HiP-HOPS
Automated Fault Tree, FMEA and Optimisation Tool

User Manual

Version 2.5
(July 2013)

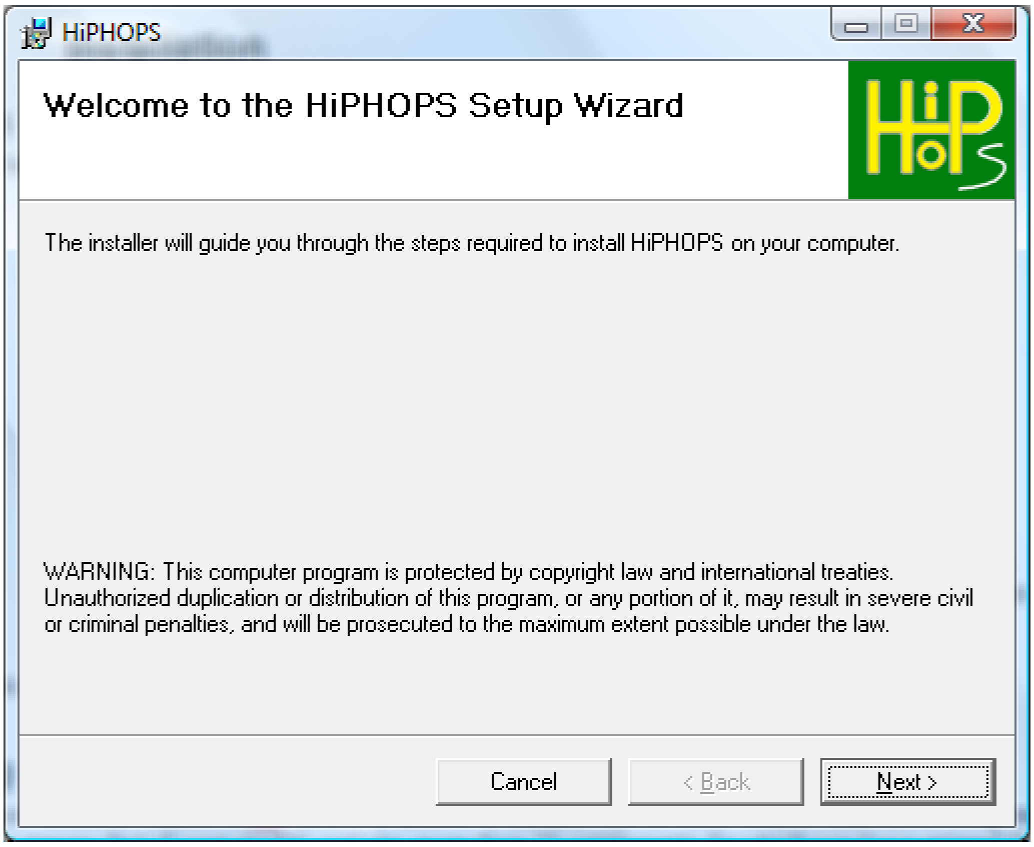
CONTENTS

1 INSTALLATION

1.1 Installing

Installing HiP-HOPS is very simple:

1. Run the HiP-HOPS installer:

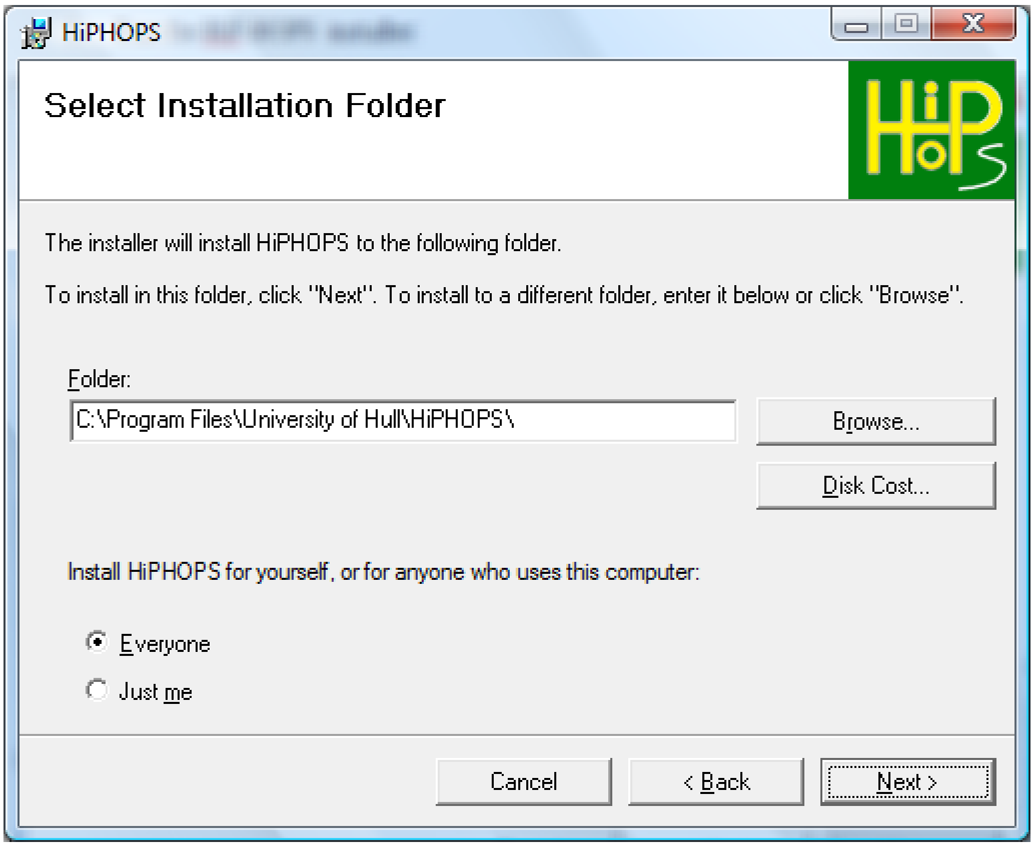


Welcome to the HiPHOPS Setup Wizard

The installer will guide you through the steps required to install HiPHOPS on your computer.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

2. Select a directory to install to and choose which profiles to make HiP-HOPS visible for;



HiPHOPS

Select Installation Folder

The installer will install HiPHOPS to the following folder.

To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".

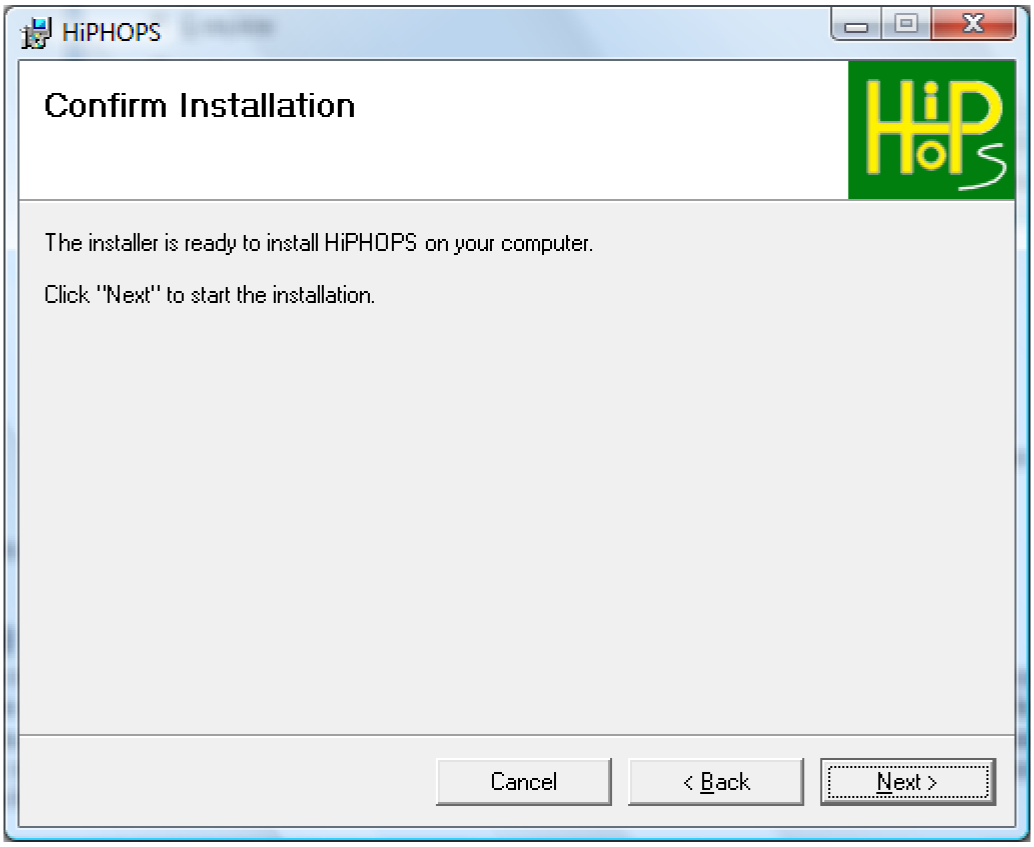
Folder: C:\Program Files\University of Hull\HiPHOPS\

Install HiPHOPS for yourself, or for anyone who uses this computer:

[✓ Everyone] Just me

3. Click "Next";

4. Click “Next” again to confirm;

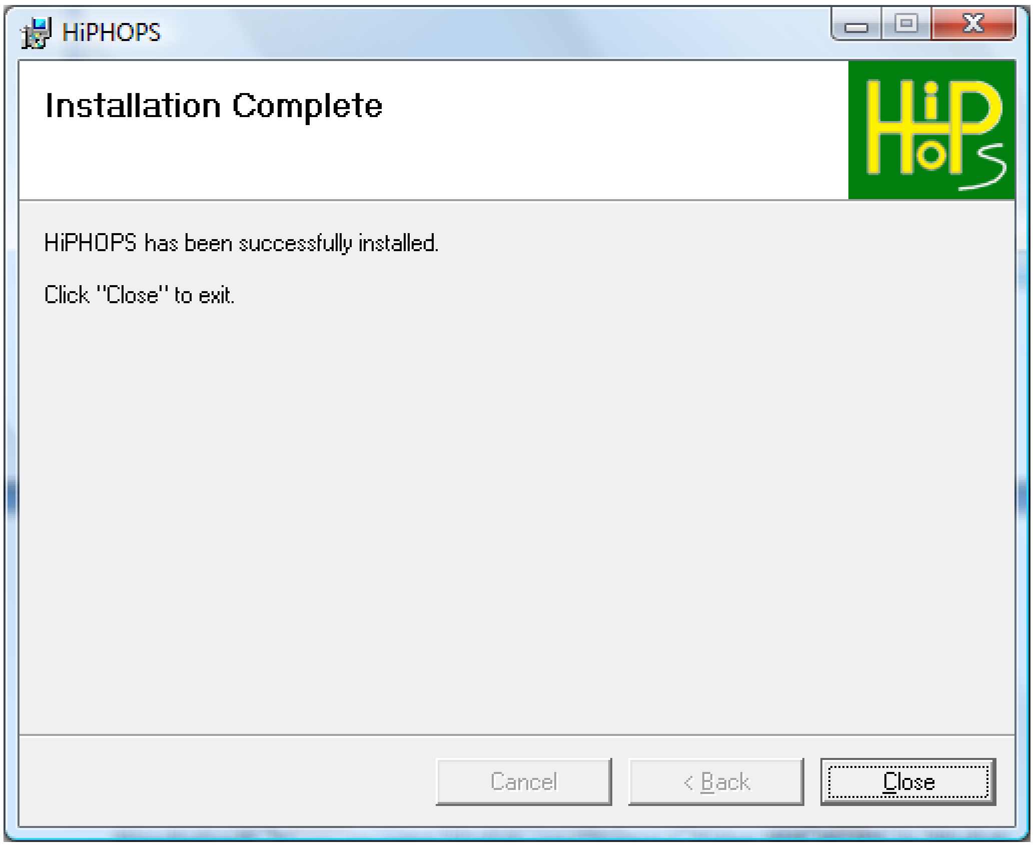

HiPHOPS

Confirm Installation

The installer is ready to install HiPHOPS on your computer.

Click "Next" to start the installation.

5. Close the installer when finished;


HiPHOPS

Installation Complete

HiPHOPS has been successfully installed.

Click "Close" to exit.

HiP-HOPS can be uninstalled by using the Add/Remove Programs (Windows XP) or Programs and Features (Vista/Windows 7) tools in the Control Panel.

1.2 Evaluation Version

The evaluation version of HiP-HOPS is limited to models with no more than 10 annotated components. This means that if your model contains more than 10 components for which you have entered failure data, i.e. basic events and output deviations, then the evaluation version will not run. If you are running the evaluation version, it will say so in the command window when you run it.

1.3 Example mdl file

The installation process creates an example Matlab model file example.mdl. It is installed in the location specified by the installer with the path: University of Hull\HiPHOPS\example.mdl.

This example model contains a small standby recovery example that is used for illustration throughout this manual.

There is also an example model for optimisation. This is a fuel oil service system for the main engine of a cargo ship. The main system is contained in optimisationExample.mdl and alternative subsystems for the components in this model are held in additional files (optimisationExampleFlow2.mdl, optimisationExampleFlow3.mdl, etc.)

2 ALL ABOUT HIP-HOPS

2.1 Why use HiP-HOPS

Fault Tree Analysis (FTA) and Failure Modes & Effects Analysis (FMEA) are classical system analysis techniques used in reliability engineering. They are methods by which we can discover information about the potential faults in a system that we can then use to correct those faults. Both are widely used in the automotive, aerospace, nuclear, and other safety critical industries.

FTA is a deductive technique, which means it works from the top down. This is done by assuming a system failure has occurred and working backwards to try to determine what possible combinations of events might have caused it; the system failure then becomes the *top event* of the fault tree, and the individual component failures form the leaf nodes (or *basic events*), and are combined through logical gates such as OR and AND. The fault tree can then be analysed either qualitatively, to determine the smallest combinations of basic events needed to cause the system failure, or quantitatively, to obtain the probability of the top event occurring.

FMEA, by contrast, is an inductive technique, and works from the bottom up. It involves proposing a certain event or condition, and then trying to assess the effects of that initial event on the rest of the system. The end result is a table of failures and their effects on the system, which provide the analyst with an overview of the possible faults.

Both techniques are useful and provide valuable information about systems, but both suffer from the same flaw: they are primarily manual techniques, and the process of performing these analyses can be laborious, especially for larger and more complex systems. In such cases, it is more likely that the analyst will make a mistake, or that the results once obtained are too numerous to interpret efficiently.

This problem means that both FTA and FMEA tend to be performed only once, either after the system has been designed, in order to check its reliability, or after the system has been put into operation and has failed, in order to find out what went wrong. This is unfortunate, because both FTA and FMEA are potentially very useful when they are integrated into the design process itself, so that a system can be designed with safety and reliability in mind. By using these system analysis techniques as part of an iterative design process, it is possible to identify and remedy potential flaws and faults much earlier, thereby saving both time and effort and producing a more reliable product.

However, before FTA and FMEA can be incorporated into the design process in this way, they need to be automated in some fashion, so that they can be carried out much more quickly and efficiently, and thus maximising their contribution to the design. The HiP-HOPS Fault Tree Synthesis (FTS) tool is intended to achieve such a goal. By including reliability annotations as part of the system model, HiP-HOPS can examine the model and automatically construct and analyse both fault trees and FMEAs. The result is a semi-automated process which takes much of the burden off the system designer and speeds up the analysis considerably, allowing the designer to quickly identify weak points in the model and take steps to remedy them.

2.2 Features of the tool

HiP-HOPS (Hierarchically Performed Hazard Origin & Propagation Studies) works with the modelling package used by the designer to obtain information about the reliability of the system being modelled. By annotating the model with data describing how individual components can fail, HiP-HOPS can then take that data and produce a series of fault trees from it, and from those trees it can generate a wealth of reliability information presented in the form of an FMEA table.

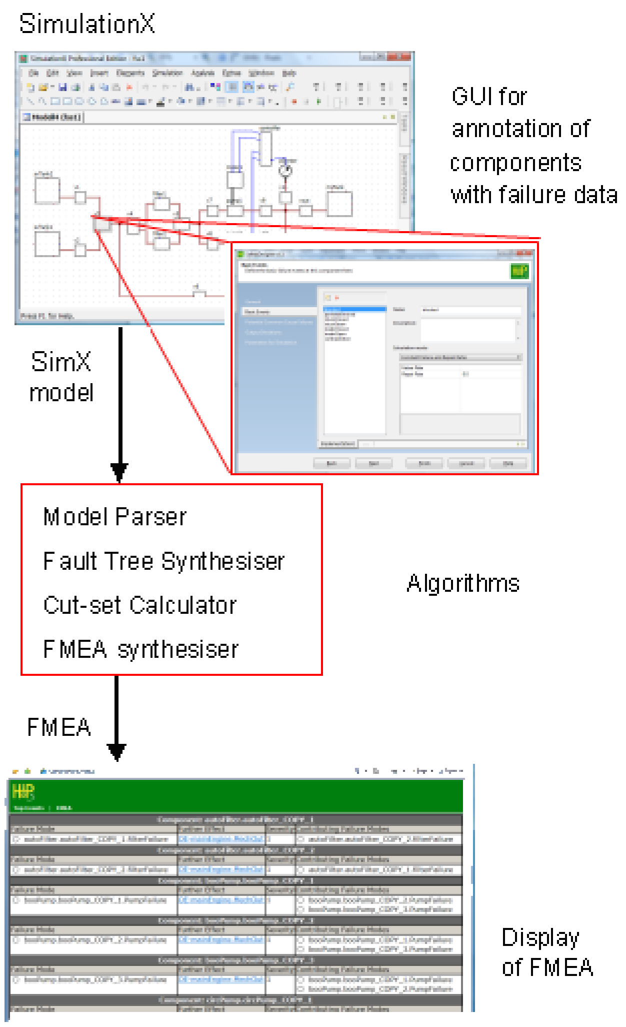


Figure 1: The HiP-HOPS Process

HiP-HOPS has been integrated with modelling package Matlab, allowing greater feedback to the user and a wider range of functionality.

The tool allows the user to load a model of a system to be loaded and parsed. Once this internal representation of the model has been loaded, the tool synthesises fault trees for every system failure in the model, and combines them to create the FMEA.

During this process, a lot of other useful analysis is carried out; the *minimal cut sets* are obtained, which are the minimum combinations of basic events required to cause the top event, and the *unavailability*, or probability, of the top event is also calculated.

These results are integrated into the resultant FMEA tables, which are presented in the form of hyperlinked web pages. This format allows the designer to locate a specific failure and click on it to find the effects it has on the rest of the system.

This whole process is illustrated in Figure 1, from the annotation of the model with the failure data at the top, through the fault tree synthesis and analysis, and culminating in the FMEA. Best of all, it does this in a matter of seconds or minutes, not days.

The tool is also capable of outputting the resultant fault trees in a format that can be read by the popular Fault Tree + fault tree analysis tool, making it easier to view the trees.

Thus, in summary, HiP-HOPS has the following capabilities:

- Works with Matlab to analyse annotated model files generated by that program;
- Can parse those models and synthesise fault trees from them;
- Can quickly generate minimal cut sets;
- Calculates unavailability for the top events;

- Generates FMEA in the form of hyperlinked web pages and produces Fault Tree + files.
- Can perform multi-objective optimisation of the models to produce a set of trade-off solutions to choice of alternative components and best sites for redundancy allocation.

2.3 How the tool works

The features of the tool can be divided into two main parts. The first part is safety analysis which performs a single pass analysis of the annotated model. The second is optimisation which uses a multi-objective genetic algorithm to automatically improve the dependability characteristics of the model. The optimisation tool makes use of the analysis functions to evaluate the automatically generated alternative designs. The two parts will be discussed now in greater detail.

2.3.1 Safety Analysis

First the process of providing a single pass analysis is described.

2.3.1.1 Overview

HiP-HOPS analysis consists of three main phases. The first phase consists of annotating the system model with the failure data needed to produce the fault trees and perform the analysis. This phase must be integrated into the modelling tool used to design the system model, since they are so closely related. In this case, the failure data is entered via a graphical user interface within Matlab, as explained in chapter 3.

The second phase of HiP-HOPS analysis is the fault tree synthesis process. In this process, the tool examines the system model and its failure data and combines it to create a series of fault trees. It works by taking the failures of system outputs and working backwards through the model to determine which components caused those failures. These failures are then joined together using the appropriate logical operators to construct fault trees with the failures at the system outputs as the top events and the root causes as basic events.

The third and final phase of HiP-HOPS takes the newly constructed fault trees and analyses them. The result is an FMEA, which is a combination of all the information stored in the fault trees and presented in the form of a table listing the effects of each component failure on the rest of the system. As part of this process, more analysis is carried out on the fault trees, both *qualitative* (logical) and *quantitative* (numerical). This provides both the minimal cut sets of the fault tree and the unavailability of the top event.

2.3.1.2 Annotation Phase

Before any fault trees can be generated, the tool needs to know how the various components of the system are interconnected, and how each one can fail. The structural data is provided by the model itself, which shows the basic topology of the system and the connections between the various components and subsystems. The models can also be hierarchically arranged, so that systems can be decomposed into subsystems which each have their own components. The types of models which HiP-HOPS can be applied to are varied: fluidic systems, electrical or electronic systems, mechanical systems, and even more conceptual data flow based models are all suitable, as long as they can be augmented with failure data.

The failure data is what needs to be entered separately; each component or subsystem needs its own *local failure data*, which describes what can go wrong with that component and how it responds to failures elsewhere in the system. This is achieved by annotating the model with a set of failure expressions showing how deviations in the component outputs (*output deviations*) can be caused either by internal failures of that component (which become basic events) or corresponding deviations in the component's inputs. Such deviations include unexpected omission of output or unintended commission of output, or more subtle failures such as incorrect output values or the output being too early or late. This logical information explains all possible deviations of all outputs of a component, and so provides a description of how that component fails and reacts to failures elsewhere. Once done, the data can then be stored in a library, so that other components of the same type can use the same failure data. This avoids the designer having to enter the same information many times.

At the same time, numerical data can be entered for the component, detailing the probability of internal failures occurring and the severity of output deviations. This data will then be used during the analysis phase to arrive at a figure for the unavailability of each top event. Other information can be added at this stage to facilitate more advanced analysis, including the option to enter information for different implementations of a component to allow an optimisation algorithm to determine the most efficient choice of component to use, and in the future, also the addition of temporal data to describe more complicated failure behaviour.

Because all of this data needs to be entered for each component, it is necessary for this to be done as part of the modelling tool itself, where the parts of the system are readily visible. This requires a user interface to be created to allow for the data entry that is separate to the main part of HiP-HOPS itself, and instead integrated with the modelling tool directly.

2.3.1.3 Synthesis Phase

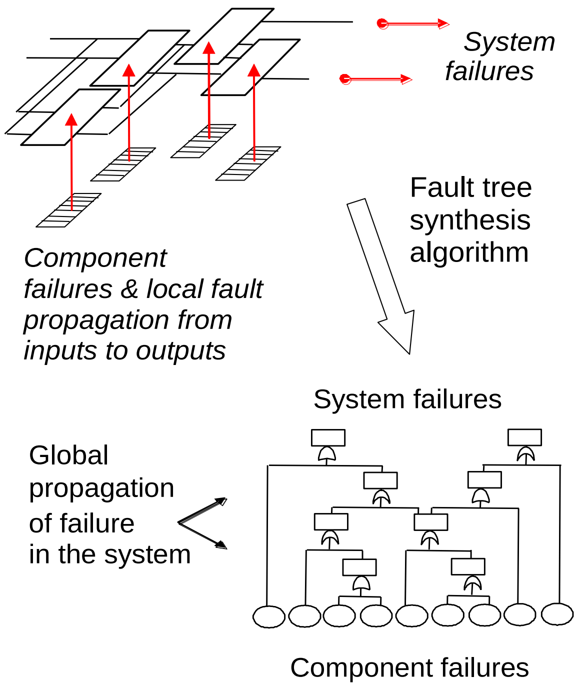

System failures
Fault tree synthesis algorithm
Component failures & local fault propagation from inputs to outputs
System failures
Global propagation of failure in the system
Component failures

Figure 2: The synthesis of fault trees from the system model

Once the local failure data has been entered into the system model, the model can then be given to HiP-HOPS proper for synthesis to take place. This phase functions by examining how local failures of components propagate through the model and cause failures at the outputs of the system. It is therefore necessary to be able to identify which parts of the model function as outputs of the system, so that the failures of these parts become top level failures of the system as a whole. In the case of an engine, for example, then the system output could be the application of motive power. A failure to provide this would be a failure of the system output, and would be treated as a starting point for the synthesis. In addition, unexpected output (i.e. the engine moving when it should not) could also be a system failure.

For each of these system outputs, HiP-HOPS generates a local fault tree describing what can lead to that output. The tool then examines the inputs to that component, and travels back to see what is connected to it. It then makes local fault trees for those fault trees, and so on until there are no connected components remaining. It then goes back through and combines the local fault trees into a single fault tree for each possible failure of the system output, which shows how malfunctions of that output are caused by combinations of malfunctions or other failures of components elsewhere in the system. For our engine, this means we would have one fault tree for “no engine power on demand” and one fault tree for “unexpected engine power”.

In order to introduce some of the major concepts, a simple example of a standby recovery system, shown in
Figure **3**, will be used.

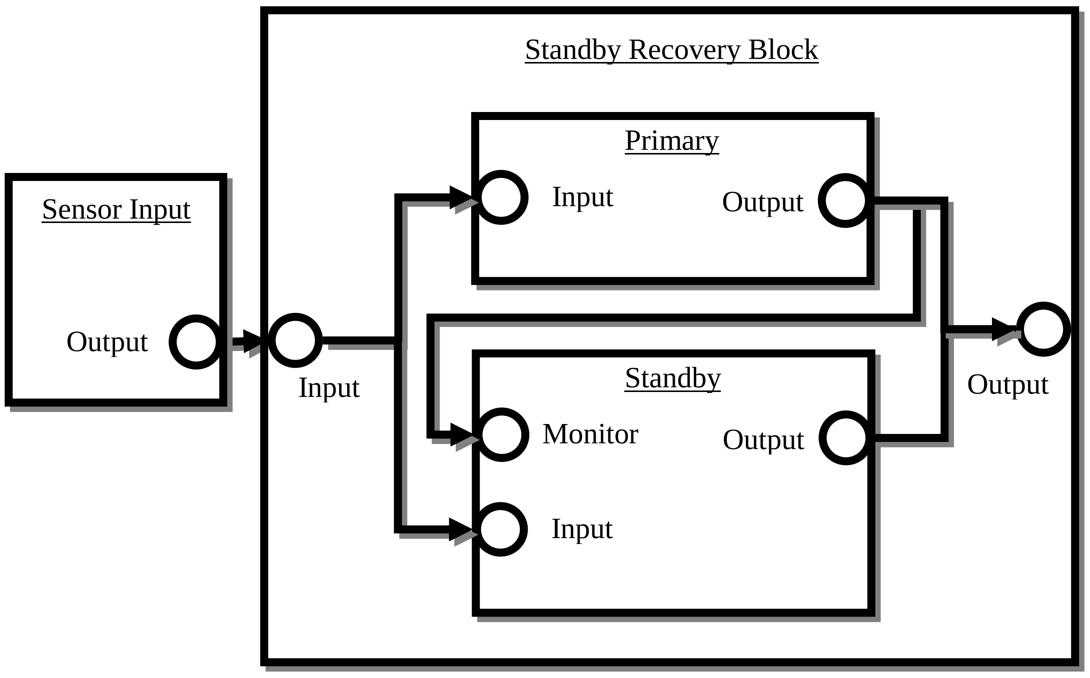


Figure 3 - A simple example of a standby recovery system

It is composed of one input component, ‘SensorInput’, and one subsystem, ‘Standby Recovery Block’, which is composed itself of two sub-components, ‘Primary’, and ‘Standby’. ‘Primary’ is the main subcomponent of the standby-recovery system and processes the input from the ‘SensorInput’. The 'Standby' component monitors the output from 'Primary' and is designed to take over operation if it detects a failure of the ‘Primary’.

Table 1 shows the failure modes of the components in the example system.

Table 1- Failure modes for standby recovery system

Component	Failure Modes	Probability
Standby Recovery Block	ElectroMagneticInterferance	0.001
Primary	InternalFailure	0.03
Standby	InternalFailure	0.02
SensorInput	InternalFailure	0.05

Table 2 shows the failure annotations for the components in the example system.

Table 2- Failure data for standby recovery system

Component	Output deviations	Failure expressions
Standby Recovery Block	Omission - Output	Omission – Primary.Output AND Omission – Standby.Output OR ElectoMagneticInterference
Primary	Omission – Output	Omission – Input OR Failure
Standby	Omission – Output	Omission – Monitor AND (Omission – Input OR Failure)
SensorInput	Omission - Output	Failure

Table 2 defines that the omission of output of the SensorInput component is caused only by its internal failure.
The ‘Monitor’ port detects omissions of output of the ‘Primary’ component before activating the ‘Standby’ component. If the ‘Primary’ component is functioning then the ‘Standby’ cannot fail (or its failure is irrelevant) as it is inactive. If the ‘Standby’ component is thus activated then an output omission is caused by either an internal failure or the propagation of input omission.
An omission of ‘Primary’ output is caused by an internal failure or the propagation of an omission of input.

At the top level, the system output of the ‘Standby Recovery Block’ has an output deviation of type ‘Omission’ that is caused by either the failure mode ‘electromagnetic interference’, or by the conjunction of omissions occurring at the outputs of both the primary and standby components.
The Boolean failure expressions in the component annotations each describe a Component Fault Tree (CFT) that describes propagation of failure through it. However, such a fault tree is incomplete because its leaf nodes will not all be failure modes – some may be input

deviations, which are failures originating in other components. Similarly the top node is an output deviation that may be relevant in further components in the system.

These CFTs for the example are shown in Figure 4 (for the 'Standby Recovery Block'), Figure 5 (for the 'Primary' component), Figure 6 (for the 'Standby' component), and Figure 7 (for the 'SensorInput' component). In these diagrams the circles with an arrow denote an input deviation where the arrow is entering the circle and an output deviation where the arrow is leaving the circle.

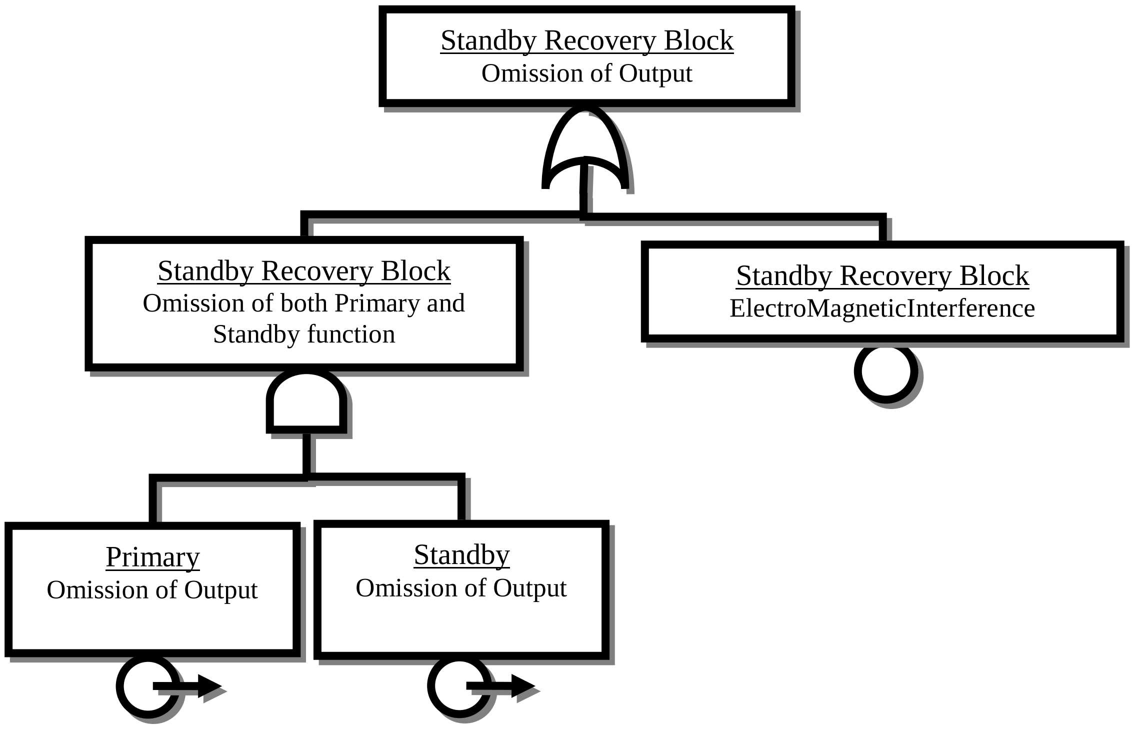


Figure 4 CFT described by annotation for standby recovery block subsystem

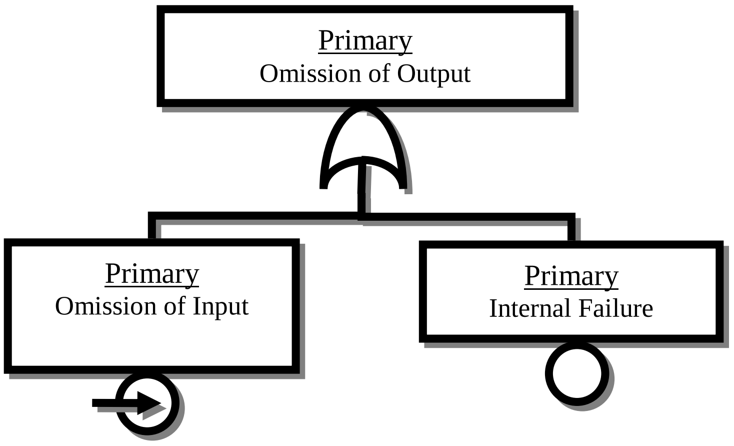


Figure 5 - CFT described by annotation for primary component

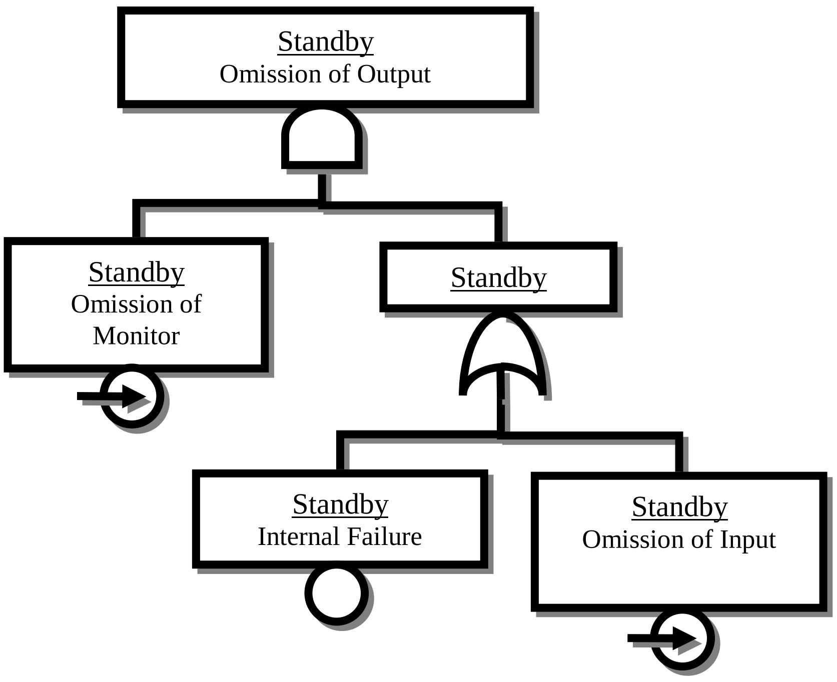


Figure 6 - CFT described by annotation for standby component

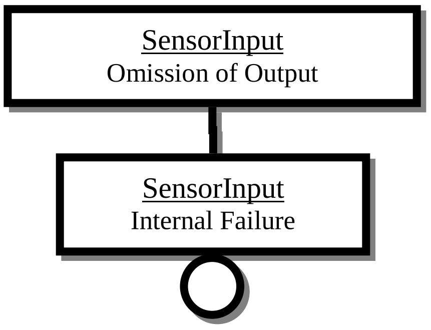


Figure 7 - CFT described by annotation for SensorInput component

Once the components of the model have been fully annotated, the manual phase of HiP-HOPS is complete and the remaining stages are fully automatic.

2.3.1.4 Synthesis Phase

During the annotation phase the designer manually adds failure expressions to each component. As explained before, these expressions represent CFTs that describe the propagation, generation, and transformation of failure between the inputs of the component and its outputs.
As the components are linked through their ports to other components, failures can be propagated between components.

The synthesis phase begins with deviations of the system outputs. The algorithm locates the CFT for each output deviation and traverses the tree until it locates a terminal input deviation. The input port that is associated with that input deviation is then selected.

The algorithm then follows the connections from the selected component port to the output ports of the connected components. The output deviation matching the failure class of the connected input deviation is then selected. Its CFT is joined to the input deviation from the connected component.

This cycle of traversing CFTs and connections is repeated until there are no more unconnected input deviations and the system fault trees are complete.

In the standby recovery example in

Figure **3**, the system output is the omission of output of the ‘Standby Recovery Block’. This is the top node of a CFT in the ‘Standby Recovery Block’ and marks the start of the synthesis for this example.

This CFT has two input deviations at its leaves, ‘omission of Primary output’ and ‘omission of Standby output’, and so the algorithm finds the corresponding components where it discovers further CFTs. The top nodes of these are added as child branches to the input deviation leaf nodes and the process is repeated, each time connecting output deviation CFTs to input deviation leaf nodes.

The omission of ‘Primary’ output is partly caused by an omission of its input, so the connection at the input is followed to the ‘SensorInput’ component where the output deviation that exists there is connected to the growing system fault tree. As the deviation of ‘SensorInput’ output is only caused by an internal failure, the propagation of that branch is terminated.

The ‘omission of Standby input’ branch is treated in the same way and thus a complete system fault tree, describing the propagation of failure throughout the whole model, is synthesised from the failure expressions. The result of synthesis of the example model is shown in Figure 8. It can be seen from this diagram how the system fault tree is composed of the mini fault trees shown in Figure 4 to Figure 7. The output deviations that are the top nodes of the component fault trees are shaded.

In the next HiP-HOPS phase the fault tree is analysed to extract quantitative and qualitative information.

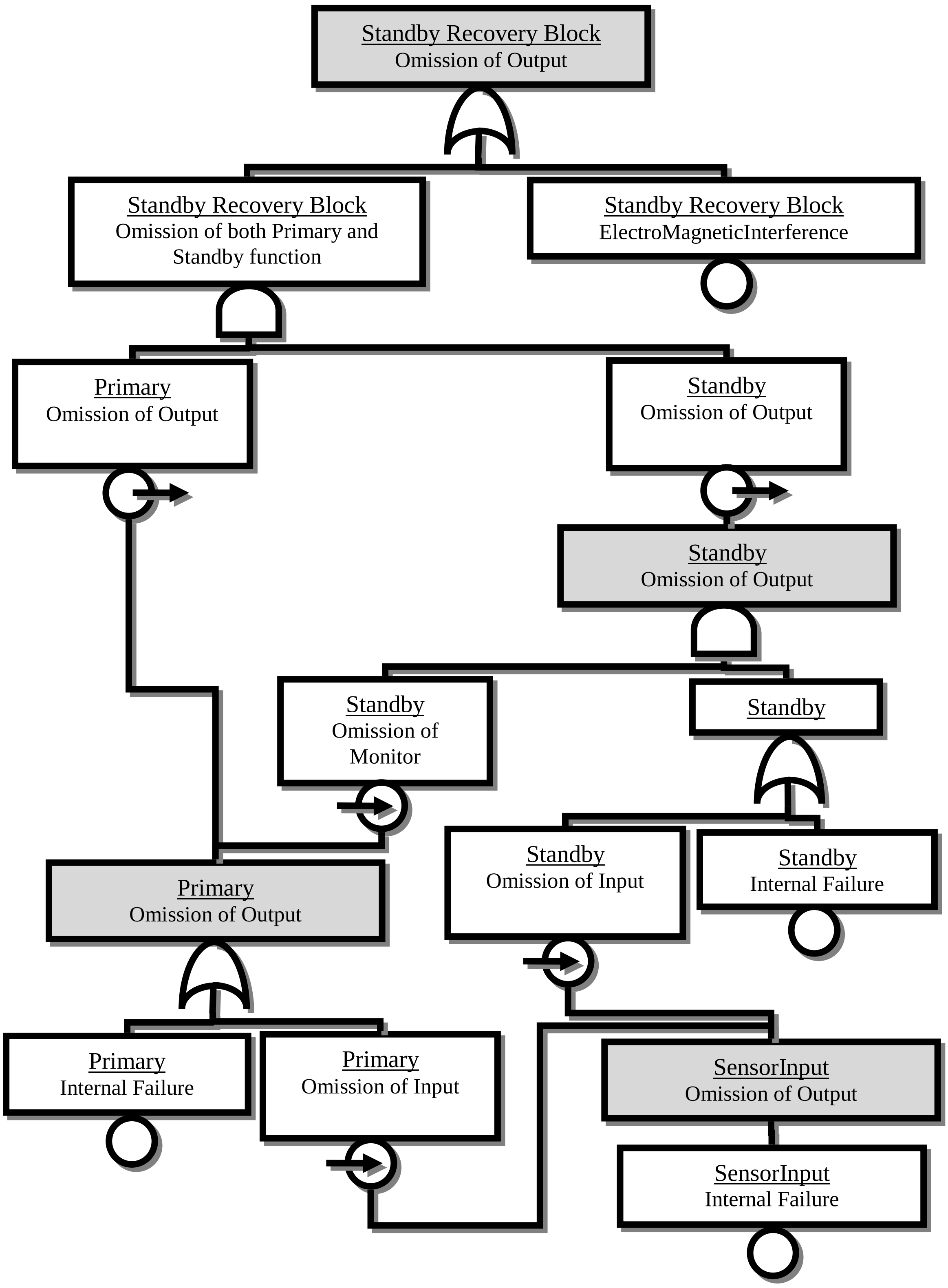


Figure 8 - Fault tree synthesised from standby recovery example.

2.3.1.5 Analysis phase

The result of the synthesis process is a set of one or more interconnected fault trees and therefore the next stage of the HiP-HOPS technique is to analyse those fault trees using FTA. The fault trees represent the propagation of failure logic through the system, but they can often be large and complex. By reducing the fault trees to their minimal cut sets we retain the relationship between the basic events and the top level system event but strip out the intermediate propagation paths.

The primary cut set generating algorithm used by HiP-HOPS is MICSUP (MInimal Cut Sets UPwards) [1]. MICSUP, as the name suggests, is a bottom-up algorithm for obtaining minimal cut sets from a fault tree.

One advantage of MICSUP is that the cut sets can be stored in the intermediate nodes as they are generated and minimised as the algorithm returns from the basic events. This ability to reuse the results in shared branches without needing to reanalyse saves computational effort.

A second advantage is that it is easy to use cut set order pruning: when a cut set exceeds the maximum size limit, it is simply discarded at that point and cannot contribute to cut sets further up the tree.

The main computational expense when minimising the cut sets is the redundancy checking. Several methods of increasing the performance of this process, including modularisation, fault tree contraction, and use of cut set cataloguing, are discussed in a later section on performance increases.

The following Boolean laws can be applied to obtain minimal cut sets:

The law of absorption: E1 + ~~E1.E2~~ = E1
The cut set containing E1.E2 was removed as the action of E1 alone is sufficient to cause the top event and is therefore in its minimal form.
The laws of idempotence: E1.~~E1~~ = E1 and E1 + ~~E1~~ = E1
The former removes repeated events within cut sets and the latter removes repeated cut sets.
In order to keep the number of checks to a minimum the cut sets are checked for redundancy as they are created so that redundant combinations are quickly identified and removed. This ensures that they cannot affect or be combined with more cut sets later in the traversal of the fault tree.
Once the minimum cut sets have been identified, they can subsequently be used for quantitative analysis to calculate the system unavailability Q_s (where basic events have quantitative data) using the approximate Esary Prochan method:

$$Q_S = 1 - \prod_{i=1}^{n}(1 - Q_{CS_i})$$

(Where n is the number of independent cut sets and Q_{CS} is the unavailability of the cut set i).
In addition to the quantitative analysis that can be performed on the minimal cut sets, a further qualitative stage can be applied to generate an FMEA. Figure 9 shows the inverse relationship between the diagnostic failure propagation information in the fault trees, where the component failure modes that cause a system failure can be determined, and the causative nature of the FMEA, where a basic event (or combination of several events) have an effect on the system level.

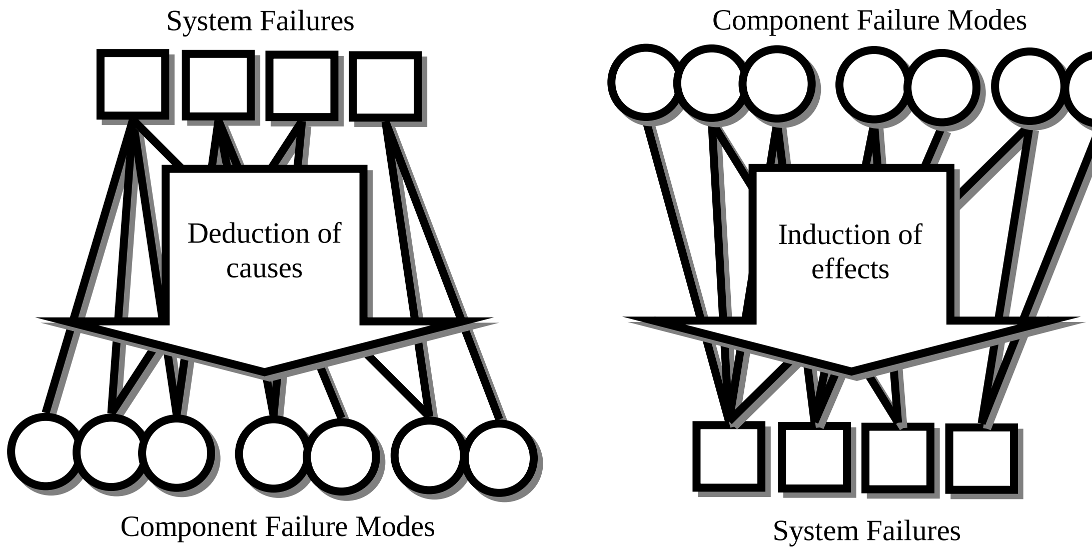
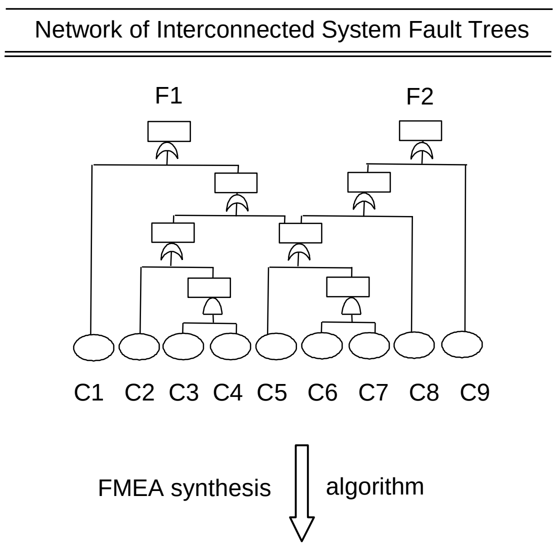


Figure 9 - Inverse relationship between fault trees (left) and FMEA (right)

The minimal cut sets contain the non-redundant propagation of failure in the fault tree and an algorithm is used to catalogue each component failure mode in each fault tree and note which system failures they cause and in combination with which other component failure modes. This information is the core of an FMEA.
The deductive nature of this process is important for safety analysis as it allows large combinations of basic events to be considered in the FMEA, unlike traditional manual methods that could only consider single points of failure or fault injection simulation methods that are similarly limited by combinatorial explosion.

The last step is to combine all of the data produced into an FMEA, which is a table that concisely illustrates the results. The FMEA shows the direct relationships between component failures and system failures, and so it is possible to see both how a failure for a given component affects everything else in the system and also how likely that failure is. However, a classical FMEA only shows the *direct effects* of single failure modes on the system, but because of the way this FMEA is generated from a series of fault trees, HiP-HOPS is not restricted in the same way, and the FMEAs produced also show what the *further effects* of a failure mode are; these are the effects that the failure has on the system when it occurs in conjunction with other failure modes. Figure 10 shows this concept.



Multiple Failure System FMEA

Component failure	Direct effects on the system	Effects caused in conjunction with (other events)
C1	F1	-
C2	F1	-
C3	-	F1 (C4)
C4	-	F1 (C3)
C5	F1,F2	-
C6	-	F1,F2 (C7)
C7	-	F1,F2 (C6)
C8	F2	-
C9	F2	-

Figure 10: The conversion of fault trees to FMEA

In Figure 10, F1 and F2 are system failures, and C1 – C9 are component failures. For C3, C4, C6 and C7, there are no direct effects on the system – that is, if only one of these components fail, nothing happens. However, they do have further effects; for example, C3 and C4 both occurring in conjunction will cause F1 to occur.

The FMEAs produced, then, show all of the effects on the system, either singly or in combination, of a particular component failure mode. This is especially useful because it allows the designer to identify failure modes that contribute to multiple system failures (e.g. C5 in the example of Figure 10). These common cause failures represent especially vulnerable points in the system, and are prime candidates for redundancy or extra reliable components.

2.3.2 Optimisation

The safety analysis tool HiP-HOPS and a multi-objective optimisation algorithm have been combined to generate un-dominated trade-off solutions that meet dependability criteria. The principle could apply to any optimisation algorithm but a promising example is the NSGA-II algorithm [2], a multi-objective genetic algorithm that uses a Pareto-based selection method to encourage a wide, even spread of trade-off results.

Figure 11 shows a fuel oil service system for a cargo ship. The example will be expanded and used to illustrate concepts throughout this section.

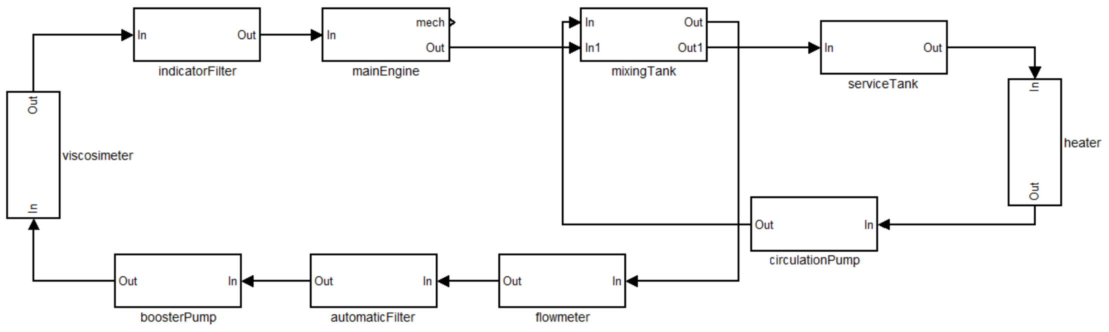


Figure 11: Fuel oil service system for a cargo ship.

When the fuel oil system fails, there is a loss of engine propulsion (OmissionEnergy-mainEngine.mech) that can lead to the ship becoming grounded as a result of drifting.

The following tables contain the implementation failure data for the components in the example in Figure 11. The main engine provides the system output (an omission of energy at the mechanical output of the engine) which is caused by an omission of flow of oil at the input. It has no internal failures.

mainEngine		
Output Deviation	**Description**	**Failure logic (Propagation)**
OmissionEnergy-mech (System output)	Omission of energy at the mechanical output of the mainEngine caused by an omission of flow of oil at the input	OmissionFlow-In

The other components in the system (indicator filter, viscosimeter, pre-heater, circulation pump, mixing tank, flow meter, automatic filter, booster pump, and service tank) each contain 3 alternative subsystems that define different levels of parallel redundancy (Figure 12 to Figure 14). In each case the propagation of the omission of flow of oil is combined in the ‘AND’ block so that both (or all three) redundant components must fail to cause failure of the subsystem.

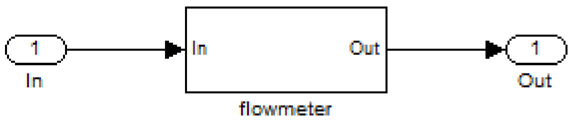


Figure 12 – Alternative 1: no redundancy

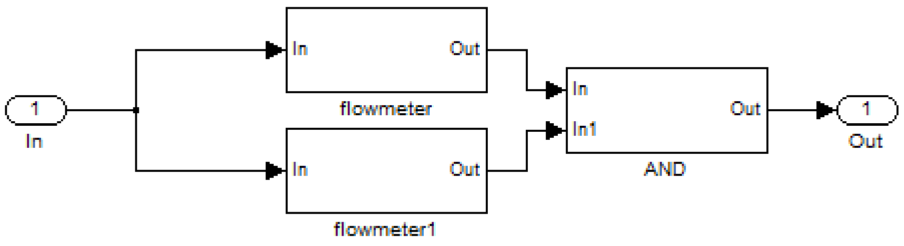


Figure 13 – Alternative 2: one parallel redundancy

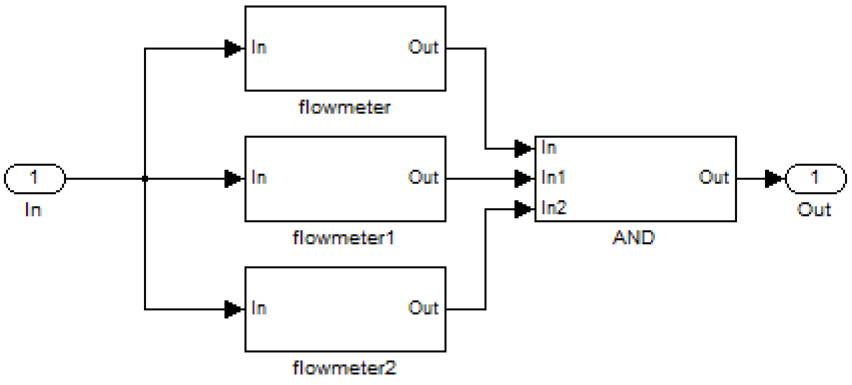


Figure 14 – Alternative 3: two parallel redundancies

All of the subcomponents have the same failure propagation logic. The omission of flow of oil at the output is caused by the omission of flow of oil at the input, or an internal failure of the component.

All other system subcomponents		
Output Deviation	**Description**	**Failure logic (Propagation)**
OmissionFlow-Out	Omission of flow of oil at the output can be caused either by an omission of flow of oil at the input or an internal failure mode of the component	OmissionFlow-In or [component]Failure

Each of the subcomponents has 3 alternative implementations with different costs and the internal failure modes have different failure rates.

	Alternative 1		**Alternative 2**		**Alternative 3**	
Components	**Cost**	**Failure Rate**	**Cost**	**Failure Rate**	**Cost**	**Failure Rate**
Indicator filter	1500	5.0E-7	2500	2.0E-7	3222	1.0E-7
Viscosimeter	2500	2.5E-6	3178	1.0E-6	3814	5.0E-7
Pre-heater	2000	6.7E-6	2505	5.0E-6	3956	1.0E-6
Circulation pump	6000	3.2E-5	13380	2.0E-5	18000	7.0E-6
Mixing tank	2000	1.6E-5	2963	8.0E-6	4444	2.0E-6
Flow meter	2000	1.0E-5	3000	1.0E-6	4444	5.0E-7
Automatic filter	2000	1.0E-5	2647	5.0E-6	3529	1.0E-6
Booster pump	5000	3.2E-5	10682	2.0E-5	12500	5.0E-6
Service tank	1500	1.6E-5	1957	5.0E-6	2739	1.0E-6

Once a functional model, complete with failure data augmentations, has been created in Matlab, it is output to a *.hipx* file. This file is then parsed by HiP-HOPS and loaded into a model data structure.

It is possible to flag both components and individual implementations for inclusion/exclusion from consideration during the optimisation.

2.3.2.1 Configuration options

During optimisation the genetic algorithm can generate different potential solutions by selecting between specified alternative implementations of the components and subsystems.

In some cases this can be simply replacing a component with a more reliable or less expensive (but functionally equivalent) version. It is also possible to select between replacement subsystems which may employ different fault tolerant architectures such as parallel redundancy or standby recovery.

The combination of these two mechanisms allow all aspects of the architecture to be altered by the genetic algorithm.

2.3.2.2 Evaluation methods

As the algorithm promotes survival of the fittest, it is necessary to evaluate the fitness of each new individual and be able to compare it to other individuals in the population.

The evaluation uses the model based evaluation tool HiP-HOPS to calculate the objective values of the individuals.

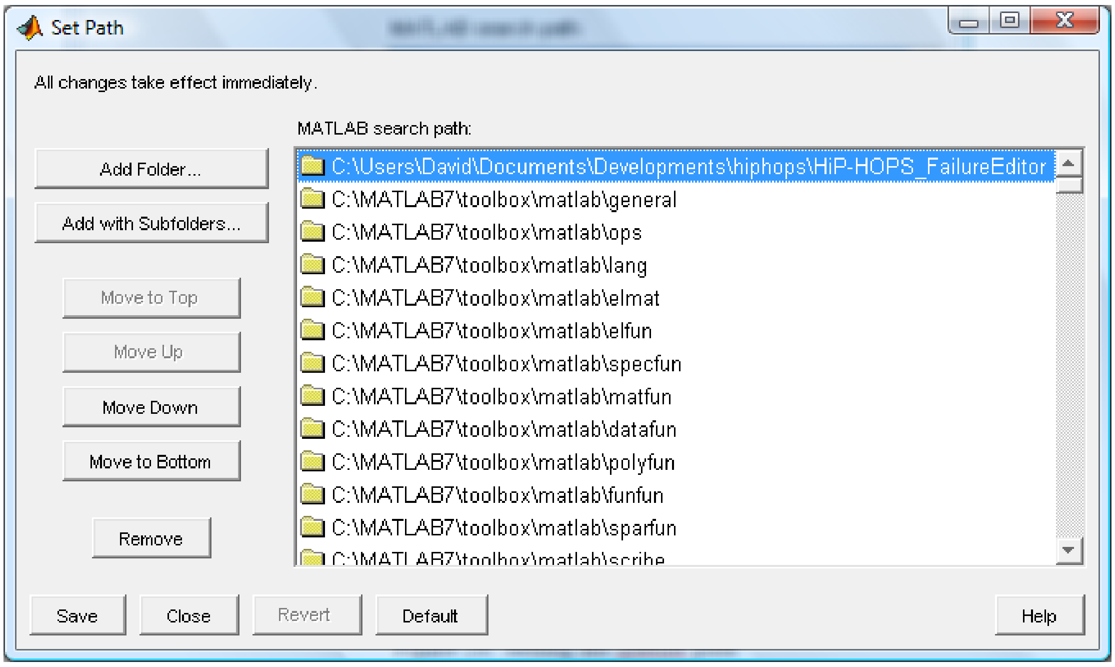
- Cost and weight values are calculated by summing the values for all the components used in the solution. For example the configuration in **Error! Reference source not found.** has a cost of (5 + 4 + 4) + (2 + 2) + (5) = 22 and a weight of (6 + 6 + 6) + (5 + 5) + (6) = 34.
- Unavailability is calculated through HiP-HOPS analysis, first traversing the model to mechanically synthesise interconnected system fault trees, converting the fault trees into their minimal cut sets, and finally using an order 3 inclusion exclusion algorithm as detailed in the existing HiP-HOPS specification. The unavailability of the **Error! Reference source not found.** example is 2.56E-07.
- Reliability is equal to 1 – unavailability where repair rate = 0.
- A safety metric could be established qualitatively from the cumulative severity of all single points of failure in a given design. In the example there is one single point of failure located in the Actuator unit where no redundancy is employed. The severity for the deviation caused by this failure mode is 1e-03, hence safety is (1 x 1e-03) = 1e-03.
- Risk can be calculated from failure probabilities and severities of the top events of the fault trees produced by the tool. In the example the unavailability is 2.56E-07 and the severity of the top event of the single fault tree is 1e-03, therefore the risk would be (2.56E-07 x 1e-03) = 2.56e-10.

Once the model has been evaluated, calculating the values that correspond to the objectives specified by the user, the values are stored with the individual encoding. The evaluation is complete and the individual will not be changed therefore the model can be deleted for memory efficiency.

3 USING HIP-HOPS WITH MATLAB SIMULINK

An interface for using HiP-HOPS with Matlab Simulink models has been developed to allow the evaluation of the technology. The commands are operated through the use of buttons in the HiP-HOPS Launcher.

When you first install HiP-HOPS you need to add the Interface folder to the Matlab path. To do this click on File > Set Path in the Matlab menu. This opens up the window below.



Set Path

All changes take effect immediately.

MATLAB search path:

```
C:\Users\David\Documents\Developments\hiphops\HiP-HOPS_FailureEditor
C:\MATLAB7\toolbox\matlab\general
C:\MATLAB7\toolbox\matlab\ops
C:\MATLAB7\toolbox\matlab\lang
C:\MATLAB7\toolbox\matlab\elmat
C:\MATLAB7\toolbox\matlab\elfun
C:\MATLAB7\toolbox\matlab\specfun
C:\MATLAB7\toolbox\matlab\matfun
C:\MATLAB7\toolbox\matlab\datafun
C:\MATLAB7\toolbox\matlab\polyfun
C:\MATLAB7\toolbox\matlab\funfun
C:\MATLAB7\toolbox\matlab\sparfun
C:\MATLAB7\toolbox\matlab\scribe
```

Figure 15: Setting the Matlab path

Click on the ‘Add Folder’ button and use the file dialog window to navigate to the location that you installed HiP-HOPS. Select the HiP-HOPS_FailureEditor folder and click the ‘OK’ button. Click on ‘Save’ to commit the path entry and then you can click on ‘Close’ to exit the window. You are now ready to use HiP-HOPS in Matlab.

In order to perform HiP-HOPS functions, the user is provided with the HiP-HOPS Launcher. To display the launcher interface (see Figure 16) type ‘HiPHOPS_Launcher’ without quotes into the Matlab command line.

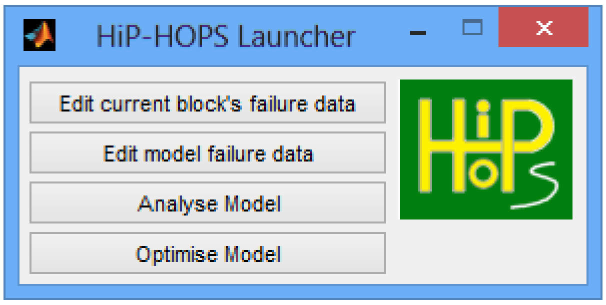


Figure 16: HiP-HOPS Launcher interface in Matlab

To initiate HiP-HOPS commands you can click on the three buttons on this interface. These will be covered in detail below.

3.1 Annotating the model with failure data

Before HiP-HOPS can create fault trees, it needs to know how the components in the model are connected together and how they can fail. This information is entered by adding failure annotations to the components using the HiP-HOPS FailureEditor.

To demonstrate the features of the interface a small standby recovery example will be annotated, see Figure 17 and Figure 18. At the top level there are two components, a sensor component and a standby recovery block. The standby recovery block receives input from the sensor and performs some function on the data. The primary function and output of the block occurs under normal conditions. If there is a detected failure then a standby function is engaged with corresponding standby output.

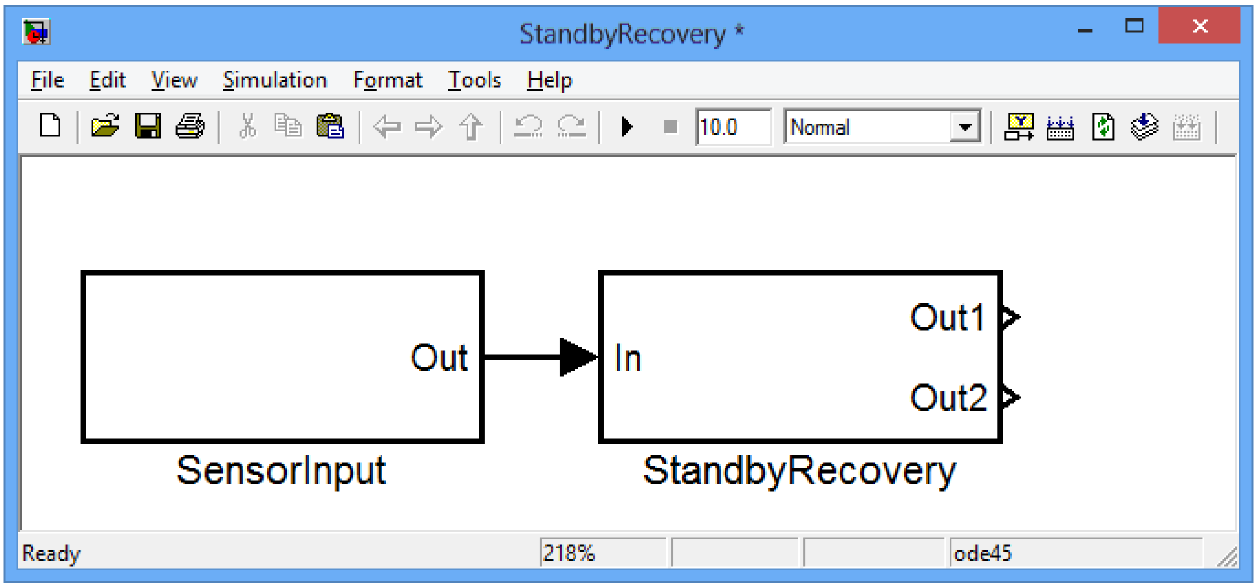


Figure 17: Standby recovery system modelled in Simulink

Figure 18 shows the hierarchical decomposition of the standby recovery block. It is composed of two components, and primary and a standby. The standby component monitors the output of the primary component and when an omission is detected the standby is activated.

Both primary and standby components receive the same input.

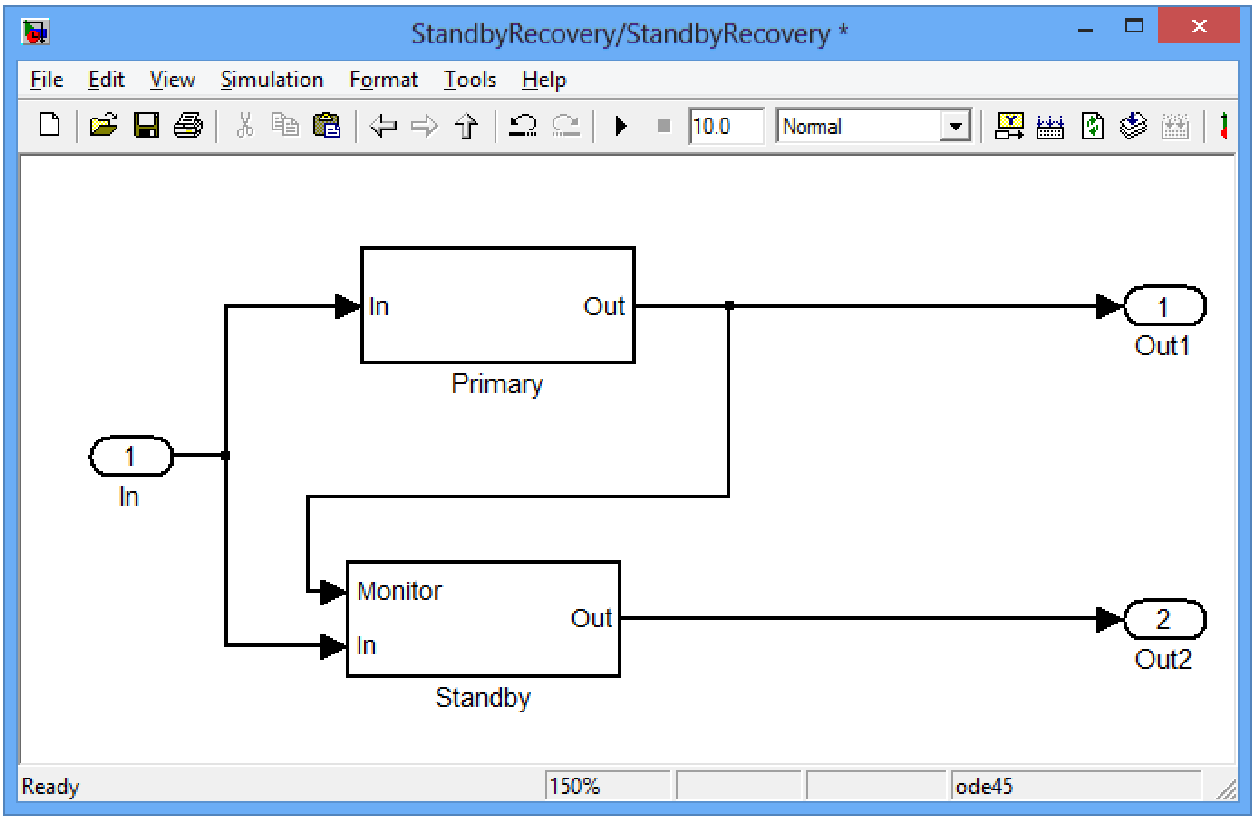

StandbyRecovery/StandbyRecovery *
File Edit View Simulation Format Tools Help
10.0
Normal
1
In
In
Out
Primary
1
Out1
Monitor
In
Out
Standby
2
Out2
Ready
150%
ode45

Figure 18: Decomposed StandbyRecovery subsystem from model in Figure 17

For each component in the model, you need to open the FailureEditor by selecting the component in the model view window and then pressing the 'Edit current block's failure data' button on the HiP-HOPS Launcher interface (Figure 16). The following screen will then appear (Figure 19):

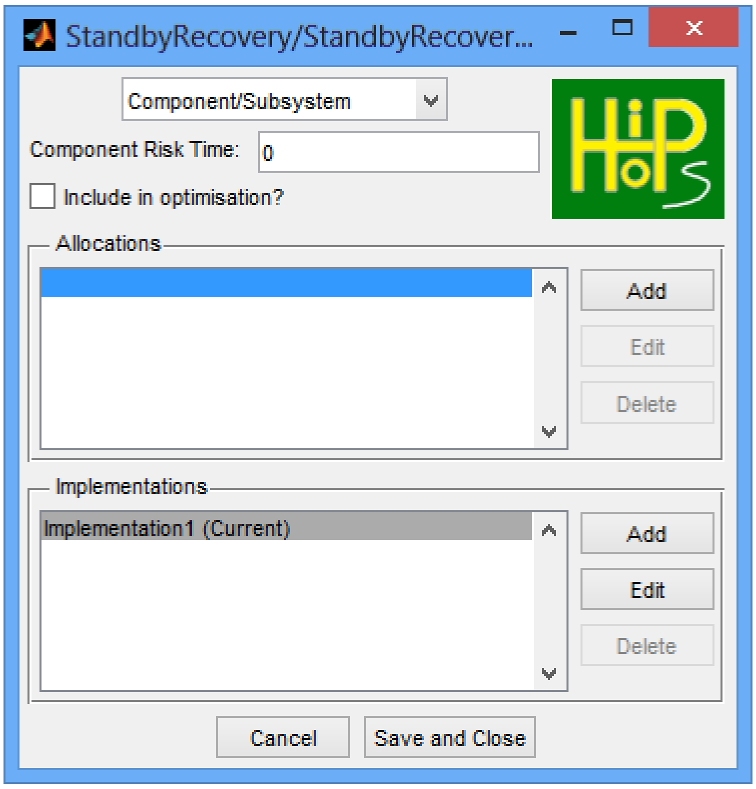


Figure 19: HiP-HOPS Failure Editor window for the standby recovery block

This screen allows you to set the general failure behaviour of the selected component.

3.1.1 General

The FailureEditor is a modal window so you will not have access to Matlab until you have finished editing the current component.

The title bar of the FailureEditor interface shows the name of the currently selected component.

A local risk time for the component can be specified that will override the model (or parent component). This can be used if, for example, a component is only active (and can only fail) during a portion of the overall model's lifetime and therefore has a reduced risk exposure time.

'Include in optimisation?' is a check box that is used to specify whether this component can be altered by the optimisation algorithm when/if the model is optimised.

Below this is a list of the implementations of this component. The implementation that is being currently used by the component is indicated by '(Current)' being appended to the name.

There is must always be at least one implementation, therefore an empty implementation is automatically generated the first time that this window is displayed for a component.

To add a new implementation to the component click on the 'Add' button.

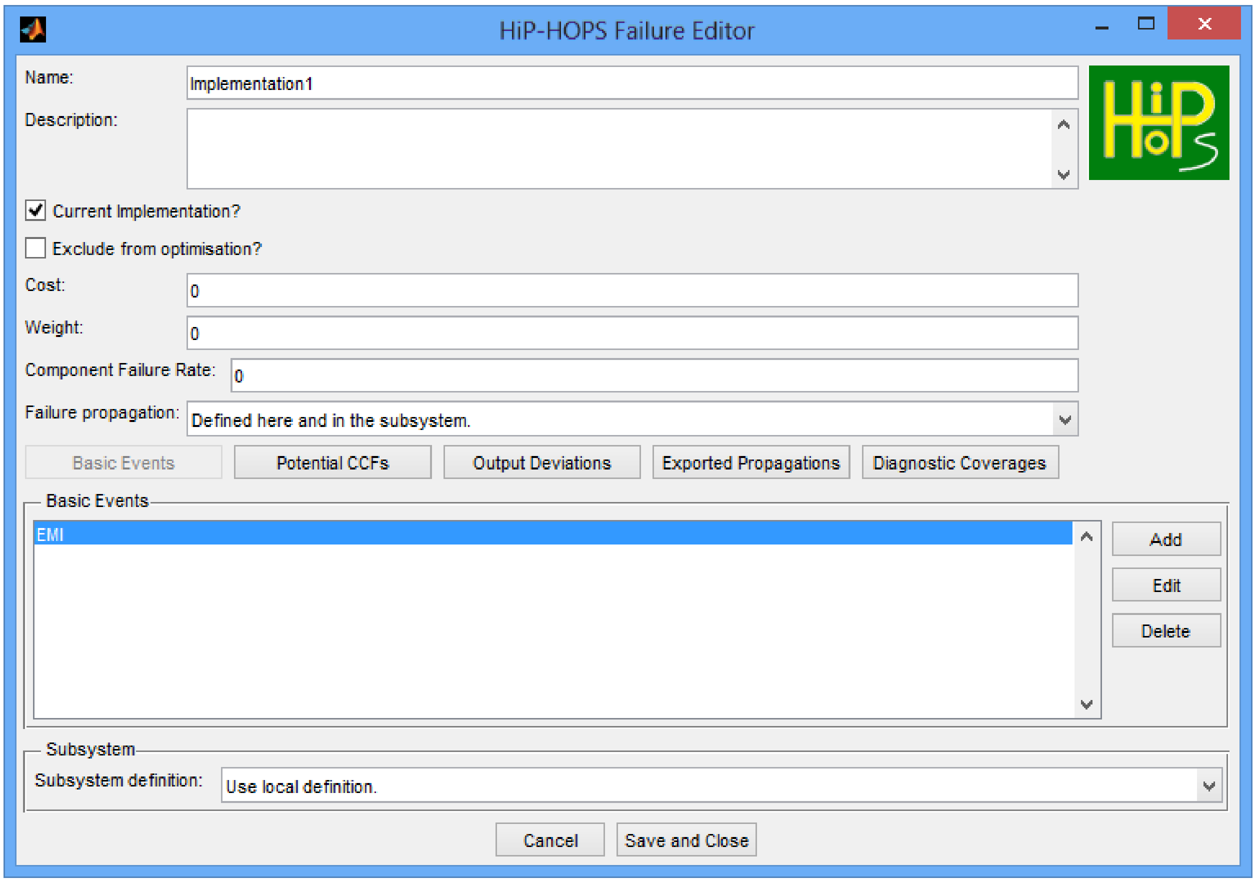
If you wish to edit a previously created implementation then select the implementation in the list and click on the 'Edit' button.

Deleting a previously created implementation can be achieved by selecting the implementation from the list and clicking the 'Delete' button. As there must be at least one implementation, the delete button will be disabled if only one implementation is present.

At any point during the editing you can click on the 'Cancel' button to discard any changes or on the 'Save and Close' button to commit the changes to the component. Both buttons close the FailureEditor window and return control to Matlab.

The failure data for the component and its implementations is stored in an RTWData field called HiPHOPS_Component_Failure_Data in the mdl file.

If you clicked on either the 'Add' or 'Edit' implementation button then the Implementation dialog (Figure 20) would appear.


HiP-HOPS Failure Editor

Name: Implementation1

Description:

[✓] Current Implementation?

[ ] Exclude from optimisation?

Cost: 0

Weight: 0

Component Failure Rate: 0

Failure propagation: Defined here and in the subsystem.

Basic Events | Potential CCFs | Output Deviations | Exported Propagations | Diagnostic Coverages

Basic Events

EMI

Add | Edit | Delete

Subsystem

Subsystem definition: Use local definition.

Cancel | Save and Close

Figure 20: Implementation dialog Matlab

3.1.2 The Implementation dialog

Using this dialog you can specify the failure data for an implementation of the selected component.

The name field is used to specify the identifying name for the implementation and this has to be unique within the implementations for this component. This is enforced.

A textual description for the implementation can also be entered.

The 'Current Implementation?' check box is used to specify whether this implementation is the one designated to be used as current by the component.

During optimisation, components that are eligible for configuration by the optimisation algorithm will select between the different implementations to be used. By default it will select between all implementations unless an implementation has been flagged for exclusion from selection by checking the 'Exclude from optimisation?' check box.

Different implementations have different costs associated with them. These can be specified in the 'Cost' and 'Weight' fields.

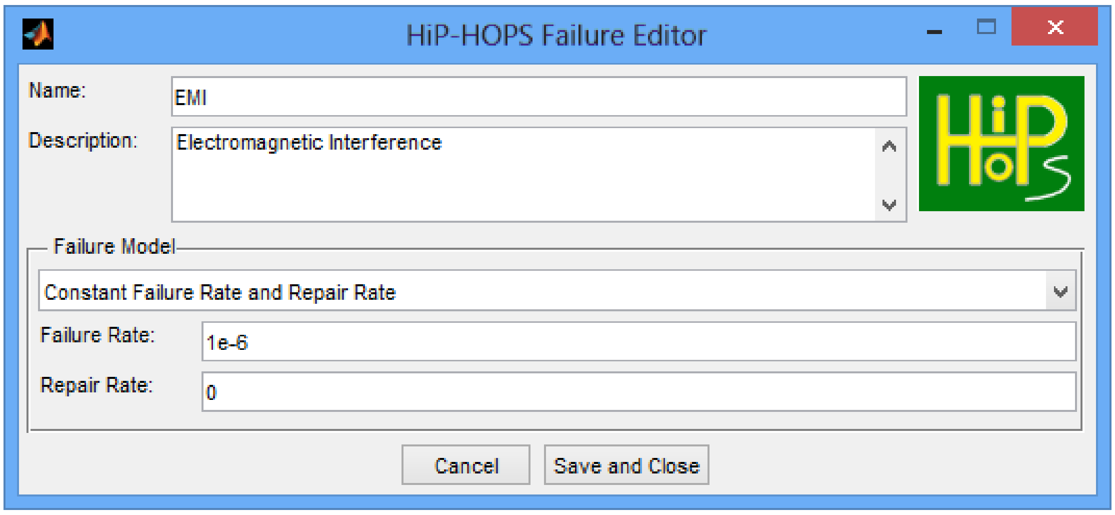
As HiP-HOPS can be used with hierarchical models it contains a mechanism for blocking propagation of failures at a particular level of the model. By default failure propagations can be 'Defined here and in the subsystem'. When this option is chosen failure propagation is unimpeded and failures that occur at this level of the model are propagated as well as those defined in the subsystems. There are two other options for this setting. The first is 'Defined here only' and the second is 'Defined in the subsystem only'. These block the propagation of failure in the subsystem and from this level respectively.

Next is a list of basic events followed by a list of output deviations for the implementation. Basic events and output deviations can be added, edited, or deleted by clicking on the respective 'Add', 'Edit', and 'Delete' buttons. To edit or delete a basic event or output deviation they must first be selected from the respective list.

Again the 'Cancel' button will discard any changes to the implementation and the 'Save and Close' button will commit the changes before closing the implementation dialog window.

Adding and editing basic events and output deviations are described in the following sections.

3.1.3 Basic Events


HiP-HOPS Failure Editor

Name: EMI

Description: Electromagnetic Interference

Failure Model: Constant Failure Rate and Repair Rate

Failure Rate: 1e-6

Repair Rate: 0

Figure 21: Basic Event dialog Matlab

The Basic Event dialog is where you can enter the information about the basic events for this implementation, i.e. the failures that originate with the current component. Basic events can include things like wear, environmental factors (e.g. temperature), faults in the component (e.g. overheating, short circuits etc), or anything else appropriate.

The dialogue is shown in Figure 21, but the controls are very simple.

The name of a basic event uniquely identifies it within the implementation and as such must be unique. This is enforced.

The basic events entered here are referenced by name in the Output Deviations section, as well as the results of the analysis, so it is important to give the basic events meaningful names. The results will indicate which component a basic event belongs to, however, so it is not necessary to do this manually, e.g. "Component1_blocked", "Component2_blocked"; just naming it "blockage" or similar is sufficient.

You can also enter a description for the basic event.

Using this dialog you can enter the failure model for the basic event. These are described in the next section.

As with the other dialogs you can press the 'Cancel' or 'Save and Close' buttons to exit the dialog.

3.1.3.1 Failure Rates

Entering the failure models for the basic events, allows quantitative analysis to take place. The calculation mode section on the right allows you to assign a failure model to the basic events. There are a number of different formulae, each with different parameters, and each will yield a different unavailability for the basic event. These can be selected from the drop down menu, and you can then fill in the appropriate parameters.

The current formulae are listed below.

Constant Failure and Repair Rate
Parameters:

λ	- Failure Rate
μ	- Repair Rate

This is the currently implemented calculation method, and assumes an exponential distribution. The formula for unavailability is as follows:

$$u = \frac{\lambda}{\lambda + \mu} \times (1 - e^{-(\lambda+\mu)t}) \qquad (3.1)$$

Constant Failure and Mean Time To Repair (MTTR)
Parameters:

λ	- Failure Rate
MTTR	- Mean Time To Repair

Very similar to above, with the exception that the repair data is entered as the MTTR instead. This is then converted to the repair rate μ (since μ = 1 / MTTR) and the unavailability calculation 3.1 is used.

Mean Time To Failure(MTTF) and constant Repair
Parameters:

MTTF	- Failure Rate
μ	- Repair Rate

Very similar to above, with the exception that the repair data is entered as the MTTF instead. This is then converted to the repair rate μ (since λ = 1 / MTTF) and the unavailability calculation 3.1 is used.

Mean Time to Failure and Repair
Parameters:

MTTF	- Mean Time To Failure
MTTR	- Mean Time To Repair

Like the last, except that both the failure and the repair data are entered as mean times. These values will be converted and the formula in 3.1 used.

Fixed Unavailability
Parameters:

Unavailability	- The constant unavailability

This is the option to choose if you already knows the unavailability of the event.

Binomial Failure Model
Parameters:

λ	- Failure Rate
μ	- Repair Rate
n	- Number of components
m	- Minimum number of components needed to fail to cause subsystem failure
T	- The time of operation of the subsystem

This model is useful for representing situations where m failed components out of n will result in failure, such as in a voter. The formula is as follows:

$$u = \sum_{k=m}^{n} \left(\frac{n!}{k! \times (n-k)!} \times q^k \times (1-q)^{(n-k)} \right) \tag{3.2}$$

Poisson Failure Model
Parameters:

λ	- Failure Rate
n	- Number of components in operation at any one time
s	- Number of spare components available
t	- Time of operation of the subsystem

This method can be used to model the effects of limited numbers of replacement components. Fomula:

$$u = \sum_{k=0}^{s} \left(\frac{x^k \times e^{-x}}{k!} \right) \tag{3.3}$$

Dormant Failure Model with periodic inspection
Parameters:

λ	- Failure Rate
MTTR	- Mean time to repair
T	- Time between inspections

This model can be used when a component is a standby component, and only activated when its primary fails. Formula:

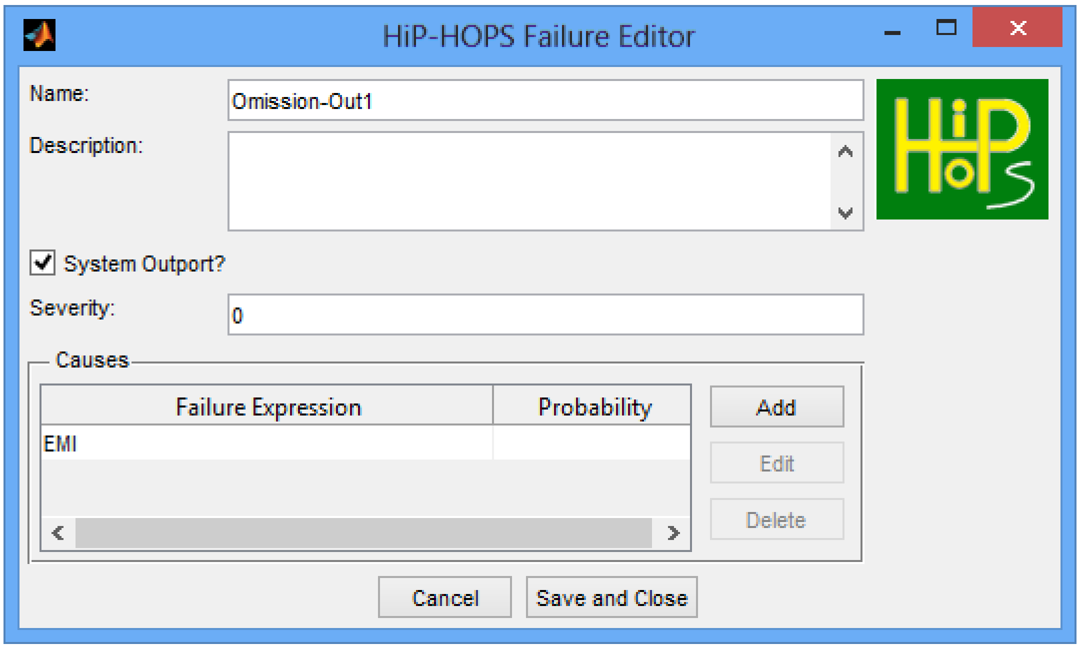
$$u = \frac{\lambda t - (1 - e^{-\lambda t}) + \lambda \times MTTR \times (1 - e^{-\lambda t})}{\lambda t + \lambda \times MTTR \times (1 - e^{-\lambda t})} \tag{3.4}$$

Variable Failure Rate
Parameters:

Slope parameter 1
Scale parameter 1
End interval 1
Scale parameter 2
End interval 2
Slope parameter 3
Scale parameter 3

This model allows the failure of the event to be described as a bath tub with Weibull variable failure rates.

3.1.4 Output Deviations


HiP-HOPS Failure Editor

Name: Omission-Out1

Description:

☑ System Outport?

Severity: 0

Causes

Failure Expression	Probability
EMI	

Figure 22: Output Deviation dialog

The Output Deviations dialog (Figure 22) is the section where the failure behaviour of the component is defined by specifying relationships between the basic events of a component and the inputs and outputs of the component.

The name of an output deviation is comprised of two parts. The first part is the failure class; this can be anything such as ‘Omission’ or ‘Commission’. You can also use abbreviations such as ‘O’ for omission. It is important that the failure classes used are consistently spelt and used throughout the model for the failure propagation to work properly. The second part of the name is the port at which the deviation occurs. It is important that the name of the port refers to a port of the current component as these are local failures. The two parts are separated by a hyphen. For example: ‘Omission-port1’.

You can enter a description for the output deviation. This is especially useful where abbreviations are used for failure classes.

The ‘System Outport’ check box is used to specify whether the output deviation is a system level failure output. System Outport output deviations are the top nodes of the system fault trees that are generated during the analysis.

The severity of the output deviation can be specified in the appropriate field. This is used when the output deviation is a system outport in the FMEA tables and for calculating a risk value.

Lastly, there is a box for the logical expression this output deviation represents. An output deviation is just an expression that connects component inputs and basic events to the component outputs, and that expression is defined here. You can reference any of the basic events of this component (as defined in the "Basic Events" dialog) as well as referencing any input deviations. Input deviations are not defined elsewhere, they are represented only in the expression by specifying a failure class and an input port, just as with output deviations. You can then join these together using logical operators AND and OR, or their symbols, a star ("*") for AND and a plus sign ("+") for OR. The operators do not need to be capitalised. You can also use brackets.

It should be noted that identifiers in HiP-HOPS are case sensitive so ‘failure’ is not the same as ‘Failure’.

Use the ‘Cancel’ and ‘Save and Close’ buttons to discard and save the changes respectively before exiting.

3.1.5 Subsystems

The final setting in the implementation failure editor window is defining the location of the subsystem. Sometimes for optimising a model it is necessary for a component to have implementations that are decomposed with different subsystems. By default the setting is ‘Use local definition’. In this case the subsystem in the model currently being edited is used.

The other option is to import the subsystem from another mdl file (Import external model). When this option is selected an extra field is displayed with a ‘Browse’ button. When the ‘Browse’ button is clicked a file dialog is displayed and the user can navigate to and select an existing mdl file to use as the subsystem for this component implementation.

The path is stored relative to the current model path.

Care should be taken to ensure that the external model has the inputs and outputs to match those of the current component.

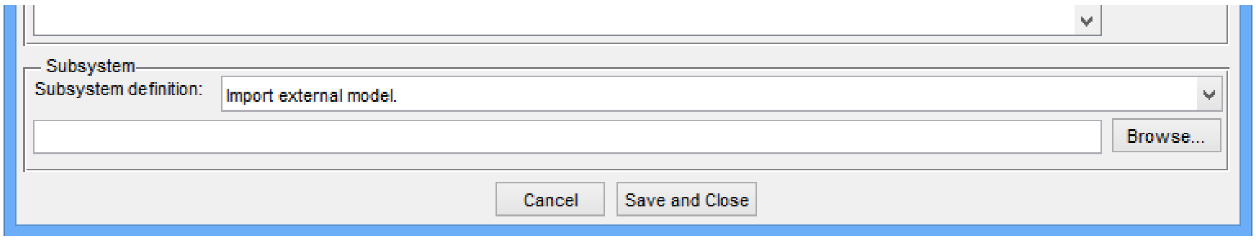
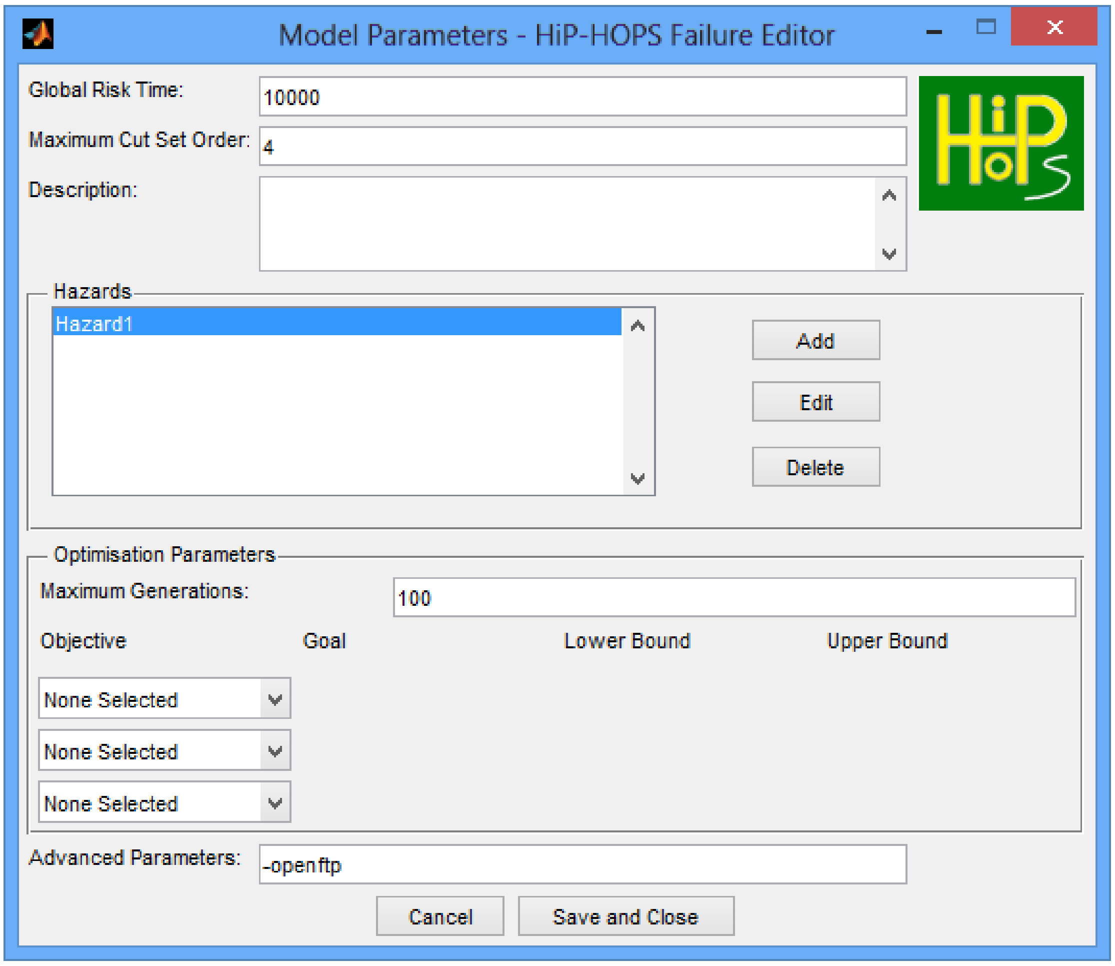

Subsystem
Subsystem definition: Import external model.

Figure 23: Importing an alternative subsystem

3.2 Performing safety analysis

Once you have annotated your system model, you can run HiP-HOPS to analyse it and generate fault trees and FMEA information. To do this, select "Analyse" from the HiP-HOPS Launcher. This opens the window shown in Figure 24.


Model Parameters - HiP-HOPS Failure Editor

Global Risk Time: 10000

Maximum Cut Set Order: 4

Description:

Hazards

Hazard1

Optimisation Parameters

Maximum Generations: 100

Objective	Goal	Lower Bound	Upper Bound
None Selected			
None Selected			
None Selected			

Advanced Parameters: -openftp

Figure 24: Safety analysis Matlab

Before analysis of the model is carried out the Risk Time field allows you to specify a model-wide duration of exposure to risk for the quantitative calculations.

You can also add a model description.

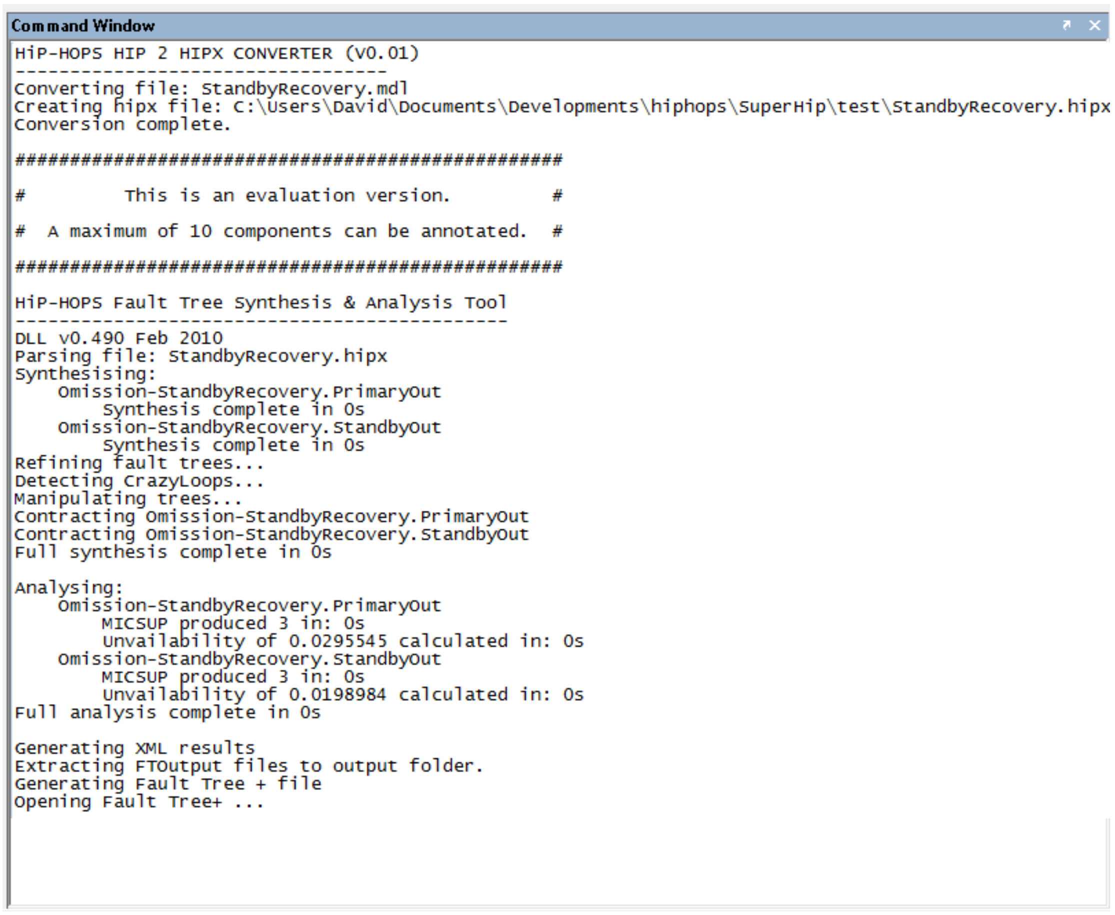
The advanced parameters field is used to specify additional HiP-HOPS exe arguments. This will normally be left blank as it would primarily be used for testing new features and helping with debugging.

Changes made here are discarded if you click on the ‘Cancel’ button. Alternatively if you click on ‘Save and Run’ then the changes are saved and these model parameters are stored in an XML tag <HiP-HOPS_Model_Parameters> in the model’s description field.

The mdl file is saved automatically and it is then passed as a parameter to the HiP-HOPS Matlab2Hipx converter. This uses the mdl file (and the contained failure data) to create the hipx interface file that is readable by the HiP-HOPS analysis tool.

Once the hipx file is created it is passed as a parameter to the HiP-HOPS analysis tool. The model is analysed and the results output. Viewing the results is discussed in a later section.

The progress of the analysis, along with any errors, is displayed in the Matlab command window as shown in Figure 25.



```
HiP-HOPS HIP 2 HIPX CONVERTER (V0.01)
----------------------------------
Converting file: StandbyRecovery.mdl
Creating hipx file: C:\Users\David\Documents\Developments\hiphops\SuperHip\test\StandbyRecovery.hipx
Conversion complete.

##################################################

#         This is an evaluation version.         #

#  A maximum of 10 components can be annotated.  #

##################################################

HiP-HOPS Fault Tree Synthesis & Analysis Tool
---------------------------------------------
DLL v0.490 Feb 2010
Parsing file: StandbyRecovery.hipx
Synthesising:
    Omission-StandbyRecovery.PrimaryOut
        Synthesis complete in 0s
    Omission-StandbyRecovery.StandbyOut
        Synthesis complete in 0s
Refining fault trees...
Detecting CrazyLoops...
Manipulating trees...
Contracting Omission-StandbyRecovery.PrimaryOut
Contracting Omission-StandbyRecovery.StandbyOut
Full synthesis complete in 0s

Analysing:
    Omission-StandbyRecovery.PrimaryOut
        MICSUP produced 3 in: 0s
        Unvailability of 0.0295545 calculated in: 0s
    Omission-StandbyRecovery.StandbyOut
        MICSUP produced 3 in: 0s
        Unvailability of 0.0198984 calculated in: 0s
Full analysis complete in 0s

Generating XML results
Extracting FTOutput files to output folder.
Generating Fault Tree + file
Opening Fault Tree+ ...
```

Figure 25: Analysis progress Matlab

3.3 Viewing the analysis results

If no errors were found during the analysis, it will produce a folder containing the results in XML format. The folder will have the same name as the model file, but with "-FMEAOutput" added to it. When specified by the user the tool also generates a Fault Tree + file (with the extension ".psa") in the same folder as the model file, which can be opened with Fault Tree + if you have it installed. The .psa file has the same name as the name of the model itself, rather than the name of the model file. There is also an option to output the results as a Microsoft Excel compatible XML file. The file name for this is made up of the model name plus "_Excel" with file extension ".xml".

Once it has finished, you can view the results by looking in the newly created folder and then double clicking to open the start page "Index.html". This will attempt to open the results in your default browser, allowing you to navigate the results via hyperlinks just like on a web page. It should be noted that only Internet Explorer 6 and above is currently supported although we are working to include other browsers.

When you first open the index page it may prompt you to authorise the viewing of the active scripts used to display the page, as in Figure 26.

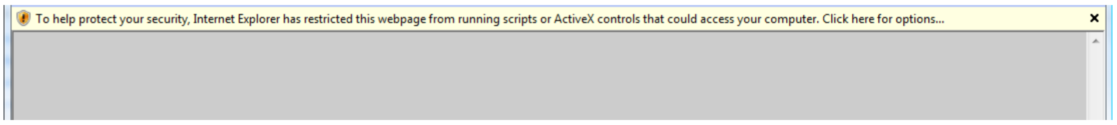

To help protect your security, Internet Explorer has restricted this webpage from running scripts or ActiveX controls that could access your computer. Click here for options...

Figure 26: Authorisation message

In order to continue you must click on the visible message causing a pop-up menu to appear. You need to click on “Allow Blocked Content...” as in Figure 27.

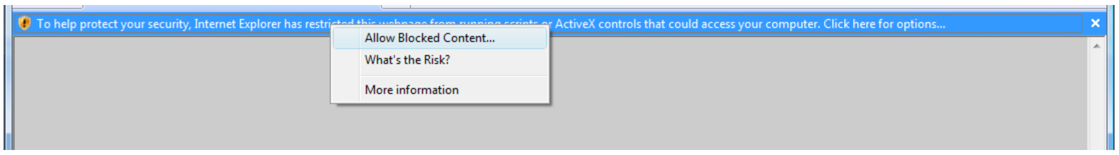
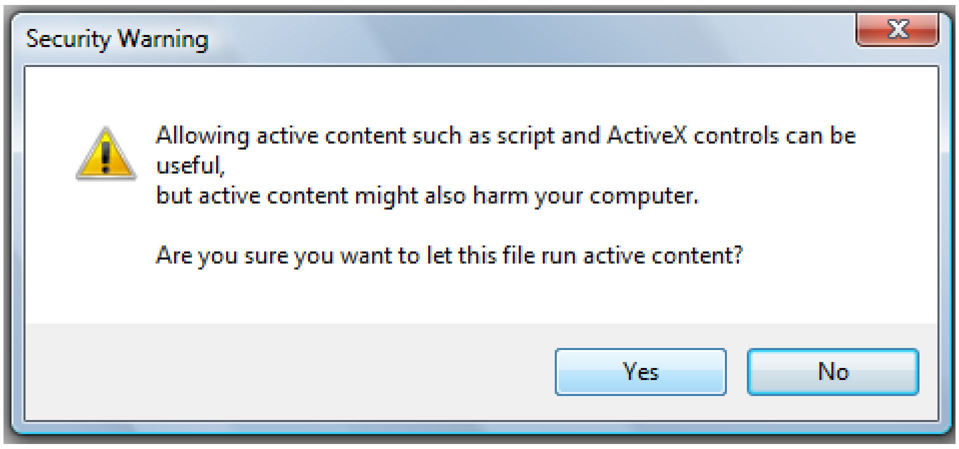


Figure 27: Authorisation pop-up

This action causes a final dialogue box to appear and you must click yes to allow the scripts used in the page to run.


Security Warning

Allowing active content such as script and ActiveX controls can be useful,
but active content might also harm your computer.

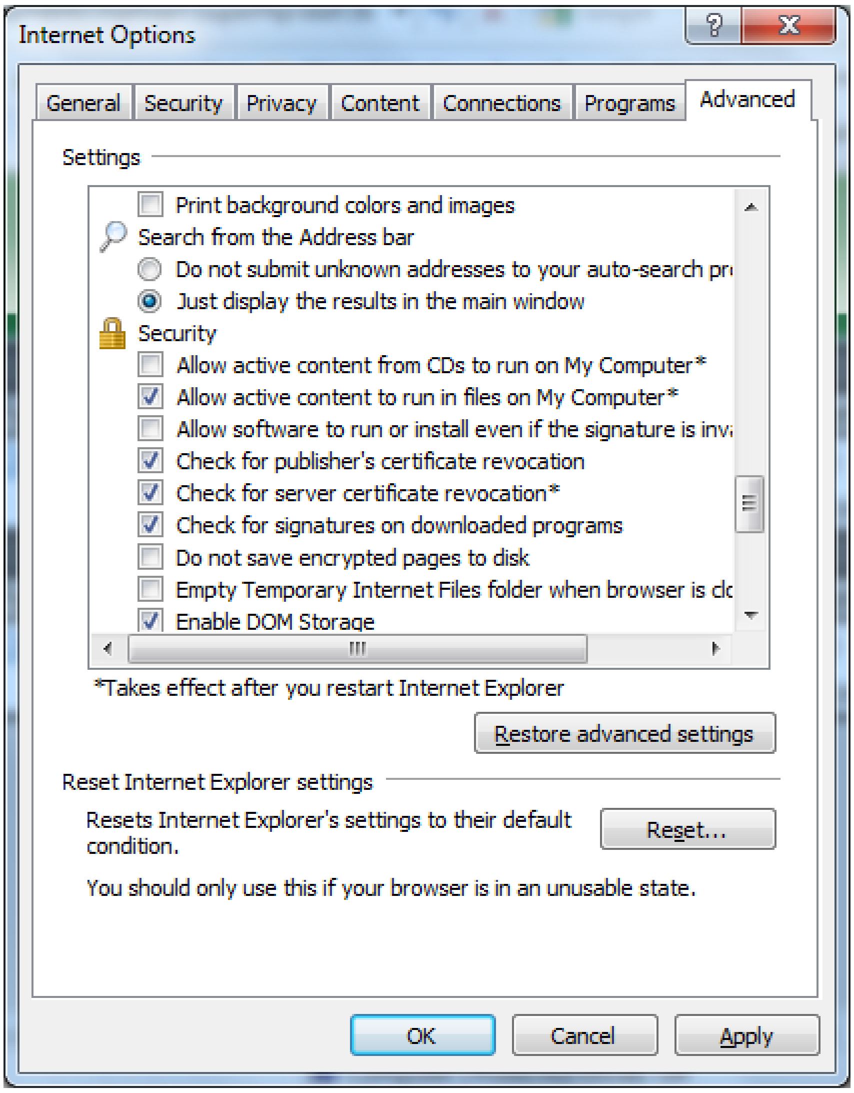
Are you sure you want to let this file run active content?

Figure 28: Authorisation dialogue confirmation

Alternatively the user can permanently disable the authorisation dialog through the Internet Options. To access this, the user must click on the ‘Tools’ menu and then select ‘Internet Options’.

Next the ‘Advanced’ tab must be selected and the scroll-box must be scrolled down until the ‘Security’ section. The checkbox ‘Allow active content to run in files on My Computer’ is by default deselected. If you select this option then the Authorisation Dialog will not appear (Figure 29).

It should be noted that this setting is not applicable to only HiP-HOPS and will also affect other web pages.


Internet Options

General | Security | Privacy | Content | Connections | Programs | Advanced

Settings

- [ ] Print background colors and images
- Search from the Address bar
  - ○ Do not submit unknown addresses to your auto-search pr
  - ● Just display the results in the main window
- Security
  - [ ] Allow active content from CDs to run on My Computer*
  - [✓] Allow active content to run in files on My Computer*
  - [ ] Allow software to run or install even if the signature is inv
  - [✓] Check for publisher's certificate revocation
  - [✓] Check for server certificate revocation*
  - [✓] Check for signatures on downloaded programs
  - [ ] Do not save encrypted pages to disk
  - [ ] Empty Temporary Internet Files folder when browser is cl
  - [✓] Enable DOM Storage

*Takes effect after you restart Internet Explorer

Reset Internet Explorer settings

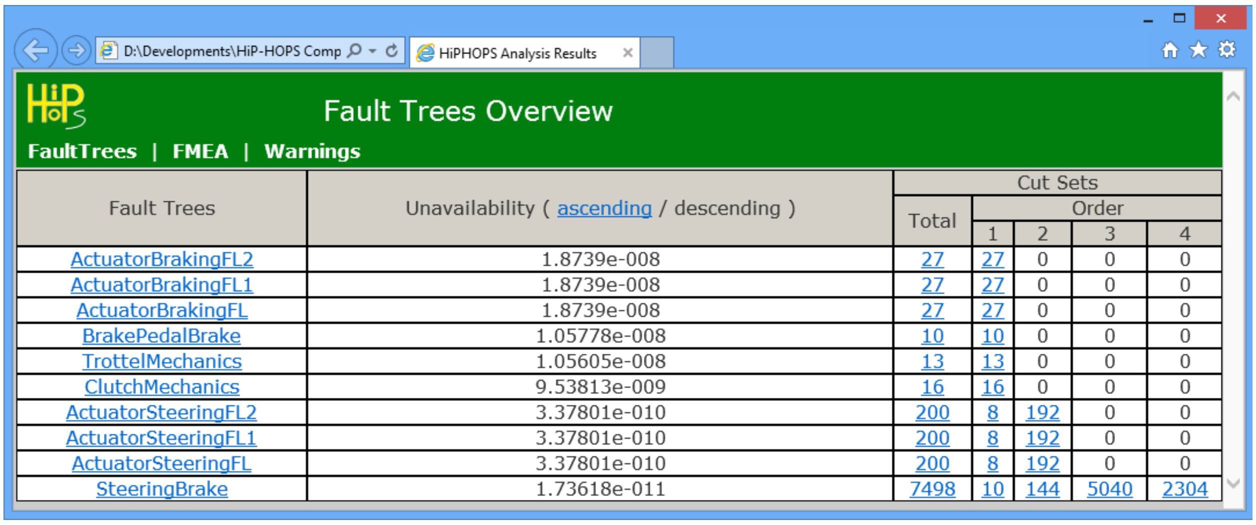
Resets Internet Explorer's settings to their default condition.

You should only use this if your browser is in an unusable state.

Figure 29: Internet options setting.

Once you have authorised the scripts the page should render, displaying a list of all the fault tree top events generated during the analysis as in **Error! Reference source not found.**.

The first page of the results displays a summary of the FTA results; it shows all system failures (fault trees), together with the unavailability if present, and then a break down of the cut sets of each size (or ‘order’). You can click on any of these to view those particular results/cut sets in more detail. In addition, at the top are options to view the FMEA results or the warnings (if any) instead.

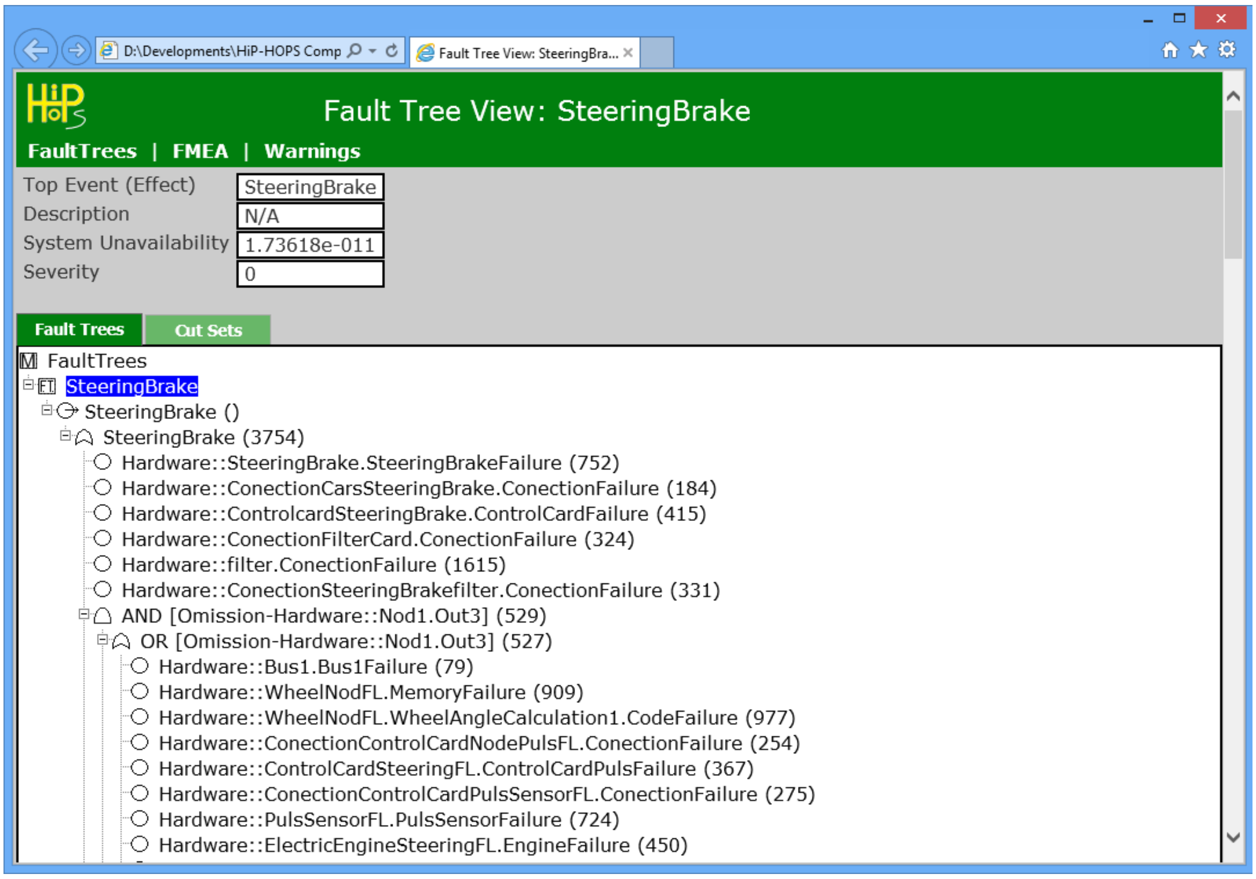


Fault Trees Overview

Fault Trees	Unavailability (ascending / descending)	Cut Sets				
		Total	Order			
			1	2	3	4
ActuatorBrakingFL2	1.8739e-008	27	27	0	0	0
ActuatorBrakingFL1	1.8739e-008	27	27	0	0	0
ActuatorBrakingFL	1.8739e-008	27	27	0	0	0
BrakePedalBrake	1.05778e-008	10	10	0	0	0
TrottelMechanics	1.05605e-008	13	13	0	0	0
ClutchMechanics	9.53813e-009	16	16	0	0	0
ActuatorSteeringFL2	3.37801e-010	200	8	192	0	0
ActuatorSteeringFL1	3.37801e-010	200	8	192	0	0
ActuatorSteeringFL	3.37801e-010	200	8	192	0	0
SteeringBrake	1.73618e-011	7498	10	144	5040	2304

Figure 30 - The FTA summary page

Each fault tree is given a name, which is constructed from the name of the output deviation they come from, e.g. " HP-HM.portB" is the fault tree generated by the output deviation of failure class HP at port "portB" on component "HM". This matches up to an equivalent output deviation in the original model. Clicking on a fault tree name opens the results for that fault tree:


D:\Developments\HiP-HOPS Comp
Fault Tree View: SteeringBra...
FaultTrees | FMEA | Warnings
Fault Trees
Cut Sets

Fault Tree View: SteeringBrake

Top Event (Effect)	SteeringBrake
Description	N/A
System Unavailability	1.73618e-011
Severity	0

```
FaultTrees
SteeringBrake
  SteeringBrake ()
    SteeringBrake (3754)
      Hardware::SteeringBrake.SteeringBrakeFailure (752)
      Hardware::ConectionCarsSteeringBrake.ConectionFailure (184)
      Hardware::ControlcardSteeringBrake.ControlCardFailure (415)
      Hardware::ConectionFilterCard.ConectionFailure (324)
      Hardware::filter.ConectionFailure (1615)
      Hardware::ConectionSteeringBrakefilter.ConectionFailure (331)
      AND [Omission-Hardware::Nod1.Out3] (529)
        OR [Omission-Hardware::Nod1.Out3] (527)
          Hardware::Bus1.Bus1Failure (79)
          Hardware::WheelNodFL.MemoryFailure (909)
          Hardware::WheelNodFL.WheelAngleCalculation1.CodeFailure (977)
          Hardware::ConectionControlCardNodePulsFL.ConectionFailure (254)
          Hardware::ControlCardSteeringFL.ControlCardPulsFailure (367)
          Hardware::ConectionControlCardPulsSensorFL.ConectionFailure (275)
          Hardware::PulsSensorFL.PulsSensorFailure (724)
          Hardware::ElectricEngineSteeringFL.EngineFailure (450)
```

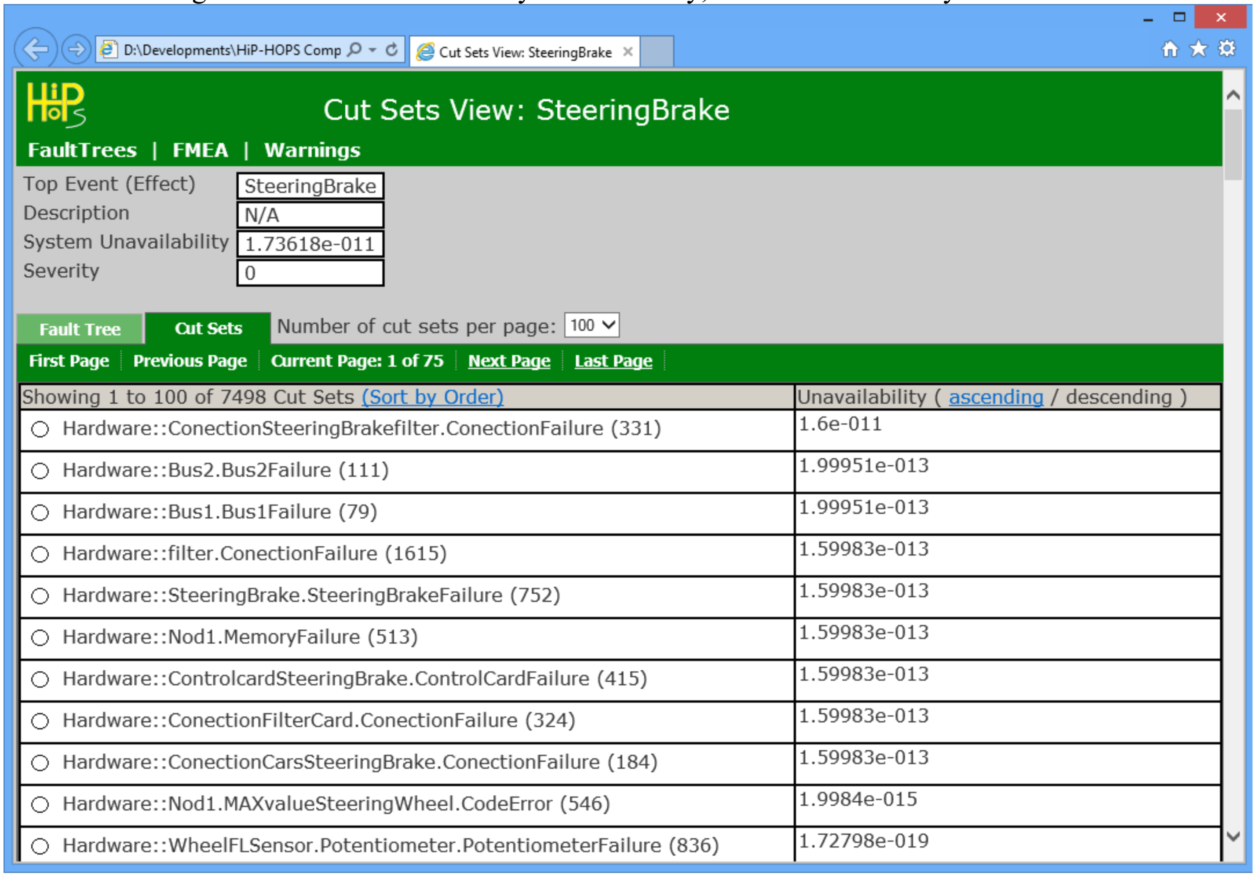
Figure 31 - The fault tree results

The fault tree results, as shown in Figure 31, are divided into two sections, indicated by the tabs. There is also a summary of the fault tree (name, description, unavailability and severity) at the top. The first section shows the fault tree itself, in a tree view-style layout. You can expand/contract the tree by clicking on the +/- boxes. Each line in the tree view is an event (basic or normal event) or gate (e.g. AND or OR gate). Clicking a gate or event's name will open all branches below that point; beware that this may take some time for very large trees.

Selecting the Cut Sets tab shows the second section of the fault tree results. If you selected a cut set order from the FTA summary page, only the cut sets of that size are shown; otherwise, all cut sets are shown. In both cases, they are divided into pages if necessary. You can select how many results to show per page; the default is 100, but you can also display 200, 400, or all results at once, though this can potentially be very slow. You can also move from page to page using the links provided.

Each page shows some minimal cut sets for the current fault tree – all of the events contained must occur to cause the top even of the fault tree to occur. On the left of the view are the events themselves, showing the component name (e.g. valveB1) and the event name (e.g. CloseStuck), together with an ID number. A circle indicates a normal basic event, a circle with arrows indicates a circle node, and a house-shaped symbol indicates a normal event. On the right is the unavailability for each cut set (if quantitative failure data has been provided).

You can also optionally re-order the cut sets to display by order (size), by clicking 'Sort by Order'. To change back to the default sort by unavailability, click 'Unavailability'.


D:\Developments\HiP-HOPS Comp
Cut Sets View: SteeringBrake
FaultTrees | FMEA | Warnings
Fault Tree
Cut Sets
First Page
Previous Page
Current Page: 1 of 75
Next Page
Last Page

Cut Sets View: SteeringBrake

Top Event (Effect)	SteeringBrake
Description	N/A
System Unavailability	1.73618e-011
Severity	0

Number of cut sets per page: 100

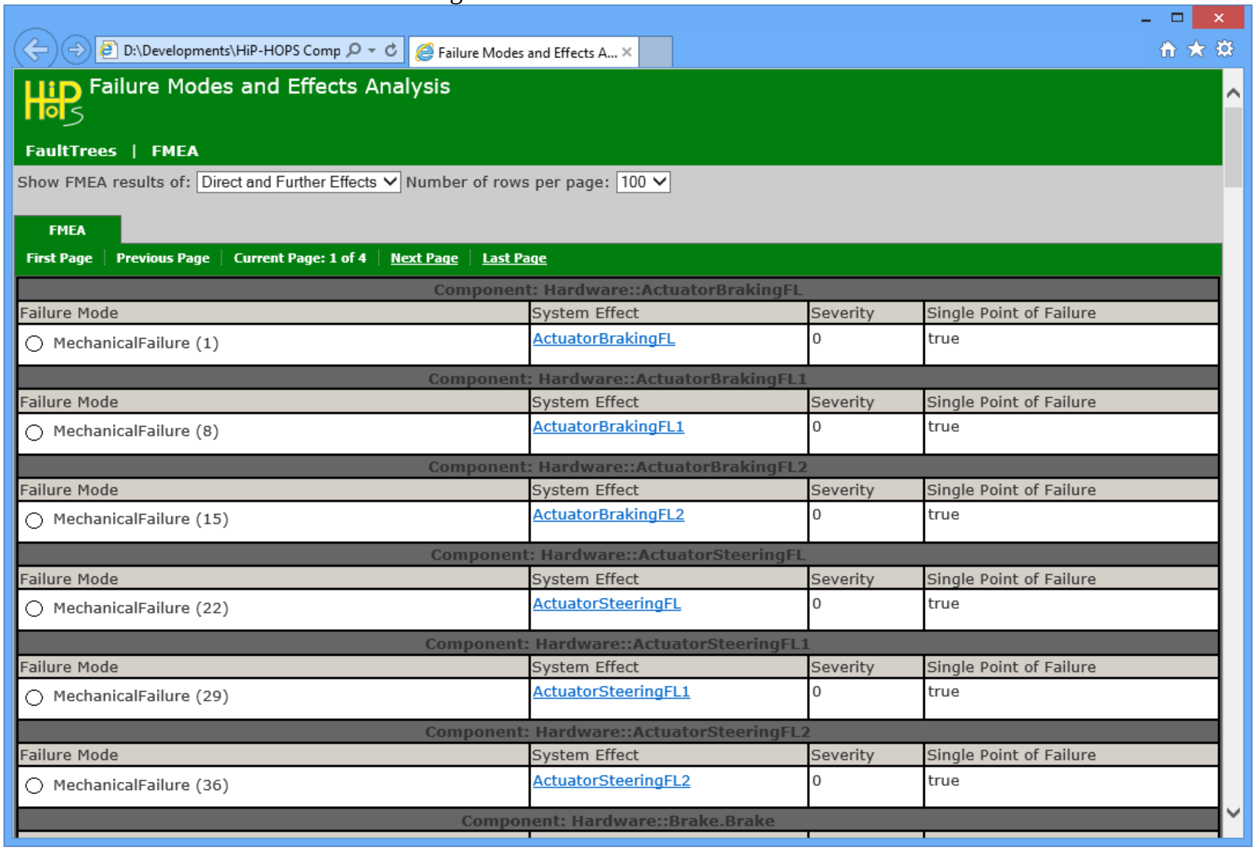
Showing 1 to 100 of 7498 Cut Sets (Sort by Order)	Unavailability (ascending / descending)
○ Hardware::ConectionSteeringBrakefilter.ConectionFailure (331)	1.6e-011
○ Hardware::Bus2.Bus2Failure (111)	1.99951e-013
○ Hardware::Bus1.Bus1Failure (79)	1.99951e-013
○ Hardware::filter.ConectionFailure (1615)	1.59983e-013
○ Hardware::SteeringBrake.SteeringBrakeFailure (752)	1.59983e-013
○ Hardware::Nod1.MemoryFailure (513)	1.59983e-013
○ Hardware::ControlcardSteeringBrake.ControlCardFailure (415)	1.59983e-013
○ Hardware::ConectionFilterCard.ConectionFailure (324)	1.59983e-013
○ Hardware::ConectionCarsSteeringBrake.ConectionFailure (184)	1.59983e-013
○ Hardware::Nod1.MAXvalueSteeringWheel.CodeError (546)	1.9984e-015
○ Hardware::WheelFLSensor.Potentiometer.PotentiometerFailure (836)	1.72798e-019

Figure 32 - The cut set results

To view the FMEA results, you must click the FMEA option in the menu at the top. Like the fault tree cut sets, the FMEA is divided up into pages, and you can choose how many and what sort of results to display in each page. Firstly you can select what ‘order’ of FMEA results to display; an order of 1 is the traditional FMEA in which each failure mode shown has a direct effect on the system, whereas higher orders illustrate the effects of failure modes occurring in conjunction with other events, shown as “contributing failure modes.” In addition you can again choose how many results to display per page and navigate between pages using the “First/Previous/Next/Last page” buttons.

The FMEA itself is divided up by component (all failure modes in a given component are displayed together) and contains four columns. The first column shows the failure mode in question, composed of the component name and the event name, then an ID in brackets. The second column shows the system failure(s) that this failure mode can cause; clicking on the name of the system failure takes you to the relevant fault tree. The third column shows the severity of this failure and the fourth column displays the contributing failure modes (i.e. cut sets), if any. Each set of contributing failure modes must occur in conjunction with the current failure mode in order to cause the system effect.

The FMEA results are shown in Figure 33 overleaf.



Failure Modes and Effects Analysis

FaultTrees | FMEA

Show FMEA results of: Direct and Further Effects Number of rows per page: 100

FMEA

First Page | Previous Page | Current Page: 1 of 4 | Next Page | Last Page

Component: Hardware::ActuatorBrakingFL

Failure Mode	System Effect	Severity	Single Point of Failure
○ MechanicalFailure (1)	ActuatorBrakingFL	0	true

Component: Hardware::ActuatorBrakingFL1

Failure Mode	System Effect	Severity	Single Point of Failure
○ MechanicalFailure (8)	ActuatorBrakingFL1	0	true

Component: Hardware::ActuatorBrakingFL2

Failure Mode	System Effect	Severity	Single Point of Failure
○ MechanicalFailure (15)	ActuatorBrakingFL2	0	true

Component: Hardware::ActuatorSteeringFL

Failure Mode	System Effect	Severity	Single Point of Failure
○ MechanicalFailure (22)	ActuatorSteeringFL	0	true

Component: Hardware::ActuatorSteeringFL1

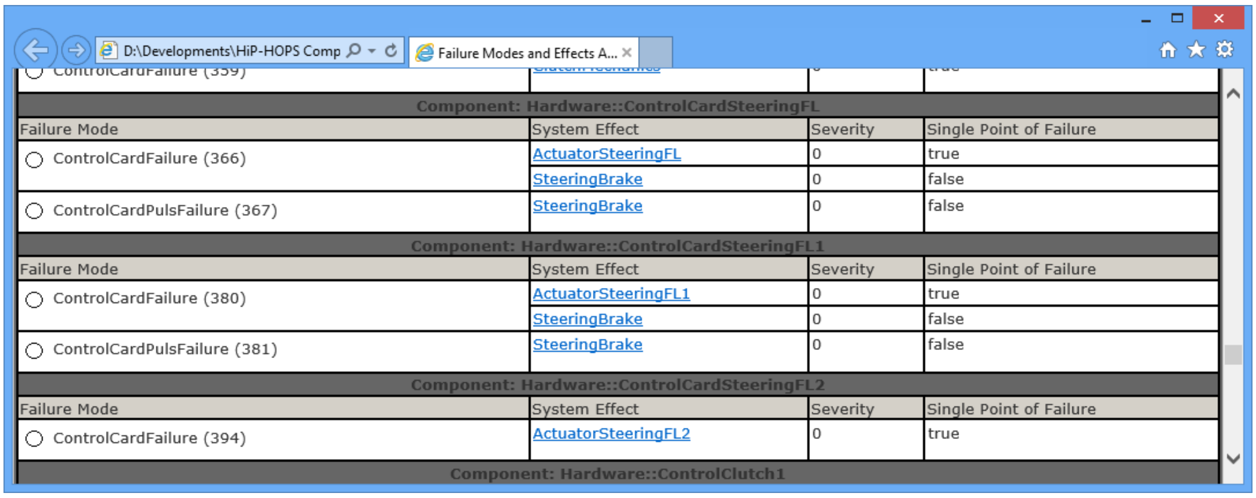
Failure Mode	System Effect	Severity	Single Point of Failure
○ MechanicalFailure (29)	ActuatorSteeringFL1	0	true

Component: Hardware::ActuatorSteeringFL2

Failure Mode	System Effect	Severity	Single Point of Failure
○ MechanicalFailure (36)	ActuatorSteeringFL2	0	true

Component: Hardware::Brake.Brake

Figure 33 – The FMEA results


D:\Developments\HiP-HOPS Comp | Failure Modes and Effects A...

ControlCardFailure (359)

Component: Hardware::ControlCardSteeringFL

Failure Mode	System Effect	Severity	Single Point of Failure
ControlCardFailure (366)	ActuatorSteeringFL	0	true
	SteeringBrake	0	false
ControlCardPulsFailure (367)	SteeringBrake	0	false

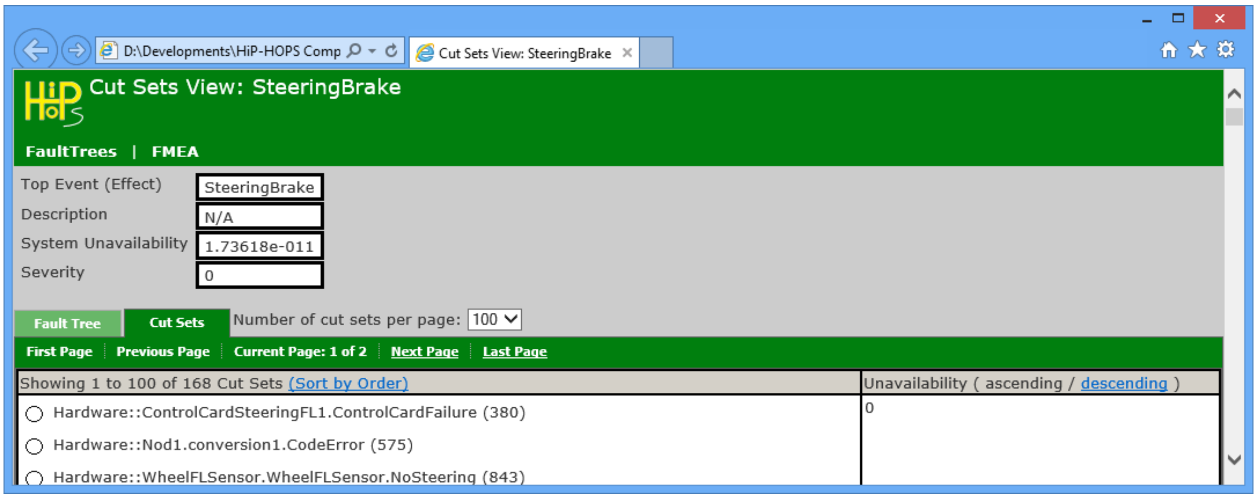
Component: Hardware::ControlCardSteeringFL1

Failure Mode	System Effect	Severity	Single Point of Failure
ControlCardFailure (380)	ActuatorSteeringFL1	0	true
	SteeringBrake	0	false
ControlCardPulsFailure (381)	SteeringBrake	0	false

Component: Hardware::ControlCardSteeringFL2

Failure Mode	System Effect	Severity	Single Point of Failure
ControlCardFailure (394)	ActuatorSteeringFL2	0	true

Component: Hardware::ControlClutch1


D:\Developments\HiP-HOPS Comp | Cut Sets View: SteeringBrake

Cut Sets View: SteeringBrake

FaultTrees | FMEA

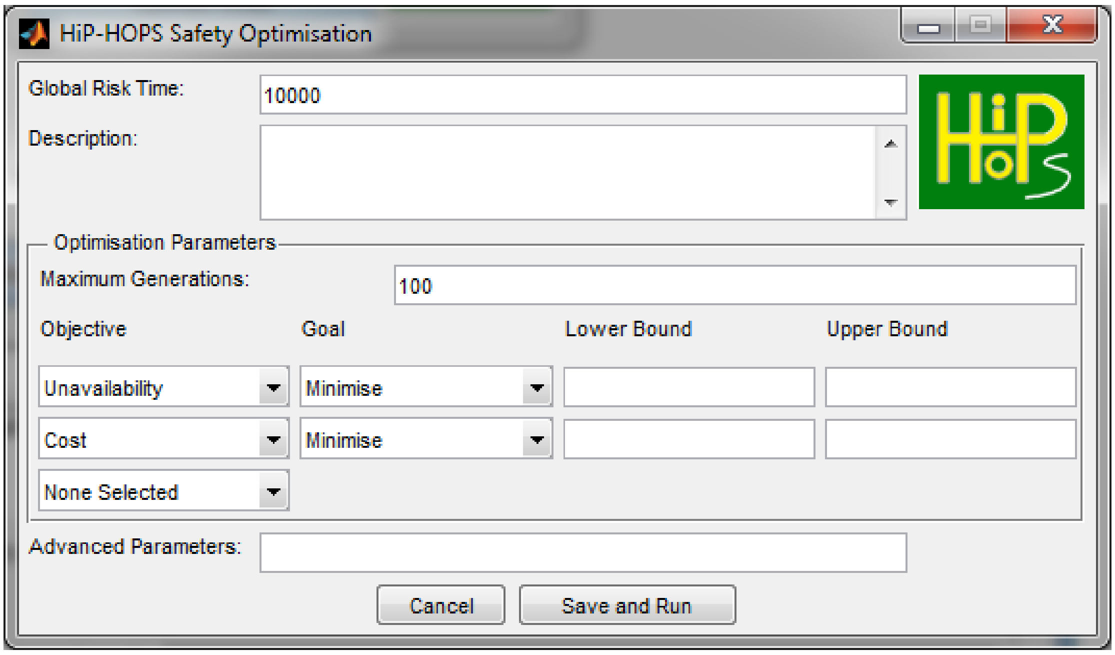
Top Event (Effect)	SteeringBrake
Description	N/A
System Unavailability	1.73618e-011
Severity	0

Fault Tree | Cut Sets | Number of cut sets per page: 100

First Page | Previous Page | Current Page: 1 of 2 | Next Page | Last Page

Showing 1 to 100 of 168 Cut Sets (Sort by Order)	Unavailability (ascending / descending)
Hardware::ControlCardSteeringFL1.ControlCardFailure (380)	0
Hardware::Nod1.conversion1.CodeError (575)	
Hardware::WheelFLSensor.WheelFLSensor.NoSteering (843)	

3.4 Performing optimisation


HiP-HOPS Safety Optimisation

Global Risk Time: 10000

Description:

Optimisation Parameters

Maximum Generations: 100

Objective	Goal	Lower Bound	Upper Bound
Unavailability	Minimise		
Cost	Minimise		
None Selected			

Advanced Parameters:

Figure 34: Optimisation Matlab

If instead of selecting analysis you choose to perform an optimisation of the model by clicking on the ‘Optimise’ button on the HiP-HOPS Launcher window, you will see the dialogue in Figure 34.

This allows you to set the parameters required for optimisation.

As with the analysis dialog you can set a global exposure to risk duration time and a description of the model.

There is an entry box for the maximum number of generations to run the optimisation for.

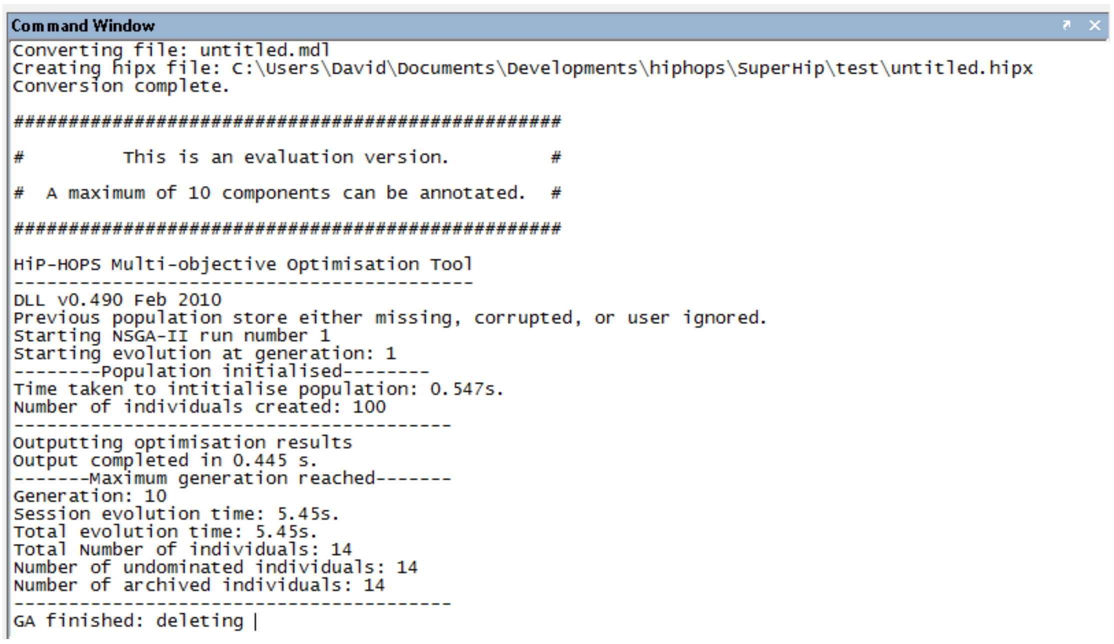
Finally you need to specify the objectives of the optimisation. You can select multiple objectives from a drop down list and then whether the goal is to minimise or maximise that objective.

In the latter two columns you can set optional upper and lower bounds for the objectives. These values provide a non-strict guide for the desired search space.

The advanced parameters field is used to specify additional HiP-HOPS exe arguments. This will normally be left blank as it would primarily be used for testing new features and helping with debugging.

Once you have filled in the settings for the optimisation you can select ‘Save and Run’. Again as with the analysis the mdl file is save and given to the converter tool to create a hip interface file. This interface file is then given to the optimisation tool and the model is optimised.

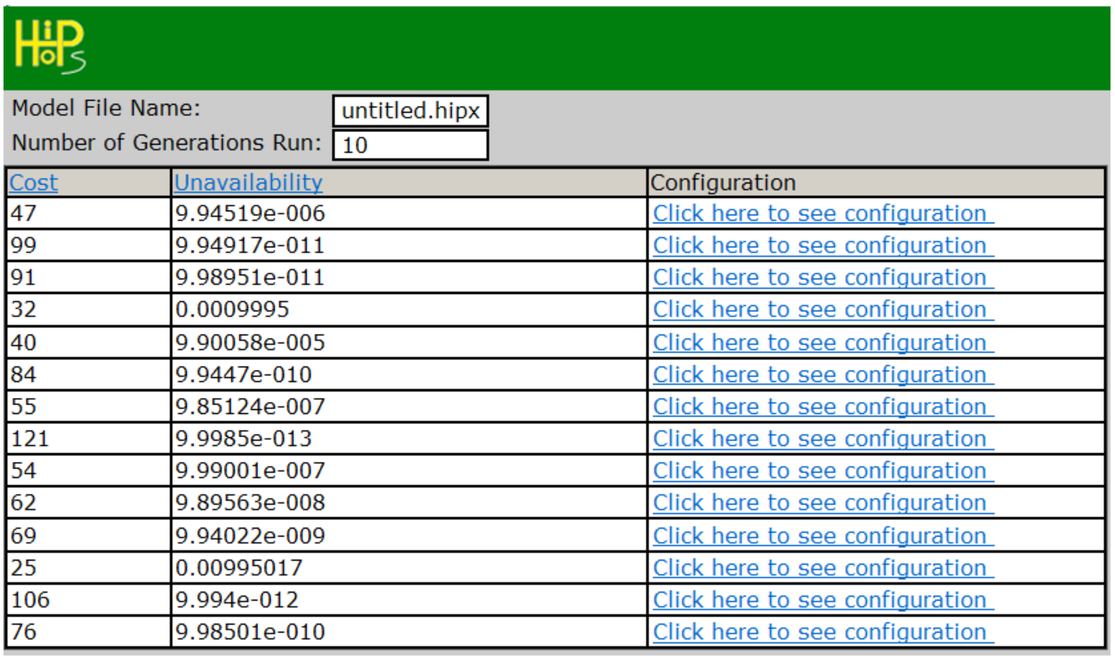
The progress of the optimisation will be displayed in the Matlab command window as in Figure 35.



```
Converting file: untitled.mdl
Creating hipx file: C:\Users\David\Documents\Developments\hiphops\SuperHip\test\untitled.hipx
Conversion complete.

##################################################

#         This is an evaluation version.         #

#  A maximum of 10 components can be annotated.  #

##################################################

HiP-HOPS Multi-objective Optimisation Tool
------------------------------------------
DLL v0.490 Feb 2010
Previous population store either missing, corrupted, or user ignored.
Starting NSGA-II run number 1
Starting evolution at generation: 1
--------Population initialised--------
Time taken to intitialise population: 0.547s.
Number of individuals created: 100
----------------------------------------
Outputting optimisation results
Output completed in 0.445 s.
-------Maximum generation reached-------
Generation: 10
Session evolution time: 5.45s.
Total evolution time: 5.45s.
Total Number of individuals: 14
Number of undominated individuals: 14
Number of archived individuals: 14
----------------------------------------
GA finished: deleting |
```

Figure 35: Optimisation progress Matlab

3.5 Viewing the optimisation results



Model File Name: untitled.hipx
Number of Generations Run: 10

Cost	Unavailability	Configuration
47	9.94519e-006	Click here to see configuration
99	9.94917e-011	Click here to see configuration
91	9.98951e-011	Click here to see configuration
32	0.0009995	Click here to see configuration
40	9.90058e-005	Click here to see configuration
84	9.9447e-010	Click here to see configuration
55	9.85124e-007	Click here to see configuration
121	9.9985e-013	Click here to see configuration
54	9.99001e-007	Click here to see configuration
62	9.89563e-008	Click here to see configuration
69	9.94022e-009	Click here to see configuration
25	0.00995017	Click here to see configuration
106	9.994e-012	Click here to see configuration
76	9.98501e-010	Click here to see configuration

Figure 36: Optimisation output

When the optimisation is complete there should be a new folder in the directory where your model file was. It will be called <modelname>-OptimisationResults. This folder contains the output of the optimisation.

HiP-HOPS will automatically display the trade-off solutions in Internet Explorer, as in Figure 36. The results are listed along with the values of each solutions objective evaluation. In the final column is a hyperlink which shows the configuration of the model required to achieve the individual result.

Following this hyperlink will display the model configuration in a tree format as in Figure 37.

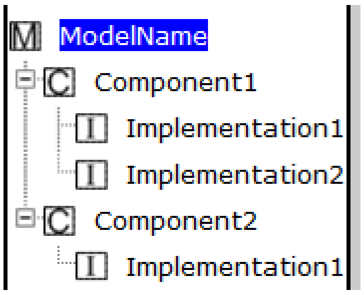


Figure 37: Solution configuration

Each configurable component in the model is listed along with the implementations required.

4 REFERENCES

[1] Pande, P. K., Spector, M. E., & Chatterjee, P. (1975). *Computerised Fault Tree Analysis: 'TREEL' and 'MICSUP'*. University Of California Operations Research Centre, California, USA.

[2] DEB, K, S AGRALWAL, A PRATAP, and T MEYARIVAN. 2000. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II. *In*: Schoenauer et AL, (ed). *Parallel Problem Solving From Nature PPSN VI*, Berlin, Germany: Springer, pp.849-858.