Virtual Satellite 4 FDIR

User Manual

Version 4.10.0, 2020-05-05T12:13:58Z

Table of Contents

Chapter 1. What is Virtual Satellite 4 FDIR?

This user manual describes how to use the Software Virtual Satellite 4 FDIR (VirSat4 FDIR). The software is an extension to Virtual Satellite 4 CORE, for a guide on how to use Virtual Satellite in general, please refer to the Virtual Satellite 4 CORE Manual.

Chapter 2. Purpose

VirSat4 FDIR focuses on modeling Fault Detection, Isolation, and Recovery. This includes among others: Models on failure behavior, recovery behavior, detection behavior, analysis results, and more. The main purpose of VirSat4 FDIR is to enable the analysis of FDIR concepts by means of mathematically well founded evaluation.

The software primarily follows the ESA ECSS standards by means of the SAVOIR FDIR Handbook (http://savoir.estec.esa.int/SAVOIRDocuments.htm). For clarification on vocabulary, further details on FDIR analysis and process, please check the SAVOIR FDIR Handbook. Accessing the handbook requires registration / login. Please note that this in return is restricted to ESA member states.

Chapter 3. Getting Started

3.1. Modeling Workflow

Learn in this section about the recommended workflow for approaching FDIR modeling & analysis. The workflow for creating fault models is up to the preference of the users. Nevertheless, basing the workflow on the following steps is recommended:

- Create a system model using the product structures concepts (PS Concept). For details check the Virtual Satellite 4 CORE User Manual.

- Create a SubSystem dedicated to Risk / FDIR.

- Create a list of Feared Events in this sub system and assign severity categories to them.

- Create a list of faults, with their basic events, for each equipment.

- Perform a Fault Tree Analysis to determine the fault propagations.

- Create the FDIRParameters category at the top of your system model and configure it.

- Create FDIR analysis categories for the desired faults

- Refine the system model and the Fault Tree Analysis and update the FDIR analysis

In the following sections, the various categories, means of analysis, etc. will be elaborated.

Chapter 4. Fault Modeling

Fault modeling forms the core of VirSat4 FDIR and is the primary activity required to perform any FDIR analysis. Learn in this section how to use the graphical editor to perform the main analysis of VirSat4 FDIR, Fault Tree Analysis (FTA), and how to use it to build up fault models.

4.1. Modeling Fault Trees

Faults, their propagations, and inhibiting fault propagation through means of FDIR is modeled using fault trees. Fault trees are graphical models describing how faults combine with each other, propagate through the system, and eventually turn into a feared event. A fault tree is usually constructed top-down, but is read bottom-up, starting with initiating basic events. The recombination of faults is modeled via so-called gates, such as "AND" and "OR". In this section, you will learn the basics on fault tree analysis, which gates are supported by the software, and how to use the graphical editor to create a fault tree. For further in-depth information on how to perform an FTA, we refer to the standards.

4.1.1. Fault Tree Elements

The following contains a comprehensive list of fault tree elements supported by VirSat FDIR. The element descriptions are structured in the following format:

[Name of the fault tree element] **[Icon]**

— **Description**

[Purpose of the element, its propagation behavior, additional parameters, etc.]
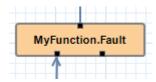
— **Graphical representation**

[Representation in the graphical editor]

In addition to its inherent parameters, each fault tree node also has a name, a list of inputs, and a list of outputs. For gates, the name is by default the type of the node.

Fault 🎛️

— **Description**

Faults represent logical, named events. They are used either to represent a *top-level event* of a fault tree or an *intermediate event*. Faults are also the logical containers for all other fault tree elements. As such, deleting a fault also deletes all contained elements such as gates, propagations, analysis information, etc. A fault always propagates if at least one input fails. It is, however, recommended to only have one input per fault (plus basic events).

— **Graphical representation**

BasicEvent ⬤

— **Description**

Basic events typically form the leaf elements of a fault tree. They represent basic anomalies that are not further broken down in the course of the fault tree analysis. In practice, basic events most commonly correspond to causes of equipment failure. A basic event is always directly associated to a fault. All fault propagations in a fault tree ultimately originate from basic events. Ideally, a basic event is equipped with a failure rate, quantifying its likelihood of occurrence over time. The failure rate states how often the basic event is expected to occur within a time unit. Optionally, it may also be equipped with a repair rate which conversely captures the likelihood of repair over time. Finally, a basic event may also be equipped with a cold failure rate, which comes into play when interacting together with the SPARE gate.

— **Graphical representation**

Propagation ⌶

— **Description**

Fault propagations are the edges of a fault tree, and connect the fault tree nodes. A fault propagation has a direction. It connects the output of a fault tree node with the input of another fault tree node. Since fault trees are acyclic graphs, fault propagations may not create any cycles.

— **Graphical representation**

AND ⬠

— **Description**

A gate that propagates if all inputs have failed.

— **Graphical representation**

OR ⬡

— **Description**

A gate that propagates if at least one input has failed.

— **Graphical representation**

VOTE

— **Description**

A gate, that is also equipped with a voting threshold k, and propagates if at least k inputs have failed. The voting threshold has to be at least 1.
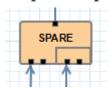
— **Graphical representation**

SPARE

— **Description**

A gate with two types on inputs: Primaries and spares. If at least one primary input fails, the SPARE gate activates and claims one of the spares. Should no spares be available or failed, then the SPARE gate propagates. All spares are considered to be dormant. This means that contained basic events will use their cold failure rate, instead of their hot failure rate, as long as they are unclaimed. Once a spare is claimed, it is set to be activated and its hot failure rate is used again. Spares are claimed from left to right. In the case of a repair, the SPARE gate switches back. Spares may be shared between spare gates. However, there must not be common nodes between spares or between spares and primaries. The only exception of this rule, are functional dependency gates.

— **Graphical representation**

POR

— **Description**

A Priority OR (POR) gate propagates if the left-most input occurs before any other input.

— **Graphical representation**

PORI

— **Description**

An Inclusive Priority OR (PORI) gate propagates if the left-most input occurs before any other input, or at the same time as another input.

— Graphical representation

PAND 🔶

— Description

A Priority AND (PAND) gate propagates if the inputs fail exactly in sequence from left to right.

— Graphical representation

PANDI 🔶

— Description

An Inclusive Priority AND (PANDI) gate propagates if the inputs fail exactly in sequence from left to right, or at the same time.

— Graphical representation

SAND 🔶

— Description

A Simultaneous AND (SAND) gate propagates if all inputs fail at the same time.
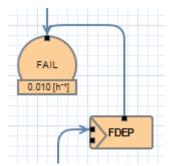
— Graphical representation

FDEP 🔶

— Description

The functional dependency (FDEP) gate allows to trigger basic events. In the event of any input event occurring, all connected basic events get triggered.
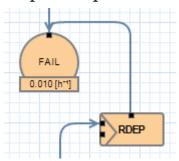
— Graphical representation
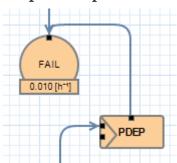
RDEP

— **Description**

The rate dependency (RDEP) gate allows to increase the failure rate of a basic event. An RDEP is equipped with a rate change parameter r. In the event of any input event occurring, the failure rate of all connected basic events is multiplied by r.

— **Graphical representation**

PDEP

— **Graphical representation**

— **Description**

The probability dependency (PDEP) gate allows to trigger basic events. The PDEP gate is equipped with a trigger probability p. In the event of any input event occurring, each connected basic event is triggered with probability p.

DELAY

— **Description**

The DELAY gate can be used to describe time delays in propagation. The gate is equipped with a delay parameter d. Propagation occurs if any input fails and remains failed for a duration of d time units. If the failed inputs are repaired before the DELAY gate perfoms a propagation, then the propagation process is stopped.
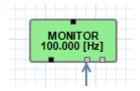
— **Graphical representation**

MONITOR 🖵

— **Description**

The MONITOR gate is used in fault trees where not all events are observable. If a monitor gate is used, the semantics of fault occurrence are changed as follows: By default, the observation of a basic event or any further propagated gate, is not guaranteed. In order for it to be observed it must propagate to an observation input of a MONITOR gate. A MONITOR gate has two types of inputs: Fail inputs and observation inputs. Fail inputs work as usual, in the event of any fail input occurring, the MONITOR gate fails as well and propagates. Failed MONITOR gates may no longer perform any observations. In the case of an observation input occurring, the event is marked as observed. Only then can reactive gates such as SPARE gates react and claim a spare. The recovery actions in partial observable fault trees are managed by recovery automata. A MONITOR gate may also be equipped with an observation rate o. In the event of it being 0, all observation events are observed immediately. If the observation rate is non-zero, all observation events are observed with time delay o.
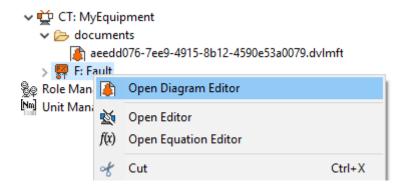
— **Graphical representation**

4.1.2. The graphical Fault Tree Editor

Besides the usual table based user interface, VirSat FDIR offers a graphical diagram editor interface. Learn in this section how to create new fault tree diagrams and how to use them for building fault tree models. Fault tree diagrams can also be used to simply visualize existing fault tree models.
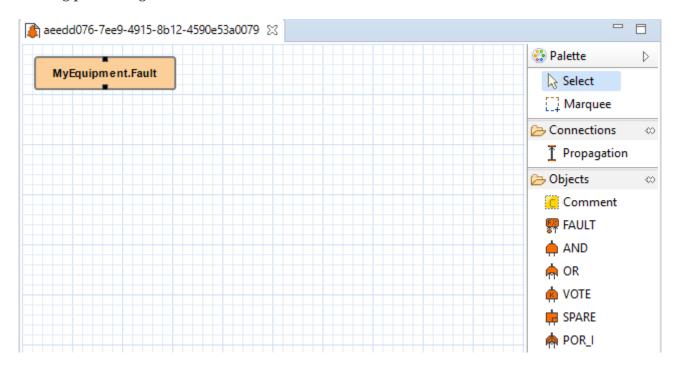
Creating a new Fault Tree Diagram

Each fault tree diagram is associated with a fault. The fault locally constitutes a top-level event. Note that on a system wide level, the fault might be just an intermediate event. A diagram may also contain multiple top-level events, but it is not recommended, as all elements inserted into a fault tree diagram are automatically associated with the corresponding fault corresponding to the diagram.

A new diagram can be created by selecting a fault in the navigator and then choosing **Context › Open Diagram Editor**. A new editor window named according to the UUID of the selected fault will pop up and also automatically contain the selected fault. The diagram editor can be opened again using the same process. The file of the newly created diagram can be found in the *documents* folder of the structural element instance the fault is attached to.

Basic Usage

This section introduces the basic concepts needed to operate the diagram editor. The diagram editor is based on the same technology as other Virtual Satellite diagrams. Previous experience with Virtual Satellite diagram editors should at least partially translate. The diagram editor consists of two main areas shown in the figure below: The actual modeling canvas (left-hand side) and the modeling palette (right-hand side).

Elements can be placed in the canvas by selecting an element type in the palette, and then left-clicking on the canvas. Alternatively, existing elements can be added to the canvas by drag & dropping them from the navigator view. This is useful for example, for referencing other, already existing faults. For further documentation on the editoring capabilities provided by the underlying technology, please consult the official documentation available at https://www.eclipse.org/graphiti/documentation/.

Connecting elements

Each element in the canvas is equipped with input and output ports. These can be connected via the propagation element. The editor only allows creating propagations that start in an input port and end in an output port. Propagations can also be reconnected by selecting the desired port and moving the propagation end to the new desired port. However, the same restrictions apply as when creating new propagations. If a port is used, a new port of the respective type will be generated. Likewise, if a used port becomes unused, other free ports of the type are automatically deleted.

Deleting an element with incoming and outgoing propagations also deletes those propagations, if the user has the necessary rights to delete them.

Finally, it is also possible to insert into a propagation. This redirects the propagation to end in the input port of the newly added element. Additionally, a new propagation is created connecting the output of the newly created element to the input of the old end of the propagation.
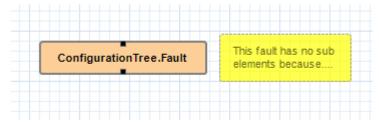
Diagram Specific Features

In addition to the normal diagram and modeling functionalities, the fault tree diagram offers some additional features. Their usage and functions are explained in this section.

Comment 🔲

— **Description**

A comment contains a simple multi-line text. Line breaking can be achieved by pressing `SHIFT`+`ENTER`. It can be used to add clarity to the model, describe the purpose behind potentially difficult to understand fault tree constructs, etc. Unlike other elements in the palette, comments do not have a model representation in Virtual Satellite. They only exist in the diagram. This also means, that if a diagram is deleted, then so are the contained comments.
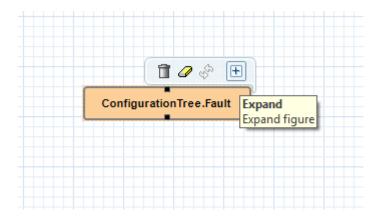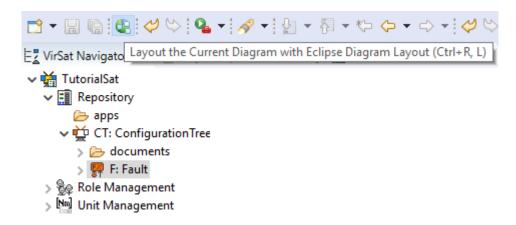
— **Graphical representation**

Collapse and Expand

— **Description**

The collapse and expand operations aim to simplify the management of the logical level of detail in a fault tree diagram. They are available only for faults. Gates cannot be expanded or collapsed. Expanding a fault means to add all directly contained fault tree elements into the diagram. Likewise, collapsing a fault removes all elements directly contained in the fault from the diagram. If a fault is not fully expanded, then the expand operation is shown in the user interface. If a fault is fully expanded, the collapse operation is shown. Performing either of the operations leads to a automatic layouting of the fault tree diagram.
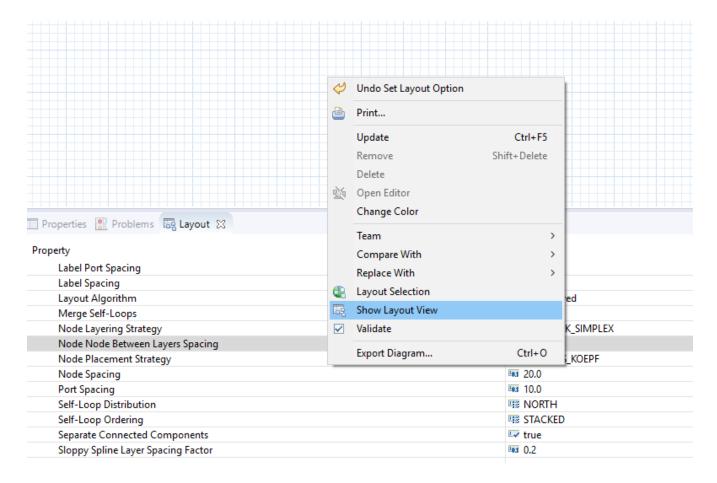
— **Graphical representation**

Using the Auto Layout functionality

The fault tree editor provides the functionality to auto layout diagrams. Auto layouting them makes it so that the top-level elements are located at the top, and lower level elements are located further to the bottom. The auto layout can be invoked by pressing the auto layout bottom located at the top-right above the navigator.

For advanced users, it is possible to customize the layouting. This can be done by editing the layouting properties accessible via the layout view. The layout view is opened by right-clicking on the diagram canvas and selecting **Show Layout View**.

For example, the default minimum distance between two node levels can be adjusted via the **Node Node Between Layers Spacing** property. For further documentation regarding the auto layout functionality in general, and the available customization options, please refer to the official documentation available at https://www.eclipse.org/elk/reference.html.

4.2. Modeling Detection

Fault trees by default are considered to be fully observable. This means, that e.g. SPARE gates can immediately react to the occurrence of basic events. Adding a MONITOR gate to a fault tree turns the model into a partial observable fault tree. In this model, only events that are directly linked to the observation input for a MONITOR gate, or that can be logically derived, can be observed. For example, if all inputs to an AND gate are observed, then the AND gate is observable as well. Using MONITOR gates is absolutely necessary, if the user wishes to perform analysis on the observation behavior and times.

Partial observable fault trees with SPARE gates require a recovery automaton to be defined, otherwise their behavior is undefined. On the other hand, partial observable fault trees with static gates only (e.g. AND, OR, VOTE, etc.) do not require a recovery automaton to be specified.

4.3. Data Exchange With the GALILEO Format

Connecting external fault trees with Virtual Satellite trees is possible is possible with the GALILEO file format. Virtual Satellite can import and export fault trees into this textual format, extended by the node types supported in Virtual Satellite. The GALILEO file format is a simple fault tree format, and further descriptions on its syntax can be found at https://www.cse.msu.edu/~cse870/Materials/

. Exporting and importing a fault tree can be done using the **Galileo DFT Export** and **Galileo DFT Import** wizard, respectively. The wizards are available under **File › Export › FDIR** and **File › Import › FDIR**.

For identification, the exchange uses the UUIDs as identifiers. This means that importing a GALILEO fault tree with some specified names will create appropriately named fault trees in Virtual Satellite. However, when re-exporting the fault tree to the GALILEO format, Virtual Satellite will use the UUIDs, giving a different output. Should the user decide to modify the fault tree and re-import, Virtual Satellite can identify existing fault tree elements via the UUIDs. VirSat FDIR also ships with a very simple GALILEO file format textual editor meant for simple viewing and editing of GALILEO fault trees. It is automatically used when opening a file with the *.dft* extension. And example of the export output created by VirSat and viewed with the textual editor is given below.

Chapter 5. Recovery Modeling

Recovery behavior can be modeled using **recovery automata**. They specify what recovery actions should be executed upon occurrence of a fault. Recovery automata are necessary when dealing with fault trees with complex recovery behavior. This includes especially partial observable fault trees with MONITOR gates. Learn in this section how the recovery automaton model is structured, and how to manage it.

5.1. Modeling Recovery Automata

A recovery automaton (RA) is a finite, deterministic state machine without timed transitions. An RA contains is constituted of two main elements: States and transitions. States represent a recovery internal state of knowledge. Each RA must have an initial state specified. Transitions state the actual recovery behavior. They are labeled with a list of **guards** that states the condition that triggers the transition and a list of recovery actions, which are then executed. RAs operate under maximum progress assumption, i.e., whenever a transition is enabled it must be taken.

5.1.1. Recovery Automata Elements

The following section gives a more in-depth description of the available elements for creating recovery models. It is structured similarly to the description of the fault tree elements using the format:

***[Name of the recovery automaton element]* [Icon]**

— **Description**

[Purpose of the element, its behavior, additional parameters, etc.]
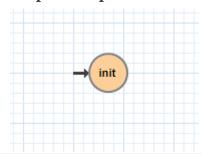
— **Graphical representation**

[Representation in the graphical editor]

If there is no applicable graphic representation, the item is left out. Like all Virtual Satellite elements, in addition to its parameters, every recovery element also has a name.

***State* 🔳**

— **Description**

A state is recovery internal information. Changing a state changes the recovery behavior. If possible, states should be given meaningful names. Every recovery automaton has an initialState property.
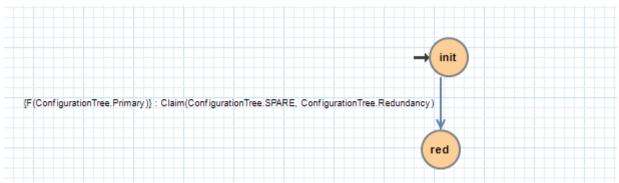
— **Graphical representation**

FaultEventTransition ⟱

— **Description**

A fault event transition is a transition that has fault tree events listed in the guards. That means, it is triggered if exactly the specified nodes in the fault tree fail at the same time. As a transition, is is also equipped with a list of recovery actions, executed upon triggering the transition.
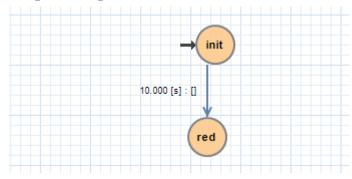
— **Graphical representation**

TimedTransition ⟱

— **Description**

A timed transition has a `time` property. It is used as the guard for the transition. As a transition, is is also equipped with a list of recovery actions, executed upon triggering the transition. Every state may be equipped with at most **one** timed transition. If multiple are declared, only the one with the smallest time property is used due to the maximum progress assumption.

— **Graphical representation**

FreeAction ⟰

— **Description**

The free action is a recovery action, describing that all claims on a specified spare should be cleared.

ClaimAction ⟰

— **Description**

THe claim action is a recovery action, describing that some SPARE gate should claim the specified spare.

5.1.2. The Graphical Recovery Automaton Editor

Creating a New Recovery Automaton Diagram

Basic Usage

Using the Auto Layout functionality

Chapter 6. FDIR Analysis

6.1. Configuring Analysis information

6.2. Qualitative Analysis

6.3. Quantitative Analysis

6.4. Using STORM

Chapter 7. FDIR Reporting

7.1. Using the SAVOIR/FDIR Report Template

7.2. Excel Exports

Chapter 8. FDIR Synthesis

8.1. Fault Tree Generation

8.2. Recovery Automata Synthesis

Legal - License & Copyright

Product Version:	4.10.0
Build Date Qualifier:	2020-05-05T12:13:58Z
Travis CI Job Number:	

Copyright (c) 2008-2020 DLR (German Aerospace Center), Simulation and Software Technology. Lilienthalplatz 7, 38108 Braunschweig, Germany

This program and the accompanying materials are made available under the terms of the Eclipse Public License 2.0 which is available at https://www.eclipse.org/legal/epl-2.0/ . A copy of the license is shipped with the Virtual Satellite software product.