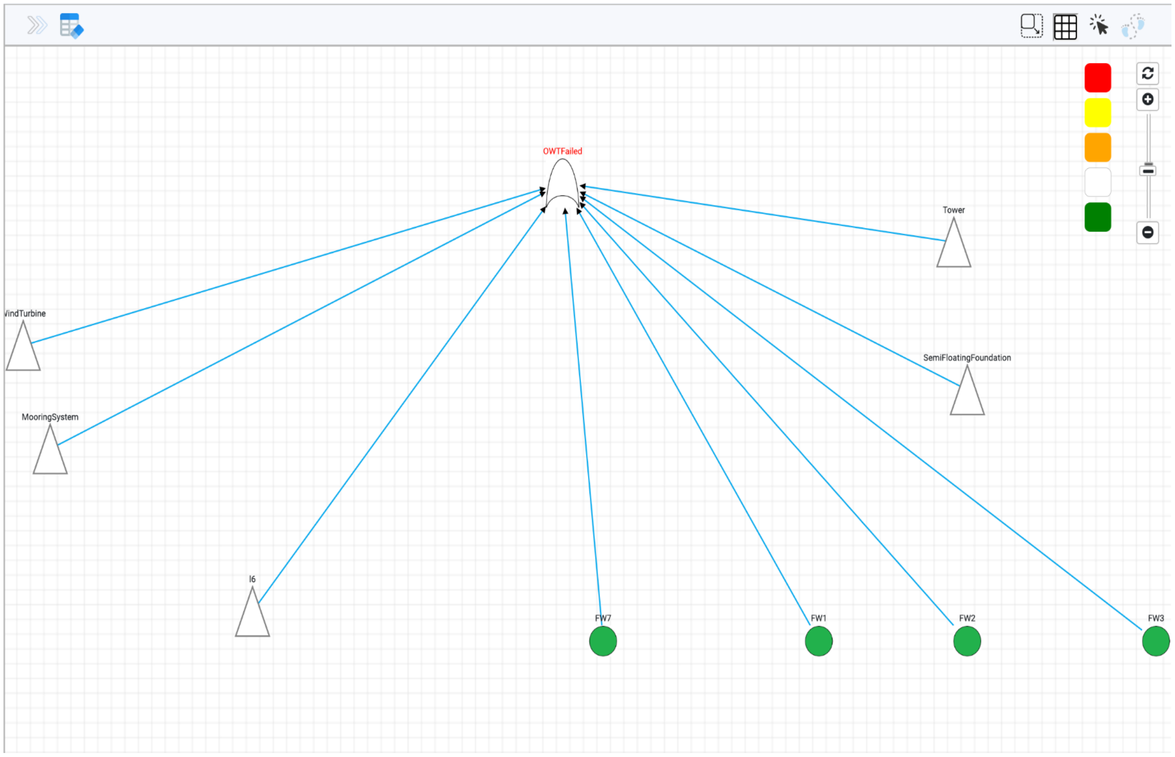
Simplification Rules

☐	Name	Description
☐	SPLIT_FDEPS	Split FDEPs with two or more children into single FDEPs with only one child.
☐	MERGE_BES	Try to merge BEs under an OR-gate into one BE.
☐	TRIM	Trim parts of the DFT in place which do not contribute to the top level element.
☐	REMOVE_DEPENDENCIES_TLE	Try to remove superfluous dependencies. These dependencies have a trigger which already leads to failure of the top level element.
☐	MERGE_IDENTICAL_GATES	Try to merge gates with the same type and identical successors. These gates surely fail simultaneously and thus, one gate can be removed.
☐	REMOVE_SINGLE_SUCCESSOR	Remove gates with just one successor. These gates will fail together with this child, so they can directly be eliminated.
☐	FLATTEN_GATE	Flattening of AND-/OR-/PAND-gates.
☐	SUBSUME_GATE	Subsumption of OR-gate by AND-gate or of AND-gate by OR-gate.
☐	REPLACE_FDEP_BY_OR	Eliminate FDEPs by introducing an OR-gate. Let A be the trigger and B be the dependent element. Both must be connected to the top level element. B must have only one predecessor and no SPARE or PAND/POR in its predecessor closure.
☐	REMOVE_SUPERFLUOUS_FDEP	Eliminate superfluous FDEP from AND or OR. This FDEP is triggered after the failure of the dependent element and thus, it does not influence anything else.
☐	REMOVE_SUPERFLUOUS_FDE...	Eliminate FDEP between two successors of an OR or PAND. Only supports FDEPs with one common predecessor.

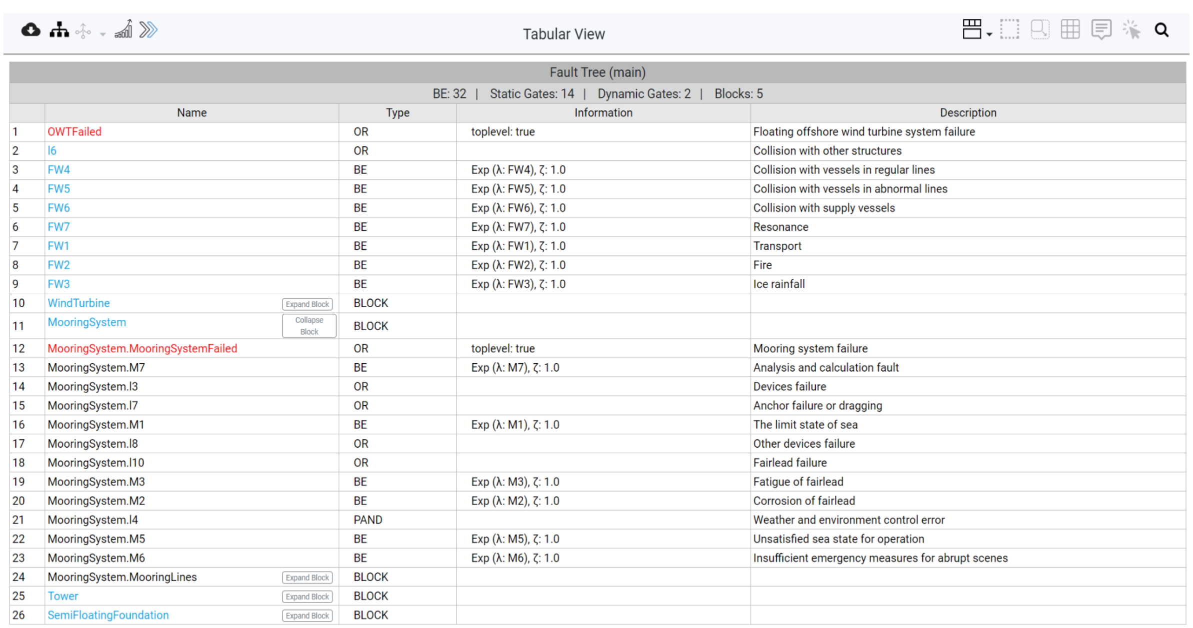
14. Click on the icon  (not visible when sub-trees are in focus) to do basic analysis that includes reliability, mean-time-to-failure and average-failure-probability per hour. It will take the user to the “Analysis” window under “Computing” in the left panel.



15. Click on the icon  to start interactive simulation. It will take the user to the "Interactive Simulation" window under "Computing" in the left panel.



16. Click on the down arrow new the icon , it will give three options to display a tree:
    a. Canvas view. It shows the tree in the grid.
    b. Tabular view. It will show the tree in a tabular form



Fault Tree (main)

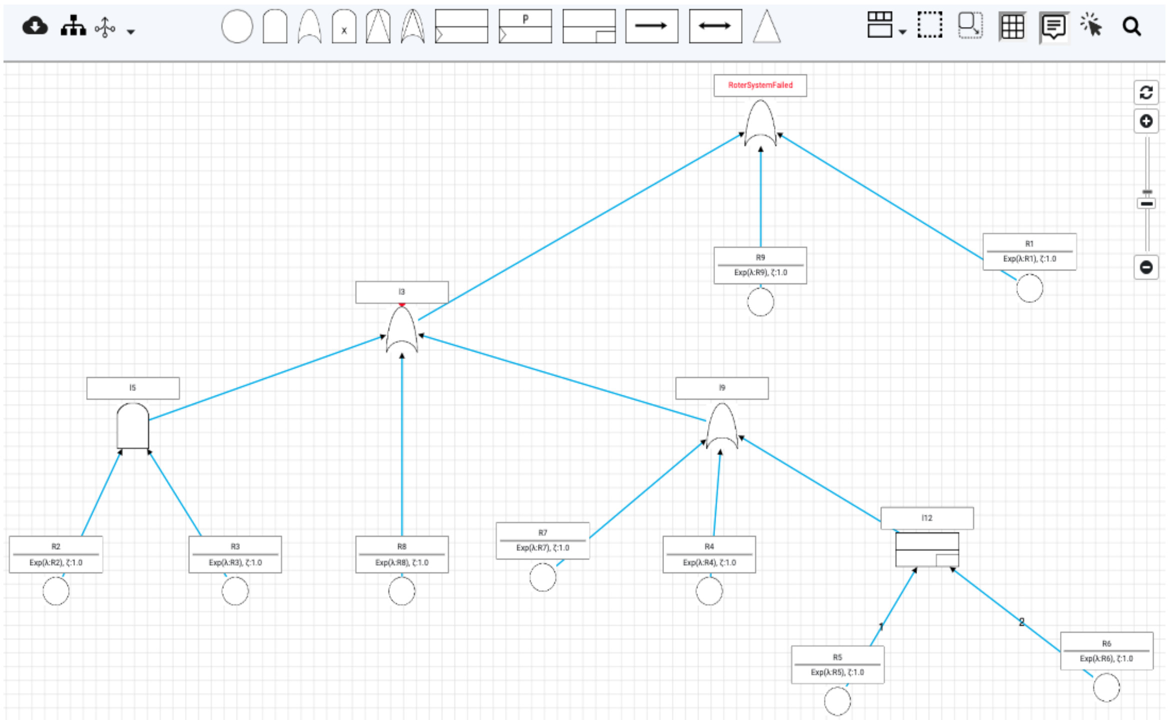
BE: 32 | Static Gates: 14 | Dynamic Gates: 2 | Blocks: 5

	Name	Type	Information	Description
1	OWTFailed	OR	toplevel: true	Floating offshore wind turbine system failure
2	I6	OR		Collision with other structures
3	FW4	BE	Exp (λ: FW4), ζ: 1.0	Collision with vessels in regular lines
4	FW5	BE	Exp (λ: FW5), ζ: 1.0	Collision with vessels in abnormal lines
5	FW6	BE	Exp (λ: FW6), ζ: 1.0	Collision with supply vessels
6	FW7	BE	Exp (λ: FW7), ζ: 1.0	Resonance
7	FW1	BE	Exp (λ: FW1), ζ: 1.0	Transport
8	FW2	BE	Exp (λ: FW2), ζ: 1.0	Fire
9	FW3	BE	Exp (λ: FW3), ζ: 1.0	Ice rainfall
10	WindTurbine Expand Block	BLOCK		
11	MooringSystem Collapse Block	BLOCK		
12	MooringSystem.MooringSystemFailed	OR	toplevel: true	Mooring system failure
13	MooringSystem.M7	BE	Exp (λ: M7), ζ: 1.0	Analysis and calculation fault
14	MooringSystem.I3	OR		Devices failure
15	MooringSystem.I7	OR		Anchor failure or dragging
16	MooringSystem.M1	BE	Exp (λ: M1), ζ: 1.0	The limit state of sea
17	MooringSystem.I8	OR		Other devices failure
18	MooringSystem.I10	OR		Fairlead failure
19	MooringSystem.M3	BE	Exp (λ: M3), ζ: 1.0	Fatigue of fairlead
20	MooringSystem.M2	BE	Exp (λ: M2), ζ: 1.0	Corrosion of fairlead
21	MooringSystem.I4	PAND		Weather and environment control error
22	MooringSystem.M5	BE	Exp (λ: M5), ζ: 1.0	Unsatisfied sea state for operation
23	MooringSystem.M6	BE	Exp (λ: M6), ζ: 1.0	Insufficient emergency measures for abrupt scenes
24	MooringSystem.MooringLines Expand Block	BLOCK		
25	Tower Expand Block	BLOCK		
26	SemiFloatingFoundation Expand Block	BLOCK		

c. Textual view. It shows the tree in Galileo format.


```
param G7;
param M5;
param R6;
param G1;
toplevel "OWTFailed";
"OWTFailed" or "I6_I6" "FW7" "FW1" "FW2" "FW3" "WindTurbine_WindTurbineFailed" "MooringSystem_MooringSystemFailed" "Tower_TowerFailed" "SemiFloatingFoundation_SFFFailed";
"I6_I6" or "I6_FW4" "I6_FW5" "I6_FW6";
"FW7" lambda=FW7 dorm=1.0;
"FW1" lambda=FW1 dorm=1.0;
"FW2" lambda=FW2 dorm=1.0;
"FW3" lambda=FW3 dorm=1.0;
"WindTurbine_WindTurbineFailed" or "WindTurbine_W2" "WindTurbine_W3" "WindTurbine_W4" "WindTurbine_W7" "WindTurbine_W8" "WindTurbine_W9" "WindTurbine_W10" "WindTurbine_W11"
"WindTurbine_W12" "WindTurbine_RoterSystem_RoterSystemFailed" "WindTurbine_Sensors_SensorsFailed" "WindTurbine_Generator_GeneratorFailed";
"MooringSystem_MooringSystemFailed" or "MooringSystem_M7" "MooringSystem_I3" "MooringSystem_I4";
"Tower_TowerFailed" or "Tower_I2" "Tower_T1" "Tower_T2" "Tower_T3" "Tower_T10" "Tower_T4" "Tower_T9";
"SemiFloatingFoundation_SFFFailed" or "SemiFloatingFoundation_F1" "SemiFloatingFoundation_I3" "SemiFloatingFoundation_I4" "SemiFloatingFoundation_F9";
"I6_FW4" lambda=FW4 dorm=1.0;
"I6_FW5" lambda=FW5 dorm=1.0;
"I6_FW6" lambda=FW6 dorm=1.0;
"WindTurbine_W2" lambda=8e-06 dorm=1.0;
"WindTurbine_W3" lambda=1e-05 dorm=1.0;
"WindTurbine_W4" lambda=1e-05 dorm=1.0;
"WindTurbine_W7" lambda=1.6e-05 dorm=1.0;
"WindTurbine_W8" lambda=2e-06 dorm=1.0;
"WindTurbine_W9" lambda=3.9e-05 dorm=1.0;
"WindTurbine_W10" lambda=1.3e-05 dorm=1.0;
"WindTurbine_W11" lambda=4e-06 dorm=1.0;
"WindTurbine_W12" lambda=3.3e-05 dorm=1.0;
"WindTurbine_RoterSystem_RoterSystemFailed" or "WindTurbine_RoterSystem_R9" "WindTurbine_RoterSystem_I3" "WindTurbine_RoterSystem_R1";
"WindTurbine_Sensors_SensorsFailed" vot2 "WindTurbine_Sensors_W61" "WindTurbine_Sensors_W62" "WindTurbine_Sensors_W63";
```

17. Click on the icon  to enable selection of multiple elements in the grid view. This can be done by clicking on a screen and then drawing the mouse. All elements with a rectangle will be selected, which can then be moved together.
18. Click on the icon  to enable navigator at the bottom of the screen.
19. Click on the icon  to show the grid on the screen.
20. Click on the icon  to show the summary information about each element on the screen.



21. Click on the icon to display summary information about an element on hovering.

22. Click on the icon to search any element on the screen.

Quantifiable States

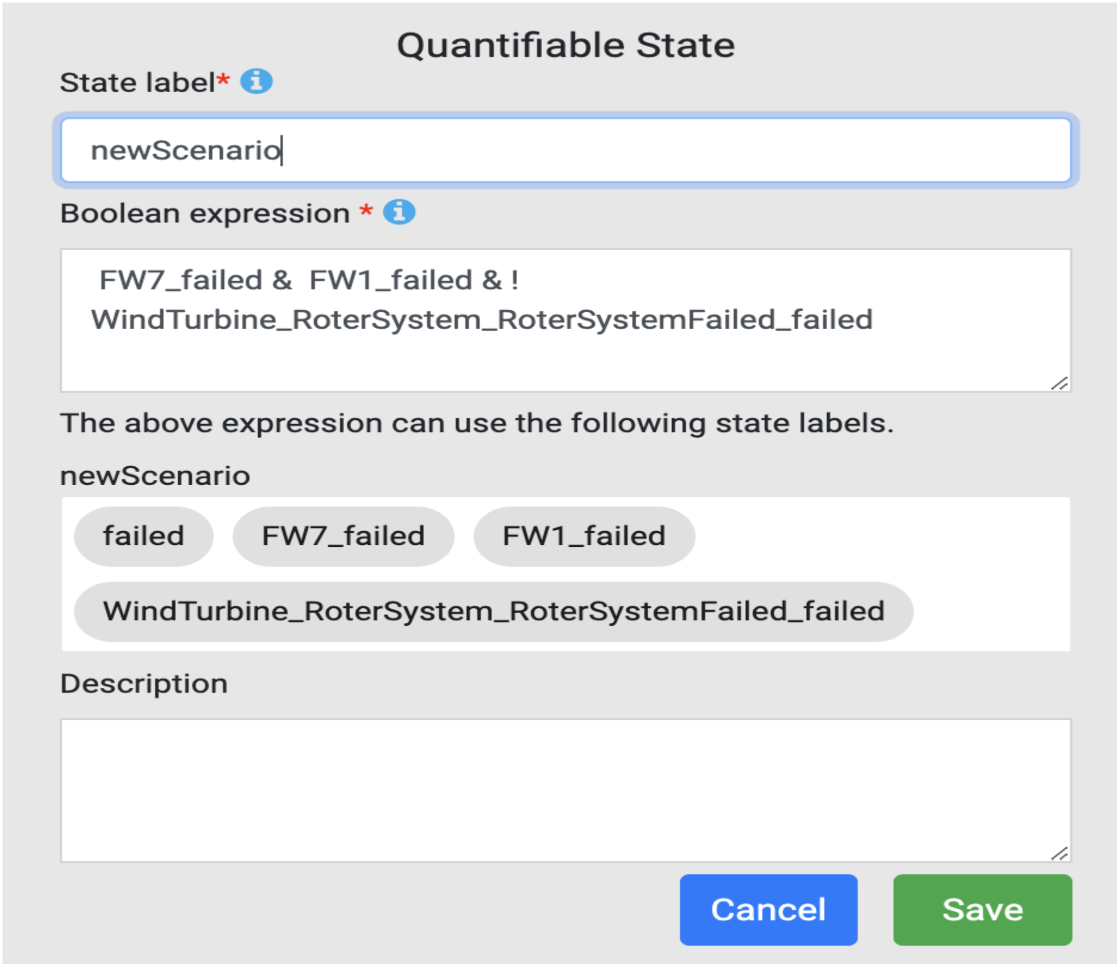
1. In advance view, “Quantifiable States” are visible in the left panel under each failure model. Click on the “Quantifiable States”, it will show a screen with two tabs:
   Basic: It will display all elements of the fault tree whose “Failure probability is quantifiable” checkbox is selected. Each element represents a sub-scenario about which metrics can be verified.

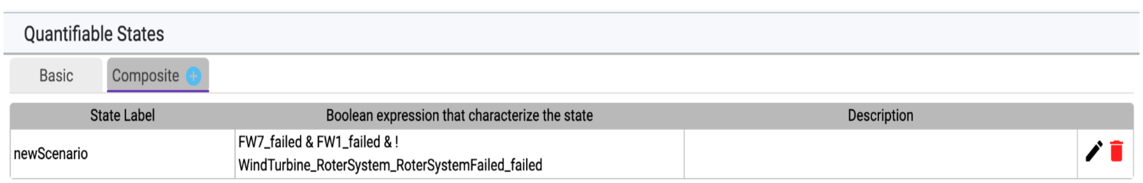
Quantifiable States

Basic | Composite

State Label	Element that fail in state	Description
failed	Root Element	
FW7_failed	FW7	
FW1_failed	FW1	
WindTurbine_RoterSystem_RoterSystemFailed_failed	WindTurbine_RoterSystem_RoterSystemFailed	

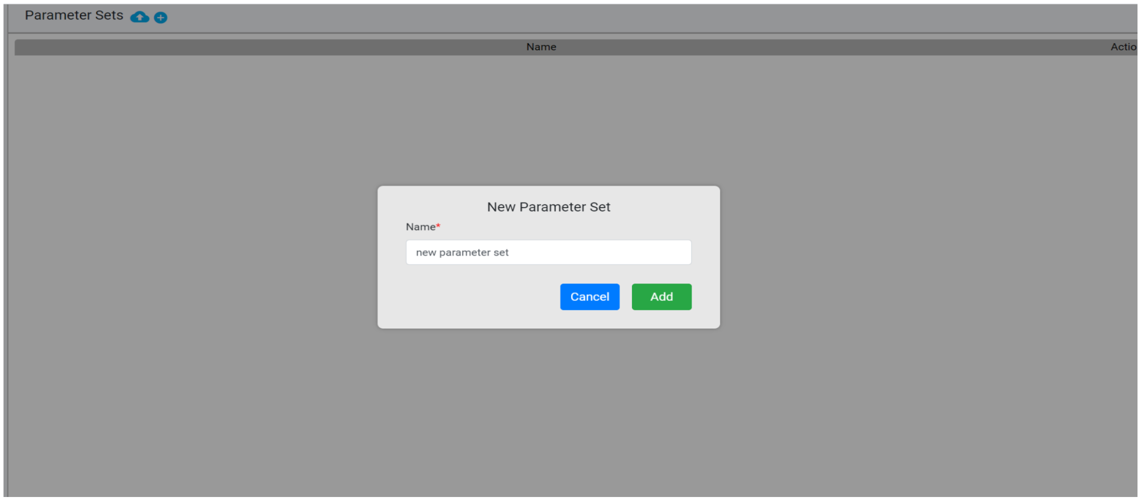
   Composite: On this tab, we can create new sub-scenarios by writing boolean expressions on the basic scenarios shown in the Basic tab.



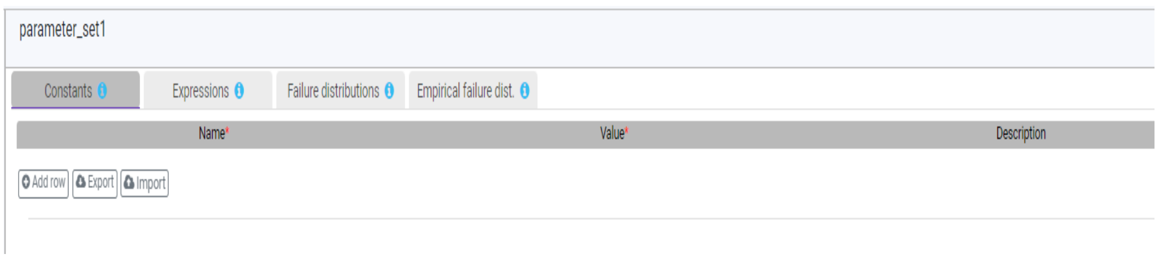


Parameter Sets

1. A parameter set contains constants, real-value expressions, failure distributions and empirical failure distributions (calculated from data sets), which are used inside fault trees. By changing the values of parameters, different variants of fault models can be created. Add a new parameter set by clicking on the plus icon.



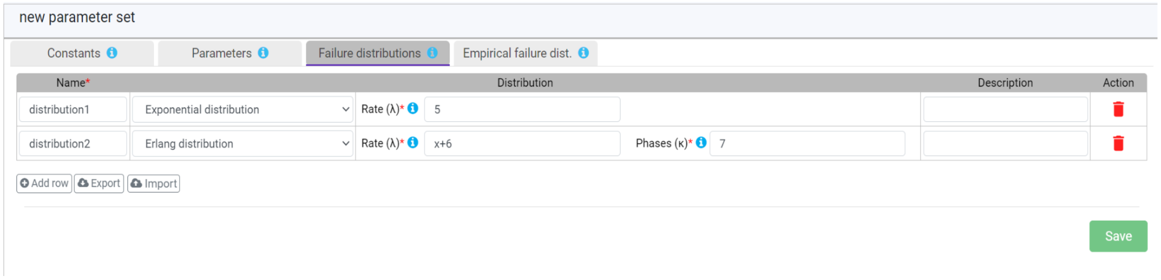
After adding, you see a screen with tabs: constants, parameters (real-value expressions), failure distributions and empirical failure distributions.



- **Constants**
  Constants can only be numeric e.g. 4 , 2.3, 4e-6 etc. Their value can be changed at the time of analysis. For example, graphs can be plotted for matric results against ranges of values of constants.

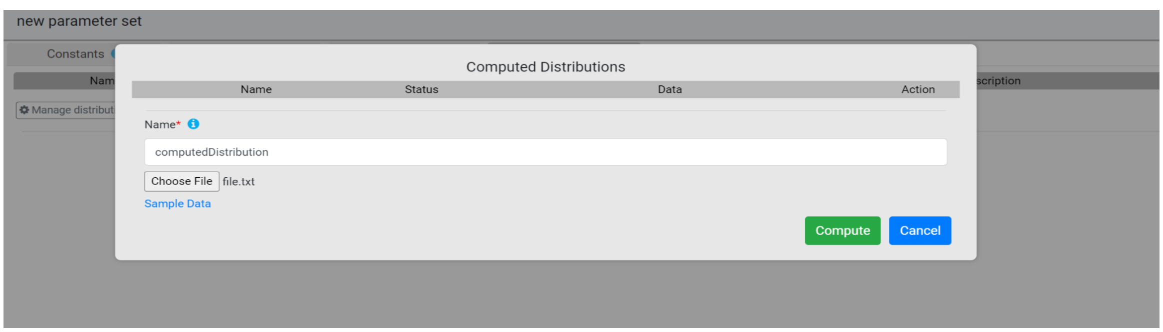
- **Real-value Expressions**
  These are non-negative, real-value expressions, which can use constants (defined above) e.g. x + 2 where x is a constant. The grammar of expression is given here.

- **Failure distributions**
  Failure distributions can be exponential, erlang, Weibull, log normal, and constant probability. Multiple distributions can be added by pressing add row.

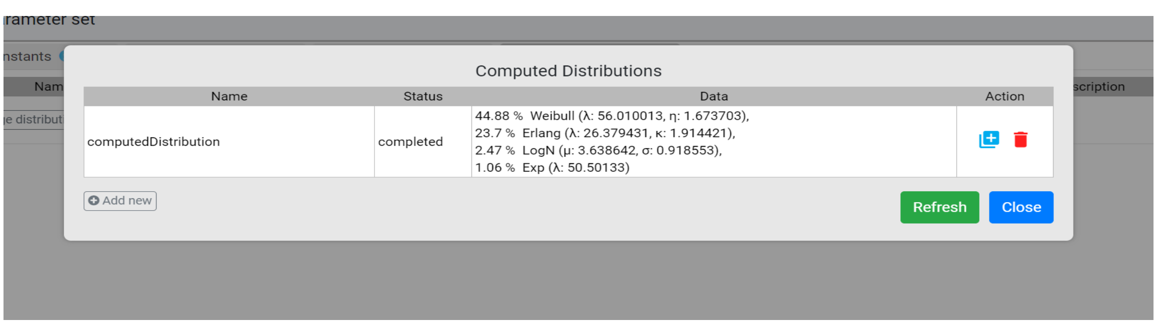


- **Empirical failure dist.**
  Empirical distributions are calculated from a data set using statistical methods. For example, a historical failure data of a component is used to estimate the tentative failure probability distributions, which might have generated it, sorted according to their goodness-to-fit (GTF) values -- GTF value indicates the chance the data was generated by the corresponding distribution.

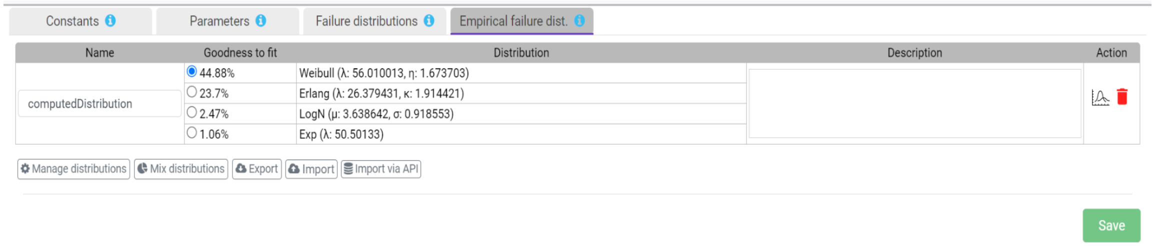
  - Generating empirical distribution. An empirical distribution can be added by clicking the “Manage distributions” button. It will display all empirical distributions that have been generated previously and stored at the server side. One can compute a new distribution by specifying the file that contains data on which distribution is to be approximated.



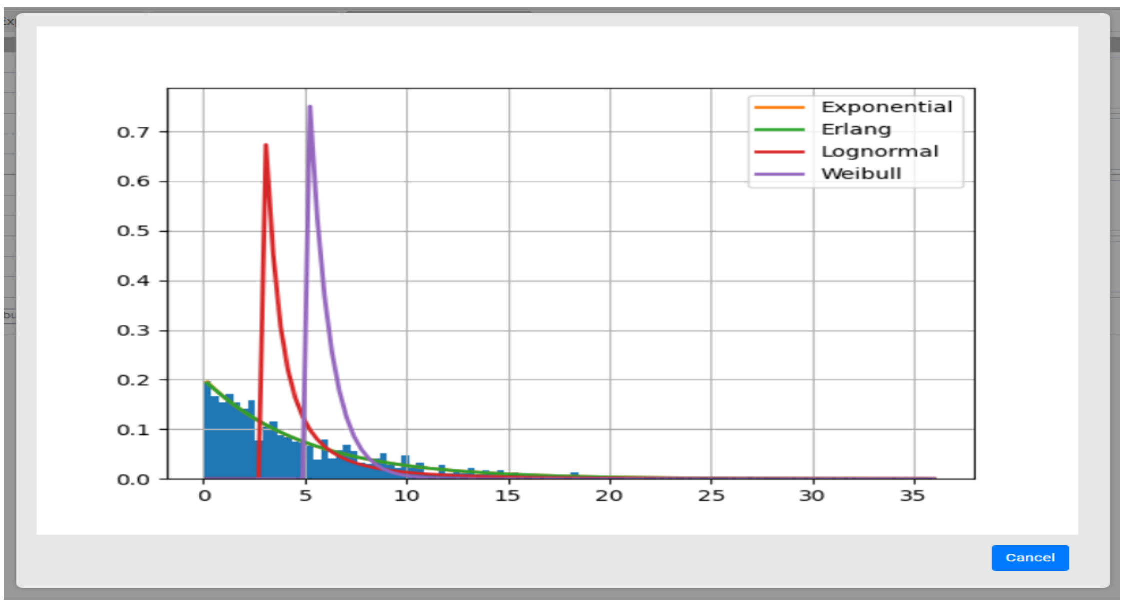
After computations are done, update the data on the popup after pressing the refresh button. The newly computed distribution will become visible.

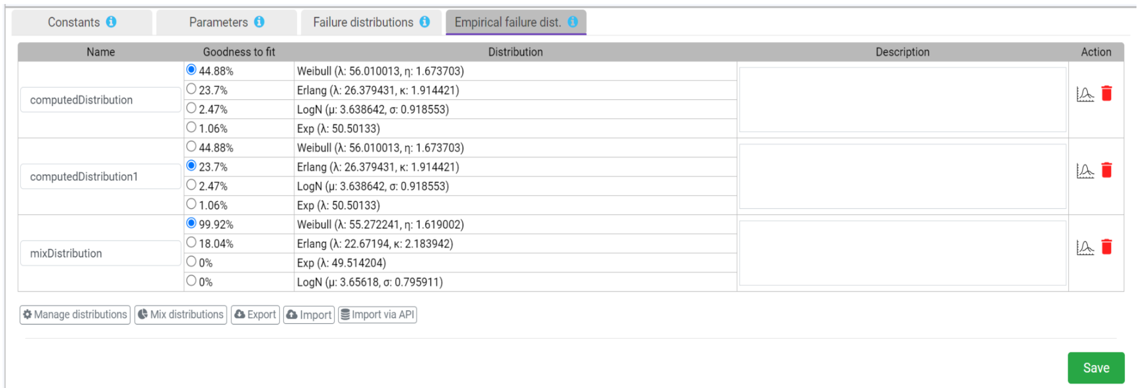


You can add these results as a new distribution in empirical distribution by pressing  icon.

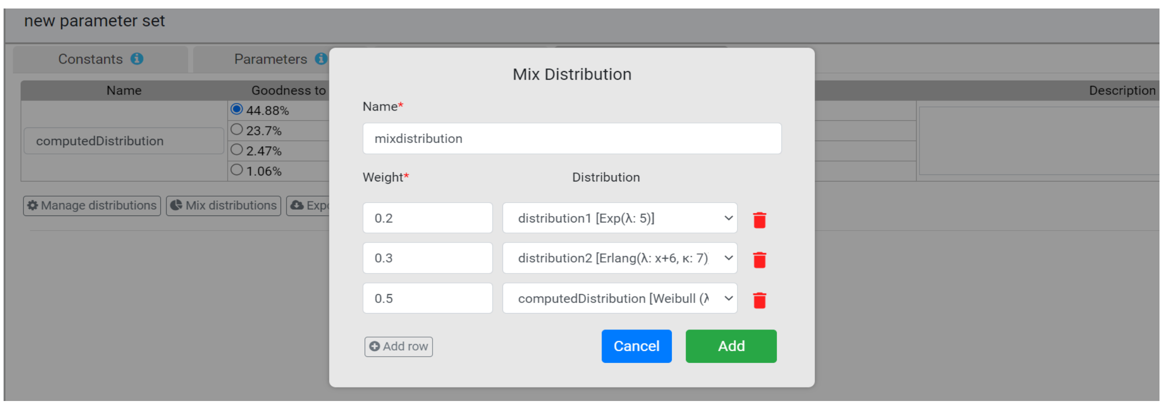


Each distribution in the set has a goodness-to-fit value (GTF) which indicates the chance the data was generated by the corresponding distribution. You can see how these distributions fit the data by clicking on the graph icon in the action column.





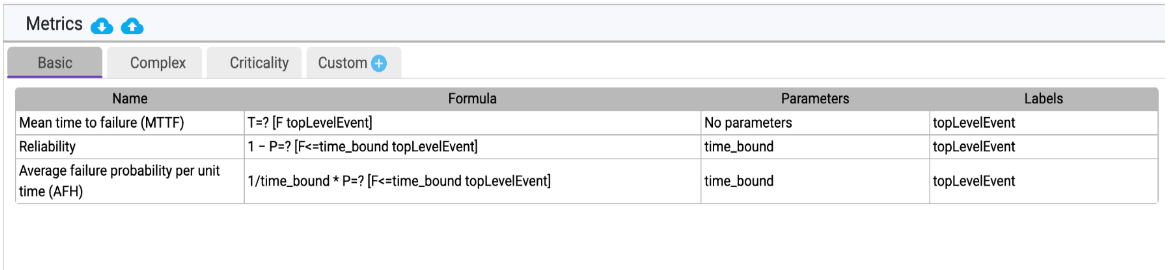
- Generating mixture distribution: a mixture distribution can also be generated by clicking the "Mix distributions" button. It will show a popup where distributions along with their weights can be added. For example, d3 = 0.3*d1 + 0.7*d2, where d1 and d2 are existing empirical failure distributions.



- Using Empirical distribution: Each data may fit on multiple distributions, which are sorted according to their goodness-to-fit values, therefore we provide a radio button to select any distribution that we want to use.
- Moreover you can import and export empirical distributions and use them in other projects. You can also import and export the whole parameter set.

Metrics

1. In advance view "Metrics" link is visible in the left panel. Click on it, a screen with four tabs: Basic, Complex, Criticality and Custom will be visible.



Name	Formula	Parameters	Labels
Mean time to failure (MTTF)	T=? [F topLevelEvent]	No parameters	topLevelEvent
Reliability	1 – P=? [F<=time_bound topLevelEvent]	time_bound	topLevelEvent
Average failure probability per unit time (AFH)	1/time_bound * P=? [F<=time_bound topLevelEvent]	time_bound	topLevelEvent

Basic. It contains four important metrics that are verified in most of the reliability analysis cases:

i. Mean-time-to-failure. Expected time to system failure or scenario occurrence.
ii. Reliability: Probability of failure in a given time bound.
iii. Unreliability: The complement of reliability (1- Reliability).
iv. Average failure probability per hour.

Complex. It contains six metrics. These metrics cannot be verified directly by the Storm model-checker. They need some additional computations at the back end for their verification.

v. Full Function Availability (FFA) describes the time-bounded probability that the system provides full functionality, i.e., it has neither failed nor degraded. It is described as the complement of the time-bounded reachability of a failed or degraded state.
vi. Failure Without Degradation (FWD) describes the time-bounded probability that the system fails without being degraded first. It is the time-bounded reach-avoid probability of reaching a failed state without reaching a degraded state.
vii. Mean Time from Degradation to Failure (MTDF) describes the expected time from the moment of degradation to system failure. It is obtained by taking the expected time of failure for each degraded state and scaling it with the probability to reach this state while not being degraded before.
viii. Minimal Degraded Reliability (MDR) describes the criticality of degraded states by giving the worst-case failure probability when using the system in a degraded state. For all degraded states the time-bounded reachability of a TLE failure is computed. The MDR is the minimum over the complement of this result for all degraded states.
ix. Failure under Limited Operation in Degradation (FLOD) describes the probability of failure when imposing a time limit for using a degraded system. For all degraded states the time-bounded reachability probability of a failed state is computed within the restricted time-bound given by a drive cycle. This value is scaled by the time-bounded reach-avoid probability of reaching a degraded state without degradation before.
x. System Integrity under Limited Fail-Operation (SILFO) considers the system-wide impact of limiting the degraded operation time. SILFO is split

into two parts considering failures without degradation (FWD) and failures with degradation (FLOD).

Metrics

Basic | Complex | Criticality | Custom

Name	Formula	Parameters	Labels
Full Function Availability (FFA)	1 − P=? [F<=time_bound topLevelEvent | degraded]	time_bound	topLevelEvent, degraded
Failure without degradation (FWD)	P=? [(!degraded) U<=time_bound (topLevelEvent & !degraded)]	time_bound	topLevelEvent, degraded
Mean time from degradation to failure (MTDF)	Σ_{s}∈degraded (P=? [(!degraded U s)] * T^{s}=? [F topLevelEvent])	No parameters	topLevelEvent, degraded
Minimal degraded reliability (MDR)	argmin$_{s}$ ∈ degraded (1 − P^{s}=? [F<=time_bound topLevelEvent])	time_bound	topLevelEvent, degraded
Failure under limited operation in degradation (FLOD)	Σ_{s}∈degraded(P=? [(!degraded) U<=time_bound s] * P^{s}=? [F<=drive_cycle topLevelEvent])	time_bound, drive_cycle	topLevelEvent, degraded
System integrity under limited fail-operation (SILFO)	1 − (FWD + FLOD)	time_bound, drive_cycle	topLevelEvent, degraded

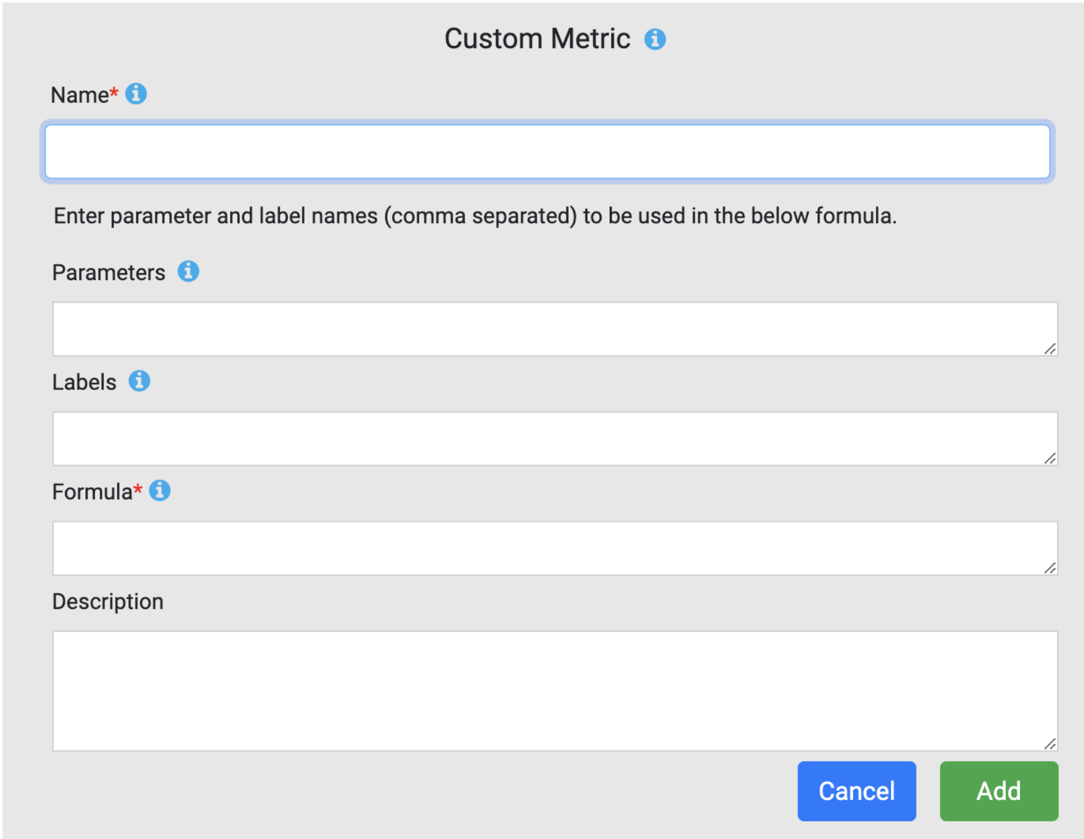
Criticality. It contains only the Birnbaum Index (BI) at the moment – a common way to measure the sensitivity of the system to an element is the Birnbaum importance index.

Metrics

Basic | Complex | Criticality | Custom

Name	Formula	Parameters	Labels
Birnbaum Index (BI)	(P=? [F<=time_bound component_failed]/$P_{\|s_{o}}$=? [F<=time_bound component_failed]) * (P=? [F<=time_bound topLevelEvent & component_failed] / P=? [F<=time_bound component_failed] - P=? [F<=time_bound topLevelEvent & !component_failed]/P=? [F<=time_bound !component_failed])	time_bound	topLevelEvent, component_failed

Custom. One can create custom metrics on the “Custom” tab. It allows specifying metrics using continuous stochastic logics (CSL).



xi. Parameters and labels used inside metrics formulae must have unique names among themselves, starting with a letter or underscore (_) followed by underscores, letters, and/or numbers. They must not be from the list of keywords - - true, false, Pin, Pmax, Smin, Smax, Tmin, Tmax, LAmin, LRAmax, P, R, T, S, LRA, min, max, G, U, F, W, C, I, failed.

xii. The formula can be defined using probabilistic computation tree logic (PCTL)/continuous stochastic logic (CSL). For example, P = ? [true U <=10000 (failed & ! mode1)], where failed and mode1 represent quantifiable states. The grammar of expressions is given here.

xiii. Note. The parameters given on the above screen are exclusively dedicated to metrics. Their values cannot be taken from the parameter set that is attached with a failure model at the time of analysis. However, their values can be changed at the time of analysis, and plots can be drawn for metric values.

System analysis with SAFEST

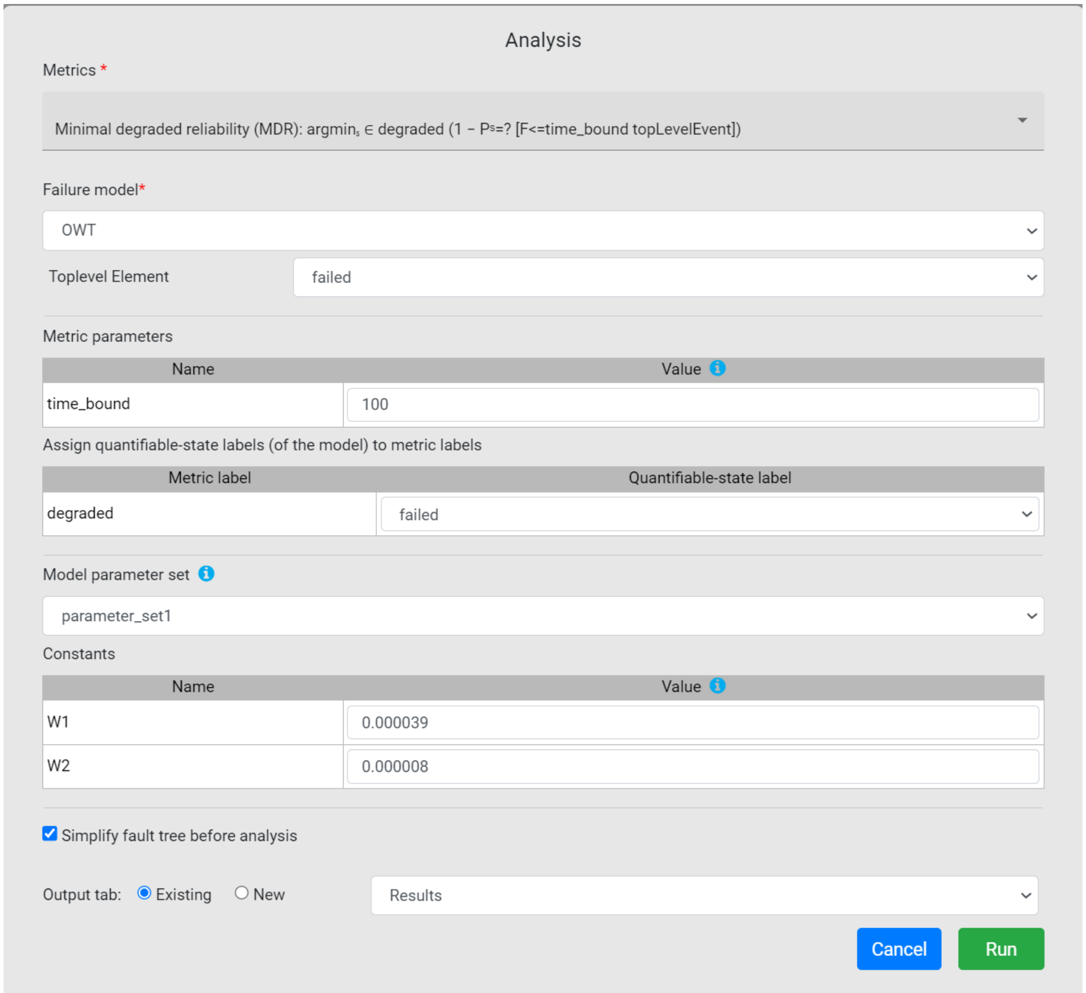
In the SAFEST tool, different types of analysis – from basic to complex – can be performed for each failure model in the project.

1. (Exact) Analysis

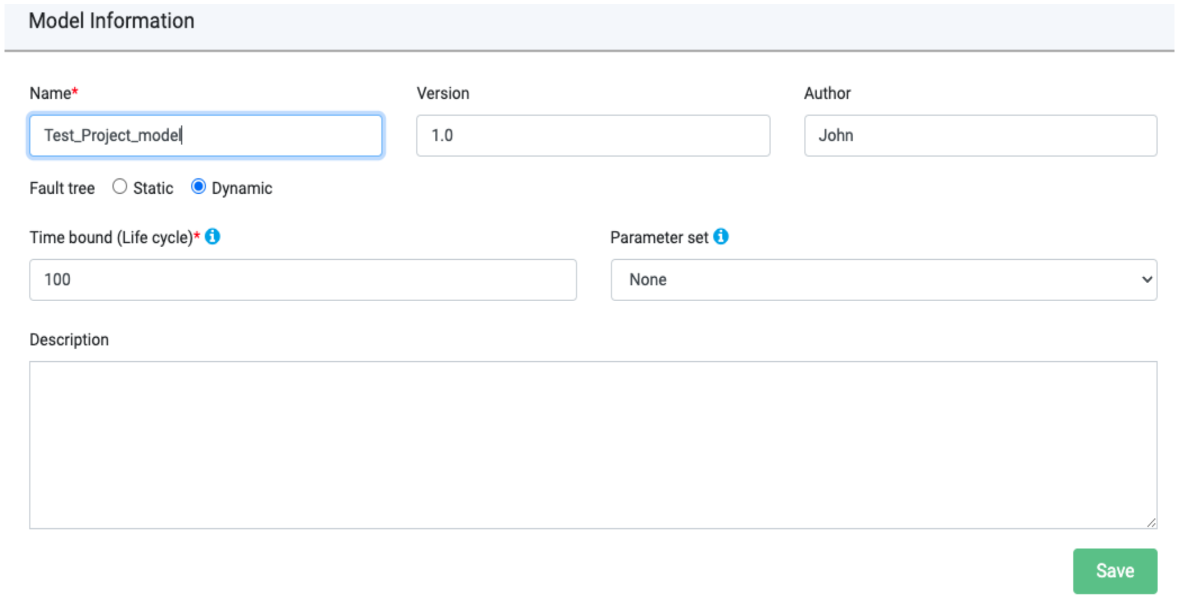
Complex systems usually have dynamic behavior because of e.g. spare components, failure sequence among components, functional dependencies, etc. The analysis of such systems is usually quite complex which is usually based on simulation or generalization techniques. Unlike others we implement formal verification techniques e.g. probabilistic model-checking, and thus provide exact results on measures of interest.

Click on the “Analysis” link under “Computing” in the left panel. The following window will appear with four tabs for different classes of metrics.

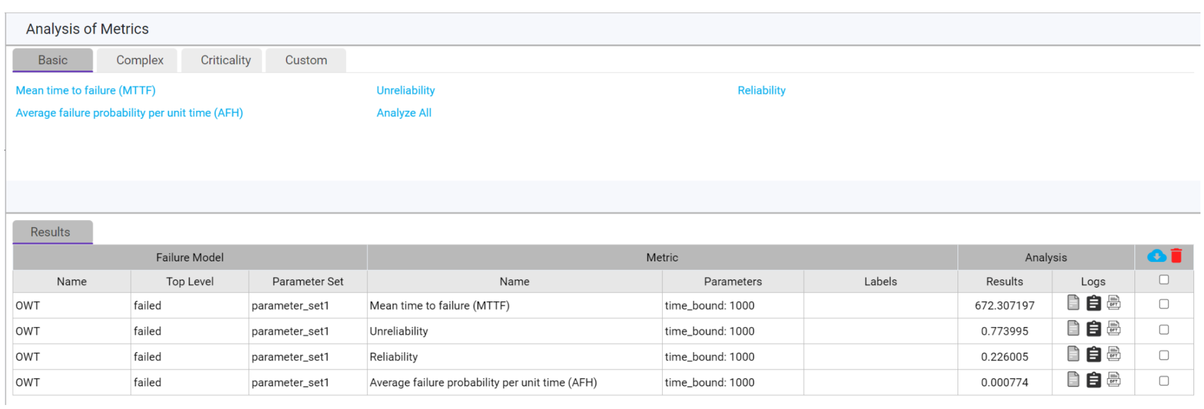
- One can verify a metric on each tab, the mechanism is more or less the same. For example, click on the “Minimal degraded reliability (MDR)” link on the complex tab. The following window will appear



- In the "Failure model" dropdown, a model that is selected as a default model in the "Failure Models" window is automatically selected.
- The time-bound (life cycle) parameter of Metric is assigned a value that is entered at the time of failure model creation.



- Give values to metric parameters. Note that metric parameters cannot take values defined in the parameter set attached with the model.
- Assign each label in the metric (which represents a class of states) a quantifiable-state (which indicates a class of model sub-scenarios) of the model.
- A parameter set which is attached with the selected failure model (above) is automatically selected. It can be changed at this point.
- Optionally, change the values of constants defined in the parameter set which is selected. Note that values of other elements in the parameter set (real-value expressions, (empirical) failure distributions) cannot be changed at the time of analysis.
- Select analysis method either Markov or BDD. Note that BDD analysis does not work for all of the metrics.
- Finally, select a tab on which result of the analysis has to be displayed.

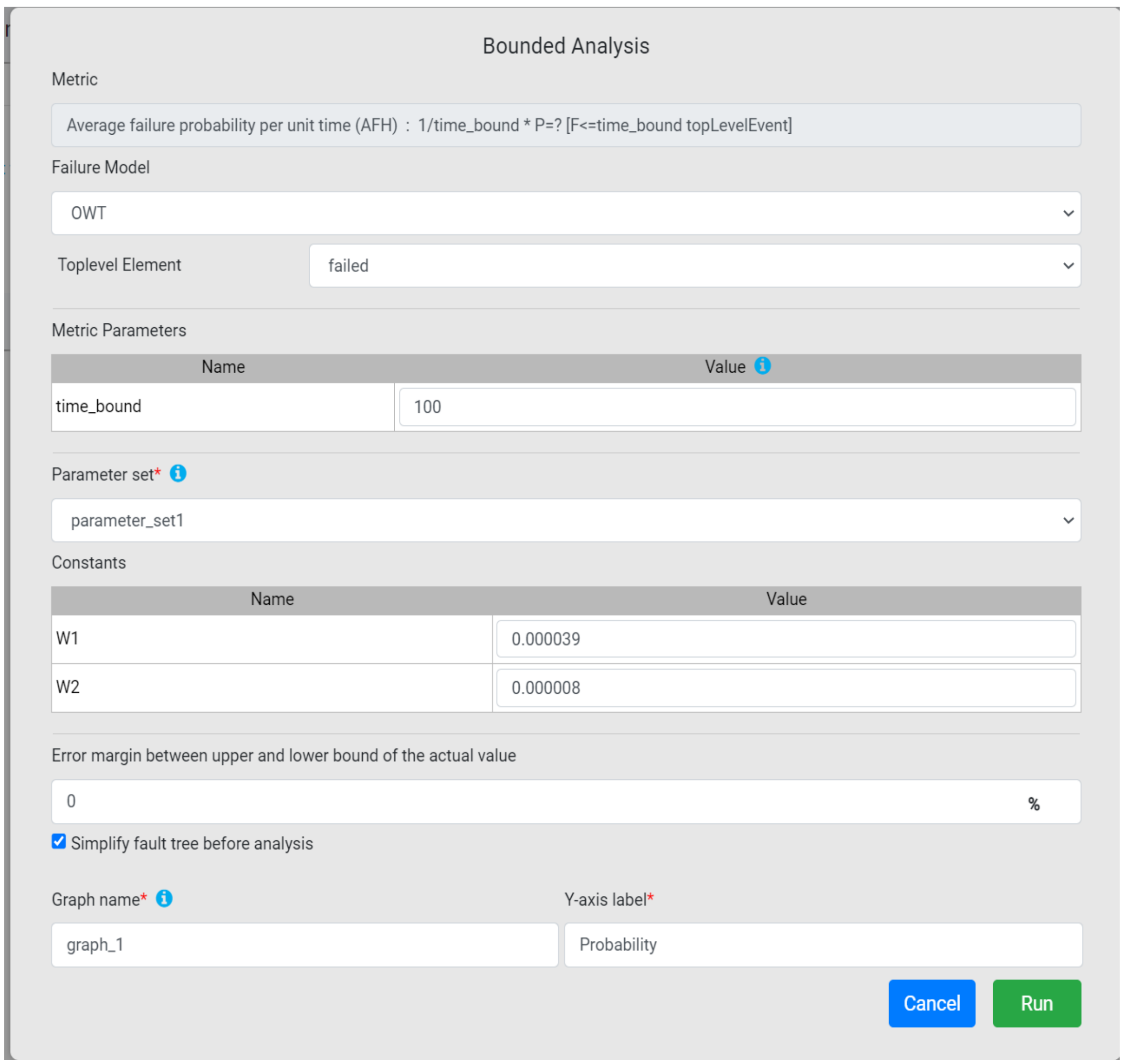


- The results can be downloaded as a csv file for each tab separately.

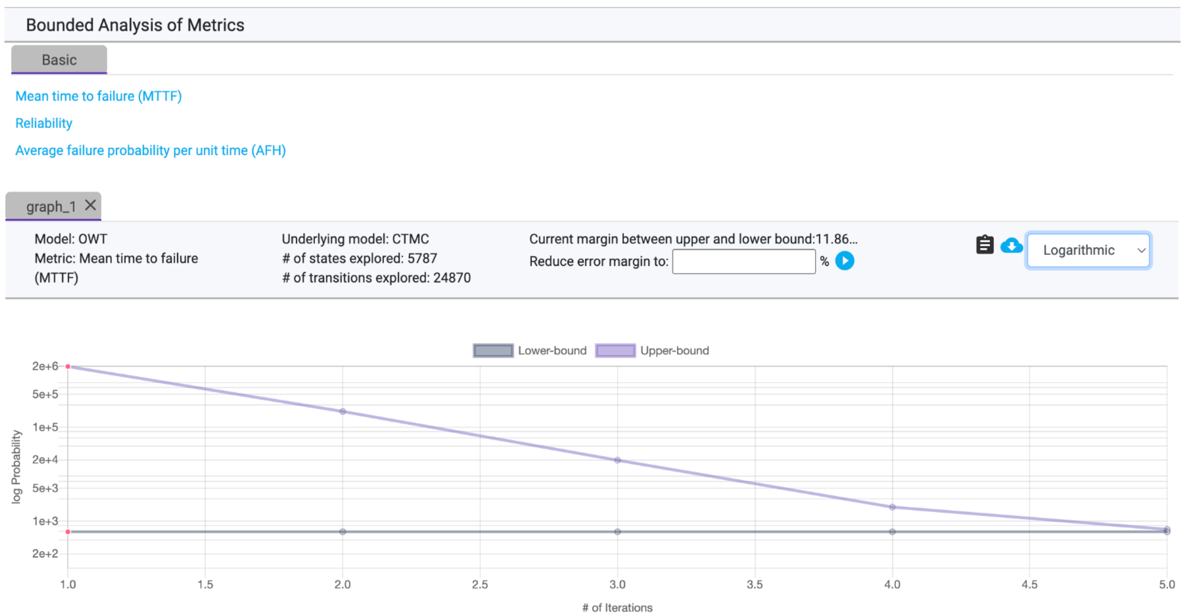
2. Bounded analysis

In order to compute exact results for measures, first the full state space is constructed, and then analyzed. However, many states in the state space only marginally contribute to the result. If one is interested in an approximation of the MTTF (or the reliability), these states are of minor interest. We implemented the algorithms, proposed by Dr. Matthias Volk et. al., that generate state-space on-the-fly, and then compute an upper and a lower bound to the exact results on a partially unfolded system, which might be much smaller as compared to the fully unfolded system. The approximation is sound ensuring the exact result lies between these two bounds.

Click on the “Bounded Analysis” link under “Computing” in the left panel and then click e.g. “Mean-time-to-failure” link. The following window will appear:



- All fields are filled up as described in the “Analysis” case with few additions:
- Enter acceptable error margin between upper and lower of the actual value.
- Optionally enter graph name and the label of its Y-axis. Note X-axis will always represent the number of iterations in this case.

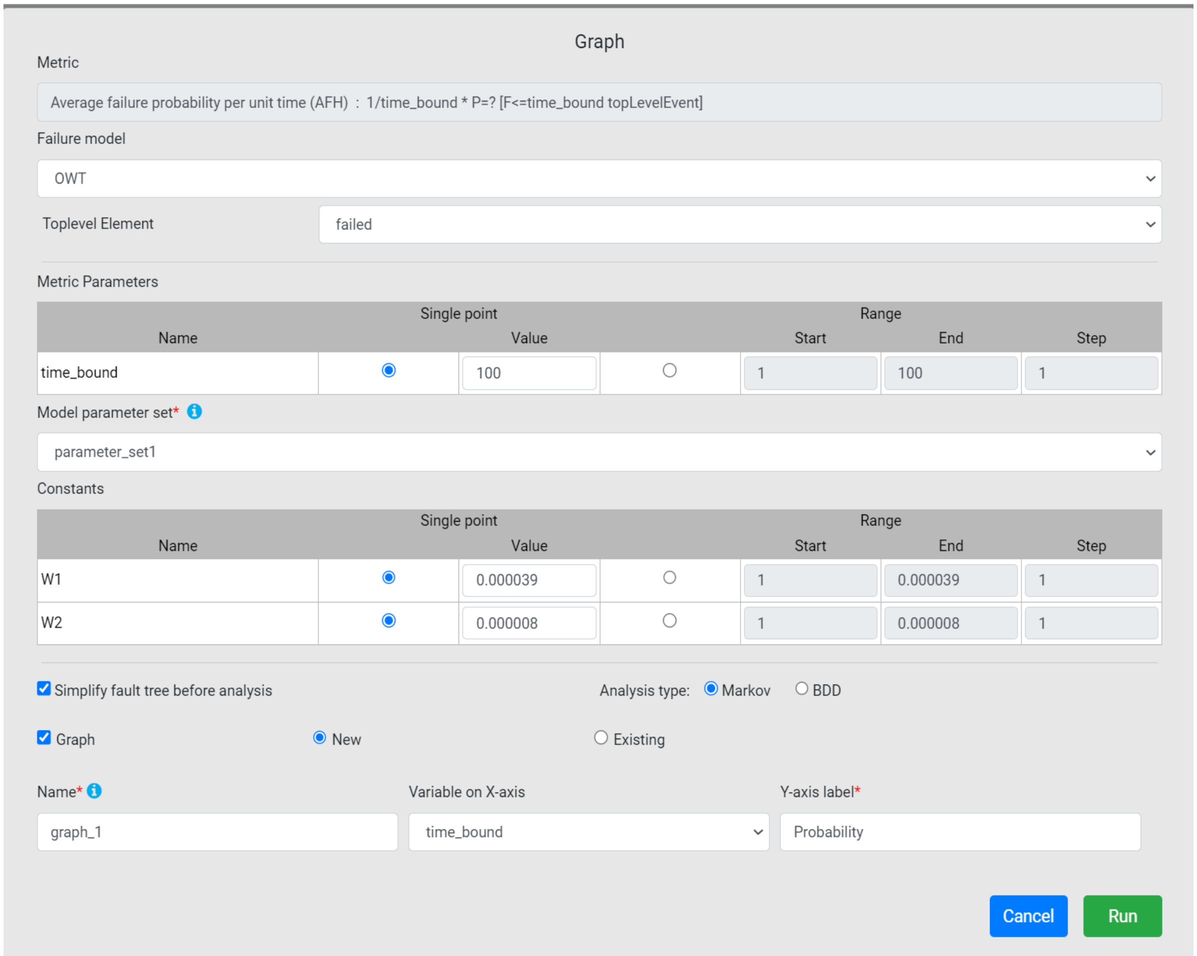


- The upper line in the graph shows the upper bound whereas the lower line shows the lower bound on the actual value of the metric.
- In addition, we show the number of generated states and the transitions explored so far.
- In case one is interested to further reduce the error margin, it will be insert a new value in the text field and click the play button  .
- One can apply a log function on the values of Y-axis by selecting it on the right side of the graph.
- The graph values can be downloaded by clicking on the download icon.

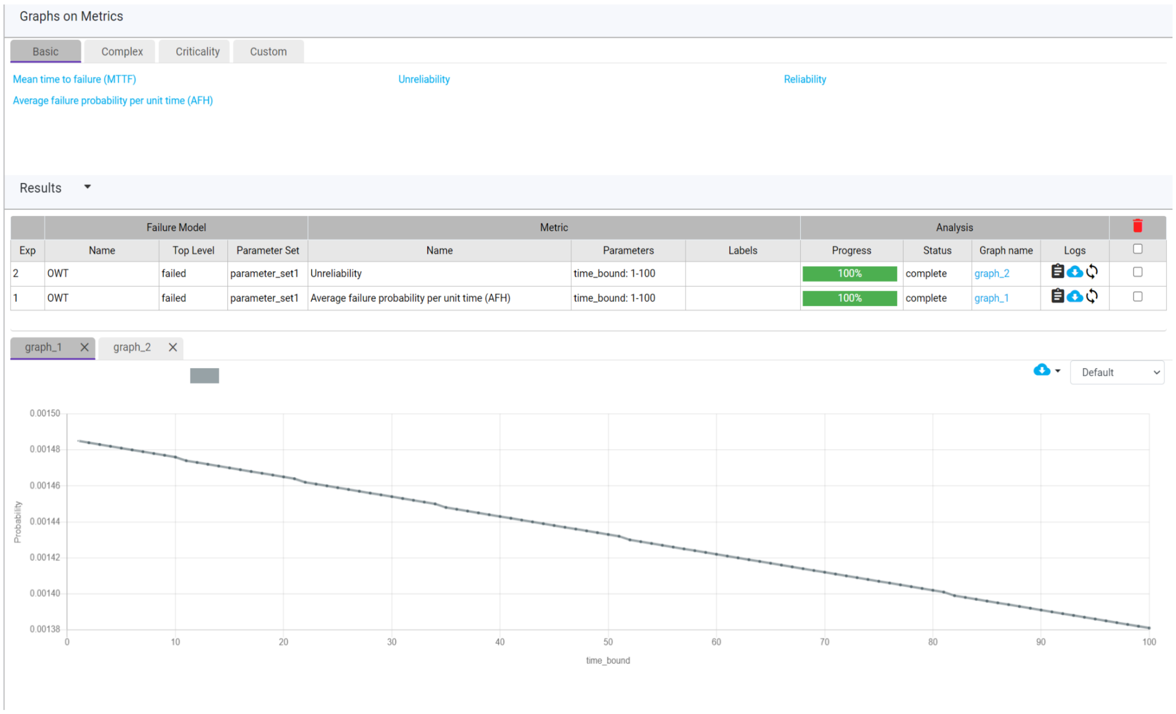
3. Graphs

Reliability measures help figuring out optimal maintenance schedules of systems thus reducing their downtimes and saving cost at the same time. Fault trees that model sub-systems of systems can even predict their health individually thus helping make even more detailed maintenance schedules. We provide a graphical interface to plot and compare measures of interest e.g. reliability of different sub-systems, which is helpful in deciding maintenance schedules.

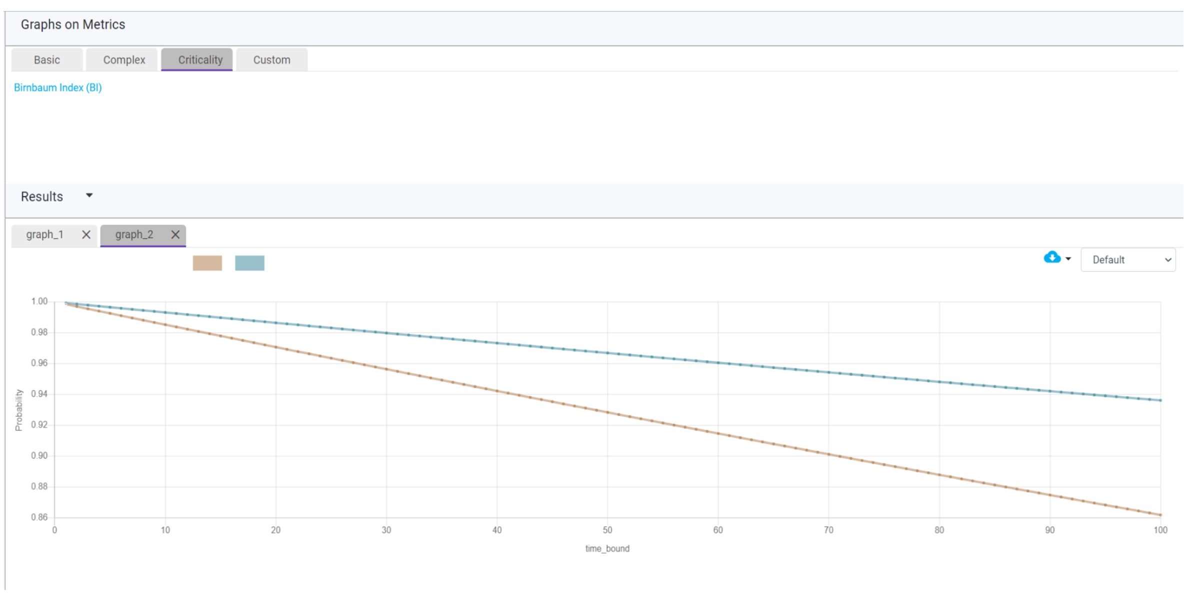
Click on the “Graphs” link under “Computing” in the left panel and then click “Reliability”. The following window will appear:



- One can specify a range of values of Metric parameters and Constants defined in the parameter set which is selected above.
- A graph can be plotted on an existing graph as well that has the same variable on X-axis.
- The variable on the X-axis of the graph can be specified either from the Metric parameters or Constans in the parameter set which is selected above.



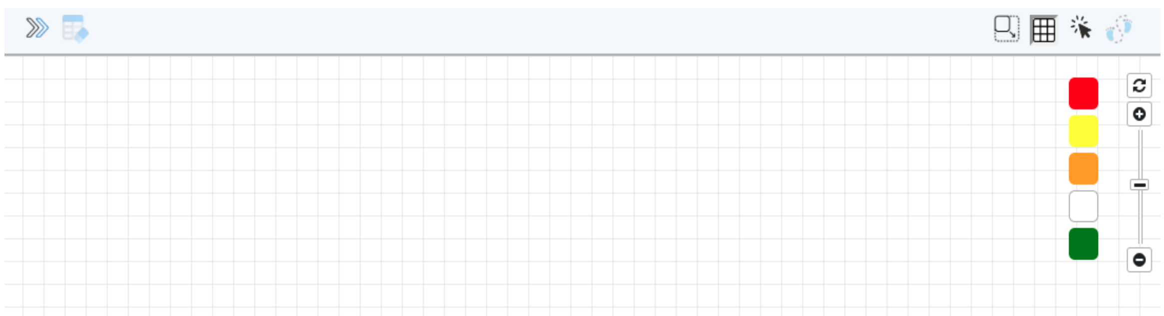
- In case of Birnbaum Index (on Criticality tab), one can draw plot for multiple components at the same time as:



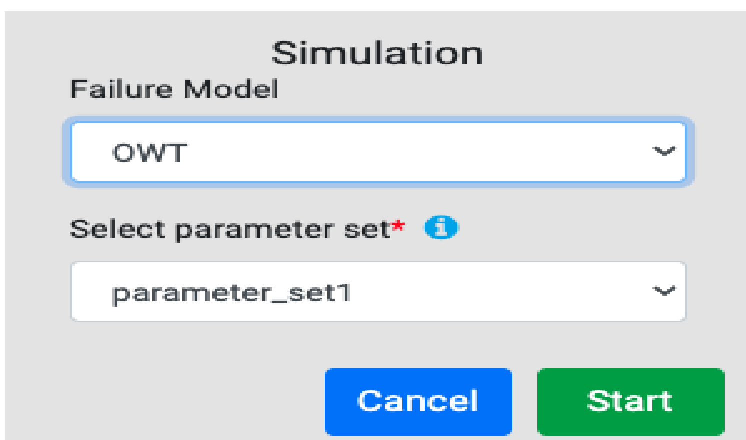
4. Interactive simulation

The idea is to interactively visualize a sequence of failures in a DFT. The user would start with a usual DFT and could select one of the basic events (BE) that should fail first. Based on this, the status of each DFT element (failed, operational, fail-safe, claiming in SPAREs, etc.) is redetermined and then visualized. Afterwards, another BE can be selected to fail and so forth. The main benefit of this feature is that the idea of DFTs should become much clearer as users can try out the behavior by themselves.

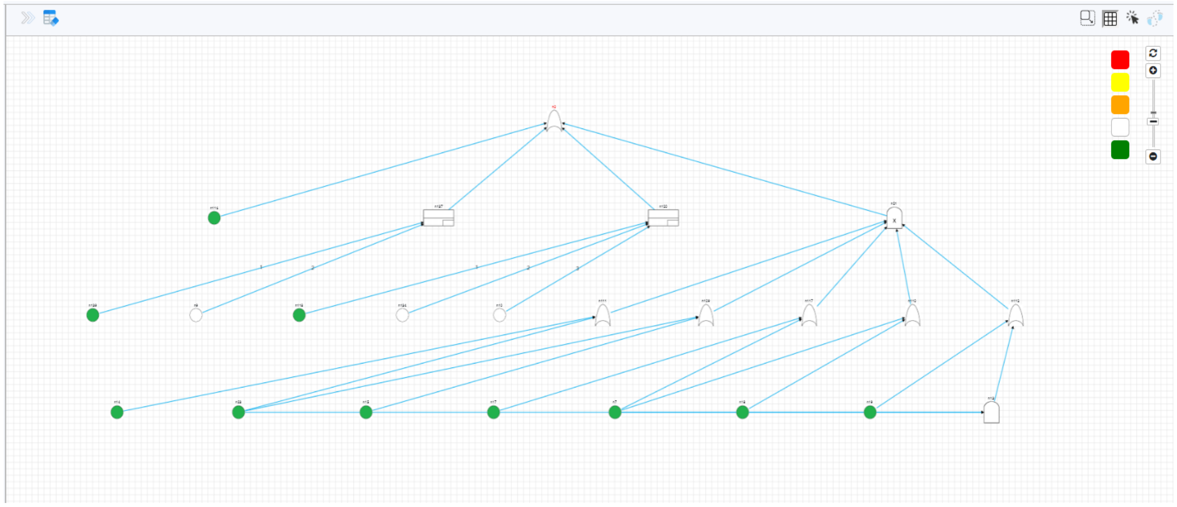
Click on the “Interactive Simulation” link under “Computing” in the left tab. The following screen will appear.



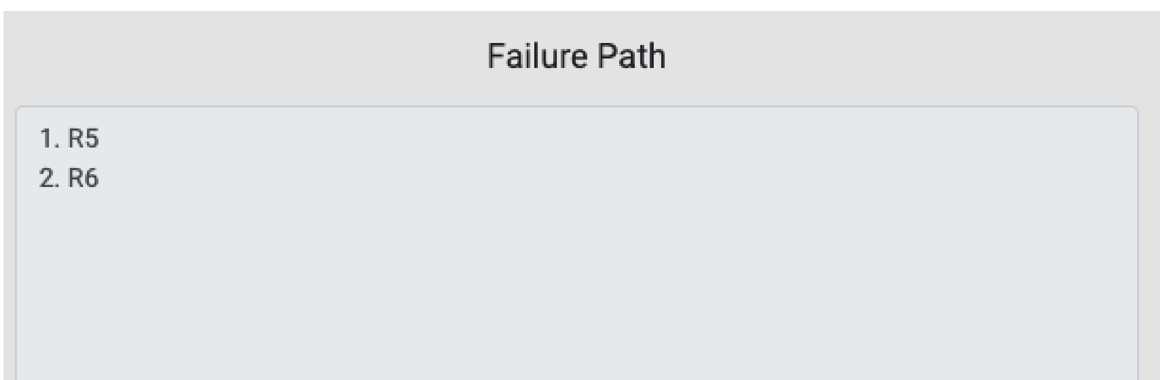
Click on the icon to start the simulation. User will be prompted to select a failure model and a parameter set as:



On clicking “Start” the following screen appears:



- All basic events (BEs) that can fail are shown as green.
- User clicks any green BE to fail it. Its color will be turned into Red. After this, BEs which are operational and cannot fail remain White, those which are in fail-safe state are Orange, those which are in dont-care state are Yellow.
- User keeps on failing green BEs, and in return the failure keeps on moving up the tree until the top level event turns Red showing the failure of the top level event.
- The sequence of failures can be shown by click on the icon  as:

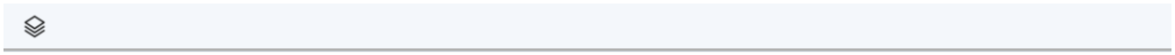


- User can restart simulation by clicking on the icon  .

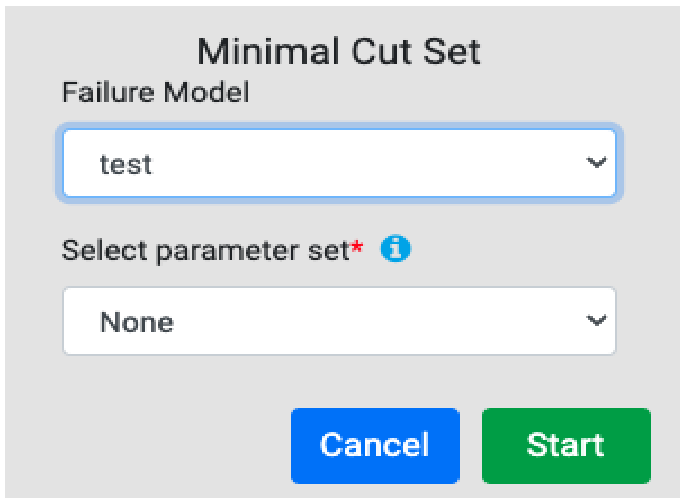
5. Minimal cut set (for static fault trees)

Cut sets represent sets of BEs whose failure leads to the failure of the top level element of a fault tree. A minimal cut set is a set whose proper subset cannot be a cut set itself. Cut sets cannot be calculated for dynamic fault trees because of the dynamic nature of the system.

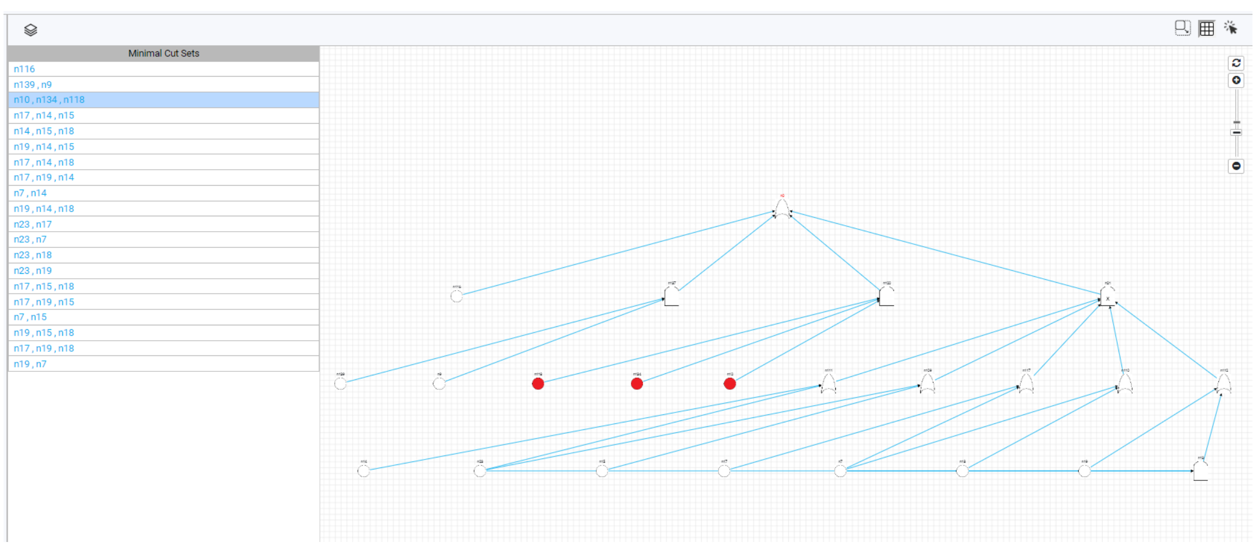
Click on the “Minimal cut set” link under “Computing” in the left tab. The following screen will appear.



Click on the icon  to start. User will be prompted to select a failure model and a parameter set as:



On clicking “Start”, minimal cut sets are computed and displayed on the screen as:



- All minimal cut sets will be shown on the left of the screen.
- On clicking a cut set, the corresponding BEs will be highlighted (in Red) in the tree.

Main Toolbar

1) File Menu:
    a) New Project: it starts a process to generate a system failure model from scratch.
    b) Open Project. It opens an already existing system failure model.
    c) Open Recent Project: It'll restart a recently finished project. Even if the SAFEST crashes for any reason, a project is still in the working folder and can be accessed again.
    d) Export Project. It exports the current project in a file with .safest extension.
2) View:
    a) Simple View. It is for simple users. Under this view, a user cannot create quantifiable states as well as define custom metrics.
    b) Advance View. It is for advanced users or researchers. This view gives the full functionality of the SAFEST tool.

3) Help:
    a) Documentation: It contains the grammar for expressions.
    b) Activation key: Here you can add a license key and activate SAFEST tool functionality.

Annotate SysML Models with Safety Information

In order to annotate SysML model elements with safety information, we have created a few packages, which are to be used inside the SysML models against which fault trees are to be generated. These packages are:

- DGBMetadata: It contains a package DFTElements with following sub-packages and elements:

- DFTGates package: It defines all gates that are used to construct fault trees.

```
package DFTGates {

    metadata def AND;
    metadata def OR;
    metadata def VOT {
        /* if any k-out-of-n components fail (input events),
        the system will fail (output event) n number of input events */
        attribute k : Number;
    }
    metadata def SPARE;
    metadata def PAND;
    metadata def POR;
    metadata def FDEP {
        /* The failure of the trigger_element renders the children of FDEP failed
        as per the value of probability attribute. The default value of probability
        is 1. */
        occurrence trigger_element;
        attribute probability: Real;
    }
    metadata def FSEQ;
    metadata def MUTEX;
}
```

- DFTBEs package: It defines all basic elements that may be used in fault trees.

```
package DFTBEs{
    abstract metadata def BE{
        /* The value of dormancy attribute is only relevant if the BE is a
        child of a spare spare gate. The default value of dormancy is 1. */
        attribute dormancy:Real;
    }
    abstract metadata def BE_CONSTANT_DISTRIBUTION specializes BE {
        attribute prob:Real;
    }
    abstract metadata def BE_EXPONENTIAL_DISTRIBUTION specializes BE {
        attribute rate:Real;
    }
    abstract metadata def BE_ERLANG_DISTRIBUTION specializes BE {
        attribute rate:Real;
        attribute phases:Real;
    }
    abstract metadata def BE_NORMAL_DISTRIBUTION specializes BE {
        attribute mean:Real;
        attribute stddev:Real;
    }
    abstract metadata def BE_WEIBULL_DISTRIBUTION specializes BE {
        attribute rate:Real;
        attribute shape:Real;
    }
}
```

- TOP_LEVEL metadata: It is used to annotate an element of a fault tree as a top-level element. More than one element can be annotated as top-level elements. This helps generate multiple fault trees (for different scenarios) collectively that may share Gates and BEs.

- FailureModes: It defines all failure modes that may be used to annotate elements of SysML models with safety information. At the moment we allow failure modes to be modeled with following failure distributions:
  - Exponential distribution
  - Erlang distribution
  - Weibull distribution
  - Log-normal distribution, and
  - Constant distribution

  Moreover, within this package we allow to define model constants as (DFTParameters) enumerations. These constants can be used to define failure rates, probabilities, shares etc. of failure modes.

```
package FailureModes {

    import DGBMetadata::DFTElements::*;
    /* DFTParameters enumiration defines constants used to annotate
    failure rates/probabilites/shares/etc. of BEs in fault trees. */
    enum def DFTParameters :> Real {
        FIT1 = 0.000000001;
        FIT2 = 0.000000002;
        FIT3 = 0.000000003;
        FIT4 = 0.000000004;
        prob = 0.2;
        param1 = 10.2;
        param2 = 1.1;

    }

    metadata def FIT1 specializes BE_EXPONENTIAL_DISTRIBUTION{
            attribute redefines rate = DFTParameters::FIT1;
    }
    metadata def FIT2 specializes BE_EXPONENTIAL_DISTRIBUTION{
            attribute redefines rate = DFTParameters::FIT2;
    }
    metadata def FM1 specializes BE_CONSTANT_DISTRIBUTION{
            attribute redefines prob = DFTParameters::prob;
    }
    metadata def FIT3 specializes BE_ERLANG_DISTRIBUTION{
            attribute redefines rate = DFTParameters::FIT3;
            attribute redefines phases = 2;
    }
    metadata def FM_4 specializes BE_NORMAL_DISTRIBUTION{
            attribute redefines mean = DFTParameters::param1;
            attribute redefines stddev = DFTParameters::param2;
    }
    metadata def FIT4 specializes BE_WEIBULL_DISTRIBUTION{
            attribute redefines rate = DFTParameters::FIT4;
            attribute redefines shape = 3;
     }
}
```

Laptop Example.

The following example explains how elements within the SysML model can be annotated to generate fault trees out of them.

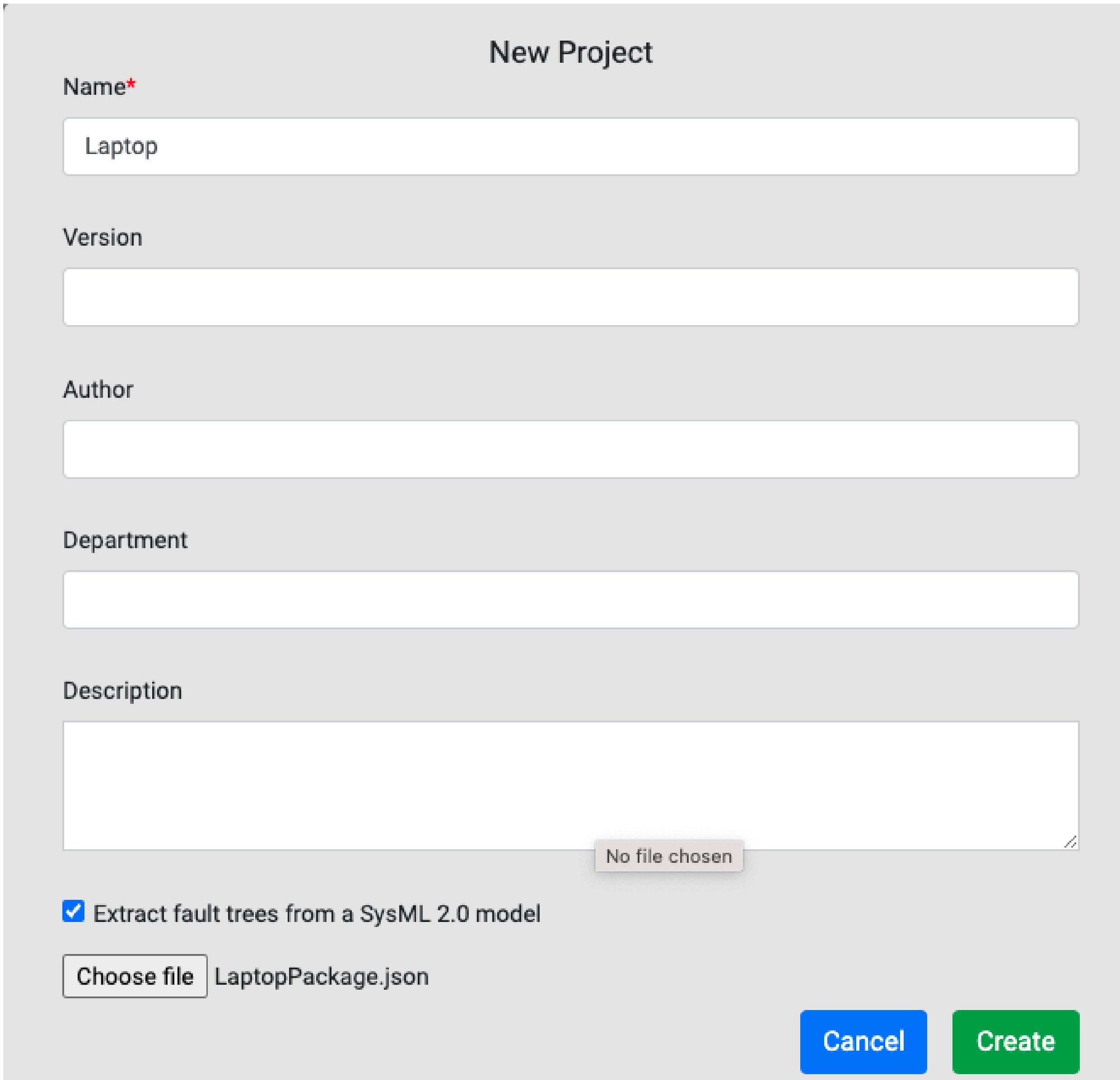
```
package LaptopPackage {
    import FailureModes::*;
    part Laptop {
        part CPU1 {
            metadata Failure:FIT2;
        }
        part CPU2 {
            metadata Failure:FIT1;
        }
        part cooling {
            metadata Failure:FIT1;
        }
        part plug {
            metadata Failure:FIT2;
        }
        part battery {
            metadata Failure:FIT2;
        }
        part switch {
            metadata HWF:FIT1;
        }
        metadata power:SPARE about plug::Failure, battery::Failure;
        metadata processor:AND about CPU1::Failure, CPU2::Failure;
        metadata laptop:OR about power, processor;
        metadata Dep:FDEP about CPU1::Failure, CPU2::Failure {
            trigger_element = cooling::Failure;
        }
        metadata TLE1:TOP_LEVEL about laptop;
        metadata TLE2:TOP_LEVEL about power;
    }

}
```

After compilation of the SysML model annotated with safety information using our packages in jupyter notebook, run the following command to export the package in JSON format. **Currently we support the latest version – v0.33.0 – of SysML 2.0.**

%export <package_name>

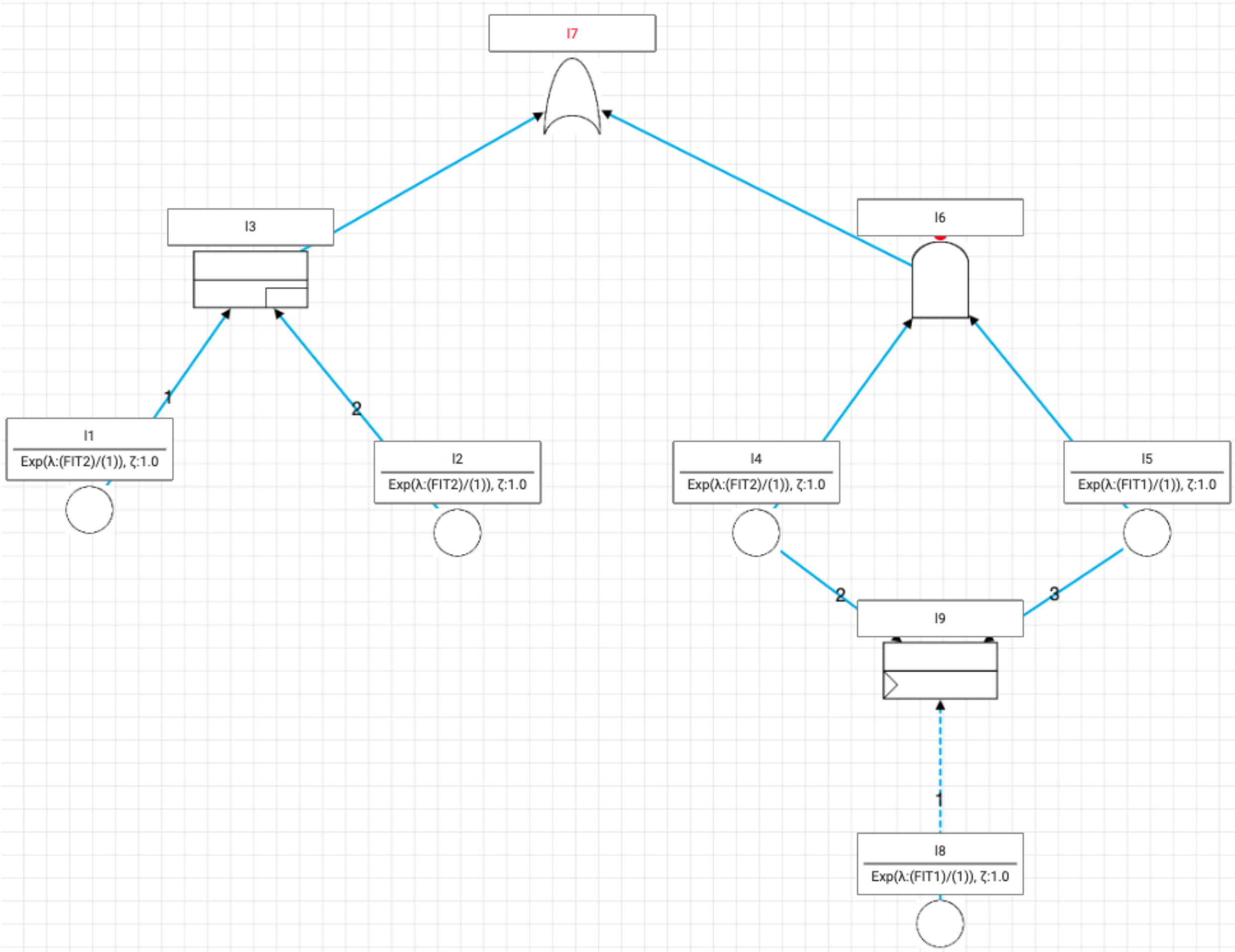
After downloading, it can be uploaded inside the SAFEST tool while creating a new project.

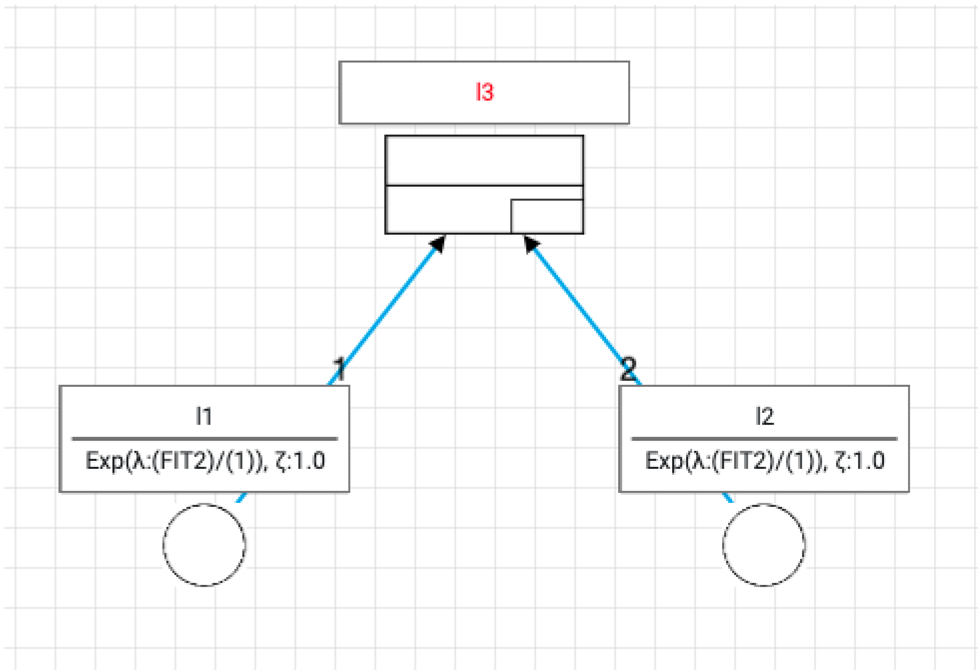


Click the “Create” button to generate failure models against metadata elements annotated as top-level elements inside the SysML model.

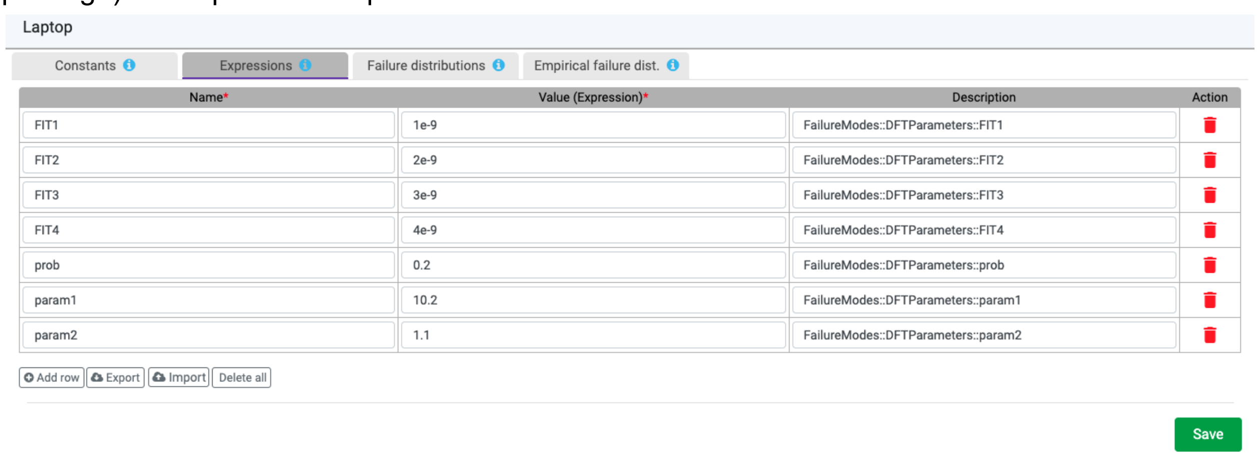
Failure Models

Model name	Fault tree	Version	Author	Time bound (Life cycle)	Description	Default	Actions
LaptopPackage_Laptop_laptop	Dynamic			100		●	
LaptopPackage_Laptop_power	Dynamic			100		○	





Moreover, all constants (DFTParameters enumerations inside the SysML FailureModes package) are imported as a parameter set.



Name*	Value (Expression)*	Description	Action
FIT1	1e-9	FailureModes::DFTParameters::FIT1	
FIT2	2e-9	FailureModes::DFTParameters::FIT2	
FIT3	3e-9	FailureModes::DFTParameters::FIT3	
FIT4	4e-9	FailureModes::DFTParameters::FIT4	
prob	0.2	FailureModes::DFTParameters::prob	
param1	10.2	FailureModes::DFTParameters::param1	
param2	1.1	FailureModes::DFTParameters::param2	

Grammer:

Expressions detail

- Identifier (id):
  - started with a capital and small letters(a-z A-Z) followed by letters, or numbers (a-z A-Z 0-9)
- Mode Name (mode):
  - started with a capital, and small letters(a-z A-Z) followed by letters, or numbers (a-z A-Z 0-9)
- Numeric constant (nc):

    - Simple,decimals and exponential i.e 123, 123.123, 123e+1, 123e-1, 123e1, 123.123e+1, 123.123e-1, 123.123e1, 123.123E1, 0.12

Real expressions

- Keywords:
    - [pow, log]
- Context Free Grammar:
    - RE → E | + nc | -nc
    - E → E OP E | nc | id | (RE) | pow(RE,RE) | log(RE,RE)
    - OP → + | - | * | /

Boolean Logic

- Keywords:
    - No keywords
- Context Free Grammar:
    - E → E OP E | mode | (E) | !mode | !(E)
    - OP → | | &

Continuous Stochastic Logic

- Keywords:
    - [true,false,Pmin,Pmax,Smin,Smax,Tmin,Tmax,LRAmin,LRAmax,P,R,T,S,LRA,min,max,G,U,F,W,C,I,failed]
- Context Free Grammar:
    - PROP → P OP2 TYPE [PathFormula] | T OP2 TYPE [RewardFormula] | LongRun OP2 TYPE [StateFormula]
    - TYPE → =? | OP3 E
    - LongRun → LRA | S
    - PathFormula → OP4 BoundedExpression StateFormula | StateFormula OP5 BoundedExpression StateFormula
    - BoundedExpression → ^ { Bound } | { Bound } | Bound | null
    - Bound → [E,E] | OP3 TIME
    - TIME → (E) | nc
    - RewardFormula → I = E | C <= E | F StateFormula | LongRun
    - StateFormula -> StateFormula OP6 StateFormula | P OP2 OP3 E [PathFormula] | LongRun OP2 OP3 E [StateFormula] | mode | failed | (StateFormula) | true | !StateFormula
    - OP → => | & | | | = | != | <= | >= | > | < | + | - | * | / | %
    - OP1 → + | -
    - OP2 → min | max | null
    - OP3 → <= | >= | > | <
    - OP4 → G | F
    - OP5 → U | W | R
    - OP6 → | | &

- E → E OP E | id | nc | (E) | !(E) | (OP1 nc)