



Deliverable D2.3

Report: Fault Tree Applications

Deliverable

<i>Number and title:</i>	D2.3 – Report: Fault Tree Applications		
<i>Work package:</i>	WP2 (Reliability and Resilience)		
<i>Lead author:</i>	Thomas Noll (RWTH)		
<i>Contributors:</i>	Mareike Metzler (RWTH)		
<i>Reviewers:</i>	Mariëlle Stoelinga (UT), Leonardo Handszok (INVAP)		
<i>Due date (GA):</i>	M50 (2025-11-30)	<i>Dissemination level:</i>	Public
<i>Due date (revised):</i>	2025-12-31	<i>Version:</i>	1.0 (final)
<i>Submission date:</i>	2025-12-29	<i>Pages:</i>	22

Version history

<i>Version</i>	<i>Date</i>	<i>Notes</i>
1.0	2025-12-29	First official release

Project

<i>Title:</i>	Models in Space Systems: Integration, Operation, and Networking		
<i>Acronym:</i>	MISSION	<i>Start date:</i>	01-10-2021
<i>GA no.:</i>	101008233	<i>Duration:</i>	48 months
<i>Call:</i>	H2020-MSCA-RISE-2020	<i>Website:</i>	mission-project.eu



This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233. This document reflects only the authors' view. It does not represent the opinion of the European Union, and the European Union is not responsible for any use that may be made of the information it contains.

Contents

1	Introduction	3
2	Fault Trees: An Overview	3
2.1	Static Fault Trees	4
2.2	Dynamic Fault Trees	4
3	Automated Derivation of Fault Trees	4
3.1	AADL and COMPASS	5
3.2	SysML and SAFEST	5
4	Case Study: Satellite Constellations for Earth Observation	6
4.1	Modelling Principles	7
4.2	The AADL Model	8
4.3	The SysML Model	10
5	Validation of Fault Tree Technology	13
5.1	Analysis of Minimal Cut Sets	13
5.2	Discussion	15
6	Efficient Fault Tree Analysis	16
6.1	Automated Isolation of Static Parts in Dynamic Fault Trees	16
6.2	Modular Criticality Analysis	17
7	Enhancements of the Fault Tree Approach	18
7.1	Attack-Fault-Defense Trees	18
7.2	Statistical Model Checking	18
8	Conclusion	19

Acronyms

UT	University of Twente (beneficiary)
RWTH	Rheinisch-Westfälische Technische Hochschule Aachen (beneficiary)
USAAR	Universität des Saarlandes (beneficiary)
D3TN	D3TN GmbH (beneficiary)
OSM	Osmium Italy SRL (beneficiary)
UNC	Universidad Nacional de Córdoba (third-country partner)
UNRC	Universidad Nacional de Rio Cuarto (third-country partner)
INVAP	INVAP Sociedad del Estado (third-country partner)
ASC	Ascentio Technologies SA (third-country partner)
IISG	Institute of Intelligent Software Guangzhou (third-country partner)

1 Introduction

Assessing and ensuring Reliability, Availability, Maintainability and Safety (RAMS) is an important task in the design of any safety-critical system, in particular in the domain of aerospace systems and spacecraft [11]. No matter how well designed a system is, it still has to deal with the presence of faults to some extent. Faults in this context can be events such as equipment failures, wrong sensor readings, external interferences, and many more. To raise trust in handling system failures, reliability engineering tries to embed Fault Detection, Isolation and Recovery (FDIR) concepts. These concepts are derived using various tools and methodologies such as Fault Tree Analysis (FTA) [20], which is a methodology commonly used in industry for performing state-of-the-art failure analysis [27] and is also the main formalism for failure modelling adopted in the MISSION project.

Engineers used to construct fault trees manually, which requires specialised knowledge and considerable effort. It can be especially challenging when dealing with complex systems, particularly those made up of multiple components and featuring dynamic behaviour [27]. This applies, e.g., to the case of systems of satellite constellations, where satellites can take over functions from others or adapt their configuration. Here, manually creating fault trees is not only inefficient but also prone to human errors.

To address these limitations, approaches have been proposed that generate fault trees automatically [4]. The idea is to extract fault behaviour directly from formal system specifications written in modelling languages such as the Architecture Analysis and Design Language (AADL) or the Systems Modeling Language (SysML). Such automated methods and their accompanying tools promise better scalability, consistency, and refinement when analysing complex systems.

The purpose of this report, which is allocated to Work Package 2 (Reliability and Resilience), is to provide an integrated report on MISSION's advances in fault tree applications to the New Space setting, building on the automated generation of fault trees. It is based on previous Deliverables D2.1 [40] and, in particular, D2.2 [26]. The latter gives a survey of fault tree extensions with repair and maintenance aspects, investigates issues concerning semantics, and defines a modular formal semantics onto an appropriate automata model.

This document is organised as follows. Section 2 starts with an overview of (static) fault trees and their extension with dynamic elements. Section 3 describes two approaches for the automated generation of fault tree models from full-fledged system descriptions of fault-tolerant space architectures, which are respectively using the Architecture Analysis and Design Language (AADL) and the Systems Modeling Language (SysML) as underlying specification languages. Section 4 introduces our case study, a Swarm Satellite Constellation designed for an Earth observation mission. Section 5 presents the results of applying the techniques described in Section 3 to the case study. With the aim of further improving efficiency and scalability of FTA, Section 6 gives an overview of techniques for the automated isolation of static parts in dynamic fault trees, which allows to combine their BDD-based analysis with state-space based techniques for the dynamic parts. To broaden the context of our work, Section 7 briefly describes two enhancements of the fault tree approach, namely, the extension to attack-fault-defense trees and the combination with statistical model checking. Finally, Section 8 concludes our report.

2 Fault Trees: An Overview

Fault Tree Analysis (FTA) is a methodology commonly used in industry for performing state-of-the-art failure analysis [27]. Fault trees describe how faults propagate through components and sub-systems of a system and eventually lead to a top-level system failure. They enable various forms of both qualitative and quantitative analyses. The former aim to identify fault combinations that lead to system failure. The latter are based on assigning failure probabilities to components, which allows for the computation of important RAMS metrics such as overall reliability or mean time to failure.

In the following, we provide a survey of related work on (static) fault trees and their extension with

dynamic elements as well as repair and maintenance aspects, putting the emphasis on approaches with a solid formal foundation.

2.1 Static Fault Trees

The most basic case of a fault tree is known as a *Static Fault Tree (SFT)*, which models how failures of (sub-)components propagate through the system and eventually lead to a system failure. The creation of a fault tree follows a top-down approach. It starts with the *Top-Level Event (TLE)*, which represents the failure of the entire system. The system failure is subdivided into failures of sub-components, which can then be further subdivided. This hierarchical approach is continued down to the desired level of detail. The leaves in the resulting tree represent components which are not further decomposed, so-called *Basic Events (BEs)*. A basic event fails according to its associated probability distribution, if given. If a BE fails, the failure is propagated upwards through the fault tree. The intermediate nodes – so-called *gates* – fail themselves if their failure condition is satisfied. For instance, gates of type AND fail if all of their sub-components have failed whereas gates of type OR fail if at least one of their sub-components has failed. The semantics of static fault trees is therefore clearly defined as a Boolean function over the BEs encoding the system's failure conditions. For a detailed overview of different analysis techniques for static fault trees, we refer to [27].

2.2 Dynamic Fault Trees

While static fault trees are widely used in practice, they are not equipped to faithfully model more complex – dynamic – behaviour as it is usually present in modern hardware and software systems. For this reason, they have been extended to *Dynamic Fault Trees (DFTs)*, which incorporate temporal aspects and new features to analyse both spare management and temporal as well as functional dependencies [14]. More concretely, DFTs introduce several new types of gates such as

- priority-AND (PAND) and sequence enforcer (SEQ) for order-dependent failures,
- SPARE for spare management and activation, and
- FDEP for functional dependency.

These gates increase the expressiveness and allow for more faithful modelling compared to the basic (static) formalism. However, this advantage comes at the price of raising semantic problems – the dynamic nature of the additional gates makes a clear definition of the semantics more complicated [12]. While each dynamic gate on its own seems to have a concise definition, the interplay between different gates leads to intricate issues which are often overlooked and not properly defined in the literature [21]. These semantic issues are addressed in the work presented in [22, 37], which aims to capture and compare the existing DFT semantics. To this end, it introduces a thorough semantic of DFTs in terms of Generalised Stochastic Petri Nets [1], which is described in detail in Deliverable D2.2 [26].

3 Automated Derivation of Fault Trees

In an international workshop at INVAP in March 2025, the usage of DFTs for analysing the failure behaviour of satellite constellations was discussed, motivated by a maritime surveillance case study developed earlier (see Section 4 and [18]). Such systems usually exhibit a high degree of dynamic behaviour due to, for example, reconfiguration operations triggered by component failures or communication issues. Reconfiguration operations are applied both locally (e.g., by switching to a backup processor) and globally (e.g., by exchanging the role of master and slave satellites). The attempt to manually create a DFT that fully reflects those aspects soon confirmed the general observation made

in the introduction: The manual construction of fault trees for such complex systems is both inefficient and error-prone and therefore actually not feasible.

This was the starting point for our approach to investigate, develop and apply techniques for automatically generating fault trees from system descriptions [4]. Amongst others, such methods have been applied to the model-based fault analysis of aerospace systems specified in AADL [2, 16] and SysML [24, 25, 41]. However, to the best of our knowledge only single-satellite systems have been considered so far. The only formal approach to model-based fault analysis of satellite constellations that we are aware of is described in [17] and employs a stochastic Petri net model rather than fault trees.

Our work aims to fill this gap by modelling and analysing an exemplary (small) satellite constellation using the AADL and SysML specification languages and two accompanying software tools. More concretely, we assess the capabilities of the COMPASS and SAFEST frameworks with regard to the automated generation of fault trees from system models and their subsequent evaluation and analysis. These frameworks are briefly introduced in the following two sections.

3.1 AADL and COMPASS

The *COMPASS Toolset*¹ (*CORrectness, Modeling and Performance of AeroSpace Systems*) is a model-based framework designed to support the software/hardware co-engineering of aerospace systems. Its goal is to enable systematic analysis of correctness, reliability and performance from the early design stages, following a holistic approach. It has been developed by RWTH in collaboration with the European Space Agency (ESA) and Fondazione Bruno Kessler (FBK) in Trento, Italy [6].

COMPASS employs the *System-Level Integrated Modelling (SLIM)* language, which extends the *Architecture Analysis and Design Language (AADL*²) [7]. SLIM allows detailed representation of nominal behaviour as well as hardware and software failure scenarios. The modelling approach is component-based, distinguishing between hardware, software, and composite components. Components are defined by their type, which specifies functional interfaces, and their implementation, which describes internal structures, connections, and mode transitions. Fault models are similarly defined by error types, propagation, and implementation details, including probabilistic fault occurrences and repair mechanisms. Models are typically developed from a nominal system description and extended with potential faults and their propagation to support safety analyses. For our work, we are using the current version COMPASS 3.0.

With regard to our application scenario, the tool's capabilities concerning the automated generation of fault trees are particularly important. They are based on model-checking techniques as implemented in the NuSMV Tool [10], which is part of the COMPASS Toolset. It systematically generates all possible minimal combinations of component failures, called *Minimal Cut Sets* or *Min-Cut Sets (MCS)*, that cause a top-level system failure, i.e., trigger the top-level event. Next, those MCS are arranged in the fault tree representation [9]. Later, this technique has been extended to the contract-based design paradigm, where components at different levels of abstraction are characterized in terms of the properties that they have to guarantee and the assumptions that must be satisfied by their environment [8]. This enables the efficient generation of hierarchically organised fault trees.

3.2 SysML and SAFEST

The *Systems Modeling Language (SysML*³) is a modelling language developed by the Object Management Group (OMG) in 2006. The initiative aimed to provide a framework for designing, analysing, and verifying complex systems in both hardware and software. SysML extends UML 2.0 with features

¹<https://compass.fbk.eu>

²<https://www.sei.cmu.edu/projects/architecture-analysis-and-design-language-aadl/>

³<https://www.omg.org/sysml/>

such as value types and support for continuous system behaviour. It focuses less on software alone and enables system engineers to describe systems from multiple perspectives, including behaviour, structure, and requirements. SysML is widely used across engineering disciplines and has been the subject of numerous research studies. Here we use SysML v2.

The *SAFEST Tool*⁴ (*Static/Dynamic Fault Tress & Reward Event Trees Analysis Tool*) is a web-based platform designed for the comprehensive modelling, visualisation, and interactive simulation of both static and dynamic fault trees, as well as event trees. Just like COMPASS, it applies probabilistic model-checking algorithms to provide efficient and reliable assessments of system safety and reliability. Additionally, SAFEST incorporates optimisation techniques to enhance the analysis process, ensuring a powerful toolset for probabilistic risk assessment in modern complex systems [39]. For our work, we employed SAFEST 2.0.0, as the current version 3.0.0 was not yet available.

Concerning our application, we note that SAFEST extracts fault-tree information from manually-made safety annotations in the SysML specification rather than generating them automatically. Once the fault tree has been extracted, it can be visualised and further analysed by SAFEST, using the probabilistic model checker STORM⁵ as the underlying engine.

4 Case Study: Satellite Constellations for Earth Observation

Swarm Satellite Constellations (SSC) represent a novel approach to Earth observation missions. The concept of a Segmented Architecture (SA), developed by the Argentinian Space Agency (CONAE), reflects a new philosophy in space system design. This architecture is based on a networked constellation of small, cooperative satellites, which offers more flexibility and a higher degree of autonomy in comparison to classical designs.

An example is the hypothetical SARE (Sistema Argentino de Alta REvisita) mission, which aims to monitor Argentina's Exclusive Economic Zone (EEZ) in order to detect illegal maritime activities [18]. The case study considered in our work investigates a system composed of three Low-Earth Orbit (LEO) satellites. Each satellite is equipped with an X-band Synthetic Aperture Radar (SAR) payload designed for maritime target detection. The satellites are placed in Sun-synchronous orbits. This orbital configuration ensures constant geometric observation conditions. Several formation options exist, each with distinct advantages and limitations; however, the design of formations is beyond the scope of this document.

The satellites can communicate with each other using so-called *Inter-Satellite Links (ISL)*. These links enable the sharing of memory resources, processing functions, and downlink capabilities between the spacecrafts in orbit. In this SSC, the satellites can assume different roles: The *master* (or leader) dictates the flight formation and the *slaves* (or followers) adjust accordingly. An animation of the conception is shown in Figure 1.

There are two different scenarios regarding the downlink of data. In the first scenario, all satellites are capable of downlinking data independently (all-downlink – AD). In the second scenario, only the master satellite performs the downlink (master-downlink – MD). The MD scenario reflects a more realistic operational constraint, as the ground segment is currently equipped with only a single X-band receiver. However, the AD scenario permits higher image resolution and is therefore more likely in future applications. Consequently, it is also assumed in [18].

In conclusion, we examine the concept of SSCc, where several small, networked satellites cooperate to accomplish more flexible and autonomous Earth observation missions. We focus on a system of three LEO satellites equipped with SAR payloads. They are interconnected with each other through inter-satellite links and operate under different roles (master/slave).

⁴<https://www.safest.dgbtek.com>

⁵<https://www.stormchecker.org>

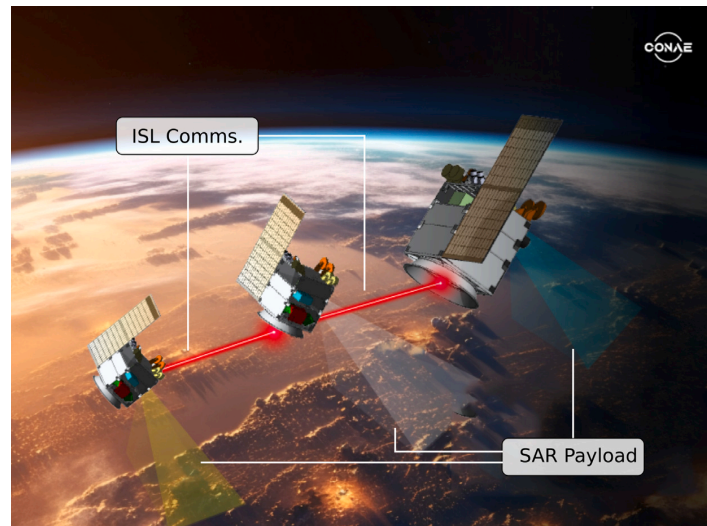


Figure 1: Satellite constellation for Earth observation

4.1 Modelling Principles

Here we give a brief overview of our model, which represents an exemplary system comprising three satellites and two ground stations. Details can be found in [23].

System components The satellites are modelled by four components each. The *Ground-to-Space Link (GSL)* enables communication between individual satellites and the ground stations. The satellites' capability to communicate with each other is modelled by the *Inter-Satellite Link (ISL)*. The *Data Acquisition Sensor (Data)* enables the collection of sensor data that are subsequently transmitted to the ground stations. Various other components such as processors and their software, power, and propulsion systems are summarised as *Core*. Each of the three satellites possesses all of these components, which are susceptible to failures. If a fault occurs within any of the supplementary systems categorised as *Core*, the entire satellite becomes inoperable. On the other hand, a fault in either the ISL, GSL, or Data does not immediately lead to total satellite failure; instead, it may require a reconfiguration of the entire system. The ground segment comprises two redundant *Ground Stations (GS)*, with at least one required to maintain system functionality. The failure of a ground station affects all satellites concurrently, as connections to the failed station are lost for all satellites simultaneously.

Operational configuration To ensure our system is operational, at least one satellite must be controllable and capable of collecting sensor data. *Controllable* means that the satellite has a connection to the ground, i.e., that its GSL system and at least one ground station are available. Here it is worth mentioning that our analysis models the GSL and ground stations as binary components without explicitly considering the orbital geometry or the contact windows inherent to a non-geostationary LEO mission. Implicitly, continuous availability of the ground segment is assumed, limited only by technical failures rather than operational unavailability due to orbital constraints causing the temporary absence of communication windows

Collecting sensor data can either be done by the satellite with a ground connection or via another satellite, provided the corresponding ISL connection is available. For a detailed analysis, we define different roles for the satellites. Each satellite can be either a (unique) *master* or a *slave*. The master is responsible for maintaining the connection with the ground to receive commands, which it can then forward to the slaves. Therefore, the master needs to have a functional GSL. A slave is capable of acquiring sensor data and – in case the of all-downlink scenario – can independently downlink this data. If required, the satellites can also change their roles.

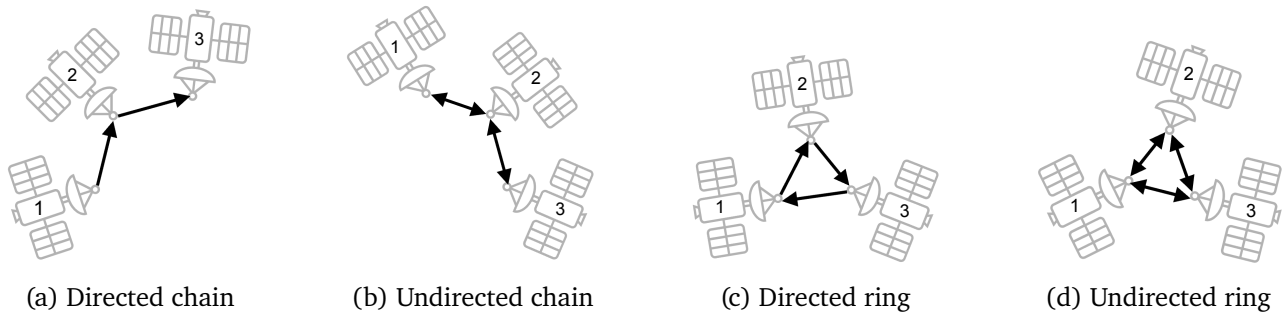


Figure 2: Communication topologies

Communication topologies The communication of the satellites can be defined in multiple ways. To this end, we have defined four distinct communication typologies, which are visualised in Figure 2.

- *Directed chain*: In this configuration, satellite 1 is capable of forwarding commands to satellite 2, which, in turn, can relay commands to satellite 3. However, communication does not occur in the reverse direction.
- *Undirected chain*: This model is similar to the directed chain, but it allows bidirectional communication. In this case, satellite 3 has the additional capability to forward commands back to satellite 2, while satellite 2 can communicate with satellite 1.
- *Directed ring*: The satellites are arranged in a circular configuration, where satellite 1 transmits commands to satellite 2, satellite 2 to satellite 3, and satellite 3 subsequently returns commands to satellite 1.
- *Undirected ring*: This topology also features a circular arrangement, but it permits communication in both directions among the satellites.

For the purposes of this report, we have chosen to adopt the undirected chain model, as it represents the most realistic framework for satellite communication.

Failure behaviour The most straightforward operational configuration involves the master receiving commands, acquiring data independently of the slaves and downlinking them. If the master encounters an issue with its data sensor, the acquisition command can be passed to the next satellite to collect the required sensory data. This arrangement is called *one-hop situation*. If the second satellite also experiences a failure in its data sensor, the command can then be forwarded to the third satellite, which will collect the data and send it to the ground. This scenario is referred to as *two-hop situation*. In the event of a failure in the GSL of the master, the satellite will be unable to fulfil its function, as it will be incapable of receiving commands from the ground. In such a scenario, the function of the master will be taken over by another satellite equipped with a functional GSL. This process is known as *dynamic reconfiguration*. As the concrete master election mechanism (in particular, the order in which slaves are chosen to become a master) is irrelevant for analysing the general failure behaviour, we refrain from stating it explicitly.

4.2 The AADL Model

Based on the assumptions established in the previous section, we provide a sketch of the AADL model of the satellite constellation. Figure 3a shows the first lines of the description of a satellite. The full specification consists of almost 300 lines of code. Again, details can be found in [23].

The initial step is to create a generic satellite composed of three parts: a data sensor, a GSL, and an ISL. The model is illustrated by the diagrams shown in Figure 4. Here, ports are depicted as triangles. The entire system is modelled using boolean data ports, with the colour of each port indicating its state: green for true and red for false. The lines connecting the ports represent their interconnections, and

```

system implementation Satellite.Imp
subcomponents
  dataSensor: system dataSensor.Imp;
  GSL: system GSL.Imp;
  ISL: system ISL.Imp;
  Core: system Core.Imp;
...

```

(a) AADL

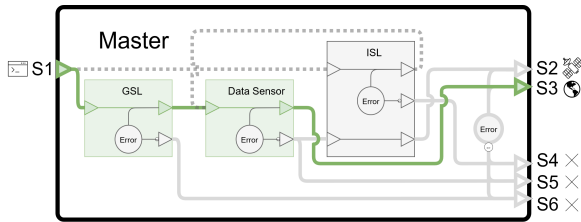
```

part def Satellite {
  part GSL {metadata GSLFailure:FIT1;}
  part Data {metadata DataFailure:FIT1;}
  part ISL {metadata ISLFailure:FIT1;}
  part Core {metadata CoreFailure:FIT1;}
}

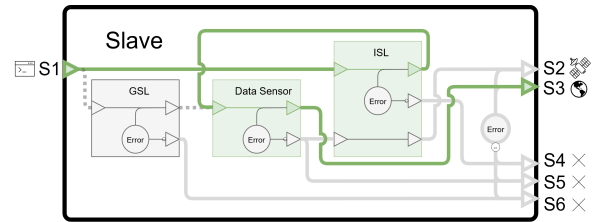
```

(b) SysML

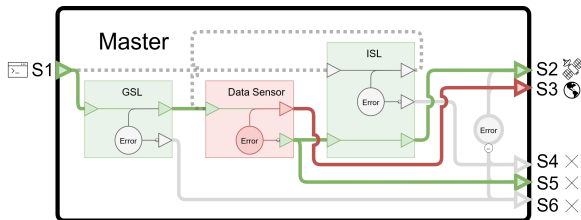
Figure 3: Fragments of satellite specifications



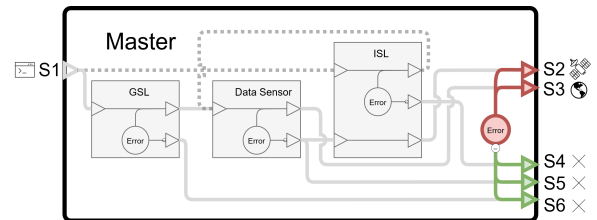
(a) A functional master satellite



(b) A functional slave satellite



(c) Master with defect data sensor



(d) Failure in Core of master

Figure 4: Satellite models

the triangles should be interpreted as arrows to indicate the direction of these connections.

Functional satellites Figure 4a shows an example of a functional satellite acting as a master. The S1 port is true. This indicates that the command to collect data has been sent. Because the satellite is the master, the command originates directly from one of the ground stations. This is the reason why it is forwarded to the GSL component of the satellite. Subsequently, the GSL transmits the command to the data sensor, which acquires the data and downlinks it to the ground. The status of the sending operation is indicated by the outgoing port S3 coloured in green, meaning it is running without failure. The situation is analogous for a slave, as illustrated in Figure 4b. The only difference is that the ISL is used instead of the GSL, because the command originates from another satellite rather than the ground.

Ground segment The ground segment comprises two redundant stations (cf. Figure 5). Port Gr1 is responsible for sending commands to the satellites, while Gr2 is designated for receiving data. Port Gr3 indicates the operational state of the system. This is why Gr3 is employed as the top-level event for fault tree analysis of the system using the COMPASS Toolset.

Satellites with defective data sensor We will now examine cases in which the satellites have defective components. In addition to the ports indicating nominal behaviour, each of a satellite's GSL, ISL, and Data component has a specific outgoing data port which indicates a failure when activated. These ports connect to the satellite's outgoing ports S4 to S6, enabling the system to signal which component is malfunctioning and to initiate necessary reconfigurations. When a defect occurs in a satellite's data sensor, the nominal port of the Data component becomes false, affecting also the S3 port of the satellite

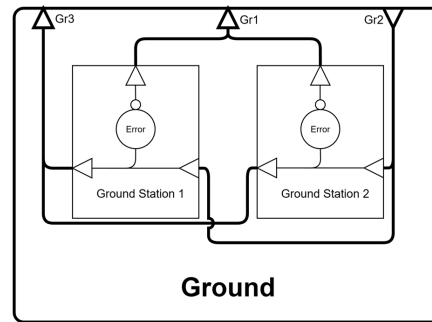


Figure 5: Ground segment with two stations

(see Figure 4c. As a result, data transmission from this particular satellite to the ground is no longer possible. Meanwhile, the error port of Data becomes true, which is connected to the incoming port of the ISL. The ISL then connects to the S2 port and enables forwarding of the command to acquire data to the next satellite. Additionally, the S5 port of the satellite indicates that there is a defect in the data sensor. A defective data sensor of a slave satellite is handled similarly.

Defective Core If a fault occurs in any of the supplementary systems categorised as Core, the entire satellite becomes inoperable. This is modelled by connecting every outgoing port to the Core component. When a failure occurs, ports S2 and S3 turn false, and ports S4–S6 turn true to indicate that none of the satellite’s components can be used any more. This situation is illustrated in Figure 4d, with a similar behaviour defined for a slave satellite.

Satellite connections We will now connect the satellites according the undirected chain communication topology. This means that port S2 of satellite 1 is connected to port S1 of satellite 2, port S2 of satellite 2 is connected to port S1 of satellites 3 and 1, and port S2 is connected to port S1 of satellite 2.

Implementation The SLIM extension of AADL includes mode and state transitions to facilitate the implementation of the different modes used by satellites and their components. Unfortunately, during simulation in the COMPASS Toolset, significant delays occurred in the mode transitions of the three satellites and their components. This made analysing the model and developing a functioning version quite challenging. Hence, we modelled the modes as ports that are activated when the component is in that specific mode. The mode changes are implemented by a controller in our model, which connects all these ports.

Once the modelling process is complete, COMPASS is used to generate a fault tree. A part of this fault tree is shown in Figure 6.

4.3 The SysML Model

The alternative approach uses SysML as the specification language in combination with the SAFEST Tool. Figure 3b on page 9 shows a fragment of the specification, which altogether consists of approx. 220 lines of code. Again, details can be found in [23].

While the overall structure of the (nominal) specification is similar to the AADL case, the main task is to manually add fault tree annotations to the SysML specification, from which SAFEST can later generate a graphical representation.

Modelling data acquisition and ground The first step is to establish a top-level event for the fault tree. In this example, it should be triggered if the overall system is in an Non-Operational Configuration (NOC). Thus, NOC becomes the root node of the overall FT (see Figure 7).

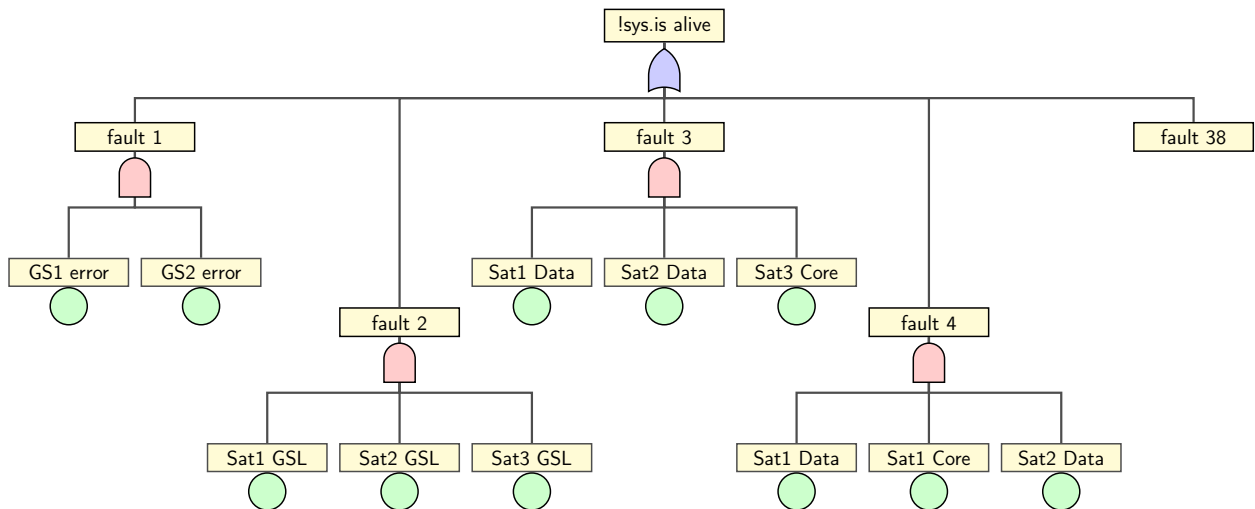


Figure 6: Part of the fault tree generated by COMPASS

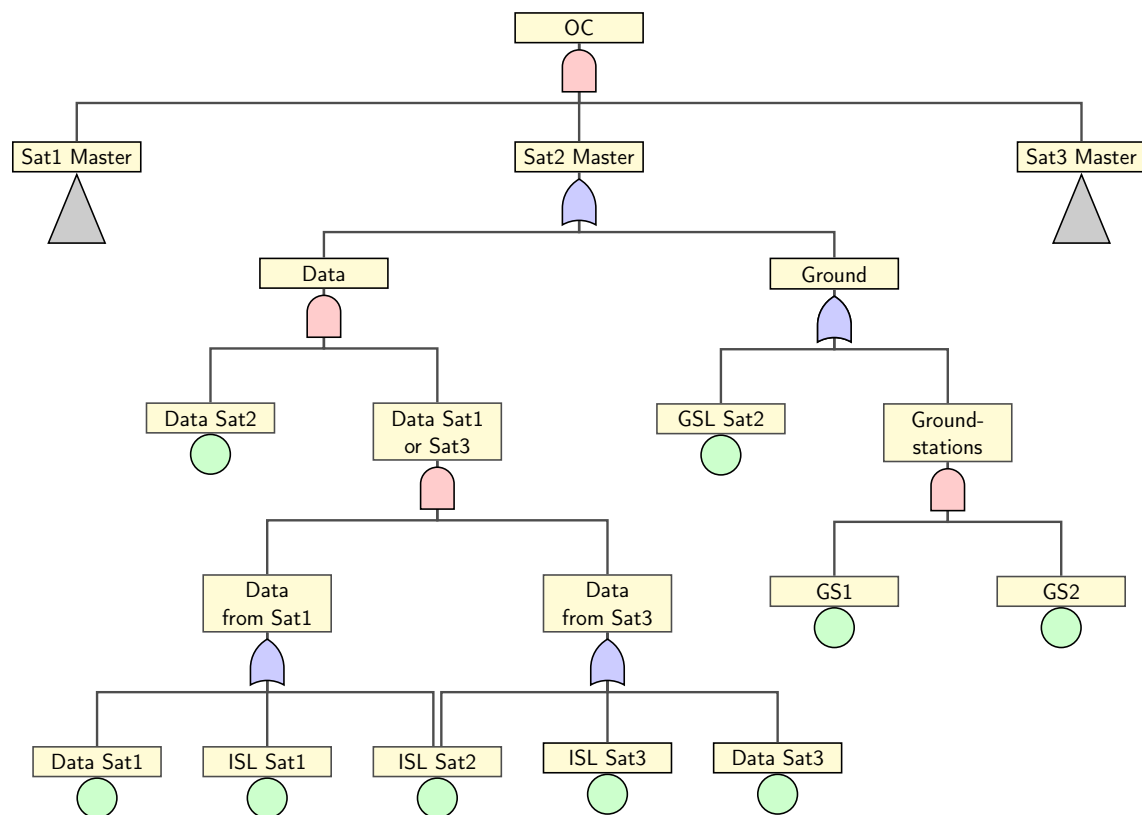


Figure 7: Fragment of fault tree extracted by (with details for Satellite 2, without Core) SAFEST

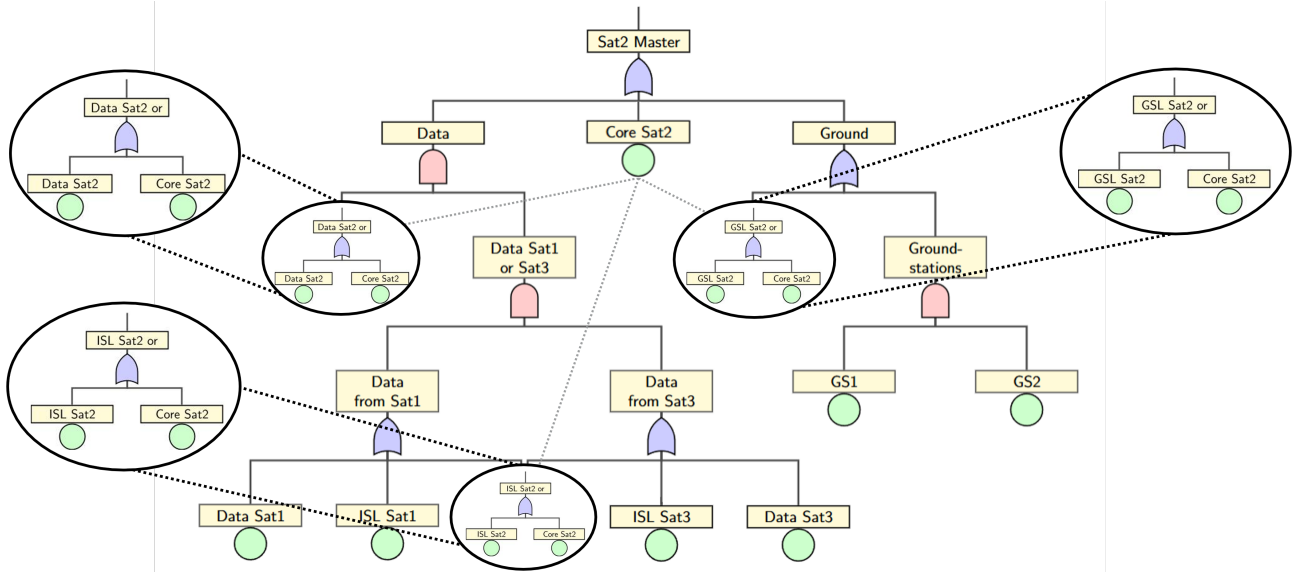


Figure 8: Complete fault tree for satellite 2 as master

The system is modelled from the perspective of the satellites. As explained in Section 4, it is operational if at least one satellite is able to act as a master, given the necessity of at least one satellite to be controllable. This is represented by an AND gate at the root of the fault tree, with child nodes indicating various operational configurations, such as Sat2 Master. A satellite qualifies as a master if it maintains a connection with at least one ground station and can either acquire data independently or via another satellite. Consequently, the nodes Sat1 Master, Sat2 Master (see Figure 7), and Sat3 Master are represented as OR gates connecting the respective sub-trees.

To ensure the master connection to the ground, both a functioning GSL and at least one functional ground station are required, which can again be seen in Figure 7. The master satellite has the capability to acquire data either independently or via another satellite. Initially, a functional data sensor of the master is the only required component. In order to acquire the data via an additional satellite (a slave), it is necessary for both satellites to have functional ISL and additionally for the slave to have a functional data sensor.

Adding the Core component So far, Figure 7 captures only three components of satellite 2: Data, GSL, and ISL (the latter also for the connected satellites). The other components, summarised as Core, are not included. To account for this, we introduce the basic event Core Sat2 directly as a child of node Sat2 Master (see Figure 8). This adjustment is necessary because a failure in any of these components can lead to a total failure of the satellite, rendering it unable to operate as a master. Moreover, it is important to note that Data, GSL, and ISL of satellite 2 do not automatically fail in the fault tree when Core Sat2 fails because the node is just added at the root. However, this is important as those events are referred to by the other satellite models. Therefore, each of the three satellite components is replaced with an OR gate which has the original satellite component and the Core component as children.

Implementation As explained before, the SysML specification must be annotated with safety information. The SAFEST Tool provides two dedicated packages for this purpose. The first, DFTGates, provides all types of gates that can be used to construct fault trees. The second, DFTBEs, defines all basic events that may be used in fault trees and requires the declaration of a failure distribution for each BE. Since no specific failure rates are known in our application, we assigned an exponential distribution.

Satellite 1	Satellite 2	Satellite 3
Core	Core	Core
Data	Data	Data
GSL	GSL	GSL
Core	GSL	GSL
GSL	Core	GSL
GSL	GSL	Core
GSL	Core	Core
Core	GSL	Core
Core	Core	GSL
Core	Data	Data
Data	Core	Data
Data	Data	Core
Data	Core	Core
Core	Data	Core
Core	Core	Data

(a) Standard cuts

Satellite 1	Satellite 2	Satellite 3
GSL ISL	Data	Data
Data	GSL ISL	Data
Data	Data	GSL ISL
ISL Data	GSL	GSL
GSL	ISL Data	GSL
GSL	GSL	ISL Data

(b) Further topology-invariant cuts

Table 1: Min-cut sets for satellite component failures

5 Validation of Fault Tree Technology

As seen before, after modelling the satellite constellation system in AADL and SysML, the resulting fault trees can be generated by COMPASS and SAFEST, respectively (cf. [23] for details). Their visual representation shows notable differences. The FT extracted by the SAFEST Tool from manual annotations to the SysML code presents a hierarchical structure, whereas the model-checking algorithm employed by COMPASS yields a flatter and more horizontally-oriented FT. Also, the number of nodes varies significantly between both. The FT generated by COMPASS has 166 nodes, while the variant extracted by SAFEST system only comprises 44 nodes. This difference arises from COMPASS's approach to directly calculating the min-cut sets (MCS) and presenting them as an FT: Each level-one node, labelled fault 1, fault 2, fault 3, etc. in Figure 6, corresponds to an MCS of the system. MCS can also be computed by the SAFEST Tool by applying standard algorithm to the FT extracted from the specification. We note already here that the MCS calculated by SAFEST are identical to those derived in COMPASS using model checking, i.e., both tools yield consistent results. In the following, we provide an in-depth analysis.

5.1 Analysis of Minimal Cut Sets

Basic analysis The first MCS involves the failure of both ground stations and is the smallest cut. All other MCS are related to satellite components. To give a systematic overview, they are organised as shown in Table 1a with one column for each satellite and each row representing a distinct MCS. The table is organised into groups where the MCS within a group differ only by permutation. We designate one representative for each group, which is highlighted in grey. In the analysis, 15 instances were identified where all satellites are either malfunctioning, unable to collect data, or incapable of connecting with the ground. In such scenarios, achieving an operational configuration is not feasible any more.

There are six additional scenarios in which the system cannot operate regardless of the communication topology used. They are listed in Table 1b. In the first three cases, one satellite is still capable of acquiring data but is unable to connect to the ground (due to a GSL failure) or to receive commands

Directed ring			Undirected ring		
Satellite 1	Satellite 2	Satellite 3	Satellite 1	Satellite 2	Satellite 3
GSL ISL	GSL	Data	GSL ISL	GSL ISL	Data
Data	GSL ISL	GSL	Data	GSL ISL	GSL ISL
GSL	Data	GSL ISL	GSL ISL	Data	GSL ISL
ISL Data	GSL	Data	ISL Data	GSL	ISL Data
Data	ISL Data	GSL	ISL Data	ISL Data	GSL
GSL	Data	ISL Data	GSL	ISL Data	ISL Data

(a) Additional ISL failure in undirected ring

Directed ring			Undirected ring		
Satellite 1	Satellite 2	Satellite 3	Satellite 1	Satellite 2	Satellite 3
GSL	Data	Core			
Core	GSL	Data			
Data	Core	GSL			
GSL ISL	Core	Data	GSL ISL	Core	Data
Data	GSL ISL	Core	GSL ISL	Data	Core
Core	Data	GSL ISL	Data	GSL ISL	Core
			Core	GSL ISL	Data
			Core	Data	GSL ISL
			Data	Core	GSL ISL
ISL Data	GSL	Core	ISL Data	GSL	Core
Core	ISL Data	GSL	ISL Data	Core	GSL
GSL	Core	ISL Data	Core	ISL Data	Core
			GSL	Core	ISL Data
			Core	GSL	ISL Data

(b) Further different min cuts

Table 2: Comparison of min cuts for directed and undirected ring

from the other satellites (due to an ISL failure). In the second group, only one satellite can establish a ground connection (with its functional GSL), but it cannot forward commands to another satellite because of an ISL failure.

Taking communication topology into account To demonstrate the influence of the communication topology on the failure behaviour, we compare the MCS for different types of communication, beginning with the directed and undirected ring (please refer to Section 4.1 for a description of communication topologies). Table 2 illustrates the resulting differences. In the case of the undirected ring, an additional failure in the ISL is necessary for the entire system to fail (see Table 2a). This is because communication can be forwarded in both directions in this topology.

Even in the cases shown in Table 2b, the failure of an ISL component is necessary for an overall system failure. But now there are two potential satellites where an ISL must fail, so there are two different scenarios. They are represented by the same background colour. This clarifies the reason behind the observed difference in the number of MCS: 37 in the case of undirected ring versus 40 for directed ring. Next, we will examine MCS of the undirected chain, which was chosen as the communication topology of Section 4. Beyond the scenarios illustrated in Table 1a and 1b, there are additional 16 MCS for the undirected chain presented, as shown in Table 3. Specifically, ten of these cases arise due to the failure

Undirected Chain		
Satellite 1	Satellite 2	Satellite 3
GSL	Rest	Data
Data	Rest	GSL
GSL	ISL GSL	Data
Data	ISL GSL	GSL
GSL	ISL Data	Data
Data	ISL Data	GSL
Data	ISL GSL	Rest
Rest	ISL GSL	Data
GSL	ISL Data	Rest
Rest	ISL Data	GSL

(a) Min cuts with defect ISL of Sat2

Undirected Chain		
Satellite 1	Satellite 2	Satellite 3
GSL ISL	Data	Rest
Rest	Data	GSL ISL
ISL Data	GSL	Rest
Rest	GSL	ISL Data
GSL ISL	Data	GSL ISL
ISL Data	GSL	ISL Data

(b) Further cases

Table 3: Additional min cuts for undirected chain

	Satellite 1	Satellite 2	Satellite 3
GSL	13	14	13
ISL	6	10	6
Data	13	14	13
Core	11	9	11
Sum	43	47	43

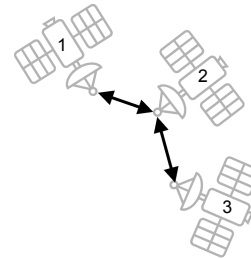


Table 4: Frequency of components in min-cut sets (left) for undirected chain (right)

of the ISL component in the middle satellite (see Table 3a). Unfortunately, the other two satellites are not fully operational on their own, making it necessary to forward commands through satellite 2. However, since this is not feasible, the configuration is rendered non-operational. The first six cases are solely related to the types of communication employed. The remaining four cases can also be identified for the undirected ring. The same applies to the six additional cases shown in Table 3b.

Criticality analysis A common analysis question is how critical a component failure is for the overall system reliability. As we do not have quantitative information about the failure rates of specific components, we follow a “qualitative” approach based on min-cut sets to address this issue. More exactly, we have created a statistics about the frequency of each component of every satellite occurring in the MCS. The results are shown in Table 4 for the undirected chain topology. The data indicates that – not surprisingly – satellite 2 plays a crucial role in this type of communication, along with the GSL and data sensor component for the satellites in general.

5.2 Discussion

Our results show that qualitative analyses of the fault trees generated from manually designed system specifications are feasible and yield interesting results, especially with regard to the occurrence frequency of individual components within the MCS (Table 4). However, additional information such as probabilistic failure rates of a system’s basic components is required in order to enable a full quantitative analysis of reliability metrics and to derive more precise conclusions about the satellite system design.

The frameworks COMPASS and SAFEST exhibit different strengths and weaknesses. The former is a dated tool, but it allows for the automatic and exhaustive identification of failure scenarios from a formal architectural model, which is especially useful in early design stages. The representation of the FT is less intuitive in structure, but the MCS can be read directly from the tree. The operation of the tool is often cumbersome, due to outdated software and the implementation as a virtual machine and due to scalability limitations as system complexity increases.

SAFEST is more modern, but it does not (yet) offer a fully automatic generation of FTs from the system specification. Rather, it extracts a graphical representation of FTs from user-given annotations in the SysML specification. Thus, it offers greater control over the abstraction level by requiring manual definition of the fault tree, favouring computational scalability at the cost of a reduced ability to discover scenarios that are not explicitly modelled. Although SAFEST generally offers a more intuitive handling, two major drawbacks remain. The first is the generic definition of satellite components. SAFEST currently generates only one BE per component type. To model each satellite component individually, all components must therefore be given unique names. Consequently, generating FTs in SAFEST requires repeated copy-and-paste, which is prone to human error. The second drawback is the missing export function for MCS in a machine-readable format. This limitation prevents further automated processing. According to the developers, this function will be introduced in a future version of the tool.

In summary, the approaches followed by COMPASS and SAFEST should be considered complementary and applied based on design maturity and specific analysis objectives.

6 Efficient Fault Tree Analysis

The analysis of a static fault tree is commonly performed by translating it into a binary decision diagram (BDD) and calculating the relevant metrics on this BDD [29]. This yields compact representations enabling the analysis of large systems. Ongoing improvements in BDD tools such as parallelisation [35] enable further progress.

The flexibility and increased expressiveness of dynamic fault trees, though, requires more involved analysis methods. BDDs cannot represent DFTs directly as the latter consider failure sequences instead of Boolean combinations. A common method is to translate DFTs into Markov models [38].

However, for large DFTs, this approach can suffer from the problem of state space explosion. A possible mitigation strategy is the exploitation of independent sub-trees in a DFT, also called *modules*. Fault tree modules allow for hybrid analysis in which different modules are analysed with the techniques best suited for them: static subtrees are handled via BDD-based techniques and dynamic sub-trees via state-space based techniques. In the following, we sketch two algorithms and associated implementations that exploit this idea.

6.1 Automated Isolation of Static Parts in Dynamic Fault Trees

In [19], Gulati and Dugan proposed a modular DFT analysis approach combining several analysis techniques. It divides the DFT into independent sub-parts which are analysed individually. Here, Markov models are used for dynamic parts and BDDs for static parts.

The idea of this modularisation is depicted in Figure 9. First, dynamic modules, i.e., sub-trees containing dynamic elements, are identified (the blue boxes in the leftmost tree). Each dynamic module is analysed independently with state-of-the-art analysis techniques for Markov models. Afterwards, each such module is replaced by a single basic event which represents the corresponding failure probabilities (tree in the middle). The remaining (static) fault tree is then translated into a BDD (rightmost part) and is analysed by BDD techniques. Modularisation works especially well when the dynamic parts are located in the bottom region of the fault tree and the static parts are on the top levels. As dynamic

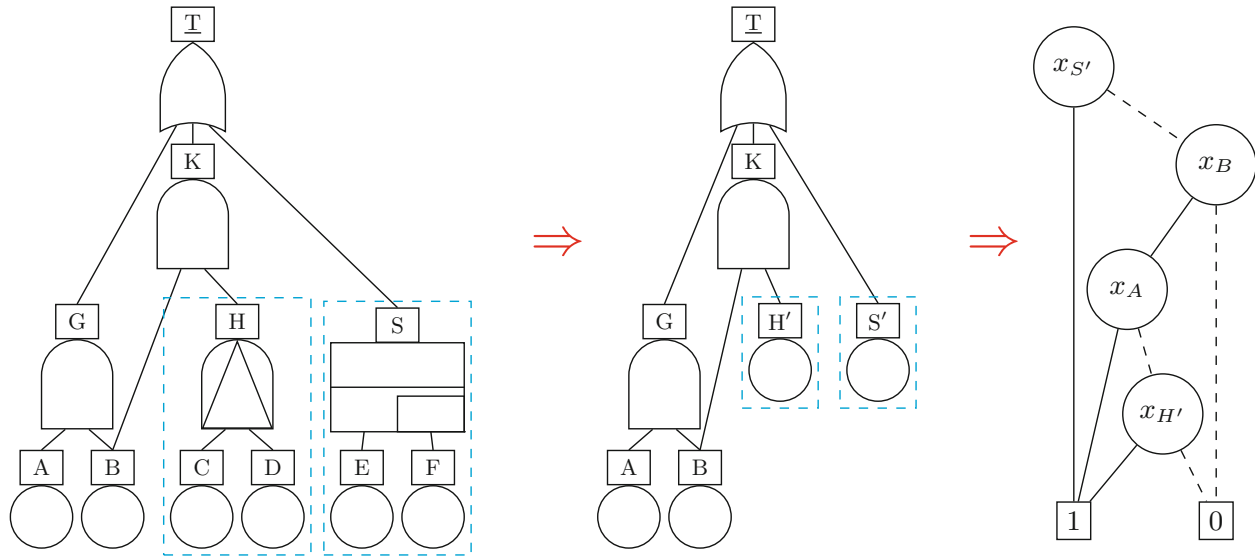


Figure 9: Modularisation of dynamic fault trees

parts usually model single components of a system, this structure is present in many DFTs.

This approach was first implemented in the DIFtree tool [13] and its successor Galileo [34], which both are not developed further. The paper [3] introduces a highly competitive implementation of the modularisation technique in the STORM-DFT component of the STORM tool, combining the multi-core BDD library Sylvan [35] and the efficient DFT analysis via Markov models [38]. As noted before, the STORM tool is employed as the underlying engine for FTA in the SAFEST framework and was thus also used for obtaining the results presented in Section 5.

Current activities are concentrated on improvements to obtain smaller BDDs during the translation of static sub-trees, for example by improving the heuristics for variable orderings employed in the BDDs. As the modularisation method cannot be fully exploited if large dynamic modules are present, another possible research direction is to approximate the results for sub-modules, either by smaller fault trees or by the approximation approach of [38].

6.2 Modular Criticality Analysis

Besides analysing the reliability of a system, a typical measure of interest is the criticality of components [15]. Here, we want to answer the questions “how critical is the failure of a specific component for the overall system reliability?” and “which components are the most important ones for system reliability?”. A common criticality metric is the Birnbaum importance index [5]. However, its computation requires the availability of failure rates for individual components, which are not known for the case study presented in Section 4. In Section 5.1, we have therefore shown a basic “qualitative” criticality analysis based on the frequency of component occurrences in minimal cut sets of fault trees.

If failure distributions are known, then a more exact quantitative analysis is possible, again by essentially exploring the state space of a probabilistic model such as a Markov chain. And again, modularisation techniques can be employed to mitigate state-space explosion.

The paper [28] presents a modular criticality analysis which exploits modules at arbitrary levels in the fault tree. The idea is to replace irrelevant modules – i.e., those that do not contain the component(s) of interest – by single basic events capturing the failure behaviour of the complete module. Based on this modular calculation of the failure probability, the Birnbaum importance index is computed. This approach is implemented in the SAFEST Tool for fault tree analysis [39], which was also used for the

computation of minimal cut sets as shown in Section 5.1. The experimental evaluation in [28] shows that modular criticality analysis drastically decreases the memory consumption and therefore allows to tackle large DFTs that were previously out of reach.

Current activities focus on improving the efficiency of computing criticality values for all DFT components and for multiple time bounds. Moreover, an extension of the modular analysis to other criticality metrics such as the Diagnostic Importance Factor, Risk Achievement Worth and Risk Reduction Worth [36] is planned.

7 Enhancements of the Fault Tree Approach

To broaden the context of our work, we briefly describe two enhancements of the fault tree approach that have been elaborated in close cooperation between MISSION project partners.

7.1 Attack-Fault-Defense Trees

Complex cyber-physical systems often exhibit a dynamic and intricate interaction between security, safety, and defence aspects. To support integrated safety–security reasoning for such systems, Attack–Fault–Defence Trees (AFDT) have been developed as a unified tree-based model that combines (i) fault propagation, (ii) attacker actions, and (iii) defensive measures [31]. A main contribution of this work is a precise formal semantics for AFDTs together with metrics based on cut sets, enabling systematic qualitative analysis (e.g., by identifying minimal combinations of adverse events that lead to an undesired top-level event). Feasibility of the approach is demonstrated in a smart-grid/charging-station setting, showing how cut-set analysis supports structured risk assessment across safety and security viewpoints. Altogether, [31] provides an analysable risk-modelling formalism that aligns very well with MISSION’s model-based dependability goals.

This methodology is further investigated in subsequent work carried out in cooperation between UT and ASC, where the AFDT approach is applied to satellite mission contexts, targeting risk mitigation strategies where operational faults and adversarial actions must be assessed jointly [30]. Here, mission-relevant failures, attacks, and defences are modelled in a single AFDT structure, supporting qualitative analysis via minimal cut sets and enabling traceable mitigation reasoning (e.g., how adding or removing defence measures changes the set of critical scenarios). The case study is framed around satellite ground-segment operational settings, highlighting security-dominant scenarios with many defence measures and complex dependencies.

7.2 Statistical Model Checking

The work described in [32] considers spare-part provisioning as a quantitative optimisation problem, balancing non-availability of a system against inventory cost. The approach combines reliability modelling (via fault trees) with statistical model checking, encoding the spare-management decision problem as a stochastic priced timed game automaton and applying the UPPAAL Stratego Tool⁶ to synthesize an optimal policy (e.g., an optimal number of spares). The evaluation was done in close cooperation between UT and INVAP and uses an emergency shutdown setting from a research reactor context. The results demonstrate that the method can compute cost-effective spare levels while maintaining very high expected availability under the model assumptions. It thus provides a reusable formal-method workflow for availability/cost trade-offs that transfers to space-system logistics and maintainability reasoning.

⁶<https://uppaal.org/features/#span-classscuppaalspan-stratego>

This spare-management line of work is consolidated and extended in [33] by refining the formal modelling/analysis pipeline, and by discussing practical aspects of applying statistical model checking to high-dependability systems (including considerations regarding rare events). Using the same core idea – reliability modelling by fault trees mapped to a stochastic automata model solved with UPPAAL Stratego –, it reports optimisation results for single- and multi-subsystem scenarios and further positions the method as a general approach for maintainability/availability planning under cost constraints.

8 Conclusion

In this report, we have first described two approaches for the automated generation of fault tree models from full-fledged system specifications of fault-tolerant space architectures. Next, we have evaluated those on a case study introduced in previous work, a swarm satellite constellation designed for an Earth observation mission. Our results show that qualitative analyses of the fault trees generated from those specifications are feasible and yield interesting results. However, additional information such as probabilistic failure rates of a system's basic components is required in order to enable a full quantitative analysis of RAMS metrics such as Mean Time To Failure (MTTF), and to derive more precise conclusions about the satellite system design. As a starting point, (approximate) rate values obtained from literature or space reliability databases could be used to compare architectural alternatives in a more informed manner. Moreover, we have discussed two modularisation approaches that aim to improve the scalability of analysis algorithms by automatically isolating static parts in dynamic fault trees. This enables the combination of efficient BDD-based analysis of static fault trees with state-space based techniques for the dynamic parts.

References

- [1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.
- [2] J. Anjali, S. Vestal, and P. Binns. Automatic generation of static fault trees from AADL models. In *DSN Workshop on Architecting Dependable Systems*, 2007. URL: <https://hdl.handle.net/11299/217313>.
- [3] D. Basgöze, M. Volk, J.-P. Katoen, S. Khan, and M. Stoelinga. BDDs strike back: Efficient analysis of static and dynamic fault trees. In *NASA Formal Methods*, pages 713–732. Springer, 2022. doi:10.1007/978-3-031-06773-0_38.
- [4] A. Berres and H. Schumann. Automatic generation of fault trees: A survey on methods and approaches. In L. Walls, M. Revie, and T. Bedford, editors, *Risk, Reliability and Safety: Innovating Theory and Practice*, chapter Systems Reliability Models and Applications. CRC Press, 2016. doi:10.1201/9781315374987.
- [5] Z. W. Birnbaum. On the importance of different components in a multicomponent system. Technical Report TR-54, University of Washington, USA, 1968. doi:10.21236/AD0670563.
- [6] M. Bozzano, H. Brientjes, A. Cimatti, J.-P. Katoen, T. Noll, and S. Tonetta. COMPASS 3.0. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 379–385. Springer, 2019. doi:10.1007/978-3-030-17462-0_25.
- [7] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri. Safety, dependability and performance analysis of extended AADL models. *The Computer Journal*, 54(5):754–775, 2011. doi:10.1093/comjnl/bxq024.

- [8] M. Bozzano, A. Cimatti, C. Mattarei, and S. Tonetta. Formal safety assessment via contract-based design. In *Automated Technology for Verification and Analysis*, pages 81–97. Springer, 2014. doi:S10.1007/978-3-319-11936-6_7.
- [9] M. Bozzano, A. Cimatti, and F. Tapparo. Symbolic fault tree analysis for reactive systems. In *Automated Technology for Verification and Analysis*, pages 162–176. Springer, 2007. doi:10.1007/978-3-540-75596-8_13.
- [10] M. Bozzano and A. Villaflorita. The FSAP/NuSMV-SA safety analysis platform. *Software Tools for Technology Transfer*, 9:5–24, 2007. doi:10.1007/s10009-006-0001-2.
- [11] M. Bozzano and A. Villaflorita. *Design and Safety Assessment of Critical Systems*. CRC Press, 2010. doi:10.1201/b10094.
- [12] D. Coppit, K. Sullivan, and J. Dugan. Formal semantics of models for computational engineering: a case study on dynamic fault trees. In *Int. Symposium on Software Reliability Engineering*, pages 270–282. IEEE, 2000. doi:10.1109/ISSRE.2000.885878.
- [13] J. Dugan, B. Venkataraman, and R. Gulati. DIFtree: a software package for the analysis of dynamic fault tree models. In *Reliability and Maintainability Symposium*, pages 64–70. IEEE, 1997. doi:10.1109/RAMS.1997.571666.
- [14] J. B. Dugan, S. J. Bavuso, and M. A. Boyd. Fault trees and sequence dependencies. In *Reliability and Maintainability Symposium*, pages 286–293. IEEE, 1990. doi:10.1109/ARMS.1990.67971.
- [15] Y. Dutuit and A. Rauzy. Efficient algorithms to assess component and gate importance in fault tree analysis. *Reliability Engineering & System Safety*, 72(2):213–222, 2001. doi:10.1016/S0951-8320(01)00004-7.
- [16] M.-A. Esteve, J.-P. Katoen, V. Y. Nguyen, B. Postma, and Y. Yushtein. Formal correctness, safety, dependability and performance analysis of a satellite. In *Int. Conference on Software Engineering*, pages 1022–1031. ACM and IEEE, 2012. doi:10.1109/ICSE.2012.6227118.
- [17] D. Farias, B. Nogueira, I. F. Júnior, and E. Andrade. A modeling-based approach for dependability analysis of a constellation of satellites. *Software and Systems Modeling*, 24:209–224, 2025. doi:10.1007/s10270-024-01197-7.
- [18] J. A. Fraire, S. Henn, G. Stock, R. Ohs, H. Hermanns, F. Walter, L. Van Broock, G. Ruffini, F. Machado, P. Serratti, and J. Relloso. Quantitative analysis of segmented satellite network architectures: A maritime surveillance case study. *Computer Networks*, 255:110874, 2024. doi:10.1016/j.comnet.2024.110874.
- [19] R. Gulati and J. Dugan. A modular approach for analyzing static and dynamic fault trees. In *Reliability and Maintainability Symposium*, pages 57–63. IEEE, 1997. doi:10.1109/RAMS.1997.571665.
- [20] International Electrotechnical Commission, Geneva, Switzerland. *Fault Tree Analysis (FTA)*, 2006. International Standard IEC 61025.
- [21] S. Junges, D. Guck, J.-P. Katoen, and M. Stoelinga. Uncovering dynamic fault trees. In *Int. Conference on Dependable Systems and Networks*, pages 299–310. IEEE, 2016. doi:10.1109/DSN.2016.35.
- [22] S. Junges, J.-P. Katoen, M. Stoelinga, and M. Volk. One net fits all. In *Application and Theory of Petri Nets and Concurrency*, pages 272–293. Springer, 2018. doi:10.1007/978-3-319-91268-4_14.

- [23] M. Metzler. Evaluating tool support for fault tree analysis of satellite constellations. Bachelor's thesis, RWTH Aachen University, Aachen, Germany, 2025. doi:[10.18154/RWTH-2025-09682](https://doi.org/10.18154/RWTH-2025-09682).
- [24] F. Mhenni, N. Nguyen, and J.-Y. Choley. Automatic fault tree generation from SysML system models. In *Int. Conference on Advanced Intelligent Mechatronics*, pages 715–720. IEEE, 2014. doi:[10.1109/AIM.2014.6878163](https://doi.org/10.1109/AIM.2014.6878163).
- [25] F. Mhenni, N. Nguyen, and J.-Y. Choley. SafeSysE: A safety analysis integration in systems engineering approach. *IEEE Systems Journal*, 12(1):161–172, 2018. doi:[10.1109/JSYST.2016.2547460](https://doi.org/10.1109/JSYST.2016.2547460).
- [26] T. Noll, M. Volk, and J.-P. Katoen. Inventory and specification: Extended fault trees. Technical report, MISSION Consortium, 2023. Deliverable D2.2. URL: <https://mission-project.eu/deliverables/d22-inventory-extended-fault-trees.pdf>.
- [27] E. Ruijters and M. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15–16:29–62, 2015. doi:[10.1016/j.cosrev.2015.03.001](https://doi.org/10.1016/j.cosrev.2015.03.001).
- [28] F. Sher, M. Stoelinga, and M. Volk. Modular criticality analysis for dynamic fault trees. In *Principles of Verification: Cycling the Probabilistic Landscape, Part III*, pages 274–293. Springer, 2025. doi:[10.1007/978-3-031-75778-5_13](https://doi.org/10.1007/978-3-031-75778-5_13).
- [29] R. Sinnamon and J. Andrews. Fault tree analysis and binary decision diagrams. In *Reliability and Maintainability Symposium*, pages 215–222. IEEE, 1996. doi:[10.1109/RAMS.1996.500665](https://doi.org/10.1109/RAMS.1996.500665).
- [30] R. Soltani, P. Diale, M. Lopuhaä-Zwakenberg, and M. Stoelinga. Safety and security risk mitigation in satellite missions via attack-fault-defense trees. In *European Safety and Reliability & Society for Risk Analysis Europe Conference*. arXiv, 2025. doi:[10.48550/arXiv.2504.00988](https://doi.org/10.48550/arXiv.2504.00988).
- [31] R. Soltani, M. Lopuhaä-Zwakenberg, and M. Stoelinga. Safety-security analysis via attack-fault-defense trees: Semantics and cut set metrics. In *Int. Conference on Computer Safety, Reliability, and Security*, pages 218–232. Springer, 2024. doi:[10.1007/978-3-031-68606-1_14](https://doi.org/10.1007/978-3-031-68606-1_14).
- [32] R. Soltani, M. Volk, L. Diamonte, M. Lopuhaä-Zwakenberg, and M. Stoelinga. Optimal spare management via statistical model checking: A case study in research reactors. In *Int. Conference on Formal Methods for Industrial Critical Systems*, pages 205–223. Springer, 2023. doi:[10.1007/978-3-031-43681-9_12](https://doi.org/10.1007/978-3-031-43681-9_12).
- [33] R. Soltani, M. Volk, L. Diamonte, M. Lopuhaä-Zwakenberg, and M. Stoelinga. Optimal spare management via statistical model checking: A case study in research reactors. *Software Tools for Technology Transfer*, 27:361–376, 2025. doi:[10.1007/s10009-025-00791-4](https://doi.org/10.1007/s10009-025-00791-4).
- [34] K. Sullivan, J. Dugan, and D. Coppit. The Galileo fault tree analysis tool. In *Int. Symposium on Fault-Tolerant Computing*, pages 232–235. IEEE, 1999. doi:[10.1109/FTCS.1999.781056](https://doi.org/10.1109/FTCS.1999.781056).
- [35] T. van Dijk and J. van de Pol. Sylvan: multi-core framework for decision diagrams. *Software Tools for Technology Transfer*, 19:675–696, 2017. doi:[10.1007/s10009-016-0433-2](https://doi.org/10.1007/s10009-016-0433-2).
- [36] W. Vesely, T. Davis, R. S. Denning, and N. Saltos. Measures of risk importance and their applications. Technical Report NUREGICR-3385, Battelle Columbus Labs, USA, 1983. URL: <https://inis.iaea.org/records/qbrmf-1w519>.
- [37] M. Volk. *Dynamic Fault Trees: Semantics, Analysis and Applications*. PhD thesis, RWTH Aachen University, Aachen, Germany, 2022.

- [38] M. Volk, S. Junges, and J.-P. Katoen. Fast dynamic fault tree analysis by model checking techniques. *IEEE Transactions on Industrial Informatics*, 14(1):370–379, 2018. doi:[10.1109/TII.2017.2710316](https://doi.org/10.1109/TII.2017.2710316).
- [39] M. Volk, F. Sher, J.-P. Katoen, and M. Stoelinga. SAFEST: Fault tree analysis via probabilistic model checking. In *Reliability and Maintainability Symposium*, pages 1–7. IEEE, 2024. doi:[10.1109/RAMS51492.2024.10457719](https://doi.org/10.1109/RAMS51492.2024.10457719).
- [40] M. Volk, M. Stoelinga, and L. A. Jimenez-Roa. Training: Fault trees from data and RES. Technical report, MISSION Consortium, 2022. Deliverable D2.1. URL: <https://mission-project.eu/deliverables/d21-training-fault-trees.pdf>.
- [41] Z. Zhao, R. Li, and T. Meng. A MBSE-based fault modeling approach for satellite. *Journal of Physics: Conference Series*, 2977(1):012077, 2025. doi:[10.1088/1742-6596/2977/1/012077](https://doi.org/10.1088/1742-6596/2977/1/012077).