Bachelor Thesis

# Evaluating Tool Support for Fault Tree Analysis of Satellite Constellations

submitted to

Lehrstuhl Informatik 2
“Software Modeling and Verification”
Fakultät für Mathematik, Informatik und Naturwissenschaften
RWTH Aachen University

by

**Mareike Metzler**

Supervision:

Prof. Dr. Thomas Noll
Lehrstuhl Informatik 2, RWTH Aachen University

| **First examiner** | **Second examiner** |
|---|---|
| Prof. Dr. Thomas Noll<br>Lehrstuhl Informatik 2<br>RWTH Aachen | Prof. Dr. Ir. Dr. h. c. Joost-Pieter Katoen<br>Lehrstuhl Informatik 2<br>RWTH Aachen |

Communicated by Prof. Dr. Thomas Noll

September 9, 2025

# Abstract

To monitor Argentina's maritime areas for illegal activities such as unauthorised fishing, satellite-based approaches have been proposed. Previous research has examined aspects such as coverage of the observed area, communication dynamics, data handling strategies, and the fault behaviour of individual satellites. However, the fault behaviour of multiple satellites, so-called satellite constellations, has not yet been systematically analysed. This thesis addresses the gap by modelling an exemplary satellite system consisting of three satellites using the modelling languages AADL and SysML. Fault trees are then automatically generated from these models using the COMPASS and SAFEST tools and are subsequently analysed. The thesis demonstrates that automatic fault tree generation using COMPASS is feasible for satellite constellation, allowing for the evaluation of design questions. However, it also addresses the limitations of the current tool environment. While COMPASS is functional, it is outdated. Additionally, SAFEST primarily extracts fault trees from manually made annotations in the system specification rather than generating them automatically.

# Contents

**Abstract** iii

**List of Figures** v

**List of Tables** vi

**1 Introduction** 1

**2 Background** 2
- 2.1 Safety Analysis . . . 2
- 2.2 Fault Tree Analysis . . . 2
- 2.3 Description of Used Tools . . . 3
  - 2.3.1 COMPASS and AADL . . . 3
  - 2.3.2 SAFEST and SysML . . . 4
- 2.4 Satellite Constellations . . . 4

**3 Related Work** 6
- 3.1 Automatic Fault Tree Generation for Satellites . . . 6
- 3.2 Automatic Fault Tree Generation Using SysML . . . 6
- 3.3 Automatic Fault Tree Generation Using AADL . . . 7
  - 3.3.1 Previous Work with COMPASS . . . 7

**4 Concept and Approach** 8
- 4.1 Concept . . . 8
  - 4.1.1 System Components . . . 8
  - 4.1.2 Operational Configuration . . . 8
  - 4.1.3 Data Sensor and GSL Failure . . . 9
  - 4.1.4 Communication Typologies . . . 9
- 4.2 SysML Model . . . 10
- 4.3 AADL Model . . . 14
  - 4.3.1 Functional Satellites . . . 14
  - 4.3.2 Satellites with Defective Components . . . 15

**5 Implementation** 18
- 5.1 SysML and SAFEST . . . 18
- 5.2 AADL and COMPASS . . . 18

**6 Fault Tree Analysis** 20
- 6.1 Analysis of Min Cuts . . . 20
- 6.2 Comparison of Fault Trees for Different Typologies of Satellite Communication . . . 21
  - 6.2.1 Fault Tree Analysis for Undirected Chain . . . 22
- 6.3 Discussion . . . 23
  - 6.3.1 Tool Comparison . . . 23

**7 Future Work** **24**
7.1 Extension of System Specification . . . . . . . . . . . . . . . . . . . . . . . . . 24
7.2 Improvement of Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
7.3 Background Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

**8 Conclusion** **26**

**A Appendix** **29**

# List of Figures

2.1 Static gates . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
2.2 Dynamic gates . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

4.1 Communication typologies . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
4.2 Operational configuration as root of the FT and defined as TLE in SysML . 10
4.3 OC with Sat 1, 2 or 3 as master . . . . . . . . . . . . . . . . . . . . . . . . . 11
4.4 Satellites modelled as OR gates with data acquisition and ground connection 11
4.5 Partial FT for ground communication of Sat2 . . . . . . . . . . . . . . . . . 11
4.6 Data acquisition of Sat2 either independently or via another satellite . . . . 12
4.7 Partial FT of Sat 2 without Rest . . . . . . . . . . . . . . . . . . . . . . . . 12
4.8 Final partial FT for satellite 2 as master . . . . . . . . . . . . . . . . . . . . 13
4.9 Satellite components with port numbers . . . . . . . . . . . . . . . . . . . . 14
4.10 Example of a functional satellite in the role as a master . . . . . . . . . . . 14
4.11 Example of a functional satellite in the role of a slave . . . . . . . . . . . . 15
4.12 Master with defect Data Sensor . . . . . . . . . . . . . . . . . . . . . . . . . 15
4.13 Slave with defect Data Sensor . . . . . . . . . . . . . . . . . . . . . . . . . . 16
4.14 Failure in Rest of master satellite . . . . . . . . . . . . . . . . . . . . . . . . 16
4.15 Ground with two ground stations . . . . . . . . . . . . . . . . . . . . . . . . 16
4.16 Part of generated FT from COMPASS . . . . . . . . . . . . . . . . . . . . . 17

# List of Tables

6.1 15 standard min cuts for satellite component failures . . . . . . . . . . . . . 20
6.2 Further typology invariant min cuts . . . . . . . . . . . . . . . . . . . . . . . . 21
6.3 Comparison of min cuts from directed and undirected circle . . . . . . . . . . 21
6.4 Additional min cuts for undirected chain . . . . . . . . . . . . . . . . . . . . . 22
6.5 Frequency of components in min cut sets for undirected chain . . . . . . . . 22

# 1 Introduction

Monitoring maritime areas has become a strategic priority for many coastal states. Argentina, with its vast and remote maritime zones, faces significant challenges in detecting illegal activities at sea. Satellite systems offer a powerful means of addressing these challenges because they can provide continuous surveillance over wide areas. The use of multiple satellites, known as satellite constellations, enhances this capability [1].

Research in this field has already examined topics such as communication between satellites and ground stations, strategies for data handling, and coverage of observation areas. Furthermore, studies have analysed the fault behaviour of individual satellites [1, 2]. However, the fault behaviour of satellite constellations, in which several satellites cooperate and reconfigure their roles in the case of failures, has not yet been systematically investigated.

Fault Tree Analysis (FTA) is a widely used method to analyse fault behaviour. Engineers used to construct fault trees (FTs) manually, which required specialised knowledge and considerable effort. Designing manually can be especially challenging when dealing with complex systems, particularly those made up of multiple components and featuring dynamic behaviour [3]. In the case of satellite constellations, where satellites can take over functions from others or adapt their configuration, manually creating fault trees is not only inefficient but also prone to human errors.

To address these limitations, new approaches have been proposed that generate fault trees automatically from formal system specifications. Tools such as COMPASS and, recently, SAFEST aim to support this process. They can extract fault behaviour directly from models written in established modelling languages such as the Architecture Analysis and Design Language (AADL) and Systems Modeling Language (SysML). This automation promises better scalability, consistency, and refinement when analysing complex systems.

This thesis investigates whether the modelling languages AADL and SysML can be used to describe a satellite constellation system and whether fault trees can then be generated automatically using COMPASS and SAFEST. It also examines whether the results provide useful insights into the reliability of satellite constellations. This thesis extends previous work on individual satellites to an integrated constellation.

The key findings of this thesis are twofold. First, it demonstrates that automatic fault tree generation in COMPASS is feasible for a satellite constellation. From these generated trees, system designers can derive important design questions, for example, concerning redundancy strategies of specific components. Second, it assesses the limitations of the current tool environment. COMPASS is functional but outdated, and SAFEST does not yet support automatic generation; instead, it extracts fault trees from manually made annotations in the system specification.

# 2 Background

## 2.1 Safety Analysis

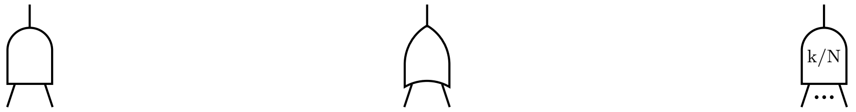
Safety-critical systems are systems whose failures may lead to injury or death of people, environmental damage, or economic loss. The complexity of such systems is steadily increasing due to growing functionality. The primary objective of safety analysis techniques is to identify risks and hazards and to eliminate or reduce the likelihood of failure. When carried out manually, safety analysis is time-consuming and prone to human error, as it depends heavily on the engineer's ability to predict system behaviour. With the rising complexity of modern systems, the limits of manual analysis become evident, which motivates the use of formal techniques to support and improve safety assessments. Typical examples are Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA). FMEA employs an inductive approach, examining the causes of hazards and tracing them to identify potential safety problems. In contrast, FTA employs a deductive approach that initiates with the unintended behaviour of a system and systematically works backwards to identify the underlying causes [4]. In this thesis, we will use FTA.

## 2.2 Fault Tree Analysis

FTA is a well-established method for analysing risks in safety and economically critical systems. Typical application areas include transportation, energy infrastructure, space exploration, and medical technology [3].

The method is based on FTs, which are directed acyclic graphs with a unique root node. They illustrate how component failures propagate and eventually lead to a system failure. The leaves of the tree represent component failures, while logical gates model the propagation of failures through the system. Events can be classified as basic events (BE), which occur spontaneously, or intermediate events, which are triggered by one or more other events. The Top-Level-Event (TLE), which is the root of the FT, represents the system failure that is investigated by tracing the failures of subsystems. Common gate types are AND, OR, and k/N gates. Pictures are shown in Figure 2.1.

| | |
|---|---|
| AND gate: | Output event occurs if all input events have occurred. |
| OR gate: | Output event occurs if any input event has occurred. |
| k/N gate: | Output event occurs if at least k of N input events have occurred. |

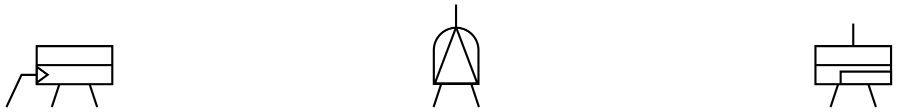


(a) AND gate (b) OR gate (c) k/N gate

Figure 2.1: Static gates

In addition to Static Fault Trees (SFT), Dynamic Fault Trees (DFT) have been developed. DFTs extend the classical approach by considering the temporal order of component failures by introducing additional gates such as the FDEP, SPARE and PAND gate as shown in Figure 2.2.

| | |
|---|---|
| FDEP gate: | (Functional DEPendency) triggers multiple dependent events when the initiating event on the left occurs. |
| PAND gate: | (Priority AND) Output event occurs if all inputs occur from left to right. |
| SPARE gate: | primary component can be replaced by other spares. |



(a) FDEP gate (b) PAND gate (c) SPARE gate

Figure 2.2: Dynamic gates

The FTA consists of four distinct stages: defining the system, constructing the FT, and conducting qualitative and quantitative evaluations. During the system description phase, the system, its intended function, and potential failures are outlined. From this description, the structure of the FT is developed. Events that could result in undesirable system behaviour, potentially leading to system failure, are then identified [5].
After the FT is created, **qualitative** and **quantitative** analyses can be performed.
**Qualitative** FTA focuses on the structure of the tree. One typical method is to compute min cut sets, which are minimal sets of BE whose simultaneous failure leads to the failure of the entire system. Small min cut sets may indicate system vulnerability, as a failure of a few components could lead to overall system failure.
**Quantitative** FTA, on the other hand, computes probabilities under the assumption that probabilistic failure rates of the system components are known. This allows for the calculation of reliability measures such as system reliability, system availability and mean time to failure (MTTF) [3].

## 2.3 Description of Used Tools

For this thesis, the primary focus of examination was put on tools that automatically generate and extract FTs from system specifications using SysML and AADL as a modelling language. Additionally, these tools enable subsequent analysis of the generated FTs.

### 2.3.1 COMPASS and AADL

The COMPASS toolset[1] (COrrectness, Modeling and Performance of AeroSpace Systems) is a model-based framework designed to support the co-engineering of aerospace systems.

---

[1] https://compass.fbk.eu

Its goal is to enable systematic analysis of correctness, reliability and performance from the early design stages, following a holistic approach. COMPASS employs the System-Level Integrated Modelling (SLIM) language, which extends the Architecture Analysis and Design Language (AADL). SLIM allows detailed modelling 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. COMPASS integrates established hazard analysis methods such as FMEA and FTA. By combining these methods with formal verification techniques, it enables systematic exploration of both nominal and faulty system behaviour. The toolset relies on advanced verification engines, including NuSMV, FSAP, Sigref, and MRMC. It has been developed by the Software Modeling and Verification Group at RWTH Aachen University, in collaboration with the European Space Agency (ESA) and the Fondazione Bruno Kessler in Trento, Italy [6, 7]. For this thesis, we are using the COMPASS 3.0 version.

### 2.3.2 SAFEST and SysML

SysML (Systems Modeling Language) is a modelling language developed by the Object Management Group (OMG) in 2006 [8]. 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 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. For about thirteen years, SysML has been widely used across engineering disciplines and has been the subject of numerous research studies [9]. For this thesis we use the SysML 2.0 version.

The SAFEST (Static/Dynamic Fault Tress & Reward Event Trees Analysis Tool)[2] 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. It applies probabilistic model checking, like COMPASS, 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 [10]. For this thesis, we use the SAFEST 2.0.0 version, as SAFEST 3.0.0 was published several months after the thesis began.

## 2.4 Satellite Constellations

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. An example is the hypothetical SARE

---

[2] https://www.safest.dgbtek.com

(Sistema Argentino de Alta REvisita) mission, which aims to monitor Argentina's Exclusive Economic Zone (EEZ) in order to detect illegal maritime activities.
The case study considered in this thesis 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 a consistent illumination and shadow profile during successive passes. Several formation options exist, each with distinct advantages and limitations; however, the design of formations is beyond the scope of this thesis.
The satellites can communicate with each other using so-called Inter-Satellite Link. These links enable the exchange of memory resources, processing functions, and downlink capabilities between the spacecrafts in orbit.
In this SSC the satellites have different roles. In this thesis, we use the words master and slave however leader and follower are also used in the literature. The master, for example, dictates the flight formation and the slaves adjust accordingly.
There are two different scenarios for 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 probable in future applications. Consequently, it is also employed in this study [1].
In conclusion, we examine the concept of Swarm Satellite Constellations (SSC), where multiple small, networked satellites cooperate to achieve 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).

# 3 Related Work

This chapter provides an overview of various approaches and tools for the automatic generation of fault trees. We will begin by discussing fault tree generation specifically for satellites. Subsequently, we will examine studies that have employed SysML or AADL as their modelling languages.

## 3.1 Automatic Fault Tree Generation for Satellites

In [2], Zhao et al. address the increasing complexity of modern space-based infrastructures. The paper proposes an Model-Based Systems Engineering (MBSE) approach for satellite fault modelling and analysis. This method uses SysML as the modelling language. It leverages SysML's ability to represent system information from multiple viewpoints. Additionally, it supports breaking down systems into levels of system, subsystem, and component. To extend this capability, the Risk Analysis and Assessment Modeling Language (RAAML) is employed. RAAML enriches SysML elements with fault-specific semantics and enables a uniform representation of potential failure modes. Implemented in the commercial software Cameo Systems Modeler[1], the approach combines the satellite design model with the satellite fault model to establish data model relationships across levels. This integration enables the systematic identification of failure modes and their interconnections, facilitating the analysis of fault propagation across various levels of the system architecture. This approach could also be used for FTA.
Unfortunately, there is a scarcity of publicly available research papers focusing on fault analysis for satellites due to confidentiality. For this reason, we will delve into automatic fault tree generation more broadly, focusing particularly on system specifications that utilise AADL and SysML, as these are the languages we have used.

A comprehensive overview can be found in the paper by Berres and Schumann [5]. The paper differentiates between fault tree generations based on either system description or system behaviour. The system description specifies the static hardware and software architecture of the system, its functions, and the potential failures that may arise. This description must be analysed during the generation phase, which requires using a formal language to define the system. We utilise SysML and AADL for this purpose.
System behaviour can be characterised by states, state transitions, and the related events. The goal is to describe behaviours that could lead to system failures. With this description in hand, corresponding fault trees can be constructed.

## 3.2 Automatic Fault Tree Generation Using SysML

An approach that utilises both generation types is presented by Mhenni et al. [11, 12]. They propose a method for automatically generating safety analysis artefacts, specifically FMEA table and FTs, directly from SysML system models. In this method, system

---

[1] https://docs.nomagic.com/spaces/NMDOC/pages/218759282/2024x+Refresh3+Version+News

components and their interactions are modelled using Internal Block Diagrams (IBDs), where flow ports capture the communication between subsystems. The structural model is then translated into a directed multigraph. By applying graph traversal algorithms and identifying characteristic structural patterns, generic fault trees with corresponding logic gates and events are automatically generated. This approach has been implemented within the SafeSysE methodology [12]. Beyond fault tree generation based on system description, SafeSysE also incorporates behavioural safety analysis (BSA) through model checking, a formal verification of whether system dynamics satisfy safety requirements expressed in temporal logic.

## 3.3 Automatic Fault Tree Generation Using AADL

Joshi et al. [13] present an approach for the automatic generation of static fault trees from AADL models. In this method, the system architecture is described using AADL and enriched with failure annotations through the Error Annex, an extension of the language for modelling faults and errors. Based on this annotated model, fault trees are generated in three steps: first, the system is represented as a graph in which nodes correspond to components and edges to their connections; second, component failures defined in the error model are mapped onto the graph; and third, a recursive graph algorithm derives fault trees while breaking potential cycles by tracking already visited components. The resulting structures are then processed by the commercial tool CAFTA to produce formal fault trees and derive the corresponding logical failure relationships. Mhenni criticises that there are limitations in failure modes because certain ones are not captured by the AADL error model and must be handled separately. Furthermore, the cycle-breaking strategy is relatively simplistic, as criticised by Mhenni in [11].

There is a lot more research on automatic fault tree generation, but most of it uses commercial tools or does not publish their source code. That is why we used the COMPASS and SAFEST tools for this thesis.

### 3.3.1 Previous Work with COMPASS

Previous studies have already employed the COMPASS toolset for spacecraft modelling. For example, a case study to evaluate the toolset can be found in [14, 15]. The study modelled the satellite as a composition of payload and platform subsystems. While the payload is mission-specific and developed from scratch, the platform relies on design heritage and supports long-term functionality in orbit. In this studies, the development of the platform is examined. The analysis in COMPASS addressed multiple layers of faults within the satellite. Unfortunately, the model is not publicly available due to confidentiality.

# 4 Concept and Approach

## 4.1 Concept

After discussing related work, this chapter presents the modelling concepts of the satellite constellations system. It begins with an overview of the general modelling approach for the constellation described in section 2.4. Then, it moves on to a concrete implementation in AADL and SysML. The approach also incorporates elements developed during the workshop held in March 2025 at INVAP[1]. This workshop served as an important foundation for the models.

### 4.1.1 System Components

We begin by modelling a generic satellite that comprises four essential components: **Ground-to-Space Link** (GSL), **Inter-Satellite Link** (ISL), **Data Acquisition** (Data Sensor) and **Rest**.

- **GSL**: enables communication between individual satellites and the ground.
- **ISL**: models the satellites' capability to communicate with each other.
- **Data Sensor**: enables the collection of sensor data and its transmission to the ground.
- **Rest**: summarises various other components such as software, power, and propulsion systems

Each of the three satellites possesses all of these components. It is crucial to acknowledge that all these components are susceptible to failures. If a fault occurs within any of the supplementary systems categorised as Rest, the entire satellite becomes inoperable. On the other hand, a failure in either the ISL, GSL, or Data Sensor does not immediately lead to total satellite failure; instead, it may initially require a reconfiguration of the entire system.

The **ground** segment includes two redundant ground stations, with at least one needed 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.

### 4.1.2 Operational Configuration

To ensure our system is operational, at least one satellite must be **controllable** and capable of **acquiring sensor data**.

- **Controllable** means that the satellite has a connection to the ground.
- **Acquiring sensor data** can be done either independently by the satellite with a ground connection or via another satellite.

---

[1] https://mission-project.eu/2025/03/08/workshop-invap/

For this purpose, we are defining different roles for the satellites. Each satellite can be either a **master** or a **slave**.

- The **master** is responsible for maintaining a connection with the ground to receive commands, which it then can forward to the other satellites. Therefore, the master needs to have a functional GSL.
- A **slave** is capable of acquiring sensory data and can independently downlink this data. Our system always operates with one master, while all other satellites serve as slaves.

### 4.1.3 Data Sensor and GSL Failure

The most straightforward operational configuration involves the master receiving commands, acquiring data independently and downlink them.

If the master encounters an issue with its Data Sensor, the command can be passed to the next satellite to collect the required sensory data. This arrangement is called **1-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 a **2-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 assumed by another satellite equipped with a functional GSL. This process is known as **dynamic reconfiguration**. The order in which the next satellite is chosen depends on the initial assumptions, but this detail is irrelevant to this thesis.

### 4.1.4 Communication Typologies

The communication of the satellites can be defined in multiple ways. To this end, we have defined four distinct communication typologies. A graphical representation can be seen in Figure 4.1:

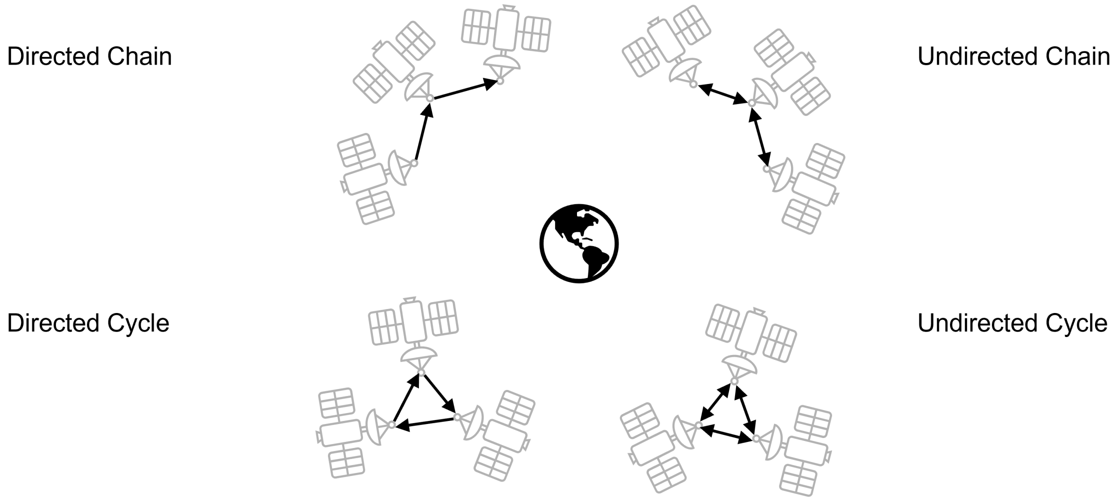


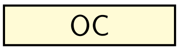
Figure 4.1: Communication typologies

- **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 capability to forward commands back to satellite 2, while satellite 2 can communicate with satellite 1.
- **Directed Cycle**: 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 Cycle**: This typology also features a circular arrangement, but it permits communication in both directions among the satellites.

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

## 4.2 SysML Model

Based on the assumptions established in the preceding chapters, the system of satellite constellations was modelled in SysML. The corresponding code is provided in the Appendix (see section A.1.1). We begin by defining a generic satellite (refer to lines 4-17 in Appendix A.1.1). The satellite comprises four components outlined in section 4.1.1. Each of these components is associated with a specific probabilistic failure rate. For more information about the annotation with safety information, see section 5.1. A central part of the modelling activity was the manual construction of a Fault Tree (FT) in SysML. The SAFEST tool is then generating a graphical view of the FT. In the graphical representation are the satellite components, the basic events (BE) of the FT. After the generic satellite is defined, the next step is to establish a Top-Level-Event (TLE) for the FT. The Top-Level-Event metadata is used; see 4.2b. In this example, we choose "the system is operational" as the TLE. This means that the system has an operational configuration (OC). The decision establishes the root of the FT, as depicted in Figure 4.2a.



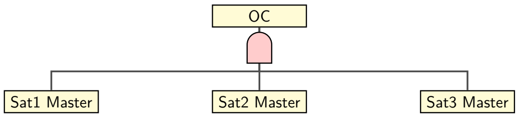
(a) Graphical representation

```
metadata TLE:TOP_LEVEL about OC;
```

(b) Implementation in SysML

Figure 4.2: Operational configuration as root of the FT and defined as TLE in SysML

The system is modelled from the perspective of the satellites, focusing on their ability to act as master satellites. The overall system is considered operational if at least one satellite can successfully assume the role of a master, because at least one satellite must be controllable (see section 4.1.2). This is represented by an AND gate positioned at the root of the FT in Figure 4.3. The child nodes within this framework represent the different operational configurations, such as the scenario where satellite 1 functions as the master: `Sat1 Master`.



(a) Graphical representation

```
metadata OC:AND about
    Sat1Master,
    Sat2Master,
    Sat3Master;
```

(b) Implementation in SysML

Figure 4.3: OC with Sat 1, 2 or 3 as master

Each 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` and `Sat3 Master`) are represented as OR gates (refer to Figure 4.4). These gates fail if any of their inputs fail, which happens when a satellite either loses its connection to the ground or its capability to acquire data. The implementation can be seen in lines 154-156, 182-184 and 211-213 of the code in the Appendix A.1.1.

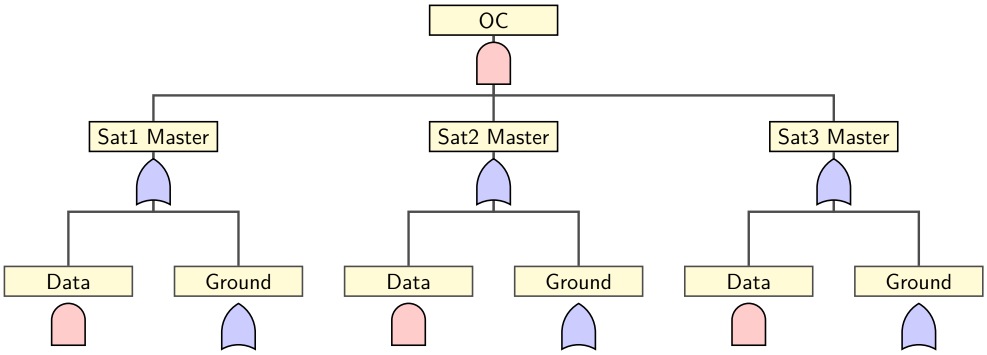
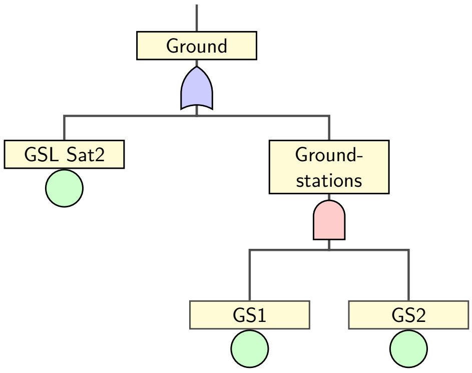


Figure 4.4: Satellites modelled as OR gates with data acquisition and ground connection

To ensure the master operates, it requires both a functioning GSL and at least one functional ground station. This is why the `Ground` node is modelled as an OR gate, as shown at the top of Figure 4.5a. In scenarios where satellite 2 takes on the role of the master, the first child event represents the BE `GSL Sat2`. The other child event, `Groundstations`, utilises an AND gate, since it is sufficient for only one ground station to be operational. Figure 4.5 depicts this aspect of the model for satellite 2.



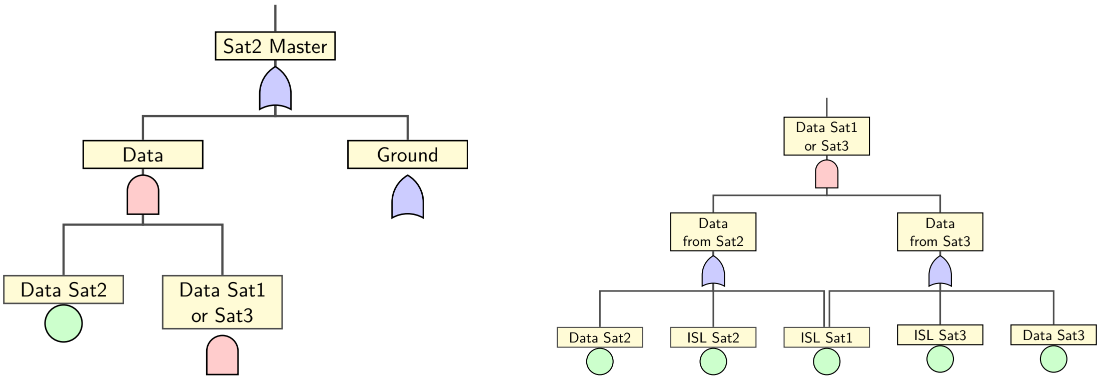
(a) Graphical representation

```
metadata GroundSat2:OR about
    basic::sat2::GSL::GSLFailureSat2,
    Groundstations;
metadata Groundstations:AND about
    basic::GS1::FailureGS1,
    basic::GS2::FailureGS2;
```

(b) Implementation in SysML

Figure 4.5: Partial FT for ground communication of Sat2

A master has the capability to acquire data either independently or through another satellite. This is why the `Data` node is represented as an AND gate. One of its child nodes is the BE `Data Sat1`, which stands for the Data Sensor of satellite 2 and represents the case where satellite 2 is acquiring data independently, see BE of FT in Figure 4.6a.



(a) Data acquisition of Sat2 (b) Data acquisition via Sat1 or Sat3

Figure 4.6: Data acquisition of Sat2 either independently or via another satellite

The alternative approach is to collect data via an additional satellite (a slave). For satellite 2, this means that it can either acquire data through satellite 1 or satellite 3, which is depicted as an AND gate of the node `Data Sat1 and Sat3`, see Figure 4.6a and root in Figure 4.6b. To enable the data acquisition via satellite 1, an ISL is required between satellite 1 and satellite 2. So that is the reason why the ISL components of both satellites need to be functional along with the Data Sensor of satellite 1, so there are child nodes of `Data from Sat1`, see Figure 4.6b. The same principle applies when satellite 2 seeks to obtain data from satellite 3.
Figure 4.7 illustrates the partial FT described so far for satellite 2 as master.

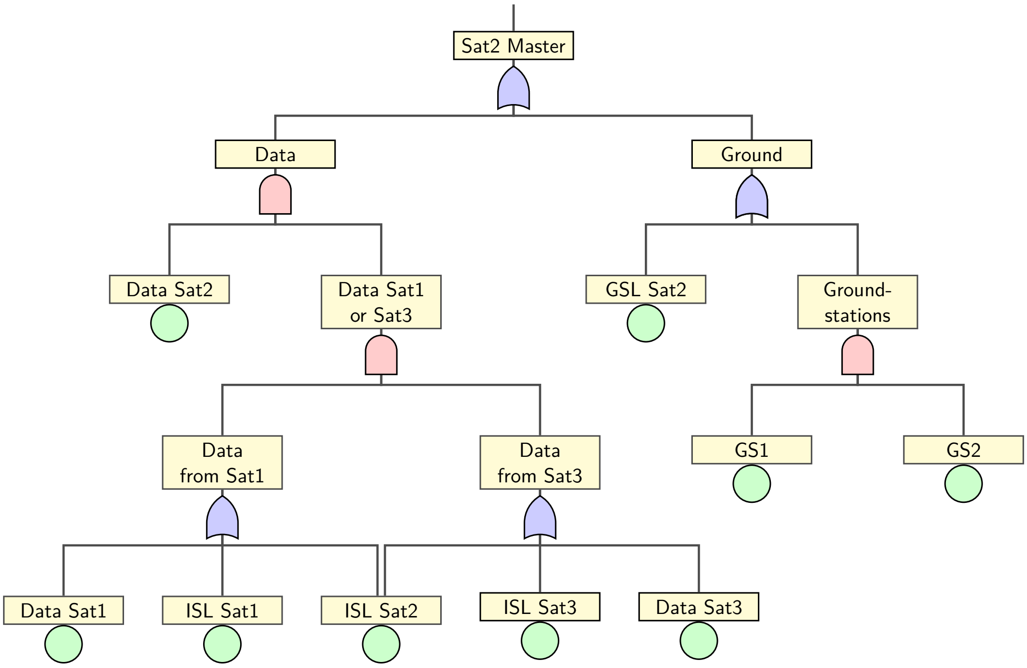


Figure 4.7: Partial FT of Sat 2 without Rest

In Figure 4.7, we see that only three components of satellite 2 are represented: GSL, ISL, and Data. However, the other components, collectively referred to as Rest, are not included. To account for this, we introduce the BE `Rest Sat2` directly as a child of the node `Sat2 Master` (see Figure 4.8). 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. It is important to note that the Data, GSL, and ISL of the satellite do not automatically fail when `Rest Sat2` fails in the FT because the node is just added at the root. To effectively address this issue, we propose implementing a functional dependency gate (FDEP), with the `Rest Sat2` node serving as the trigger and the `GSL Sat2`, `ISL Sat2`, and `Data Sat2` functioning as dependent nodes. Since FDEP is dynamic, it would ordinarily create a dynamic fault tree. However, for our analysis, we prefer to work with a static FT. Therefore, we will replace the FDEP gate with an OR gate, which includes the original BE along with `Rest Sat2` as a child. Figure 4.8 illustrates the final partial FT for satellite 2 in its role as a master. The substitutions for satellite 2 are depicted in the circles of Figure 4.8. To maintain clarity, it has been necessary to add some BE multiple times in the FT. It is therefore important to note that if the names of the nodes are exactly the same, then they refer to the same BE. Lines 94-128 of the code in the Appendix A.1.1 provide a detailed representation of all these substitutions.

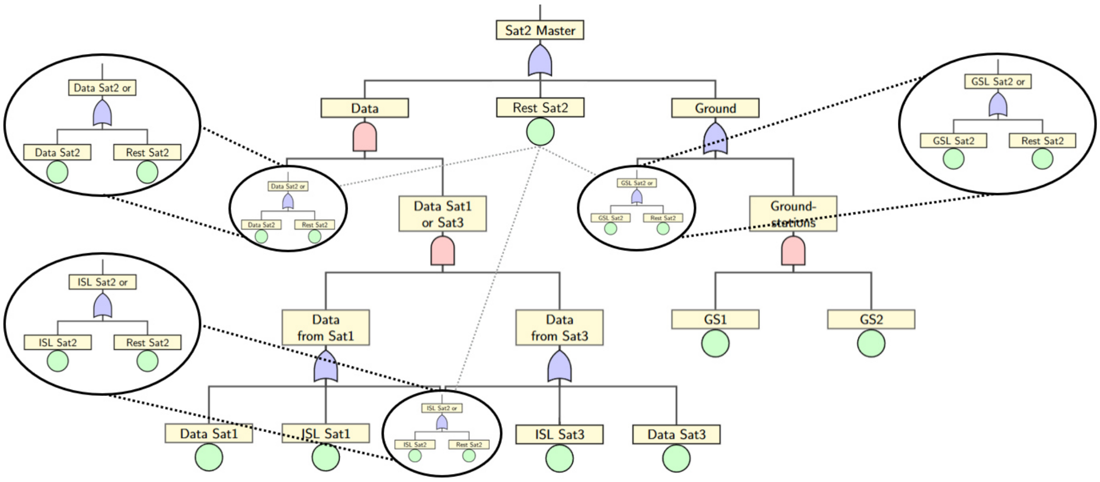
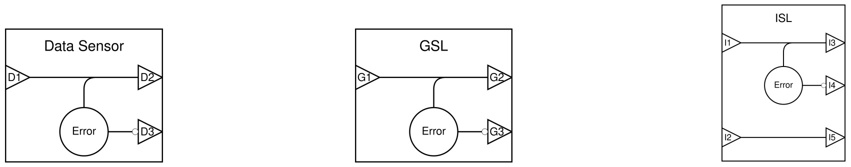


Figure 4.8: Final partial FT for satellite 2 as master

The FT generation was performed for satellite 2 as the master. The structure of the partial FTs for the other satellites closely resembles the FT shown in Figure 4.8, with the main difference being the numbering of the components and the process of data acquisition through other satellites (for satellite 2, see Figure 4.6b). The method of data acquisition via other satellites is dependent on the communication topology used; in this study, we employed a directed chain topology. For example, if satellite 1 acts as the master and aims to gather sensor data through satellite 3, all ISLs involving satellites 1, 2, and 3 need to be functional. As a result, the appearance of the partial FTs changes based on the communication topology. The final FT, which encompasses all satellites and their interdependencies, is depicted in the Appendix A.1.2.

## 4.3 AADL Model

An alternative approach is to analyse the system using the COMPASS toolset. To undertake the analysis, it is first necessary to establish an AADL model with the SLIM extension. As in the SysML Model, the initial step is to create a generic satellite composed of three parts: a Data Sensor, a Ground-to-Space Link (GSL) and an Inter-Satellite Link (ISL).



(a) Data Sensor (b) GSL (c) ISL

Figure 4.9: Satellite components with port numbers

### 4.3.1 Functional Satellites

The model is illustrated in the diagrams. The complete code of the AADL model can be found in Appendix A.2.1. In the diagrams, 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 the triangles should be interpreted as arrows to indicate the direction of these connections. The satellite components with their port numbers are shown in Figure 4.9. To make the diagrams cleaner and easier to read, the port numbers have been removed later on.

Satellites can function as either masters or slaves, with the utilised connections differing between these two modes. However, all satellites are equipped with the necessary connections to operate in either role. Connections that are not active in a specific mode are represented by dotted lines, while grey lines signify connections that hold negligible importance in this particular example.

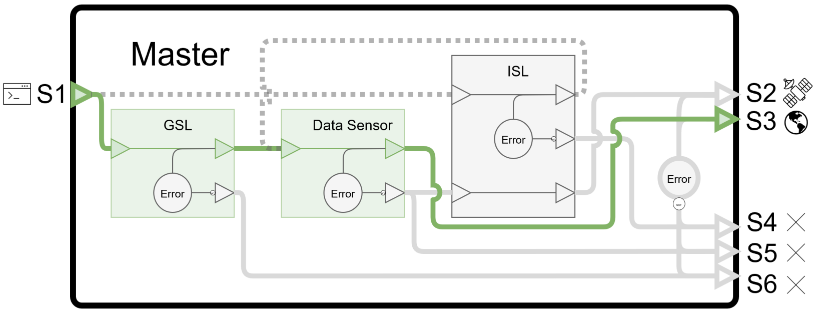


Figure 4.10: Example of a functional satellite in the role as a master

Figure 4.10 shows an example of a functional satellite in the role of a master. The satellite port S1 is true. That shows the command to collect data. Because it is a master, the command comes directly from one of the ground stations. That is the reason why it is forwarded to the GSL component of the satellite. The GSL then passes the command to the Data Sensor which acquires the data and sends it to the ground. The sending process is shown by the true outgoing port S3.

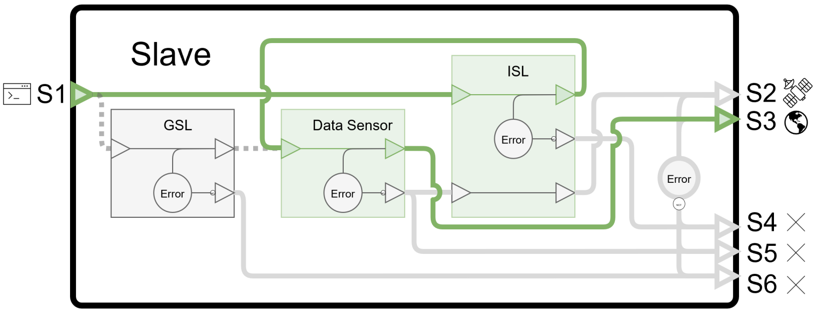


Figure 4.11: Example of a functional satellite in the role of a slave

The situation is quite similar for a slave, as illustrated in Figure 4.11. However, in this scenario, the S1 port is linked to the ISL because the command originates from another satellite rather than the ground. The ISL then forwards the command to the Data Sensor which acquires the data and sends it down to Earth. This process is also represented by the true outgoing port S3.

### 4.3.2 Satellites with Defective Components

We will now examine cases in which the satellites have defective components. Each component of a satellite has a specific outgoing data port (G3, D3, I4) that indicates a failure in the satellite when activated. These ports connect to the satellite's outgoing ports (S4-S6), allowing the system to signal which component is malfunctioning and enabling necessary reconfigurations. More details on this will follow.

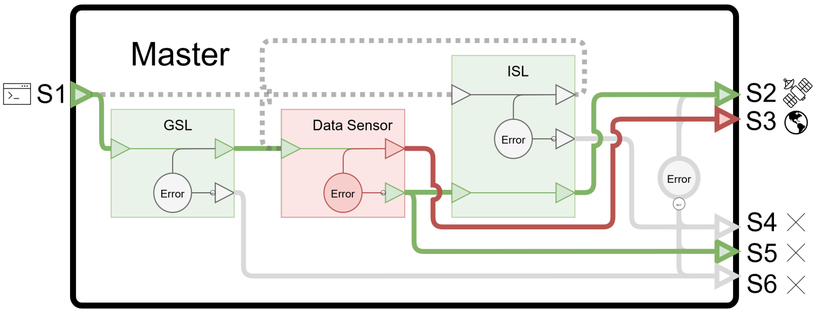


Figure 4.12: Master with defect Data Sensor

When a defect occurs in the master satellite's data sensor, the D2 port of the data sensor becomes false, affecting also the S3 port of the satellite, see Figure 4.12. As a result, data transmission from this particular satellite to the ground is no longer possible. Meanwhile, the port D3 of the Data Sensor becomes true, which is connected to the I2 port of the ISL. The ISL then connects to the S2 port and enables forwarding 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.
The same process occurs with a slave satellite, as shown in Figure 4.13.

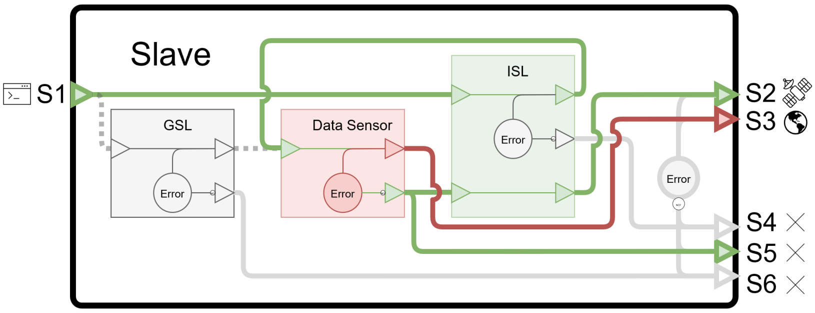


Figure 4.13: Slave with defect Data Sensor

If a fault occurs in any of the supplementary systems categorised as Rest, the entire satellite becomes inoperable. In AADL, it is modelled so that every outgoing port connects to the Rest component. When a failure occurs, ports S2 and S3 turn false, and the ports S4-S6 turn true to indicate that none of the satellite's components can be used anymore. This situation is illustrated in Figure 4.14. The same situation occurs with a slave satellite.

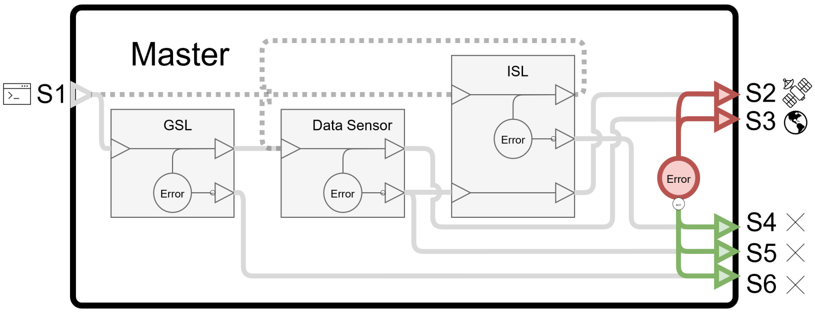


Figure 4.14: Failure in Rest of master satellite

The ground comprises two redundant stations (refer to Figure 4.15). The port Gr1 is responsible for sending commands to the satellites, while Gr2 is designated for receiving data. If Gr3 is true, it indicates that the system is operational. This is why Gr3 is utilised as a Top-Level-Event for further analysis of the system within the COMPASS toolset. The definition of the Top-Level-Event can be seen in the property file line 8 in Appendix A.2.2.

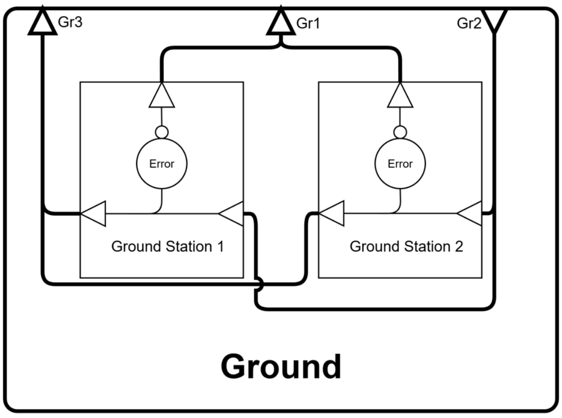


Figure 4.15: Ground with two ground stations

We will now connect the satellites into the desired communication typology. In this context, we are employing the undirected chain. The connections between the satellites are specified in lines 158, 160, and 162 of the code provided in Appendix A.2.1.
After the modelling process is complete, COMPASS generates an FT. A part of the generated FT is illustrated in Figure 4.16. The full FT is in Appendix A.2.3.

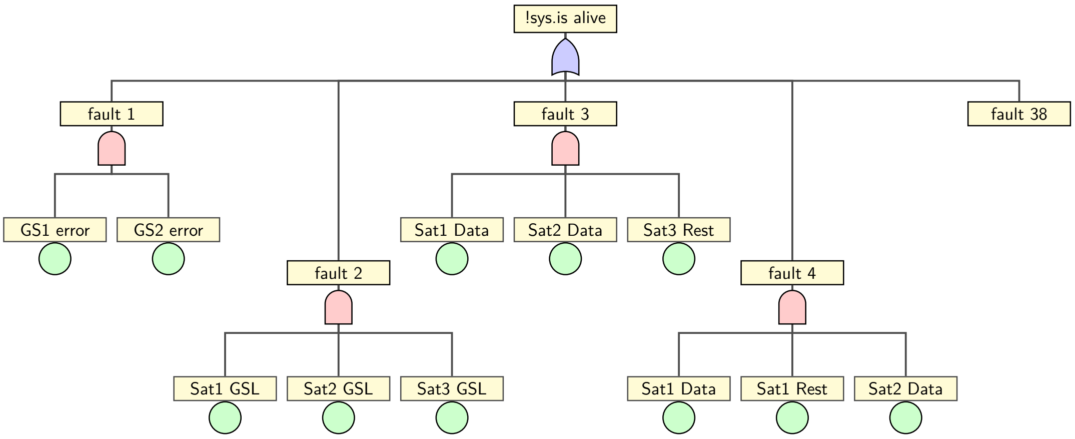


Figure 4.16: Part of generated FT from COMPASS

# 5 Implementation

To ensure compatibility with the tools, several adjustments had to be made during implementation. These are described in the following section.

## 5.1 SysML and SAFEST

The SysML specification must be annotated with safety information. The SAFEST tool provides two dedicated packages for this purpose. These packages enable the automatic generation of FTs from SysML models. The packages are documented in a separate section of the SAFEST manual entitled “Annotation of SysML Models with Safety Information”. The two packages are called DGBMetadata and FailureModes. The DGBMetadata package includes a subpackage called DFTElements. This subpackage defines all elements required to construct FTs. It includes DFTGates for all gate types, such as AND, OR, and SPARE. The FailureModes package is used to associate the BE with probabilistic failure rates. In our case, an exponential distribution was assigned to the BE. This choice was arbitrary, since no specific failure rate was known. For the current analysis, qualitative methods are primarily applied. Therefore, the choice of distribution is not critical, but it could be refined in future work.

If multiple scenarios or FTs are required, more than one element can be defined as a Top-Level-Event element.

During the implementation, we encountered another problem. It is not possible to associate multiple satellites through the generic model because SAFEST currently generates only one BE per component type. To model each satellite component individually, each component must therefore be assigned a unique name. As a result, symmetry in the system cannot be exploited. Every component of each satellite must be written out explicitly (see line 29-72 in Appendix A.1.1).

In total, the model consists of 222 lines, plus an additional 107 lines for the packages provided by the SAFEST tool.

## 5.2 AADL and COMPASS

In section 4.3, we discussed the modes used by satellites, as well as their components and the overarching system. The SLIM extension includes mode and state transitions to facilitate this. Unfortunately, during the simulation of the model in the COMPASS tool, there was a significant delay in mode transitions for three satellites and their components. This issue made it quite challenging to analyse the model and develop a functioning version.

Hence, we modelled the modes as ports that activate when the instance is in that specific mode. For example, this occurs when a component is defective or when the satellite is designated as a master. That is why each component has an outgoing port for this defect (e.g., line 46 for the Data Sensor in Appendix A.2.1).

The overall system can operate in nine distinct modes. These modes result from the configurations of three satellites. Each satellite can act as the master (S1-as-M, S2-as-M, S3-as-M). The master can either acquire data itself (S1-as-M-S1-data) or delegate this task to another satellite (S1-as-M-S2-data, S1-as-M-S3-data). This results in nine possible system states.

The mode changes are implemented by a controller. The controller has nine input ports, three for each satellite. These ports represent the three components of each satellite that may fail (lines 87-95 in Appendix A.2.1). Thus, the controller is connected to S4-S6 of each satellite. It also has nine output ports, one for each system mode (lines 96-104 in Appendix A.2.1).

Inside the controller, the ports are connected according to the communication topology of the constellation (see lines 107-123 in Appendix A.2.1). For example, the port go_to_s1_as_M_S1_data is true if satellite 1 is the master and acquires data independently. This is only possible if Sat1_GSL_defect and Sat1_data_defect are not true. Therefore, the port is connected with Sat1_GSL_defect and Sat1_data_defect. Further cases are described in the Appendix.

In lines 177-179 of the code in the Appendix A.2.1, the port defining which satellite acts as master is connected to the active system mode.

In the controller, more than one output port can be true simultaneously. For example, at the start without faults, three ports are true. In reality, only one satellite can be the master. However, the analysis focuses on negative cases; faults that lead to system failure. To avoid these conflicts a predefined order that selects the active mode when more than one port is true can be implemented.

The entire AADL specification consists of 298 lines plus 12 for the property definition.

# 6 Fault Tree Analysis

After modelling the satellite constellation system in both languages, the resulting FT is generated by the tools (see Appendix A.1.2 and A.2.3). The visual representation of the FTs shows notable differences. The manually created fault tree in the SAFEST tool presents a hierarchical structure, whereas COMPASS has a flatter and more horizontal format. Also, the number of nodes varies significantly between the two FTs. The FT generated from COMPASS has 166 nodes, while the SAFEST system only utilises 44 nodes. This variation arises from COMPASS's approach of directly calculating the min cut sets and presenting them as an FT. Each node, labelled fault1, fault2, fault3, etc., corresponds to a min cut of the system (see min cut definition in section 2.2).
It is also possible to calculate the min cuts using the SAFEST tool. For the analysis of satellite systems, we are particularly interested in a more in-depth analysis of these cuts.

## 6.1 Analysis of Min Cuts

The first min cut involves the failure of both ground stations and is the smallest cut.
To improve readability in the following analysis, the min cuts are organised into a table, with each row representing a distinct min cut. Given that these relate to errors in the satellite components, there are three columns for the three satellites. The table is organised into groups where the min cuts 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 any operational configuration is not feasible. The table with the min cuts is shown in Table 6.1.

| Satellite 1 | Satellite 2 | Satellite 3 |
|---|---|---|
| Rest | Rest | Rest |
| Data | Data | Data |
| GSL | GSL | GSL |
| Rest | GSL | GSL |
| GSL | Rest | GSL |
| GSL | GSL | Rest |
| GSL | Rest | Rest |
| Rest | GSL | Rest |
| Rest | Rest | GSL |
| Rest | Data | Data |
| Data | Rest | Data |
| Data | Data | Rest |
| Data | Rest | Rest |
| Rest | Data | Rest |
| Rest | Rest | Data |

Table 6.1: 15 standard min cuts for satellite component failures

There are six additional scenarios in which the system can not operate, regardless of the communication typology used, see Table 6.2. 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 receive commands from the other satellites (due to an ISL failure). In the second group, only one satellite can establish a ground connection (with GSL still functional), but it cannot forward commands to another satellite because of an ISL failure.

| 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 |

Table 6.2: Further typology invariant min cuts

## 6.2 Comparison of Fault Trees for Different Typologies of Satellite Communication

Now, we will compare the min cuts for two different types of communication: directed circle and undirected circle (communication typologies see section 4.1.4). Table 6.3 illustrates the differences in min cuts between the two communication typologies. In the case of the undirected circle, an additional failure in the ISL is required for the entire system to fail (see Table 6.3a). This is because communication can be forwarded in both directions in the undirected circle.

| Directed Circle | | | Undirected Circle | | |
|---|---|---|---|---|---|
| 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 circle

| Directed Circle | | | Undirected Circle | | |
|---|---|---|---|---|---|
| Satellite 1 | Satellite 2 | Satellite 3 | Satellite 1 | Satellite 2 | Satellite 3 |
| GSL | Data | Rest | | | |
| Rest | GSL | Data | | | |
| Data | Rest | GSL | | | |
| GSL ISL | Rest | Data | GSL ISL | Rest | Data |
| | | | GSL ISL | Data | Rest |
| Data | GSL ISL | Rest | Data | GSL ISL | Rest |
| | | | Rest | GSL ISL | Data |
| Rest | Data | GSL ISL | Rest | Data | GSL ISL |
| | | | Data | Rest | GSL ISL |
| ISL Data | GSL | Rest | ISL Data | GSL | Rest |
| | | | ISL Data | Rest | GSL |
| Rest | ISL Data | GSL | Rest | ISL Data | GSL |
| | | | GSL | ISL Data | Rest |
| GSL | Rest | ISL Data | GSL | Rest | ISL Data |
| | | | Rest | GSL | ISL Data |

(b) Further different min cuts

Table 6.3: Comparison of min cuts from directed and undirected circle

Even in the cases shown in Table 6.3b, the failure of an ISL component is required for a complete 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 colour in Table 6.3b. This clarifies the reason behind the observed difference in the number of min cuts: 37 in the case of undirected circles versus 40 for directed circles.

### 6.2.1 Fault Tree Analysis for Undirected Chain

To illustrate the practical applications of the analysis of min cuts, we will now examine min cuts of the directed chain, which remains a realistic typology of communication. Beyond the scenarios illustrated in Table 6.1 and Table 6.2, there are an additional 16 min cuts for the undirected chain presented in Table 6.4. Specifically, 10 of these cases arise due to the failure of the ISL component in the middle satellite, see Table 6.4a. 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 within the undirected circle. The same applies to the 6 additional cases shown in Table 6.4b.

| 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 |

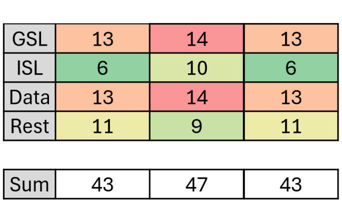
(a) Min cuts with defect ISL of Sat2

| 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 6.4: Additional min cuts for undirected chain

The min cuts can also be utilised to create a detailed table that displays the frequency of each component of each satellite found in the min cuts. This table is illustrated in Table 6.5 for the undirected chain.



| | Satellite1 | Satellite2 | Satellite3 |
|---|---|---|---|
| GSL | 13 | 14 | 13 |
| ISL | 6 | 10 | 6 |
| Data | 13 | 14 | 13 |
| Rest | 11 | 9 | 11 |
| Sum | 43 | 47 | 43 |

Table 6.5: Frequency of components in min cut sets for undirected chain

The data indicates that satellite 2 plays a crucial role in this type of communication, along with the GSL and Data Sensor component for the satellites in general.

## 6.3 Discussion

Section 6 shows that first analyses with the created models are already possible, particularly by using the table that shows the frequency of individual components within the min cut sets. However, additional information such as the probabilistic failure rates are still missing in order to perform a full quantitative analysis and to derive more precise conclusions about the satellite system design.

### 6.3.1 Tool Comparison

COMPASS and SAFEST show different strengths and weaknesses. COMPASS is a dated tool, but it allows automatic FT generation. The representation of the FT is less intuitive in structure, but the min cut sets can be read directly from the tree. The operation of COMPASS is cumbersome due to outdated software and running it in a virtual machine. SAFEST is more modern, but it does not yet offer a fully automatic generation of FTs from the system specification. It mainly extracts from user-given annotations in the SysML specification a graphical representation of FTs. Although SAFEST generally offers a more intuitive handling, two major drawbacks remain. The first drawback is the generic creation 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 copying and pasting, which is prone to human error, see lines 27-72 in Appendix A.1. The second drawback is the missing export function for min cut sets 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.

# 7 Future Work

Several directions for future work can be identified, with respect to both the system specification and the FT analysis.

## 7.1 Extension of System Specification

The current model could be further developed by including a larger number of satellites and by providing more detailed specifications for their components. For instance, the Data Sensor could be divided into two distinct parts: one dedicated to data acquisition and another focused on data downlinking. This separation would enable inter-satellite data exchange, allowing one satellite to acquire data and transfer it to the other satellite if the downlinking component were to fail. Such an improvement could enhance the system's reliability. The approach outlined in AADL should be easily adaptable to more refined system specifications. Each component has been modelled as a distinct system, allowing for a more detailed specification within the overall framework. Additionally, adding multiple satellites should be relatively straightforward, as a generic satellite model has been used. The only manual work required involves establishing the connections between the satellites, based on the chosen typology. While the COMPASS tool may be considered somewhat outdated, it is reasonable to infer that it can also handle larger models, given its successful application in a model with around 80 components [15]. Our current model, in comparison, has significantly fewer components.

With the SysML model, scaling the approach presents challenges, as many elements are hard-coded, necessitating individual copying and manual connections for each satellite within the system. Moreover, even if the satellite components are specified more precisely, the fault tree would need to be recreated from scratch.

## 7.2 Improvement of Analysis

An improvement of the analysis methods for the generated FT is necessary. For example, methods to make the results of the min cut analysis more comparable need to be developed. The frequency of occurrence of a specific basic event alone does not necessarily indicate higher criticality; additional criteria need to be employed. Moreover, probabilistic analyses should be used by assigning probabilistic failure rates to the satellite components. This would enable quantitative analysis of the system, including metrics such as mean time to failure (MTTF) and overall system reliability, and would give important details in the system design.

## 7.3 Background Questions

Several practical aspects regarding the modelling remain open and require further investigation. First, the choice of the most suitable communication topology must be clarified. Second, it is important to examine the implementation of dynamic reconfiguration in real-world scenarios. In particular, consideration must be given to how satellites can identify faults in other satellites. The reasoning for this satellite constellation approach is to facilitate distributed and autonomous satellites, which minimises reliance on ground communication. This is why such a reconfiguration cannot be managed from the ground.

# 8 Conclusion

This thesis aimed to automatically generate fault trees for satellite constellations out of system specifications. It confirms that the generation is feasible and that initial analyses of the FT can provide insights into design questions. However, to conduct a thorough quantitative analysis, additional information, like probabilistic failure rates of the components and more knowledge about the physical background, is necessary.
The tools under examination have both advantages and disadvantages. COMPASS can successfully generate FTs for satellite constellations out of a system specification. However, the FTs generated by COMPASS lack intuitiveness regarding their structure and the tool is cumbersome to use due to its outdated software. But the AADL model is generally flexible and can be easily refined, as each subsystem is specified separately. It is also reasonable to assume that COMPASS can handle a more specified model, given that it has successfully managed even larger models in previous studies.
In contrast, SAFEST is more modern but does not yet fully support automatic FT generation from system specifications. It mainly offers a graphical representation of manually created FTs in SysML.
In summary, this thesis provides a systematic approach to modelling and analysing the fault behaviour of satellite constellations. It demonstrates the potential of automatic FT generation to improve the reliability assessment of complex space systems. At the same time, it highlights the current constraints of available tools, thereby providing a foundation for further development in automated FT analysis.

# Bibliography

[1] J. A. Fraire, S. Henn, G. Stock, R. Ohs, H. Hermanns, F. Walter, L. Van Broock, G. Ruffini, F. Machado, P. Serratti, J. Relloso. “Quantitative analysis of segmented satellite network architectures: A maritime surveillance case study”. *Computer Networks* 255 (2024), p. 110874. DOI: `10.1016/j.comnet.2024.110874`.

[2] Z. Zhao, R. Li, T. Meng. “A MBSE-based fault modeling approach for satellite”. *Journal of Physics: Conference Series* 2977.1 (2025), p. 012077. DOI: `10.1088/1742-6596/2977/1/012077`.

[3] E. Ruijters, M. Stoelinga. “Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools”. *Computer Science Review* 15-16 (2015), pp. 29–62. DOI: `10.1016/j.cosrev.2015.03.001`.

[4] M. Bozzano, A. Villafiorita. *Design and safety assessment of critical systems.* My-iLibrary. Boca Raton, London, and New York, NY: CRC Press, 2010. DOI: `10.1201/b10094`.

[5] A Berres, H Schumann. “Automatic generation of fault trees: A survey on methods and approaches”. *Risk, Reliability and Safety.* Ed. by L. Walls, M. Revie, T. Bedford. Boca Raton: CRC Press, 2016, pp. 2485–2492. DOI: `10.1201/9781315374987-377`.

[6] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, M. Roveri. “The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems”. *Computer safety, reliability, and security.* Ed. by B. Buth, G. Rabe, T. Seyfarth. Vol. 5775. Lecture Notes in Computer Science. Berlin and Heidelberg: Springer, 2009, pp. 173–186. DOI: `10.1007/978-3-642-04468-7_15`.

[7] M. Bozzano, H. Bruintjes, A. Cimatti, J.-P. Katoen, T. Noll, S. Tonetta. “COMPASS 3.0”. *Tools and algorithms for the construction and analysis of systems.* Ed. by T. Vojnar, L. Zhang. Lecture notes in computer science ARCoSS, Advanced research in computing and software science. Cham: Springer, 2019, pp. 379–385. DOI: `10.1007/978-3-030-17462-0_25`.

[8] M. Hause. “The OMG Modelling Language (SYSML)”. *Fifteenth European systems engineering conference Vol.9* (2006), pp. 1–12.

[9] S. Wolny, A. Mazak, C. Carpella, V. Geist, M. Wimmer. “Thirteen years of SysML: a systematic mapping study”. *Software and Systems Modeling* 19.1 (2020), pp. 111–169. DOI: `10.1007/s10270-019-00735-y`.

[10] M. Volk, F. Sher, J.-P. Katoen, M. Stoelinga. “SAFEST: Fault Tree Analysis Via Probabilistic Model Checking”. *2024 Annual Reliability and Maintainability Symposium (RAMS).* IEEE, 2024, pp. 1–7. DOI: `10.1109/RAMS51492.2024.10457719`.

[11] F. Mhenni, N. Nguyen, J.-Y. Choley. “Automatic fault tree generation from SysML system models”. *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics.* IEEE, 2014, pp. 715–720. DOI: `10.1109/AIM.2014.6878163`.

[12] F. Mhenni, N. Nguyen, J.-Y. Choley. “SafeSysE: A Safety Analysis Integration in Systems Engineering Approach”. *IEEE Systems Journal* 12.1 (2018), pp. 161–172. DOI: `10.1109/JSYST.2016.2547460`.

[13] A. Joshi, S. Vestal, P. Binns. *Automatic Generation of Static Fault Trees from AADL Models*. 2007. DOI: `https://hdl.handle.net/11299/217313`.

[14] M.-A. Esteve, J.-P. Katoen, V. Y. Nguyen, B. Postma, Y. Yushtein. “Formal correctness, safety, dependability, and performance analysis of a satellite”. *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 1022–1031. DOI: `10.1109/ICSE.2012.6227118`.

[15] V. Y. Nguyen, ed. *Trustworthy spacecraft design using formal methods*. 2012,17. Aachen: Dep. of Computer Science, RWTH Aachen, 2013. DOI: `https://publications.rwth-aachen.de/record/229446`.

# A Appendix

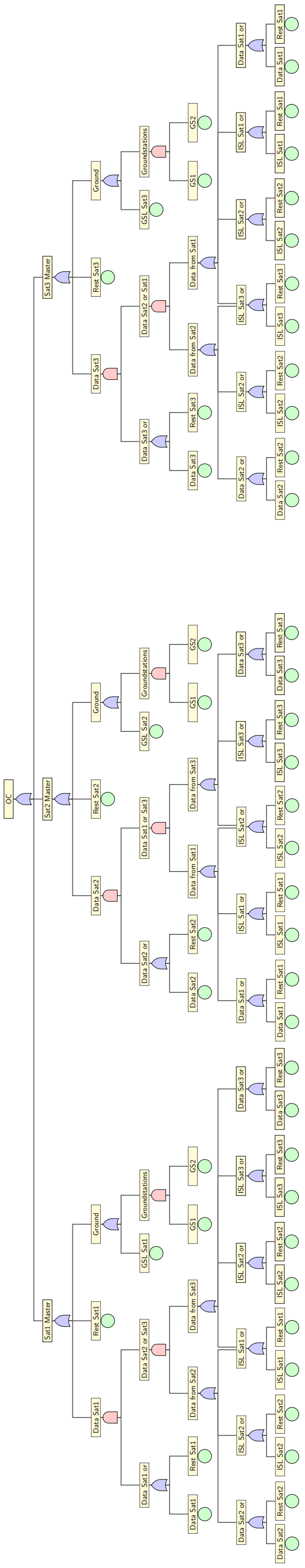
## A.1 SysML

### A.1.1 Code

```
package SystemPartDefinitions {
    public import FailureModes::*;

    part def Satellite {
        part GSL {
            metadata GSLFailure:FIT1;
        }
        part Data {
            metadata DataFailure:FIT1;
        }
        part ISL {
            metadata ISLFailure:FIT1;
        }
        part Rest {
            metadata RestFailure:FIT1;
        }
    }

    part def GroundStation {
        metadata FailureGS:FIT1;
    }
}

package SystemPartsTree {
    public import SystemPartDefinitions::*;

    part def basic {
        // 3 Satellites and 2 Groundstations
        part sat1 : Satellite[1]{
            part GSL {
                metadata GSLFailureSat1:FIT1;
            }
            part Data {
                metadata DataFailureSat1:FIT1;
            }
            part ISL {
                metadata ISLSat1:FIT1;
            }
            part Rest {
                metadata RestFailureSat1:FIT1;
            }
        }

        part sat2 : Satellite[1]{
            part GSL {
                metadata GSLFailureSat2:FIT1;
            }
            part Data {
                metadata DataFailureSat2:FIT1;
            }
            part ISL {
                metadata ISLSat2:FIT1;
            }
            part Rest {
                metadata RestFailureSat2:FIT1;
            }
        }

        part sat3 : Satellite[1]{
            part GSL {
                metadata GSLFailureSat3:FIT1;
            }
            part Data {
                metadata DataFailureSat3:FIT1;
            }
            part ISL {
                metadata ISLSat3:FIT1;
            }
            part Rest {
                metadata RestFailureSat3:FIT1;
            }
        }

        part GS1 : GroundStation[1]{
            metadata FailureGS1:FIT1;
        }

        part GS2 : GroundStation[1]{
            metadata FailureGS2:FIT1;
        }
    }
}

package FaultTree {
    public import SystemPartDefinitions::*;
    public import SystemPartsTree::*;
    public import DGBMetadata::DFTElements::*;
    public import FailureModes::*;

    metadata Groundstations:AND about
        basic::GS1::FailureGS1,
        basic::GS2::FailureGS2;

    metadata ISLSat1OR:OR about
        basic::sat1::ISL::ISLSat1,
        basic::sat1::Rest::RestFailureSat1;

    metadata ISLSat2OR:OR about
        basic::sat2::ISL::ISLSat2,
        basic::sat2::Rest::RestFailureSat2;

    metadata ISLSat3OR:OR about
        basic::sat3::ISL::ISLSat3,
        basic::sat3::Rest::RestFailureSat3;

    metadata GSLSat1OR:OR about
        basic::sat1::GSL::GSLFailureSat1,
        basic::sat1::Rest::RestFailureSat1;

    metadata GSLSat2OR:OR about
        basic::sat2::GSL::GSLFailureSat2,
        basic::sat2::Rest::RestFailureSat2;

    metadata GSLSat3OR:OR about
        basic::sat3::GSL::GSLFailureSat3,
        basic::sat3::Rest::RestFailureSat3;

    metadata DataSat1OR:OR about
        basic::sat1::Data::DataFailureSat1,
        basic::sat1::Rest::RestFailureSat1;

    metadata DataSat2OR:OR about
        basic::sat2::Data::DataFailureSat2,
        basic::sat2::Rest::RestFailureSat2;

    metadata DataSat3OR:OR about
        basic::sat3::Data::DataFailureSat3,
        basic::sat3::Rest::RestFailureSat3;

    //sat1 is master, sat2 and sat3 are slaves
    metadata datafromSat3Sat1M:OR about
        DataSat3OR,
        ISLSat3OR,
        ISLSat2OR,
        ISLSat1OR;

    metadata datafromSat2Sat1M:OR about
```

```
        DataSat2OR,
        ISLSat1OR,
        ISLSat2OR;

    metadata datafromSat2orSat3:AND about
        datafromSat3Sat1M,
        datafromSat2Sat1M;

    metadata DataSat1:AND about
        DataSat1OR,
        datafromSat2orSat3;

    metadata GroundSat1:OR about
        GSLSat1OR,
        Groundstations;

    metadata Sat1Master:OR about
        GroundSat1,
        DataSat1,
        basic::sat1::Rest::RestFailureSat1;

    //sat2 is master
    metadata datafromSat1Sat2M:OR about
        DataSat1OR,
        ISLSat1OR,
        ISLSat2OR;

    metadata datafromSat3Sat2M:OR about
        DataSat3OR,
        ISLSat2OR,
        ISLSat3OR;

    metadata datafromSat3orSat1:AND about
        datafromSat1Sat2M,
        datafromSat3Sat2M;

    metadata DataSat2:AND about
        DataSat2OR,
        datafromSat3orSat1;

    metadata GroundSat2:OR about
        GSLSat2OR,
        Groundstations;

    metadata Sat2Master:OR about
        GroundSat2,
        DataSat2,
        basic::sat2::Rest::RestFailureSat2;

    //sat3 is master
    metadata datafromSat1Sat3M:OR about
        DataSat1,
        ISLSat1OR,
        ISLSat2OR,
        ISLSat3OR;

    metadata datafromSat2Sat3M:OR about
        DataSat2,
        ISLSat2OR,
        ISLSat3OR;

    metadata datafromSat2orSat1:AND about
        datafromSat1Sat3M,
        datafromSat2Sat3M;

    metadata dataAcquisitionSat3:AND about
        DataSat3OR,
        datafromSat2orSat1;

    metadata GroundSat3:OR about
        GSLSat3OR,
        Groundstations;

    metadata Sat3Master:OR about
        GroundSat3,
        dataAcquisitionSat3,
        basic::sat3::Rest::RestFailureSat3;

    metadata operationalConfig:AND about
        Sat1Master,
        Sat2Master,
        Sat3Master;

    metadata TLE:TOP_LEVEL about operationalConfig;
}
```

## A.1.2. Generated Fault Tree



## A.2 AADL

### A.2.1 SLIM Code

```
package SatelliteConstellation
public

-- Top-level enclosing the system
system Enclosure
end Enclosure;
-- defining all systems and their features/ports
system basicSystem
features
    is_alive: out data port bool {Observable => true;};
    Sat1_data_defect: out data port bool {Default => "false"; Observable => true;};
    Sat2_data_defect: out data port bool {Default => "false"; Observable => true;};
    Sat3_data_defect: out data port bool {Default => "false"; Observable => true;};
    Sat1_GSL_defect: out data port bool {Default => "false"; Observable => true;};
    Sat2_GSL_defect: out data port bool {Default => "false"; Observable => true;};
    Sat3_GSL_defect: out data port bool {Default => "false"; Observable => true;};
    Sat1_ISL_defect: out data port bool {Default => "false"; Observable => true;};
    Sat2_ISL_defect: out data port bool {Default => "false"; Observable => true;};
    Sat3_ISL_defect: out data port bool {Default => "false"; Observable => true;};
    go_to_s1_as_M_S1_data: in data port bool {Default => "false";};
    go_to_s1_as_M_S2_data: in data port bool {Default => "false";};
    go_to_s1_as_M_S3_data: in data port bool {Default => "false";};
    go_to_s2_as_M_S1_data: in data port bool {Default => "false";};
    go_to_s2_as_M_S2_data: in data port bool {Default => "false";};
    go_to_s2_as_M_S3_data: in data port bool {Default => "false";};
    go_to_s3_as_M_S1_data: in data port bool {Default => "false";};
    go_to_s3_as_M_S2_data: in data port bool {Default => "false";};
    go_to_s3_as_M_S3_data: in data port bool {Default => "false";};
```

```
end basicSystem;

system Satellite
    features
        command_data_from_this_sat: in data port bool {Default => "false";};
        Master: in data port bool {Default => "false";};
        data_to_ground: out data port bool {Default => "false";};
        command_to_other_sat: out data port bool {Default => "false";};
        data_defect: out data port bool {Default => "false";};
        GSL_defect: out data port bool {Default => "false";};
        ISL_defect: out data port bool {Default => "false";};
end Satellite;

system dataSensor
    features
        get_command: in data port bool {Default => "true";};
        output_data: out data port bool {Default => "true";};
        defect: out data port bool {Default => "false";};
end dataSensor;

system Ground
    features
        receive_data: in data port bool {Default => "true";};
        command_data: out data port bool {Default => "true";};
        data_for_processing: out data port bool {Default => "true"; Observable => true;};
end Ground;

system Groundstation
    features
        receive_data: in data port bool {Default => "true";};
        command_data: out data port bool {Default => "true";};
        data_for_processing: out data port bool {Default => "true"; Observable => true;};
        defect: out data port bool {Default => "false";};
end Groundstation;

system GSL
    features
        get_command_GSL: in data port bool {Default => "true";};
        output_command: out data port bool {Default => "true";};
        defect: out data port bool {Default => "false";};
end GSL;

system ISL
    features
```

```
        get_command_ISL: in data port bool {Default => "true";};
        get_command_other_sat: in data port bool {Default =>"false";};
        output_command: out data port bool {Default => "true";};
        command_other_sat: out data port bool {Default =>"false";};
        defect: out data port bool {Default => "false";};
end ISL;

system Rest
    features
        defect: out data port bool {Default => "false";};
end Rest;

system controller
    features
        Sat1_data_defect: in data port bool {Default => "false";};
        Sat2_data_defect: in data port bool {Default => "false";};
        Sat3_data_defect: in data port bool {Default => "false";};
        Sat1_GSL_defect: in data port bool {Default => "false";};
        Sat2_GSL_defect: in data port bool {Default => "false";};
        Sat3_GSL_defect: in data port bool {Default => "false";};
        Sat1_ISL_defect: in data port bool {Default => "false";};
        Sat2_ISL_defect: in data port bool {Default => "false";};
        Sat3_ISL_defect: in data port bool {Default => "false";};
        go_to_s1_as_M_S1_data: out data port bool {Default => "true";};
        go_to_s1_as_M_S2_data: out data port bool {Default => "true";};
        go_to_s1_as_M_S3_data: out data port bool {Default => "true";};
        go_to_s2_as_M_S1_data: out data port bool {Default => "true";};
        go_to_s2_as_M_S2_data: out data port bool {Default => "true";};
        go_to_s2_as_M_S3_data: out data port bool {Default => "true";};
        go_to_s3_as_M_S1_data: out data port bool {Default => "true";};
        go_to_s3_as_M_S2_data: out data port bool {Default =>
            "true";};
        go_to_s3_as_M_S3_data: out data port bool {Default =>
            "true";};
    end controller;

system implementation controller.Imp
    connections
        -- Sat1 is Master
        flow not Sat1_GSL_defect and not Sat1_data_defect ->
            go_to_s1_as_M_S1_data;
        flow not Sat1_GSL_defect and Sat1_data_defect and not
            Sat2_data_defect and not Sat1_ISL_defect and not
            Sat2_ISL_defect -> go_to_s1_as_M_S2_data;
        flow not Sat1_GSL_defect and Sat1_data_defect and
            Sat2_data_defect and not Sat3_data_defect and not
            Sat1_ISL_defect and not Sat2_ISL_defect and not
            Sat3_ISL_defect -> go_to_s1_as_M_S3_data;

        -- Sat2 is Master
        flow not Sat2_GSL_defect and not Sat2_data_defect ->
            go_to_s2_as_M_S2_data;
        flow not Sat2_GSL_defect and Sat2_data_defect and not
            Sat3_data_defect and not Sat2_ISL_defect and not
            Sat3_ISL_defect -> go_to_s2_as_M_S3_data;
        flow not Sat2_GSL_defect and Sat2_data_defect and not
            Sat1_data_defect and not Sat2_ISL_defect and not
            Sat1_ISL_defect -> go_to_s2_as_M_S1_data;

        -- Sat3 is Master
        flow not Sat3_GSL_defect and not Sat3_data_defect ->
            go_to_s3_as_M_S3_data;
        flow not Sat3_GSL_defect and Sat3_data_defect and not
            Sat2_data_defect and not Sat3_ISL_defect and not
            Sat2_ISL_defect -> go_to_s3_as_M_S2_data;
        flow not Sat3_GSL_defect and Sat3_data_defect and
            Sat2_data_defect and not Sat1_data_defect and not
            Sat1_ISL_defect and not Sat2_ISL_defect and not
            Sat3_ISL_defect -> go_to_s3_as_M_S1_data;
end controller.Imp;

system implementation Enclosure.Imp
    subcomponents
        sys: system basicSystem.Imp;
        controller: system controller.Imp;
    connections
        -- propagating information about defects to controller
            and modes to basicSystem
        port sys.Sat1_data_defect ->
           controller.Sat1_data_defect;
        port sys.Sat2_data_defect ->
           controller.Sat2_data_defect;
        port sys.Sat3_data_defect ->
           controller.Sat3_data_defect;
        port sys.Sat1_GSL_defect -> controller.Sat1_GSL_defect;
        port sys.Sat2_GSL_defect -> controller.Sat2_GSL_defect;
        port sys.Sat3_GSL_defect -> controller.Sat3_GSL_defect;
        port sys.Sat1_ISL_defect -> controller.Sat1_ISL_defect;
        port sys.Sat2_ISL_defect -> controller.Sat2_ISL_defect;
        port sys.Sat3_ISL_defect -> controller.Sat3_ISL_defect;
        port controller.go_to_s1_as_M_S1_data ->
           sys.go_to_s1_as_M_S1_data;
        port controller.go_to_s1_as_M_S2_data ->
           sys.go_to_s1_as_M_S2_data;
        port controller.go_to_s1_as_M_S3_data ->
           sys.go_to_s1_as_M_S3_data;
        port controller.go_to_s2_as_M_S1_data ->
           sys.go_to_s2_as_M_S1_data;
        port controller.go_to_s2_as_M_S2_data ->
           sys.go_to_s2_as_M_S2_data;
        port controller.go_to_s2_as_M_S3_data ->
           sys.go_to_s2_as_M_S3_data;
        port controller.go_to_s3_as_M_S1_data ->
           sys.go_to_s3_as_M_S1_data;
        port controller.go_to_s3_as_M_S2_data ->
           sys.go_to_s3_as_M_S2_data;
        port controller.go_to_s3_as_M_S3_data ->
           sys.go_to_s3_as_M_S3_data;
end Enclosure.Imp;

system implementation basicSystem.Imp
subcomponents
    Sat1: system Satellite.Imp;
    Sat2: system Satellite.Imp;
    Ground: system Ground.Imp;
    Sat3: system Satellite.Imp;
connections
    flow (Sat2.command_to_other_sat and (go_to_s1_as_M_S1_data
       or go_to_s2_as_M_S1_data or go_to_s3_as_M_S1_data)) or
       (Ground.command_data and (go_to_s1_as_M_S1_data or
       go_to_s1_as_M_S2_data or go_to_s1_as_M_S3_data)) ->
       Sat1.command_data_from_this_sat;

    flow ((Sat1.command_to_other_sat or
       Sat3.command_to_other_sat) and (go_to_s1_as_M_S2_data or
       go_to_s2_as_M_S2_data or go_to_s3_as_M_S2_data)) or
       (Ground.command_data and (go_to_s2_as_M_S1_data or
       go_to_s2_as_M_S2_data or go_to_s2_as_M_S3_data)) ->
       Sat2.command_data_from_this_sat;

    flow (Sat2.command_to_other_sat and (go_to_s1_as_M_S3_data
       or go_to_s2_as_M_S3_data or go_to_s3_as_M_S3_data)) or
       (Ground.command_data and (go_to_s3_as_M_S1_data or
       go_to_s3_as_M_S2_data or go_to_s3_as_M_S3_data)) ->
       Sat3.command_data_from_this_sat;

    -- Sat1, Sat2, Sat3 data to Groundstation
    flow (Sat1.data_to_ground and (go_to_s1_as_M_S1_data or
       go_to_s2_as_M_S1_data or go_to_s3_as_M_S1_data)) or
       (Sat2.data_to_ground and (go_to_s1_as_M_S2_data or
       go_to_s2_as_M_S2_data or go_to_s3_as_M_S2_data)) or
       (Sat3.data_to_ground and (go_to_s1_as_M_S3_data or
       go_to_s2_as_M_S3_data or go_to_s3_as_M_S3_data)) ->
       Ground.receive_data;
    -- Infos for Controller
    port Sat1.data_defect -> Sat1_data_defect;
    port Sat2.data_defect -> Sat2_data_defect;
    port Sat3.data_defect -> Sat3_data_defect;
    port Sat1.GSL_defect -> Sat1_GSL_defect;
    port Sat2.GSL_defect -> Sat2_GSL_defect;
    port Sat3.GSL_defect -> Sat3_GSL_defect;
    port Sat1.ISL_defect -> Sat1_ISL_defect;
    port Sat2.ISL_defect -> Sat2_ISL_defect;
    port Sat3.ISL_defect -> Sat3_ISL_defect;
    -- "modes" in which the Satellite are Master
    flow go_to_s1_as_M_S1_data or go_to_s1_as_M_S2_data or
       go_to_s1_as_M_S3_data -> Sat1.Master;
    flow go_to_s2_as_M_S1_data or go_to_s2_as_M_S2_data or
       go_to_s2_as_M_S3_data -> Sat2.Master;
    flow go_to_s3_as_M_S1_data or go_to_s3_as_M_S2_data or
       go_to_s3_as_M_S3_data -> Sat3.Master;
    flow Ground.data_for_processing -> is_alive; -- used as
       TOP_LEVEL_EVENT
end basicSystem.Imp;

system implementation Satellite.Imp
    subcomponents
        dataSensor: system dataSensor.Imp;
        GSL: system GSL.Imp;
        ISL: system ISL.Imp;
        Rest: system Rest.Imp;
    connections
        flow command_data_from_this_sat and not Master ->
           ISL.get_command_ISL;
        flow (ISL.output_command and not Master) or
           (GSL.output_command and Master) ->
```

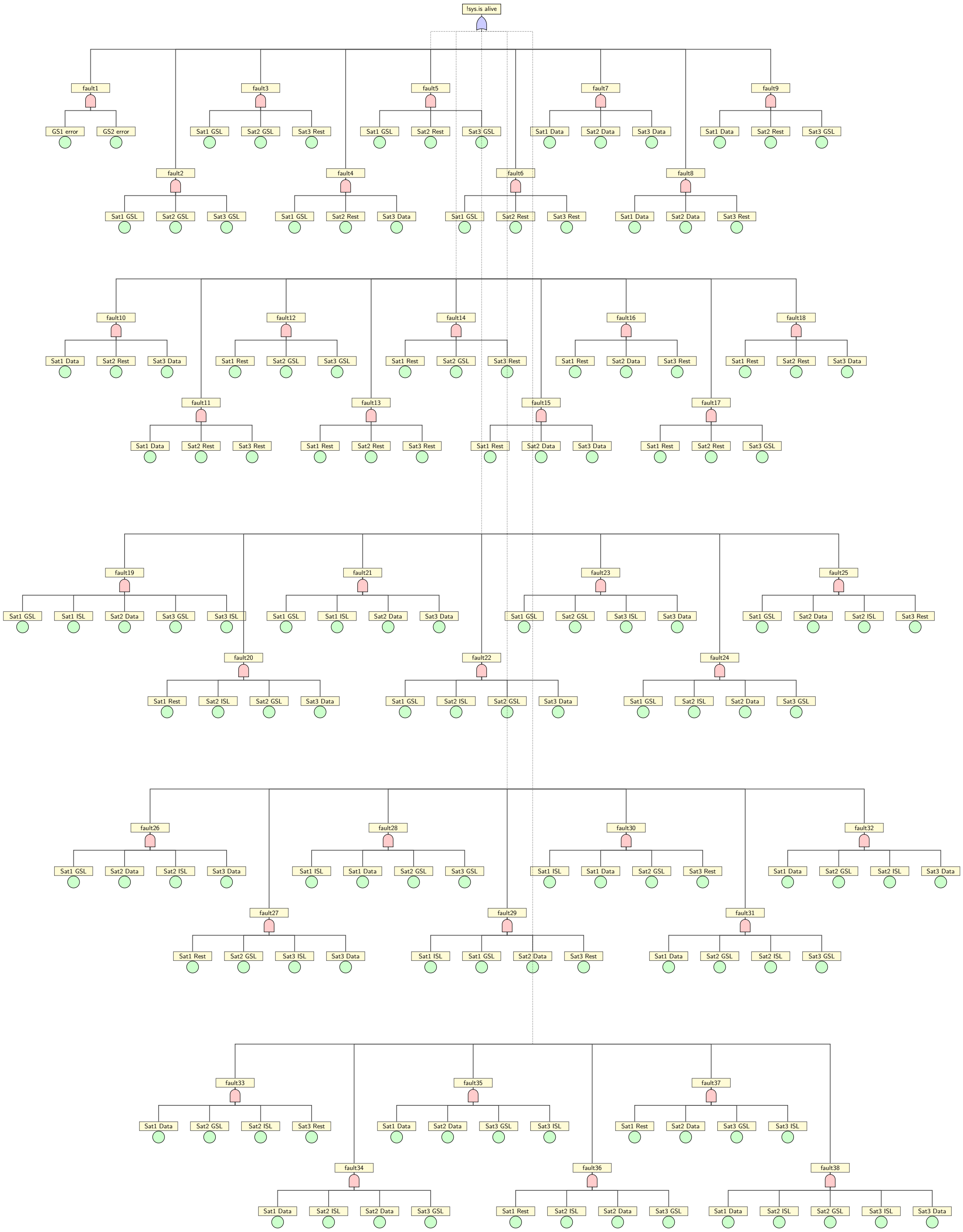
```
                dataSensor.get_command;
            flow command_data_from_this_sat and Master ->
                GSL.get_command_GSL;
            flow dataSensor.output_data -> data_to_ground;
            flow dataSensor.defect -> ISL.get_command_other_sat;
            flow ISL.command_other_sat -> command_to_other_sat;
            flow GSL.defect or Rest.defect -> GSL_defect;
            flow dataSensor.defect or Rest.defect -> data_defect;
            flow ISL.defect or Rest.defect-> ISL_defect;
end Satellite.Imp;

--satellite component implementation
system implementation Rest.Imp
      states
            base: initial state;
            defect_state: state;
      transitions
            base -[ when not defect]-> base;
            base -[ when defect]-> defect_state;
      properties
            ErrorModel => classifier(PermanentFailure.Imp);
            FaultEffects => ([State=>"dead";
                Target=>reference(defect); Effect=>"true";]);
end Rest.Imp;

system implementation dataSensor.Imp
      connections
            flow get_command and not defect -> output_data;
      states
            base: initial state;
            defect_state: state;
      transitions
            base -[ when not defect]-> base;
            base -[ when defect]-> defect_state;
      properties
            ErrorModel => classifier(PermanentFailure.Imp);
            FaultEffects => ([State=>"dead";
                Target=>reference(defect); Effect=>"true";]);
end dataSensor.Imp;

system implementation GSL.Imp
      connections
            flow get_command_GSL and not defect -> output_command;
      states
            base: initial state;
            defect_state: state;
      transitions
            base -[ when not defect]-> base;
            base -[ when defect]-> defect_state;
```

```
    properties
        ErrorModel => classifier(PermanentFailure.Imp);
        FaultEffects => ([State=>”dead”; Target=>reference(defect); Effect=>”true”;]);
end GSL.Imp;

system implementation ISL.Imp
    connections
        flow get_command_ISL and not defect -> output_command;
        flow get_command_other_sat and not defect -> command_other_sat;
    states
        base: initial state;
        defect_state: state;
    transitions
        base -[ when not defect]-> base;
        base -[ when defect]-> defect_state;
    properties
        ErrorModel => classifier(PermanentFailure.Imp);
        FaultEffects => ([State=>”dead”; Target=>reference(defect); Effect=>”true”;]);
end ISL.Imp;

-- Ground
system implementation Groundstation.Imp
    connections
        flow receive_data and not defect -> data_for_processing;
        flow not defect -> command_data;
    states
        base: initial state;
        defect_state: state;
    transitions
        base -[ when not defect]-> base;
        base -[ when defect]-> defect_state;
    properties
        ErrorModel => classifier(PermanentFailure.Imp);
        FaultEffects => ([State=>”dead”; Target=>reference(defect); Effect=>”true”;]);
end Groundstation.Imp;

system implementation Ground.Imp
    subcomponents
        Groundstation1: system Groundstation.Imp;
        Groundstation2: system Groundstation.Imp;
    connections
        flow Groundstation1.command_data or Groundstation2.command_data -> command_data;
        port receive_data -> Groundstation1.receive_data;
        port receive_data -> Groundstation2.receive_data;
        flow Groundstation1.data_for_processing or Groundstation2.data_for_processing -> data_for_processing;
end Ground.Imp;

-- Generic Permanent Failure Dynamic

error model PermanentFailure
features
    ok: activation state;
    dead: error state;
end PermanentFailure;

error model implementation PermanentFailure.Imp
events
    fault: error event;
transitions
    ok -[fault]-> dead;
end PermanentFailure.Imp;
end SatelliteConstellation;
```

### A.2.2 Properties

```
<?xml version='1.0' encoding='UTF-8'?>
<ns0:properties xmlns:ns0="http://www.compass-toolset.org/properties">
  <ns0:property>
    <ns0:name>always System is_alive</ns0:name>
    <ns0:description> The atomic proposition basicSystem.is_alive always holds. </ns0:description>
    <ns0:ltlPatternInstance patternId="universalityGlobal">
      <ns0:stateFormulaInstance xmlns:ns0="http://www.compass-toolset.org/ltl" placeholderId="P">
        <ns0:atomicProposition>sys.is_alive</ns0:atomicProposition>
      </ns0:stateFormulaInstance>
    </ns0:ltlPatternInstance>
  </ns0:property>
</ns0:properties>
```

### A.2.3 Generated Fault Tree



# Acknowledgement

I want to thank Prof. Thomas Noll for his support and feedback during this time. I also want to thank for the opportunity to write my bachelor's thesis at the Chair for Software Modeling and Verification.