Project Number 101017258

D4.6 Tools for Automated Safety Analysis of MRS and for Production of EDDIs (Final Version)

Version 1.0
July 2023
Final

Public Distribution

University of Hull and Fraunhofer IESE

Project Partners: **Aero41, ATB, AVL, Bonn-Rhein-Sieg University, Cyprus Civil Defence, Domaine Kox, FORTH, Fraunhofer IESE, KIOS, KUKA Assembly & Test, Locomotec, Luxsense, The Open Group, Technology Transfer Systems, University of Hull, University of Luxembourg, University of York**

Every effort has been made to ensure that all statements and information contained herein are accurate, however the SESAME Project Partners accept no liability for any error or omission in the same.

© 2023 Copyright in this document remains vested in the SESAME Project Partners.

PROJECT PARTNER CONTACT INFORMATION

Aero41 Frédéric Hemmeler Chemin de Mornex 3 1003 Lausanne Switzerland E-mail: frederic.hemmeler@aero41.ch	**ATB** Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany E-mail: scholze@atb-bremen.de
AVL Martin Weinzerl Hans-List-Platz 1 8020 Graz Austria E-mail: martin.weinzerl@avl.com	**Bonn-Rhein-Sieg University** Nico Hochgeschwender Grantham-Allee 20 53757 Sankt Augustin Germany E-mail: nico.hochgeschwender@h-brs.de
Cyprus Civil Defence Eftychia Stokkou Cyprus Ministry of Interior 1453 Lefkosia Cyprus E-mail: estokkou@cd.moi.gov.cy	**Domaine Kox** Corinne Kox 6 Rue des Prés 5561 Remich Luxembourg E-mail: corinne@domainekox.lu
FORTH Sotiris Ioannidis N Plastira Str 100 70013 Heraklion Greece E-mail: sotiris@ics.forth.gr	**Fraunhofer IESE** Daniel Schneider Fraunhofer-Platz 1 67663 Kaiserslautern Germany E-mail: daniel.schneider@iese.fraunhofer.de
KIOS Panayiotis Kolios 1 Panepistimiou Avenue 2109 Aglatzia, Nicosia Cyprus E-mail: kolios.panayiotis@ucy.ac.cy	**KUKA Assembly & Test** Michael Laackmann Uhthoffstrasse 1 28757 Bremen Germany E-mail: michael.laackmann@kuka.com
Locomotec Sebastian Blumenthal Bergiusstrasse 15 86199 Augsburg Germany E-mail: blumenthal@locomotec.com	**Luxsense** Gilles Rock 85-87 Parc d'Activités 8303 Luxembourg Luxembourg E-mail: gilles.rock@luxsense.lu
The Open Group Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium E-mail: s.hansen@opengroup.org	**Technology Transfer Systems** Paolo Pedrazzoli Via Francesco d'Ovidio, 3 20131 Milano Italy E-mail: pedrazzoli@ttsnetwork.com
University of Hull Yiannis Papadopoulos Cottingham Road Hull HU6 7TQ United Kingdom E-mail: y.i.papadopoulos@hull.ac.uk	**University of Luxembourg** Miguel Olivares Mendez 2 Avenue de l'Universite 4365 Esch-sur-Alzette Luxembourg E-mail: miguel.olivaresmendez@uni.lu
University of York Simos Gerasimou & Nicholas Matragkas Deramore Lane York YO10 5GH United Kingdom E-mail: simos.gerasimou@york.ac.uk nicholas.matragkas@york.ac.uk	

DOCUMENT CONTROL

Version	Status	Date
0.1	Initial outline	5 June 2023
0.2	Added new UOH content	12 June 2023
0.3	Incorporated input from IESE on the ODE Updater	19 June 2023
0.4	Added information about ODE update; expanded KIOS example	20 June 2023
0.5	Further minor updates	22 June 2023
0.6	Review ready version	26 June 2023
0.7	Updated after review feedback	3 July 2023
0.9	Final version for QA	4 July 2023
1.0	Final QA version	5 July 2023

TABLE OF CONTENTS

TABLE OF FIGURES

EXECUTIVE SUMMARY

The Executable Digital Dependability Identity — or EDDI — is a composable model-based artefact that contains dependability information about a system. The EDDI concept is intended for dynamic dependability management at runtime. However, in order to generate runtime EDDIs, relevant information must be captured at design time.

It is these design-time activities that are covered in this deliverable. Existing safety analysis tools are targeted to create appropriate system models and safety artefacts, which can then be converted to ODE-compliant models via tool adapters and model converters. The tools, adapters, and converters are described within, along with information about how they can be used.

To demonstrate the use of the tools, the HiP-HOPS tool is applied to the Cyprus Civil Defence/KIOS power station inspection case study.

LIST OF ABBREVIATIONS

ACCF	Actual Common Cause Failure
BE	Basic Event (i.e., root cause of a fault tree)
BN	Bayesian Network
CCF	Common Cause Failure
CFT	Component Fault Tree
DDI	Digital Dependability Identity
DFT	Dynamic Fault Tree
EDDI	Executable Digital Dependability Identity
FHA	Functional Hazard Analysis
FMEA	Failure Modes & Effects Analysis
FMEDA	Failure Modes, Effects, & Diagnostic Analysis
FTA	Fault Tree Analysis
HARA	Hazard Analysis & Risk Assessment (or Hazard And Risk Analysis)
MAS	Multi-Agent System
MBSA	Model-based Safety Analysis
ML	Machine Learning
MRS	Multi-Robot System
ODE	Open Dependability Exchange metamodel
PCCF	Potential Common Cause Failure
SACM	Structured Assurance Case Metamodel

1. INTRODUCTION

The Executable Digital Dependability Identity — or EDDI — is a composable model-based artefact that contains dependability information about a system. The EDDI concept is primarily aimed at runtime usage, where the EDDIs are intended to be executed onboard or alongside their target system to perform dynamic dependability management.

However, in order to generate runtime EDDIs, relevant information must be captured at design time. Furthermore, EDDIs can also serve as purely design-time artefacts, storing dependability information about a system. In this form, they can serve as safety argumentation and can support additional design-time dependability analyses. It is these design-time activities that are covered in this deliverable; the runtime aspects are detailed further in the various WP7 deliverables:

- **D7.1: Runtime Safety and Security Concept – EDDI Runtime Model Specification**
- **D7.2: Tools for Generation of Runtime EDDIs**.
- **D7**.3: **Runtime Safety and Security Concept – EDDI-based MAS & communication**
- **D7.4: Open Source Components for Explainable EDDIs**

EDDIs are based on the Open Dependability Exchange (ODE) metamodel. The ODE is intended to be the common interchange format between the different EDDI-related tools so that common models can be created and used to generate or interact with runtime EDDIs regardless of the original design-time tool. The updated ODE is defined in **D4.2/D5.2: Safety/Security ODE and EDDI Specification** (though please check section 5.1 for more recent changes to the ODE).

To support design-time EDDI activities such as system modelling and safety analysis, three primary existing safety analysis tools are being targeted: HiP-HOPS, safeTbox, and Dymodia. These tools have been chosen for their good support for model-based safety analysis. Additionally, in the case of the first two, we develop them ourselves and thus can more readily make any necessary changes. All three are tools that allow the creation or import of architectural models of a system which can then be annotated with failure data to record the failure behaviour of the system. The data can then be analysed to produce safety analysis artefacts such as fault trees and FMEAs. More general information about each tool and the principles that underpin them can also be found in **D4.5: Safety Analysis Concept & Methodology**.

Although not a safety analysis tool per se, we also target the GeNIe tool for Bayesian network analysis, as described in Section 6.

To ensure these tools can be integrated into the EDDI toolchain, their models need to be convertible to the ODE in order to facilitate exchange with other tools. For this purpose, we have developed a number of ODE-related tools, including a Common Tool Adapter and a standalone converter & editor (the EDDI Editor). These are discussed in section 6.

Finally, to demonstrate how the tools can be applied, an example based on the Cyprus Civil Defence/KIOS power station inspection use cases is provided in section 7. This follows on from a high-level example based on the same use case found in section 6 of **D4.5 Safety Analysis Concept and Methodology**.

1.1 UPDATES SINCE D4.4

1.1.1 Response to reviewers

As D4.6 is the 'final' version of the interim deliverable D4.4, we wished to address the valuable feedback received during the M18 midterm review, which focused on the following issues. We have tried to address this feedback wherever possible in this new version of the deliverable. Our rationale is also provided below.

1. **Clarify difference between EDDIs and DDIs**

    - This confusion likely arose due to outdated terminology in the technical description of some of the tools. This has been updated, as the EDDI essentially replaces the older DDI concept, which was primarily a design-time concept. The plan to extend the DDI to runtime is what led to the EDDI.

2. **Conduct usability evaluation of the ODE metamodel**

    - Usability is certainly an important attribute and one that we do wish to address. However, the ODE itself is a metamodel intended primarily to act as a superset of the information present in different processes and tools, thereby allowing it to act as a repository and common interchange format, rather than something directly used by an end user, like a DSL. The ODE is instead primarily relevant to the developers of analysis tools and model converters.

    - Nevertheless, we plan to include usability as one of the criteria in the use case evaluations taking place during the last phase of the project. These evaluations will look at tool usability and will attempt to assess whether the EDDI methodology, including the use of models enabled by the ODE, offers an improvement on existing dependability processes in use by industrial partners.

3. **Consider moving some technical content in the model converter chapter to code repository documentation**

    - Parts of Section 5 have been moved to specific user guides for the Common Tool Adapter and the EDDI Editor.

4. **To what extent is tool heterogeneity supported?**

    - Tool heterogeneity is supported by the Common Tool Adapter and the EDDI Editor. The former allows models to be transformed to and from the ODE interchange format, while the latter allows models from different tools to be combined as part of a single ODE model.

1.1.2 Summary of changes

As a 'final' version of an earlier deliverable, D4.6 uses the earlier document as a base to build upon. Changes from D4.4 include:

- Section 2.4 has been added to cover the changes to HiP-HOPS that have been implemented after the first round of use case evaluations. HiP-HOPS screenshots earlier in Section 2 have also been updated.

- Section 5.1 has been added to describe minor changes made to the ODE since the publication of **D4.2/D5.2 ODE Specification**.

- Parts of Section 5.2 on the Common Tool Adapter have been moved to a separate user guide.

- Section 0 has been added to describe the new ODE Updater tool, which facilitates updates to the Common Tool Adapter after changes to the ODE.

- Section 5.4 replaces the earlier section on the ODE Converter with a new section on the EDDI Editor, which is derived from the earlier converter and provides more functionality to support tool heterogeneity. Further information is also provided in a separate user guide.

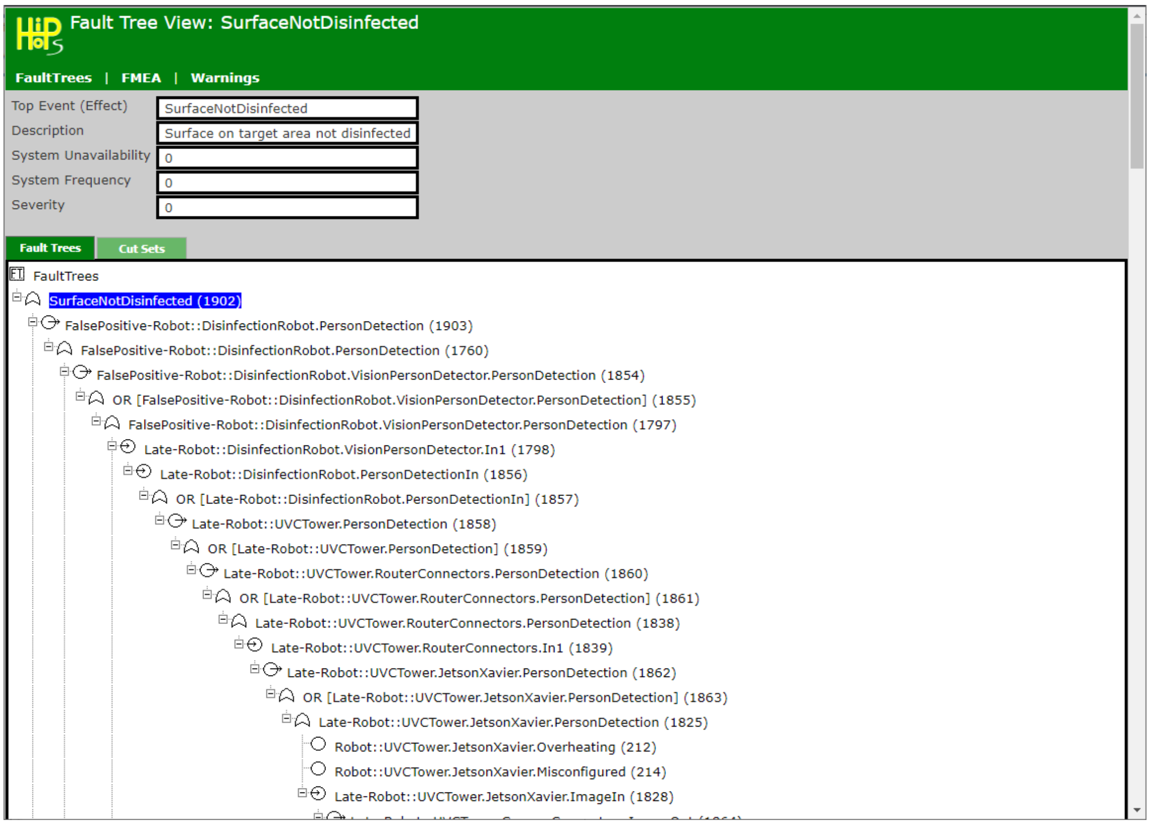
- Section 7.1 has been rewritten. The previous Locomotec example in D4.4 has been replaced by a new example describing the process of using HiP-HOPS to model and analyse the KIOS/Cyprus Civil Defence use case and produce a design-time EDDI.

2. HiP-HOPS

2.1 What is HiP-HOPS

Hierarchically Performed Hazard Origin & Propagation Studies, or HiP-HOPS[1] to use its more convenient name, is a comprehensive model-based safety analysis methodology with a tool of the same name.

Originally created in the late 1990s [1], it has been the focus of continuous development at the University of Hull over the ensuing 20+ years. Over that time it has evolved from a simple fault tree analysis tool to a powerful dependability engine with a wide range of capabilities and additional functionalities. These include multiple failure mode FMEAs, optimisation capabilities (for architecture, maintenance, and safety requirements) [2], dynamic analysis (using state machines and/or dynamic fault trees) [3], and automatic allocation of safety integrity levels according to safety requirements to support ISO 26262-style workflows [4].


Fault Tree View: SurfaceNotDisinfected

FaultTrees | FMEA | Warnings

Top Event (Effect)	SurfaceNotDisinfected
Description	Surface on target area not disinfected
System Unavailability	0
System Frequency	0
Severity	0

Fault Trees | Cut Sets

```
FaultTrees
SurfaceNotDisinfected (1902)
  FalsePositive-Robot::DisinfectionRobot.PersonDetection (1903)
    FalsePositive-Robot::DisinfectionRobot.PersonDetection (1760)
      FalsePositive-Robot::DisinfectionRobot.VisionPersonDetector.PersonDetection (1854)
        OR [FalsePositive-Robot::DisinfectionRobot.VisionPersonDetector.PersonDetection] (1855)
          FalsePositive-Robot::DisinfectionRobot.VisionPersonDetector.PersonDetection (1797)
            Late-Robot::DisinfectionRobot.VisionPersonDetector.In1 (1798)
              Late-Robot::DisinfectionRobot.PersonDetectionIn (1856)
                OR [Late-Robot::DisinfectionRobot.PersonDetectionIn] (1857)
                  Late-Robot::UVCTower.PersonDetection (1858)
                    OR [Late-Robot::UVCTower.PersonDetection] (1859)
                      Late-Robot::UVCTower.RouterConnectors.PersonDetection (1860)
                        OR [Late-Robot::UVCTower.RouterConnectors.PersonDetection] (1861)
                          Late-Robot::UVCTower.RouterConnectors.PersonDetection (1838)
                            Late-Robot::UVCTower.RouterConnectors.In1 (1839)
                              Late-Robot::UVCTower.JetsonXavier.PersonDetection (1862)
                                OR [Late-Robot::UVCTower.JetsonXavier.PersonDetection] (1863)
                                  Late-Robot::UVCTower.JetsonXavier.PersonDetection (1825)
                                    Robot::UVCTower.JetsonXavier.Overheating (212)
                                    Robot::UVCTower.JetsonXavier.Misconfigured (214)
                                    Late-Robot::UVCTower.JetsonXavier.ImageIn (1828)
```

Figure 1 - HiP-HOPS analysis output

2.2 How it works

A full manual is available on the HiP-HOPS website. However, a shorter explanation on the core HiP-HOPS functionality is provided below.

A standard HiP-HOPS analysis consists of four main phases plus an extra optional step:

[1] https://hip-hops.co.uk

1. **System modelling**. In this phase, an external modelling package such as Matlab Simulink[2], SimulationX[3], or MetaEdit+ (with EAST-ADL)[4] is used to create a model of the system architecture — i.e., components and the connections between them. This provides the basis for the origin of failure modes and the channels along which they can propagate.

2. **Failure annotation**. Using a specific interface or HiP-HOPS plugin to one of the tools above, the system model is annotated with additional information to describe the system's failure behaviour. Components are annotated with potential failure modes — failures that originate within them, or which are first detectable via them — as well as how deviations of their outputs are caused either by these internal failure modes or deviations received at their inputs. Probabilistic information can also be added to these failure modes if available. Further logic can be added to describe the propagation of failures between components if required. Finally, system-level hazards are also identified and linked to the potential component output deviations that could cause them.

3. **Synthesis of fault propagation models**. With the model complete, it is passed to the HiP-HOPS engine itself. The tool begins by generating a fault propagation model: working backwards from each identified hazard, HiP-HOPS works its way through the system by following failure propagations and establishes all the possible component-level causes of that hazard. The result is a network of interconnected fault trees.

4. **Failure analysis**. Having generated fault trees in the previous step, HiP-HOPS can now analyse them. It does this by removing redundancies, resolving contradictions, and simplifying causes until it achieves the minimal set of possible causes for each hazard, along with an estimate for unavailability if the necessary data was provided. From these it can also generate a system-wide multiple failure mode FMEA, which indicates the potential hazards each failure mode can lead to.

5. **Optional extra steps.** If desired, and if appropriate data is present, these analysis results can also be used as the basis of other activities, e.g. architectural optimisation or allocation of safety requirements.

Each of these first four steps will be described further below.

2.2.1 System Modelling

A HiP-HOPS-compatible system model requires two main elements: components and connections. Components represent functional system elements. They do not necessarily have to be physical hardware components; software components and abstract functions are also possible components, and indeed it is not uncommon for different models to be created at different stages of the design lifecycle, e.g. beginning with a basic functional design architecture and then progressing through to a joint hardware/software architecture later. HiP-HOPS provides the concept of 'perspectives', which are essentially top-level subsystems, to separate out different forms of components if

[2] https://www.mathworks.com/products/simulink.html
[3] https://www.esi-group.com/products/system-simulation
[4] https://www.metacase.com/solution/east-adl.html

required. This allows, for instance, separate modelling of hardware and software components while still allowing software to be allocated to the hardware that executes it.

The other main elements are the connections between components. Depending on the modelling tool, these can vary from simple unidirectional point-to-point connections to more complicated undirected lines connecting multiple components with their own propagation logic. Connections connect to components via ports, which represent the interface between a component and the rest of the system.

As the name would suggest, hierarchical modelling is also possible. Any component can have a subsystem within it, which can host more components and more connections. HiP-HOPS can automatically combine failures that originate from a component's subsystem with failures that originate at the top level. This can be particularly useful for describing failures that affect the component subsystem as a whole; for example, individual failure modes can be modelled at the subsystem level, while generic issues like electromagnetic interference or a power cut can be modelled once at the top component level.

Although different modelling tools handle it in different ways, in general a HiP-HOPS input file — containing the description of the system architecture and any failure annotations in XML form — is generated by the tool separately from its own model format. In this way, HiP-HOPS has a common input interface regardless of the originating modelling tool used.

2.2.2 Failure annotation

HiP-HOPS failure annotations come in three main forms:

- Failure modes / basic events, which are component-level failures;
- Input & Output deviations, which are propagated failures at inputs/outputs;
- Common Cause Failures, which are failures that exist at a system level rather than component level.

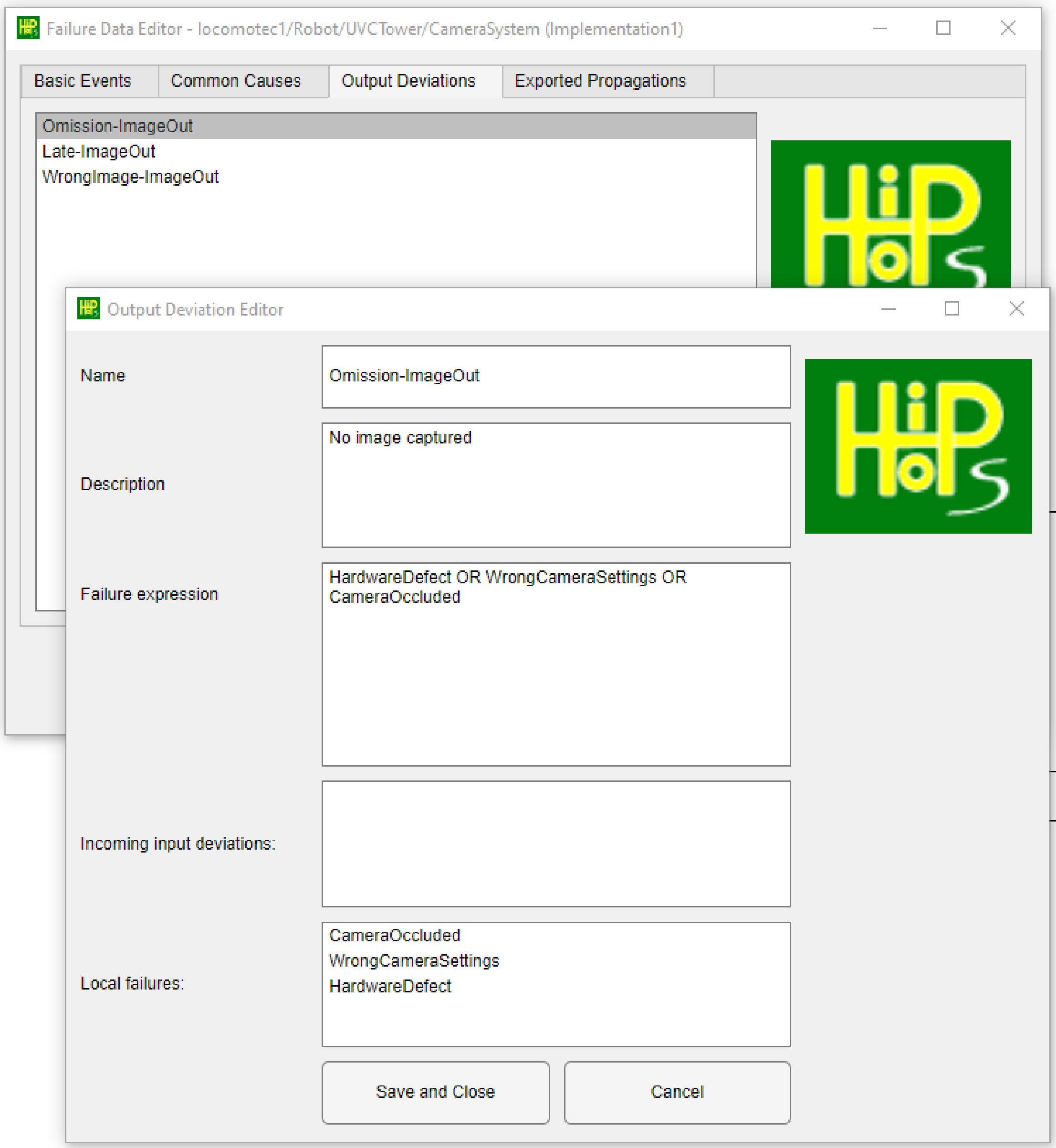
Failure modes (generally referred to as basic events in HiP-HOPS) are typically random hardware failures experienced by components, but in general they represent any failure that originates (or is initially detected) within a component. Software bugs, overheating, jamming, electromagnetic interference, water ingress etc are all possible basic events.

Output deviations represent the way these basic events (or input deviations) can cause unwanted deviations from nominal output at the output ports. Deviations all require a particular failure class, which indicates the general type of failure. Common classes include omission (lack of input/output when expected), commission (unintended/unwanted input/output), timing (e.g. late, early), and value (e.g. high, low) failures. Deviations are expressed as a combination of this failure class and the port name, e.g. `Omission-ImageOut` means an omission at the ImageOut port.

Output deviations specifically are also provided with logical expressions to describe their causes. This is how HiP-HOPS understands how failures propagate through components. Most expressions consist of basic events, input deviations, and simple Boolean operators (AND/OR), but more complex logic is also possible. As an example:

```
Omission-ImageOut = HardwareDefect OR Omission-PowerIn
```

Here the omission at image out for an imaging component (e.g. a camera) is caused either by an internal hardware defect or a lack of power at the electrical input port.



Omission-ImageOut
Late-ImageOut
WrongImage-ImageOut

Name: Omission-ImageOut
Description: No image captured
Failure expression: HardwareDefect OR WrongCameraSettings OR CameraOccluded
Incoming input deviations:
Local failures:
CameraOccluded
WrongCameraSettings
HardwareDefect

Figure 2 - HiP-HOPS failure annotation interface for Matlab Simulink

Output deviations can also refer to common cause failures. HiP-HOPS differentiates between what it calls *potential* common cause failures (PCCFs) and *actual* common cause failures (ACCFs). PCCFs are defined at the component level and represent effects from possible CCFs, should they exist; if the CCF does not exist in a particular operating context, then the PCCF is ignored.

To 'activate' a PCCF, a corresponding ACCF must be defined at the system level. These are common failures not specific to any particular component and typically represent environmental effects, e.g. flooding, fire etc.

In this way, the failure logic is more generic and can be stored in e.g. a component library without regard for whether or not the system environment features a particular CCF. For instance, components originally modelled for use aboard a ship may have a PCCF "water flooding" defined, but if re-used aboard an aircraft, the water flooding ACCF is (hopefully) absent and thus these PCCFs remain inert with no need to modify the logic of every output deviation.

2.2.3 Synthesis of propagation models

The internal workings of the tool are beyond the scope of this document, but different options are available when conducting the synthesis. The most important affect the structure of the resultant fault trees, which can either be left in their original form to better reflect the propagation of failures, or simplified and contracted to make them easier to read, albeit at the cost of most of the intermediate steps.

2.2.4 Failure Analysis

HiP-HOPS performs two main forms of analysis: fault tree analysis (FTA) and failure modes & effects analysis (FMEA). The former can involve both qualitative analysis (deduction of logical causes) and quantitative analysis (estimation of failure probability), while, as mentioned, FMEA can include both direct effects of failures and contributing effects that only occur in conjunction with other failures.

HiP-HOPS generates its analysis output in the form of multiple XML files which are by default accompanied by a HTML wrapper that enables them to be viewed in a web browser. Other forms of output are also possible, according to the options set, including output of a single XML file with no HTML (most useful for importing into other tools) and generation of an Excel spreadsheet.

2.3 SESAME-SPECIFIC EXTENSIONS AND FUNCTIONALITY

Both HiP-HOPS and the various compatible modelling tools are closed source tools. Therefore, in order to be able to use them effectively as part of the EDDI toolchain, it needs to be possible to convert from the native file formats (e.g. HiP-HOPS architecture or results XML files) to ODE files. This is done via one of the converters described in section 5.

Because HiP-HOPS served as one of the sources of input to the ODE design process, the two metamodels are broadly compatible. HiP-HOPS is fully capable of generating an ODE-compliant system architecture along with integrated ODE failure models like fault trees and FMEAs.

However, as a design time tool, HiP-HOPS lacks the capability to add runtime-specific information such as event monitoring or responses to fault detection/diagnosis. This must be added after the fact via other tools (e.g. Eclipse Epsilon or the EDDI Editor).

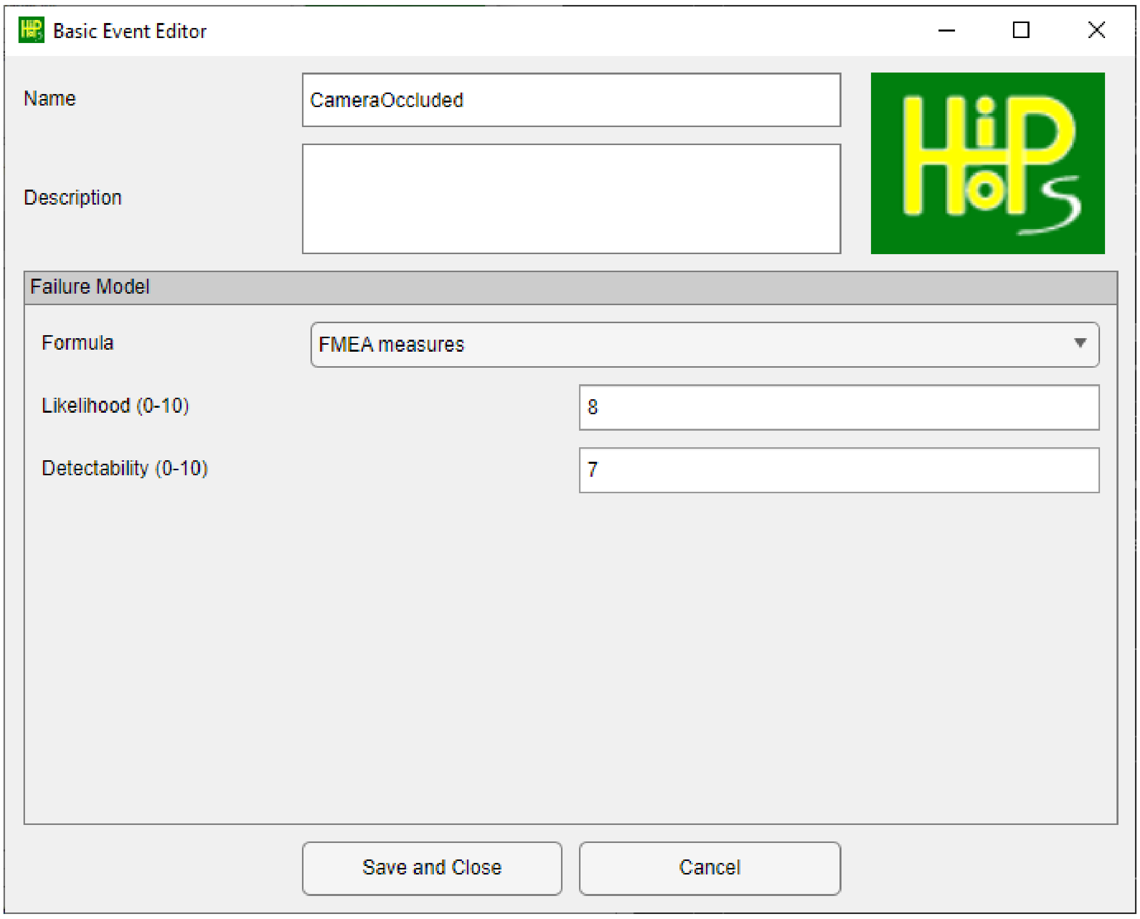
2.4 UPDATES SINCE D4.4

In light of feedback received during the initial round of use case evaluations by Locomotec and AVL, HiP-HOPS has been updated. Screenshots in the preceding section were updated to reflect these updates while the details of the changes are described below, along with comparative screenshots of the older interface.

Key points F1-F5 from the feedback were:

- From Locomotec (D8.5):
  - **F1**) the desire to include explicit qualitative likelihoods and RPNs in the FMEA;
- From AVL (D8.4):
  - **F2**) The ability to use SysML models;
  - **F3**) The UI should show a list of existing input deviations when defining output deviations, to avoid having to remember them;
  - **F4**) Most importantly, the UI should show all output deviations when defining hazards. Since these are non-local, it is even harder to remember them in large models.
- In addition, there was informal feedback that the UI was confusing in places (**F5**). This was apparent from some of the questions asked during the evaluation process. While this confusion was remedied at the time, it was decided that the UI needed to be overhauled and streamlined to improve usability and help avoid similar issues arising in the future.

Some initial work to the user interface has taken place to support F1, namely the input of qualitative-only FMEA data (see Figure 3). However, implementing this feature necessitates changes to the entire HiP-HOPS pipeline — not just the UI or input format, but the internal processing and analysis and the output format and display — so this work is still in progress.


Basic Event Editor

Name	CameraOccluded
Description	

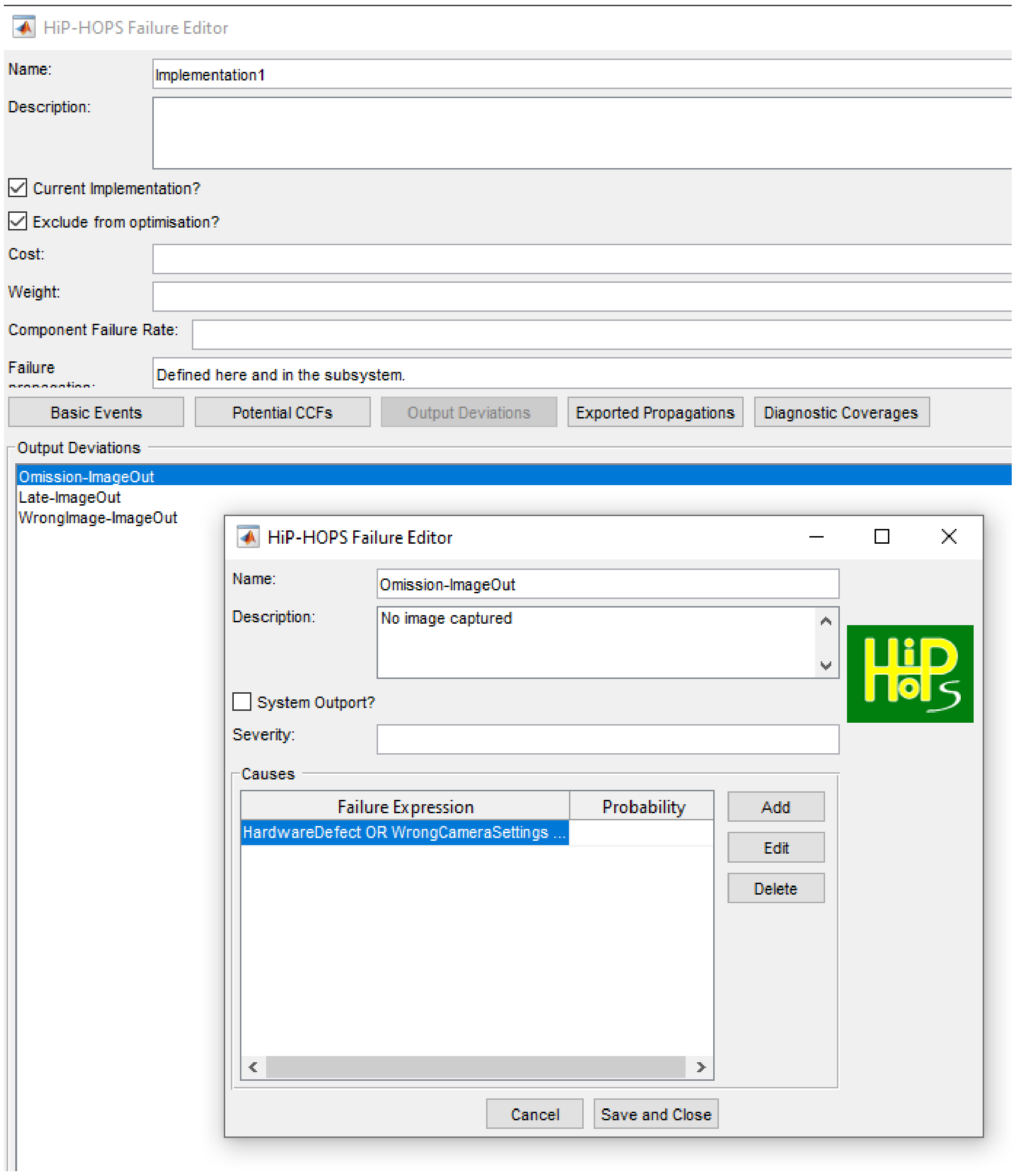
Failure Model

Formula	FMEA measures
Likelihood (0-10)	8
Detectability (0-10)	7

Figure 3 - Qualitative FMEA measures

Feedback F2, the ability to use SysML models, is infeasible. The issue here is that HiP-HOPS requires two things: a system architecture model and local failure data annotations. Even if a model transformation was created to import SysML architectural models directly, without the ability to add the failure data annotations, HiP-HOPS cannot function. Creating these annotations requires a plugin or UI of some kind that forms part of the modelling tool itself (like the Simulink interface above), so that the model and annotations can be created together. Implementing such a UI for a suitable SysML modelling package, together with the required XML export and model transformation, would be a major undertaking in its own right.

The remaining points (F3-F5) relate primarily to the usability of the existing Simulink user interface. To this end, a major overhaul of the Simulink-based user interface took place. Many of the more confusing options — specialised options that are rarely used, or that relate to optimisation functionality — were moved into a separate window, while the commonly used fields for normal failure data annotation were streamlined. This can be seen by comparing Figure 4 below with the earlier Figure 2: both show the same component failure data, but the updated interface is cleaner and less cluttered.


HiP-HOPS Failure Editor

Name: Implementation1

Description:

☑ Current Implementation?

☑ Exclude from optimisation?

Cost:

Weight:

Component Failure Rate:

Failure propagation: Defined here and in the subsystem.

Output Deviations

- Omission-ImageOut
- Late-ImageOut
- WrongImage-ImageOut

HiP-HOPS Failure Editor

Name: Omission-ImageOut

Description: No image captured

☐ System Outport?

Severity:

Causes

Failure Expression	Probability
HardwareDefect OR WrongCameraSettings ...	

Figure 4 - The previous HiP-HOPS failure data interface (cf. Fig 2)

Points F3 and F4 were also addressed directly. Ideally, these would be achieved by adding autocomplete and syntax highlighting functionality to the text box where the failure expressions are defined; however, this is not possible with the API Matlab exposes.

Instead, new list boxes were added to the output deviation and hazard definition dialogs. These list boxes query the underlying model and HiP-HOPS annotations to populate the lists with incoming input deviations or global output deviations. Double-clicking any deviation copies it directly into the failure expression box at the current cursor position. This achieves the primary objective of not having to remember the deviations to type them in manually, which can be frustrating and error-prone.

To illustrate, an example is shown below.

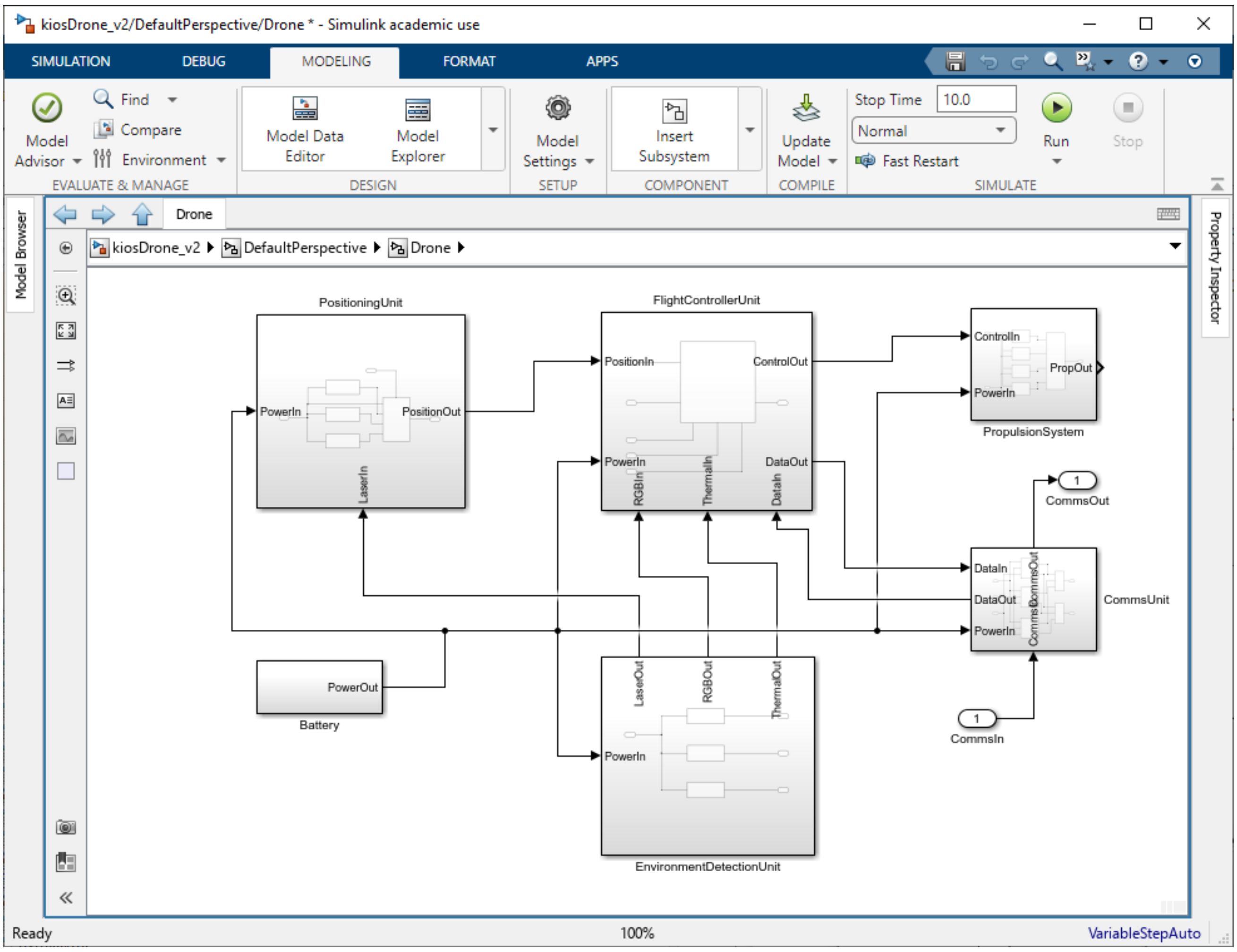


Figure 5 - KIOS drone model in Matlab Simulink

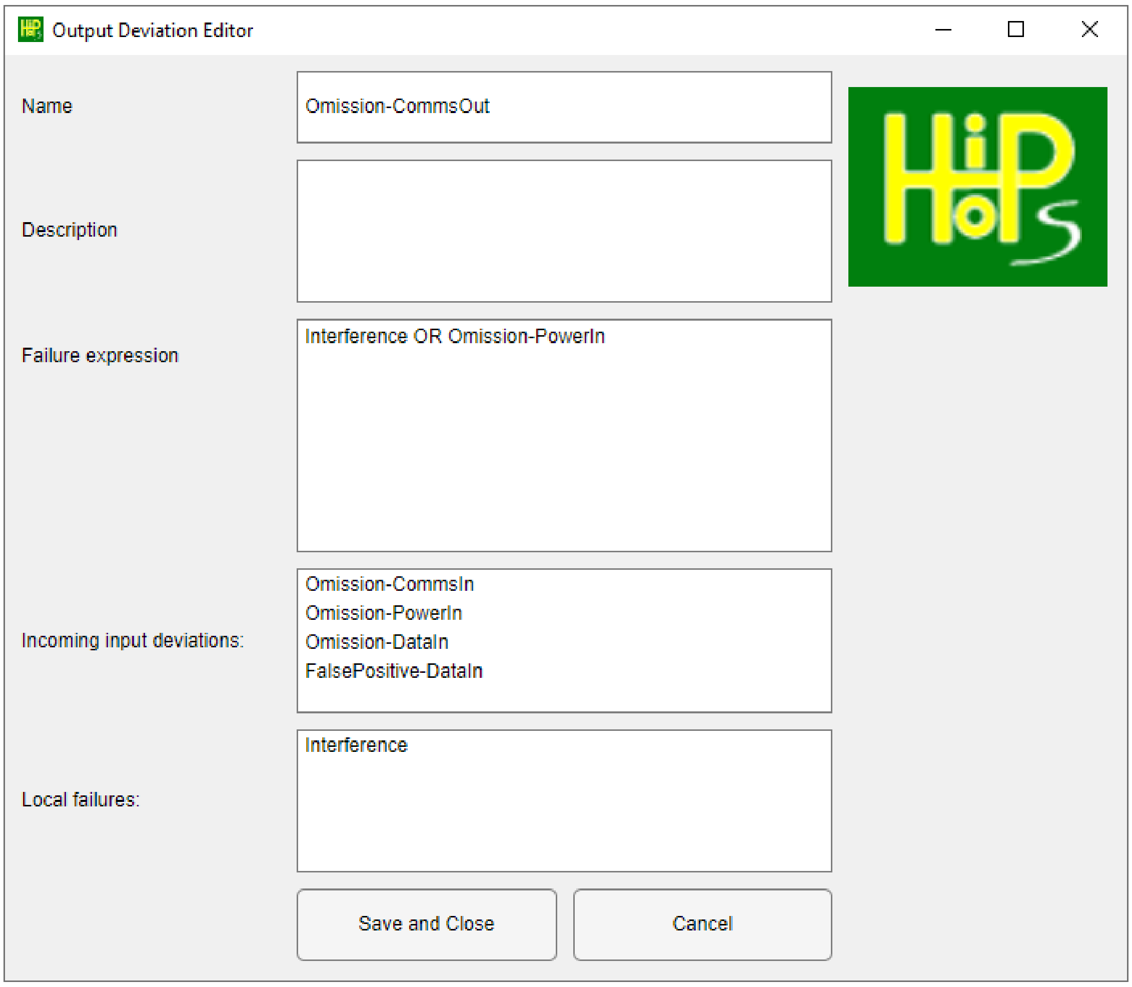
This architecture is explained further in section 7.1, but this particular part of the model shows the top-level subsystems of a drone:

- Positioning Unit (top left)
- Flight Controller Unit (top middle)
- Propulsion Unit (top right)
- Battery (bottom left)
- Environmental Detection Unit (bottom middle)
- Communications Unit (bottom right)

Except for the battery, all of these subsystems have subcomponents that may in turn generate output deviations. In addition, more output deviations may come from outside this subsystem via the CommsIn port (the small rounded rectangle under the Comms Unit).

Previously, when defining output deviations for a component (e.g. the Communications Unit), the user would have to remember all of the output deviations that lead to this component. For simple, flat models, this is not an insurmountable issue; however, for more complex hierarchical models like this, where output deviations may come from subcomponents or supercomponents not even visible in the main view, this can be challenging and error-prone.

The new interface solves this by providing ready-made lists of both incoming input deviations (whether from the same level of the system, a subcomponent, or a supercomponent elsewhere) as well as any internal local failures that have been defined:



Output Deviation Editor

Name: Omission-CommsOut

Description:

Failure expression: Interference OR Omission-PowerIn

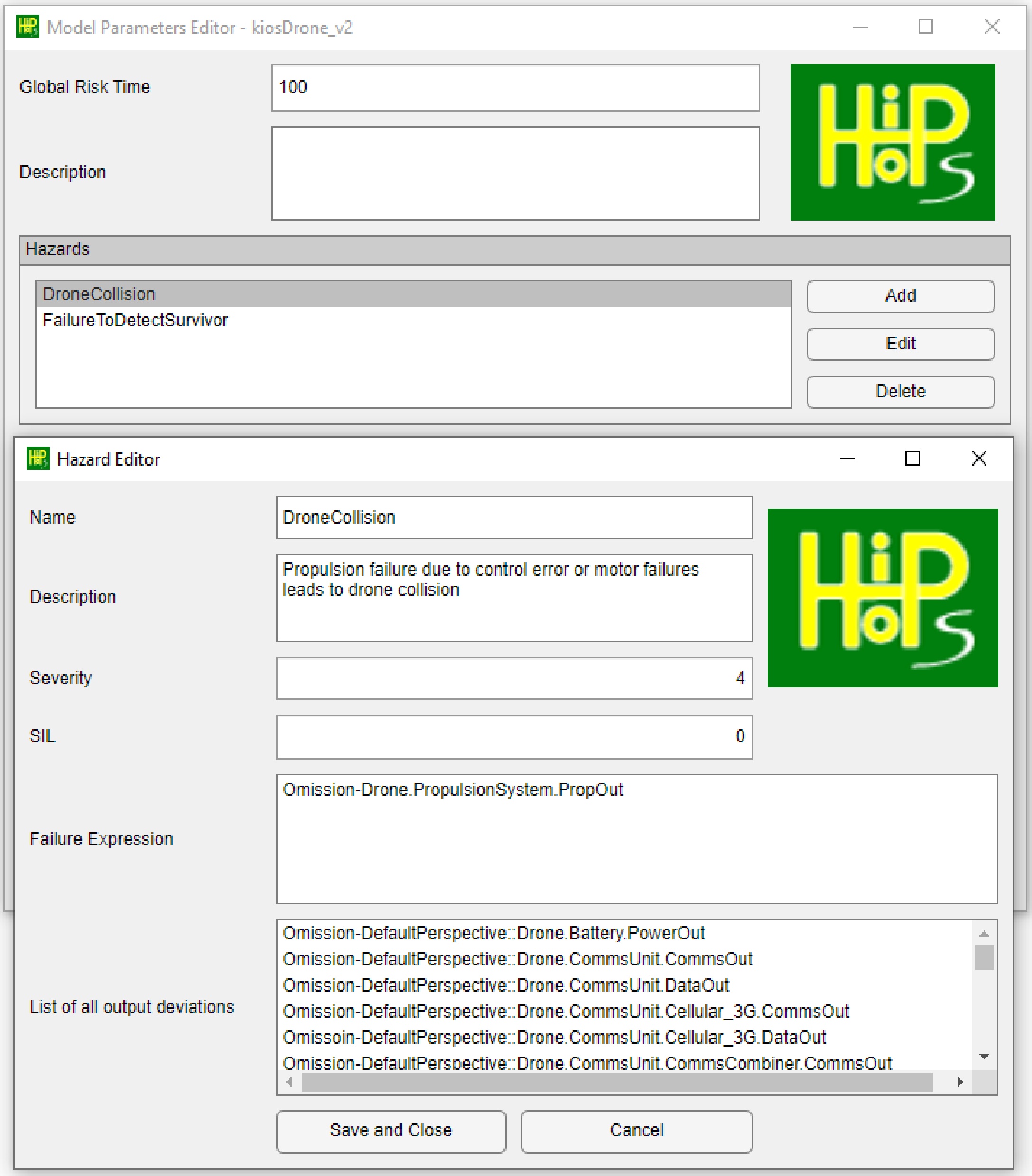
Incoming input deviations:
Omission-CommsIn
Omission-PowerIn
Omission-DataIn
FalsePositive-DataIn

Local failures:
Interference

Figure 6 - New incoming input deviations and internal failures lists

In this figure, Omission-PowerIn is the only deviation on the same level — it originates from the battery. The omission and false positive at DataIn originate from subcomponents of the Flight Controller Unit, while Omission-CommsIn originates from the ground station, which is an entirely different system.

Similar functionality has been added to the hazard definition dialog (see Figure 7 below), where a list of all output deviations across the model is generated and displayed. For large models with many components (and thus many deviations), this results in significant savings in time, effort, and sanity.


Model Parameters Editor - kiosDrone_v2

Global Risk Time: 100

Description:

Hazards

DroneCollision
FailureToDetectSurvivor

Hazard Editor

Name: DroneCollision

Description: Propulsion failure due to control error or motor failures leads to drone collision

Severity: 4

SIL: 0

Failure Expression: Omission-Drone.PropulsionSystem.PropOut

List of all output deviations:

```
Omission-DefaultPerspective::Drone.Battery.PowerOut
Omission-DefaultPerspective::Drone.CommsUnit.CommsOut
Omission-DefaultPerspective::Drone.CommsUnit.DataOut
Omission-DefaultPerspective::Drone.CommsUnit.Cellular_3G.CommsOut
Omissoin-DefaultPerspective::Drone.CommsUnit.Cellular_3G.DataOut
Omission-DefaultPerspective::Drone.CommsUnit.CommsCombiner.CommsOut
```

Figure 7 - Global list of output deviations for hazard definition

3. SafeTbox

SafeTbox is a model-based safety modelling and analysis tool developed by and commercially available from Fraunhofer IESE. The tool supports the safety engineering lifecycle, focusing particularly on the stages before implementation, i.e. requirements analysis and system design. Thus, the tool allows modelling a system's architecture, analysing its risks and potential failures, and structuring assurance cases regarding the system's dependability.

3.1 What is safeTbox

SafeTbox is an add-in to the Enterprise Architect (EA) modelling framework. Thus, it extends EA's system modelling support for popular languages UML and SysML, featuring its own component-based system modelling variant. An example of how this variant appears graphically in the tool can be seen in Figure 8.

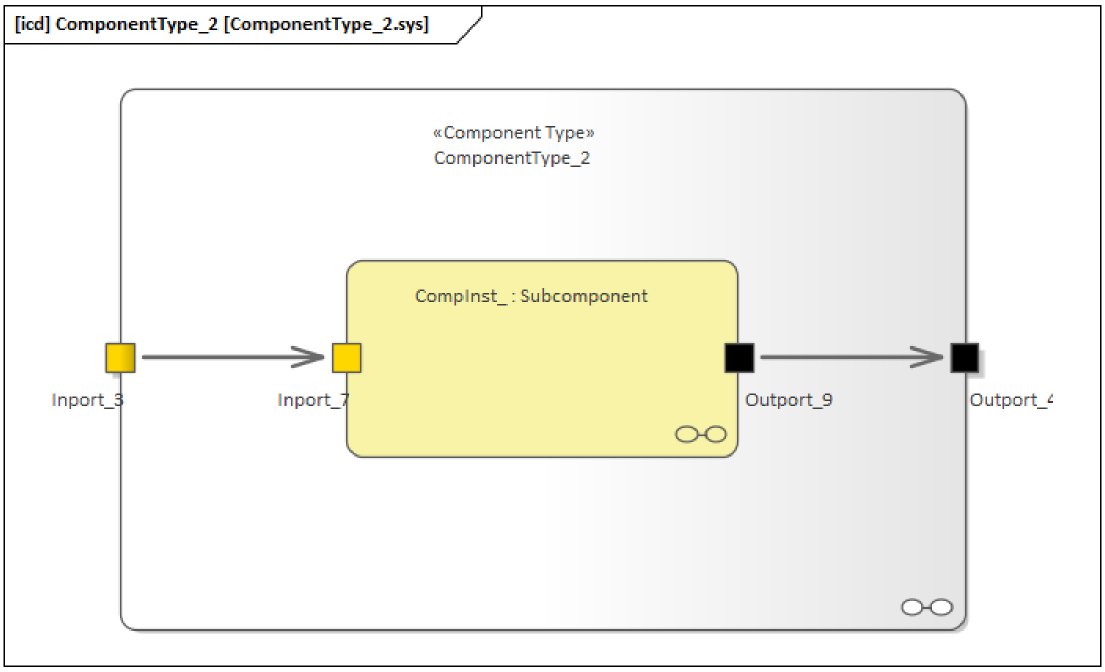
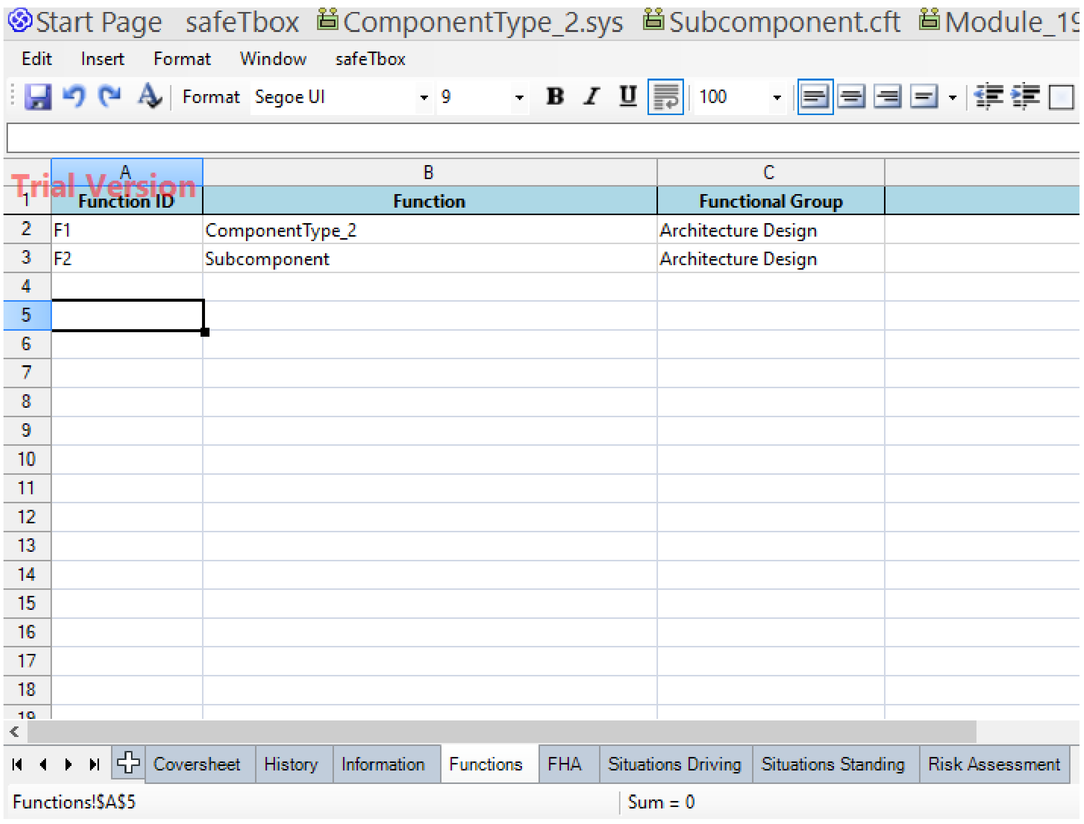


Figure 8 - Example of abstract component in safeTbox

In this figure, a host component (whose type is 'Component_Type2') contains input and output ports with which it connects to its external environment, and simultaneously contains an internal component (whose type is 'Subcomponent' and whose instance's name is 'CompInst'). The host component's ports are linked to the contained component's ports, allowing information, energy, or similar relations to flow from outside the host component, to the contained component, and vice versa.

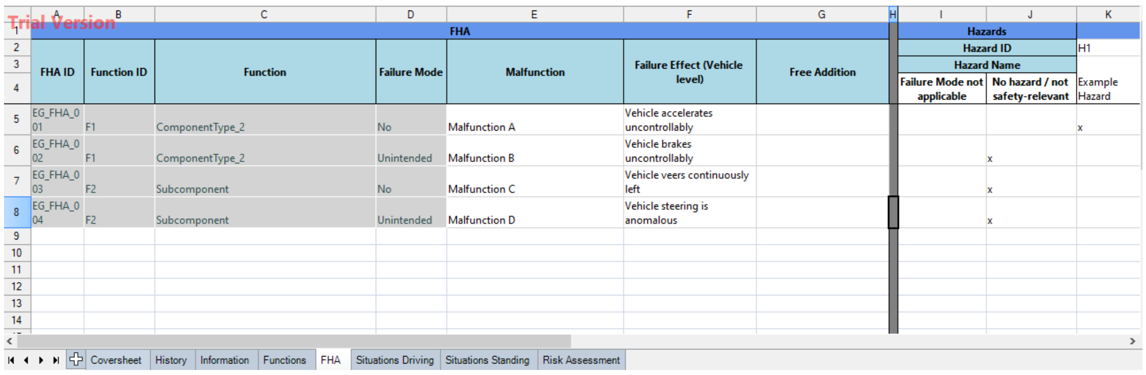
While these are listed as 'components', they are abstract and flexible enough to also represent complex system architectures. At the system level, dependability analysis often begins with a Hazard Analysis and Risk Assessment (HARA), a part of which can be seen in Figure 9. In the figure, a spreadsheet view allows the user to navigate between sheets, each of which is dedicated to a different process step of the HARA. The initial sheets are mostly for process documentation and information; the first process step begins with the 'Functions' sheet, as seen in Figure 9. In this sheet, the user can specify their own system functions manually, or add existing ones defined in the system model previously. For example, each of the components listed before can be considered as a function and included in the list.



Function ID	Function	Functional Group
F1	ComponentType_2	Architecture Design
F2	Subcomponent	Architecture Design

Figure 9 - Example of Functions sheet in safeTbox HARA

Once functions are included, the next step is identifying the different possible modes of functional failure. This is performed in the ‘FHA’ sheet, as seen in Figure 10. A standard list of keywords is typically used here, including ‘Omission’/’Commission’ (function was not provided when needed/provided when not needed), ‘Too Early’/’Too Late’, ‘Too Low’/’Too High’, etc. Each of the functions listed previously can be considered against each of the keywords (and beyond, if the analyst finds a unique failure). Failure modes that are considered potentially hazardous (could cause harm to people, environment, etc.) can be mapped to specific hazards. Each of the row entries corresponds to a functional failure, and each vertical entry on the rightmost columns represents a user-specified hazard that can be associated with those corresponding functional failures.

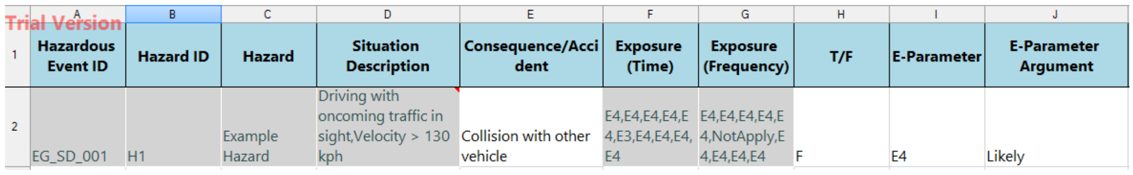


FHA							Hazards		
							Hazard ID		H1
							Hazard Name		
FHA ID	Function ID	Function	Failure Mode	Malfunction	Failure Effect (Vehicle level)	Free Addition	Failure Mode not applicable	No hazard / not safety-relevant	Example Hazard
EG_FHA_001	F1	ComponentType_2	No	Malfunction A	Vehicle accelerates uncontrollably				x
EG_FHA_002	F1	ComponentType_2	Unintended	Malfunction B	Vehicle brakes uncontrollably			x	
EG_FHA_003	F2	Subcomponent	No	Malfunction C	Vehicle veers continuously left			x	
EG_FHA_004	F2	Subcomponent	Unintended	Malfunction D	Vehicle steering is anomalous			x	

Coversheet | History | Information | Functions | FHA | Situations Driving | Situations Standing | Risk Assessment

Figure 10 - Example of FHA sheet in safeTbox HARA

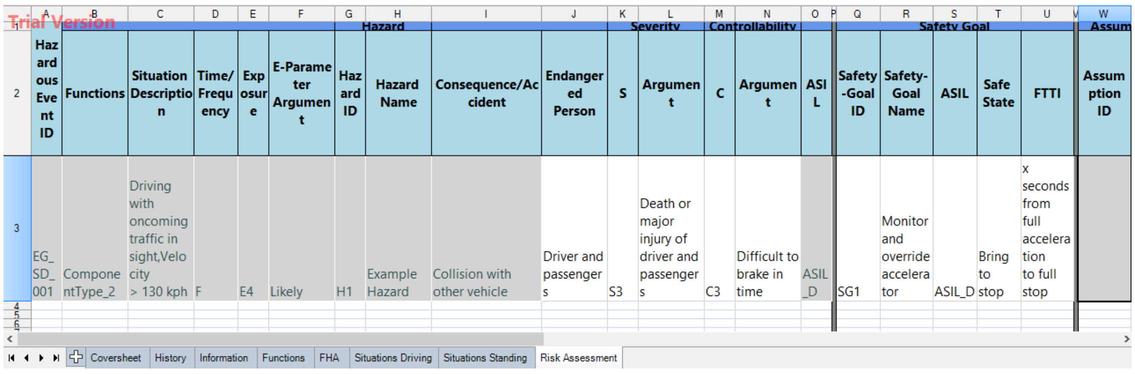
Once a set of hazards has been collected, the ‘Situations Driving’ and ‘Situations Standing’ sheets allow detailing of the exposure risk parameter for specific situations involving those hazards (while the vehicle is driving or standing still accordingly). See Figure 11 for an example. Exposure reflects how likely a given hazardous situation is expected to occur during the vehicle’s operation. The combination of a hazard and such a situation is referred to as a hazardous event.



Hazardous Event ID	Hazard ID	Hazard	Situation Description	Consequence/Accident	Exposure (Time)	Exposure (Frequency)	T/F	E-Parameter	E-Parameter Argument
EG_SD_001	H1	Example Hazard	Driving with oncoming traffic in sight,Velocity > 130 kph	Collision with other vehicle	E4,E4,E4,E4,E4,E3,E4,E4,E4,E4	E4,E4,E4,E4,E4,NotApply,E4,E4,E4,E4	F	E4	Likely

Figure 11 - Partial example of situations driving sheet in safeTbox HARA

In the final sheet (‘Risk Assessment’), the hazardous events from both situation sheets are collected, and each is further rated with respect to the severity of the hazard (e.g. death or severe injury of the driver) and with respect to its controllability by the driver (e.g. the driver can easily observe the problem and halt the vehicle). The more severe, frequently occurring, and difficult to control a hazard (-ous event), the higher the overall assessed risk rating it receives (represented as an ISO26262 Automotive Safety Integrity Level - ASIL). To mitigate one or more hazardous events, appropriate safety goals must be specified, and be assigned an ASIL corresponding to the highest ASIL of all the hazardous events they are intended to address.



						Hazard				Severity		Controllability			Safety Goal					Assum
Hazardous Event ID	Functions	Situation Description	Time/ Frequency	Exposure	E-Parameter Argument	Hazard ID	Hazard Name	Consequence/Accident	Endangered Person	S	Argument	C	Argument	ASIL	Safety-Goal ID	Safety-Goal Name	ASIL	Safe State	FTTI	Assumption ID
EG_SD_001	ComponentType_2	Driving with oncoming traffic in sight,Velocity > 130 kph	F	E4	Likely	H1	Example Hazard	Collision with other vehicle	Driver and passengers	S3	Death or major injury of driver and passengers	C3	Difficult to brake in time	ASIL_D	SG1	Monitor and override accelerator	ASIL_D	Bring to stop	x seconds from full acceleration to full stop	

Coversheet | History | Information | Functions | FHA | Situations Driving | Situations Standing | Risk Assessment

Figure 12 - Example of risk assessment sheet in safeTbox HARA

Either during or after the HARA, more detailed failure analysis may be needed to dig deeper into underlying causes of system-level failures. Fault trees are one type of causal analysis that can be used in this way. Figure 13 shows what the failure logic of the subcomponent might look like. In the figure, the component's structure is represented by the outer box ('<<Component Type>>Subcomponent'), whereas the inner box ('<<CFT>>Subcomponent') represents its Component Fault Tree (CFT) [5] [6]. CFTs (described further in **D4.5: Safety Analysis Concept**) are a variant of traditional fault trees and allow us to modularize fault tree models and map them to their corresponding components.

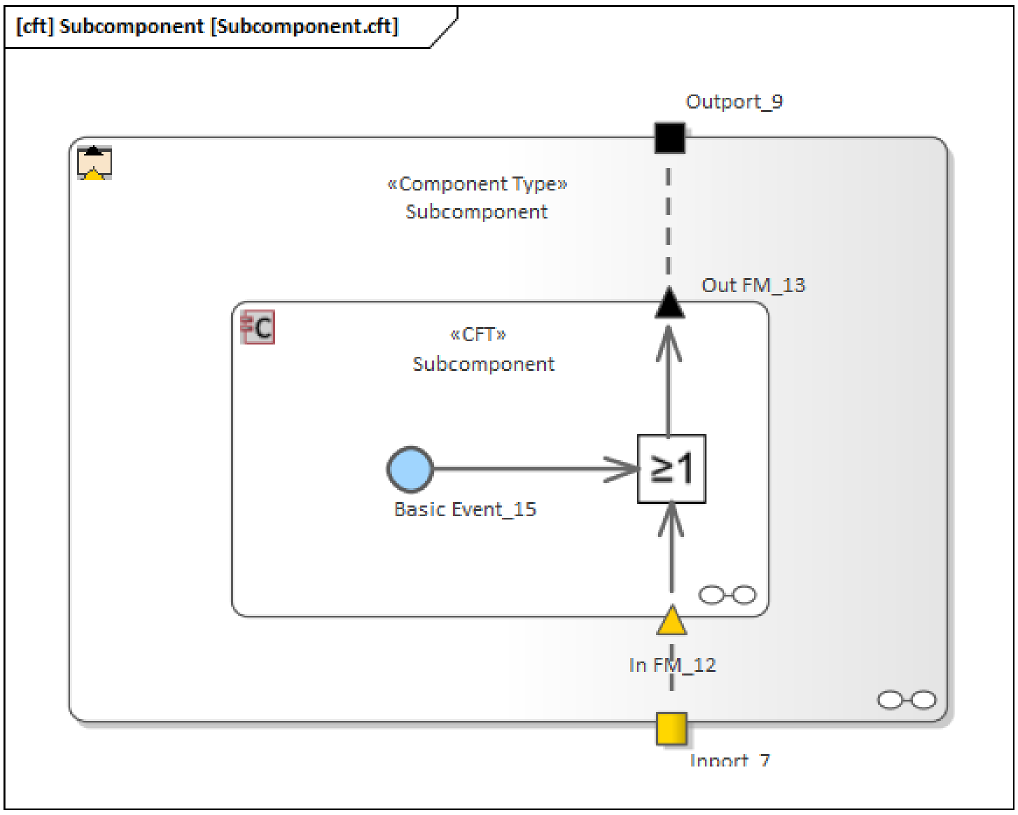


Figure 13 - Example of failure logic for subcomponent in safeTbox

The CFT contains an input failure mode (In FM_12) and output failure mode (Out FM_13). Each is associated with the outer box's component ports, indicating where that failure might occur and/or propagate to, either inside or outside the component. Inside the CFT, the failure modes are linked by a logical 'OR' gate ("at least 1" symbol) to a basic event (Basic Event_15). The basic event represents an internal component fault that does not need to be decomposed further. Automatic qualitative analysis of the CFT can reveal the minimal combinations of necessary and sufficient basic events and input failure modes needed to trigger a specific failure. The analysis can also be quantitative, evaluating the likelihood of the subject failure of occurring, based on assumed probability models of the basic events.

Finally, assurance cases can also be modeled using the Goal Structuring Notation (GSN) [7]. An example of an abstract GSN structure in safeTbox can be seen in Figure 14, where the safety of 'ComponentType_2' is claimed to be acceptable. The claim is depicted using a rectangle (Goal_20), also known as a 'Goal' in GSN. To support the claim, a strategy of argument (represented as a parallelogram Strategy_21) is used, arguing that acceptable safety is achieved by evaluating the system's residual risk after its safety goals have been validated. Another Goal (Goal_22) claims that the residual risk is acceptable; an associated Assumption (Assumption_25, represented with an ellipse) assumes that the safety goals contribute no risk of their own to the system. Finally, evidence supports the final claim, by referring to the CFT of the subcomponent seen previously in Figure 13, and notes that the risk of critical failures after the safety goals have been implemented is acceptably low. Of course, we should note that this is an oversimplified example, for illustration purposes.

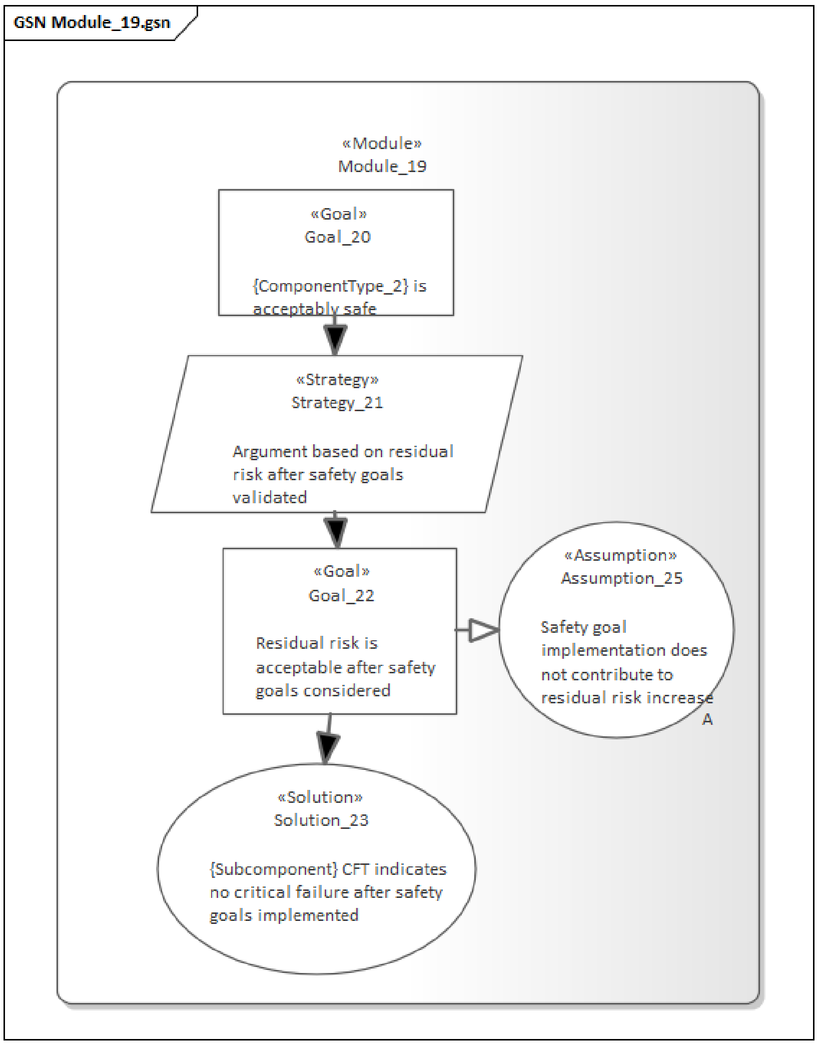

GSN Module_19.gsn
«Module»
Module_19
«Goal»
Goal_20
{ComponentType_2} is acceptably safe
«Strategy»
Strategy_21
Argument based on residual risk after safety goals validated
«Goal»
Goal_22
Residual risk is acceptable after safety goals considered
«Assumption»
Assumption_25
Safety goal implementation does not contribute to residual risk increase
A
«Solution»
Solution_23
{Subcomponent} CFT indicates no critical failure after safety goals implemented

Figure 14 - Example of abstract GSN structure in safeTbox

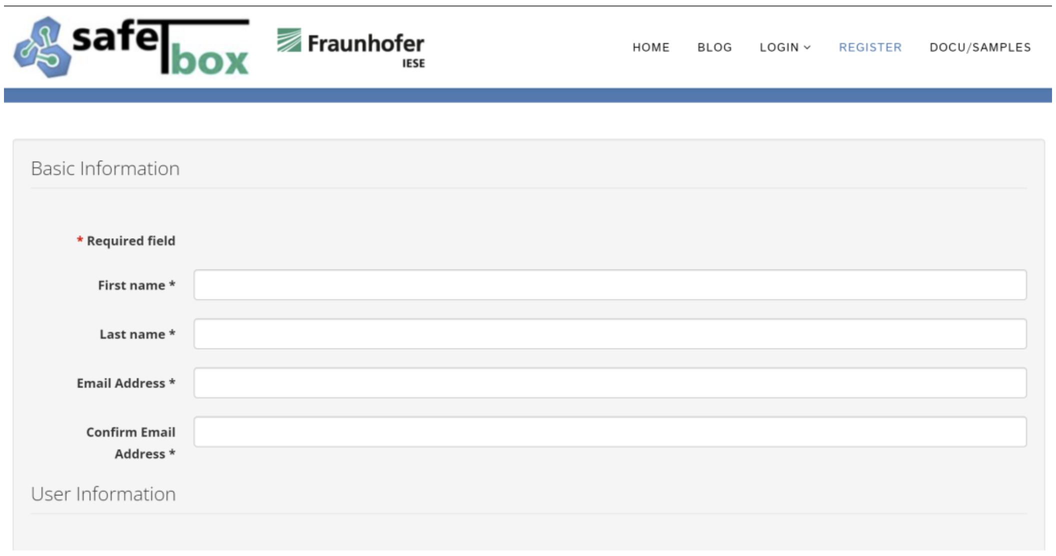
3.2 How it works

For detailed guidance on using safeTbox, the user manual and provided example model should be useful references[5]. An abridged guide is included below for users in SESAME.

To use safeTbox, an existing installation of Enterprise Architect (EA) is required, up to version 15.1. Trial versions are also supported. SafeTbox can be downloaded and

[5] https://safetbox.de/docu-samples

installed for free from its dedicated website[6]; the user must register an account, as seen in Figure 15, and they can then download the installer executable and a trial license, as seen in Figure 16.



Basic Information

* Required field

First name *

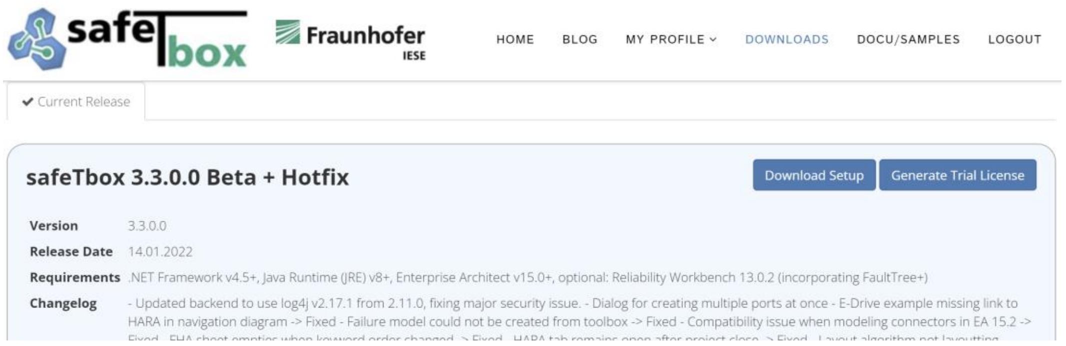
Last name *

Email Address *

Confirm Email Address *

User Information

Figure 15 - safeTbox website registration page



safeTbox 3.3.0.0 Beta + Hotfix

Version	3.3.0.0
Release Date	14.01.2022
Requirements	.NET Framework v4.5+, Java Runtime (JRE) v8+, Enterprise Architect v15.0+, optional: Reliability Workbench 13.0.2 (incorporating FaultTree+)
Changelog	- Updated backend to use log4j v2.17.1 from 2.11.0, fixing major security issue. - Dialog for creating multiple ports at once - E-Drive example missing link to HARA in navigation diagram -> Fixed - Failure model could not be created from toolbox -> Fixed - Compatibility issue when modeling connectors in EA 15.2 ->

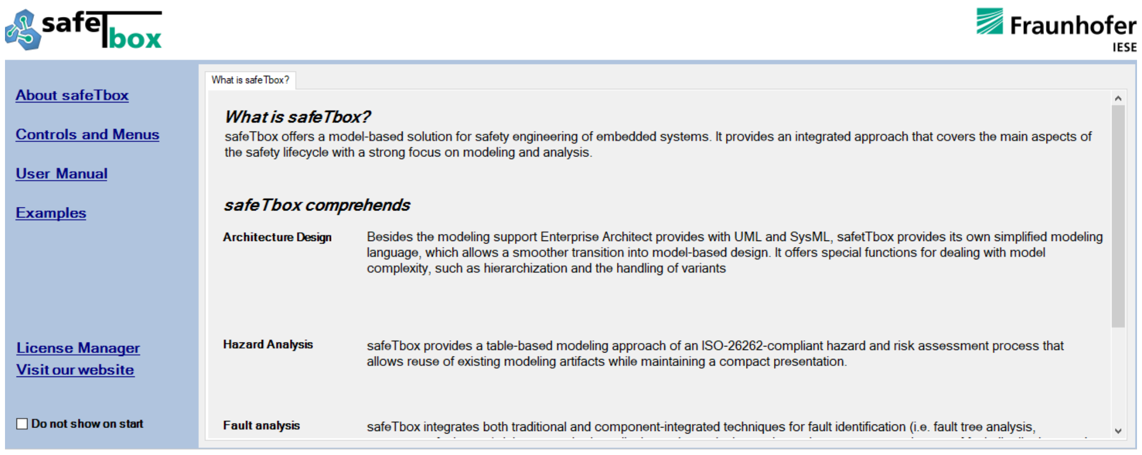
Figure 16 - safeTbox website downloads page

Once downloaded, run the installer executable and follow the instructions (installing any additional requirements that are needed as well).

Once installed, safeTbox should be usable the next time EA is launched.

The first time EA is launched, you will be requested to provide your downloaded license. Simply use the dialog to navigate to the license file and confirm. Upon starting EA from here onwards, you should be greeted by the safeTbox welcome screen, as seen in Figure 17.

[6] www.safetbox.de



What is safeTbox?

safeTbox offers a model-based solution for safety engineering of embedded systems. It provides an integrated approach that covers the main aspects of the safety lifecycle with a strong focus on modeling and analysis.

safeTbox comprehends

Architecture Design	Besides the modeling support Enterprise Architect provides with UML and SysML, safetTbox provides its own simplified modeling language, which allows a smoother transition into model-based design. It offers special functions for dealing with model complexity, such as hierarchization and the handling of variants
Hazard Analysis	safeTbox provides a table-based modeling approach of an ISO-26262-compliant hazard and risk assessment process that allows reuse of existing modeling artifacts while maintaining a compact presentation.
Fault analysis	safeTbox integrates both traditional and component-integrated techniques for fault identification (i.e. fault tree analysis,

Figure 17 - safeTbox welcome screen

If this is not the case, review your installation steps, consult the user manual, or contact safeTbox user support via the website.

To create a model, open the (project) ‘Browser’ pane and click to open or create a new project, as per Figure 18.

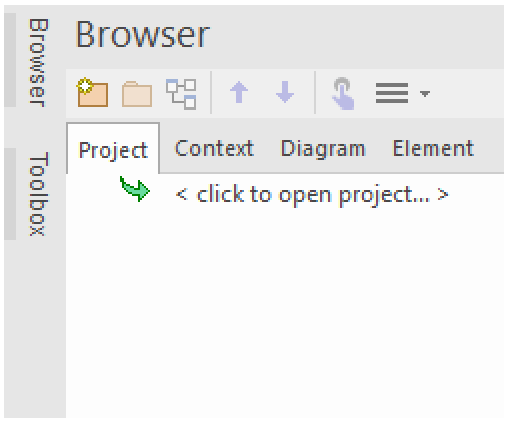


Figure 18 - Creating a new project in safeTbox

Use the file dialog to either select the file path of the new project, or the file path of an existing one. After successful creation/loading, the project’s ‘Model’ should be visible in the Browser pane, as in Figure 19. By selecting and pressing Ctrl+Space, the safeTbox ‘Smart Menu’ should appear, allowing access to options such as Create, as seen in Figure 20.

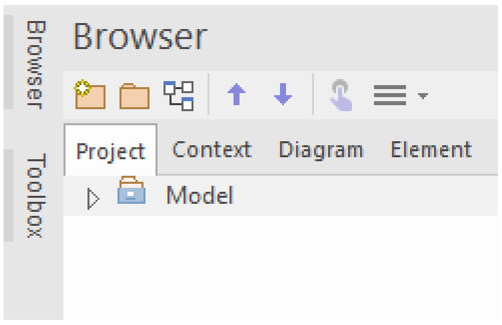


Figure 19 - A project with a loaded model in safeTbox

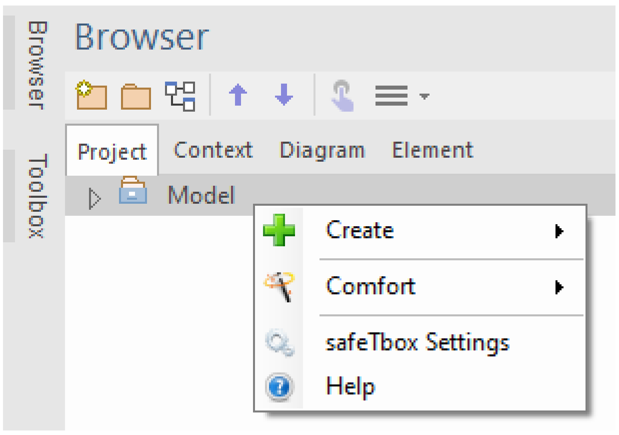


Figure 20 - safeTbox Smart Menu

The Create option is likely the most useful, as it lets the user add new system, HARA, CFT, and GSN models to the project model.

3.2.1 Using SafeTbox for Systems Modelling

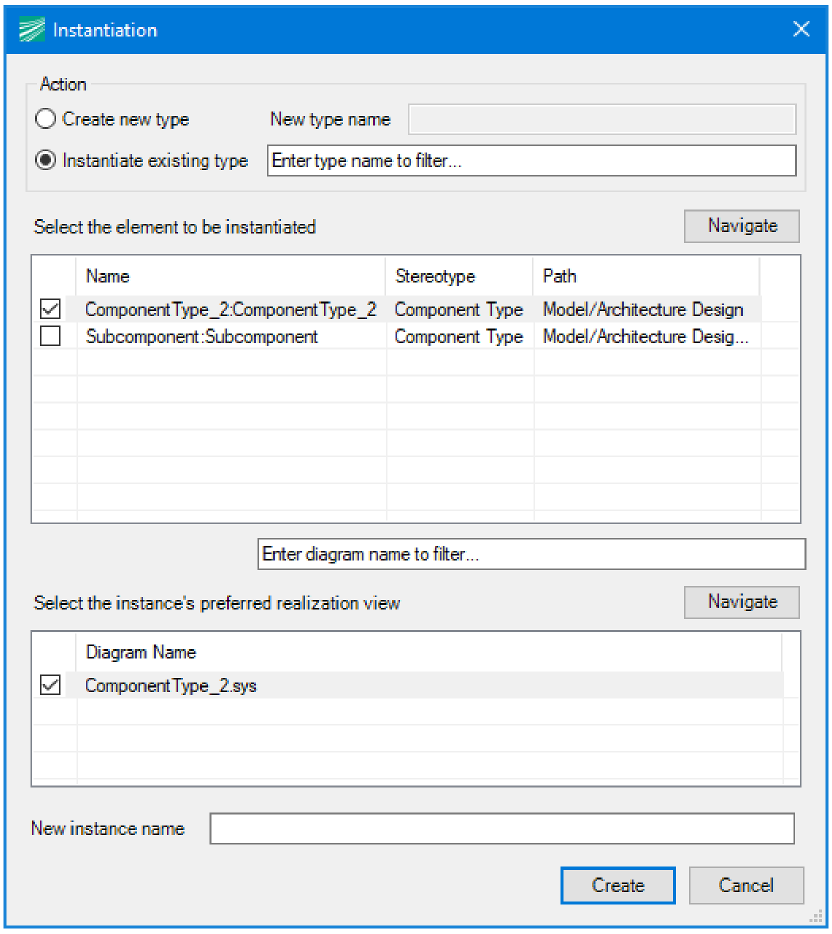
To model a system e.g. an individual robot or even MRS, create a new Architecture Model via the Smart Menu.

A new tab should appear, where a randomized name has been given to the new component, seen at the center of the view. To rename the component and set its properties, either double-click on it, or select and use the Smart Menu to access its safeTbox Properties. In the dialog that appears, you can set various options, importantly its name and description.

Upon confirming your changes, the dialog should close and the component should reflect its new name and other visual properties that changed.

New ports and subcomponents can also be added via the Smart Menu. Ports can reflect communication or interaction points between the subject system and the external environment.

Subcomponents can represent participating elements of their host system e.g. individual robots within an MRS. Components can be re-used by selecting the corresponding option in the component creation dialog, as seen Figure 21.


Instantiation

Action

( ) Create new type   New type name

[✓ Instantiate existing type]   Enter type name to filter...

Select the element to be instantiated

	Name	Stereotype	Path
✓	ComponentType_2:ComponentType_2	Component Type	Model/Architecture Design
	Subcomponent:Subcomponent	Component Type	Model/Architecture Desig...

Enter diagram name to filter...

Select the instance's preferred realization view

	Diagram Name
✓	ComponentType_2.sys

New instance name

Figure 21 - Instantiating existing component types in safeTbox

The first list in the dialog allows the user to select which existing component to instantiate. The second list refers to a more advanced feature that is explained in the user manual. At the end of the dialog, an optional name for the new instance of the component can be provided directly.

3.2.2 Using SafeTbox for CFT Modelling

Once a new component type has been created, a CFT can be added to it using the Smart Menu => Create => Add Failure Model. A new tab should appear, depicting an emptier but otherwise similar view to Figure 13. Once again, the Smart Menu should allow access to creating all elements relevant to the CFT diagram.

3.2.3 Using SafeTbox for HARA

A HARA can be created from the (project) Browser pane via the Smart Menu. Note that the HARA model requires user-triggered synchronization with the rest of the model; this option is available via the tab menu => safeTbox => Synchronization. Synchronization allows updates in both Model-HARA directions to occur.

As you progress through the sheets, you can often add safeTbox-related content to the rows and columns by right-clicking on the margin. For example, this applies in the Functions sheet, where the user can add Functions by right-clicking on the row margin and selecting the Add Function option in the menu that appears, as seen in Figure 22.

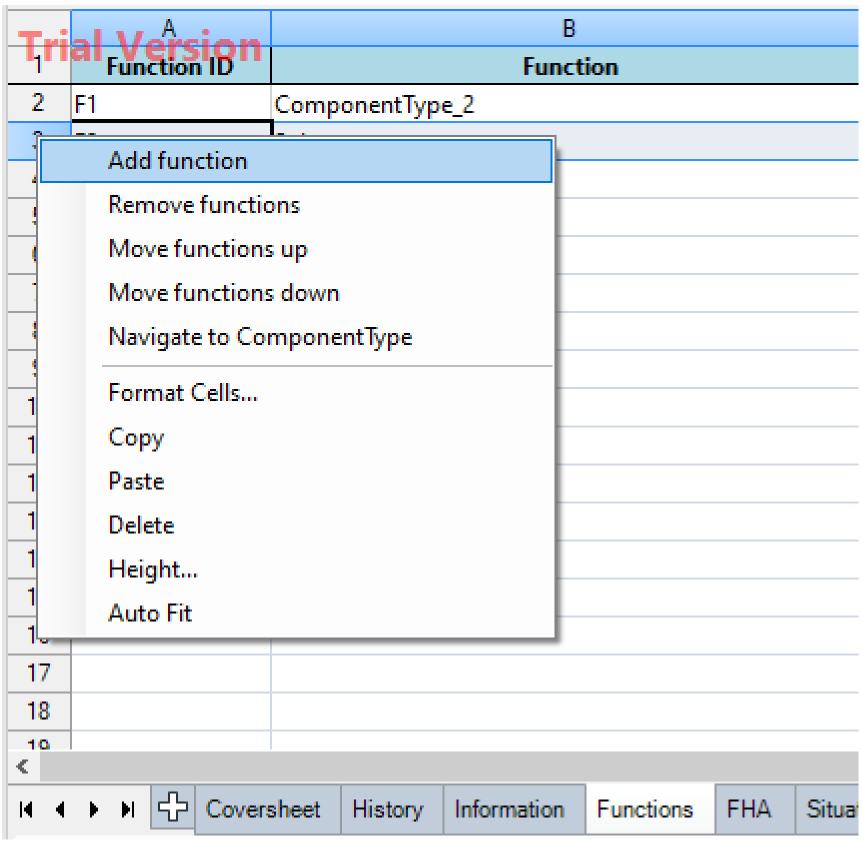


Figure 22 - Adding Functions to the HARA in safeTbox

Similar functionality can be found throughout the HARA spreadsheet, including:

- In the FHA sheet, new Hazard entries can be added by right-clicking on the rightmost columns' margin
- In the Situations sheets, new entries can be added by right-clicking on the row margin
- In the Risk Assessment sheet, new Safety Goals can be added by right-clicking on cells under the columns in the Safety Goal section.

The embedded safeTbox menu is also important for some of the sheets:

- In the Functions sheet, safeTbox => "Load existing component types" opens a dialog to conveniently select which system model elements shall be imported into the sheet
- In the FHA sheet, safeTbox => "Permutate functions with failure modes" automatically generates entries based on combinations of the chosen guidewords and the Functions sheet's functions.
- In the Situations sheets, safeTbox => "Update situations sheet" automatically generates situations based on the hazards specified in the FHA sheet.

- In the Risk Assessment sheet, safeTbox => "Update risk assessment sheet" automatically collects hazardous events from the Situations sheets.

3.2.4 Using SafeTbox for GSN Assurance Cases

To create a new GSN case, use the Smart Menu in the (project) Browser. A new GSN Module should appear in the modelling view. A GSN Module represents a part of an assurance case, and its contained elements can be referenced by other Modules.

The Smart Menu allows the user to create new Goals, Strategies, Solutions, and other GSN elements. For referencing other Modules, use the Toolbox pane to access Away elements (e.g. Away Goals), which can be dragged onto the module to add them. Adding a new Away element to a Module will prompt the user with a dialog to specify which GSN element should be referenced. Note that only GSN elements with the property 'Public' can be referenced, so make sure to set that property on the elements you wish to be referenceable (via the safeTbox properties dialog).

3.2.5 Using SafeTbox for ConSerts

Creating ConSerts involves adding at least two models: a Collaborative System Group (CSG), and a Collaborative System (CS). This can be done via the Smart Menu, as seen in Figure 23.

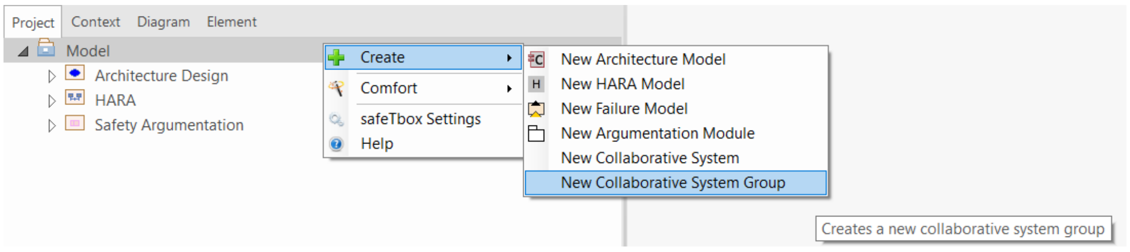


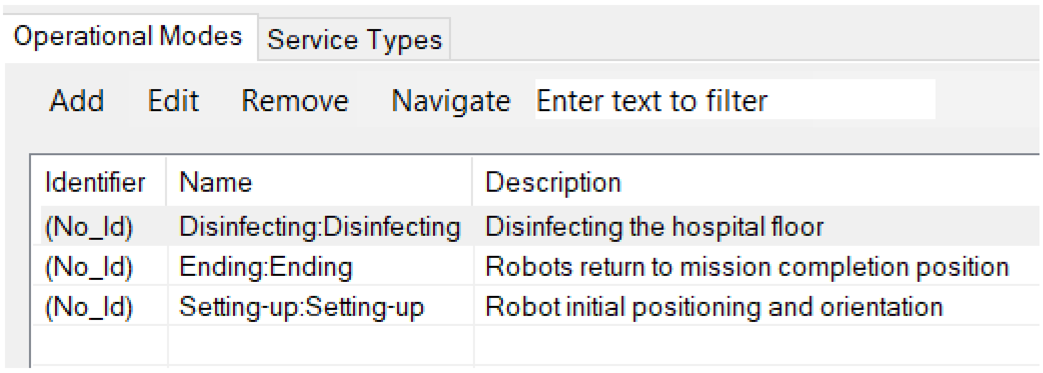
Figure 23 - Adding ConSerts in safeTbox

CSGs represent sets of systems that collaborate to deliver an application-level service e.g. hospital disinfection for the corresponding SESAME use case. To implement this service, each constituent CS provides some of its own services as the application service itself, or as a supporting service to other CS.

The CSG acts as a specification of service types and operational modes for the CS that are associated to it. Each CS must offer services compatible with those types.

To begin the process, a CSG should be created first; to edit the service types and operational modes of the CSG, use the Smart Menu => Specify Services option. A form should appear in the main view, as partially seen in Figure 24.

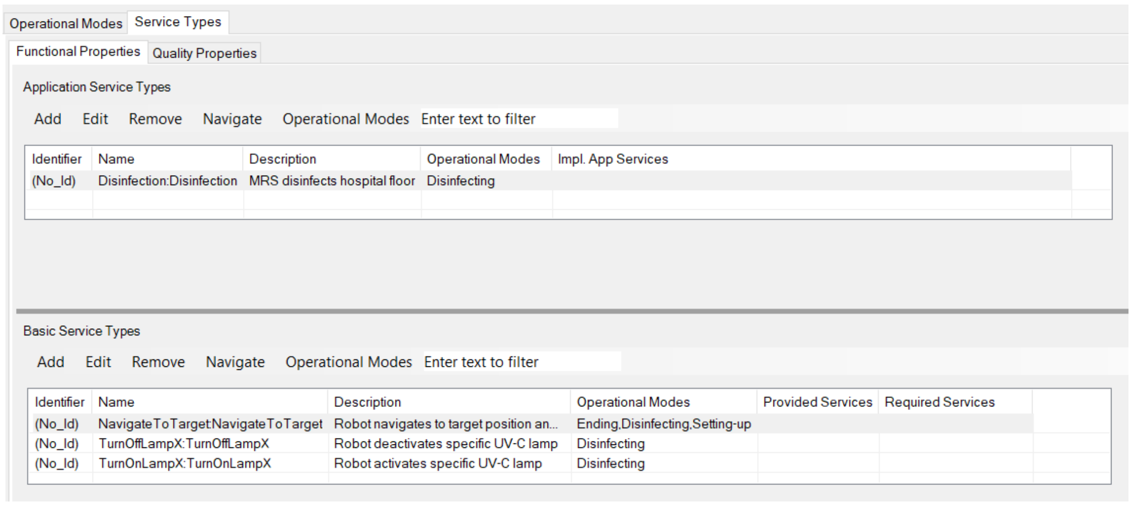
The Operational Modes tab allows adding, editing and removing operational modes. In the figure, an example based on the hospital disinfection case of 3 operational modes is shown. "Setting-up" refers to preparing the disinfection mission e.g. by initializing the robot positions and orientations, "Disinfecting" refers to the mode where the robots are performing the disinfection, and "Ending" refers to the robots returning to their mission completion position.



Identifier	Name	Description
(No_Id)	Disinfecting:Disinfecting	Disinfecting the hospital floor
(No_Id)	Ending:Ending	Robots return to mission completion position
(No_Id)	Setting-up:Setting-up	Robot initial positioning and orientation

Figure 24 - Editing a Collaborative System Group's Operational Modes in safeTbox

The Service Types tab allows specification of different types of services provided by the constituents to the CSG i.e. the CSs. The Functional Properties sub-tab allows each service type to be specified and be associated with a set of operational modes it is applicable for. In the example seen in Figure 25, an application-level service type, 'Disinfection', is defined in the first list, associated with the 'Disinfecting' operational mode. In the second list, CS-level services are specified; 'NavigateToTarget' refers to the robot navigating to a destination, 'TurnOffLampX' refers to the robot deactivating a specific UV-C lamp and 'TurnOnLampX' the opposite.



Application Service Types

Identifier	Name	Description	Operational Modes	Impl. App Services
(No_Id)	Disinfection:Disinfection	MRS disinfects hospital floor	Disinfecting	

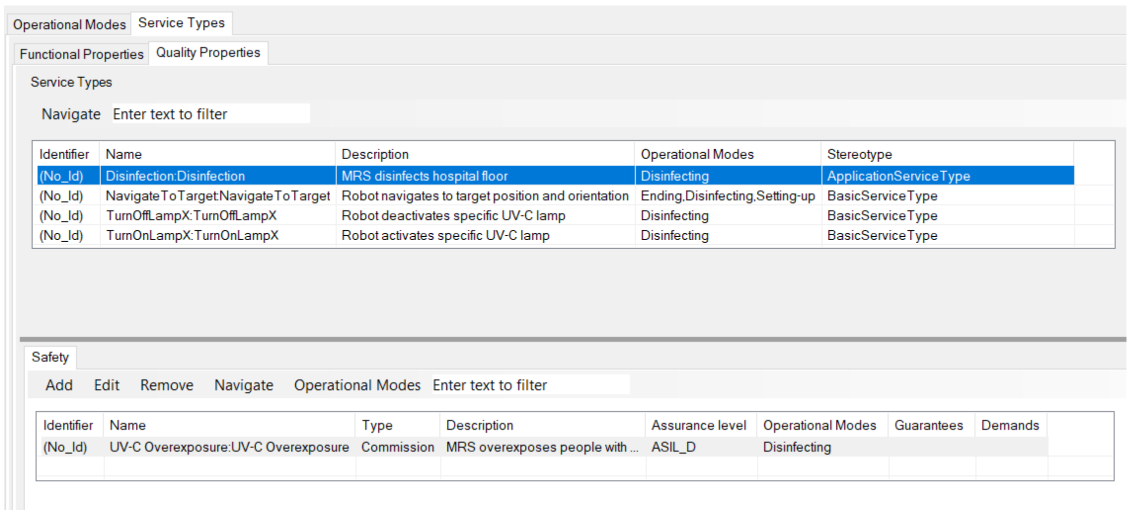
Basic Service Types

Identifier	Name	Description	Operational Modes	Provided Services	Required Services
(No_Id)	NavigateToTarget:NavigateToTarget	Robot navigates to target position an...	Ending,Disinfecting,Setting-up		
(No_Id)	TurnOffLampX:TurnOffLampX	Robot deactivates specific UV-C lamp	Disinfecting		
(No_Id)	TurnOnLampX:TurnOnLampX	Robot activates specific UV-C lamp	Disinfecting		

Figure 25 - Specifying CSG Service Types in safeTbox

In the Quality Properties sub-tab, each (application or basic) service's quality properties can be specified. For now, only 'safety' properties are supported semantically by the interface, but the interface can be used to define non-safety properties as well, as there is no semantical constraint.

To specify quality properties for a service, select it in the first list. The second list should update to reflect its current quality properties, if it has any. Use the buttons on top of the second list to add/edit/remove or set the operational modes a quality is relevant in. In the example seen in Figure 26, the application-level service 'Disinfection' has one safety quality property 'UV-C Overexposure', reflecting the situation where one or more people have been overexposed by the one or more of the MRS robots' lamps during the disinfection. The safety quality property is assigned a risk rating (based on the automotive standard ISO26262 currently, this will be adapted to the use cases), and can also be given a ‘failure mode’ type (‘Commission’ reflects that the overexposure happened due to the UV-C lamps being on when not intended) and assigned to an operational mode that applies to its service type owner.



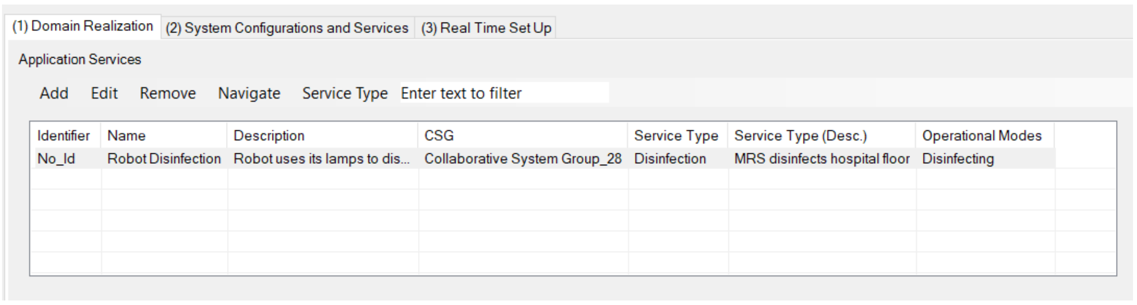
Identifier	Name	Description	Operational Modes	Stereotype
(No_Id)	Disinfection:Disinfection	MRS disinfects hospital floor	Disinfecting	ApplicationServiceType
(No_Id)	NavigateToTarget:NavigateToTarget	Robot navigates to target position and orientation	Ending,Disinfecting,Setting-up	BasicServiceType
(No_Id)	TurnOffLampX:TurnOffLampX	Robot deactivates specific UV-C lamp	Disinfecting	BasicServiceType
(No_Id)	TurnOnLampX:TurnOnLampX	Robot activates specific UV-C lamp	Disinfecting	BasicServiceType

Identifier	Name	Type	Description	Assurance level	Operational Modes	Guarantees	Demands
(No_Id)	UV-C Overexposure:UV-C Overexposure	Commission	MRS overexposes people with ...	ASIL_D	Disinfecting		

Figure 26 - Specifying CSG Service Type Quality Properties in safeTbox

For each robot modeled, a CS can be specified, using the Smart Menu in the (project) Browser. Once created, use the Smart Menu => Specify Configurations and Services actions to edit the properties of the CS.

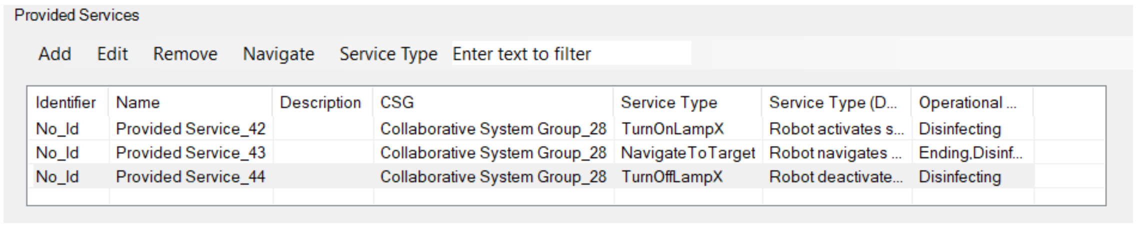
The first tab in the CS editing form is the Domain Realization, seen in .



Identifier	Name	Description	CSG	Service Type	Service Type (Desc.)	Operational Modes
No_Id	Robot Disinfection	Robot uses its lamps to dis...	Collaborative System Group_28	Disinfection	MRS disinfects hospital floor	Disinfecting

The first section of the tab allows us to specify which, if any, of the application-level services are supported by the given CS. In the case of our example, each robot contributes to the MRS’ application-level service of ‘Disinfection’. Therefore, an application-level service is added here, and its service type is set to match be the Disinfection service, defined previously in the CSG tab (see Figure 25). Use the buttons above the list area to add/edit/remove and set the service type.

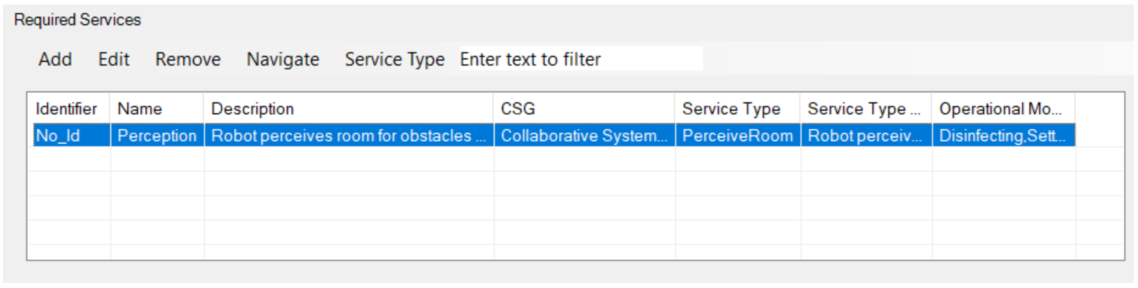
The second and third sections of the tab allows us to specify which Provided and Required Services are associated with this CS. In our example, this refers to the individual services a given robot can perform. As seen in Figure 27, a given robot can navigate to a target and turn its lamps on and off. The required services can be specified similarly, and reflect which services a robot depends on to yield its provided services. For example, a robot must perceive its environment to navigate and detect people, therefore its required service is ‘Perception’, as seen in Figure 28.


Provided Services

Add Edit Remove Navigate Service Type Enter text to filter

Identifier	Name	Description	CSG	Service Type	Service Type (D...	Operational ...
No_Id	Provided Service_42		Collaborative System Group_28	TurnOnLampX	Robot activates s...	Disinfecting
No_Id	Provided Service_43		Collaborative System Group_28	NavigateToTarget	Robot navigates ...	Ending,Disinf...
No_Id	Provided Service_44		Collaborative System Group_28	TurnOffLampX	Robot deactivate...	Disinfecting

Figure 27 - Specifying CS Provided Services in safeTbox

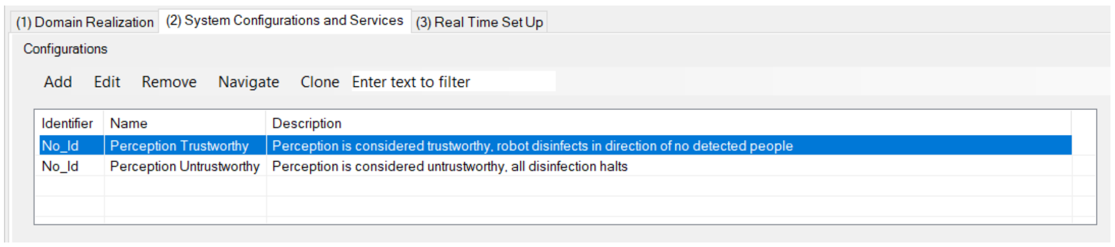

Required Services

Add Edit Remove Navigate Service Type Enter text to filter

Identifier	Name	Description	CSG	Service Type	Service Type ...	Operational Mo...
No_Id	Perception	Robot perceives room for obstacles ...	Collaborative System...	PerceiveRoom	Robot perceiv...	Disinfecting,Sett...

Figure 28 - Specifying CS Required Services in safeTbox

In the next tab, as seen in Figure 29, two configurations are specified for the given CS. In our minimal example, a robot can be considered to either be detecting a person around them, or not.


(1) Domain Realization (2) System Configurations and Services (3) Real Time Set Up

Configurations

Add Edit Remove Navigate Clone Enter text to filter

Identifier	Name	Description
No_Id	Perception Trustworthy	Perception is considered trustworthy, robot disinfects in direction of no detected people
No_Id	Perception Untrustworthy	Perception is considered untrustworthy, all disinfection halts

Figure 29 - Specifying CS System Configurations in safeTbox

3.3 SESAME-SPECIFIC EXTENSIONS AND FUNCTIONALITY

Building on prior work with DDIs, safeTbox has also been extended as part of joint work for WP4 and WP7 to import/export EDDIs of its contained models. Simply use the Smart Menu => Import/Export to EDDI File… to parse or serialize an EDDI. This feature uses the Tool Adapter concept, explained later in Section 5.2. (Note that the interface lists DDIs instead of EDDIs, but the models currently supported are indeed updated to the new EDDI metamodels, as described in **D4.2/D5.2: Safety/Security ODE Specification**).

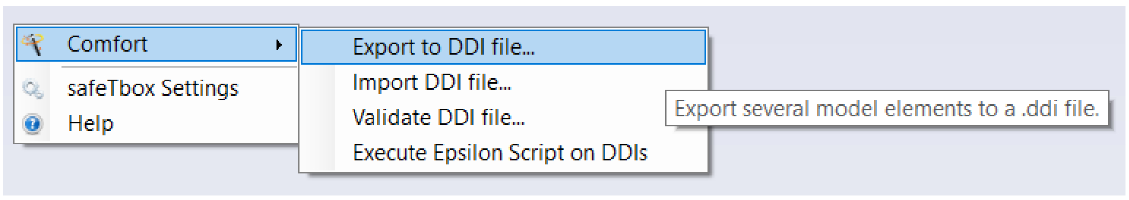


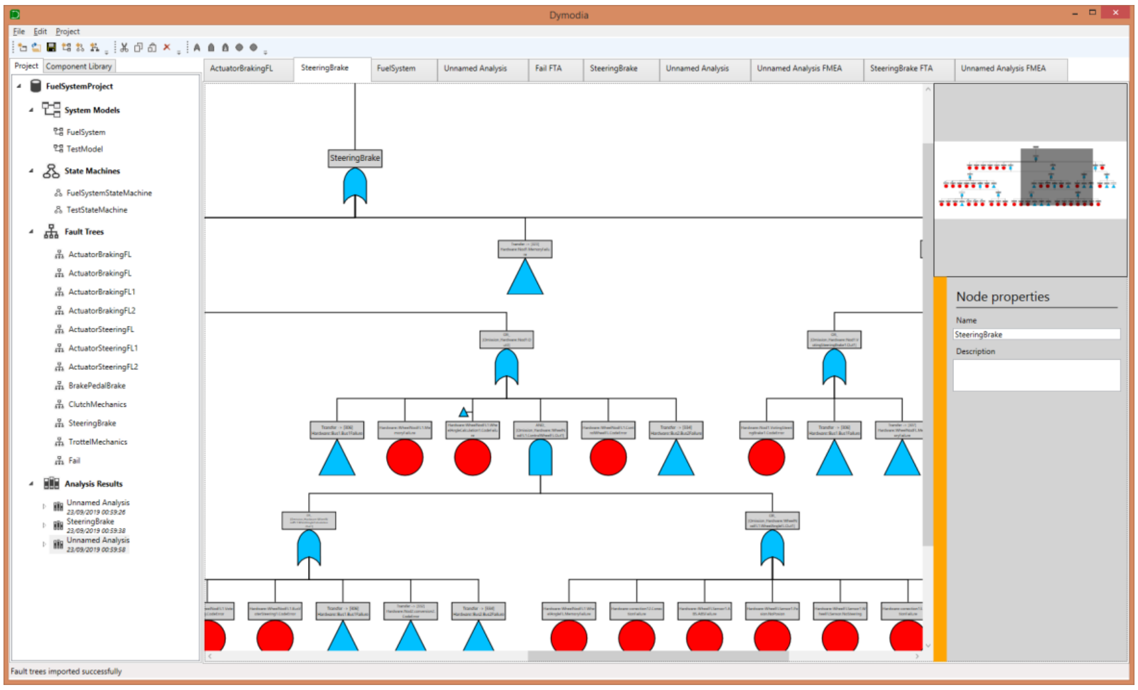
Figure 30 - safeTbox (E)DDI I/O

4. DYMODIA

4.1 WHAT IS DYMODIA

Like HiP-HOPS and safeTbox, the Dymodia tool[7] is a model-based safety analysis tool that supports common analyses like FTA and FMEA. However, unlike the other two tools, Dymodia integrates architectural, behavioural, and failure modelling and analysis as part of a single platform. In this way, it is not reliant on an external modelling package like Matlab Simulink or Enterprise Architect, which limits the other tools to a uni-directional flow of separate steps: model, then analyse. By combining the model and the failure analysis as part of the same package, Dymodia enables tighter integration and allows a more rapid iterative process where design changes can quickly be reflected in the analysis results

Another difference is that unlike the primarily static analyses of HiP-HOPS and safeTbox, Dymodia also supports the use of state machines to model dynamic system behaviour, which can then be linked directly to the architectural and failure models. Standalone dynamic fault trees can also be defined to model failure-related behaviour that does not correspond directly to system architecture elements or system states, e.g. when modelling more complex hazardous scenarios.



Fault trees imported successfully

Figure 31 - Dymodia UI

Both FTA and FMEA are supported. FTA results track changes in state, so that not only combinations but also sequences of events that cause failures can be included. The analysis results themselves are also integrated as part of the package, allowing e.g. users to click on a result and be taken to the part of the system model that generated it.

However, Dymodia is still under development and is not as mature as the other tools. It also features a much more complex input format, since it combines several elements in

[7] https://dymodiansystems.com/dymodia/

a package more like an IDE rather than a simple modelling tool. This makes it more difficult to adapt to the ODE.

4.2 How it works

Dymodia follows roughly the same process as the preceding tools; the primary difference is that it all takes place within the same interface. Three types of models are possible to create:

- System models, to describe system architecture;
- State machines, to describe dynamic behaviour;
- Standalone fault trees, to describe failure logic that is not system-specific.

System models consist of components and connections, with ports serving as the interface:

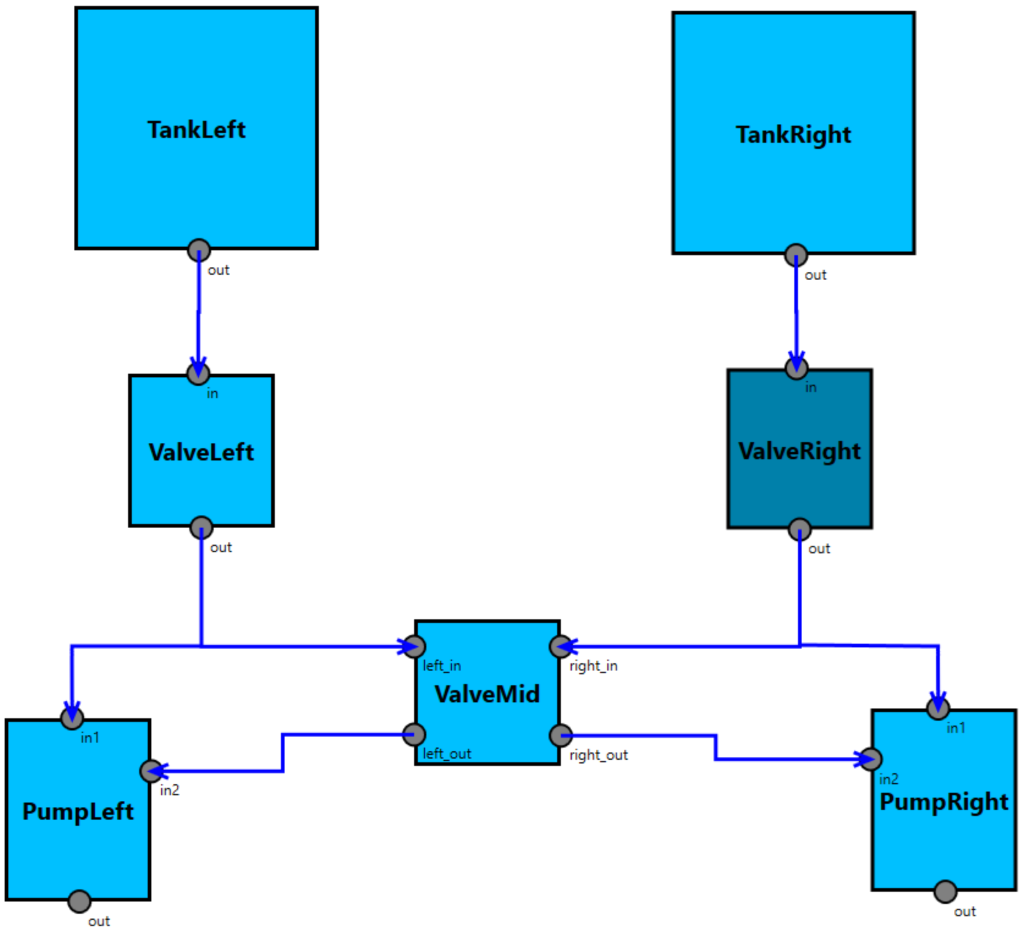
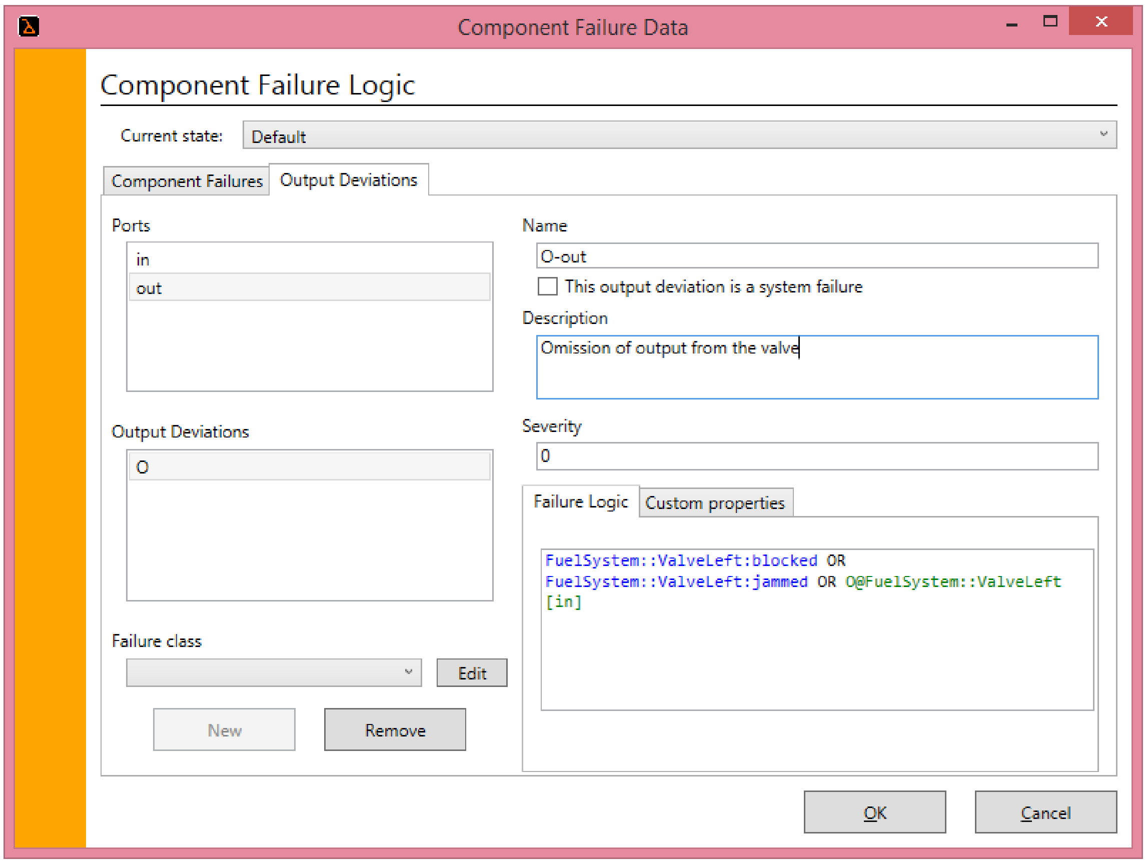


Figure 32 - Example Dymodia system model

As with the other tools, components can be annotated with failure data to describe internal failure modes and deviations at their outputs. A particular grammar is used for the logic, effectively forming a kind of simple domain-specific language.



Current state: Default

Ports: in, out

Output Deviations: O

Failure class:

Name: O-out

This output deviation is a system failure

Description: Omission of output from the valve

Severity: 0

```
FuelSystem::ValveLeft:blocked OR
FuelSystem::ValveLeft:jammed OR O@FuelSystem::ValveLeft
[in]
```

Figure 33 - Failure data in Dymodia

State machines can also be created and linked to system models. When linked in this way, the states defined by the state machine are assumed to be the different modes of operation for the system architecture, thus allowing different failure logic to be defined for each state per component.

Transitions in the state machine can have a variety of triggers, including failure modes and output deviations defined within the system architecture model. They can also be triggered by standalone fault trees or even by transitions in other state machines, thus allowing a form of hierarchical state machine to be created.

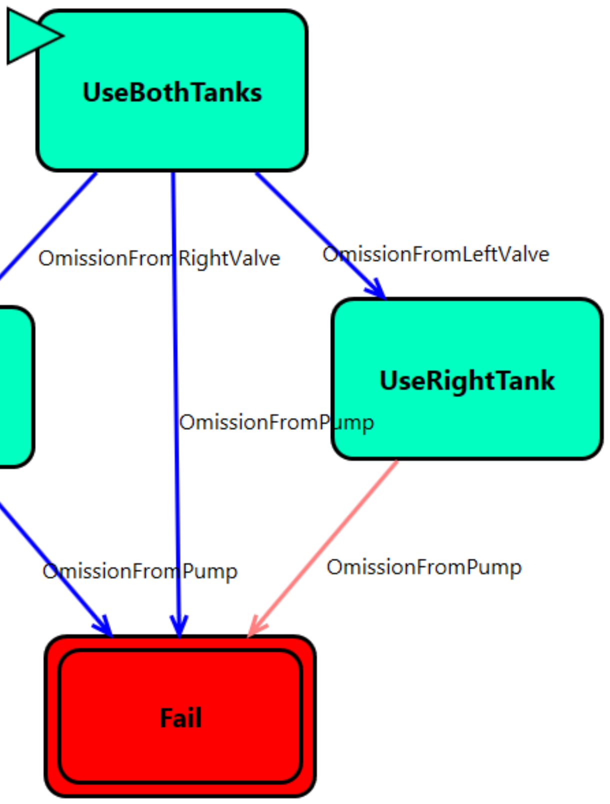
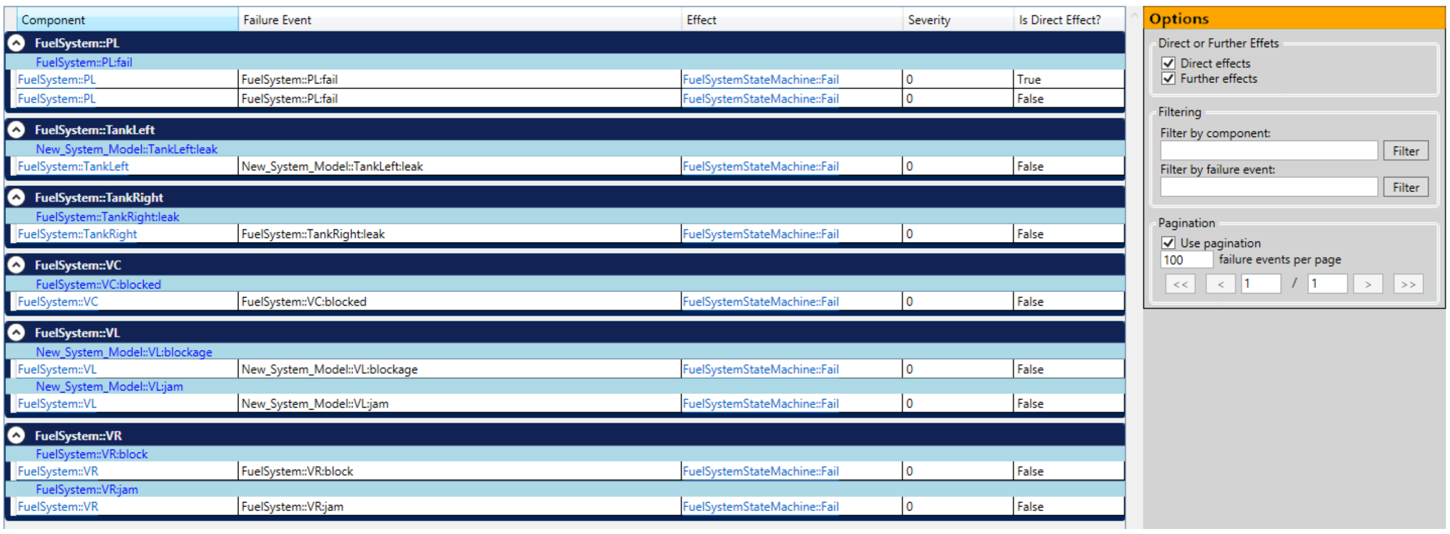


Figure 34 - Dymodia state machine

Once created, the model(s) can be analysed. As with HiP-HOPS, both FTA and FMEA is conducted. Unlike HiP-HOPS, these analyses can incorporate elements from multiple models, including more than one model of each type.



Component	Failure Event	Effect	Severity	Is Direct Effect?
FuelSystem::PL				
FuelSystem::PL:fail				
FuelSystem::PL	FuelSystem::PL:fail	FuelSystemStateMachine::Fail	0	True
FuelSystem::PL	FuelSystem::PL:fail	FuelSystemStateMachine::Fail	0	False
FuelSystem::TankLeft				
New_System_Model::TankLeft:leak				
FuelSystem::TankLeft	New_System_Model::TankLeft:leak	FuelSystemStateMachine::Fail	0	False
FuelSystem::TankRight				
FuelSystem::TankRight:leak				
FuelSystem::TankRight	FuelSystem::TankRight:leak	FuelSystemStateMachine::Fail	0	False
FuelSystem::VC				
FuelSystem::VC:blocked				
FuelSystem::VC	FuelSystem::VC:blocked	FuelSystemStateMachine::Fail	0	False
FuelSystem::VL				
New_System_Model::VL:blockage				
FuelSystem::VL	New_System_Model::VL:blockage	FuelSystemStateMachine::Fail	0	False
New_System_Model::VL:jam				
FuelSystem::VL	New_System_Model::VL:jam	FuelSystemStateMachine::Fail	0	False
FuelSystem::VR				
FuelSystem::VR:block				
FuelSystem::VR	FuelSystem::VR:block	FuelSystemStateMachine::Fail	0	False
FuelSystem::VR:jam				
FuelSystem::VR	FuelSystem::VR:jam	FuelSystemStateMachine::Fail	0	False

Options

Direct or Further Effets
- ✓ Direct effects
- ✓ Further effects

Filtering
- Filter by component:
- Filter by failure event:

Pagination
- ✓ Use pagination
- 100 failure events per page
- 1 / 1

Figure 35 - Dymodia FMEA output

4.3 SESAME-SPECIFIC EXTENSIONS AND FUNCTIONALITY

As it is the least mature tool and still in development, it is harder to use Dymodia for the purpose of creating ODE-compliant EDDI models. However, its capabilities — particularly for dynamic analysis with integrated state machines — makes it an attractive prospect. Furthermore, because Dymodia is inspired by similar preceding tools like HiP-HOPS, it too uses a metamodel that is broadly compatible with ODE.

The reverse is unlikely to be possible, however, because of the fact that Dymodia is an integrated modelling and analysis tool. The models imported by HiP-HOPS and safeTbox do not need to know about e.g. the position of each component or other display information (like colours or images); this is instead handled by the external modelling tool.

This same extra detail makes it more challenging to extract the relevant information out of a Dymodia file to generate a corresponding ODE model. For that reason, only Dymodia state machines can currently be converted to ODE models.

5. ODE Tools

Two model converters are available for conversion of other models/file formats to ODE models. In addition, this section also describes the minor changes that have been made to the ODE metamodel since the publication of **D4.2 & D5.2 Safety/Security-Targeted ODE and EDDI specification**.

5.1 Recent Changes to the ODE

The ODE metamodel was slightly changed after issues were found with the ODE::Dimension and ODE::ConSert packages. In addition, a few entities were renamed to avoid name collision with SACM elements.

5.1.1 Removal of SubSetType enumeration from ODE::Dimension package

In the older version of the ODE, as described in **D4.2 & D5.2 Safety/Security-Targeted ODE and EDDI specification**, the ODE::Dimension package included the enumeration "SubsetType" that was used as an attribute for a general dimension. The idea was to be able to specify how two dimensions should be compared, i.e. one being a subset of the other.

This SubsetType enumeration is only relevant when using dimensions in the context of ConSerts, where it has to be checked to see if the demand could be fulfilled by the provided guarantee. However, dimensions are also referred to from other packages and elements where the subset type is not relevant at all, e.g. from within a Bayesian Network NodeState. In general, dimensions should be generic and be usable by any other element types where comparisons based on numeric, categorical, or binary dimensions are relevant, and therefore they should not contain attributes that are relevant for a specific modelling approach (like ConSerts).

In the new ODE version, this subset type aspect has been moved from the ODE::Dimension package to the ODE::ConSert package. The SubsetType described how the values of provided guarantee have to be related to the expected values defined for the demands. For the new revision, the possible relations are:

1. **Guarantee in Demand (GiD):** Values provided by the guarantee are a subset of the expected values defined for the demand.

2. **Demand in Guarantee (DiG):** Expected values defined for the demand are a subset of the guarantee → "Demand in Guarantee".

3. **Equal:** Both values (the one expected by the demand and the one provided by the guarantee) have to be equal.

4. **Intersection not** Empty: Assuming the guarantee and demand both define ranges, the demand would be fulfilled if the intersection of both ranges is not empty.

Thus the "DemandGuaranteeDimensionRelation" type and associated enumeration were introduced to define how the relation for each guarantee/demand dimension pair is expected to be fulfilled. For example, assume a demand requires a minimum safe operating distance between a robot and human workspace to fulfil requirements —

"safe_distance > 1 metre". If the robot guarantees that the safe distance will be at least 2 metres, then the demand is a subset of the guarantee (Demand in Guarantee) and the requirement is fulfilled. Alternatively, a demand may require a maximum operating temperature of 50° C, while the robot guarantees a max temperature of 40° C. In this case, the guarantee should be a subset of the demand (Guarantee in Demand) to ensure safe operation.

Figure 36 depicts how the newly introduced element type and enumeration are integrated into the ODE::ConSert package.

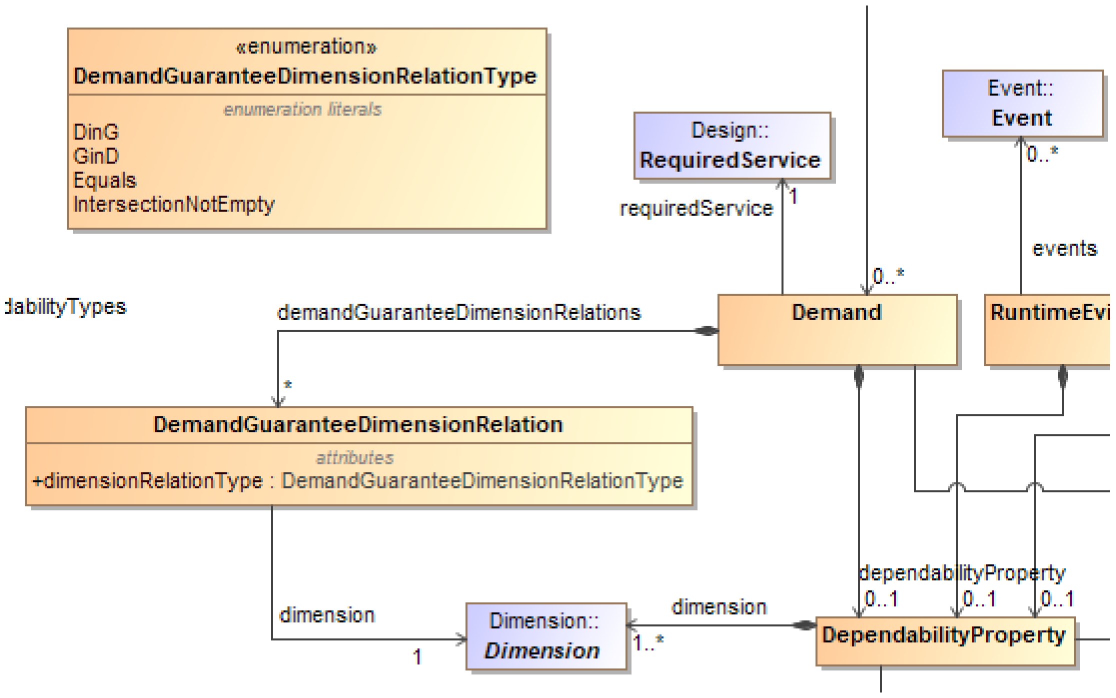


Figure 36 - DemandGuaranteeDimensionRelation in ODE::ConSert package

5.1.2 Numeric dimension

The old "Numeric Dimension" from the previous ODE version was found to be unintuitive and not detailed enough to model different types of numeric dimensions. With the older version, it was only possible to define a numeric dimension as a bounded range (lower bounded, upper bounded, or both). Thus it was not able to explicitly cover one single value (e.g., for comparing numbers to be equal). Furthermore, it was also not possible to specify bounds inclusivity — whether or not the number that defines the boundary should be included in the boundary (i.e., less/greater than **or** equal).

The refactored ODE::Dimension package including the new numeric dimension definition is depicted in Figure 37. The specific types of dimensions have been renamed in that they now end with the "Dimension" suffix and additionally the "Numeric" is now called "NumericRangeDimension". This numeric representation now does not refer to one or more "Limits" (as in the old version) but can have a starting and/or an ending "NumericBoundary". Further, each numeric boundary can be specified to be inclusive, which allows the "<=" or ">=" checks using the "NumericRangeDimension".

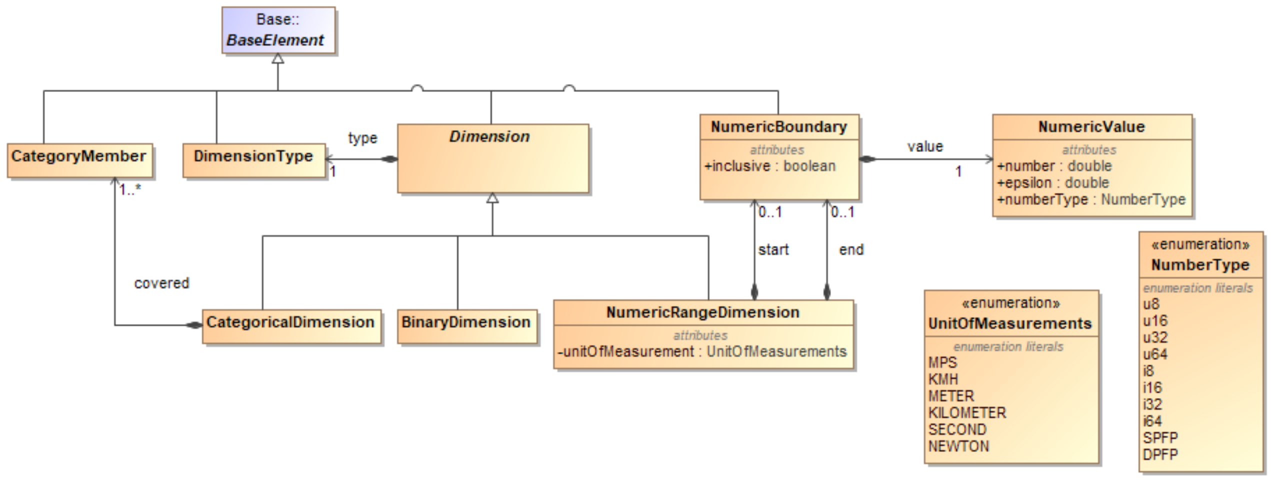


Figure 37 - Updated ODE::Dimension package

The actual numeric value which defines the boundary is no longer specified by a simple attribute of the type of double, but is instead represented by the referenced "NumericValue" element type. For this "NumericValue" type, besides the actual value, an epsilon value as well as the number type can be defined. Given this new representation, it is now possible to define following classes of numeric dimensions:

- A single numeric value (e.g., for checking equality): starting and ending "NumericBoundary" are set to the same value, where both boundaries have the "inclusive" flag set to true.
- An upper boundary to check whether another numeric dimension is less than (or equal to) this defined boundary: only the ending "NumericBoundary" is set with its "inclusive" flag set as appropriate.
- A lower boundary to check whether another numeric dimension is greater than (or equal to) this defined boundary: only the starting "NumericBoundary" is set with its "inclusive" flag set as appropriate.
- A closed or (half-)open interval: both starting and ending boundaries are set to different numbers with the "inclusive" set as desired. This allows us to check if other numeric range dimensions are contained in the interval or even if there are intersections between two intervals.

5.1.3 Minor name changes

A few minor changes were made to the names of some ODE elements, whether for clarity or to avoid clashes with similarly named elements elsewhere. These are briefly described below:

- `allocatedRequiredService` in ProvidedService was renamed to `allocatedRequiredServices` to reflect its 0..* cardinality;

- Similarly, `guaranteePropagation`, `runtimeEvidence`, and `conSertGate` in ConSert were renamed to `guaranteePropagations`, `runtimeEvidences`, and `conSertGates`;

- `Event` was renamed to `OdeEvent` to avoid collision with the Event in SACM;

- `Action` in the TARA package was renamed to `ThreatAction` to avoid collision with the new `Action` in the FailureLogic package;

5.2 COMMON TOOL ADAPTER

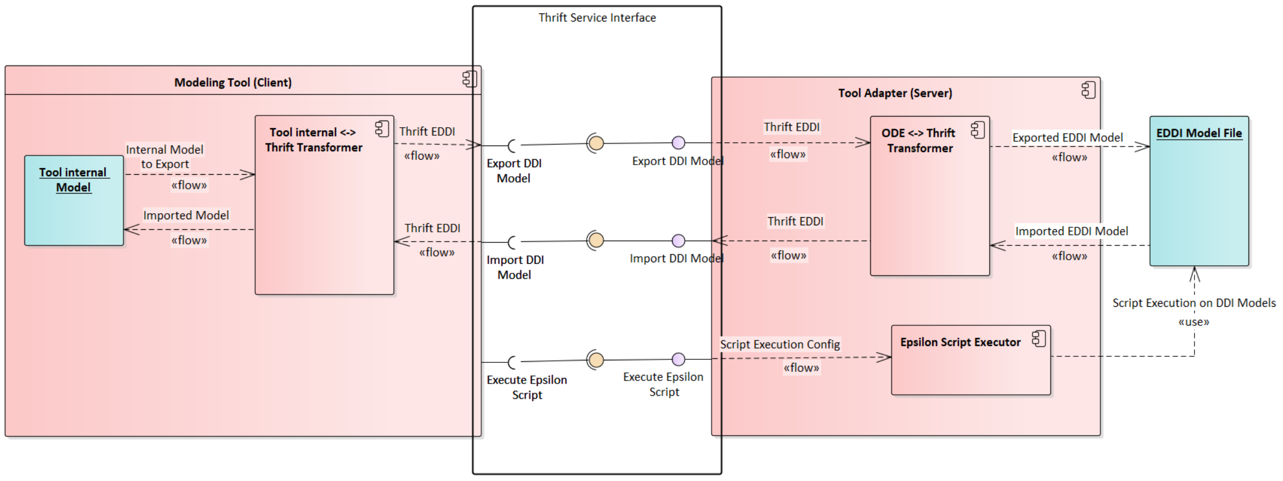


Figure 38 - Tool Adapter Service Interface

The Tool Adapter is a service-providing software component that allows importing and exporting EDDI models that conform to the ODE metamodel into or from an arbitrary software (e.g. modelling software). Additionally, it provides a service for executing Epsilon scripts on existing EDDI models. The standardized, language-agnostic and generic service interface is realized using Apache Thrift. Figure 38 shows the service interface provided by the Tool adapter and how it is used by modelling tools. This chapter describes the service interface definition and usage in a modelling tool. For further information about the Tool Adapter and its architecture, please refer to SESAME project deliverable **D5.4: Tailorability of EDDIs** (Section 3.1).

Apache Thrift is used as an intermediate service interface provider. The open-source framework comes with its own Interface Definition Language (IDL). Using the IDL allows us to define service interfaces and the data types that shall be exchanged through the services in a language-independent manner. The Thrift compiler is then used to generate language-specific source code from the IDL definitions. This source code (e.g. Java, C#, C++, etc.) can then be used in server- or client-side applications. For instance, the Tool Adapter (Server) and modelling tools (Client) use Thrift to establish a standardized service provision and consumption. Sending and receiving models to and from the Tool Adapter needs a transformation between the tool internal model representation and the Thrift data types on the client side (see Section 5.2 for further information).

Figure 39 provides a brief overview on how generated code is integrated into client and server software solutions. The ServiceInterface.thrift element represents the service and data type definition using IDL. On each side, the generated Code is integrated and the language-specific Thrift library is used to invoke the client or server-side specific request and response procedures.

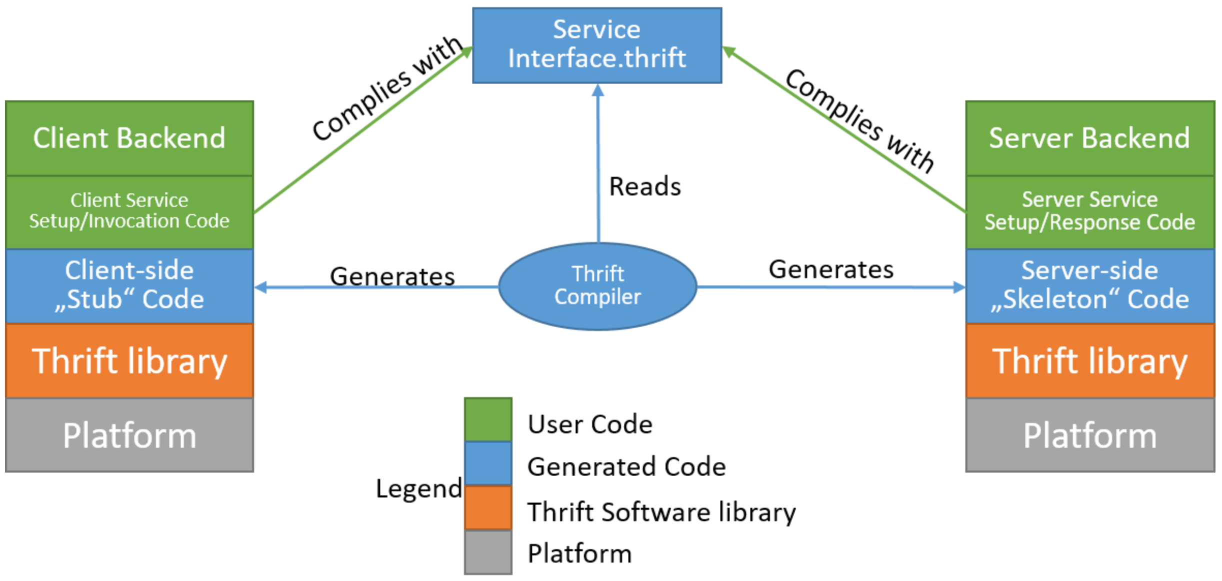


Figure 39 - Overview of Apache Thrift (DEIS D4.2)

As mentioned above, within the Tool Adapter, Apache Thrift is used to define the service interface for importing/exporting EDDI models into/from modelling tools and for executing Epsilon scripts on EDDI models. Furthermore, all ODE metamodel elements and their relationships are defined as Apache Thrift datatypes. Thus, generating language-specific code out of the Thrift contract defined in IDL means the ODE data structure including the server and client code for providing and consuming mentioned services is available in the specific language and is ready to be used in software that shall communicate with each other.

5.2.1 User Guide

More information on setting up and using the Common Tool Adapter can be found in the **SESAME Common Tool Adapter user guide** located in the **design_time_eddis/tool_adapter** directory of the SESAME git repository.

5.3 ODE FRAMEWORK UPDATER

As described in the preceding section, the Common Tool Adapter is a software component providing a service that allows importing and exporting EDDI models from arbitrary software tools. The exchanged EDDI models conform to the ODE metamodel. Figure 40 depicts the Tool Adapter framework containing the components required for the import and export functionality.

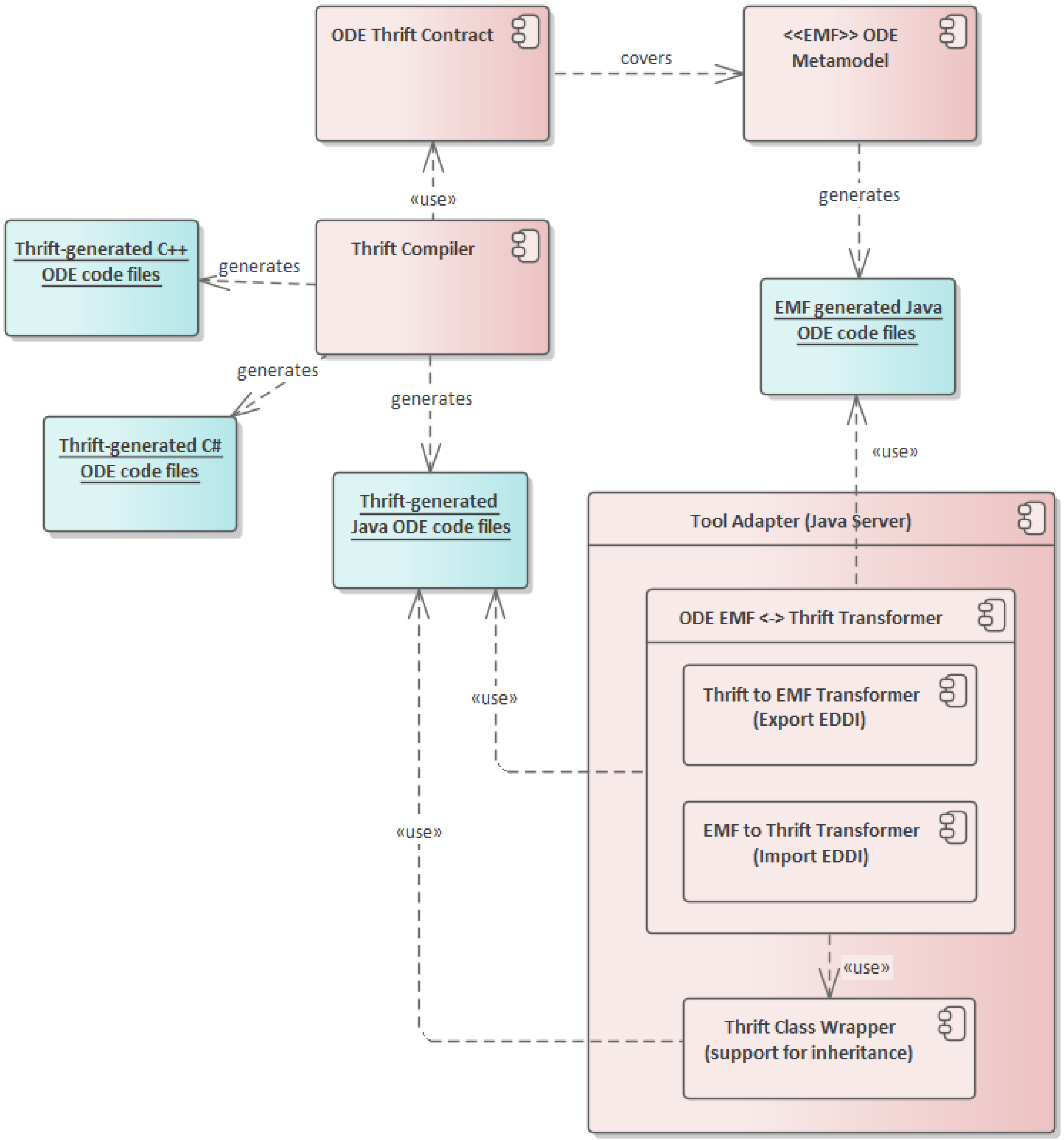

ODE Thrift Contract
covers
<<EMF>> ODE Metamodel
«use»
generates
Thrift-generated C++ ODE code files
generates
Thrift Compiler
EMF generated Java ODE code files
generates
generates
Thrift-generated C# ODE code files
«use»
Thrift-generated Java ODE code files
Tool Adapter (Java Server)
ODE EMF <-> Thrift Transformer
«use»
Thrift to EMF Transformer (Export EDDI)
«use»
EMF to Thrift Transformer (Import EDDI)
«use»
Thrift Class Wrapper (support for inheritance)

Figure 40 - Tool Adapter framework

Apache Thrift is used to provide programming language-independent service interfaces (not depicted in figure) to support the EDDI export and import functionality from and to modelling tools. The model element types defined in the ODE Metamodel are adapted in internal data structures defined in the ODE Thrift Contract to use the adapter's services and transfer the models from and to the external modelling tools.

Using the Thrift compiler, programming-language-specific source code files (e.g. Java, C#, C++) are generated for the services and data structures defined in the Thrift contract. These code files can again be integrated into modelling tools and into the Tool Adapter (written in Java) to establish connections between server (Tool Adapter) and clients (modelling tools) and invoke the EDDI import and export services. The ODE metamodel is defined using the Eclipse Modeling Framework, which allows generation of Java classes for each defined element type.

When the services for exporting or importing EDDI models are invoked, the Tool Adapter receives or sends out the model from and to modelling tools in the Thrift representation respectively. The exchangeable EDDI model is serialized in XML style using the EMF representation of the model. Therefore, within the Tool Adapter a transformation between the Thrift representation and the EMF representation has to be performed. Additionally, wrapper classes for the Thrift data structures are defined for allowing to use inheritances during these transformations.

When the ODE metamodel is updated or generally changed, the following steps have to be performed in order to further support the export and import functionalities of the Tool Adapter:

1. Update the Thrift contract.

    a. Needs additional workaround steps to imitate inheritance relationships in the ODE metamodel as Thrift's interface definition language does not support defining inheritance for data types natively. Therefore, abstract datatypes for super types are defined and unions together with enumerations are used to point out to the actual subtype that is wrapped within the abstract datatype.

2. Re-generate the programming-language-specific code files using the Thrift compiler.

3. Adapt the wrapper classes in the Tool Adapter.

4. Update the EMF-to-Thrift transformation algorithms.

5. Update the Thrift-to-EMF transformation algorithms.

Besides being time-consuming, the above-mentioned steps are also error-prone, which further increases time spent debugging the adapter tool. One method to mitigate errors and development resources is to automatically provide an update of the Thrift contract and the language-specific generated code files, the Thrift wrapper classes, and the transformation algorithms, after the ODE metamodel has been changed.

This has been implemented in the form of the **ODE Framework Updater**. Figure 41 depicts the workflow of the ODE Framework Updater, which follows this approach. First, the (updated) ODE metamodel is parsed and the defined element types are validated such that all element types have unique names. An EMF-Thrift Mapping Metamodel specifies the Thrift- and EMF-specific metamodeling structures and how they can be mapped to each other. The mapping model is required, as the lack of native inheritance definition support in Thrift does not allow a one-to-one mapping between the datatypes defined in Thrift and EMF. An instance of this mapping metamodel is created ("Mapping Model" in the figure), where the EMF and Thrift representation of all ODE-related element types are mapped to each other. This mapping model is then used to generate the new Thrift contract and the necessary transformation and wrapper class code files. Finally, the generated code files are copied to the ODE Tool Adapter code base to ensure it supports the export and import functionalities for the updated ODE metamodel.

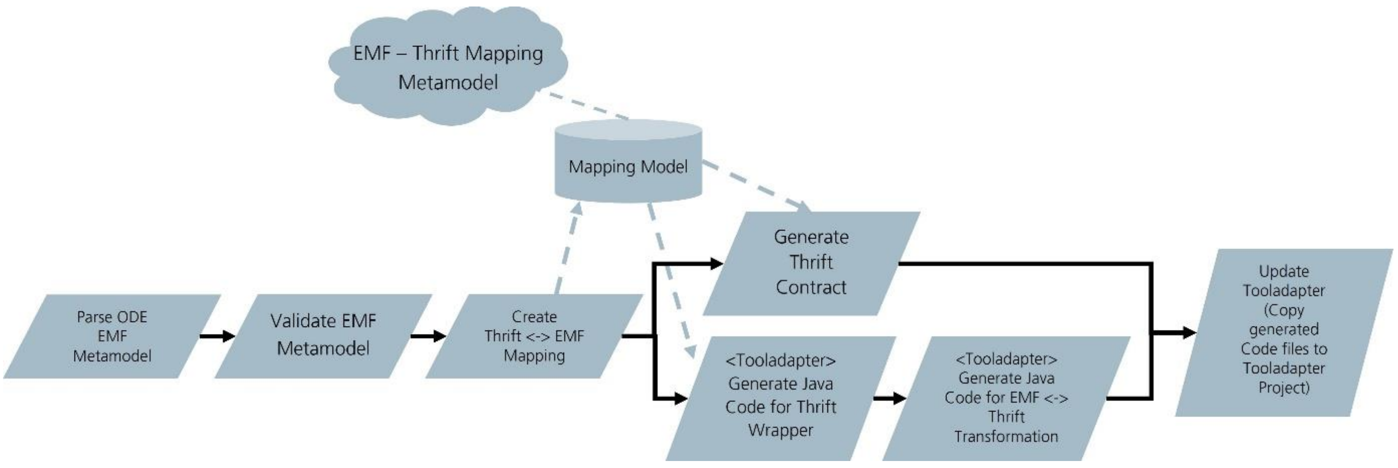
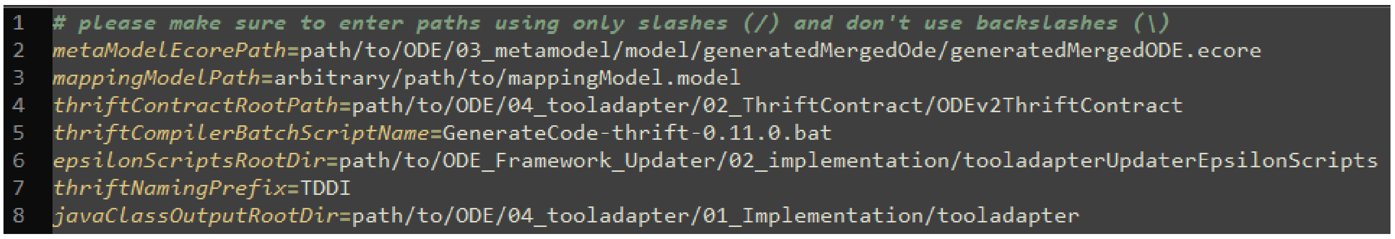

EMF – Thrift Mapping Metamodel
Mapping Model
Generate Thrift Contract
Parse ODE EMF Metamodel
Validate EMF Metamodel
Create Thrift <-> EMF Mapping
<Tooladapter> Generate Java Code for Thrift Wrapper
<Tooladapter> Generate Java Code for EMF <-> Thrift Transformation
Update Tooladapter (Copy generated Code files to Tooladapter Project)

Figure 41 - Workflow of automatic ODE Framework Updater

To use the ODE Framework Updater, a configuration file (see example in Figure 42) has to be prepared. Most of the configuration entries define paths to required directories and files, including the updated metamodel and the location where the mapping model should be serialized to.



```
# please make sure to enter paths using only slashes (/) and don't use backslashes (\)
metaModelEcorePath=path/to/ODE/03_metamodel/model/generatedMergedOde/generatedMergedODE.ecore
mappingModelPath=arbitrary/path/to/mappingModel.model
thriftContractRootPath=path/to/ODE/04_tooladapter/02_ThriftContract/ODEv2ThriftContract
thriftCompilerBatchScriptName=GenerateCode-thrift-0.11.0.bat
epsilonScriptsRootDir=path/to/ODE_Framework_Updater/02_implementation/tooladapterUpdaterEpsilonScripts
thriftNamingPrefix=TDDI
javaClassOutputRootDir=path/to/ODE/04_tooladapter/01_Implementation/tooladapter
```

Figure 42 - Example ODE Framework Updater Configuration File

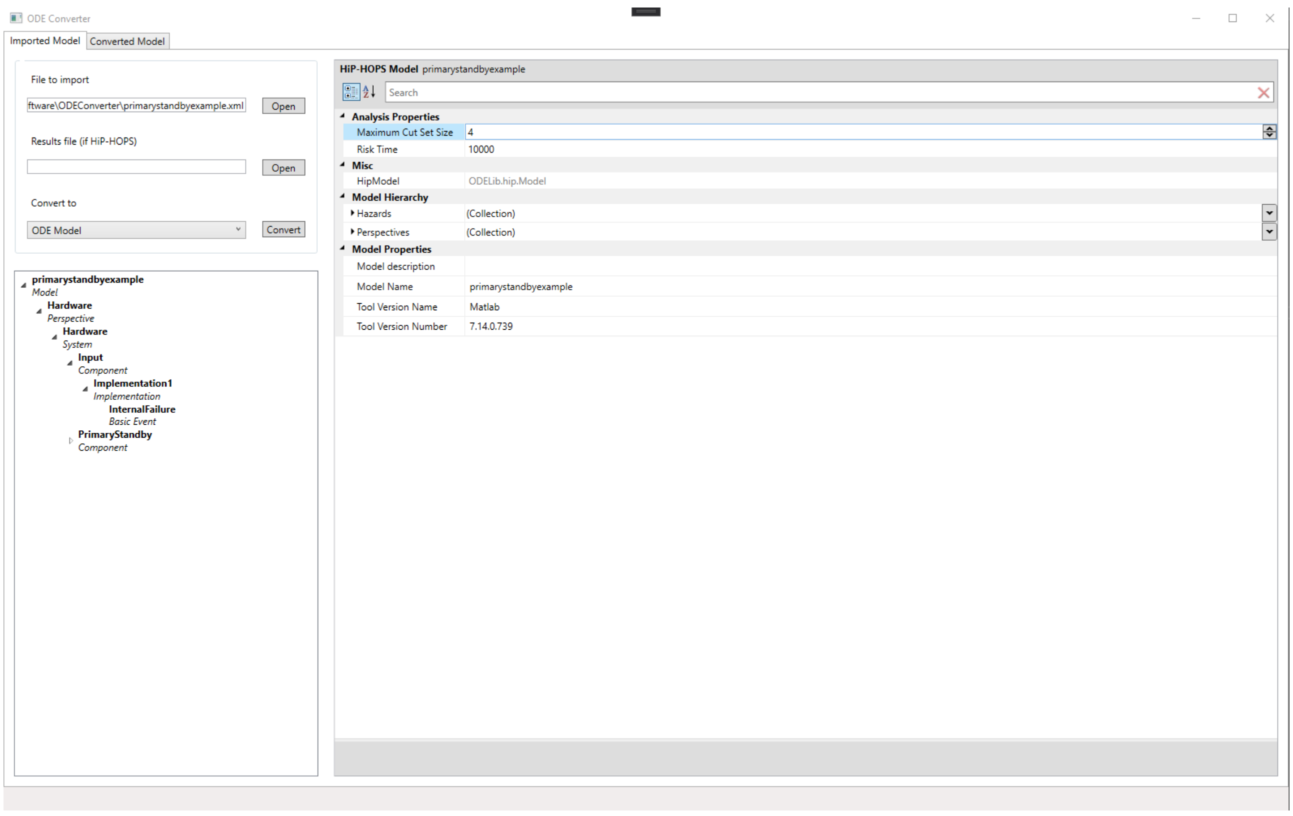
5.4 EDDI Editor

5.4.1 The ODE Model Converter

The EDDI Editor was initially developed as a straightforward model transformation tool, the "ODE Model Converter". Since none of the tools described in sections 2-4 support the ODE directly, converters are required to allow interoperability. In addition to the Common Tool Adapter described in section 5.2, the ODE Model Converter was developed as a standalone converter to enable direct conversion from HiP-HOPS to ODE. The motivation for this lay in the fact that to capture the full output of HiP-HOPS, two files are required:

- The input XML file, which contains the system architecture and local failure data;
- The output XML file, which contains the FTA and FMEA analyses.

For the purposes of a simple design time analysis, the second file is sufficient. In order to support the generation of runtime EDDIs, however, more information about the system itself is required. Thus to enable a true conversion of a full HiP-HOPS model, conversion of both separate files are necessary: one containing the system architecture and the other containing the failure propagation model and analysis results. The ODE Model Converter fulfilled this purpose by allowing both files to be imported and exported as a single, combined ODE model. Equally, it also allowed just one file to be imported, though this limits the usefulness of the converted model in turn.



File to import: ftware\ODEConverter\primarystandbyexample.xml

Results file (if HiP-HOPS):

Convert to: ODE Model

primarystandbyexample *Model*
- Hardware *Perspective*
  - Hardware *System*
    - Input *Component*
      - Implementation1 *Implementation*
        - InternalFailure *Basic Event*
    - PrimaryStandby *Component*

HiP-HOPS Model primarystandbyexample

Analysis Properties	
Maximum Cut Set Size	4
Risk Time	10000
Misc	
HipModel	ODELib.hip.Model
Model Hierarchy	
Hazards	(Collection)
Perspectives	(Collection)
Model Properties	
Model description	
Model Name	primarystandbyexample
Tool Version Name	Matlab
Tool Version Number	7.14.0.739

Figure 43 - ODE Converter – imported model

The Converter additionally allowed limited editing of certain properties of both the imported model(s) and the generated ODE model (prior to export).

5.4.2 The EDDI Editor

The EDDI Editor is a development of the original model converter with a greater range of functionality. It is intended to act as a form of glue between the different safety analysis tools and runtime EDDI generators and support a degree of tool heterogeneity.

Functionality includes:

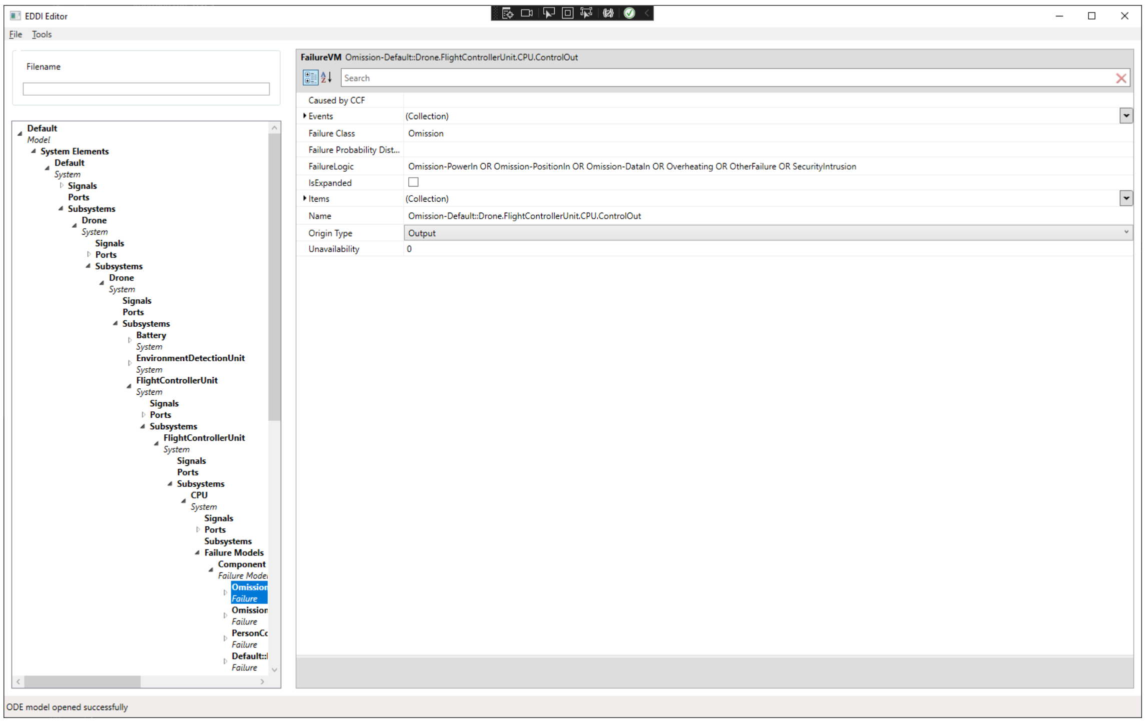
- Import of HiP-HOPS models (either separate system architecture or analysis models, or both combined) and conversion of them to ODE models;
- Import of Dymodia state machines and conversion of them to ODE models;
- Import of safeTbox EDDI models (containing both system architecture and generated fault trees);
- Loading ODE models directly (e.g. system architectures, fault trees, FMEAs, state machines);
- Simple editing of entity properties in each model (both before and after conversion);
- Merging/combining of ODE models, e.g. by importing a model/subsystem hierarchy to replace an existing (empty) placeholder system, or adding new failure models etc;
- (In development) experimental validation and test execution of the resulting models.

Note that the EDDI Editor is not a modelling or analysis tool in itself: models must still be created and/or analysed in an appropriate tool first, like HiP-HOPS or safeTbox. Similarly, some degree of further post-processing (e.g. code generation) is necessary to produce an actual runtime EDDI.

The main interface of the EDDI Editor is similar to the earlier Converter, except rather than having two tabs — one for the imported HiP-HOPS model, one for the converted ODE model — the main interface now displays only the converted ODE model. Editing of the imported model is now performed in a separate window as part of the import process.

More information on the use of the EDDI Editor can be found in its user guide in the **design_time_eddis/eddi_editor** directory of the SESAME git repository. However, a subset is provided here to present some of its key capabilities.

The interface displays a hierarchical view of the system and its failure models on the left, while on the right is a properties grid. Different elements can be selected from the hierarchy, allowing their properties to be viewed and edited in the grid on the right. All simple properties can be edited; however, only very limited changes to the actual structure (i.e., adding or removing model elements) are possible.



```
Default
Model
  System Elements
    Default
    System
      Signals
      Ports
      Subsystems
        Drone
        System
          Signals
          Ports
          Subsystems
            Drone
            System
              Signals
              Ports
              Subsystems
                Battery
                System
                EnvironmentDetectionUnit
                System
                FlightControllerUnit
                System
                  Signals
                  Ports
                  Subsystems
                    FlightControllerUnit
                    System
                      Signals
                      Ports
                      Subsystems
                        CPU
                        System
                          Signals
                          Ports
                          Subsystems
                          Failure Models
                            Component
                            Failure Mode
                              Omission
                              Failure
                              Omission
                              Failure
                              PersonC
                              Failure
                              Default::
                              Failure
```

FailureVM Omission-Default::Drone.FlightControllerUnit.CPU.ControlOut

Caused by CCF	
Events	(Collection)
Failure Class	Omission
Failure Probability Dist...	
FailureLogic	Omission-PowerIn OR Omission-PositionIn OR Omission-DataIn OR Overheating OR OtherFailure OR SecurityIntrusion
IsExpanded	☐
Items	(Collection)
Name	Omission-Default::Drone.FlightControllerUnit.CPU.ControlOut
Origin Type	Output
Unavailability	0

ODE model opened successfully

Figure 44 - EDDI Editor

Menu commands allow for file manipulation, including opening an existing ODE model or importing a new model from a different tool (e.g. HiP-HOPS). Additionally, right-clicking on the system hierarchy opens context-dependent import options:

- **Import model as subsystem**: opens the import dialog and lets you import a model as a new subsystem under the current (right-clicked) system.
- **Import model and replace this subsystem**: opens the import dialog and lets you import a model to replace the current (right-clicked) system. Used for replacing placeholders/dummy systems with full ones.
- **Import new failure model**: imports a new failure model and adds it to the current system.

Models imported this way can be existing ODE models or models from another tool (which will get converted). In this way, it is possible to merge and combine different models from different tools. For example, a high-level ODE model can have its abstract subsystems refined by detailed subsystems imported from HiP-HOPS or safeTbox, along with accompanying analysis results, which can then all be combined with a state machine imported from Dymodia to represent overall system dynamic behaviour.

To support generation of runtime EDDIs, the Editor also allows creation and editing of runtime-specific elements not found in any of the separate tools, e.g. information about Events and Actions. These can be added by right-clicking appropriate model elements, e.g. Failures for Events and States or Causes for Actions.

Examples of how to perform various actions are provided below.

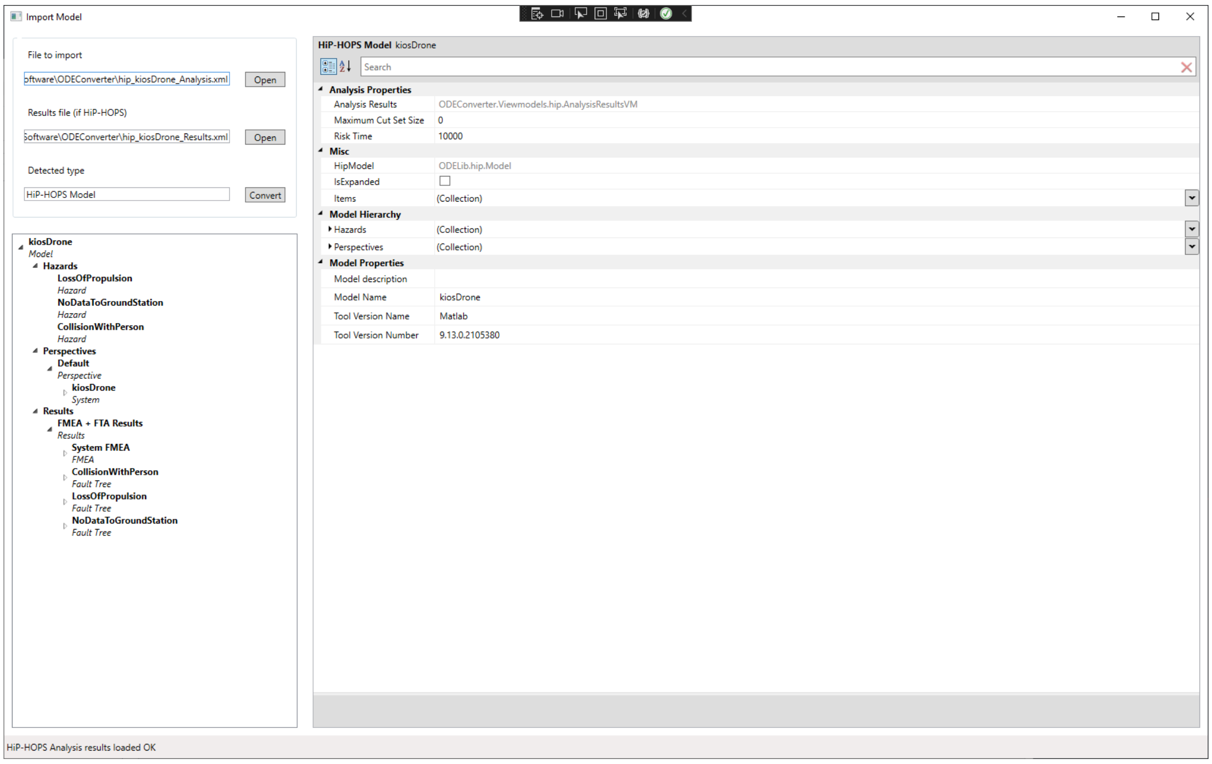
5.4.2.1 Importing and converting a HiP-HOPS model

To import any model for conversion into a new ODE model, click the **Import** option from the **File** menu. This opens the Import dialog. At the top-left of this window are slots for two files; for HiP-HOPS, these are the system architecture file (usually called `XYZ_Analysis.xml`) and the analysis results file (`XYZ_Results.xml`). Note that the `outputtype=RESULTS` option should be used with HiP-HOPS to generate a single output XML file rather than one per fault tree.

For other tools, e.g. safeTbox or Dymodia, only the first slot is required.

Once loaded, the model hierarchy tree and properties pane will be populated with the imported model and the "Detected type" box should identify the source of the model (e.g. HiP-HOPS, Dymodia, etc).

This will load, populating the model hierarchy tree view and the properties pane. The "Detected type" box should say that it is a HiP-HOPS model. You may notice that the Results in the hierarchy view is empty, but if the load was successful there should be hazards and a default perspective with components and failure data, etc.



File to import: oftware\ODEConverter\hip_kiosDrone_Analysis.xml

Results file (if HiP-HOPS): Software\ODEConverter\hip_kiosDrone_Results.xml

Detected type: HiP-HOPS Model

- **kiosDrone** *Model*
  - **Hazards**
    - **LossOfPropulsion** *Hazard*
    - **NoDataToGroundStation** *Hazard*
    - **CollisionWithPerson** *Hazard*
  - **Perspectives**
    - **Default** *Perspective*
      - **kiosDrone** *System*
  - **Results**
    - **FMEA + FTA Results** *Results*
      - **System FMEA** *FMEA*
      - **CollisionWithPerson** *Fault Tree*
      - **LossOfPropulsion** *Fault Tree*
      - **NoDataToGroundStation** *Fault Tree*

HiP-HOPS Model kiosDrone

Analysis Properties	
Analysis Results	ODEConverter.Viewmodels.hip.AnalysisResultsVM
Maximum Cut Set Size	0
Risk Time	10000
Misc	
HipModel	ODELib.hip.Model
IsExpanded	☐
Items	(Collection)
Model Hierarchy	
Hazards	(Collection)
Perspectives	(Collection)
Model Properties	
Model description	
Model Name	kiosDrone
Tool Version Name	Matlab
Tool Version Number	9.13.0.2105380

HiP-HOPS Analysis results loaded OK

Figure 45 - EDDI Editor import dialog

At this stage, you can choose to edit some or all of the properties of the model. Note that editing is restricted to the properties of existing model entities; you cannot change the model structure by adding/removing entities.

Pressing the Convert button next to the detected type will convert the imported model to ODE and return to the main view. At this stage, you can again edit the resulting ODE model if you wish, though again you cannot edit the overall structure, only the properties.

5.4.2.2 Replacing a subsystem

One of the goals of the EDDI Editor is to support tool heterogeneity, i.e., to allow different tools to be used together to produce EDDIs. For example, an initial functional design model may be created in safeTbox, then more detailed implementation information can be imported from HiP-HOPS models representing individual subsystems. This also helps address scalability by ensuring that separate concerns can be addressed in detail using different models (even different tools) but then combined in order to produce an overall EDDI.

Doing this is relatively straightforward. Once an initial ODE model has been opened or converted, you can import and merge other models directly within the hierarchy. Take a simple standby-recovery system as an example:

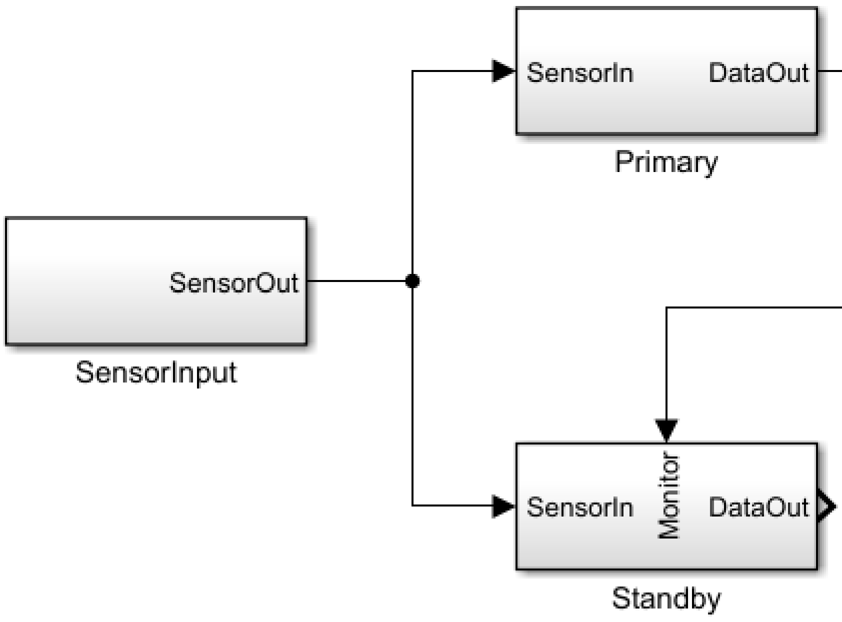


Figure 46 - example standby-recovery model

This is a simple primary-standby model with three components: a sensor for input, a primary actuator/processor, and a standby actuator/processor that takes over if the primary fails. The standby is only a placeholder where the actual implementation should be. This simulates a high-level architecture model where detailed information about subsystem implementation is not available yet, e.g. because the design is still immature or because the subsystem is produced by a third-party contractor.

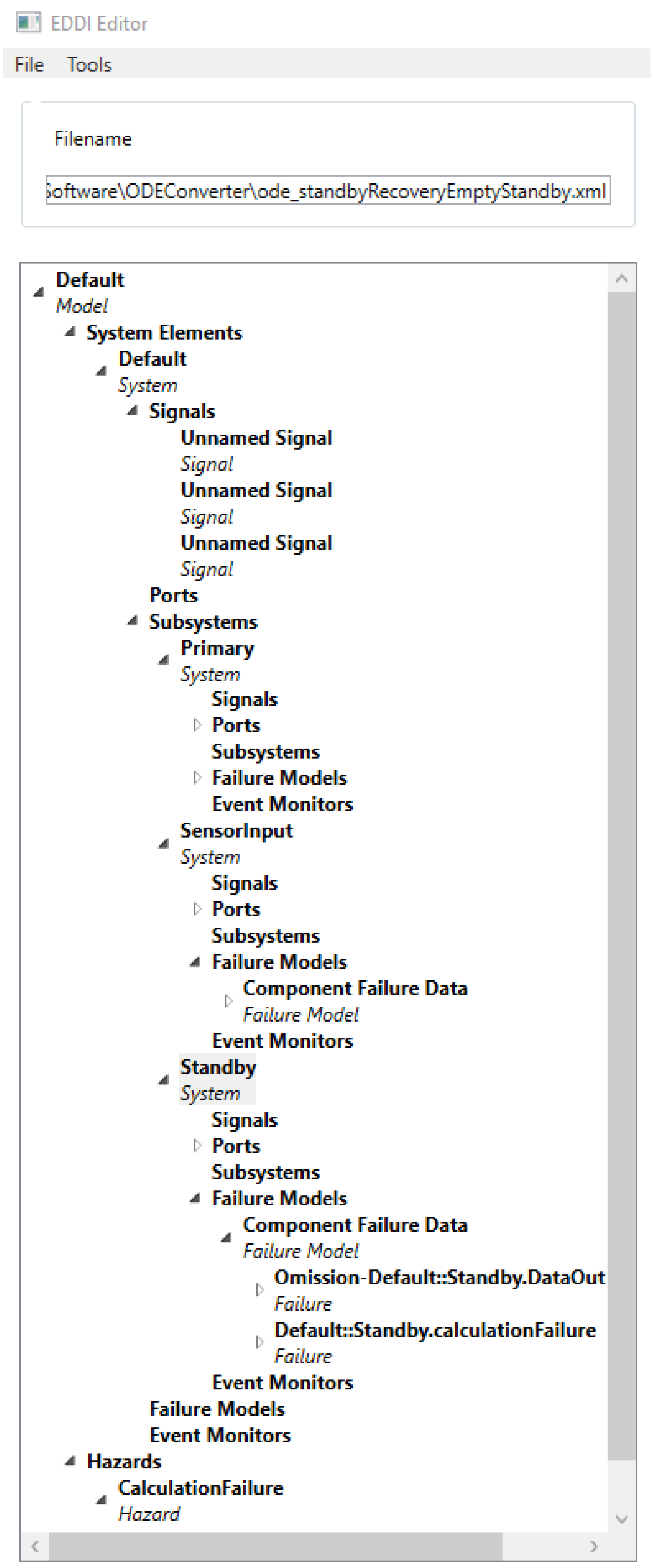
The EDDI Editor then allows you to add or replace subsystems and thereby merge models (potentially from different tools).

To demonstrate, use the hierarchy tree view to navigate down to the Standby system:

- Default (model)

- System Elements
    - Default (system)
        - Subsystems
            - Standby (System)

For a placeholder, although the system element is present, there will be no failure data defined. Right-clicking the placeholder opens up a context menu; selecting **Import model and replace this subsystem** allows us to replace it (via merging) with an imported model for that subsystem.



```
;oftware\ODEConverter\ode_standbyRecoveryEmptyStandby.xml

Default
Model
  System Elements
    Default
    System
      Signals
        Unnamed Signal
        Signal
        Unnamed Signal
        Signal
        Unnamed Signal
        Signal
      Ports
      Subsystems
        Primary
        System
          Signals
          Ports
          Subsystems
          Failure Models
          Event Monitors
        SensorInput
        System
          Signals
          Ports
          Subsystems
          Failure Models
            Component Failure Data
            Failure Model
          Event Monitors
        Standby
        System
          Signals
          Ports
          Subsystems
          Failure Models
            Component Failure Data
            Failure Model
              Omission-Default::Standby.DataOut
              Failure
              Default::Standby.calculationFailure
              Failure
          Event Monitors
      Failure Models
      Event Monitors
  Hazards
    CalculationFailure
    Hazard
```

Figure 47 - Merged subsystem

This should return to the main view, except now the previous placeholder has been replaced by the imported model with associated failure data (e.g. output deviations and failure modes).

5.4.2.3 Importing a state machine

Just as we can import and merge system architecture models, so can we import failure models such as state machines. This allows us to add dynamic behaviour to otherwise static models by e.g. combining a HiP-HOPS model with a Dymodia state machine. By then adding appropriate events and actions, we help pave the way towards the generation of a runtime EDDI.

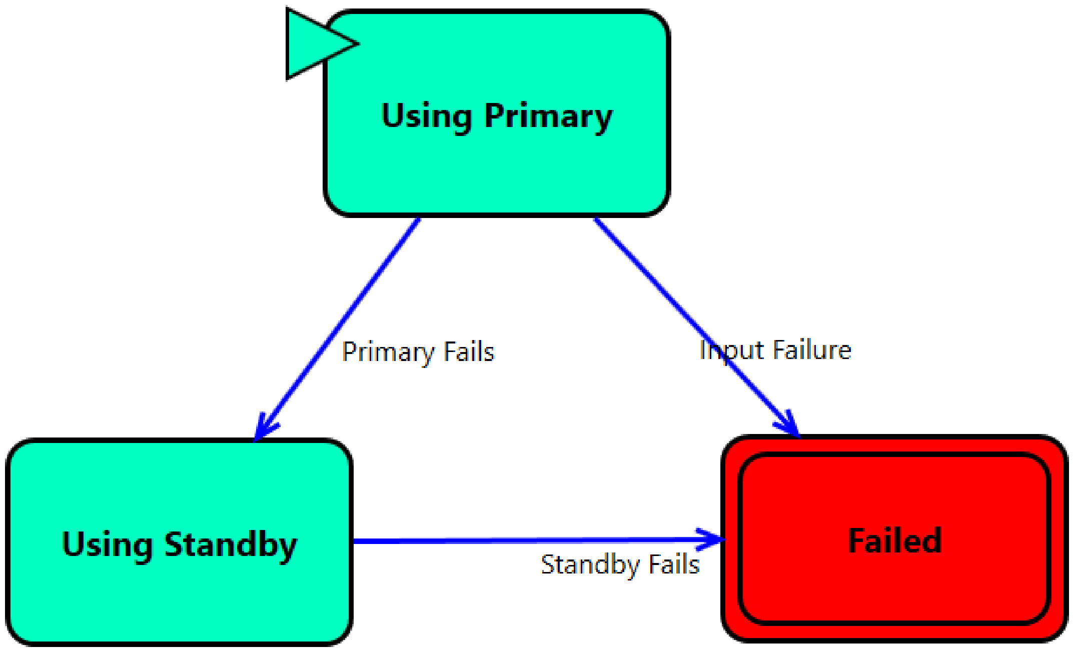
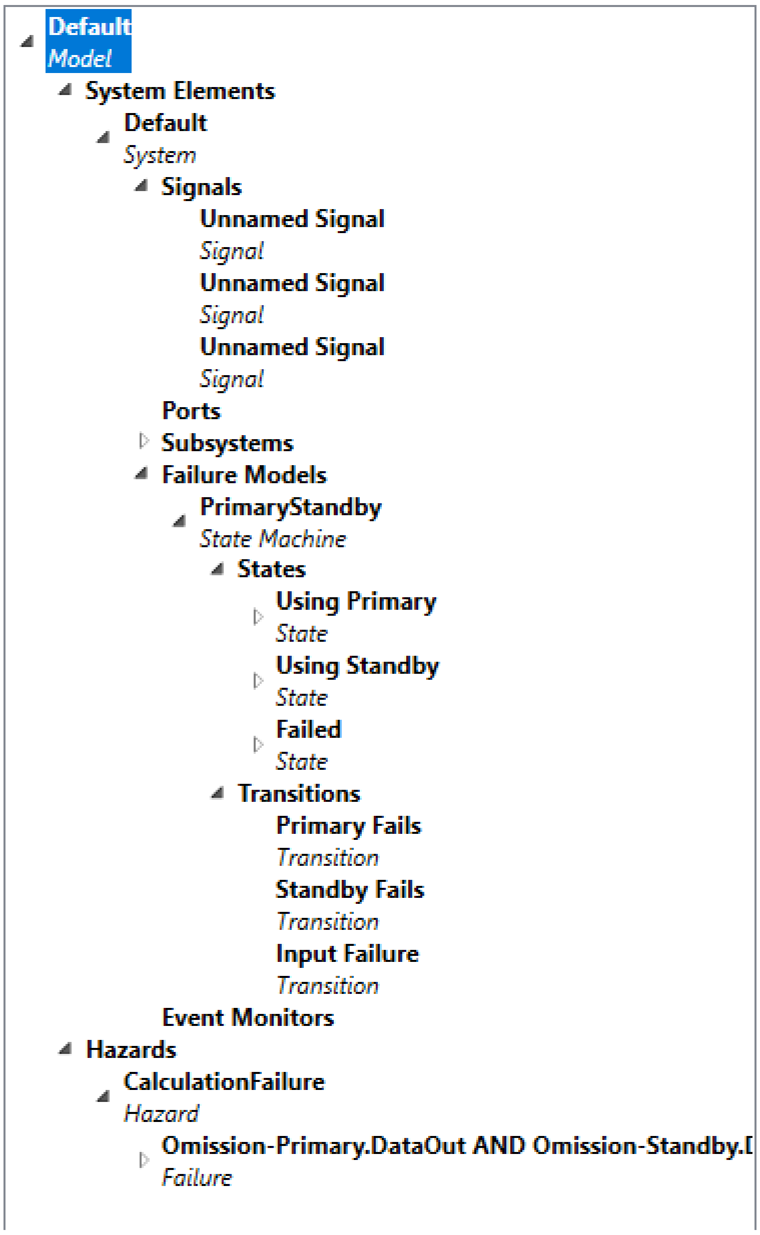


Figure 48 - Dymodia state machine for the standby-recovery system

To do this, select **Import New Failure Model** and choose an appropriate file in the import dialog, e.g. a Dymodia state machine (.uproj file). Pressing **Convert** converts it to an ODE model and then merges it into the main ODE model.



```
◢ Default
  Model
   ◢ System Elements
      ◢ Default
        System
         ◢ Signals
              Unnamed Signal
              Signal
              Unnamed Signal
              Signal
              Unnamed Signal
              Signal
            Ports
         ▷ Subsystems
         ◢ Failure Models
            ◢ PrimaryStandby
              State Machine
               ◢ States
                  ▷ Using Primary
                    State
                  ▷ Using Standby
                    State
                  ▷ Failed
                    State
               ◢ Transitions
                    Primary Fails
                    Transition
                    Standby Fails
                    Transition
                    Input Failure
                    Transition
            Event Monitors
   ◢ Hazards
      ◢ CalculationFailure
        Hazard
         ▷ Omission-Primary.DataOut AND Omission-Standby.
           Failure
```

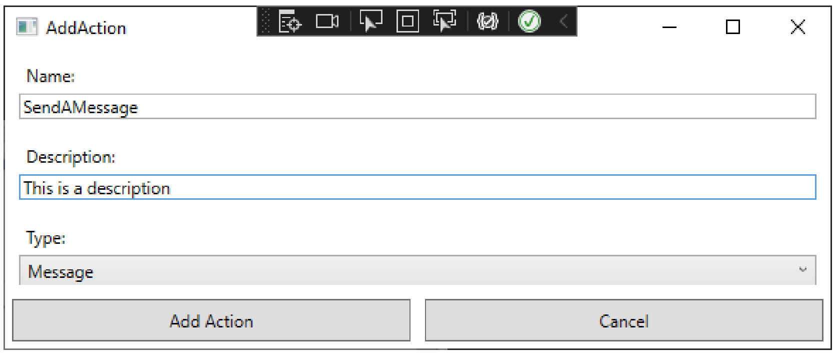
Figure 49 - Imported state machine

5.4.2.4 Adding Events and Actions

As mentioned earlier, to support generation of runtime EDDIs, it is necessary to add information about how to execute the EDDI: namely, Events and Actions. Events are occurrences triggered by something (usually a specific condition that is being monitored), and Actions are pre-determined actions that are executed in response.

These can be added by selecting appropriate system elements and right-clicking. Both States (in state machines) and Causes (in fault trees or attack trees) can have Actions attached; for states, these can be "on entry" (executed when entering a state) or "on exit) (executed when leaving a state), while for Causes, the Actions are executed when the Cause becomes true.

Clicking on either will open an Add Action dialog:



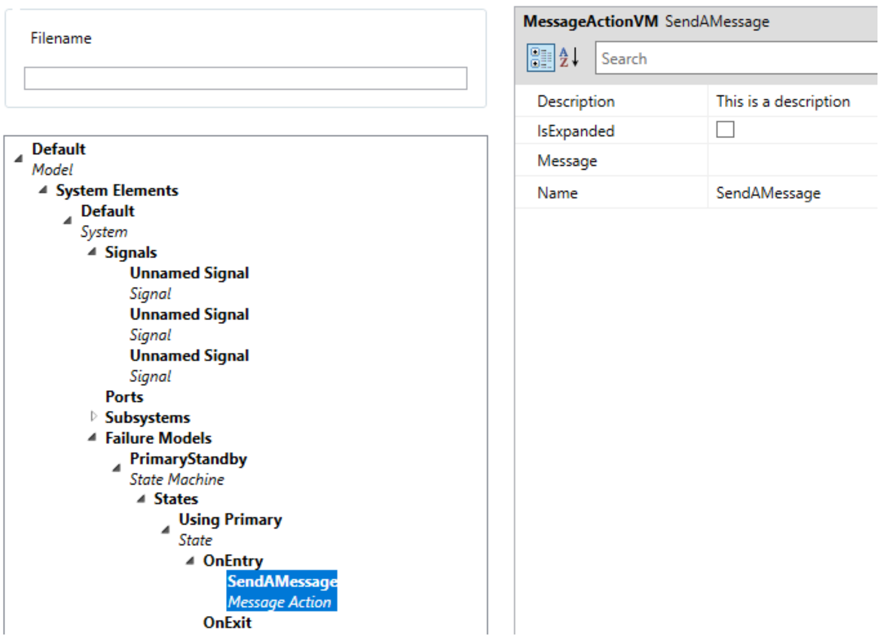
Name: SendAMessage

Description: This is a description

Type: Message

Figure 50 - Add Action dialog

This allows you to set basic information about the Action and to select its type from the drop down menu. Press Add Action and the action will be created and added to the model:



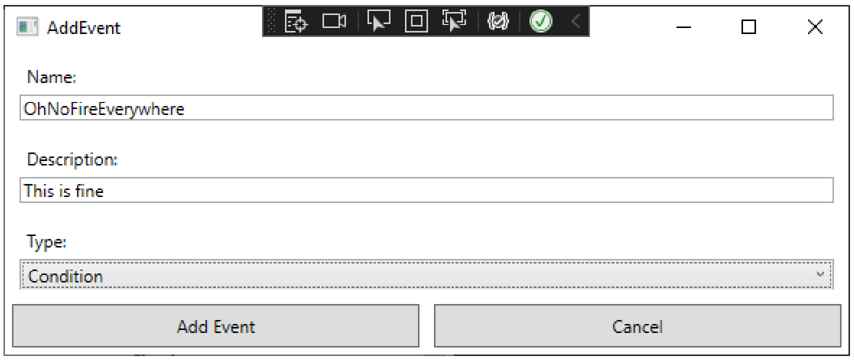
MessageActionVM SendAMessage

Description	This is a description
IsExpanded	☐
Message	
Name	SendAMessage

```
Default
Model
  System Elements
    Default
    System
      Signals
        Unnamed Signal
        Signal
        Unnamed Signal
        Signal
        Unnamed Signal
        Signal
      Ports
      Subsystems
      Failure Models
        PrimaryStandby
        State Machine
          States
            Using Primary
            State
              OnEntry
                SendAMessage
                Message Action
              OnExit
```

Figure 51 - Action added

Events, on the other hand, are primarily defined as part of Failures (i.e., the occurrence of the failure triggers the Event), but are added in a similar manner.



Name: OhNoFireEverywhere

Description: This is fine

Type: Condition

Figure 52 - Add Event dialog

Note that in both cases, type-specific information (e.g. the actual condition for ConditionEvents or the message for MessageActions) can only be defined in the properties grid after the Action/Event has been added.

6. Other Tools: GeNIe Bayesian Network Tool

6.1 What is GeNIe Modeler

GeNIe Modeler[8] is a commercial modelling and analysis tool by BayesFusion[9]. It allows graphical modelling of Bayesian networks, and supports Bayesian inference at design time to get fast feedback during the modelling phase. Further, a plethora of machine learning algorithms are supported for either learning the structure, i.e., the causal connections, of a Bayesian network or for parametrization of the network's conditional probability distributions with the help of data. Diverse analysis tools, e.g. sensitivity analysis, are available in the tool as well to deepen the understanding of a specific Bayesian network and its respective inference process.

In addition to Bayesian network modelling and inference, GeNIe Modeler supports the following machine learning algorithms:

- Algorithms for learning the network's structure:
    - PC (Peter Clark)
    - Bayesian Search
    - Greedy Thick Thinning
    - Tree Augmented Naïve Bayes
    - Augmented Naïve Bayes
    - Naïve Bayes
- Algorithms for learning the network's parameters:
    - EM (Expectation Maximization

Further, the application supports the following methods for evaluation and analysis:

- Evaluation using normal testing
- Evaluation using a k-fold cross validation
- Strength of influence analysis
- Sensitivity analysis

Notice that GeNIe Modeler is not only used for Bayesian networks, but also supports influence diagrams, dynamic Bayesian networks, equation-based models, and hybrid models.

[8] https://www.bayesfusion.com/genie/
[9] https://www.bayesfusion.com/

For a more in-depth perspective on GeNIe Modeler, we refer to the official BayesFusion website[10] and their support page[11].

6.2 HOW IT WORKS

GeNIe Modeler is a commercial software. So, beforehand a license must be acquired and the software must be installed. In addition to their commercial business license, BayesFusion distributes a free academic license as well. The following steps show the process of getting a license, installing, and setting up GeNIe Modeler:

To use GeNIe Modeler, the user must:

1. Install the respective version of GeNIe Modeler as described on the official website[12]. Notice that you must decide between the business[13] and the academic[14] version. So, navigate to https://www.bayesfusion.com/downloads/.

    a. If an academic version is desired, navigate to https://download.bayesfusion.com/files.html?category=Academia and download Genie academic version (compatible with Windows, can be used in MacOS and Linux with emulation software e.g. Wine)

    b. The business version requires purchasing a license by contacting Bayesfusion. 30-day trial versions are available at https://download.bayesfusion.com/files.html?category=Business

2. Execute the installer (in the emulated environment, if applicable) and follow the instructions to install Genie Modeller.

3. Launch Genie Modeller for the first time.

4. Get a license for your GeNIe Modeler version after launching the application for the first time. For that, you just need to follow the dialog box that opened automatically.

    a. In case of a business license, you must have the individual license file provided by BayesFusion on hand.

Now, GeNIe Modeler can be used. In addition to the following tutorial sections, we refer to the official user handbook[15] for GeNIe Modeler for specific questions and more details in general.

6.2.1 Using GeNIe Modeler for Bayesian Network Modelling

1. Launch GeNIe Modeler.

2. Now, GeNIe Modeler starts with an empty network like shown in Figure 53.

[10] https://www.bayesfusion.com/
[11] https://www.bayesfusion.com/resources/
[12] https://www.bayesfusion.com/downloads/
[13] https://download.bayesfusion.com/files.html?category=Business
[14] https://download.bayesfusion.com/files.html?category=Academia
[15] https://support.bayesfusion.com/docs/GeNIe/

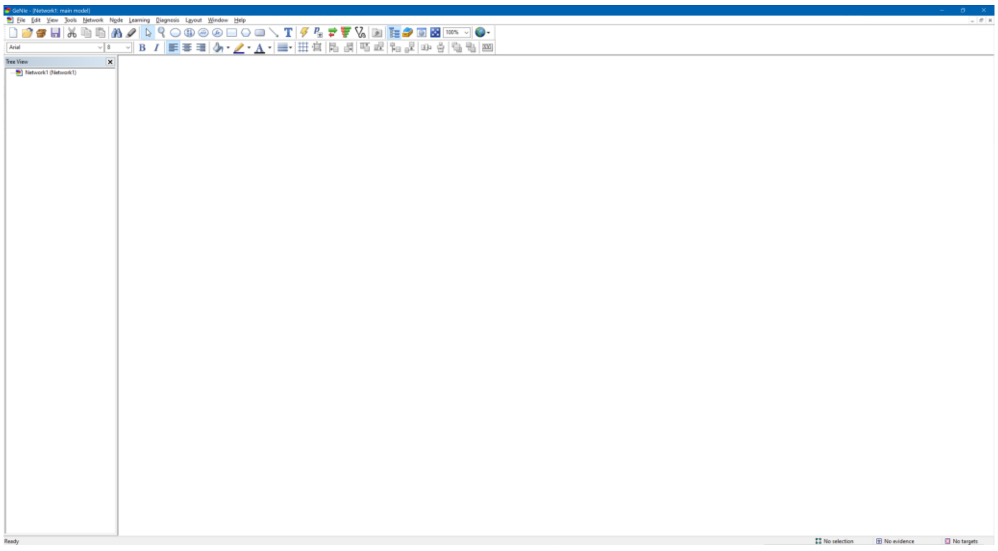


Figure 53 - Initial screen of GeNIe Modeler

3. Now, you can start modelling a Bayesian network by adding nodes via the button in the topbar that symbolizes a network node (as shown in Figure 54). Hint: You have to click on the canvas after selecting the node button to create new nodes.

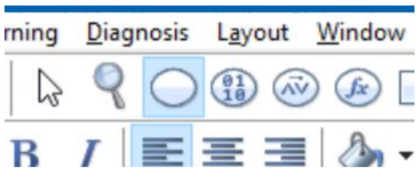


Figure 54 - Button to select for creating new Bayesian network nodes

4. Similar, with the arc button (as shown in Figure 55), you can connect two nodes in the canvas to model a causal relationship.

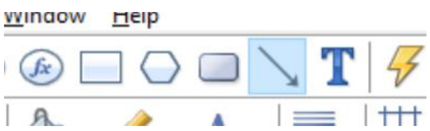


Figure 55 - Button to select for creating new causal relationships between two nodes

5. In the “Node properties” you can specify each node. You can open the “Node properties” dialog window by double clicking on a node on the canvas.
    a. In the “Definition” tab, you can define the concrete states of that node. You can rename them by double clicking on the node’s states (left part in Figure 57). Via the buttons in the bar above (as shown in Figure 56), you can add or remove states.
    b. In the same “Definition” tab, you can define the conditional probability distribution in form of a table for that specific node. This is the concrete parameterization for this specific node, thus, it decides the outcomes when the Bayesian network is inferred given evidence. This table gives the probabilities for each state of the node given a state permutation over all parents of this node (as shown in Figure 57).

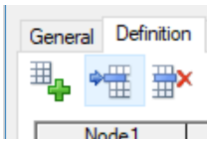
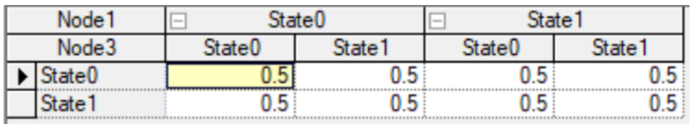


Figure 56 - Buttons for adapting the number of states of a node



Node1	State0		State1	
Node3	State0	State1	State0	State1
State0	0.5	0.5	0.5	0.5
State1	0.5	0.5	0.5	0.5

Figure 57 - Conditional probability table for a node that has two parents in an example Bayesian network

6. Finally, for saving the modelled Bayesian network, you can use the standard save functionality. This option can be reached via the "File" menu as seen in Figure 58 - Save option via the "File" menu in GeNIe Modeler. Accordingly to Figure 59, assure that you save the file as a .xdsl format.

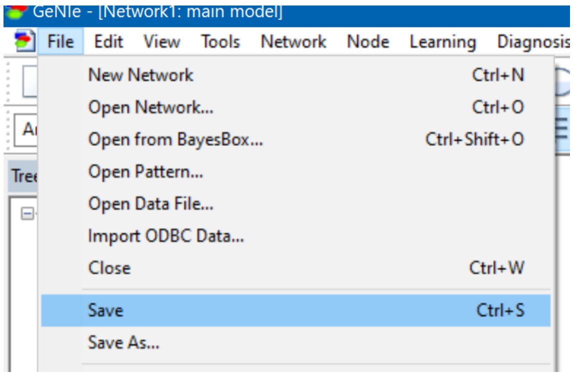


Figure 58 - Save option via the "File" menu in GeNIe Modeler

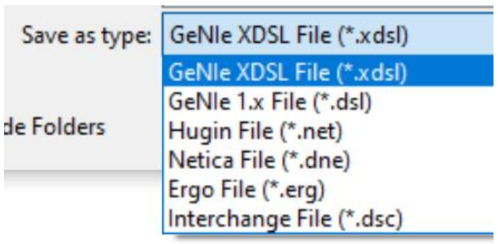


Figure 59 - File format options provided by GeNIe Modeler

6.2.2 Using GeNIe Modeler for Bayesian Network Inference

1. To test your Bayesian network, you can run inferences in GeNIe Modeler. The lightning button in the top bar updates all nodes in the network at once (as shown in Figure 60).

   a. To easier observe the outcomes of the nodes, you can change the view option on the canvas to include the states with their probability distribution. For this, mark all nodes and right click on them. Then, select the "View As" – "Bar Chart" option.

b. To set specific evidence for one node, you can simply double click on the respective states when the "Bar Chart" view option is active.

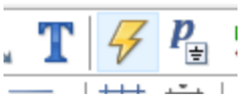


Figure 60 – Update button to run an inference over the network

2. After setting an evidence but before updating the network (= inference), the Bayesian network still has nodes without probability distributions like in the example (Figure 61).

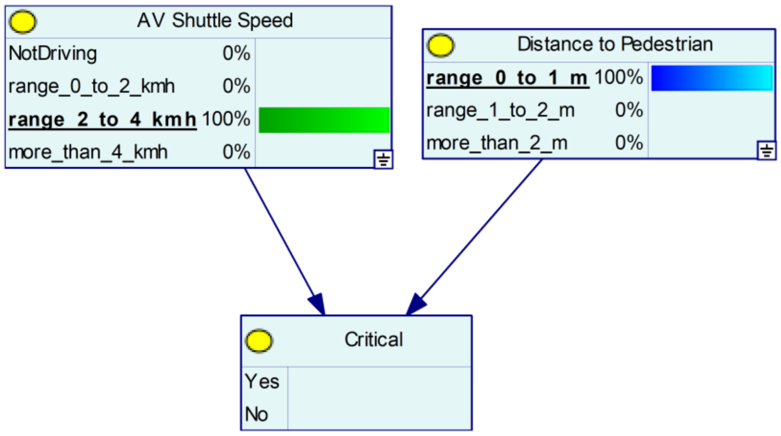


Figure 61 - Simplified example Bayesian network which is computing the criticality of a situation (node "Critical") for a autonomous shuttle given the shuttle's speed (node "AV Shuttle Speed") and the distance to the closest pedestrian (node "Distance to Pedestrian"). Two evidence (bold and underlined states) are manually set.

3. Now, after the network is updated, there is a probability distribution of the node states assigned to, respectively, computed for, all the nodes in the network. So, the node of interest can be observed that way, like the "Critical" node in the example (Figure 62).

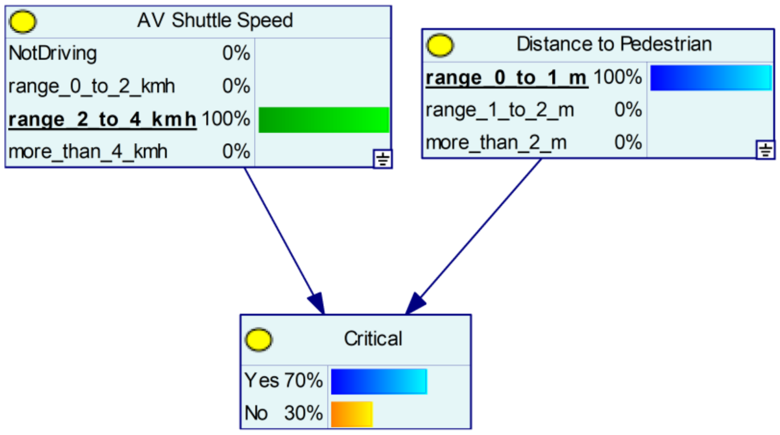


Figure 62 - Simplified example Bayesian network from Figure 61 after the inference

6.2.3 Using GeNIe Modeler for Machine Learning Tasks

1. Open the dataset that shall be used for the evaluation in GeNIe Modeler. This can be achieved via drag-and-drop or via the “File” menu using the “Open Data File…” option. Typical dataset formats like .csv, .txt, or GeNIe’s proprietary .gdat format are supported.

    a. Notice that the dataset must have a data column for each evidence and output node with the data values being the node’s states each.

2. Open the ML dialog box.

    a. In case of parameter learning:

        i. Simply open the “Learn Parameters…” option in the “Learning” menu like shown in Figure 63.

        ii. Confirm the “March Network and Data” window. There you can assure that all the data entries are linked to the correct network nodes and states. In case that the dataset columns and values align with the network’s nodes and states identifiers, GeNIe Modeler automatically matches everything correctly. Conflicts are highlighted in this window as well and you can manually link entries by drag-and-drop.

    b. In case of structure learning:

        i. Assure that you have the dataset that you want to use actively selected as the open window in GeNIe Modeler. For that you have to select the dataset in the “Window” menu like shown in Figure 64.

        ii. Now, you have different menu options available. Select the option “Learn New Network…” in the “Data” menu like shown in Figure 65.

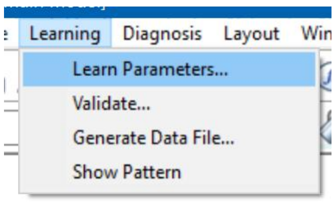


Figure 63 - Option to open the parameter learning dialog window

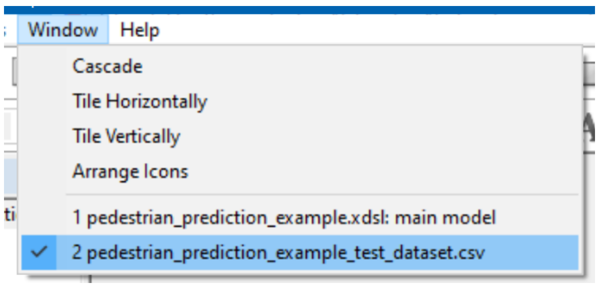

Window
Help
Cascade
Tile Horizontally
Tile Vertically
Arrange Icons
1 pedestrian_prediction_example.xdsl: main model
2 pedestrian_prediction_example_test_dataset.csv

Figure 64 - Option to change the active file in GeNIe Modeler

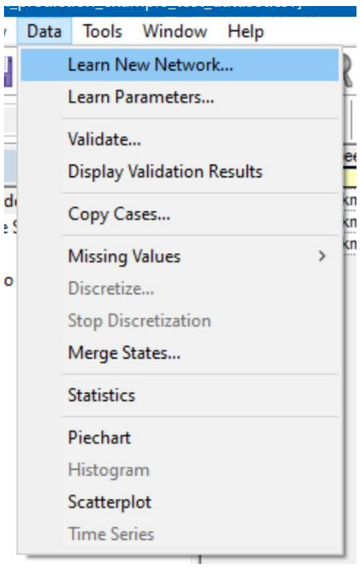


Figure 65 - Option to open the structure learning dialog window

3. Select the ML algorithm of choice, set up the respective parameters and run the ML task. For the parameter learning with the EM algorithm you can skip this step because EM is the only available option.

   a. Details on the supported datafiles, ML algorithms and their parameters etc. can be found in the official documentation[16] in the section "Using GeNIe" → "Learning".

4. After the ML task finished, the created, respectively adapted, Bayesian network will be automatically shown in GeNIe Modeler. Be aware that you must save the newly created network manually again in case of structure learning.

6.2.4 Using GeNIe Modeler for Bayesian Network Validation

Using GeNIe Modeler for Bayesian Network Evaluation

1. Open the dataset that shall be used for the evaluation in GeNIe Modeler. This can be achieved via drag-and-drop or via the "File" menu using the "Open Data File…" option. Typical dataset formats like .csv, .txt, or GeNIe's proprietary .gdat format are supported.

   a. Notice that the dataset must have a data column for each evidence and output node with the data values being the node's states each. So, an example dataset for evaluating the previous example Bayesian network (Figure 61) may look like Figure 66.

[16] https://support.bayesfusion.com/docs/GeNIe/

```
AV_Shuttle_Speed, Distance_to_Pedestrian, Critical
NotDriving, range_0_to_1_m, No
range_0_to_2_kmh, range_0_to_1_m, No
range_0_to_2_kmh, range_1_to_2_m, No
range_2_to_4_kmh, range_1_to_2_m, Yes
```

Figure 66 - Example dataset (.csv file) to evaluate the example Bayesian network from Figure 61

2. Now, the validation window must be opened like shown in Figure 67.

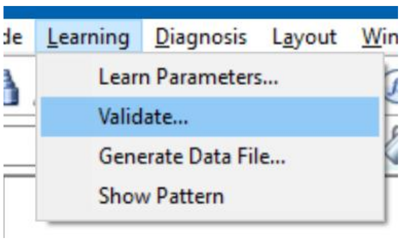


Figure 67 - Option to open the validation window in GeNIe Modeler

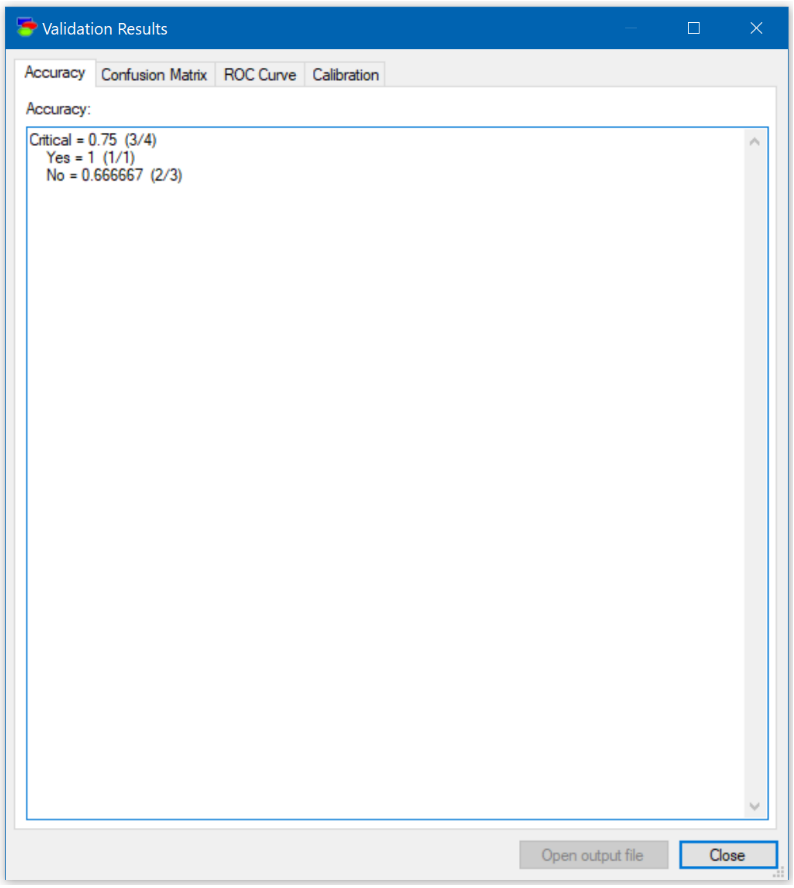
3. Confirm the “March Network and Data” window. There you can assure that all the data entries are linked to the correct network nodes and states. In case that the dataset columns and values align with the network’s nodes and states identifiers, GeNIe Modeler automatically matches everything correctly. Conflicts are highlighted in this window as well and you can manually link entries by drag-and-drop.

4. Select the “Validation method” that you want to perform and set up the respective parameters. It is important that you chose at least one node of interest (“Class nodes”) for which you want to see the evaluation results. Finally, run the evaluation.

   a. Details on the supported evaluation types, their parameter, and the evaluation result can be found in the official documentation[17] in the section “Using GeNIe” → “Learning” → “Validation”.

5. After the evaluation finished, a pop-up window with the evaluation results is shown (see Figure 68). This window includes, for instance, the accuracy and the confusion matrix for each selected class node. Details on the different types of results can be found in the official documentation[18] under “Using GeNIe” → “Learning” → “Validation”.

[17] https://support.bayesfusion.com/docs/GeNIe/
[18] https://support.bayesfusion.com/docs/GeNIe/



Accuracy:

```
Critical = 0.75  (3/4)
   Yes = 1  (1/1)
   No = 0.666667  (2/3)
```

Figure 68 - Validation results for the node "Critical" from the example Bayesian network given in Figure 61 validated with the example dataset given in Figure 66

Using GeNIe Modeler for Bayesian Network Analysis

GeNIe Modeler provides tools for performing "Strength of Influence" and "Sensitivity" analyses. For details on the advantages of those analyses, the required steps to perform such analyses, as well as, information about the expected results for each type of analysis, we refer to the official documentation which goes into great detail here.

The documentation for the "Strength of Influence" analysis can be found in the official documentation[19] under "Using GeNIe" → "Bayesian networks" → "Strength of influences".

The documentation for the "Sensitivity" analysis can be found in the official documentation[20] under "Using GeNIe" → "Bayesian networks" → "Sensitivity analysis in Bayesian networks".

6.3 SESAME-SPECIFIC EXTENSIONS AND FUNCTIONALITY

To translate the Bayesian networks modelled with GeNIe Modeler to EDDIs, the tool adapter introduced in Section 5.2 is used. The tool adapter allows via remote procedure calls using Apache Thrift to send the content of an EDDI as argument and then creates and stores the actual EDDI file for the user.

[19] https://support.bayesfusion.com/docs/GeNIe/
[20] https://support.bayesfusion.com/docs/GeNIe/

The connection and integration for Bayesian networks is implemented in a Python script that connects to the Apache Thrift[21] server that the tool adapter is running. The Python script takes an .xdsl file as an input argument and triggers the tool adapter after extracting all relevant information from the .xdsl Bayesian network model. For the extraction of the Bayesian network itself, the SMILE Engine[22] from BayesFusion is used. This library is used as backend for GeNIe Modeler as well and supports the .xdsl format out of the box. The engine is written in C++ but BayesFusion provides official wrapper for a plethora of programming languages as well, i.e., Python, Java, R, and .NET. A connection to MATLAB is established too. For the SMILE Engine to work you need a license. There are free academic licenses and commercial business licenses.

So, for running the Python script three things are mandatory beforehand:

1. The tool adapter must be set up properly and the respective Apache Thrift server must be running.

2. The transformation Python script must be set up so that the user's SMILE Engine license is used.

3. The Python environment must be set up properly for running the file format transformation. This includes installing several Python packages, e.g., Apache Thrift and PySMILE (the Python wrapper for the SMILE engine).

In the following, the three points are covered. Afterwards, it is shown how to run the actual Python script to generate the EDDI Bayesian network model out of the .xdsl Bayesian network model.

6.3.1 Setting up & Running the Tool Adapter

Information on this topic can be found in Section 5.2 and 5.3.

6.3.2 Setting up the Transformation Python Script with the User's SMILE License

First, a license must be granted by BayesFusion to the user. For this the user must go to the download page[23] and select between the free academic page[24] or the commercial business page[25]. On their the user shall follow the steps to issue a license for the SMILE engine. After receiving and downloading the SMILE Engine license, the user must unzip the archive and locate the "pysmile_license.py" file.

After that, the "pysmile_license.py" file must be copied to the code base so that the transformation Python script is able to validate the installed PySMILE version when running the script. The user must copy the file into the following directory: "./xdsl_to_ddi/license/".

[21] https://thrift.apache.org/
[22] https://www.bayesfusion.com/smile/
[23] https://www.bayesfusion.com/downloads/
[24] https://download.bayesfusion.com/files.html?category=Academia
[25] https://download.bayesfusion.com/files.html?category=Business

6.3.3 Setting up the Python environment to run the Transformation Python Script

The user must install Apache Thrift and PySMILE in the Python environment that shall be used for running the Python script. To achieve this, the following two commands are sufficient:

1. Apache Thrift Python installation: `python3 -m pip install thrift==0.11.0`

2. PySMILE Python installation (the required command depends on the type of license that was issued):

   a. In case of a free academic license: `python3 -m pip install --index-url https://support.bayesfusion.com/pysmile-A/ pysmile`

   b. In case of a commercial business license: `python3 -m pip install --index-url https://support.bayesfusion.com/pysmile-B/ pysmile`

6.3.4 Run the Transformation Python Script to get an EDDI model

You can run the transformation Python script with the following command: `python3 xdsl_to_ddi.py –bayesianNetwork <BayesianNetwork.xdsl>`

With “BayesianNetwork.xdsl” being the name of the concrete input Bayesian network .xdsl file that shall be transformed into an EDDI file.

The generated output EDDI file can be found in the “./xdsl_to_ddi/out/” directory.

7. USE CASE APPLICATIONS

Design-time EDDIs have been used as part of three use cases: the Locomotec disinfection robot use case, the KIOS/CCD power station inspection use case, and the AVL battery production use case. This section describes a HIP-HOPS example based on the KIOS/CCD use case.

7.1 KIOS/CYPRUS CIVIL DEFENCE DRONE INSPECTION USE CASE

In this section, we will discuss the application of some of the design-time safety analysis tools to the KIOS/Cyprus Civil Defence use case. It is intended to follow on from the high-level discussion of the use case in **D4.5 Safety Analysis Concept & Methodology** and lead on to the runtime aspects of the EDDI application in **D7.3 Runtime Safety & Security EDDI Concept for MAS**. The use case itself, along with preliminary results from the initial evaluation phase of the project, is discussed further in **D8.7 Power Station Interim Use Case Evaluation**.

In July 2011, an explosion at a nearby naval base caused heavy damage to the Vasilikos Power Station, the biggest power plant in Cyprus. To ensure the safety of first responders, an exclusion zone was set up around the power station to prevent further injury. Instead, Cyprus Civil Defence (CCD) made use of drones to inspect the power station for damage. Realising the potential of such drones for emergency response, CCD established a collaboration with KIOS to engage in relevant research projects with the goal of developing drone-based inspection systems for emergency search & rescue and damage inspection purposes.

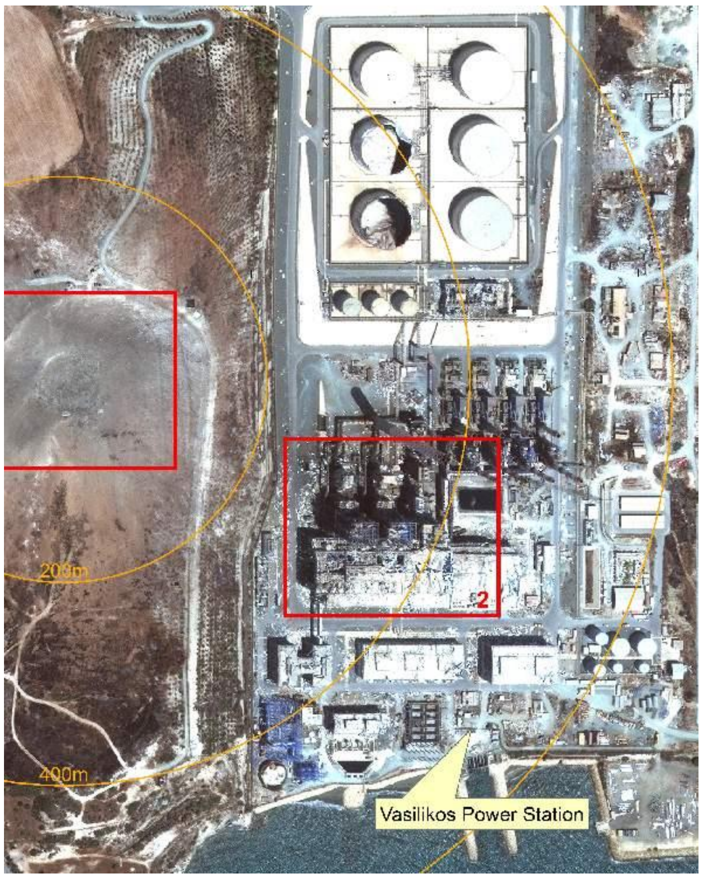


Figure 69 - Vasilikos Power Station incident

The goal of the overall MRS is therefore to:

- Gather information about the disaster (damage, safe locations, casualties & trapped survivors, possible threats to human safety etc);
- Operate under a control centre in a safe location outside the zone to coordinate operations and keep open communications with involved parties;
- Provide aerial visual assistance and assessment.

The system itself therefore consists of a central base station at the safe control centre and one or more UAVs — quadcopter drones in this case — to perform information gathering tasks and aerial support tasks.

7.1.1 System Architecture

At this stage in the design process, it is assumed that more concrete information is available about the system and its functionality. Exact details about the implementation of the drone platform are omitted for confidentiality purposes, but an overview of the architecture is provided below:

- The **Propulsion Unit** encompasses the four rotors of the drone and accompanying motors.
- The **Flight Controller Unit (FCU)** controls the Propulsion Unit (i.e., rotors) based on the inputs of different sensors. It serves as the main control unit of each drone.
- The **Positioning Unit** provides data about the drone's position to the Flight Controller Unit. It includes an Inertial Measurement Unit (IMU) that provides information about acceleration and turning rate etc, as well as a compass and a barometer to measure altitude.
- The **Communications Unit** serves as the means to remotely control the drone by sending instructions to the FCU. It also serves to relay information obtained by the drone back to the ground control station and the user(s). The Communications Unit has been established to be high priority in the earlier HARA (see **D4.5**) and so has multiple channels of communication: Wifi, 3G cellular, and standard radio.
- The **Environmental Detection Unit** (EDU) consists of the drone's sensors for detecting people — the RGB camera and the thermal camera. The LIDAR laser rangefinder is also included here, though it is intended for obstacle avoidance and navigation support.
- The **Gimbal** acts as a stabiliser for the camera(s), ensuring that the orientation remains constant as the drone moves around. As such, it needs information from the FCU to compensate for any movement.

In addition to one or more drones, there is also the Ground Control Station (GCS). This helps to automate the process of logging, managing, and monitoring UAV operation. It also governs task allocation, ensures maximal area coverage, and facilitates communication between the robotic agents of the MRS. For the purposes of the analysis, we are more interested in the drones than the GCS, but given the interaction we assume the GCS has two high-level components:

- A **Communications Unit** to allow communication with the drones.
- A **Drone Manager** which has overall responsibility for controlling the drones. It monitors which drones are connected to the MRS, what equipment and capabilities they possess, their current status (battery level etc), and can issue commands to them. Commands are generated by a task manager subsystem that enables the drones to work collaboratively to cover the search area.

Using this information, we can construct a hierarchical system architecture model of the drone, in this case using HiP-HOPS and Matlab Simulink.

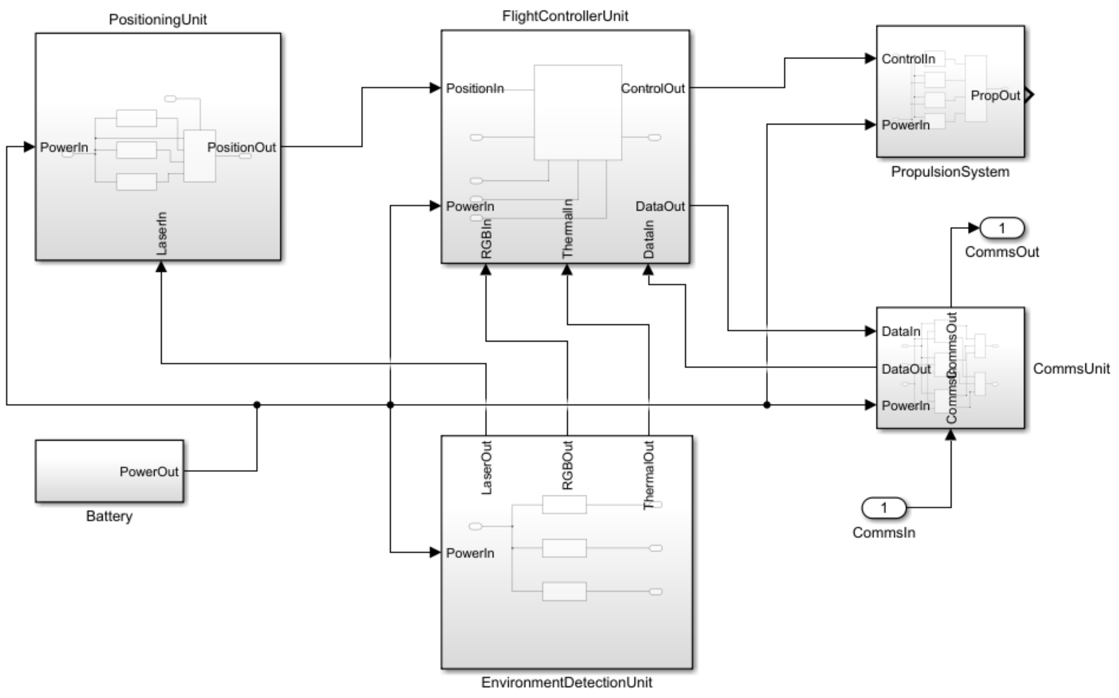


Figure 70 - KIOS drone system model – top level

Here we can see the main subsystems of the drone itself — the Positioning Unit, FCU, EDU (cameras), Propulsion system, Communications unit, and the battery.

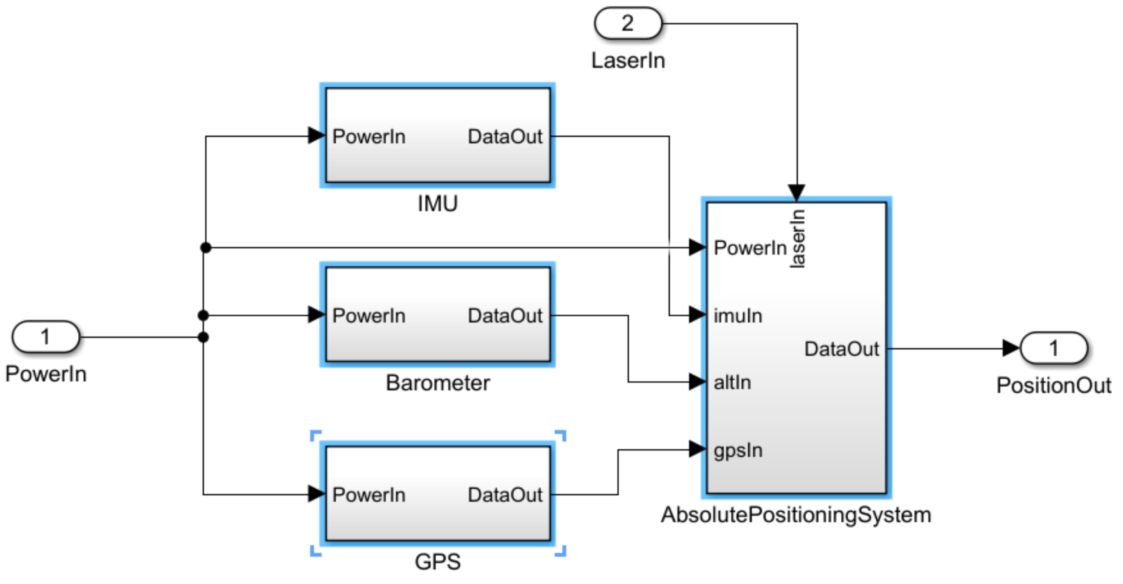


Figure 71 - Positioning Unit subsystem

Figure 71 shows the Positioning Unit subsystem, with IMU, Barometer, and GPS. The component on the right is responsible for taking readings from these instruments and deriving an actual position from them.

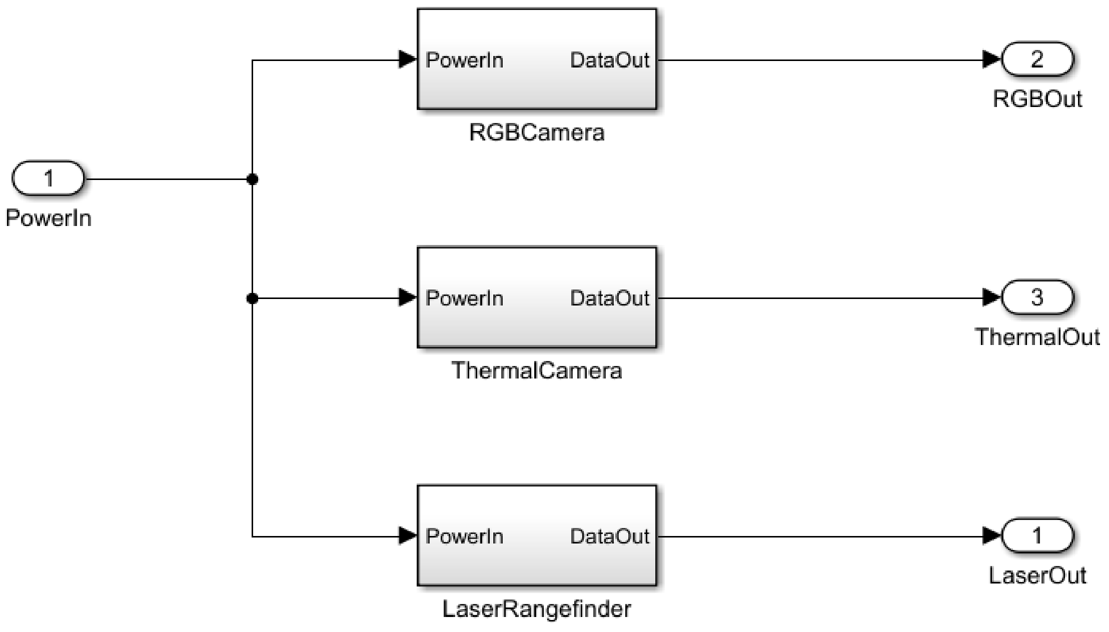


Figure 72 - EDU subsystem

This figure depicts the EDU — visual camera, thermal camera, and LIDAR.

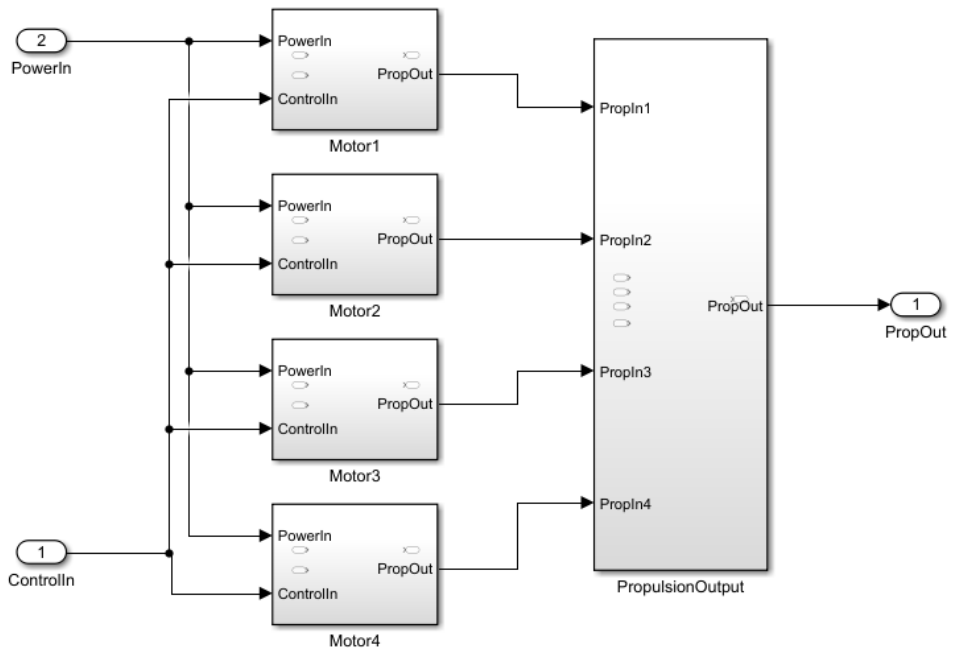


Figure 73 - Rotor propulsion subsystem

Above is the propulsion system with the four motors. The component on the right simply combines the outputs to allow us to define appropriate failure logic.

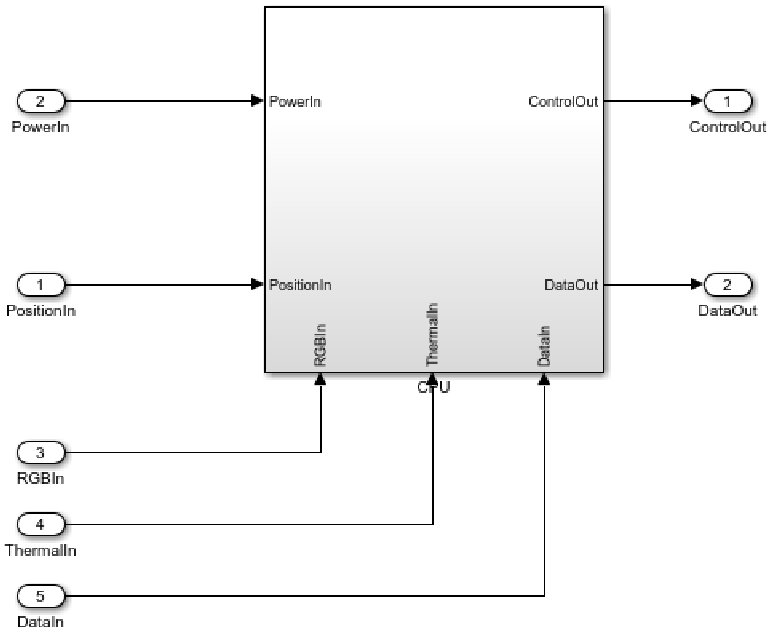


Figure 74 - FCU subsystem

Figure 74 shows the Flight Controller Unit, which contains the main CPU. It takes input from other parts of the system and is responsible for navigation and controlling the motors powering the drone's rotors. Since it acts as the drone's central computer, it also takes and processes input from the cameras to do onboard person recognition.

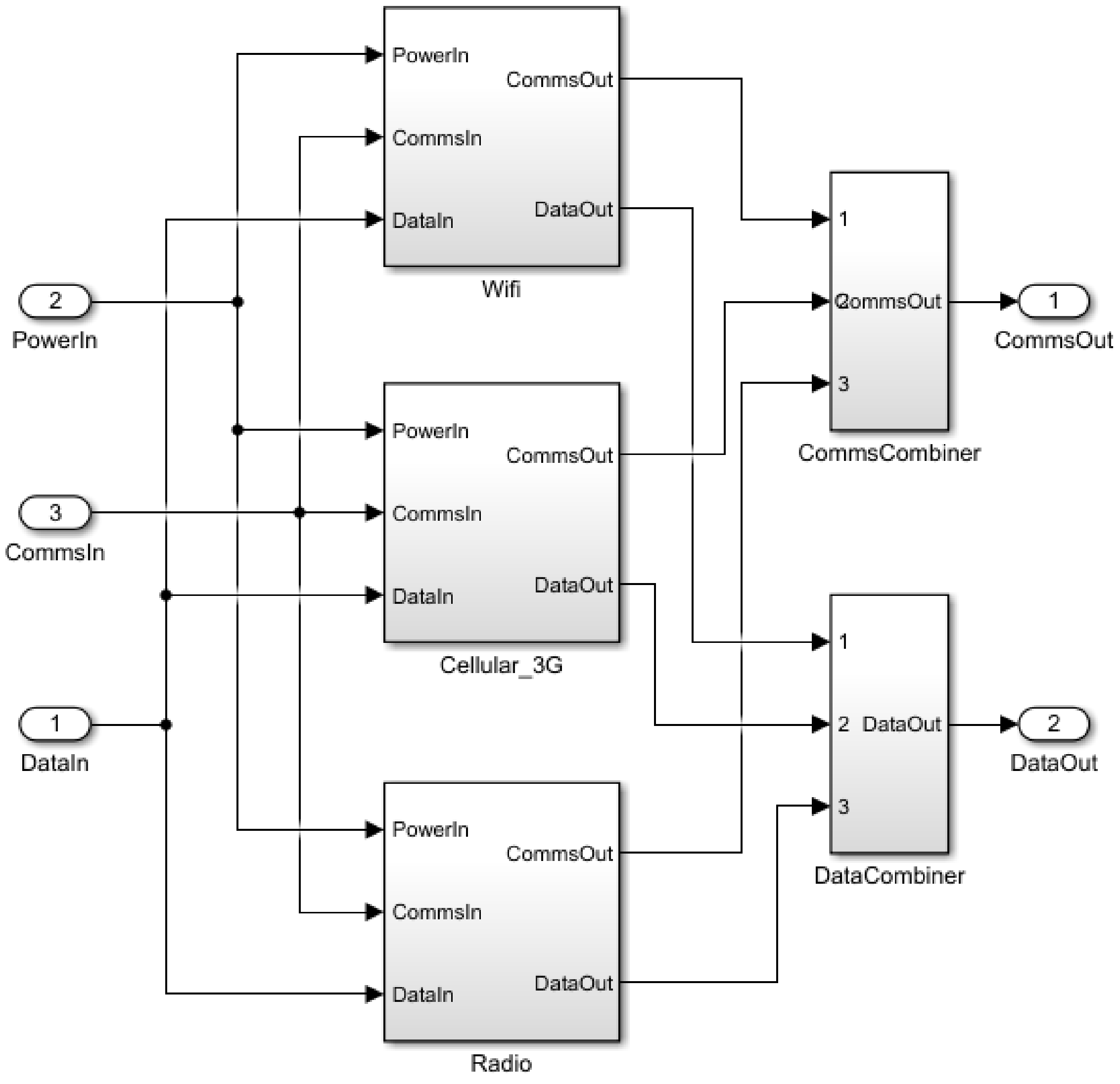


Figure 75 - Comms subsystem

Finally, the communications unit consists of three possible comms channels: wifi, cellular, and radio. These operate independently, allowing fallback from one to the other if one channel is unavailable for whatever reason. The combiner blocks on the right simplify the top-level model by combining all communication propagation into a single line for each direction.

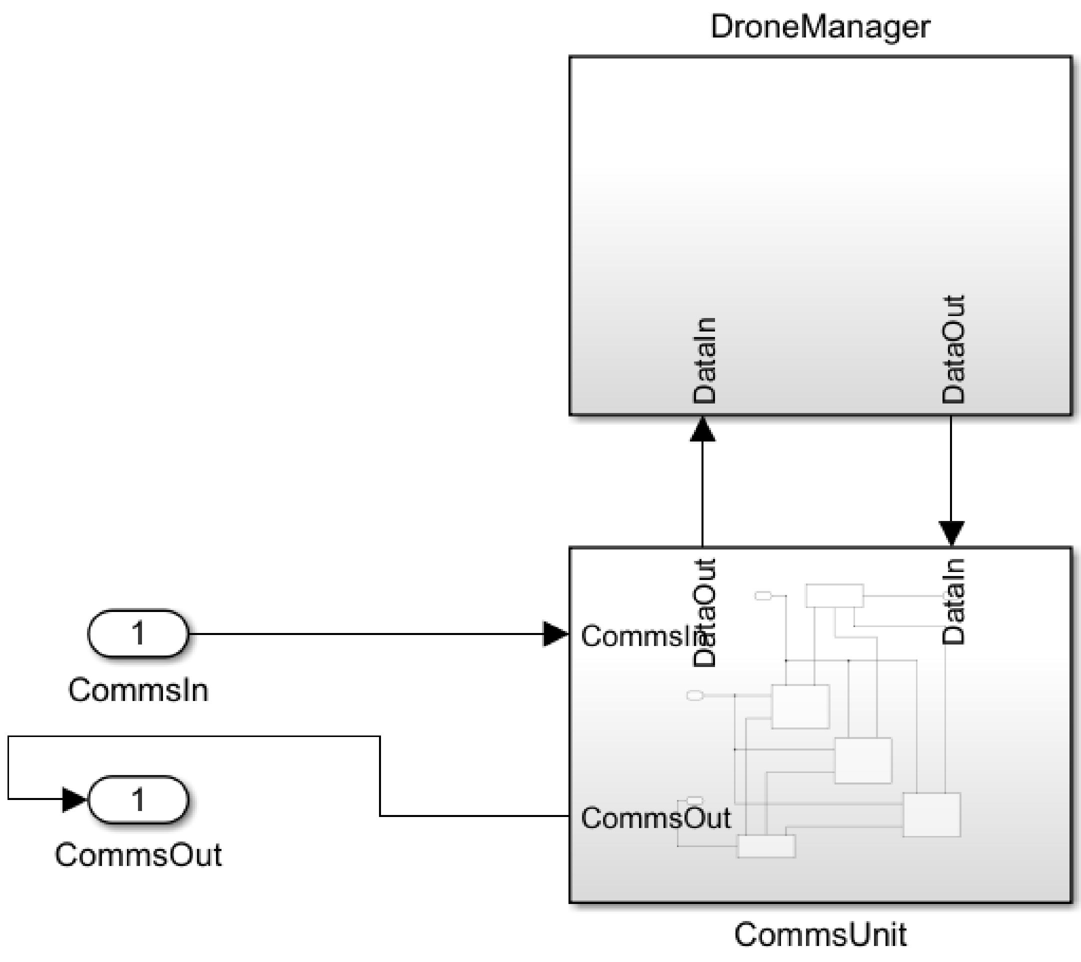


Figure 76 - GCS subsystems

This shows the two subsystems of the GCS: the comms unit and the drone manager. We assume power supply is not a point of failure here (e.g. because power is provided externally by a generator or mains supply).

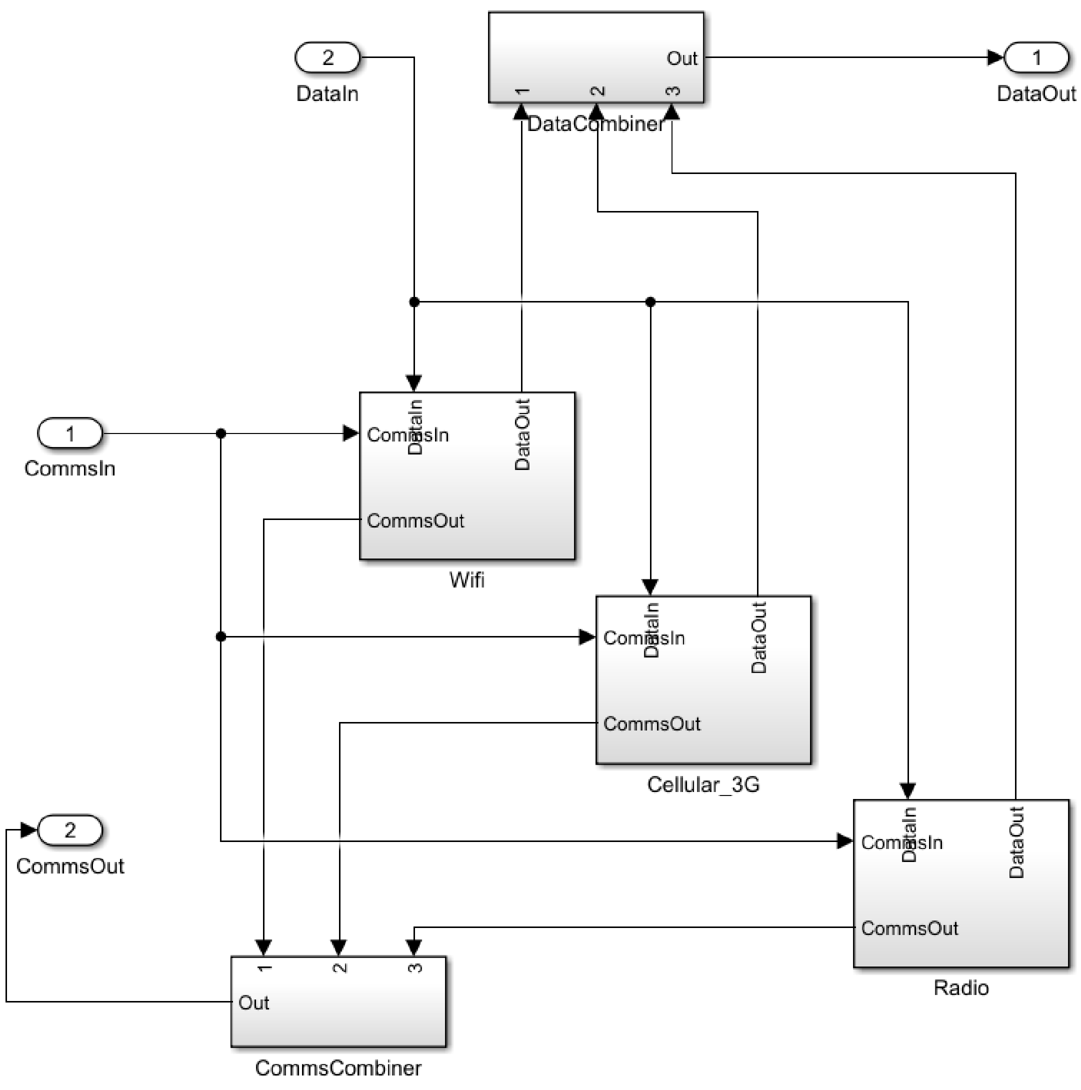


Figure 77 - GCS comms unit

Inside the GCS comms unit are the same three communications channels as the drone.

7.1.2 Annotating with failure data

With the model created, we can begin to add component-level failure data to define how each component can fail and how it reacts to failures elsewhere. This is all supported by the updated HiP-HOPS interface to Simulink, which allows definition of basic events, common causes, and output deviations etc.

Exact failure data for each basic event (i.e., component failure mode) is not available, but we can derive maximum failure rates based on the SILs assigned by the safety requirements according to the guidance provided by standards like IEC 61508:

- SIL 1 = 1e-5 failures/hr – 1e-6 failures/hr
- SIL 2 = 1e-6 failures/hr – 1e-7 failures/hr
- SIL 3 = 1e-7 failures/hr – 1e-8 failures/hr

- SIL 4 = 1e-8 failures/hr – 1e-9 failures/hr

Most components form part of the critical path, i.e., their failure directly and singly impacts one or more of the safety requirements. Only places where there is redundancy — the communications (three channels), the positioning sensors, and the cameras — can lower SILs be accepted.

As described in **D4.5 Safety Analysis Concept & Methodology**, SIL decomposition can be applied where redundancy exists and components are independent. Usually this is according to a simple arithmetic, e.g. SIL 1 + SIL 2 = SIL 3. For the purposes of the example, we have decomposed the SILs and assigned failure rates to each component as follows:

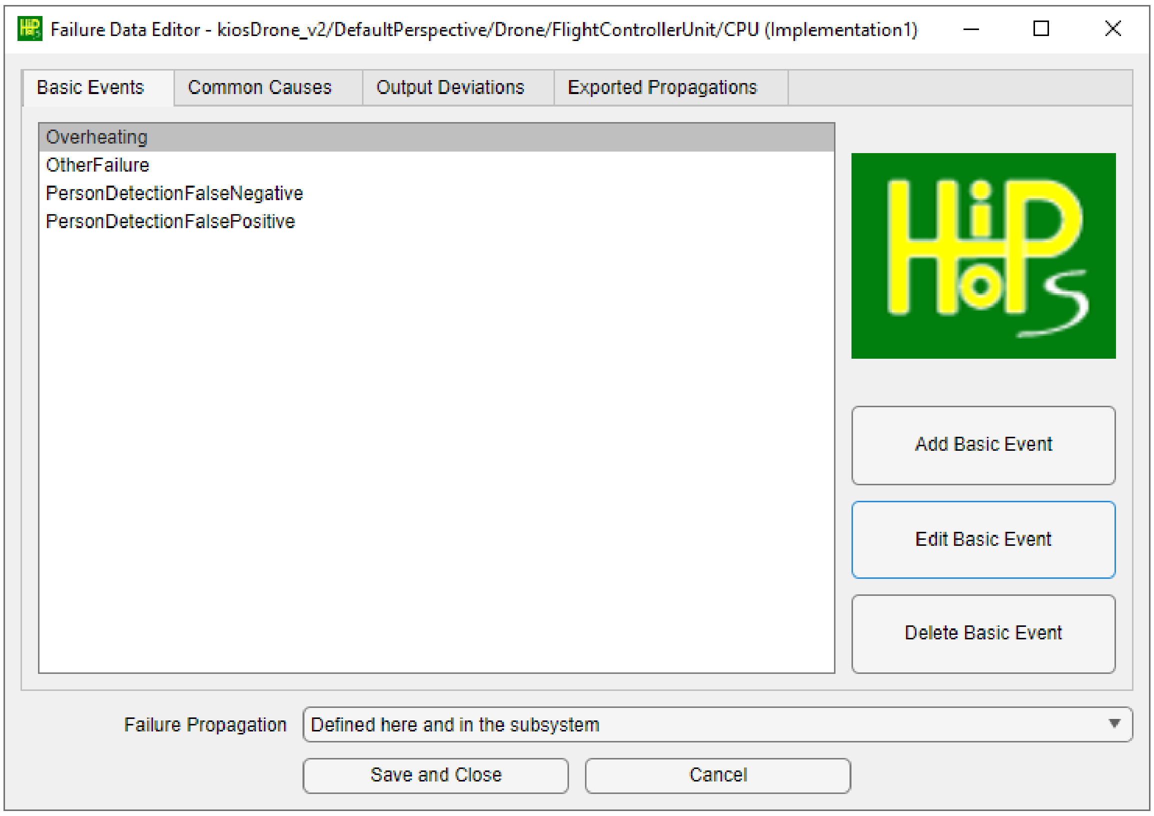
Subsystem	Component	SIL	Failure rate (/hr)
GCS	Drone Manager	4	1e-8
GCS	Comms	4 (1 + 2 + 1)[26]	
GCS\Comms	Wifi	1	1e-5
GCS\Comms	Cellular 3G	2	1e-6
GCS\Comms	Radio	1	1e-5
Drone	Battery	4	1e-8
Drone	Position Unit	4 (2 + 2, 2 + 2)[27]	
Drone\PositionUnit	IMU	2	1e-6
Drone\PositionUnit	Barometer	2	1e-6
Drone\PositionUnit	GPS	2	1e-6
Drone\PositionUnit	APU	4	1e-8
Drone\Propulsion	Motor (x 4)	4	1e-8
Drone\FCU	CPU	4	1e-8
Drone	EDU	4 (2 + 2)[28]	
Drone\EDU	RGB Camera	2	1e-6
Drone\EDU	Thermal Camera	2	1e-6

[26] Comms failure (SIL 4) = Wifi failure (SIL 1) + 3G failure (SIL 2) + Radio failure (SIL 1)
[27] Position consists of horizontal location (GPS SIL 2 + IMU SIL 2) and altitude (Barometer SIL 2 + LIDAR SIL 2)
[28] EDU failure (SIL 4) = RGB Camera (SIL 2) + Thermal Camera (SIL 2)

Drone\EDU	LIDAR	2	1e-6
Drone	Comms Unit	4 (1 + 2 + 1)	
Drone\Comms	Wifi	1	1e-5
Drone\Comms	Cellular 3G	2	1e-6
Drone\Comms	Radio	1	1e-5

Table 1 - SIL and failure rates for all components

However, this only applies to the default hardware failure of each component. In some cases, there may be other types of failure mode. For example, for the FCU, we can define five basic events (failure modes):


Failure Data Editor - kiosDrone_v2/DefaultPerspective/Drone/FlightControllerUnit/CPU (Implementation1)

Basic Events | Common Causes | Output Deviations | Exported Propagations

Overheating
OtherFailure
PersonDetectionFalseNegative
PersonDetectionFalsePositive

Failure Propagation: Defined here and in the subsystem

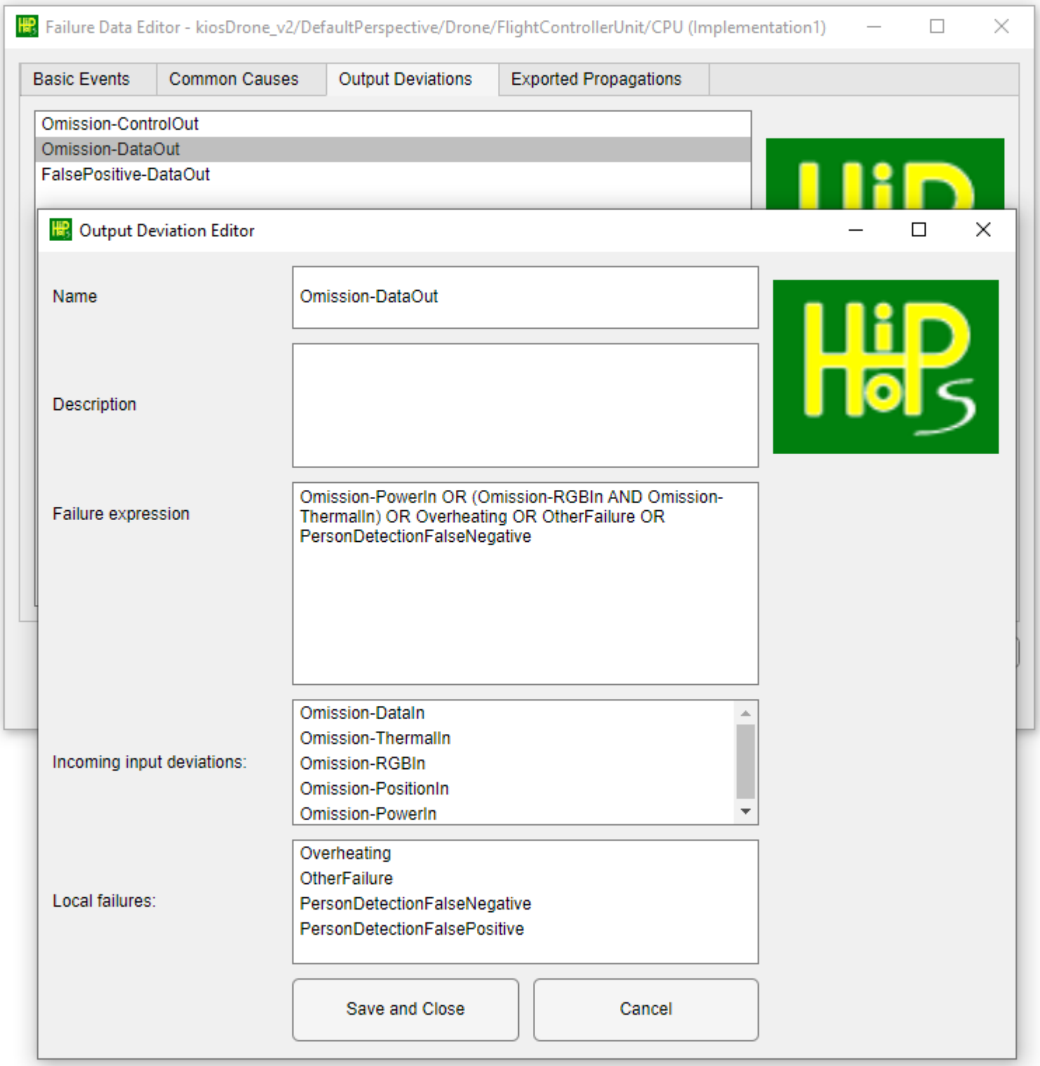
Figure 78 - Basic events of the FCU

In this case, "OtherFailure" is the default hardware failure (1e-8 failures/hr). Overheating is caused when the CPU exceeds maximum operating temperature threshold (due to environmental factors, e.g. fire); this is set as 1e-5 failures/hr, given the likely conditions the drone might operate in. Then there are two failures for false negative (0.01) and false positive (0.001) person detection. Failure rates here are simply broad estimates for the purposes of design time analysis, but techniques like SafeML can help provide more meaningful estimates once real datasets are available.

We also define the output deviations, i.e. failures being propagated from the outputs of the component. In the case of the FCU, we have two output ports (`DataOut` for data which is sent back to the GCS, and `ControlOut` for controlling the rotors). We define two output deviations for `DataOut` and one for `ControlOut` as follows:

Failure Class	Port	Cause
Omission	ControlOut	Omission-PowerIn OR Omission-PositionIn OR Omission-DataIn OR Overheating OR OtherFailure
Omission	DataOut	Omission-PowerIn OR (Omission-RGBIn AND Omission-ThermalIn) OR Overheating OR OtherFailure
FalsePositive	DataOut	PersonDetectionFalsePositive

Table 2 - Output deviations for the FCU



Omission-ControlOut
Omission-DataOut
FalsePositive-DataOut

Name: Omission-DataOut

Description:

Failure expression: Omission-PowerIn OR (Omission-RGBIn AND Omission-ThermalIn) OR Overheating OR OtherFailure OR PersonDetectionFalseNegative

Incoming input deviations:
Omission-DataIn
Omission-ThermalIn
Omission-RGBIn
Omission-PositionIn
Omission-PowerIn

Local failures:
Overheating
OtherFailure
PersonDetectionFalseNegative
PersonDetectionFalsePositive

Figure 79 - Output deviations of the FCU

For example, the figure above shows the cause of omission of data to DataOut, i.e., the failure of the FCU to send video data and information about people detected to the GCS. This is caused by:

1. No power to the CPU
2. An omission of data from *both* the RGB camera and the thermal camera
3. Overheating (CPU fails)
4. OtherFailure (CPU fails)
5. PersonDetectionFalseNegative (CPU incorrectly fails to identify a person)

Other components are annotated similarly, with output failure caused by internal hardware failure or omission of input. A few warrant a further mention, however:

- GPS can suffer from both an internal hardware failure and GPS jamming.
- Both cameras (and the LIDAR) can suffer from gimbal failure as well as hardware failure. This hardware failure can be the result of ordinary wear and tear or deliberate damage as part of a physical security attack.
- Failure of any single rotor is sufficient to cause overall propulsion failure. Again, rotor failure could be an ordinary hardware fault or it could be due to deliberate external damage.
- Position failure is caused by either a failure to establish horizontal location (both GPS *and* IMU failure) *or* failure to establish altitude (both barometer *and* LIDAR failure).

Additionally, four common cause failures (CCFs) are defined that affect communications between drone and GCS:

- Wifi jamming
- 3G jamming
- Radio jamming
- Wifi security attack (further information would be provided by subsequent security analysis)

These all have fixed unavailabilities of 1e-7, reflecting the overall low (but not impossible) chance of a malicious attack on the system.

It is also important to define the hazards, which serve as the top-level starting point for a safety analysis. The earlier HARA identified two key hazards:

- H1: Failure of the drone to locate survivors, caused by:

  - Failure of the drone's onboard sensors (camera, LIDAR, etc), whether due to hardware failure or environmental factors
  - Failure of the person detection algorithms designed to automatically identify survivors (ML-driven)
  - Failure of the drone's propulsion (inability to fly), including battery failure
  - Inability to navigate (navigation system failure, including potential GPS jamming, potentially also obscured/inoperative onboard sensors)
  - Inability to communicate (hardware failures, radio jamming, etc)

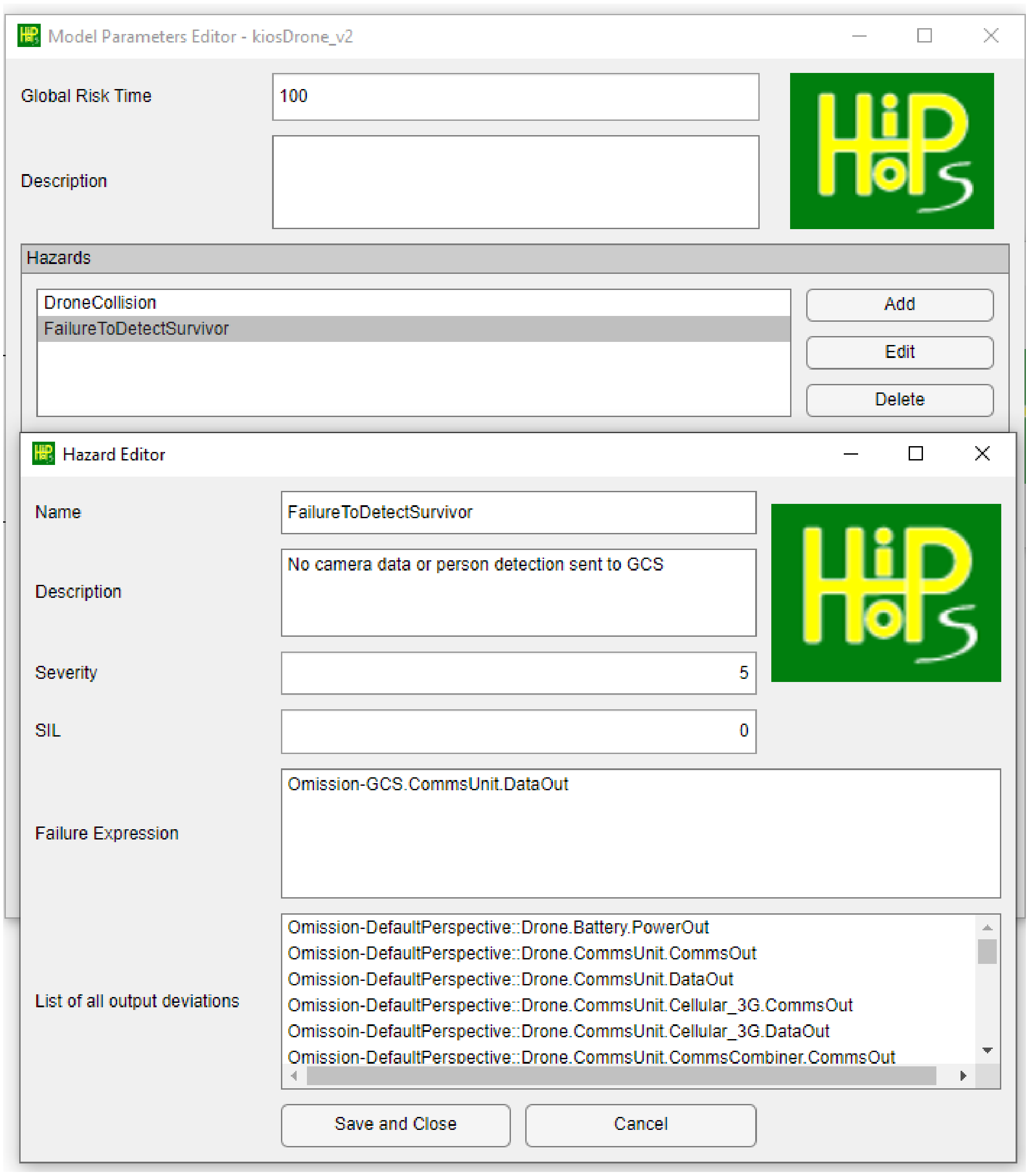
- H2: Collision between drone and the environment, caused by:
  - Navigation error (navigation system failure, GPS jamming)
  - Propulsion failure (e.g. failure of one or more motors leading to the drone crashing), including battery failure

The risk assessment found that H1 has the highest risk (16), while H2 has a slightly lower risk (10) despite being more severe, due to the lower likelihood of its causes.

In HiP-HOPS, we establish the cause of these hazards as follows:

- H1 = Omission-GCS.CommsUnit.DataOut
  - No data received from the drone, meaning we don't detect a survivor in need
- H2 = Omission-Drone.Propulsion.Out
  - Either due to lack of control signal, power failure, or motor failure, the drone flies out of control

We also assume the system is at risk for approximately 100 hours (4 days of operation).


Model Parameters Editor - kiosDrone_v2

Global Risk Time: 100

Description:

Hazards
- DroneCollision
- FailureToDetectSurvivor

Hazard Editor

Name: FailureToDetectSurvivor

Description: No camera data or person detection sent to GCS

Severity: 5

SIL: 0

Failure Expression: Omission-GCS.CommsUnit.DataOut

List of all output deviations:

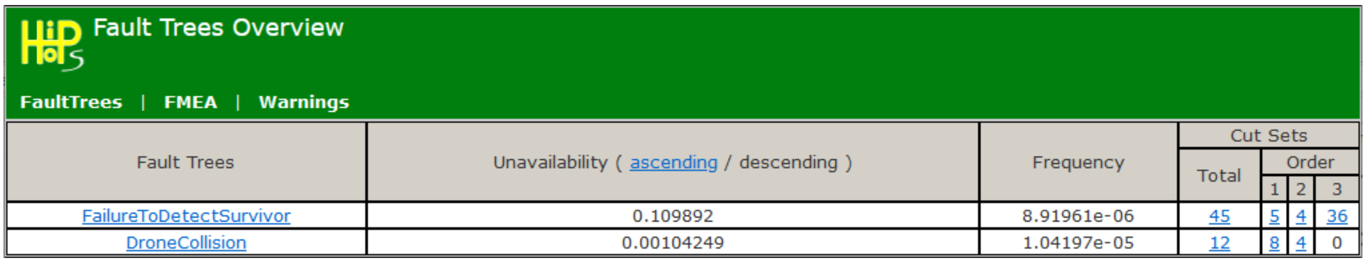
```
Omission-DefaultPerspective::Drone.Battery.PowerOut
Omission-DefaultPerspective::Drone.CommsUnit.CommsOut
Omission-DefaultPerspective::Drone.CommsUnit.DataOut
Omission-DefaultPerspective::Drone.CommsUnit.Cellular_3G.CommsOut
Omissoin-DefaultPerspective::Drone.CommsUnit.Cellular_3G.DataOut
Omission-DefaultPerspective::Drone.CommsUnit.CommsCombiner.CommsOut
```

Figure 80 - Hazard definition in HiP-HOPS

7.1.3 Safety Analysis

Once all of the failure data has been defined, it can be combined to synthesise fault trees and an FMEA so that an overall analysis of the system can take place.

HiP-HOPS produces its results in web browser-viewable form. The initial page presents a summary of the fault trees, their cut sets, and the unavailability:



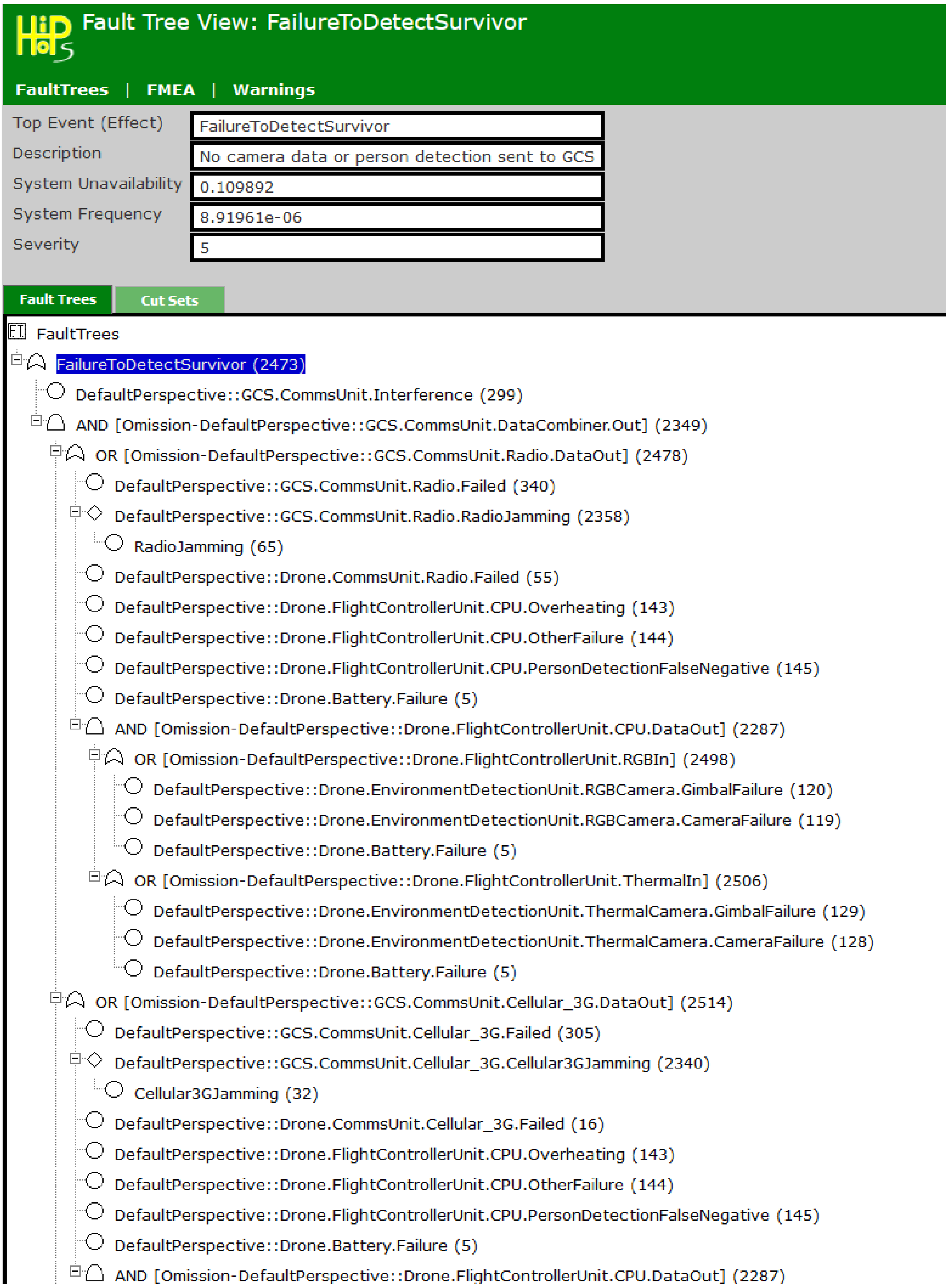
Fault Trees	Unavailability (ascending / descending)	Frequency	Cut Sets			
			Total	Order		
				1	2	3
FailureToDetectSurvivor	0.109892	8.91961e-06	45	5	4	36
DroneCollision	0.00104249	1.04197e-05	12	8	4	0

Figure 81 - Fault Tree summary

Here we can see that there are 45 cut sets for H1 (failure to detect survivor) — i.e., 45 possible combinations of failures that cause this hazard — and 12 cut sets for H2 (drone collision).

Clicking on either of the fault tree links leads us to the results for that fault tree. We can view both the fault tree itself (Figure 82) and the minimal cut set results. Since there are 36 cut sets here, they are summarised as follows:

- 5 cut sets of order 1 (single points of failure):
    - Battery.failure (1-e8)
    - CPU.otherFailure (1e-8)
    - CPU.overheating (1e-3)
    - CPU.PersonDetectionFalseNegative (0.01)
    - CommsUnit.Interference (0.1)
- 4 cut sets of order 2:
    - Combinations of (each 1e-10):
        - RGB camera failure or RGB gimbal failure
        - Thermal camera failure or thermal gimbal failure
- 36 cut sets of order 3:
    - Combinations of communications failures:
        - Wifi failure (jamming, drone wifi failure, GCS wifi failure, wifi security attack)
        - 3G failure (jamming, drone 3G failure, GCS 3G failure)
        - Radio failure (jamming, drone radio failure, GCS radio failure)


Fault Tree View: FailureToDetectSurvivor
FaultTrees | FMEA | Warnings
Fault Trees
Cut Sets

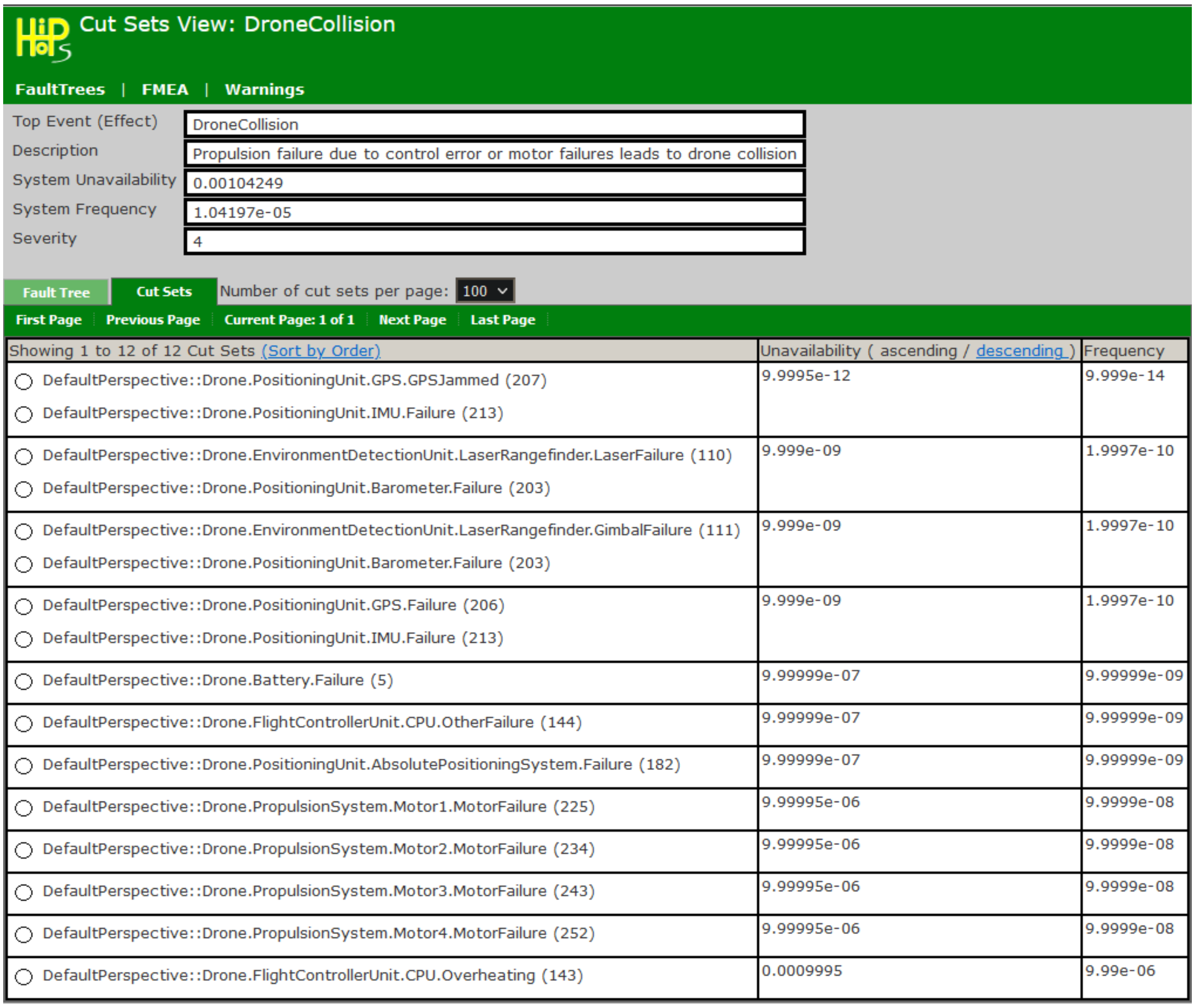
Top Event (Effect)	FailureToDetectSurvivor
Description	No camera data or person detection sent to GCS
System Unavailability	0.109892
System Frequency	8.91961e-06
Severity	5

```
FaultTrees
FailureToDetectSurvivor (2473)
  DefaultPerspective::GCS.CommsUnit.Interference (299)
  AND [Omission-DefaultPerspective::GCS.CommsUnit.DataCombiner.Out] (2349)
    OR [Omission-DefaultPerspective::GCS.CommsUnit.Radio.DataOut] (2478)
      DefaultPerspective::GCS.CommsUnit.Radio.Failed (340)
      DefaultPerspective::GCS.CommsUnit.Radio.RadioJamming (2358)
        RadioJamming (65)
      DefaultPerspective::Drone.CommsUnit.Radio.Failed (55)
      DefaultPerspective::Drone.FlightControllerUnit.CPU.Overheating (143)
      DefaultPerspective::Drone.FlightControllerUnit.CPU.OtherFailure (144)
      DefaultPerspective::Drone.FlightControllerUnit.CPU.PersonDetectionFalseNegative (145)
      DefaultPerspective::Drone.Battery.Failure (5)
      AND [Omission-DefaultPerspective::Drone.FlightControllerUnit.CPU.DataOut] (2287)
        OR [Omission-DefaultPerspective::Drone.FlightControllerUnit.RGBIn] (2498)
          DefaultPerspective::Drone.EnvironmentDetectionUnit.RGBCamera.GimbalFailure (120)
          DefaultPerspective::Drone.EnvironmentDetectionUnit.RGBCamera.CameraFailure (119)
          DefaultPerspective::Drone.Battery.Failure (5)
        OR [Omission-DefaultPerspective::Drone.FlightControllerUnit.ThermalIn] (2506)
          DefaultPerspective::Drone.EnvironmentDetectionUnit.ThermalCamera.GimbalFailure (129)
          DefaultPerspective::Drone.EnvironmentDetectionUnit.ThermalCamera.CameraFailure (128)
          DefaultPerspective::Drone.Battery.Failure (5)
    OR [Omission-DefaultPerspective::GCS.CommsUnit.Cellular_3G.DataOut] (2514)
      DefaultPerspective::GCS.CommsUnit.Cellular_3G.Failed (305)
      DefaultPerspective::GCS.CommsUnit.Cellular_3G.Cellular3GJamming (2340)
        Cellular3GJamming (32)
      DefaultPerspective::Drone.CommsUnit.Cellular_3G.Failed (16)
      DefaultPerspective::Drone.FlightControllerUnit.CPU.Overheating (143)
      DefaultPerspective::Drone.FlightControllerUnit.CPU.OtherFailure (144)
      DefaultPerspective::Drone.FlightControllerUnit.CPU.PersonDetectionFalseNegative (145)
      DefaultPerspective::Drone.Battery.Failure (5)
      AND [Omission-DefaultPerspective::Drone.FlightControllerUnit.CPU.DataOut] (2287)
```

Figure 82 - Fault tree for the "failure to detect survivor" (H1)

The overall unavailability estimate for H1 is 0.109892. This is primarily driven by the possibility of communications interference (0.1), a false negative of the person detection algorithm (0.01), or the overheating of the CPU (0.001).

For H2, the failure of the drone propulsion system leading to collision, we have 12 cut sets:


Cut Sets View: DroneCollision
FaultTrees | FMEA | Warnings
Fault Tree
Cut Sets
Number of cut sets per page: 100
First Page
Previous Page
Current Page: 1 of 1
Next Page
Last Page

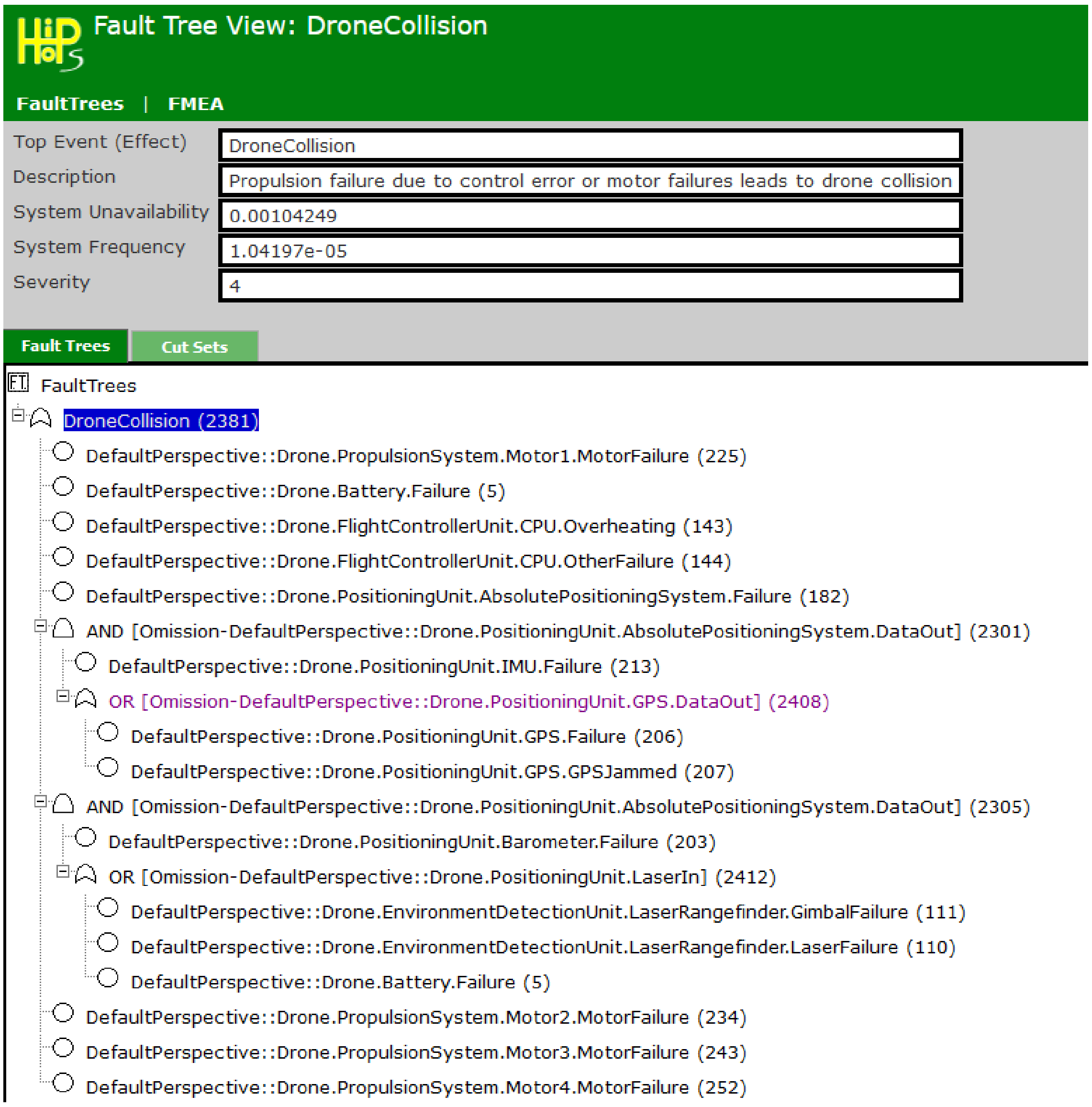
Top Event (Effect)	DroneCollision
Description	Propulsion failure due to control error or motor failures leads to drone collision
System Unavailability	0.00104249
System Frequency	1.04197e-05
Severity	4

Showing 1 to 12 of 12 Cut Sets (Sort by Order)	Unavailability (ascending / descending)	Frequency
○ DefaultPerspective::Drone.PositioningUnit.GPS.GPSJammed (207) ○ DefaultPerspective::Drone.PositioningUnit.IMU.Failure (213)	9.9995e-12	9.999e-14
○ DefaultPerspective::Drone.EnvironmentDetectionUnit.LaserRangefinder.LaserFailure (110) ○ DefaultPerspective::Drone.PositioningUnit.Barometer.Failure (203)	9.999e-09	1.9997e-10
○ DefaultPerspective::Drone.EnvironmentDetectionUnit.LaserRangefinder.GimbalFailure (111) ○ DefaultPerspective::Drone.PositioningUnit.Barometer.Failure (203)	9.999e-09	1.9997e-10
○ DefaultPerspective::Drone.PositioningUnit.GPS.Failure (206) ○ DefaultPerspective::Drone.PositioningUnit.IMU.Failure (213)	9.999e-09	1.9997e-10
○ DefaultPerspective::Drone.Battery.Failure (5)	9.99999e-07	9.99999e-09
○ DefaultPerspective::Drone.FlightControllerUnit.CPU.OtherFailure (144)	9.99999e-07	9.99999e-09
○ DefaultPerspective::Drone.PositioningUnit.AbsolutePositioningSystem.Failure (182)	9.99999e-07	9.99999e-09
○ DefaultPerspective::Drone.PropulsionSystem.Motor1.MotorFailure (225)	9.99995e-06	9.9999e-08
○ DefaultPerspective::Drone.PropulsionSystem.Motor2.MotorFailure (234)	9.99995e-06	9.9999e-08
○ DefaultPerspective::Drone.PropulsionSystem.Motor3.MotorFailure (243)	9.99995e-06	9.9999e-08
○ DefaultPerspective::Drone.PropulsionSystem.Motor4.MotorFailure (252)	9.99995e-06	9.9999e-08
○ DefaultPerspective::Drone.FlightControllerUnit.CPU.Overheating (143)	0.0009995	9.99e-06

Figure 83 - Drone collision cut sets

These consist of combinations of position unit failures (GPS and IMU or barometer and LIDAR), APS failure, power failure, CPU failure, or a motor failure.

The fault tree for H2 is shown below:

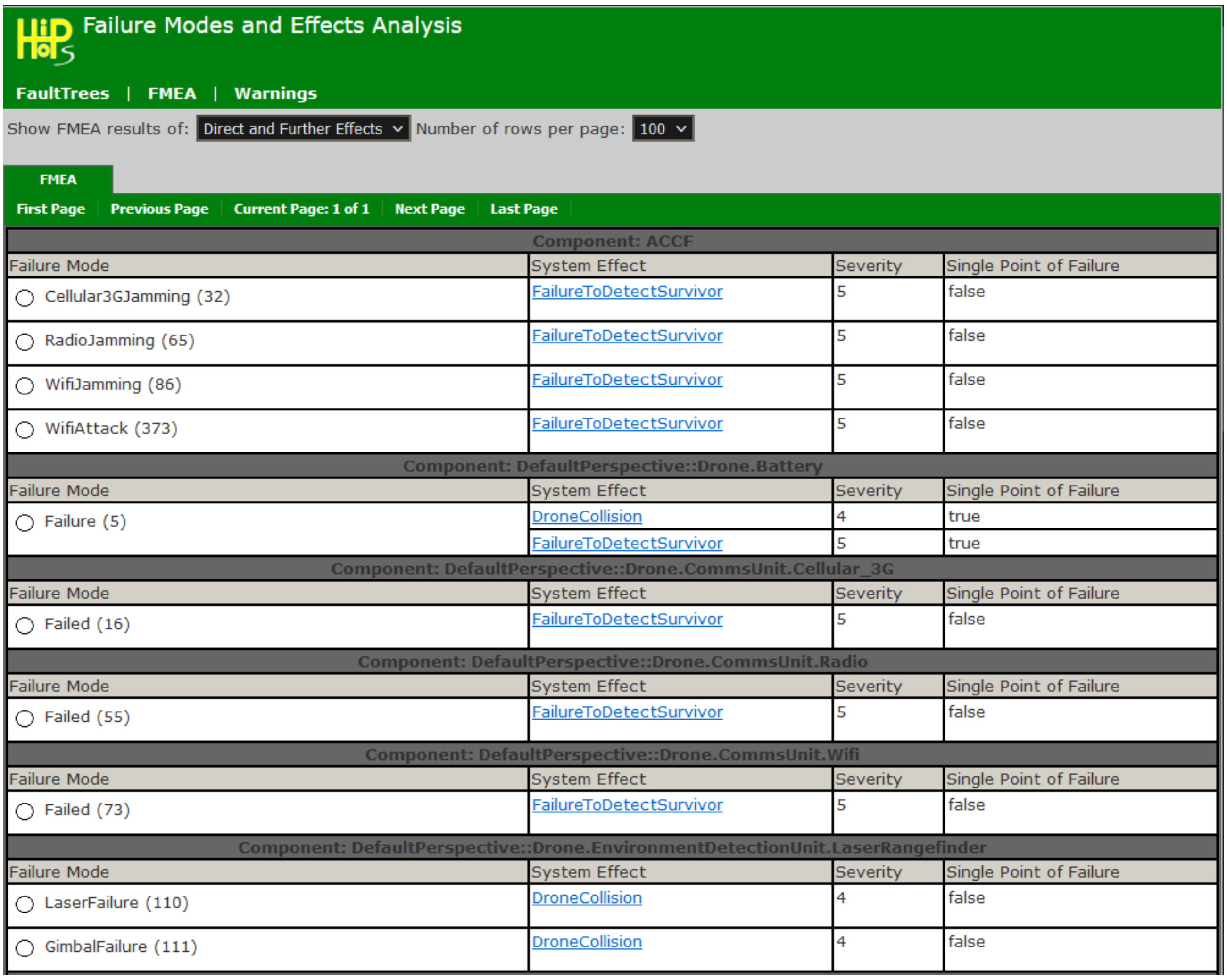


Top Event (Effect)	DroneCollision
Description	Propulsion failure due to control error or motor failures leads to drone collision
System Unavailability	0.00104249
System Frequency	1.04197e-05
Severity	4

```
FaultTrees
DroneCollision (2381)
  DefaultPerspective::Drone.PropulsionSystem.Motor1.MotorFailure (225)
  DefaultPerspective::Drone.Battery.Failure (5)
  DefaultPerspective::Drone.FlightControllerUnit.CPU.Overheating (143)
  DefaultPerspective::Drone.FlightControllerUnit.CPU.OtherFailure (144)
  DefaultPerspective::Drone.PositioningUnit.AbsolutePositioningSystem.Failure (182)
  AND [Omission-DefaultPerspective::Drone.PositioningUnit.AbsolutePositioningSystem.DataOut] (2301)
    DefaultPerspective::Drone.PositioningUnit.IMU.Failure (213)
    OR [Omission-DefaultPerspective::Drone.PositioningUnit.GPS.DataOut] (2408)
      DefaultPerspective::Drone.PositioningUnit.GPS.Failure (206)
      DefaultPerspective::Drone.PositioningUnit.GPS.GPSJammed (207)
  AND [Omission-DefaultPerspective::Drone.PositioningUnit.AbsolutePositioningSystem.DataOut] (2305)
    DefaultPerspective::Drone.PositioningUnit.Barometer.Failure (203)
    OR [Omission-DefaultPerspective::Drone.PositioningUnit.LaserIn] (2412)
      DefaultPerspective::Drone.EnvironmentDetectionUnit.LaserRangefinder.GimbalFailure (111)
      DefaultPerspective::Drone.EnvironmentDetectionUnit.LaserRangefinder.LaserFailure (110)
      DefaultPerspective::Drone.Battery.Failure (5)
  DefaultPerspective::Drone.PropulsionSystem.Motor2.MotorFailure (234)
  DefaultPerspective::Drone.PropulsionSystem.Motor3.MotorFailure (243)
  DefaultPerspective::Drone.PropulsionSystem.Motor4.MotorFailure (252)
```

Figure 84 - Drone collision fault tree

We can also look at the FMEA, which displays the effects of each component failure mode. HiP-HOPS can display both direct effects (the failure mode directly causes the hazardous system effect) and further effects (the failure mode contributes to the hazard in conjunction with other failure modes).



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

Failure Mode	System Effect	Severity	Single Point of Failure
Component: ACCF			
○ Cellular3GJamming (32)	FailureToDetectSurvivor	5	false
○ RadioJamming (65)	FailureToDetectSurvivor	5	false
○ WifiJamming (86)	FailureToDetectSurvivor	5	false
○ WifiAttack (373)	FailureToDetectSurvivor	5	false
Component: DefaultPerspective::Drone.Battery			
○ Failure (5)	DroneCollision	4	true
	FailureToDetectSurvivor	5	true
Component: DefaultPerspective::Drone.CommsUnit.Cellular_3G			
○ Failed (16)	FailureToDetectSurvivor	5	false
Component: DefaultPerspective::Drone.CommsUnit.Radio			
○ Failed (55)	FailureToDetectSurvivor	5	false
Component: DefaultPerspective::Drone.CommsUnit.Wifi			
○ Failed (73)	FailureToDetectSurvivor	5	false
Component: DefaultPerspective::Drone.EnvironmentDetectionUnit.LaserRangefinder			
○ LaserFailure (110)	DroneCollision	4	false
○ GimbalFailure (111)	DroneCollision	4	false

Figure 85 - Part of the FMEA

Figure 85 shows part of the FMEA for the system. At the top are the various common cause failures affecting the communications; each of these are insufficient to cause H1 by themselves, but can in combination (hence "Single Point of Failure" = false). The battery failure, on the other hand, is not only a single point of failure but can cause both H1 and H2.

7.1.4 MRS aspects

Note that this analysis considers only two parts of the MRS: one drone and the ground control station. There is the possibility for the drones to cooperate and in doing so mitigate failure to some degree; for instance, a drone that suffers from position unit failure may be able to use collaborative localisation, e.g. triangulation with other drones, to approximate its own location and continue its mission (or at least find a safe place to land).

On a higher level, multiple drones can cover the same area in a shorter amount of time, and the GCS can allocate replacements when a drone fails to ensure the area allocated to that drone is still surveyed.

These aspects are hard to capture in a design-time analysis. While the model could be extended with one or two additional drones, in practice this would simply bloat the results by duplicating everything. There are already 36 cut sets regarding communication failure between the drone and the GCS; if we were to attempt to model a potential localisation failure between 3 drones, this number would increase exponentially.

Approximations can be made, e.g. by adding a specific component to represent the rest of the drone swarm so that simple failures can be taken into account without cluttering the results with the simultaneous internal failures of multiple drones.

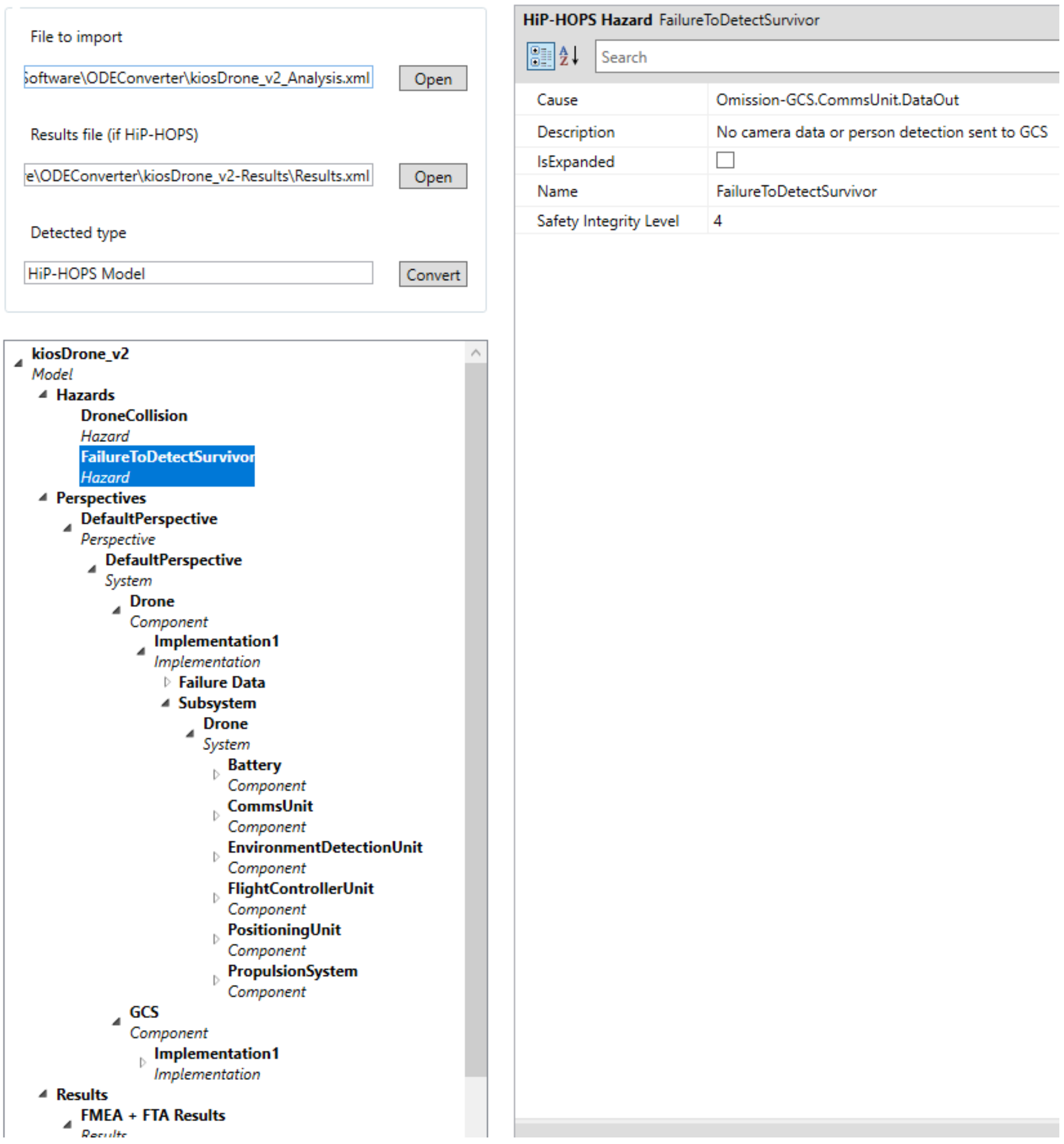
However, it is better to address these elements using runtime EDDIs that can better model the dynamic, adaptive circumstances involved in the scenario.

7.1.5 Conversion to ODE model and Preparation for Runtime

Once the HiP-HOPS model has been constructed and analysed, it can be exported to an ODE XML file. In this form, it can be kept as an information repository about the system — its architecture, its failure behaviour, decomposition of safety integrity levels, and so forth — and by adding additional information about the dependability requirements, an all-in-one safety case can be created.

This same file can also be used as the basis for a runtime EDDI, augmenting it with runtime-specific information, adding dynamic models like Bayesian networks or ConSerts, and specifying events and actions, etc.

Either way, the first step is to convert it to an ODE file. This can be done via the Common Tool Adapter or via the EDDI Editor. The latter is preferred since it can capture both aspects of the overall model: the system architecture and the analysis results. Both files can be imported into the EDDI Editor to obtain a single merged ODE model that contains both sets of data.


File to import

ware\ODEConverter\kiosDrone_v2_Analysis.xml [Open]

Results file (if HiP-HOPS)

e\ODEConverter\kiosDrone_v2-Results\Results.xml [Open]

Detected type

HiP-HOPS Model [Convert]

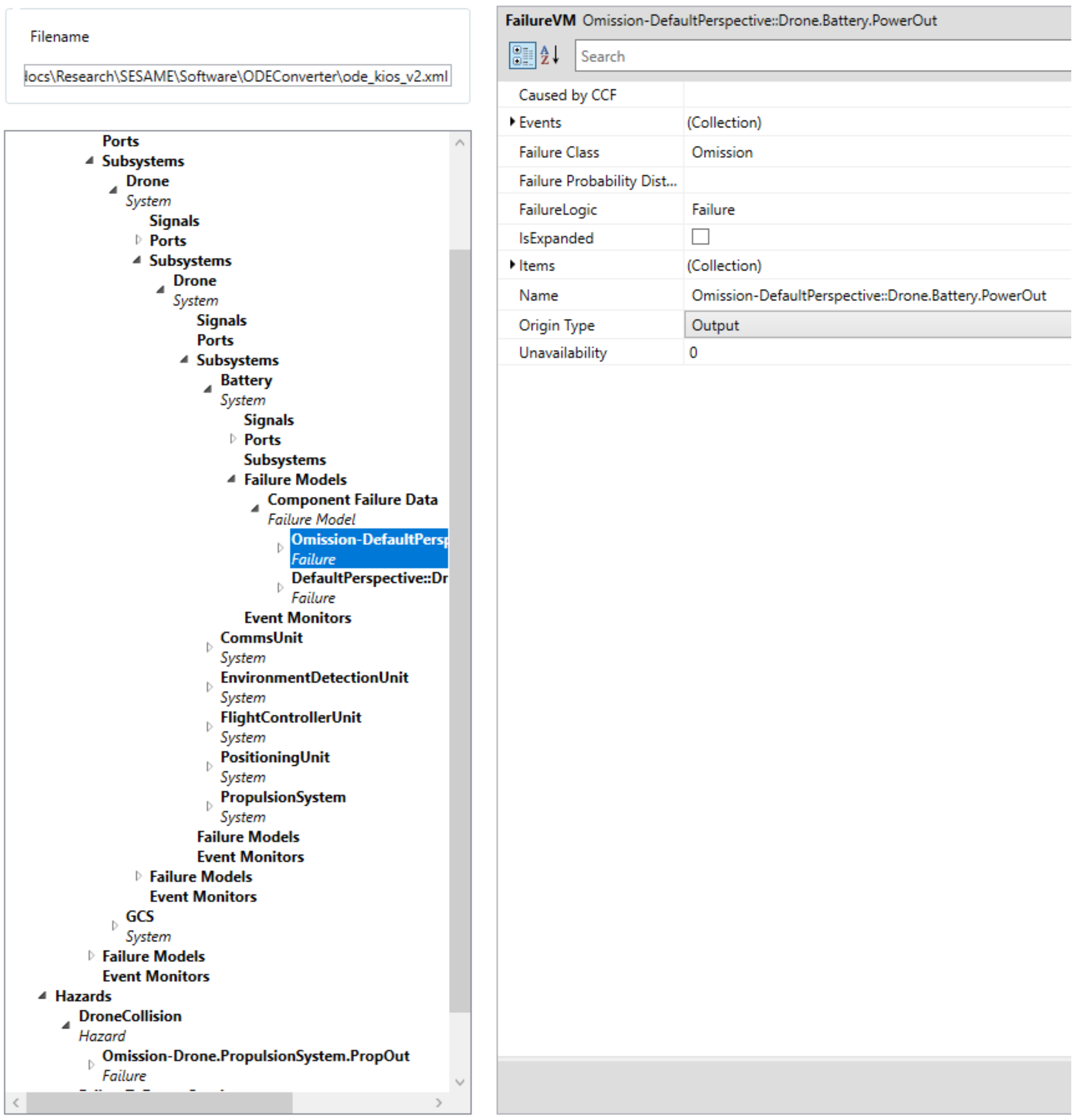
```
kiosDrone_v2
Model
  Hazards
    DroneCollision
    Hazard
    FailureToDetectSurvivor
    Hazard
  Perspectives
    DefaultPerspective
    Perspective
      DefaultPerspective
      System
        Drone
        Component
          Implementation1
          Implementation
            Failure Data
            Subsystem
              Drone
              System
                Battery
                Component
                CommsUnit
                Component
                EnvironmentDetectionUnit
                Component
                FlightControllerUnit
                Component
                PositioningUnit
                Component
                PropulsionSystem
                Component
        GCS
        Component
          Implementation1
          Implementation
  Results
    FMEA + FTA Results
```

HiP-HOPS Hazard FailureToDetectSurvivor

Cause	Omission-GCS.CommsUnit.DataOut
Description	No camera data or person detection sent to GCS
IsExpanded	☐
Name	FailureToDetectSurvivor
Safety Integrity Level	4

Figure 86 - Importing the two HiP-HOPS files

Once imported, the merged model can be edited if required before it is converted to an equivalent ODE model:



Filename

locs\Research\SESAME\Software\ODEConverter\ode_kios_v2.xml

```
Ports
Subsystems
  Drone
  System
    Signals
    Ports
    Subsystems
      Drone
      System
        Signals
        Ports
        Subsystems
          Battery
          System
            Signals
            Ports
            Subsystems
            Failure Models
              Component Failure Data
              Failure Model
                Omission-DefaultPersp
                Failure
                DefaultPerspective::Dr
                Failure
            Event Monitors
          CommsUnit
          System
          EnvironmentDetectionUnit
          System
          FlightControllerUnit
          System
          PositioningUnit
          System
          PropulsionSystem
          System
        Failure Models
        Event Monitors
    Failure Models
    Event Monitors
  GCS
  System
Failure Models
Event Monitors
Hazards
  DroneCollision
  Hazard
    Omission-Drone.PropulsionSystem.PropOut
    Failure
```

FailureVM Omission-DefaultPerspective::Drone.Battery.PowerOut

Search

Caused by CCF	
Events	(Collection)
Failure Class	Omission
Failure Probability Dist...	
FailureLogic	Failure
IsExpanded	☐
Items	(Collection)
Name	Omission-DefaultPerspective::Drone.Battery.PowerOut
Origin Type	Output
Unavailability	0

Figure 87 - Converted ODE model

The EDDI Editor allows other models (ODE or other imported models, including failure models like BNs or state machines) to be merged into the system hierarchy. It also allows the definition of events and actions that can help lay the foundation for runtime execution.

Once any edits to the model are complete, the new EDDI can be saved as an XML compliant with the ODE metamodel.

More information on the runtime aspects of EDDIs and their generation can be found in the WP7 deliverables.

8. CONCLUSION

Within this document, three safety analysis tools have been described: HiP-HOPS, safeTbox, and Dymodia, plus the Bayesian network tool GeNIe. Information about what they are and how they work has been presented, showing how these tools can be used to create system models and safety artefacts at design time — artefacts that capture the type of information needed to produce EDDIs, either for runtime usage or as design time artefacts themselves (e.g. for the purposes of safety argumentation).

The ODE metamodel serves as the foundation for the EDDI, acting as a superset of all the information an EDDI might need to store. Translation from models produced by proprietary design-time safety analysis tools to ODE-compliant models is possible via either the Common Tool Adapter or the EDDI Editor. These allow conversion of tool-specific file formats to ODE XML files that can then be further processed for generation of runtime EDDIs or used in other parts of the EDDI toolchain.

The Common Tool Adapter is supported by an automated ODE updater, which allows rapid and automatic regeneration of the necessary data to update the Adapter in response to ODE metamodel changes.

The EDDI Editor provides additional support for tool heterogeneity and modification of ODE-based EDDI models in preparation for runtime EDDI generation. Separate models, potentially created by separate tools, can be merged and composed to help address issues of scalability and distribution of dependability processes over multiple partners.

Finally, to demonstrate how this pipeline looks in action, the HiP-HOPS tool was applied to an example based on the KIOS/CCD power station inspection case study, following on from the general walkthrough provided in **D4.5 Safety Analysis Concept & Methodology** to create a structured system model (in Matlab Simulink) with failure data annotations and analysis results (via HiP-HOPS) and finally to an ODE XML file (via the EDDI Editor).

9. REFERENCES

[1] Y. I. Papadopoulos and J. A. McDermid, “Hierarchically performed hazard origin and propagation studies,” in *Proceedings of the 18th International Conference in Computer Safety, Reliability, and Security; collected in LNCS by Springer, vol 1698 (p139-152)*, Toulouse, France, 1999.

[2] Y. I. Papadopoulos, M. D. Walker, D. J. Parker, E. Rude, R. Hamann, A. Uhlig, U. Gratz and R. Lien, “Engineering Failure Analysis & Design Optimisation with HiP-HOPS,” *Journal of Engineering Failure Analysis,* vol. 18, no. 2, pp. 590-608, 2011.

[3] M. D. Walker, L. Bottaci and Y. I. Papadopoulos, “Compositional Temporal Fault Tree Analysis,” *Computer Safety, Reliability, and Security. SAFECOMP 2007. Lecture Notes in Computer Science,* vol. 4680, pp. 106-119, 2007.

[4] D. J. Parker, M. D. Walker, L. S. Azevedo, Y. I. Papadopoulos and R. E. Araujo, “Automatic Decomposition and Allocation of Safety Integrity Levels Using a Penalty-Based Genetic Algorithm,” in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems,* Amsterdam, Netherlands, 2013.

[5] B. Kaiser, P. Liggesmeyer and O. Mäckel, “A New Component Concept for Fault Trees.,” in *Safety Critical Systems and Software 2003, Eighth Australian Workshop on Safety-Related Programmable Systems, (SCS2003),* Canberra, Australia, 2003.

[6] B. Kaiser, D. Schneider, R. Adler, D. Domis, F. Mohrle, A. Berres, M. Zeller, K. Hofig and M. Rothfelder, “Advances in component fault trees,” in *Safety and Reliability - Safe Societies in a Changing World*, London, UK, CRC Press, Taylor Francis, 2018, p. 9.

[7] A. C. W. Group, “Goal Structuring Notation Community Standard (Version 2),” January 2018. [Online]. Available: https://scsc.uk/scsc-141B. [Accessed 12 11 2021].