Improving operational diagnosis with an interdisciplinary modelling approach.

Nikolena Christofi

Université de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *06/02/2023* **par :**

Nikolena CHRISTOFI

Improving operational diagnosis with an interdisciplinary modelling approach.

JURY

M. Yvan BEAUREGARD	Professeur	Rapporteur
M. Mahdi ZARGAYOUNA	Chargé de recherche	Rapporteur
M. Jean-Michel BRUEL	Professeur des Universités	Président
M. Jean-Yves CHOLEY	Professeur des Universités	Examinateur
Mme Besma ZEDDINI	Assistant Professor	Examinatrice
Mme Claude BARON	Professeure des Universités	Directrice de thèse
M. Marc PANTEL	Maître de conférences	Co-directeur de thèse
M. Xavier PUCEL	Chargé de recherche	Co-encadrant de thèse

École doctorale et spécialité :

AA : Aéronautique – Astronautique

Unité de Recherche :

Laboratoire d'Analyse et d'Architecture des Systèmes, LAAS-CNRS (UPRR 8001)

Thèse dirigée par :

Claude BARON, *Marc PANTEL* et *Xavier PUCEL*

Invités au jury :

Christophe DUCAMP et David CANU - Airbus Defence and Space

Julien BACLET et Sebastien GUILMEAU - IRT Saint Exupéry

Contents

Vitality shows in not only the ability to persist but the ability to start over.

F. Scott Fitzgerald
The Great Gatsby

Acknowledgements

This thesis is dedicated to all the people that helped me got through the long journey of my PhD and never stopped believing in me. First and foremost, to my parents: I would not be here without you. I owe you everything. To Clement, for his infinite patience and unending support. I wouldn't have done this without you. To my colleagues at the IRT and especially all the S2C and the MOSC team, for making my everyday work life cheerful and fun. Particularly, I would like to thank Jeremy, for providing me all the means for working towards my PhD, and his never ending kindness. To Simon, for everything he has taught me, and his precious support on Safety. To Xavier, for all his aid with Simfia. To Sebastian, for being there from the beginning until the end, and for never letting me down. To Romaric and Jean-Marie, for their significant feedback to my manuscript on Systems Engineering and Formal Methods. To Julien, for dealing with all of us. To my everlasting mentor, Michael, for always being there for me, and his invaluable words of wisdom. To my friend Chambis, always there to stress me more, and to be wrong, again. To Nicolas, for his immense kindness and continuous support. To Christophe, for being the first to believe I would be up for this task and for never loosing faith in me. To David, for his eminent support throughout my PhD, for all the hours he dedicated to having online meetings with me and reading my manuscript, and for everything he has thought me. I will definitely miss working with you. To Marc, for his substantial support on formal methods. To Claude, for always being there for me when I needed her, for constantly striving for perfection and settling for nothing less, for her preeminent high expectations and for helping me find my better self. Thank you for serving as an inspirational figure for me and for all women in STEM. To Xavier, for the paramount role he has played in my thesis, for his creative and forward-thinking ideas, his exceptional writing skills, his incredible knowledge and capacity to accompany students as intricate as me, for helping me get through this and for never putting me down, I hope you get to supervise many more students in your research career. Finally, to my cats, for their endless love and affection, and for being the best alarm clock I could ever ask for. To all my friends and family for constantly lifting me up, while not doubting, even for a second. And to all the underdogs out there: never stop pushing through, for "the depths of the sea are only water after all".

Abstract

This manuscript summarises the work undertaken within an interdisciplinary thesis, which tackles the juxtaposition of three main domains: System Engineering, Dependability and Operational Diagnostics, each studied under a model-based approach. Motivated by a pressing industrial need, the research objective is to increase the efficiency of complex systems' supervision while contributing to the amelioration of their architecture. To that end, the thesis work proposes the creation of operational diagnosis-appropriate models. This proposal, as well as the scientific approach to build it, are original insofar as, to date, no research work addresses this issue through an interdisciplinary approach, nor from a models' perspective.

This dissertation thus proposes a methodology for the creation of an operations-dedicated model. This model is built by extracting and exploiting data from the system functional and dysfunctional models –produced during Systems Engineering and System Dependability Analysis (reliability, availability, maintainability and safety) activities, respectively. For the construction of the operations-dedicated models, we have made the choice of the Behaviour Trees language, and have extended its semantics, in order to achieve a fault detection and diagnosis expressiveness.

The proposed methodology aims to translate system functional and dysfunctional information, produced by the system design definition phase, into executable models, representative of the system operational activities, while offering a high fidelity fault detection and mitigation architecture. The methodology consists in two successive steps. The first step defines the creation of a pivot model, through the automatic transformation of Fault Trees –produced by system dysfunctional analyses, into Behaviour Trees. The pivot model expresses the system's diagnostic objectives. The second step consists in the elicitation of the pivot Behaviour Tree model into a second Behaviour Tree model, by integrating complementary data, produced during system design. The second model is an executable model, and is composed by the system operational activities themselves. This final model constitutes the operations-dedicated model, which can be integrated inside a user-interface equipped monitoring tool, to support operators' diagnostics tasks.

The reason the methodology consists in two steps, where dysfunctional and functional system information are gradually transformed and inserted inside the operations-dedicated model, is because information on system faults alone is not enough for the creation of a model dedicated

to operations, to serve for fault detection and diagnosis. Both functional and dysfunctional system information is necessary to build a model for system monitoring.

The methodology to create the operations-dedicated model, along with the original proposal to adopt extended Behavior Trees for its representation, result from a thorough state-of-the-art analysis and substantial feedback provided by major actors of the national aerospace industry, such as Airbus Defence and Space (ADS) and the Centre National d'Etudes Spatiales (CNES). We also applied the proposed methodology on three different case studies, which allowed us to evaluate its feasibility in industrial contexts, along with its current benefits and prospective potential.

Future work consists in applying the proposed methodology on a real complex system, in order to evaluate its impact on operational diagnosis activities' efficiency, as well as its scalability. Another perspective is the construction of digital twins based on the operations-dedicated model, for the simulation of operational scenarios ahead of the system's deployment, and/or for the feeding back of operational data and return of experience information, during operations and maintenance, which could contribute to the amelioration of system architecture, within a digital data continuity system framework.

Resumé

Ce mémoire résume le travail mené dans le cadre d'une thèse interdisciplinaire, faisant intervenir trois disciplines majeures, l'ingénierie système, la sûreté de fonctionnement et le diagnostic opérationnel, chacune considérée avec une approche reposant sur des modèles. L'objectif des recherches, induit par un besoin industriel avéré, est d'améliorer l'efficacité des activités de surveillance et de maintien en conditions opérationnelles d'un système, tout en contribuant en retour à l'amélioration de son architecture. Pour ce faire, la proposition repose sur la création d'un modèle dédié au support de l'activité de diagnostic en opérations. Cette proposition, ainsi que la démarche scientifique pour la construire, sont originales dans la mesure où, à ce jour, aucun travail de recherche n'aborde cette problématique avec une approche interdisciplinaire, ni sous l'angle des modèles.

La thèse propose ainsi une méthodologie pour créer ce modèle. Celui-ci est construit à partir de données issues des analyses de sûreté de fonctionnement, relatives à la fiabilité, la disponibilité, la maintenabilité et la sécurité du système, et des activités d'ingénierie système i.e. l'analyse de la mission et des opérations du système, et la definition de l'architecture du système du point de vue fonctionnel et structurel. Pour représenter ce modèle, nous avons fait le choix du formalisme des arbres de comportement, dont nous avons étendu la sémantique afin d'obtenir une meilleure expressivité de sa sémantique, par rapport à la détection des pannes et à son aptitude au diagnostic.

La méthodologie proposée procède en deux étapes. La première étape a pour but de définir des objectifs de diagnostic du système; ceux-ci découlent de l'analyse des défaillances potentielles du système, identifiées par les experts en sûreté de fonctionnement. Pour modéliser ces objectifs, nous opérons une transformation des arbres de défaillances, résultant de l'analyse dysfonctionnelle, en arbres de comportement. Ceux-ci permettent d'exprimer les objectifs de détection et de mitigation des défaillances du système. Cependant, pour être utile à la détection et au diagnostic de pannes, ce modèle doit être complété par des données fonctionnelles sur le système. L'objectif de la deuxième étape est alors de transformer les objectifs de diagnostic dans des fonctions du système, afin de maintenir ce dernier en conditions opérationnelles à tout moment, et d'éviter des pannes critiques. La démarche consiste en une élicitation du précédent modèle en un second modèle, sous forme également d'arbre de comportement, enrichi par l'intégration de données provenant des activités d'ingénierie du système, notamment de son architecture. Ce modèle final, construit dans la continuité des modèles d'ingénierie et d'analyse de la sûreté de fonctionnement,

est ainsi spécifiquement dédié au diagnostic en opérations.

L'approche choisie pour construire ce modèle, ainsi que la proposition originale d'adopter des arbres de comportement étendus pour le représenter, résultent d'une analyse approfondie de l'état de l'art et du retour d'expérience d'acteurs majeurs de l'industrie aérospatiale nationale, comme Airbus Defence & Space (ADS) et le Centre National d'Etudes Spatiales (CNES). L'expérimentation de la méthodologie sur trois cas d'études différents nous a permis de mesurer sa pertinence en observant un diagnostic facilité ainsi que d'identifier des perspectives d'amélioration. Le déploiement de la méthodologie dans plusieurs projets industriels permettrait de mieux mesurer sa performance et d'envisager un passage à l'échelle. Par ailleurs, un tel modèle pourrait être utilisé comme jumeau numérique du système, qui, en se nourrissant des données opérationnelles et issues du retour d'expérience, pourrait contribuer à l'amélioration de l'architecture du système, dans le cadre d'une boucle de continuité des données numériques du système.

List of Figures

List of Tables

Glossary

Artifact : engineering artifacts represent engineering data as discipline and/or tool-related models, exchanged in a common data storage or file management system [1]. They can be in the form of system plans, models, tool data [2], requirements, methods, hardware components, computations, use cases [3] . 48, 51, 58, 67, 68, 75, 76, 78, 87, 92, 93, 97, 108, 112, 113

Component : set of materials, assembled according to defined and controlled processes, which cannot be disassembled without destroying its capability and which performs a simple function that can be evaluated against expected performance requirements. Notes: the term “part” is synonymous; the term "part" is preferred when referring to purely mechanical devices; the term "component" is preferred for EEE devices [4]. 38

Diagnosis process of identifying or determining the nature and root causes of a failure, problem, or disease from the symptoms resulting from selected measurements, checks or tests [5]; a diagnosis may also include the functional impacts of these root causes, the order of cuts that will lead to an operational or other kind of impact. 25, 27, 28, 30, 31, 33, 37, 50, 54, 55, 58, 59, 61, 62, 63, 65, 67, 73, 83, 87, 88, 91, 96, 102, 103, 107, 111, 112, 114, 115

Element : combination of integrated equipment, components and parts. Note: an element fulfils a major, self-contained, subset of a segment's objectives [4]. 38

Enabling Systems : they facilitate progression of the system-of-interest through its life cycle [6]. 31, 39, 40

End-of-Life : end of operational life (for satellites). 28, 51

Equipment : integrated set of parts and components; an equipment is self-contained and classified as such for the purposes of separate manufacture, procurement, drawings, specification, storage, issue, maintenance or use; the term "unit" is synonymous with the term "equipment" [4]. Note: an equipment may perform several functions. 38

Error : part of system state which is liable to lead to failure. Manifestation of a fault in a system [7]. Note: an error can change the working mode of the system e.g. nominal to degraded, cause a reconfiguration, etc. 53, 54, 56, 58, 73, 95, 96

Failure : deviation of the delivered service from compliance with the specification –and more generally from the expected result; transition from correct service delivery to incorrect service delivery [7]. 51, 53, 54, 55, 56, 57, 58, 59, 73, 76, 78, 80, 82, 83, 85, 89, 95, 101, 102, 103, 106, 108

Fault : adjudged or hypothesized cause of an error. Error cause which is intended to be avoided or tolerated [7]. 51, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 67, 69, 72, 74, 78, 84, 85, 87, 88, 89, 90, 91, 95, 97, 99, 100, 101, 102, 103, 104, 105, 108

Lifecycle : all phases in the life of a product from needs identification through disposal [4]. 28, 31

Ontology (software engineering) :

(1) Ontologies provide a shared and common understanding of a domain that can be communicated between people and application systems [8].

(2) The role of ontologies is to capture domain knowledge in a generic way and to provide a commonly agreed upon understanding of a domain. The common vocabulary of an ontology, defining the meaning of terms and their relations, is usually organised in a taxonomy and contains modelling primitives such as classes, relations, functions, and axioms [9].

(3) An ontology is a formal explicit specification of a shared conceptualization [10] . 46

Safety : freedom from those conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property [11]. 51

Segment : set of elements or combination of systems that fulfils a major, self-contained, subset of the space mission objectives. Examples are space segment, ground segment, launch segment and support segment [4]. 17, 38

Space System : system that contains at least a space, a ground or a launch segment. Note: generally a space system is composed of all three segments and is supported by a support segment [4]. 38

Subsystem : part of a system performing some service(s) or function(s) thereof [4]. 38, 78

System : set of interrelated or interacting functions constituted to achieve a specified objective [4]. 38, 51

System-of-Interest (SoI) : system or subsystem whose life cycle is under consideration [6]. 39, 44

System of Systems (SoS) :

(1) set of systems or system elements that interact to provide a unique capability that none of the constituent systems can accomplish on its own. Note: systems elements can be necessary to facilitate the interaction of the constituent systems in the system of

systems [12].

(2) an SoI whose elements are managerially and/or operationally independent systems. These inter-operating and/or integrated collections of constituent systems usually produce results unachievable by the individual systems alone. Because an SoS is itself a system, the systems engineer may choose whether to address it as wither a system or as an SoS depending on which perspective is better suited to a particular problem [6]. The following characteristics can be useful when deciding if a particular SoI can better be understood as as SoS [13]: (i) operational independence of constituent systems, (ii) managerial independence of constituent systems, (iii) geographical distribution, (iv) emergent behaviour, (v) evolutionary development process. 38, 39, 49, 50, 105

Troubleshooting : carried out by operators if the failure is not known or capitalized on, and especially if the automatic monitoring data does not directly provide the necessary information to identify the source of failure; human monitoring and therefore assisted or manual diagnosis e.g. requesting the system to provide additional or specific information or/and performing manual tests – manual investigation longplural. 25, 27, 28

Acronyms

AADL Architecture Analysis and Design Language. 48, 58

ACM Attitude Control Mode. 79, 83, 85

ADS Airbus Defence & Space. 25, 33, 34, 35, 41, 102, 104, 109, 110

AI Artificial Intelligence. 27, 59, 63, 114

AOCS Attitude and Orbit Control System. 93, 94, 95, 96, 109, 110

ASM Acquisition & Safe Mode. 79, 83, 85

BDD Block Definition Diagram. 139, 140, 141, 142, 143

BPD Business Process Diagram. 71

BPEL Business Process Execution Language. 71

BPMN Business Process Model and Notation. 71

BT Behaviour Tree. 30, 31, 33, 35, 37, 63, 64, 65, 66, 67, 72, 73, 74, 75, 76, 78, 79, 83, 84, 85, 87, 88, 89, 90, 91, 92, 93, 95, 96, 97, 103, 106, 107, 108, 109, 110, 112, 113, 114

CAM Collision Avoidance Manoeuvre. 79, 83, 85

CDF Concurrent Design Facility. 29

CEF Concurrent Engineering Facility. 29

CNES Centre National d'Etudes Spatiales / National Centre for Space Studies - French government space agency. 34, 35

ConOPS Concept of Operations. 104

DDMS Digital Design, Manufacturing & Services. 40

DES Discrete Event Systems. 29, 30, 37, 60, 62, 65, 66, 71

DT Digital Twin. 25, 26, 29, 31, 73, 113, 114

ECSS European Cooperation for Space Standardization. 38, 40, 49

EO Earth Observation. 76, 79, 80, 81

ESA European Space Agency. 26, 41, 48

ET Event Tree. 57

FC Failure Condition. 80, 82

FCM Formation Control Model. 79, 83, 85

FDD Fault Detection and Diagnosis. 28, 30, 59, 60, 69, 114

FDI Fault Detection and Isolation. 59

FDIR Fault Detection, Isolation and Recovery. 25, 27, 78, 87, 95, 97, 114

FE Feared Event. 53, 56, 97, 104, 106

FLM Failure Logic Modelling. 60

FMEA Failure Modes and Effects Analysis. 31, 57, 112

FMECA Failure Modes and Criticality Effects Analysis. 54, 57, 60, 73, 76, 78, 112

FPM Failure Propagation Modelling. 60

FPTN Failure Propagation and Transformation Notation. 57

FSA Finite State Automata. 30, 37, 61, 62, 63, 66, 71, 72, 73

FSAP Formal Safety Analysis Platform. 58

FSM Finite State Machine. 57, 62, 63, 71, 72, 79, 93, 95

FT Fault Tree. 31, 35, 53, 55, 57, 58, 66, 67, 72, 73, 80, 82, 87, 88, 89, 90, 91, 92, 93, 97, 98, 99, 100, 103, 104, 105, 106, 107, 108, 109, 110, 112, 113, 114, 146

FTA Fault Tree Analysis. 37, 50, 53, 54, 55, 57, 60, 66, 67, 74, 93, 97, 107, 112

FTS Flight Termination System. 97, 100

GNC Guidance, Navigation and Control. 78, 79, 83, 95

GPS Global Positioning System. 104

GS Ground Station. 34, 93, 102, 103, 104, 105, 107, 110, 138, 139, 140, 141, 142, 143, 144, 146

GUI Graphical User Interface. 33, 69

Hip-HOPS Hierarchically Performed Hazard Origin & Propagation Studies. 57

IEEE Institute of Electrical and Electronics Engineers. 38

IMU Internal Measurement Unit. 94

INCOSE International Council on Systems Engineering. 38, 40, 44, 45, 48, 49

ISHM Integrated System Health Management. 59, 60

ISO International Organization for Standardization. 40

LNA Low Noise Amplifier. 104

MBD Model-Based Diagnosis. 29, 30, 35, 37, 59, 60, 61, 63, 66, 67, 115

MBDA Model-Based Dependability Analysis. 55, 56, 66, 71

MBSA Model-Based Safety Analysis. 28, 31, 34, 35, 37, 51, 53, 56, 57, 58, 59, 60, 65, 66, 67, 68, 73, 76, 78, 87, 92, 112

MBSE Model-Based Systems Engineering. 28, 29, 31, 34, 35, 37, 38, 44, 45, 47, 50, 51, 59, 65, 66, 67, 68, 73, 76, 78, 87, 92, 112

MC Minimal Cutset. 80

MDB Model-Based Design. 33

ML Machine Learning. 27

NASA National Aeronautics and Space Administration. 26, 38

NPCs Non-Player Characters. 31

OBC On-Board Computer. 85

OC Operations Centre. 27, 76, 77, 79, 83

OCM Orbit Control Mode. 79, 83, 85

ODM Operations-Dedicated Model. 25, 26, 28, 30, 31, 33, 35, 40, 44, 50, 55, 58, 63, 67, 68, 69, 70, 71, 72, 73, 74, 76, 77, 78, 83, 85, 87, 88, 89, 90, 91, 92, 93, 95, 96, 97, 102, 107, 108, 109, 110, 112, 113, 114, 115

OMG Object Management Group. 45, 48

ONERA Office National d'Etudes et de Recherches Aérospatiales - ONERA, The French Aerospace Lab. 34

OOSEM Object-Oriented Systems Engineering Method. 48

PMTE Process, Methods, Tools and Environment. 47

PN Petri Net. 30, 37, 58, 61, 62, 63, 66, 72

PS Power Supply. 97, 98, 99, 100

RAMS Reliability Availability Maintainability & Safety. 31, 51, 52, 56, 57, 66, 67, 73, 87, 112, 114

REX Return of EXperience. 31, 34, 102, 109, 110, 113

RF Radio-Frequency. 103, 104, 105, 106, 137, 141, 144, 146

S2C System and Safety Continuity. 34, 57

SA Safety Assessment. 37, 50, 51, 63, 66, 67, 68, 69, 73, 74, 75, 76, 78, 79, 80, 83, 84, 87, 92, 93, 97, 108, 109, 110

SAE Society of Automotive Engineers. 58

SAFe Scaled Agile Framework. 41

SC Spacecraft. 94, 96

SE Systems Engineering. 29, 31, 37, 38, 43, 44, 45, 46, 49, 50, 51, 56, 58, 63, 66, 67, 68, 69, 73, 75, 76, 78, 83, 84, 87, 92, 93, 95, 97, 100, 108, 109, 110

SMD State Machine Diagram. 78, 79, 83, 93, 95, 96, 109

SS Sub-System. 106

STO Safe Torque Off. 102

SysML System Markup Language. 45, 48, 49, 78, 93, 95, 96, 104, 109, 110

TC TeleCommand. 96

TM Telemetry. 80, 85, 94, 95, 97, 102, 104, 105, 144, 146

TMI Telemetry Image. 102, 104

UAV Unmanned Aerial Vehicle. 29, 31, 34, 50, 93, 97, 98, 99, 100, 109, 110

UI User Interface. 28, 30, 85, 87

UML Unified Modelling Language. 48, 68, 71

V&V Verification & Validation. 29

Introduction

The dissertation whose works are presented in this manuscript was hosted by the Institute of Research and Technology (IRT) Saint Exupéry, under the System Consistency & Continuity (S2C) project, within the Methods & Tools for the Development of Complex Systems (MOSC) team. The research laboratory supporting the research was LAAS-CNRS and the System Engineering and Integration (ISI) team. The thesis was initiated and financially supported by Airbus Defence & Space (ADS).

This manuscript addresses issues related to the operation of complex systems, and especially aircrafts and space systems. The problematic addressed is the observed lack of operational models, dedicated to system monitoring and troubleshooting, in support of the operators. The absence of these models results to the absence of assistance to the operators for identifying the source of the presented symptoms, and recovering the system to an operational state.

The aim, towards a solution for the given problematic, is to facilitate the task of operational diagnosis by creating diagnosis-dedicated system models associated to its monitoring capacity, derived from existing system design (architectural, functional behaviour and safety) models. The works of this thesis enable the concurrent design of systems along with their diagnosis tools. The main objective is to improve maintenance and supervision activities' performance (embedding diagnosis), as well as the overall availability of the system.

More specifically, the objective of the research is to define the notion of an Operations-Dedicated Model (ODM), its contents, and the methodology to create it, while providing an auto-diagnostic solution, completed by troubleshooting capabilities. The goal of an ODM is to improve faults' diagnosis during system operation, by making operational troubleshooting activities more efficient.

When it comes to modern-day spacecrafts –such as satellites, their onboard Fault Detection, Isolation and Recovery (FDIR) capabilities are nowadays very advanced. However that is not the case for their respective Earth segments, since they are expected to operate with a lower level of autonomy, whilst on-the-spot intervention for physical repairs is possible. In fact, physical repairs in space operational systems are only possible onboard manned orbiting spacecrafts, such as the International Space Station (ISS).

What is more, ODMs can also serve as a basis for the construction of systems' Digital Twin

(DT) models. ODMs allow the operators to react more quickly and precisely to the raised alarms. By helping them identify malfunctions and correct them in the quickest delays, they avoid, for example, loosing the next satellite pass (can cause loss of Data but not necessarily e.g. programming on another antenna), thus saving precious time and costs. This can increase both the availability and maintainability of the system.

In a larger framework, ODMs are ideally built and connected with the engineering and safety models of the system, in a continuous system design improvement loop. While the utter purpose of ODMs is their exploitation during operations for system diagnostics, they also contribute to the betterment of system architecture and robustness, through the amelioration of its fault detection and mitigation capabilities.

Before diving into the main work of the PhD, we start by studying more in detail the elements that motivated our study in section 1.1, and lay out our research context in section 1.2. We then discuss our contribution, in regards to the current state of the art in section 1.3, before presenting the research methodology pursued throughout the undertaking of the PhD in section 1.4. Finally, in section 1.5 we provide the dissertation's global structure.

1.1 Motivation

According to the National Aeronautics and Space Administration (NASA) 2022 Strategic Plan [14], one of the four strategic goals of the Agency is to "enhance capabilities and operations to catalyze current and future mission success". In addition, and as stated in its previous strategic plan review (published in 2018), the consolidation and improvement of operations consists an important pillar within NASA's long-term objectives. The end goal being to "balance risks across services and activities to provide a safe and reliable infrastructure" [15]. In an effort to reduce costs, while moving towards an interdependence model between facilities, tools and services, NASA aims to a "more focused investment on condition-based maintenance and reliability-centered maintenance".

In line with the effort for the digitalisation of data, tools, services, and their continuity, the European Space Agency (ESA) aims to reduce the recurring cost of operations, and to invest in innovative spacecraft operations solutions, in order to "enable flexible, efficient and high-performant operations concepts". In particular, the agency aspires to "further position Europe as a strong competitor on the world market" –as stated in its *Technology Strategy for Space19+* plan [16].

The significant technological growth we have experienced in the past decades has also favored the development of sophisticated, space tolerant electronic components. In their turn, these components contributed to the increase of memory and processing capabilities onboard the spacecraft. Within a decade, onboard memory capacity has grown from the range of MB (10^6 Bytes) to GB (10^9 Bytes). Onboard processing capabilities are 25 times higher than before [17, 18, 19]. The more systems become component sophisticated, the more the complexity

of maintenance operations increases.

In this era of digital communications, models are becoming prevalent for the development of complex systems, especially when time delays and safety issues can lead to critical risks. Nevertheless, the use of models is currently not extended to the field of operations. Moreover, the needs for system operational support are relatively poorly considered in the design models implementation. When operating a satellite system, operators need to detect and isolate –or even mitigate, any occurred fault in a *very narrow time-frame*. Satellite operators have limited knowledge of the underlying system design, since they are acquainted with the system architecture through several areas of expertise and skills. Their main task is to monitor the health state of the system and detect any deviations (called *symptoms*) of its components in the so-called housekeeping data.

At the same time, FDIR software and hardware solutions have also significantly improved. Memory and processing capacities have greatly increased on-board the spacecrafts, as well as their ability to process large amounts of data. Nevertheless, on-ground operations have not met much change since the dawn of space age. The biggest operational challenge remains, which is operational (in-line) diagnosis. Diagnosis during operations is performed by subsystems experts situated in the various Mission Control and Operations Centres (OCs), as part of their spacecraft health-monitoring and quick response-to-failure tasks. Note that in the last decade Artificial Intelligence (AI), and especially Machine Learning (ML) techniques began to be considered on the ground for planning, failure detection, etc., following the popularity of AI adoption in numerous applications.

Spacecraft operations are complex processes, requiring different levels of expertise by the operators in charge of the system monitoring, in combination with advanced embedded FDIR modules [20]. Major key operational elements include availability, autonomy, robustness and trustworthiness [21, 22, 23]. A fault impeding access to satellite data can cause a major loss of time –hence money, and a risk for the satellite itself. Consequently, the improvement of FDIR techniques on-board the satellite (satellite autonomy), and on the ground (diagnosis by operators) has been a constant challenge, from the beginning of space missions.

Operators' troubleshooting tasks vary depending on the health status of the system at each given moment, as indicated by the registered data. Per contra, if a symptom –or a combination of symptoms, is unknown, i.e. system designers did not model the symptom or foresee the specific failure occurrence, operators are expected to perform troubleshooting, in order to eliminate the error, and restore functions to their nominal state (or the least degraded possible). Troubleshooting consists of the tasks of requesting for the system for additional information or/and performing manual tests/manual investigation, etc. The operators' priority is to avoid loosing the system services. Thus, unless the anomaly has an associated troubleshooting procedure e.g. in a symptoms' database, the operators must take individual action i.e. carrying out a dedicated analysis and updating the system's knowledge data base.

These actions are usually based on the operators' knowledge and experience. If this knowledge

were to be supported by organized documentation of the system itself (architectural and behavioural, as well as functional and dysfunctional), operational diagnosis could be improved significantly. For this reason we envisaged the creation of an ODM dedicated to Operations and Maintenance system tasks, for Fault Detection and Diagnosis (FDD).

As shown in Figure 1.1, system architecture and behaviour description models –created in Model-Based Systems Engineering (MBSE) activities, along with system dysfunctional behaviour models –created in Model-Based Safety Analysis (MBSA) activities, are being built in the beginning of the system lifecycle, during the system design phase. MBSE and MBSA models are used beyond the system design, particularly in the development and verification system phases, in particular for traceability reasons. The first serves in meeting the set requirements, and the latter in respecting safety objectives while complying with international standards and regulations, which are defined during the system conception phase –mostly the case for the aeronautics domain, not applicable in the aerospace sector (except for satellite End-of-Life management). As illustrated in the schema, the activities related to system design (prior to system launch), as much as operations and maintenance (post-system deployment), are –although related and interdependent, disconnected and detached.

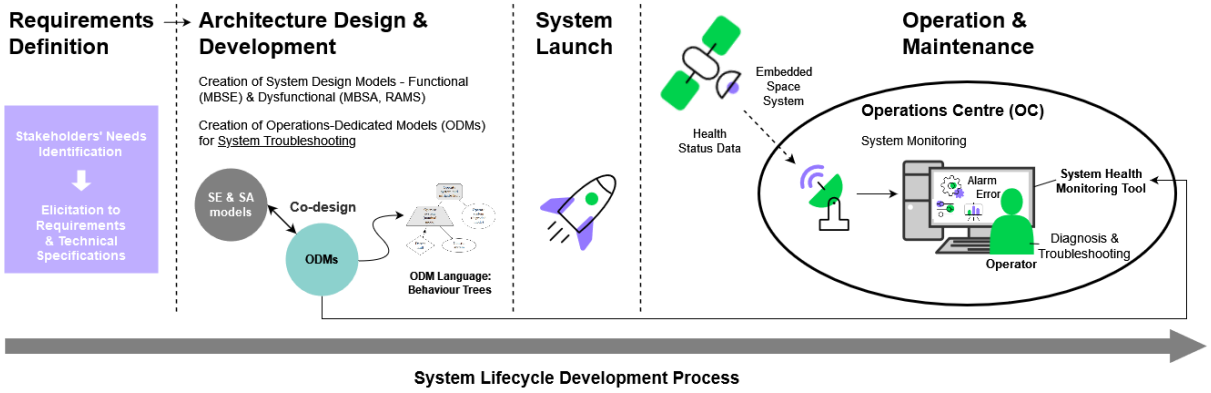


Figure 1.1: Overview of the proposed approach: ODM concurrent creation with system functional and dysfunctional models (during system design). Exploitation during operations and maintenance: imported inside system health monitoring tools, for fault detection, diagnosis and troubleshooting. System lifecycle processes based on [6].

The ODM shall help the operators perform their diagnostics tasks more efficiently by *reducing their response time to failure*, but also facilitate access to documentation and dedicated procedures so as to optimise their troubleshooting actions. Hence the ODM shall have the capability not only to provide the current system overview, but also to be exploited by a diagnosis tool. This way it shall provide possible fault candidates, in the case of a raised alarm, or erroneous received data.

Hence this new type of system model, dedicated to system monitoring, shall be able to be exploited during operations. One way to exploit the ODM is to drive a diagnosis tool, with which the operators could interact, through a dedicated User Interface (UI). Moreover, the ODM can be built during the design phase and along the production of other design models and documents, in a **co-design** manner i.e. through a digital continuity between design, operational safety and ODM construction. Since our main research interest is aerospace, the proposed approach is

illustrated with a space system use case, however it can be extended to other domains, such as airplanes, drones, etc.

Therefore, although the PhD topic is motivated by the needs of the space industry, it remains generic, with a broad scope, within the field of complex systems. During the research work, we discovered that not only space but also UAV systems were subject to similar problems –we hence proposed relevant use cases to apply or methodology to.

1.2 Study Context

Recently the practice of integrating experimental design in system modelling has been being explored. Virtual design in combination with operational data consist the future of SE practise [24], such as DLR's Virtual Satellite project [25] and their Concurrent Engineering Facility (CEF) [26]. Similar activities include the Concurrent Design Facility (CDF) at ESA, ESTEC [27], as well as the the Concurrent Engineering for Ground Segment and Operations Conceptual Design initiative at ESA, ESOC [28]. In addition to national agencies' work towards digitalisation, some commercial tools have started to dominate the field of complex systems' development, such as *Valispace*'s collaborative and open-source concurrent design tool, which can be used throughout the whole system development lifecycle.

Originally introduced as a means to break information silos and reduce Verification & Validation (V&V) costs, by allowing design improvement cycles early on the system development –and not at the integration and testing phase, MBSE is being widely adopted by large agencies and organisations around the world, for a variety of aerospace applications. Notable examples are NASA's Europa Clipper [29] and Mars 2020 [30] missions. The enabling of MBSE in combination with the development of Industry 4.0 inspired the development of the Digital Twin (DT) concept and technology. The past few years, DTs are becoming an indispensable part of model-based systems development. An example being the largest aircraft constructors, Airbus and Boeing, who are both working towards developing DTs for their most recent aircraft [31].

A DT is a virtual representation of a physical product, asset, process, system, or service that allows us to understand, predict, and optimise their performance for better business outcomes [32]. It is essentially a digital version of a system which can also be used for V&V activities. A DT provides an accurate representation of a system's physical structure, logical behavior, physics analytics, its functionality e.g. data processing and communication interfaces [33]. It is used throughout a system's life cycle for evaluating keymetrics and compliances [34]. In the past few years, the use of DTs in a variety of industrial operations is being explored exhaustively. Nevertheless, and as stated [32], "there is still a need to identify its value in industrial operations mainly in production, predictive maintenance, and after-sales services".

The field of Model-Based Diagnosis (MBD) is an auspicious candidate for combining with DTs. The greatest challenges still faced in the MBD world are related to system modelling i.e. how to better and more realistically represent the system in question. Within the context of Discrete

Event Systems (DES), Fault Detection and Diagnosis (FDD) has made significant advancements in the last decades [35] [36]. The most popular solutions consist of Finite State Automata (FSA) [37] [38] [39] or Petri Nets (PNs) [40] [41] variations. With PNs lately dominating the MBD world, they still present major drawbacks and unresolved issues regarding the representation of complex systems, hence their rare application in real-world problems.

On the one hand, approaches dedicated to DES such as [42] are not meant to represent continuous dynamics, and can only do it from a very abstract point of view, which makes them unable to distinguish some types of faults. These limitations are generally due to the fact that the key information for diagnosis is very system dependent, and can be difficult to express in a predefined generic formalism. That is because, in PN- or automata-based methods (PN transitions are similar to event firings in FSA), only the start and finish of each simulated event is considered in the modelling, resulting to neglecting the variable dynamics that take place in between [43]; ergo, DES models implement abstraction to systems' variable dynamics. Note that, although BTs also fall under the DES category, other characteristics, such hierarchy and ease of use, let us to choose them over PNs or FSA. A complete justification is provided further below.

On the other hand, current diagnosis practice based on system simulation tools is limited by the difficulty to integrate dysfunctional knowledge inside the simulation, as well as the very long simulation's execution time that impairs the online diagnosis efficiency. Approaches based on generic models suffer from limited scope (address a small part of the system), or limited precision (suggest too many fault candidates), or both. For example, diagnosis approaches based on automatic control models such as [44] cannot represent discrete dynamics, so their scope is limited.

To summarise, automated diagnosis tools are limited in scope or precision or computation resources, which is why we need operators to perform manual system-wide diagnosis. Moreover, these tools lack the capacity to introduce physical monitors (physical impracticality, cost, volume of data) which are essential for an automatic diagnosis. Our ODM shall hence support manual operations –rather than computations, and encompass the whole system. Nevertheless, it is important to keep in mind that some precise diagnosis operations can be automated, and delegated to intelligent components.

1.3 Contribution

The lack of operations-dedicated models that take into account system functional and dysfunctional data, created during the system design phase, that can be easily exploited by operators (not system architects nor computer scientists), created the need for this new type of model the definition and creation of which we were called to investigate. The end-goal is for this model to be integrated inside a monitoring tool, to provide the operators a dedicated User Interface (UI) which can help them with operational anomalies identification. This will allow them to restore the system in less time –than currently, where no operations-models are available. Furthermore, health-monitoring data received during the system's operation and lessons learnt

from operating and maintaining the system i.e. Return of EXperience (REX), can be used to feed Digital Twin (DT) models. Following the digital data continuity trend of our times, our methodological proposal consists in connecting two usually separated lifecycle stages –system architecture definition and operations and maintenance, by the notion of ODMs, as enabling systems.

As mentioned at the end of the previous section, our proposal is based on the use of the BT language semantics for the creation of ODMs. Although we investigated different pathways towards the ODM construction and associated methodologies, we found them most inadequate with regards to our needs –a further elaboration is presented in Section 3.1. Nevertheless, the approach of **concurrent model design** using BTs [45, 46], revealed to be promising. They were first introduced by D. Isla at the 2005 Game Developers Conference [47] as a tool to model the behavior of Non-Player Characters (NPCs). They have since been extensively used in high-profile video games such as Halo [48] and Bioshock and Spore [49], among others. Later works propose BTs as a multi-mission control framework for Unmanned Aerial Vehicles (UAVs) [50, 51, 52, 53], control [54] and manipulation [55] of complex robots and multi-robot systems. BTs can facilitate system behaviour modelling and simulation, by their intuitive semantics. Hence they can help bridging the gap between system operations and the currently available models, which only include architectural system information, while lacking system behavioural aspects, and that is an important element in order to gather and analyse the system's health status.

Moreover, we explored the use of FTs as the ODM input, as well as explicit system design information, found inside system architecture and nominal behavioural description models. FTs can be produced from system Reliability Availability Maintainability & Safety (RAMS) analyses, or MBSA models. They can also be inherited from a functional analysis encompassed in Failure Modes and Effects Analysis (FMEA), but not always. Therefore, FTs naturally incorporate dysfunctional system information, but also, indirectly, functional information, since they use as input SE data –including SE models. In the latter case we, refer to Model-Based Systems Engineering (MBSE). For this reason FTs constitute a good source of information for the creation of a model dedicated to operational diagnosis.

1.4 Research Methodology

Methodology is the general research strategy with which one chooses to undertake a certain task. A research methodology includes the system of beliefs and philosophical assumptions which shape the understanding of the research questions and underpin the choice of research methods. It is an integral part of a dissertation or thesis which helps to ensure the consistency between chosen tools, techniques and underlying philosophy [56]. Saunders et al. [57] first proposed in 2007 the theoretical concept of the "research onion", as a means of research methodology construction, as shown in Figure 1.2. Since then, the research onion has been being used as a reference for research methodologies description models.

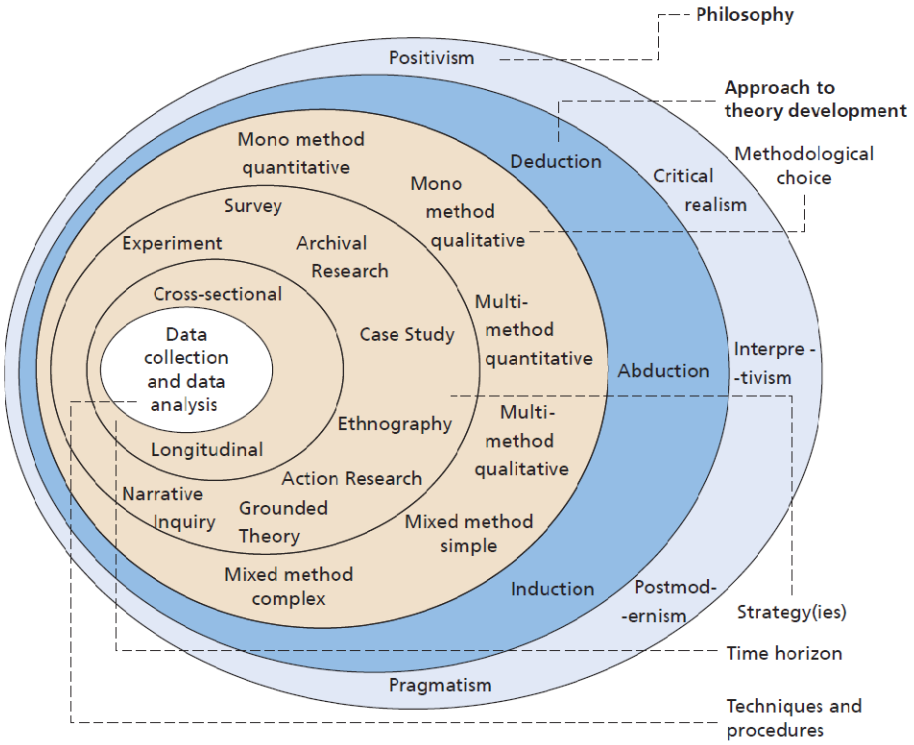


Figure 1.2: The 'research onion'. ©2018 Mark Saunders, Philip Lewis and Adrian Thornhill [57]

The research onion expresses a research process structure that unveils the underlying steps of the undertaken methodology as several layers, from the surface until the depth of the research, starting from the research philosophy, until the techniques and procedures used towards the final results. The intermediate steps by descending order, are the following, throughout the research:

- philosophy –positivism, critical realism, interpretivism, post-modernism, pragmatism;
- approach to theory development –deduction, abduction, induction;
- methodology –mono quantitative, mono qualitative, multi quantitative, multi qualitative, mixed simple, mixed complex;
- strategy(ies) –experiment, survey, archival research, case study, ethography, action research, grounded theory, narrative inquiry;
- time horizon –cross-sectional, longitudinal;
- data collection and data analysis.

For a complete description of the theory, see [57].

Based on the research onion theory, the research methodology I carried out throughout my PhD involves a pragmatic abductive approach to a multi-method qualitative study, continuously corroborated by interdisciplinary and expert feedback, also used as input for the study. The research strategies were: narrative inquiry (industrial and academic partners) combined with

archival research (literature review), and case study applications (complex systems' operational scenarios). We decided to proceed incrementally towards a solution, which consisted of three main steps, described as follows.

First, we studied the initial problematic, which was expressed by our industrial partner, ADS, as the lack of operational models for system monitoring during operations, to facilitate fault diagnosis. We hence had to investigate the nature of the model that would help fill this gap. We initially questioned the type of system information this model should contain (structural, behavioural) and the level of data description (mission/operational/functional/logical/physical). After an elaborate study of the state of the art, we realised that research done already in the field of Model-Based Design (MDB), including "Megamodel" approaches, could not provide answers to our problematic.

The second phase consisted in validating ODM language semantics candidates. Each candidate language had to go through several trials:

- analysis of its expressiveness, current use in the state of the art, availability of tools and libraries;
- relevance with respect to the industrial requirements (usability by non programming experts/simplicity of use for operators, relevance to troubleshooting activities, ease of modelling by engineers with no coding background/computer scientists);
- existence of an exploitation plan e.g. Graphical User Interface (GUI) etc.
- their dependency on complex libraries/operating systems e.g. Robot Operating System (ROS).

The third phase consisted in designing a methodology to create and exploit this model. Following our initial proposal regarding the methodological approach for an ODM construction, we needed a use case to which apply the method, so we can illustrate it, while evaluating its applicability on real case scenarios. In order to do that, several activities were interleaved and iterated.

At each step we sought feedback from our industrial partners and strove to stimulate our mutual constructive collaboration, as illustrated in Figure 1.3. We sent them every use case we developed, and actively extracted feedback from them before we implemented any changes, in a sort of an agile process development loop. Our research at each iteration stage consisted of the following tasks:

- identifying which activities we can support with BTs, hence setting ODM requirements;
- finding a case study that can illustrate these activities;
- exploring the capabilities of BT languages through existing libraries;
- inventing extensions to the existing BT libraries to match our needs.

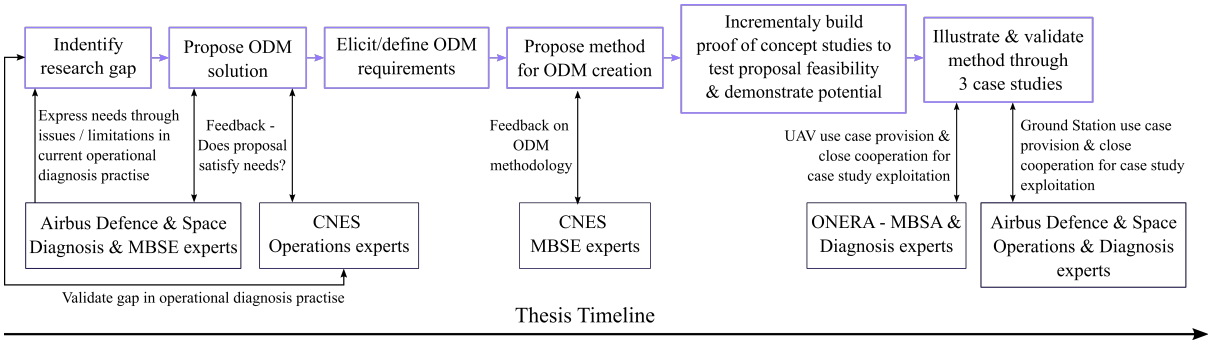


Figure 1.3: Thesis Research Methodology Activities Timeline Overview.

Each time this iteration succeeded, we would implement a use case, evaluate it, and use the evaluation results to further improve our approach. This resulted in 3 different case studies, which reflect three different maturity levels of our approach. The interactions with operational diagnosis experts from ADS and the operations department of the Centre National d'Etudes Spatiales / National Centre for Space Studies - French government space agency (CNES) did not unroll in a structured, but more in an ad-hock manner, as we solicited their feedback when we found necessary. That is, we were in close contact with our ADS partners, and when at some point we needed to have an input from actual satellite operators, we approached certain experts from CNES, in order to get their validation in terms of (a) how relevant our methodological approach is, (b) whether it responds to an actual current need in operations, (c) whether our proposal can indeed meet this need a fill the identified gap in operational diagnosis.

Initially, we were in contact with 5 experts from ADS, as part of the S2C project. Then, I personally contacted a person from CNES, who then put me in contact with 3 additional members of the CNES Operations team, to whom I presented our initial approach proposal, throughout 2 meetings. From these interactions we were able to gather more information on the current state of practise in satellite operations, while also confirming the need for providing a model dedicated to operational diagnosis for the benefit of the operators. When our methodology was more developed, we contacted 2 experts from the CNES MBSE team, with whom we had 2 meetings, and which confirmed the validity of our approach (qualitatively). For the UAV case study, a colleague from the MBSA and Diagnosis team of Office National d'Etudes et de Recherches Aérospatiales - ONERA, The French Aerospace Lab (ONERA) provided us with the use case. Then my Diagnosis supervisor, who works at ONERA and was involved in the UAV project, provided assistance throughout the case study evolution. Lastly, the final use case was provided by ADS, which consisted the result of a Return of EXperience (REX) by Satellite Ground Station operators. Throughout the case study evolution, 2 operational diagnosis experts were working closely with us, while we also had the assistance of an ADS Ground Station operator, to answer our questions related to the Ground Station architecture and operational scenarios.

Regarding the case studies, we first decided to illustrate our proposed approach with a realistic drone mission case, provided by ONERA, to show the versatility of the method for application on various use cases (not only space systems). We then used a Ground Station (GS) use case

that was partly provided by ADS, to demonstrate the application of our proposed methodology to the entire lifecycle of a system definition through models, with ODMs as the ultimate produced artifact. This concluded our research activities on the provided problematic, since we found an adequate solution, that also uses the currently available industrial tools and applied methodologies for the ODM creation, making our solution feasible and realistic.

Hither we would particularly like to thank Gerard Galet and Nathalie Corcoral from CNES for their invaluable support, advice and feedback they have provided to our work, and all the time they have kindly dedicated to our project.

1.5 Report Organisation

This dissertation manuscript is divided into five chapters. Chapter 2 considers the state of the art in related fields and their limitations, which motivate the topic. Chapter 3 presents the proposed methodology for the definition and creation of ODMs from system design model data. Chapter 4 illustrates the global ODM concept through its application on various case studies. Chapter 5 contemplates conclusions and proposes future work opportunities.

More specifically, chapter 2 introduces the notions of MBSE, MBSA, MBD, and reviews literature in each domain, with a special focus on FTs and BTs. We identify related issues and seek responses, resulting to the realisation that the current state of the art cannot provide that corresponds to our needs.

We hence propose a methodology as a response to the given problematic, presented in Chapter 3. The method is illustrated through its application in various use case scenarios in Chapter 4, which demonstrate its usability, as well as its potential. Evaluation of the method has been conducted by feedback from the expert community.

Chapter 5 recalls the research objectives and highlights the results achieved for all the objectives. It also discusses the findings and addresses the contributions to knowledge and practices in Operational Diagnosis. In addition, it underlines future work towards the novel evolution of methods and solutions. The articles published during the PhD research work are listed in Publications.

State of the art

In the past decades, the thriving of information technology favoured the emergence of formal methods [58, 59] and model-based trends for complex systems development –for the functional as much as the dysfunctional definition of the system structure and behaviour, such as MBSE and MBSA. This promoted the conception of system-specific, standardised modelling languages, along with their accompanying methods and tools.

Model-based approaches were also developed within the field of operational diagnosis, giving birth to the domain of MBD. Although the use of models to support automatic diagnosis is still an ongoing research topic, some interesting approaches are already being applied in industrial contexts.

Widely proven identified benefits related to the use of models for system development and operation include reduced risk in the development process, improved cost management, and improved design decisions. This chapter reviews the advancements in the fields of MBSE, MBSA and MBD, with an extra effort done to describe the most relevant concepts to the thesis topic and objectives.

The structure of this chapter is as follows. First, sections 2.1 to 2.3 focus on design tools and methodologies, and discuss how operational diagnosis is addressed (or not) in each of them. More precisely, section 2.1 recalls the essential terminology related to SE, in order to remove ambiguities and specify the definitions on which the study is based. Section 2.2 reviews literature on Dependability, including SA, with special emphasis on FTA, in section 2.2.2, and MBSA, in section 2.3. Then, sections 2.4 to 2.6 review formal languages being used to model systems and automate diagnosis reasoning. Section 2.4 presents a survey on the diagnosis of DES and section 2.5 on common MBD approaches i.e. Finite State Automatas (FSAs) and Petri Nets (PNs). Lastly, section 2.6 provides a literature review on BTs, while section 2.7 provides concluding remarks to the chapter.

2.1 Literature Review on Systems Engineering

In this section, basic concepts related to the discipline of Systems Engineering (SE) and their definitions are presented, with a focus on the aerospace domain i.e. aircraft and spacecraft applications. Representative definitions that get relatively high recognition are given below to offer a global view and understanding of the SE field.

Model-driven SE is an interdisciplinary approach governing the total technical effort required to transform a requirement into a system solution [60]. According to International Council on Systems Engineering (INCOSE), SE is "an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem" [61].

NASA defines SE as "a robust approach to the design, creation, and operation of systems. In simple terms, the approach consists of identification and quantification of system goals, creation of alternative system design concepts, performance of design trades, selection and implementation of the best design, verification that the design is properly built and integrated, and post-implementation assessment of how well the system meets (or met) the goals" [62].

Before we proceed with a literature review on the field of SE, we provide the structure of this section, which is as following. First, subsection 2.1.1 introduces basic definitions of the system notion. It also presents a type of systems which are at the core of our research scope, space systems. Subsection 2.1.2 talks about systems' lifecycle, to explain how a system comes to be from an idea to a physical, operable system. We then focus on a certain domain of SE, MBSE; the methods, tools, processes and languages of which we put into practise throughout this dissertation, in subsection 2.1.3. Subsections 2.1.4 and 2.1.5 tackle these concepts in more detail, providing current prevailing tools, languages and methods in the field of MBSE.

2.1.1 System

According to INCOSE, a system is "an arrangement of parts or elements that together exhibit behaviour or meaning that the individual constituents do not" [63], while the European Cooperation for Space Standardization (ECSS) defines System as "a set of interrelated or interacting functions constituted to achieve a specified objective" [4]. The Institute of Electrical and Electronics Engineers (IEEE) interprets System as "a set of functional elements organized to satisfy user needs" [64].

Is it evident that what all the above definitions have in common, is the notion of the cooperation of individual parts to achieve an externally set objective, implied by the needs of the future users, whom shall benefit from the resulting product of this synergy. As mentioned in the previous chapter, the context of this dissertation is aerospace systems, and more concretely, space systems. According to ECSS' *Glossary of Terms* [4], a space system can consist of four principal segments: *Space*, *Ground*, *Launch* and *Support* [4], as illustrated in Figure 2.1. The space system can be seen from a physical perspective, as for its constituting subsystems and segments, and subsequently, its pieces of equipment and components, as well as from a functional perspective, to include each element's functional behaviour.

A space system can be categorised as a System of Systems (SoS), consisting of several complex systems, starting with the spacecraft (payload and platform systems) i.e. space segment, along with the ground and launch segments.

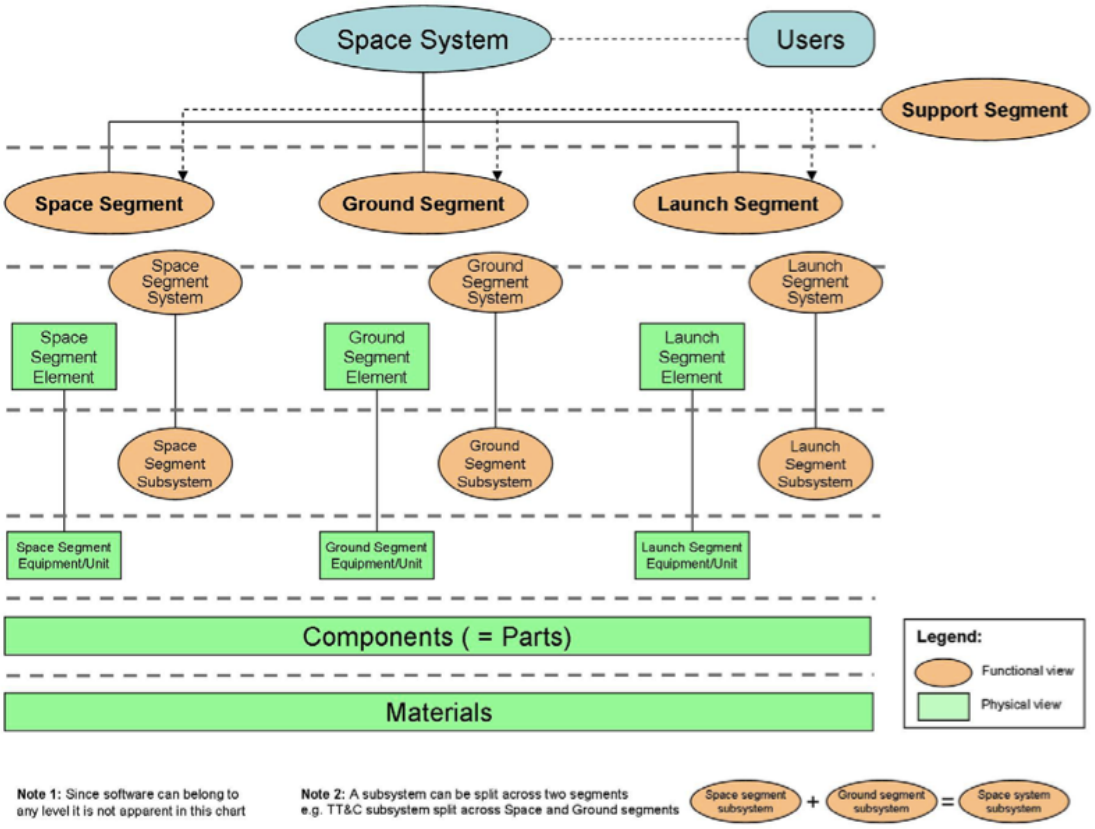


Figure 2.1: Space system architecture overview, based on [4]: breakdown into segments, systems, elements, subsystems, equipment, components and materials.

Ground and launch segments are systems that enable the spacecraft's setting into the correct orbit, as well as their operations and maintenance, as shown in Figure 2.2. The space segment in this illustration is the System-of-Interest (SoI), and a complex system itself, since it is (usually) the responsibility of a single stakeholder, has a single development process, etc.

On the contrary, ground and launch segments do not meet these criteria, hence are not considered as a part of the System-of-Interest (SoI), however can be defined as *enabling systems* of the spacecraft system.

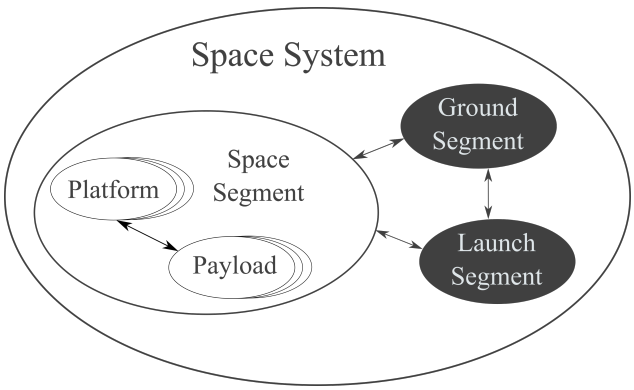


Figure 2.2: Illustration of a space system as an System of Systems (SoS), consisting of the space segment (SoI), ground and launch segment systems.

Nowadays, as the silos between Systems-of-Interest and their enabling systems keep expanding, there is a growing trend towards the concurrent design of Systems-of-Interest and enabling systems, within the worldwide *Digital Continuity* efforts put in place by large companies worldwide. An example is Airbus' Digital Design, Manufacturing & Services (DDMS) initiative [65]. Within this context, ODMs can be seen as the silo-breakers that serve exactly this purpose, as shown in Figure 2.3. That is because ODMs, which are dedicated to the Operations and Maintenance phases, are being being realised concurrently with the System design and Safety analyses. In the subsection below we position these activities in regards to the system lifecycle development.

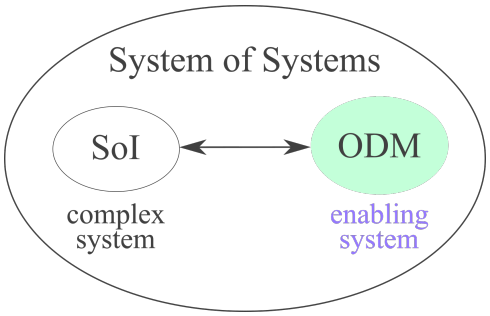


Figure 2.3: ODM as an enabling system of a satellite system; removing silos between system development/realisation and operations and maintenance phase.

2.1.2 System Lifecycle Processes

According to ECSS, a *system lifecycle* includes all phases in the life of a product from needs identification through disposal [4]. There exist different models representing the process of a system lifecycle development. These can be based on traditional project management approaches, including the three most popular methods: *Waterfall*, *PERT* and *Vee*. The latter is the most commonly known model so far, while it is used as the reference lifecycle development process by both INCOSE and ISO. The Vee process model is presented in Figure 2.4. In the space industry, the Vee model is compatible with the sequence of steps that consist the realisation of a space project, therefore it is regularly used as a basis for the conception and development of space systems.

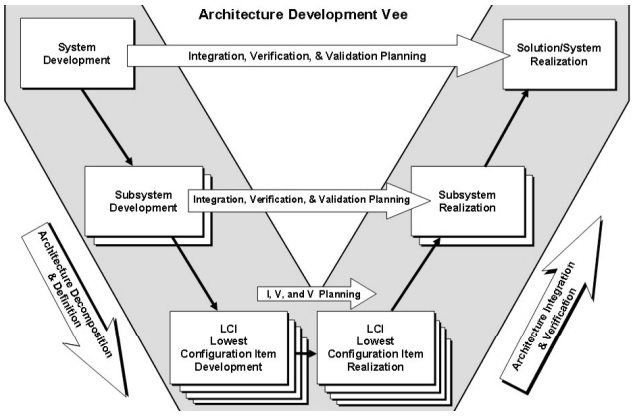


Figure 2.4: Vee Lifecycle Process Model for System Architecture Development. Original definition by Forsberg and Mooz [66].

Nevertheless, there is a big tendency today to move towards Agile techniques for the development of large space projects, once kept within the barriers of the software development community only. For example the *Scaled Agile Framework (SAFe)*, shown in Figure 2.5 is very widely deployed at the moment and put forward by many large space agencies and satellite manufacturing companies, such as ESA and ADS.

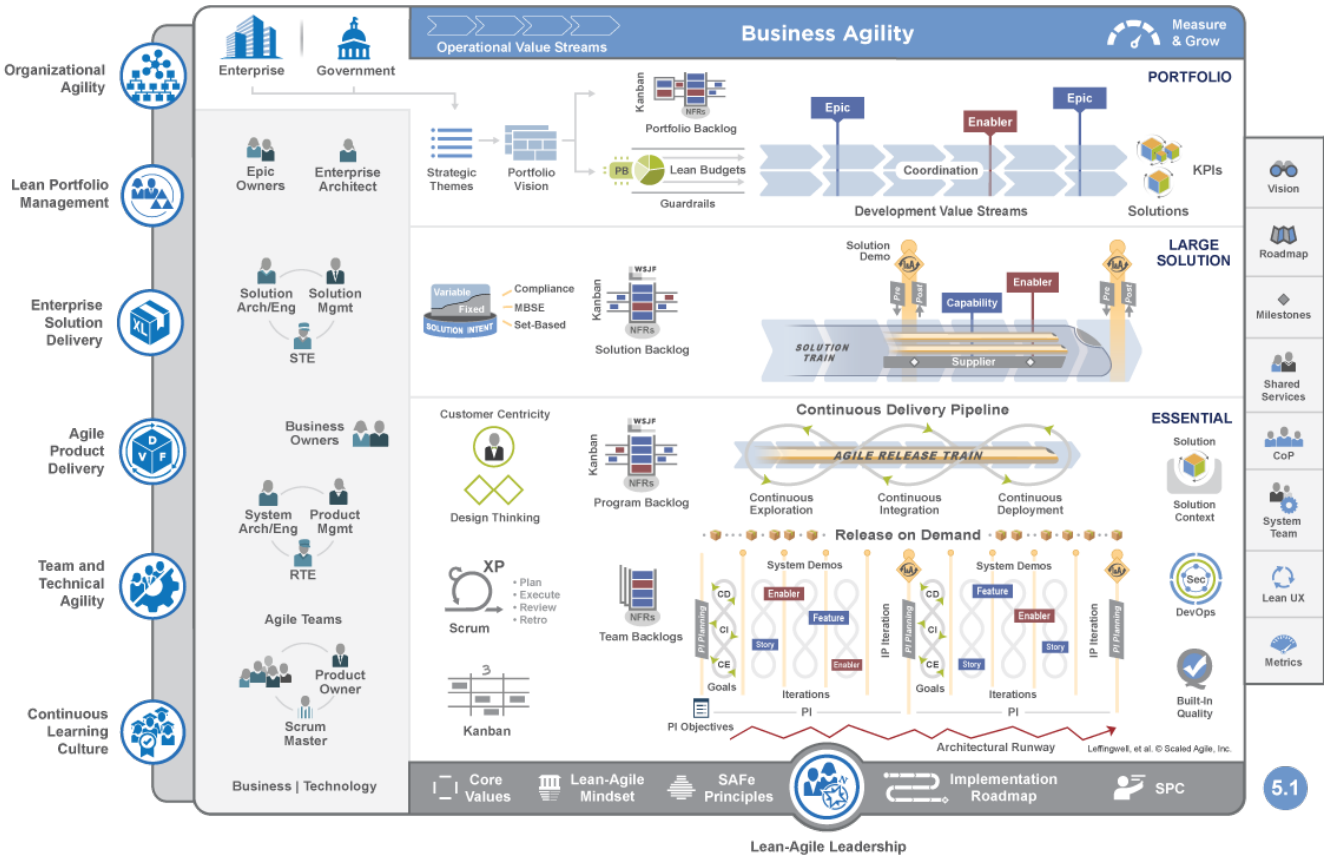


Figure 2.5: Scaled Agile Framework (SAFe). Pushed forward currently by large satellite companies and agencies.

Essentially, agile methods are based on the concept of recurring small development circles, testing and then going forward with the next development step, in contrast with the classical straightforward sequential development approaches, where there is a linear evolution of activities. The most popular methods that employ agile project management for their systems' development, are the following:

- Scrum;
- SAFe;
- EXtreme Programming (XP);
- Rational Unified Process (RUP);
- Lean;
- Feature Driven Development (FDD).

Coming back to the Vee process model, which is a fundamental system development process model, while it describes "the envisioned technical aspect of the project cycle, starting with User needs on the upper left and ending with a User-validated system on the upper right", it was first introduced by Forsberg and Mooz in 1991 [67]. The Vee model is interdisciplinary: at the highest level, it represents repeated applications of definition and decomposition leading to the synthesis of a system design, then repeated applications of integration and verification into the final product, which is validated against system requirements and customer intent [67], as illustrated in Figure 2.6.

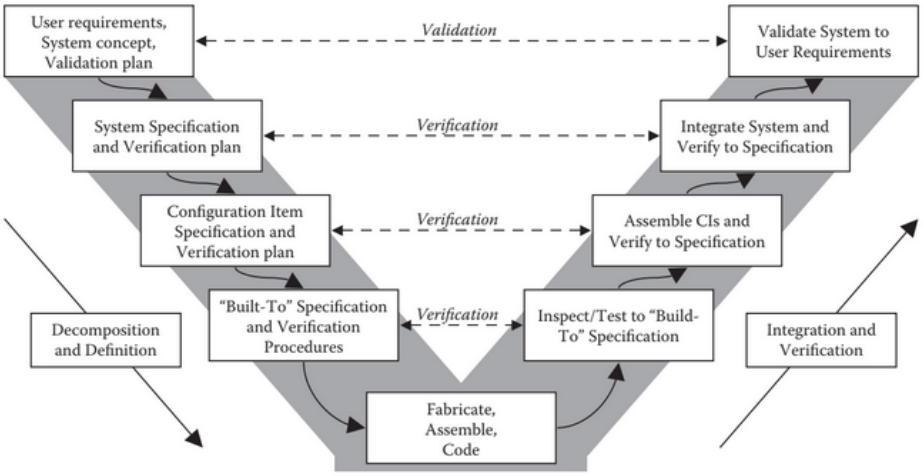


Figure 2.6: Requirements Verification throughout the System Development Lifecycle [67].

On the left side of the Vee model, as shown in Figure 2.7, analysis results in increasing detail of specification, starting with decomposition of a system concept into user requirements, which are an agreement between the customer and the developer as to what will be delivered and what will constitute acceptance of the delivery.

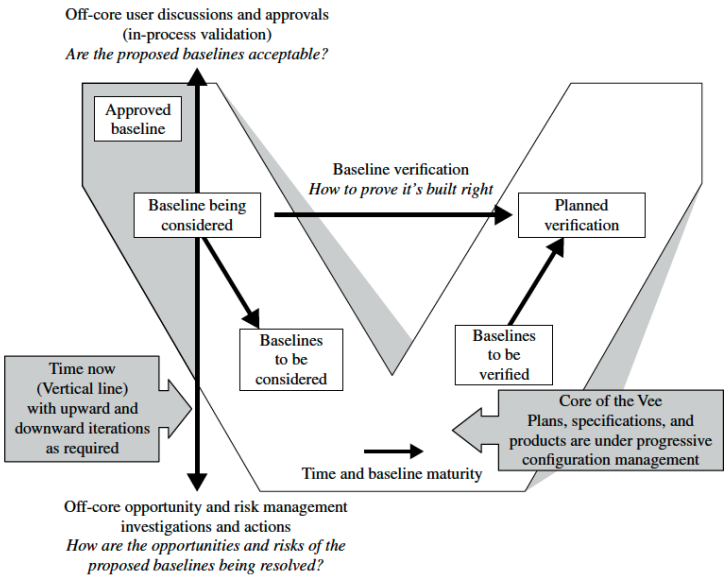


Figure 2.7: Left Side of the Sequential Vee Model [68]. Adapted from Forsberg, Mooz, and Cotterman, 2005 [66].

The analysis process generates system specifications from the concept and requirements, and from those specifications configuration items are specified. The specification process continues in greater level of detail until they are sufficient to generate computer code and be used to either buy parts for the system or build them. Agreement must be reached at every stage of the process and controlled. "Its core involves a sequential progression of plans, specifications, and products that are baselined and put under configuration management. The vertical, two-headed arrow enables projects to perform concurrent opportunity and risk analyses, as well as continuous in-process validation" [68].

SE provides methods and formalism to ease the understanding between the stakeholders and the supplier, and to ensure that "we specify the right system" i.e. the one that is needed. Creating the system requirements is a long and challenging task. User/Stakeholder Requirements are based on the specifications imposed by the client, and are gradually derived by its client's assumed and expressed needs, in the beginning of the system lifecyle. Figure 2.8 illustrates this process, within a SE context.

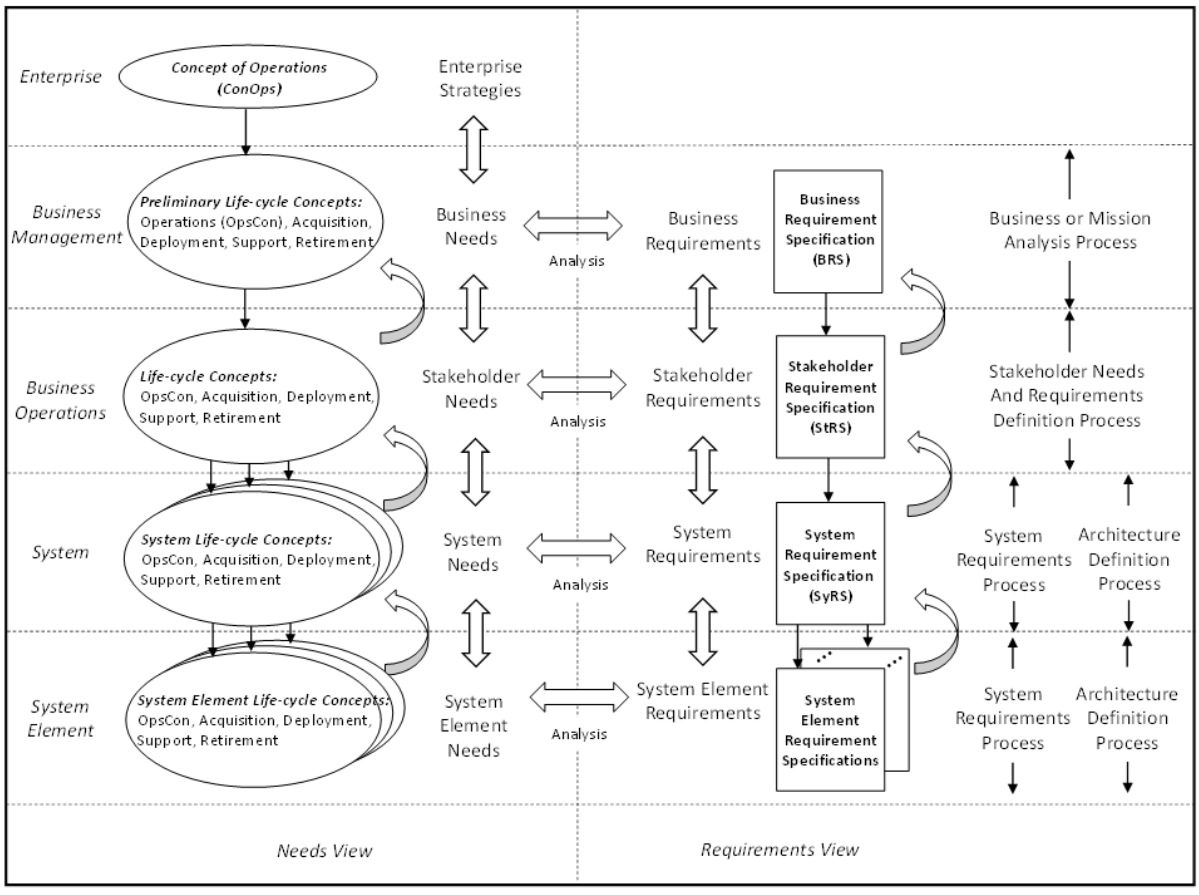


Figure 2.8: Needs elicitation to requirements [69].

The right side of the Vee model is where the repeated applications of integration and verification are performed. The left-hand side of the Vee model is synchronized with the right-hand side via validation plans at each stage of definition and decomposition in the analysis process, as shown in Figure 2.9.

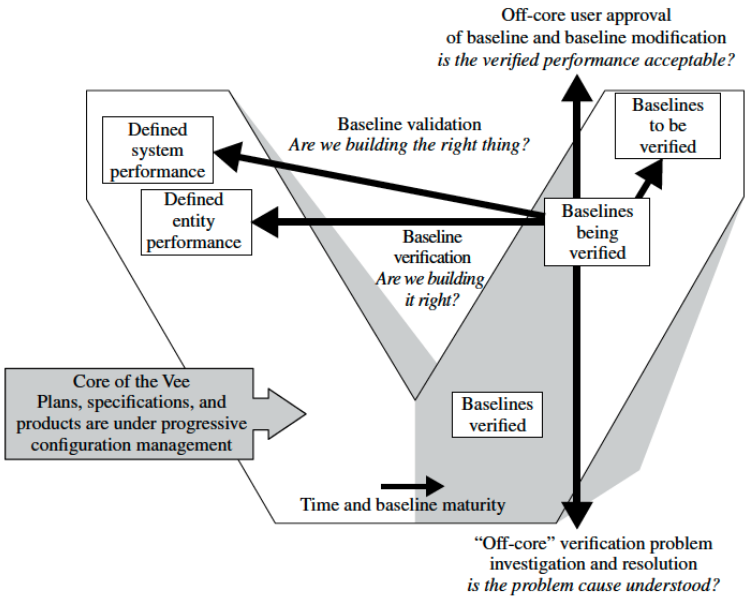


Figure 2.9: Right Side of Vee Model [66].

If we want to place our activities on the Sequential Vee Process Model, it would be on the early stages of system development –the left Vee side: setting technical specifications, high-level design (including system architecture), safety analyses etc. As explained in section 2.1.1, our work towards the definition and creation of an ODM is placed between the system development and operations and maintenance phase, acting as an enabling system of our System-of-Interest (SoI) e.g. the spacecraft. That is because, so far, there is no formal framework in place to connect these three fundamental lifecycle stages i.e. system design/architecture, safety analysis and operations, through a formal validation plan. Certainly, concerning the first two parts, international standards and regulations specifically define how safety analyses should be performed concurrently with system design activities, as for example, ARP4754A and ARP4761 for aircraft design [70, 71]. However, the third part, operations, is not covered.

Through this PhD we attempt to define these connections and introduce a framework for inter-communication and inter-optimisation of each part, through the ODM enabling system.

2.1.3 Advantages of Model-Based SE approaches

With the rapid increase of systems complexity in the past few decades, the use of model-based approaches has become widely spread in industry –including the space industry. In order to deal with system complexity, the SE approach was initially based on a series of documents and multidisciplinary teams which had to store and maintain document consistency in order to work together. In its "System Engineering Vision for 2025" [72], INCOSE proposed to improve existing approaches with models in order to deal with system design complexity including architecture, requirements, interfaces, behaviour and test vectors. Among several, MBSE offers two main interests: facilitating the system development and analyses (system design & dysfunctional analysis), as well as supporting the communication and collaboration between the engineering

teams from different disciplines and various stakeholders. In particular, models are used in SE to represent requirements, functions, architecture, and in dependability assessments to analyse whether the system design exhibits weak points.

Model-Based Systems Engineering (MBSE) allows the transition from the SE requirements management to the actual design of the system, through the modelling of its subsystems and functions. It gives the engineers the opportunity to verify the solution's design, functions and operations early on in the development process, resulting to the reduction of the overall project cost, since this way the alterations are being made in the beginning of the design process and not later on when the solution is furtherly developed.

INCOSE defines MBSE as "the formalized application of modelling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life-cycle phases". MBSE is part of a long-term trend toward model-centric approaches adopted by other engineering disciplines, including mechanical, electrical and software. MBSE is especially expected to replace the document-centric approach that has been practiced by systems engineers in the past and to influence the future practice of SE by being fully integrated into the definition of SE processes [73].

According to the Object Management Group (OMG) [74]: "Applying MBSE is expected to provide significant benefits over the document-centric approach by enhancing productivity and quality, reducing risk, and providing improved communications among the system development team". Modelling has always been an important part of SE to support functional, performance, and other types of engineering analysis. Wayne Wymore introduced a mathematical foundation for MBSE in his book entitled Model-Based Systems Engineering in 1993 [75]. However, the growth in computing technology and the introduction of modelling standards such as SysML [76], UPDM [77], Modelica [78], HLA [79], and others, are helping to enable MBSE as a standard practice, and provide a foundation to integrate diverse models needed to fully specify and analyze systems [80].

Communicating the results of MBSE methodologies' application is facilitated by the use of diagrams. Generally, there are *three types of diagrams*: **Structure** (e.g. *component* diagram, *class* diagram), **Behaviour** (e.g. *activity* diagram, *use case* diagram) and **Interaction** diagrams (e.g. *sequence* diagram, *communication* diagram). In the System Markup Language (SysML) specially, a *requirement* diagram is used, in which requirements and the relations between them and their relationship to other model elements are shown [81].

In this dissertation we assume the use of MBSE techniques, so as to contribute to the larger scale Digital Transformation research work. Below we introduce some basic concepts in the field of MBSE, for the ease of understanding of the following chapters.

2.1.4 MBSE generic concepts

For a successful implementation of an MBSE approach, it is imperative that the modelling activities support the SE processes that occur within the lifecycle stages described in 2.1.2. In order to achieve such implementation, a robust modelling methodology consisting of various "processes", "methods" and "tools" is essential such that it could support SE in a model-based context. The usage of the aforementioned terminologies has not been precise for a long time and these semantic discrepancies have implications on the enabling of any technology in an engineering environment. A survey study showed that SE consensus between the academic and industrial world is still very heterogeneous [82]. Although the study did not survey the usage of the above mentioned terms, it is critical to define these terms beforehand to avoid ambiguity. In [83], Estefan, J.A. and L. Delligatti in [84], use the following definitions to distinguish these key terminologies:

- A **Modelling Language** (L) is a semiformal language that defines the kinds of elements one is allowed to put into their model, the allowable relationships between them, and —in the case of a graphical modelling language –the set of notations one can use to display the elements and relationships on diagrams [83]; note: depending the field of application, a dedicated ontology is needed in order to fully define the model.

- A **Process (P)** is a logical sequence of tasks performed to achieve a particular objective. A process defines "WHAT" is to be done, without specifying "HOW" each task is performed. The structure of a process provides several levels of aggregation to allow analysis and definition to be done at various levels of detail to support different decision-making needs [84];

- A **Method** (M) consists of techniques for performing a task, in other words, it defines the "HOW" of each task. At any level, process tasks are performed using methods. However, each method is also a process itself, with a sequence of tasks to be performed for that particular method. In other words, the "HOW" at one level of abstraction becomes the "WHAT" at the next lower level [84];

- A **Modelling Tool (T)** is an instrument that, when applied to a particular method, can enhance the efficiency of the task; provided it is applied properly and by somebody with proper skills and training. The purpose of a tool should be to facilitate the accomplishment of the "HOWs." In a broader sense, a tool enhances the "WHAT" and the "HOW." Most tools used to support SE are computer- or software-based, which also known as Computer Aided Engineering (CAE) tools [84].

A Methodology is a combination of processes, methods and tools that are applied to a class of common problems. The supporting environment is also associated with these definitions. An Environment (E) consists of the surroundings, the external objects, conditions or factors that influence the actions of an object, individual person or group. The environment enables the "WHAT" and "HOW" [83]. Several environmental factors such as social, cultural, political or

economic have significant implications on the adoption of any technology. Similar challenges also apply for successful adoption of MBSE [15]. The Process, Methods, Tools and Environment (PMTE) diagram in Figure 2.10 shows the relationship between the process, methods, tools and environment ("PMTE" elements) and the effects of technology and people on those elements [85].

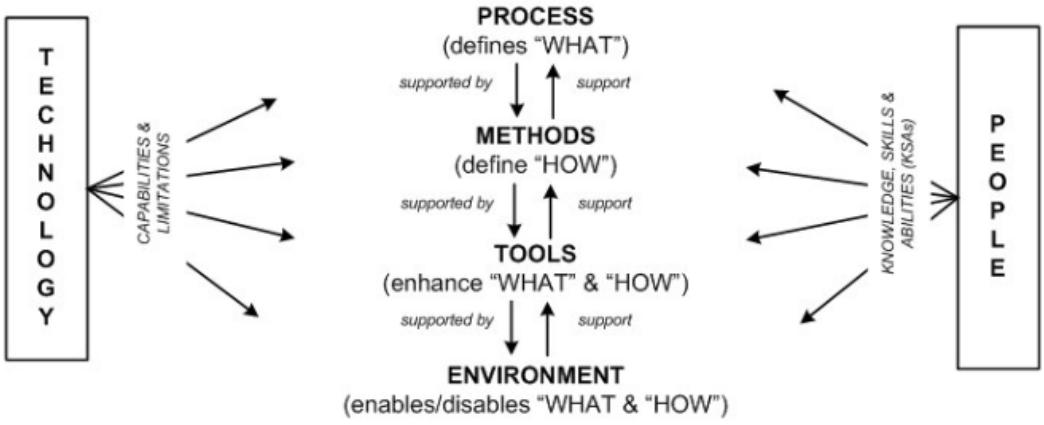


Figure 2.10: The PMTE Elements and Effects of Technology and People.

In any SE development project, the capabilities and limitations of technology must be taken into consideration. Moreover, the knowledge, skills and abilities of the people using the technology must be taken into consideration and employee enhancement initiatives must be undertaken [86]. This argument can also be extended to MBSE. The main purpose of an MBSE environment is to be able to integrate the tools and methods used on a project. Various environmental factors will impact the adoption of new MBSE tools. In order to make a smooth transition to MBSE, management and tool deployment cultural roadblocks should be overcome along with overcoming the existing engineering challenges [87]. In [88], M.E. Sampson points out the management roadblocks that usually hinder SE tool adoption. One of the main challenges is to get the organization management and customer on board with a commitment to allow the systems engineers to successfully use the tools along with proper tool support, maintenance and training.

In the next subsection is presented a short description of modelling languages followed by a list of model-based methods that were developed to meet the needs of the aerospace industry -and their appropriate languages and/or tools. Other less-used model-based methods exist, such as Dori's Object-Process Methodology (OPM), Vitech's MBSE Methodology STRATA, or Weilkiens' Systems Modelling Toolbox (SYSMOD+).

2.1.5 MBSE languages and methodologies

A major part of the ongoing work in the MBSE field is formalization of the design. A formal representation is the foundation for any computer-based interpretation, analysis, or automation. The way we describe systems will define the way engineers will work in the future. One trend surely is the development of modelling languages.

Modelling languages

The most prominent language is SysML™. It is a standardized language that was especially developed to describe complex systems. The System Markup Language (SysML) is based on the Unified Modelling Language (UML) and uses a similar graphical notation. The diagram types allow for defining static properties of the system such as interfaces, relations, hierarchies, and decomposition. In addition, it is possible to describe the system's behavior with several diagram types like the activity diagram, the sequence diagram, or the state machine diagram. There is also a special diagram for the requirements definition [89].

SysML is widely used today, e.g. [90], and the tool support has advanced a lot in recent years. It has reached a state where it is possible to collaboratively work on executable models and generate the necessary artifacts for large distributed simulations fully automatically [91].

However, SysML is not the only language used in the systems modelling world. Different communities so far have created their own formal languages in order to address their particular needs in their field. Some examples for the domain-specific languages are the following:

- Architecture Analysis and Design Language (AADL): Architecture Analysis and Design Language (AADL) is managed by the Society of Automotive Engineers. A dialect of the AADL language was used in European Space Agency (ESA)'s activities on formal system validation (Chapter 14)
- MARTE: A UML profile focusing on modelling and analysis of real-time and embedded systems
- ModelicaML: An extended subset of UML allowing graphical modelling and the generation of executable Modelica simulation code.

The modelling language is not sufficient on its own. It has to be used with numerical tools, as mentioned above. Moreover, system modelling is formalized in specific methodologies by companies e.g. IBM, No Magic; and organizations e.g. INCOSE, OMG; using the adapted tools and language.

Modelling Methodologies

Below is provided a list of modelling methodologies currently used by the academia and the industry worldwide.

- MOFLT Method by Airbus [92] –specific to SysML & Cameo;
- Arcadia Method by Thales [93, 94, 95, 96] –connected (but not specific) to Capella;
- Object-Oriented Systems Engineering Method (OOSEM) by INCOSE [97] –specific to SysML;
- Telelogic Harmony-SE Method by IBM [98] –specific to SysML & Rhapsody;

- Pattern-Based Systems Engineering (PBSE) / Systematica™ Method by INCOSE [99];
- NASA JPL State Analysis Methodology [100, 83] –specific to SysML.

In order to be consistent with ADS' working methods, the MOFLT method was used as a reference for the thesis work. This implies the use of SysML and Cameo for system modelling. As a general framework, MOFLT provides a system modelling method, compliant with ECSS' standards, and extensible with tailoring points. It relies on 5 layers of SE: Mission, Operational architecture, Functional architecture, Logical architecture, and Technical (physical) architecture. An overview of the methodology is shown in Figure 2.11.

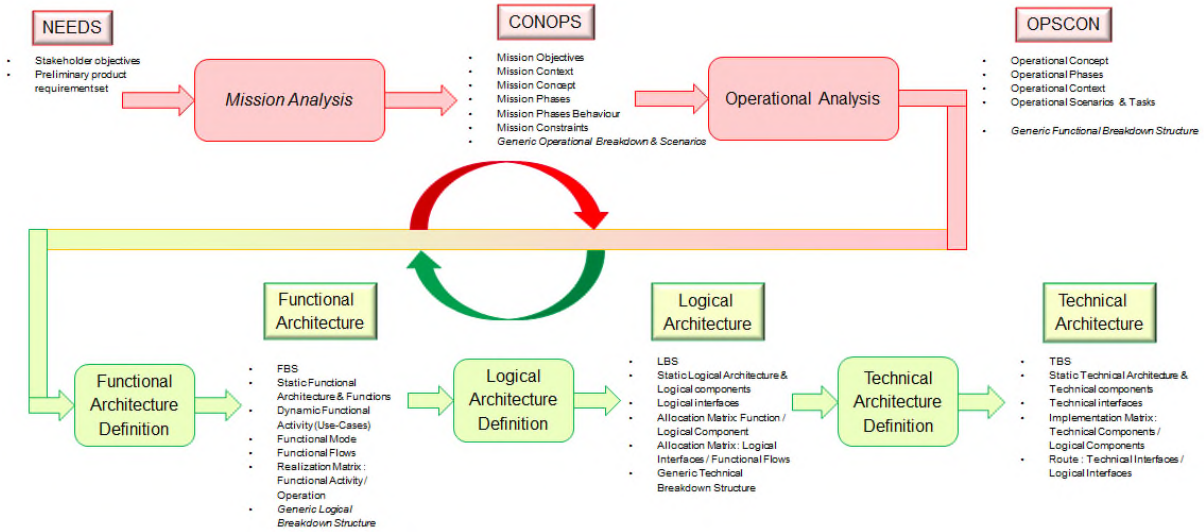


Figure 2.11: Overview of the modelling layers supported by Airbus' MOFLT MBSE Framework.

The approach is summarized as follows:

- Step 1 - Mission analysis: what is the problem we need to solve and what are the potential ways of solving it?
  - Definition of the mission: objectives and effects;
  - Determine potential ways of realizing a mission (mission concept);
- Step 2 - Operational analysis: what will the System of Systems (SoS) do to contribute to the mission? What is the context of the System of Systems (SoS)?
  - Definition of operational concept on an entity: context, constraint, System of Systems (SoS);
  - Definition of operational scenarios consistent with mission concept;
- Step 3 - Functional architecture: how will the System of Systems (SoS) work to meet the expectations?
  - Definition of execution sequence between functions to realize operations;

  - Definition of structural arrangement of functions & interfaces;

- Step 4 - Logical architecture: how is the System of Systems (SoS) organized (abstract component)?
  - Definition of logical components and logical interfaces;
  - Allocation of function to logical components;
- Step 5 - Technical (physical) architecture: how will the System of Systems (SoS) be implemented?
  - Definition of technical (physical) components and technical (physical) interfaces:
  - Realization of logical components;
  - Realization of logical interfaces by technical components.

There is an on-going work within AIRBUS to associate MOFLT steps to SysML objects. This is done by extending the standard SysML stereotypes to match the language stereotypes defined by MOFLT. Thus they can extend the users' method comprehension while facilitating MOFLT's application for the conception and design of aerospace systems, with the help of a dedicated tool. The tool used for the MOFLT method demonstration is Cameo Systems Modeller [101].

Part of the work of this dissertation is to develop a methodological aspect for the creation of ODMs. Although MOFLT was used as a methodological reference for the design of complex aerospace systems, the methodological approach we propose is by all means generic and can be integrated in all other MBSE system development methods.

Following this literature review on model-based system engineering, we attempt below to present with the same level of detail a current state of the art on system safety analyses approaches, tendencies and practises, including model-based safety assessment techniques, that are at the centre of interest of this work.

2.2 Literature Review on Dependability

Along with SE and MBSE, SA techniques are also used extensively during the design of time and safety critical systems. These techniques are traditionally based on both informal and formal design models, collections of documents, such as technical requirements [102] and a variety of dedicated methods, as we will discuss below. In the approach developed throughout the thesis we specifically focus on Fault Tree Analysis (FTA), which is a methodology being widely and successfully used in many industrial applications, such as aeronautics, UAVs, and aerospace systems [70, 71].

With the focus on the operations and maintenance phases, we concentrate on the diagnosis activities conducted by operators. In order to perform diagnosis, it is imperative for them to have at their disposal essential information regarding the system architecture, as well as

the possible failures that might occur during operation. The former is produced by system architects; the latter by Reliability Availability Maintainability & Safety (RAMS) experts, during SA activities.

Operators, in order to obtain this information, must go through the SE documents and models created during the SE activities. They also must consult the artifacts produced by SA activities e.g. *Failure Modes Effects and Criticality Analysis (FMECA)* [103], *Failure Logic Modelling (FLM)* [104, 105], *Failure Propagation Modelling (FPM)* [71], or *Fault Tree Analysis (FTA)* [106, 107]. These artifacts would consist of tables and documents, as well as SA models –in which case we refer to Model-Based Safety Analysis (MBSA) [108].

In order to ensure the long lifetime of the product, a system needs not only be functional, but also "**dependable**". While MBSE allows fulfilling the customer's (functional) requirements through system architecture, one needs also be adept in demonstrating its:

- dependability aspects e.g. through the use of models, namely MBSA, and
- operational aspects, to ensure maintaining the product in proper condition from its deployment until the product's End-of-Life.

System safety became a concern in the late 1940's and was defined as a separate discipline in the late 1950's [109, 110]. It is a sub-discipline of SE that involves the application of scientific, management, and engineering principles to ensure adequate safety within the constraints of operational effectiveness, time, and cost throughout the system lifecycle [111]. In the field of systems, the safety aspect of a system is formally expressed through the concept of Dependability, inside which the notion of safety is included.

The term Dependability is reported to have been first used as a technical term in 1960 by Hosford [112]. Its early usage was confined to the notions of availability and reliability. Since then, dependability has become a wider term encompassing attributes such as safety, security and maintainability [113]. In the early 1980s, Jean-Claude Laprie chose Dependability as the term to encompass studies of fault tolerance, system reliability and dysfunctional analysis, without the extension of meaning inherent in reliability [114].

Within the context of computing systems, Laprie [115] defines Dependability as "the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers. The service delivered by a system is its behaviour as it is perceptible by its user(s); a user is another system (human or physical) which interacts with the former. Depending on the application(s) intended for the system, different emphasis may be put on different facets of dependability, i.e., dependability may be viewed according to different, but complementary, properties, which enable the attributes of dependability to be defined: availability, reliability, safety, confidentiality, integrity, maintainability". Laprie's definition of Dependability is graphically provided in Figure 2.12.

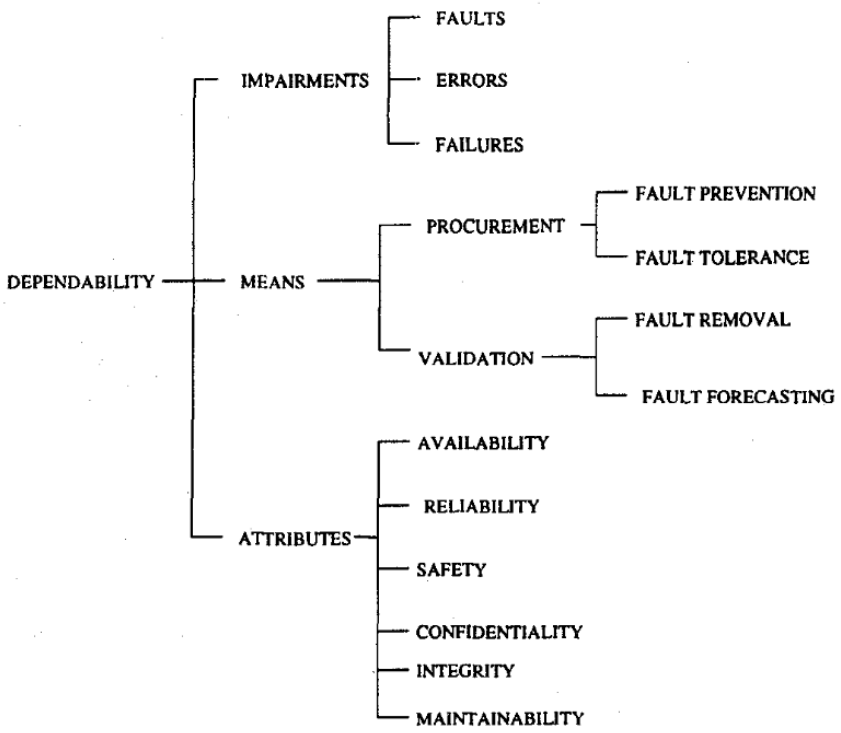


Figure 2.12: Laprie's Dependability Tree - Impairments, Means, Attributes [115].

2.2.1 RAMS

Throughout the industrial community, the term RAMS is widely used to include all the attributes from the above considered essential to describe the Dependability and Safety aspects of a system and ensure its proper functionality.

RAMS stands for **Reliability**, **Availability**, **Maintainability**, **Safety**. The definition of each element is provided below [116].

- Reliability: the system's ability to perform a specific function and may be given as design reliability or operational reliability;
- Availability: the system's ability to keep a functioning state in the given environment;
- Maintainability: the system's ability to be timely and easily maintained (including servicing, inspection and check, repair and/or modification);
- Safety: the system's ability to not harm people, the environment, or any assets during a whole life cycle.

Within the space domain, we mostly focus on Availability, Reliability and Maintainability –except for the launching aspect which contains pure safety aspects and end-of-life management. Nevertheless, given the long lifetime of the systems and their high cost, mitigation means similar to those used for safety are put into practise (even if there is no human harm at stake), i.e. costly mitigation means.

Any sufficiently complex system is subject to failure as a result of one or more subsystems or components failing. The aim of *Dysfunctional analysis techniques* is to understand all the underlying causes (inductive techniques) or consequences (deductive techniques) of a particular failure occurring at component level and observed in system level, so that the likelihood of failure can be reduced through improved system design e.g. different component selection, more stringent development assurance levels and/or via system architectural improvements [106]. Essentially, dysfunctional analyses are performed so as to ensure the Dependability of the final product, before put in operation.

In the following sections, we present two inductive techniques: the widespread Fault Tree Analysis (FTA), and the so-called MBSA (based on the use of models).

2.2.2 Fault-Tree Analysis (FTA)

FTA is a systematic and deductive method which uses a diagrammatic analytical technique for defining a single undesirable event (*fault*, or *failure event*) and determining all possible reasons that could cause that event to occur, including the minimal cut-sets that lead to reaching this failure event. The undesired event constitutes the Top event –or Feared Event (FE), of a Fault Tree (FT) diagram, and represents a complete failure of a product or process. FEs are characterised by their importance of impact to the system itself and its environment (e.g. nature, operators, passengers) and are more commonly divided into five categories –with criticality level from highest to lowest: *catastrophic*, *hazardous*, *major*, *minor*, or *no effect* [71].

The FT segments leading to an FE define all of the things that could go wrong (faults) to cause that particular FE. FT segments more standardly use AND, OR, XOR and VOT(k/n) gates (among the total of: AND, SAND, PAND, OR, SOR, XOR, FDEP, SPARE and VOT(k/n) [117]) to represent fault propagation logic i.e. to express the logical relationship between the *TOP event* and *Basic events* (i.e. lowest level of identified causes; basic initiating faults requiring no further development) [118]. Basic events are issued from a Reliability analysis that would define the basic failure modes of a single component. These failure modes are associated to a failure rate [71].

FTA performs a top-down (deductive) analysis, expressed by the FT, which proceeds through successively more detailed (i.e. lower) levels of the system design, until the probability of occurrence of the FE can be predicted in the context of its environment and operation [106, 119]. The produced FT can eventually provide the list of *minimal cut sets* leading to a specific FE i.e. the FT's Top event. A minimal cut set is a group of sets consisting of the smallest combinations of Basic events which result in the occurrence of the FE [119]. By the term "failure event", we refer to any event mentioned in a FT, including the Top event, Basic events and intermediate events. In order to avoid any terms' confusion in the rest of the manuscript, we provide below the definitions for failure, error and fault.

Definition 1 (Failure) *A* failure *is an event that occurs when the delivered service deviates from correct service.*

Definition 2 (Error) *An* error *is when a system deviates from its correct service state.*

Definition 3 (Fault) *A* fault *is the origin / cause of the error.*

Definition 4 (Active fault) *A fault is active when it produces an error; otherwise, it is dormant. An active fault is caused by either:*

1. *an internal fault that was previously dormant and that has been activated by the computation process or environmental conditions, or*

2. *an external fault that propagates from the environment.*

Definition 5 (Fault activation) Fault activation *is the application of an input (the activation pattern) to a component that causes a dormant fault to become active. Most internal faults cycle between their dormant and active states.*

Definition 6 (Error propagation) Error propagation *within a given component (i.e., internal propagation) is caused by the computation process: an error is successively transformed into other errors. Error propagation from component A to component B that receives service from A (i.e., external propagation) occurs when, through internal propagation, an error reaches the service interface of component A. At this time, service delivered by A to B becomes incorrect, and the ensuing service failure of A appears as an external fault to B and propagates the error into B via its use interface.*

Definition 7 (Service Failure) *A* service failure *occurs when an error is propagated to the service interface and causes the service delivered by the system to deviate from correct service. The failure of a component causes a permanent or transient fault in the system that contains the component. Service failure of a system causes a permanent or transient external fault for the other system(s) that receive service from the given system.*

FTA can be applied for a variety of reasons, within the context of support operations (not engineering activities), such as:

- identifying system failure modes;
- identifying potential design defects and safety hazards;
- simplifying maintenance and troubleshooting;
- identifying root causes during a root cause failure analysis;
- logically eliminate causes for an observed failure, etc.

Moreover, FTA can serve as the means to evaluate potential corrective actions or the impact of design changes [120]. It cannot, however, be used (solely) for operational diagnosis purposes. Note that the system design phase, failure modes are identified by FMECA, and are then implemented in FTA.

FTA can be used either *qualitatively*, or *quantitatively*. For qualitative reasons, to illustrate the build up of states and events, identifying common elements in different branches (dependent failures). In a quantitative way, to give either the absolute probability of existence (for a state) or rate of occurrence (for an event) of the 'consequence', or the relative probabilities of various possible causes given that the 'consequence' has occurred [121], or to calculate the failure rates of the encompassed components.

It is to note that Fault Tree Analysis (FTA) can be conducted for both positive and negative events. On the one hand, the logic tree segments leading to a Negative Event, such as an accident, defines all of the things that could go wrong to cause the negative event. Logic tree segments for negative events usually use more OR gates than AND gates, except for redundant safeguards. On the other hand, the logic tree segment leading to a Positive Event defines all of the things that must work together for the machine to operate or to complete a successful mission. Logic trees for positive events generally use more AND gates than OR gates, except for redundancy. Maintenance troubleshooting trees are a good example of logic trees for positive events. Inverting the output of a positive event converts it into a negative event [106].

Although FTA is a highly successful and widely-used method for dependability analysis of wide variety of systems, it does have a number of limitations, such as an inability to model sequence- or time- dependent dynamic behaviour and to perform quantitative analysis with uncertain failure data. Granting it is mathematically possible, it remains far too complex to implement. In addition to that, even where software tool support exists for FTA, it requires a lot of manual efforts to create and analyse FTs [122].

We can hence conclude that FTs do not fully meet all the ODM needs, since their lack of dynamic and temporal aspects, as well as of any operations- or diagnosis- related system information. However, the fact that they include models of fault propagation and mitigation system mechanisms, make them very good candidates for ODM input data sources.

What is more, as most of the MBDA approaches lack in their capability to perform dynamic dependability analysis, it is believed that it is both theoretically and practically important to explore possible ways to develop expert systems for model based dynamic analysis of systems. It has also been emphasised that more research should be done to develop an expert system by integrating uncertainty quantification approaches with MBDA approaches so that model based dependability analysis of complex systems could be performed automatically under the conditions of uncertainty. Secondly, it has been predicted that the future research may lead to more integrated approach, where different strengths of the existing state-of-the-art MBDA approaches will be used in a complementary manner. Finally, emphasis has been put on to perform further research to empower the existing MBDA techniques with new concepts such as data mining (Reliability analysis), deterministic algorithms (Safety analysis) and machine learning (Availability and Maintainability analysis), so as to make them capable to perform dependability analysis of large and complex open systems. For more information on the applicability of each approach, see Figure 2.15.

2.3 Model-Based Safety Analysis (MBSA)

Model-Based Safety Analysis (MBSA) is a technique which models the system's structure and behavior in order to provide safety analysis results [71]. It can be used not only for Safety analysis, but also for RAMS. In space missions, after the satellite has reached its final operational orbit, the focus is on Availability (system's capability to be kept in a functioning state in the given environment) and Maintainability (system's capability to be timely and easily maintained, including servicing, inspection and check, repair and/or modification). Note that, for a more complete definition, covering the entire notion of Dependability (and not only safety), some works refer to Model-Based Dependability Analysis (MBDA).

In the domain of safety-critical systems, model-based development is an increasingly popular approach for the analysis of digital control systems. In this approach, various dependability related activities are based on formal models that contain system dysfunctional data. In order to perform dysfunctional analysis at system level, it is required to have a fundamental knowledge of (a) the nominal system behavior, limited to the scope and the level of abstraction useful for dysfunctional analysis, in particular the reconfiguration and protection systems defined in the Systems Engineering (SE) model, and (b) the various ways the failures can occur and propagate inside the system [123].

Consequently, MBSA uses a formal model describing both the nominal system behavior and the possible faulty behaviors, to analyse combinations of faults and their consequences in terms of FEs, which affect operational availability (when a critical error occurs, the system is not available until the error is resolved). These models are currently manually built by the safety engineer from the available document-centric specifications of the systems under study. As a consequence, they are hard to build and to maintain throughout the systems' lifecycle. "A small change in the specifications may require revisiting completely the safety models, which is both resource consuming and error prone" [124].

One of the approaches in MBSA, as shown in Figure 2.13, is to extend existing model-based development activities and tools to incorporate the safety analysis activities, call the overloaded, or unique model approach. In this approach, both functional and dysfunctional data are included inside the same model, i.e. "overloaded" model. However, although it contributes to braking certain silos between the system architecture/behaviour definition and the safety analyses, this approach had significant drawbacks.

On the one hand, overloaded models contain an overwhelming amount of information, making them hard to read, understand and manipulate, and eventually unused. On the other hand, safety practitioners insist that a separate model is needed so as to describe the dysfunctional aspects of the system, and to which to perform safety analyses. According to safety experts, an overloaded model cannot reach the same expressiveness in terms of safety, as a dedicated model. The involved disciplines and the skills required by the modellers are significantly different. Moreover, the combination of business and data objectives is not easily negotiable with the

corresponding stakeholders. For this reason a second approach has emerged, that has been being used for the past decade, and which we are adopting in our research approach.

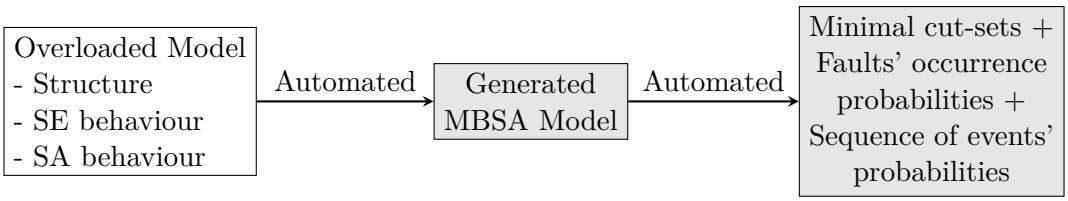


Figure 2.13: Overview of the unique-model approach.

As illustrated in Figure 2.14, the multi-model approach suggests a process by which a separate safety model is created, independently from the functional model. This ensures that the dysfunctional analysis is not mixed with the design models. By result, dysfunctional models represent a rich description and a more complete attempt to perform system RAMS analyses.

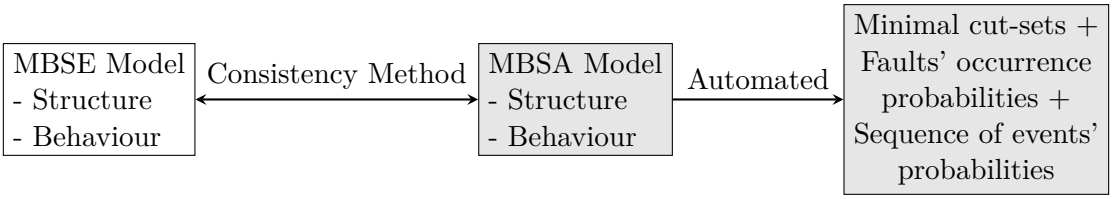


Figure 2.14: Overview of two-model approach. Credits: System and Safety Continuity (S2C) Project.

As opposed to the "traditional" risk modelling methods, such as FMECA, FTA, Event Trees (ETs) or Markov Processes, MBSA aims to provide a precise model of the system behavior and to automate parts of the safety analysis process. The application of MBSA can both reduce the cost and improve the quality of the safety analysis process [125]. Moreover, with MBSA one can calculate the *sequence* of the faults' occurrence probabilities (order of occurrence), which is not possible with the traditional safety analysis methods, that lack temporal aspects.

Some prominent tools and techniques implementing MBSA solutions are the following [122]:

- Failure Propagation and Transformation Notation (FPTN): the first modular and graphical method to specify failure behaviour of systems with complex architectures; FPTN was created to provide a simple and clean notation to reflect the way in which failures within the system interact along with the system architecture [126];
- Hierarchically Performed Hazard Origin & Propagation Studies (Hip-HOPS) [105]: one of the more advanced and well supported compositional model based dependability analysis techniques. It can provide similar functionality to FPTN but with more features and a greater degree of automation. It can also automatically generate FTs, FMEA tables, perform quantitative analysis on FTs, and has the ability to perform multi-objective optimisation of the system models [127]. Moreover, it can semi-automatically allocate safety requirements to the system components in the form of Safety Integrity Levels (SILs).
- AltaRica: high level description language [128] based on Finite State Machines (FSMs) designed to model both functional and failure behaviour of complex systems. It can

represent systems as hierarchies of components and sub-components and model both state and event like State-Event FTs, as well as model analyses supported by external tools and methods, e.g. automatic FTs and Petri Nets (PNs) generation, model-checking etc. [129]. Some known AltaRica-based tools are SimfiaNeo [130], OpenAltaRica and Cecilia OCAS [131].

- Formal Safety Analysis Platform (FSAP)/NuSMV-SA [132]: consists of a set of tools including a graphical user interface tool, FSAP, and an extension of model checking engine NuSMV. Aims to support complex systems' formal analysis and safety assessment. It allows the user to inject particular failure modes into the system (nominal or degraded mode) and observe the effects of that failure on the system behaviour. Its primary goal is system model safety requirements' verification. It is also capable of performing different types of safety analyses e.g. automatic FT generation.

- Architecture Analysis and Design Language (AADL): domain-specific language standardised by the International Society of Automotive Engineers (SAE) for the specification and analysis of hardware and software architectures of performance-critical real-time systems [133]. It has the capability to represent a system as a collection of software components mapped onto an execution platform. It also provides a standard set of component abstractions and supports different types of interaction between components e.g. events and dataflows. Interactions between hardware and software components are defined through binding. The language is extensible through customised "annexes". The AADL Error Model Annex supports fault and failure information specification i.e. allows to annotate original AADL architecture models with failure related information, thus enabling dependability assessments of systems based on these annotated models.

In spite of their advanced system fault modelling and Dependability analyses capacity, the above methods still face important challenges and limitations. Notably they lack in their capability to perform *dynamic* safety analysis and to integrate uncertainty quantification approaches [122]. In addition, these tools and techniques impose new approaches to system modelling, which in turn produce partial, or simplified models; a practise which is prominently error prone. This means that they don't take into account the global system architecture, which is precisely the aim of SE activities. The capacity to model the dynamic (temporal & behavioural) aspects of a system is crucial for an operational diagnosis-dedicated model, therefore MBSA-produced models cannot be used as a basis for the ODMs. Moreover, they include only an incomplete view of the system. That is, only the fault modelling and propagation, their expected result and their probability of occurrence. In order to effectively perform diagnosis during operations, more information is needed about the system, which can only arise from functional system artifacts. However, MBSA models, e.g. AltaRica models, can constitute a useful input for ODMs, if provided in an intuitive manner, i.e. through a visual safety analysis tool. Otherwise, information extraction by the maintenance engineers/operators would be neither easy, nor straightforward, but rather arduous.

2.4 Literature Review on Model-Based Diagnosis (MBD)

The problem of automated fault diagnosis has received considerable attention in the literature, and a wide variety of schemes have been proposed. The property of diagnosability is introduced in the context of the fault diagnosis problem. Failure detection and isolation is an important task in the automatic control of large complex systems. The increasingly stringent requirements on performance and reliability of complex man-made systems have necessitated the development of sophisticated and systematic methods for the timely and accurate diagnosis of system failures [134].

Suppose one is given a formal description of a system, together with an observation of the system's behaviour which conflicts with the way the system is meant to behave. The diagnostic problem is to determine those components of the system which, when assumed to be functioning abnormally, will explain the discrepancy between the observed and correct system behaviour [135].

The goal of diagnosis is to identify the possible causes explaining a set of observed symptoms. A set of concomitant tasks contribute to this goal and the following three tasks are commonly identified [5]:

- **fault detection**, which aims at discriminating normal system states from abnormal ones, i.e. states which result from the presence of a fault,
- **fault isolation**, also called fault localisation, whose goal is to point at the faulty components of the system,
- **fault identification**, whose output is the type of fault and possibly the model of the system impacted by this fault.

Model-Based Diagnosis (MBD) is a systems-specific diagnosis framework developed throughout the last three decades from the computer science community and Artificial Intelligence (AI) in particular. This framework is extremely general and covers a broad range of capabilities including detecting malfunctions, isolating faulty components, handling multiple faults, identifying repair actions, and automatically generating embedded software. This field grew independently of the Fault Detection and Isolation (FDI) and Fault Detection and Diagnosis (FDD) communities, and has developed its own terminologies and conventions [136].

Operational Diagnosis is part of a larger task described in [137] as Integrated System Health Management (ISHM), that includes both fault diagnosis and prognosis. The range of techniques used for system health management is represented in Fig.2.15. Each system component, diagnosis and more generally ISHM can be performed with the most appropriate technique, and generates *health indicators* such as alarms and other signals. At the global system level, the techniques most appropriate are those that integrate well with Model-Based Systems Engineering (MBSE) and MBSA models, i.e. *AI* approaches and *Discrete Event Systems* approaches. These approaches can be implemented over computation tools such as constraint solvers and SAT solvers.

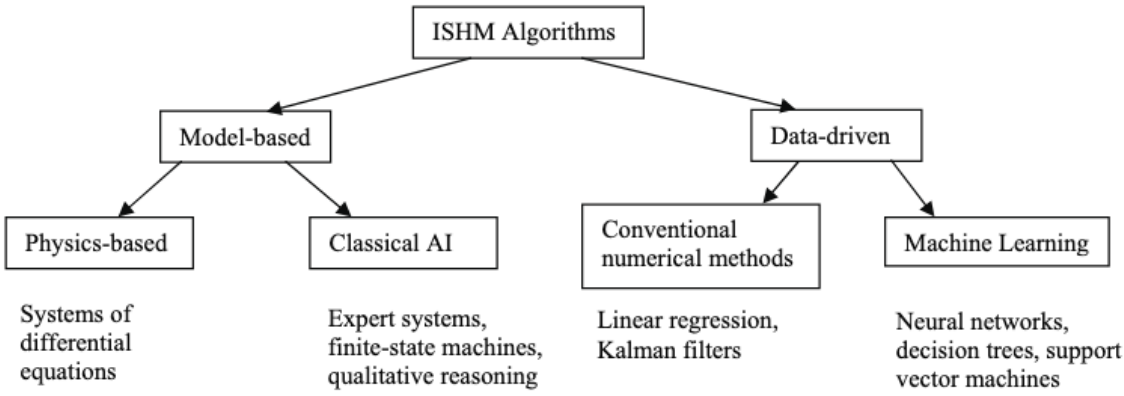


Figure 2.15: Taxonomy of ISHM algorithms [137].

The authors of [137] give Table 2.1 in which the rows represent the four types of algorithms from Fig.2.15, and the columns represent the three ISHM problems described above. In each cell, the authors provide a representative (not exclusive) example of a method that uses the specified type of algorithm to solve the specified problem. Regarding the two empty cells, is because little evidence of current activity in applying the related algorithms to diagnostics, or prognostics, respectively, is provided.

	Fault Detection	**Diagnostics**	**Prognostics**
Physics-based	System Theory		Damage propagation models
AI-model-based	Expert systems	Finite state machines	
Conventional numerical	Linear regression	Logistic Regression	Kalman filters
Machine Learning	Clustering	Decision trees	Neural networks

Table 2.1: An example method for each pair of ISHM problem and algorithm type [137].

As regards to tools, generic monitoring and diagnosis tools exist, both under the form of data-based [138] and model-based tools [139]. System design information is necessary in both cases, in order to interpret the processed data. It is also necessary for building the models that will support fault diagnosis and troubleshooting, which is not always possible. Moreover, diagnosis models fail to include the information produced from safety analysis methods, such as FMECA, FTA, FLM, FPM, or MBSA. A notable exception is [140], but it still assumes that a single model contains both functional and dysfunctional information about the entire system. For all the above reasons we strive to provide more efficient means for the operators to access key system design data.

2.5 Common MBD Approaches

The greatest challenges still faced in the MBD world are related to system modelling i.e. how to better and more realistically represent the system in question. Within the context of Discrete Event Systems (DES), FDD has made significant advancements in the last decades [35] [36]. The

most popular solutions consist of Finite State Automata [37] [38] [39] or Petri Nets (PNs) [40] [41] variations. With PNs lately dominating the MBD world, they still present major drawbacks and unresolved issues regarding the representation of complex systems, hence their limited application in real-world problems.

In this section two of the most common language semantics used for MBD are presented i.e. finite-labelled transition systems. First, Finite State Automata are introduced, followed by Petri Nets. In this domain, Sampath's approach [134] –dating from 1995, is still a reference. It has since been the basis for numerous works in an effort to deal with distributed systems, or ways to achieve better fault description.

2.5.1 Finite State Automata

A finite state automaton is defined by a tuple $G = (Q, E, T, q_0, A)$ where:

- Q is a finite set of states;
- E is a finite set of event;
- $T \subseteq Q \times E \times Q$ is the transition relation. When $(q_1, e, q_2) \in T$, also noted $q_1 \xrightarrow{e} q_2$, this means that the system can evolve from the state q_1 to the state q_2 under the occurrence of the discrete event e.
- $q_0 \in Q$ is the initial state;
- $A \subseteq Q$ is the set of accepting or final states;
- G is deterministic if and only if T is a (partial) function $T : (Q \times E) \to Q$.

In diagnosis, the automaton used for system modelling is often deterministic and accepts a prefix-closed, live language, which means $A = Q$ and every state has an outgoing transition. In some approaches, the system model is described directly as a regular language L_{sys} instead of defining it through an automaton. The set of events is partitioned into the set of *observable* events noted E_o and the set of *unobservable* events noted E_{uo}. The set of fault events, noted E_f , is a subset of E_{uo}. Faults are assumed to be *permanent*. Moreover, the system is supposed to produce observations regularly.

Diagnosis is a function that receives as input a sequence of observations, and produces as output a set of fault events. It is a function $diag : E_o^* \to 2^{E_f}$. The seminal approach to DES diagnosis [134] consists in building a *diagnoser* that implements the diagnosis function as a finite state machine itself. Diagnosers suffer from combinatorial explosion: their size is exponential in the number of system states, which makes them untractable for modern computers (much less for human operators) for all but the smallest systems.

2.5.2 Petri Nets

Petri Nets (PNs) naturally offer component oriented semantics, which makes it easier to model components separately and synchronize models together, in comparison to FSA. PNs are applied to diagnosis techniques by using observable places and transitions. The below definitions can be found in detail in Basile et al., "An Efficient Approach for Online Diagnosis of Discrete Event Systems" [141].

A *Place/Transition* net (P/T net) is a 4-tuple $N = (P, T, Pre, Post)$ where:

- P is a set of m places (represented by circles),
- T is a set of n transitions (represented by empty boxes and each one associated to an event),
- $Pre : P \times T \mapsto \mathbb{N}$ $(Post : P \times T \mapsto \mathbb{N})$ is the *pre-(post-) incidence* matrix.

$Pre(p,t) = \omega (Post(p,t) = \omega)$ means that there is an arc with weight ω from p to t (from t to p); $C = Post - Pre$ is the incidence matrix.

The symbols ${}^\bullet p$, ${}^\bullet t$ and $p^\bullet$, $t^\bullet$ are used for the *pre-set* and *post-set* of a place $p \in P$ (transition $t \in T$), respectively, e.g., ${}^\bullet t = \{\, p \in P \mid Pre(p,t) \neq 0 \,\}$.

A *marking* $m : P \mapsto \mathbb{N}$ is a function that assigns to each place of a net a non-negative integer number of tokens, drawn as black dots.

It is useful to represent the marking of a net with a vector $m \in \mathbb{N}^m$.

A net system $S = \langle N, m_0 \rangle$ is a net N with an initial marking m_0. A transition t is enabled at m iff $m \geq Pre(\bullet, t)$ and this is denoted as $m[t\rangle$.

An enabled transition t may fire, yielding the marking $m^{'} = m + C(\bullet, t)$, and this is denoted as $m[t\rangle m^{'}$.

A firing sequence from m is a sequence of transitions $\sigma = t_1 t_2 \dots t_k$ such that:
$m[t_1\rangle m_1[t_2\rangle m_2 \dots [t_k\rangle m_k$, and this is denoted as $m[\sigma\rangle m_k$.

An enabled sequence σ is denoted as $m[\sigma\rangle$, while $t_i \in \sigma$ denotes that the transition t_i belongs to the sequence σ. The empty sequence is denoted as ν.

A marking $m^{'}$ is said to be reachable from m_0 iff there exists a sequence σ such that $m_0[\sigma\rangle m^{'}$. $R(N, m_0)$ denotes the set of reachable markings of the net system $\langle N, m_0 \rangle$. A PN is a compact representation of a DES. The graph of reachable markings is the FSM representation of the same DES, usually much larger.

As in FSAs diagnosis, the set T can be partitioned into the two disjoint sets of *observable* (represented by empty boxes) and *unobservable* transitions (represented by filled boxes), named respectively T_o and T_{uo}, with $\mid T_{uo} \mid = n_{uo} \leq n$. Fault events can be unobservable, i.e. $T_f \subseteq T_{uo}$,or observable i.e. $T_f \subseteq T_o$, with $\mid T_f \mid = n_f \leq n_{uo}$.

While PNs are in theory an equivalent modelling formalism to FSAs, they are more practical to use, especially when it comes to modelling. For the ODM, the ease of use by operators, who do not necessarily have a computer science background, is a decisive criterion. In reality, PNs have seen few applications outside the field of computer science.

2.6 Literature Review on Behaviour Trees

MBD models have precise semantics, are executable and can be analyzed and verified automatically. However, they lack user-intuitiveness and have very complex semantics. That is, they are not easily created, nor manipulated, by non-computer science experts. We have therefore turned to investigate a new family of models, which seems to offer both the intuitive and executable model aspects, making them noteworthy. They are called BTs, and this section is dedicated to their description.

BTs have shown a lot of potential in the last decade [142, 143], mainly with their application to robotics and AI [51, 144, 145]. Initially developed within the gaming community to replace FSMs with more user friendly models [146], their use could be widened to the field of operational diagnosis to model, implement and monitor the state of complex systems.

According to Colledanchise and Ögren, "a Behaviour Tree is a way to structure the switching between different tasks (assuming that an activity can somehow be broken down into reusable sub-activities called tasks) in an autonomous agent, such as a robot or a virtual entity in a computer game" [45]. According to García et al., a Behaviour Tree is "a mathematical model of plan execution that allows composing tasks in a modular fashion through a set of nodes representing tasks and connections among them" [147].

The underlying formalism and semantics of BTs support top-down elicitation, thus allowing modular modelling. By this it is meant that, throughout the system development, some subsystems can be thoroughly developed and some others remain as "blackboxes", depending on the information available at the time. This way a single BT is easier to elicit, since it can be exploitable at any level of development. The model can hence stay abstract or be developed in detail. Moreover, new knowledge can be integrated in the BT when this knowledge becomes available from the elicitation of other models, namely the SE and SA models.

In this section, we present the classic formulation of BTs as described in [45] by Colledanchise and Ögren, where BTs can be considered as a form of directed tree, where the flow amongst its nodes and edges is sequential. BTs can be confused with Decision Trees but both are fundamentally different. On the one hand, a Decision Tree, in its classical formulation, implements a function from $\mathbb{B}^n$ to $\mathbb{B}$ –where $\mathbb{B}$ is the set of Boolean values $\{0, 1\}$. On the other hand, a BT implements a way to orchestrate the execution of a set of processes (called behaviours), to support both sequential and concurrent compositions. Most BTs libraries use concepts of "Success" and "Failure" of a behaviour to perform conditional branching. We present here the formulation of [45], illustrated in Figure 2.16.

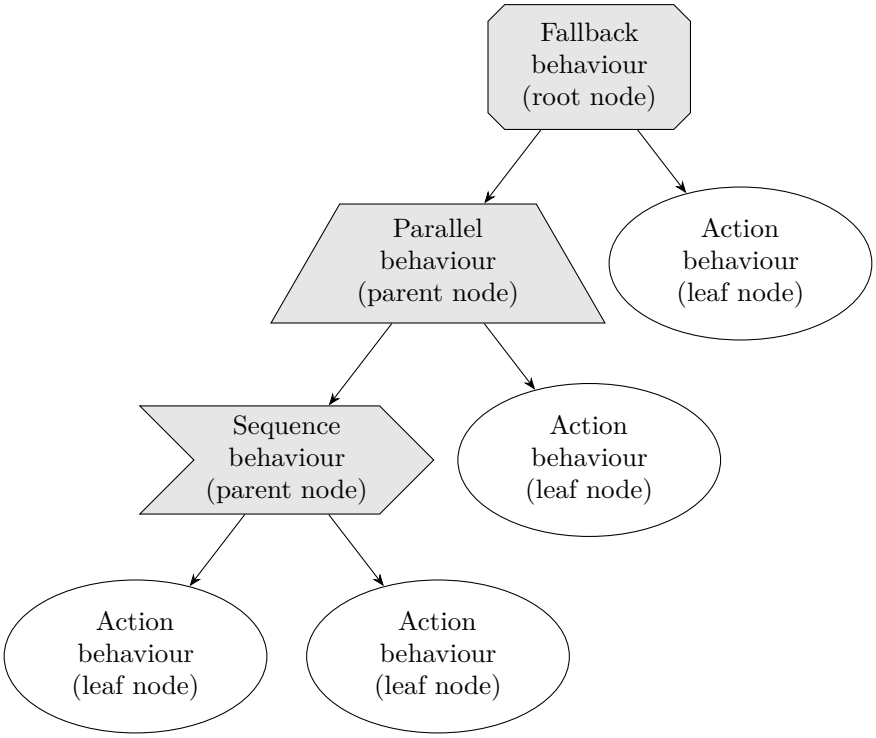


Figure 2.16: Behaviour Trees' basic elements.

A BT represents and implements a way to control the execution of a set of concurrent processes. Each concurrent process is represented by a leaf node. Parent nodes e.g. Fallback, Sequence, Parallel, can start and interrupt their children nodes, and query their status. The BT itself is another process that executes periodically. The root node has no parent nodes, and leaf nodes have no children. For a more complete description, see [45].

During execution, at regular time intervals, the BT "ticks" its nodes. The Root is ticked first, and each composite node ticks some (or all) of its children, in a top-down, left-right execution priority, as shown in Figure 2.17. The returned status of each node can be one of the three: "Success", "Failure", or "Running". Fallback and Sequence nodes tick their children sequentially (from left to right), while Parallel nodes tick all of their children in parallel.

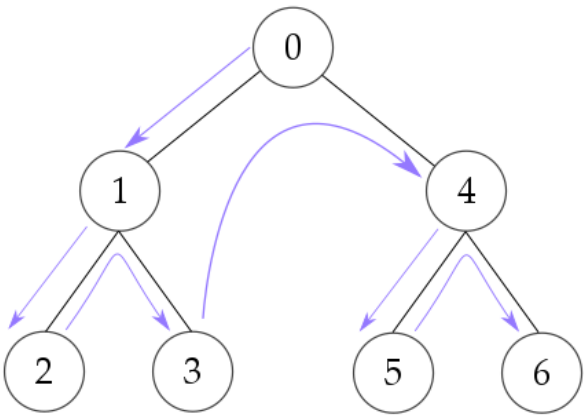


Figure 2.17: BTs' execution sequence.

At each tick, Fallback, Sequence and Parallel nodes return "Running" if and only if their currently active child returned "Running". As illustrated in Table 2.2, upon ticking, a Fallback node returns "Success" if at least one of its children nodes returns "Success", and "Failure" if all of its children return "Failure". On the other hand, a Sequence node returns "Success" if all of its children return "Success", and "Failure" if at least one of its nodes returns "Failure".

	Sequence	Fallback	Parallel	Inverter
On tick	Ticks current child	Ticks current child	Ticks all children	Ticks child
One child returns "Success"	Ticks next child	Returns "Success"	Waits all children	Returns "Failure"
Last child returns "Success"	Returns "Success"	Returns "Success"	Returns "Success"	-
One child returns "Failure"	Returns "Failure"	Ticks next child	Interrupts other children, returns "Failure"	Returns "Success"
Last child returns "Failure"	Returns "Failure"	Returns "Failure"	Returns "Failure"	-
One child returns "Running"	Returns "Running"	Returns "Running"	Returns "Running"	Returns "Running"

Table 2.2: Description of BT Standard Nodes Execution: Sequence, Fallback, Parallel & Inverter.

Parallel nodes launch all of their children in parallel. They return "Success" if all of their children nodes have returned "Success", and "Failure" if at least one of their children nodes have returned "Failure". Inverter nodes return the inverse behaviour status of their child. They return "Success" if their child node has returned "Failure", and "Failure" if their child node has returned "Success". For a more complete description, including our contribution in extending BTs' standard semantics, see Table 3.4.

Different libraries dedicated to BT modelling exist, such as "PyTrees" –python implementation of BTs [148], and "BehaviorTree.CPP" –a BT library in C^{++} [149]. Moreover, there exist Graphical Editors to create BTs, such as Groot [150] (compatible with BehaviorTree.CPP). For the needs of our work we have decided to use the python BT implementation for the ODM construction. Based on our experience, after having tested both PyTrees and BehaviorTree.CPP, we concluded that PyTrees is more adapted to our requirements. We believe that PyTrees is much easier and more intuitive to use than BehaviorTree.CPP, especially for people with no particular coding skills. Hence even without the help of a graphical tool, engineers with no programming background nor development experience (consisting our target group) can build ODMs. That would assume the modellers' accompaniment with appropriate guidance and training.

BTs represent DES in a way that seems promising, in our context of model elicitation by operators. Nevertheless, other representations provided by MBSE, MBSA and scientific research also do exist. To our knowledge, the use of BTs for operational or model-based diagnosis has not so far been explored elsewhere.

2.7 Conclusion

In this chapter we introduced the notions of Systems Engineering (SE), Model-Based Systems Engineering (MBSE), Reliability Availability Maintainability & Safety (RAMS), Dependability, Safety Assessment (SA), Model-Based Safety Analysis (MBSA), Model-Based Diagnosis (MBD) and their interrelation. For the needs of the study, special focus was put on one particular SA method, Fault Trees (FTs). Moreover, the standard semantics of Behaviour Trees (BTs) were presented, since they consist an essential part of the thesis' work.

Section 2.1 introduced the SE approach for systems development (especially model-based techniques), and their advantages on the reduction of system design time and the duration and costs for the system development and production. We also explained how these methods facilitate the elicitation of stakeholders needs into requirements, and their validation. Common languages, methods and tools were also presented, some of which are employed in our research.

Section 2.2 provided a literature review on Dependability and the justification why complex systems nowadays must be Dependable. Two prominent Dependability analysis methods were presented, namely Reliability Availability Maintainability & Safety (RAMS) and Fault Tree Analysis (FTA). The latter is used throughout the PhD work.

Section 2.3 described the notion of Model-Based Dependability Analysis, commonly known as Model-Based Safety Analysis (MBSA). MBSA methods and languages were presented, as well as the the benefits of their application in the development of complex systems –compared to classic safety analysis activities e.g. FTA. In the context of the PhD we assume the use of MBSA in system development, considering the numerous benefits of its application, as well as its prevailing use in modern day complex systems design processes.

Section 2.4 illustrated the concept of system diagnosabilty in the field of Discrete Event Systems (DES) and outlined Model-Based Diagnosis (MBD) approaches. For a complex system to be efficient, it needs not only have optimised design and be robust against failures induced by the environment and from its long operation, but also for these faults to be easily identified and repaired. The definitions of the two most prevalent MBD approaches were also presented, namely Finite State Automatas (FSAs) and Petri Nets (PNs).

Lastly, section 2.6 talked of behaviour trees, which, as we showed, have many qualities among the available modelling languages. Here only the basic semantics, as provided by common literature were provided. This is the last piece of the interdisciplinary PhD topic. Our work on extending BT semantics for the needs of our study are detailed in section 3.4.1.

Next chapter (Chapter 3) will introduce the proposed methodology using illustrated examples, and demonstrate how all the above topics are connected to each other. Additionally, extensive definitions of the standard BT nodes, as well as of the extended nodes that were created throughout the PhD work will be provided.

Methodological Approach

In the previous chapter the fundamental principles in the domains of MBSE, MBSA and MBD were provided, and their relation with the problematic of operations-dedicated models for system supervision and diagnostics was analysed. More specifically, FTA –an SA method, as well as BTs, both fundamental to the ODM methodology, were discussed in more depth. We also explained how none of the prior art is appropriate to solve the problem of how to create operational models from design data, specific for system supervision and fault diagnosis. In this chapter we describe the proposed ODM approach, by employing several operational examples.

In the previous subsections we have reviewed the models produced in current practice, in the fields of MBSE and MBSA. On the one hand, MBSE and MBSA models are intuitive and easily built and understood by engineers who don't necessarily have computer science background. However, they are very large in size and complexity, and include information one may find overwhelming. What is more, they are not made to be executable. In addition to that, they fail to include information necessary for operational diagnosis.

Although SA-produced artifacts –such as FTs, can point to several fault candidates in the case of single feared events, they do not account for multiple feared events occurring simultaneously. Moreover, RAMS analyses are performed based on the feared events identified by the safety analysts. In reality, many unexpected events can occur during system operation, which were not originally identified throughout FTA.

This chapter is organized as follows. First, section 3.1 explains the choice of BTs as the language semantics for the ODM. We justify this choice by providing a comparison with other candidate approaches, currently used in the industry and academia. Then, section 3.2 provides descriptions of (existing) standard BT nodes, as defined through the PhD work. Section 3.3 verifies whether it is possible to express SE/MBSE and SA/MBSA artifacts with BTs. Next we present the proposed approach for the creation of ODMs, in section 3.4. We also provide the definition of the extended BT nodes conceived specifically for the ODM. Finally, section 3.5 concludes the chapter.

3.1 Choosing the most appropriate semantics for the ODM

Within the general industrial trend for the use of models, along with the development of data continuity tools and practices [151, 152], various efforts were made towards the development

of single models consisting of information derived from SE and SA artifacts, and the creation of tangible documents through the system development process [153]. In the context of this dissertation, and as mentioned earlier, we aim to invent a way to create an operations-dedicated model along with system functional and dysfunctional models.

Different strategies can be investigated for the building of an ODM. On the one hand, one can contemplate a direct merging of models. This would mean imposing:

- a common ontology [154] e.g. UML [155], which is expressive enough to satisfy both SE's and SA's needs;
- the same methods and language semantics [156] to be used in the creation of the models manipulated within the two fields.

After complying with the two above perquisites, we can attempt a direct merging of the SE and SA models, to produce an overloaded model. This strategy is however intricate to put in place, since it creates many constraints to both MBSE and MBSA practises. It is also very arduous to impose the same languages and underlying formalisms [157] to different teams, which do not share the same knowledge and objectives, nor identical working methods. Moreover, their workflow would have to re-adjust to the newly defined ontology-based domain modelling methods [158]. This consumes considerable time and effort, thus the expected added value of the direct merging of models strategy has to be compared with the time and effort the strategy requires.

On the other hand, one can intent models' fusion: merging SE meta-models and SA meta-models into a third, global meta-model [159]. In literature, this global meta-model is referred to as a "Mega-Model". A Mega-Model strategy, as introduced by Bezivin et al. [160], is a collection of partially independent models that are loosely synchronized. Some parts are present in several models but not at the same level of detail, thus there must be a coordination model that specifies the relations between the common parts, and tools in order to verify the consistency between the coordinated models.

We decided to discard the Mega-Model strategy, for two main reasons. Firstly, it implies an increased complexity of the required underlying model consistency-checking tools. Secondly, it cannot provide a global solution to several companies, each one of which has in place its own internal methods, tools and practises dedicated to system development, as for example the Capella [96] workbench and its dedicated methodology, Arcadia, for the Thales Group [94], vs MOFLT and Cameo for the Airbus Group.

Our conclusion was that the most suitable approach for the building of an ODM, is to propose a methodology and the language semantics for the concurrent creation of ODMs with SE/MBSE and SA/MBSA models (see *two-model approach*, Figure 2.14, concurrent creation of MBSE and MBSA models using a *models consistency method*), throughout the system design and development phase.

In order to do that, we must choose the most appropriate semantics for expressing the ODM. For this, several languages can be considered suitable. The ability of each language semantics to meet the ODM semantic requirements was used as a criterion for their evaluation. Each criterion was assigned a weight factor, indicating its importance for the ODM creation and exploitation, on a scale of one to three –one having the lowest and three the highest impact, as shown in Table 3.1. The criteria were defined based on the:

- ability of the language to represent specific system elements, notably:
  - hierarchical structural decomposition (component breakdown), as hierarchical modelling capacity is essential for FDD purposes;
  - hierarchical behavioural decomposition / functional description (processes/tasks/activities/functions' breakdown);
  - functional *and* dysfunctional behaviour (how the system behaves in nominal and non-nominal conditions), necessary for operational diagnostics;
  - represent system supervision information / account for feared events, faults, and mitigation means such as troubleshooting and repair i.e. system diagnostic capacity;
- ability of the language to integrate textual information and model data derived by SE and SA activities;
- ability of the language to model operational activities;
- models' executability;
- ability of the executable models to return all system states;
- ease (level of semantic complexity) and intuitiveness of modelling, so as to reduce the skills required by the ODM modeller;
- ease (level of semantic complexity) and intuitiveness of model exploitation, in order to maximise user-friendliness (we consider operators as the end-users);
- maintainability of the created models, in the sense that high semantic complexity increases the difficulty of understanding by future modellers hence making the models hard to maintain;
- ability of the language to support of functional & dysfunctional system analyses, for the amelioration of system design; not part of the comparison since it consists future work, but was taken into consideration;
- ability of the language to be integrated within a diagnostic tool equipped with a GUI; not part of the comparison since it consists future work, but was taken into consideration.

	Business Process Model and Notation (BPMN)	**Finite State Machines** (FSMs)	**Markov Chains** (MCs)	**Petri Nets** (PNs)	**Decision Trees** (DTs)	**Fault Trees** (FTs)	**Behaviour Trees** (BTs)	**Weight** (1-3)
Original Application Field	Human Task Planning	System Logic Modelling	Event Probabilities	Formal Verification	Rational Decision Making	Safety Analysis	Gaming, AI, Robotics	-
Can represent system structure	No	No	No	Yes	No	Yes	Yes	3
Can represent system behaviour	Yes	No	Yes	Yes	No	Yes	Yes	3
Can represent operational sequences & fault mitigation mechanisms	Yes	No	No	Yes	No	No	Yes	3
Models' executability	No	Yes	Yes	Yes	Yes	No	Yes	3
Can return status of all system states	No	Yes	Yes	Yes	No	Yes	Yes	3
Supports hierarchical structural modelling	No	No	No	No	No	No	Yes	3
Supports hierarchical behavioural modelling	Yes	Yes	No	Yes	No	No	Yes	3
Supports functional and dysfunctional input data	No	Yes	Yes	Yes	No	Yes	Yes	3
Can intuitively illustrate current system states (system states overview)	Yes	Yes	No	No	Yes	No	Yes	2
Can integrate system design data	No	Yes	Yes	Yes	Yes	Yes	Yes	2
Low modelling semantic complexity	Yes	No	No	No	Yes	Yes	Yes	2
Level of modelling intuitiveness	High	Medium	Low	Low	High	High	High	2
Level of user understanding intuitiveness	High	High	Low	Low	High	High	High	3
Level of model maintainability	High	Low	Low	Low	High	Low	High	1

Table 3.1: Comparison between candidate language semantics for the creation and exploitation of ODMs.

3.1.1 Language Semantics Comparison

Based on their popularity, extensive usage and numerosity of applications, we have selected seven semantic approaches to evaluate. A brief description of each and an assessment of their potential application to the operation of aerospace systems is to follow.

First we tackle the *Business Process Model and Notation (BPMN)*. BPMN is a standard for business process modelling [161] that provides a graphical notation for specifying business processes. BPMN models comprise of *Business Process Diagrams (BPDs)* [162]. The latter are based on the flowcharting technique –similar to UML Activity Diagrams [163]. The objective of BPMN is to support business processes' management. It also provides a mapping between the graphics of the notation and the underlying constructs of execution languages, particularly *Business Process Execution Language (BPEL)*. BPMN is made to represent processes, and not complex embedded systems. Although it provides a notation that is intuitive to both business and technical users, while being able to represent complex process semantics, it does not meet the ODM objectives. Although with BPMN it is possible to model the behaviour of system processes, it is not possible to model the system structure. The same goes for hierarchy: sub-processes can be modelled inside other processes, but sub-activities cannot be decomposed into sub-activities. Some research work on risk analysis and MBDA has implicated the use of BPMN, but only for visualisation purposes i.e. methodological process modelling, not executable behaviour modelling [164, 165].

We then turn to *Finite State Machines (FSMs)* –otherwise called Finite State Automata (FSA) (see section 2.5.1 for more details). FSMs consist the simplest modelling description of Discrete Event Systemss, by providing a graphical representation of finite relations between the constituting system states. Each FSM can be in exactly one of a finite number of states at any given time t, based on its input and state transition function [166]. Some example model diagrams that use FSMs are *SysML Activity Diagrams* or *Harel Statecharts*. Although FSMs are based on a strong mathematical computation model, they fail to represent system structure and behaviour. Moreover, FSM diagrams can get quite overwhelming when dealing with large models. As mentioned in [167], FSM-based models "involve exhaustive searches or simulation of system behavior and are especially impractical for large and complex systems". What is more, they provide only one way (finite state automaton) to represent a system element e.g. function, component. They hence lack modelling expressiveness. What is more, automata are not good with parallelism and numbers, that cause the number of states to explode.

Defining system structure is possible by using variants of FSMs, such as synchronous state machines. The latter allows the splitting of machines into components, however the tool is difficultly deployed and the system hard to model. Another example of FSM variants are *Markov Chains*. A Markov chain describes a process in which the transition to a state at time $t+1$ depends only on the state at time t. The main difference to FSMs is that the transitions in Markov chains are probabilistic rather than deterministic. Also, Markov chains must be synchronised to be able to represent system structure. Thus they display the same limitations

as FSMs, as regards to ODM design and execution.

Petri Nets (PNs) offer a richer language to describe FSMs. In contrast to FSMs, they offer more convenient means to complex system modelling in that large systems can be represented in a more compact manner. As stated in [167], PNs "offer a much more compact state space than finite automata and are better suited to model systems with repeated structure"; a reason why they are still used in industrial research and development. PNs also allow representing system elements in many different ways, hence providing more modelling freedom than FSMs.

Nevertheless, PNs were shown to be inferior to FSM in terms of response time performance [167], for some problems. Moreover, since PNs do not have a tree structure, there can be multiple shared places and transitions between multiple nets, leading to a potential spaghetti code problem. That is in the sense that tree structures tend to be easily maintainable.

Last but not least, in petri-nets or automata-based methods (PN transitions are similar to event firings in FSA), only the start and finish of each simulated event is considered in the modelling, resulting to neglecting the variable dynamics that take place in between [43]. For all these reasons, we also determined PNs to be unsuitable for ODM construction and exploitation.

Next, we contemplate the use of Decision Trees as the language semantics to create and exploit ODMs. Although Decision Trees have very similar logic to BTs, they present a main disadvantage. That is that they represent stateless functions, i.e. functions whose output at time t depends on their input at time t. It is still possible to use them to represent a transition function, where the Decision Tree input represents the system state at time $t-1$, and the system input at time t, and the Decision Tree output represents the system state and output at time t.

Although Decision Trees are good at expressing complex functions in a compact and explainable manner, they lack the notion of *component*. Their "tree" aspect cannot be used to model a component or function hierarchy, because it is used to represent the different possible choices that a function involves. This assuredly argues that Decision Trees are incompatible with the ODM requirements, and are thus rejected. What is more, attempting to force the notion of state into a DT is not possible. In fact, BTs are known to generalise Decision Trees [168].

Finally we evaluate the sole use of Fault Trees (FTs) as an ODM alternative. FTs are considered as the most used technique in complex systems dependability assessment. Albeit their many advantages, they present many drawbacks when it comes to systems modelling with strong temporal component dependencies. That is, in FTs, there is a structural dependence between components, but this dependence is not dynamic (temporal). Hence the modelled faults are considered to occur asynchronously (asynchronous faults-sequence).

Moreover, as observed in [169], "The assumption of components independence is precisely what makes FTs so powerful, but this assumption is extremely restrictive, and may prove to be totally unrealistic and lead to grossly erroneous results for some kinds of systems. To be able to model component dependencies, one has to recur to dynamic models. The most popular are Markov processes, because of their numerous nice mathematical properties." However, and as

discussed above, we have already concluded that Markov chains do not consist adequate ODM candidates.

In addition to the lack of FTs' dynamic and synchronous properties, FTs fail to support hierarchical modelling and operational sequences representation. What is more, FTs are not easily maintainable, since a small change in the respective SE model can lead to big modification efforts in the FT. That is, unless we consider an automatic solution to generate FTs from design models, by defining a cross matching between an MBSE meta-model, an FT meta-model and a dysfunctional meta-model (failure mode). However, this kind of solutions are not yet mature, while they are based on assumptions regarding the design model structure. For this and all the above reasons, we can safely conclude that BTs consist a more optimal solution, than FTs, for the ODM semantics.

3.1.2 Discussion

Although our inclination towards the use of BTs is justified, the use of other language semantics would add some benefits to our approach. For example, FTs are far more known and popular than BTs in the industrial world, and easily interpreted by non-experts (useful for troubleshooting activities). Other than their familiarity, FTs can model a flexible number of feared events, while they are also applicable to positive events. However, BTs can also model both positive and negative events, and with no limitations to the size of the tree i.e. number of modelled nodes. Still, the industrial world will need to be convinced for adopting a new modelling language.

The main reason why we propose an activity behaviour model as the ODM, and BTs to represent it, is mainly the fact that we must remain, at all times, system agnostic. This eliminates the choice of any domain-specific models e.g. DTs, FSAs, etc. Furthermore, an activity behaviour model helps with coordinating diagnosis activities' actors, since some diagnosis activities are automated, and some other manual.

Furthermore, the ODM is supposed to help operators decide what should their next action be –and elaborate an automatic diagnosis as far as possible, which is a matter of activity, rather than structure, or interactions. Not choosing a structure or interaction diagram –as opposed to a behaviour diagram, assumes having a single "Operator" entity communicating with a single "System" entity. In a system with a more complex structure, initial work is required to narrow down "Operator-system" pairs to which our methodology can be applied.

Incidentally, our methodology proposes a semi-automated model transformation –in contrast to a fully automated approach, considering that some expert knowledge is still required to link all the complementary information coming from the SE activities in the right "place" in the new model. Moreover, using FTs as the transformation starting point means that the system dysfunctional analysis is completed by an an independent team of RAMS/SA/MBSA experts. So by having FTs as input, we ensure receiving a full system FMECA description. This means that we don't need be concerned about fault omissions, neither about introducing wrong dysfunctional elements in the system model. Errors during operational diagnosis, for example due to a bad

BT model, will result in inappropriate actions from the operators. This kind of fault is typically addressed in FTA. In that sense, our methodology does not intervene in the work of SA.

3.2 BT Language Semantics - Standard Nodes

This section provides definitions for standard BT nodes which are used in the ODM methodology, presented in Table 2.2. Although a terms description exists in literature, formal definitions are lacking. We have hence chosen to define our own formal descriptions for BTs as follows.

Definition 8 (Behaviour Tree (BT)) *A* Behaviour Tree (BT) *is a tuple* $\langle \mathcal{B}, root, children \rangle$ *where* $\mathcal{B}$ *is a set of behaviours,* $root \in \mathcal{B}$ *is the root behaviour, and* $children : B \to B^*$ *is a function which associates each behaviour with an ordered (and possibly empty) list of children behaviours. In a BT, each behaviour has exactly one parent –except the root behaviour, which has no parent.*

Figure 3.1 illustrates a BT, in which the root behaviour is named "Operate system and mitigate fault". Its children are named "Operate system (nominal mode)" and "Operate system (degraded mode)"; the former has children behaviours, while the latter is a leaf behaviour. Each behaviour *is composed of* its children –with the exception of leaf/atomic behaviour nodes.

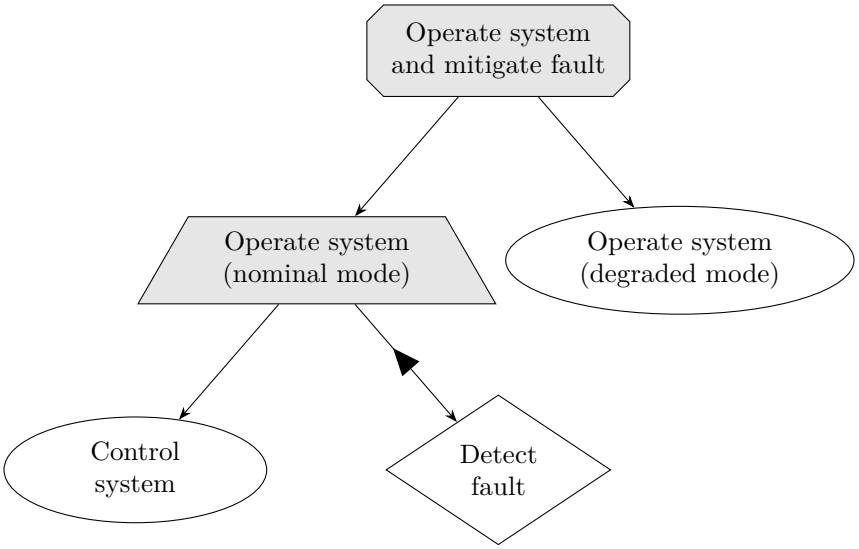


Figure 3.1: The BT for a system operation with fault detection and mitigation. Children are ordered from left to right. The BT features a new type of *Fault detection node* as presented in Definition 15

.

Definition 9 (Behaviour status) *The set of possible* statuses *for behaviours is the finite set* $\mathcal{S} = \{\textsc{Idle}, \textsc{Running}, \textsc{Success}, \textsc{Failed}\}$. *At each instant in the execution of a BT, the state of the BT is a function* $state : \mathcal{B} \to \mathcal{S}$, *which associates a status to each behaviour in the tree.*

A behaviour whose status is not Running can be started by its parent, and its status then becomes Running. A running behaviour can be interrupted by its parent; its status then becomes Idle. A running behaviour can autonomously change its status to Success or Failed.

Leaf behaviours are used to represent the actual activities implemented by the system, under the form of concurrent processes. Their execution is orchestrated by their parent behaviours, which are usually picked among a set of predefined composite behaviours.

We use four predefined types of composite behaviours: SEQUENCE, FALLBACK, PARALLEL and the INVERTER behaviour, the execution behaviour of which was described in section 2.6. For the needs of our study we have created a new type of parallel behaviour node. We hence named the behaviour node standardly called Parallel, PARALLELALL, so as to make a distinction between the two Parallel behaviours. The definitions of the non-standard behaviour nodes (contribution) are presented in section 3.4.1. As a reminder, the returned behaviour status of each standard BT node, based on the status behaviour of their children nodes is summarised in Table 2.2.

Definition 10 (Sequence behaviour) *When a* Sequence *behaviour starts, it starts its first child. When the currently running child succeeds, the Sequence behaviour starts its next child, or succeeds if it is the last child. If any child behaviour fails, the Sequence behaviour fails at the same instant. We use gray signal shapes*  *to represent Sequence behaviours.*

Definition 11 (Fallback behaviour) *When a* Fallback *behaviour starts, it starts its first child. When the currently running child fails, the Fallback behaviour starts its next child, or fails if it is the last child. If any child behaviour succeeds, the Fallback behaviour succeeds at the same instant. We use gray octagon shapes* *to represent Fallback behaviours.*

Sequence and Fallback behaviours call their children sequentially, with execution order from left to right. They both have at most one running child at each instant.

Definition 12 (ParallelAll behaviour) *When a* ParallelAll *behaviour starts, it starts all its children in parallel. It succeeds if and only if all its children have succeeded. If one child fails, the ParallelAll behaviour fails and interrupts the rest of the children. We use gray* $\wedge$*-shaped trapezia*  *to represent ParallelAll behaviours.*

Note: in common literature ParallelAll behaviours are referred to as Parallel behaviours.

Definition 13 (Inverter behaviour) *An* Inverter *behaviour has exactly one child. It starts its child when it starts, and is running when its child is running; succeeds when its child fails, and fails when its child succeeds. We represent Inverters by triangle arrow decorations*  .

3.3 Using BT standard nodes to express SE and SA artifacts

In order to evaluate the potential of BTs to express SE and SA artifacts, we designed three proof-of-concept experiments. The first one demonstrates a BT advantage: its elements can be in the form of blackboxes and be further developed when new information is available. This can be particularly useful throughout the system development, where the system's definition happens incrementally and in recurring design loops, until the final version is produced.

Based on the data provided by SE and SA models, and with operational objectives, we can

create ODMs at each models' iteration. An example illustrating the beginning of this process is presented in section 3.3.1. Then, in a more elaborate example, MBSE and MBSA artifacts are used to show how their translation into a BT is possible, in section 3.3.2. Section 3.3.3 demonstrates the ODM integration in a simple supervision tool and contemplates implications to research. Finally, section 3.3.4 summarises the chapter with a brief discussion.

3.3.1 Proof of Concept 1: translation of SE and SA information to BT semantics

In the chosen scenario, and as mentioned above, information is integrated inside ODMs gradually, when new system information is made available. This is depicted in Table 3.2, that illustrates a juxtaposition of SE and SA information and their implementation into a BT. The example uses an Earth Observation (EO) satellite mission definition for the demonstration.

When defining the mission objectives of an EO satellite, we start by its primary mission, which is to take photos of area(s) of interest on Earth and transmit them back to the Operations Centres (OCs). The information coming from preliminary SE activities, here shown in the first column, is "Satellite must perform Earth Observation". This information can be translated in a BT –as shown in the 3rd column, where a single behaviour node "Observe Earth" is included in the tree, under the Root node "Perform Mission Successfully"–here a Sequence node.

Throughout the system development process, FMECA teams produce new system information, regarding new failure modes and the associated functional effects. An example is shown in the second column: "Mission may fail". This information can be integrated in the BT with the addition of a new behaviour node ("Quit Mission"), on the right side of the "Observe Earth" node. The latter represents the system's nominal mode. This would mean that, if the "Observe Earth" node ticking returns "Failure", the "Quit Mission" node will be ticked. If the "Observe Earth" node returns "Success" or "Running", the "Quit Mission" node will not be ticked.

In the third iteration, we assume the information "A mission fail can sometimes be mitigated by putting the Satellite on Standby Mode" is available by the SE team. This information can be integrated in the BT by adding a new behaviour node between the "Observe Earth" and "Quit Mission" node, which represents the system's "Standby" mode.

Consequently, if the "Observe Earth" node ticking returns "Failure", the "Standby" node will be ticked. If the "Standby" node also returns "Failure", the "Quit Mission" node is ticked. If the "Observe Earth" node returns "Failure" and the "Standby" node returns "Success" or "Running" the "Quit Mission" node will not be ticked. The latter would signify –in case of system supervision, that the system is currently in Standby mode.

The SE activities might then conclude that the Earth observation activity has three phases: capture photos, save photos, and send photos to the OCs. The "Observe Earth" behaviour node can then be turned into a Sequence node with children: "Capture photos", "Save photos" and "Send photos".

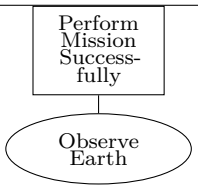
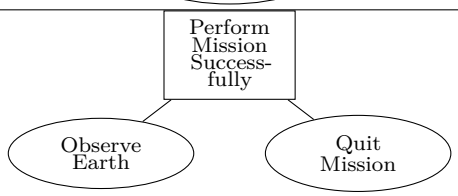
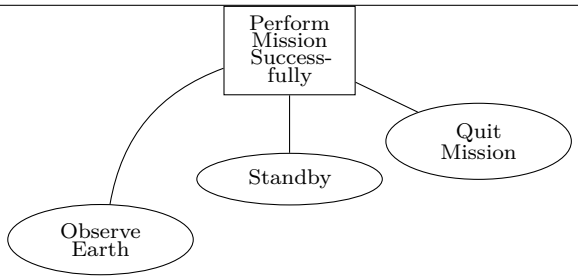
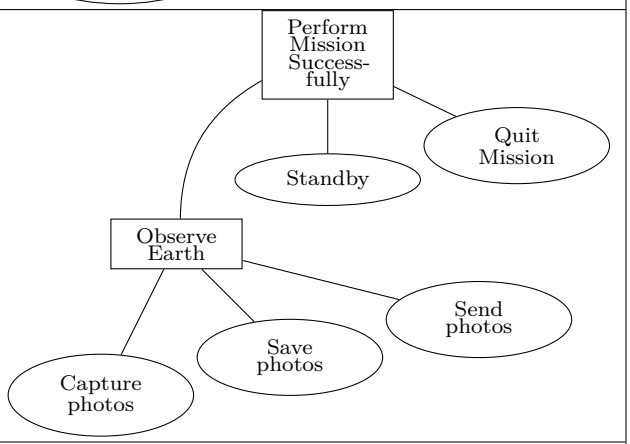
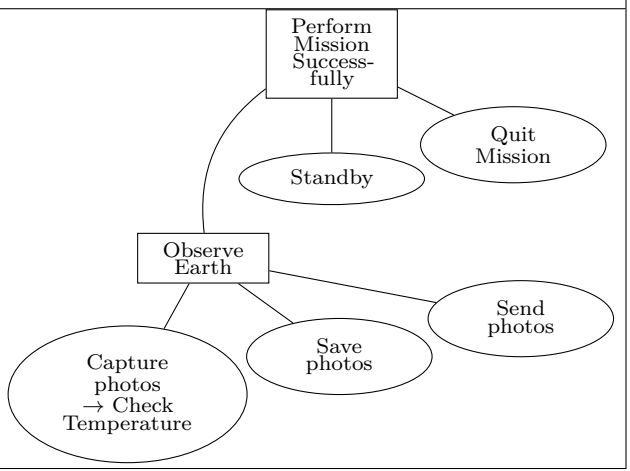
Functional textual information	Dysfunctional textual information	Behaviour Tree
Satellite must perform Earth Observation		
	Mission may fail	
A mission fail can sometimes be mitigated by putting the Satellite on Standby Mode		
Earth observation activity has 3 phases: capture photos, save photos, send photos to OC		
	A fault leading to instrument overheating causing the mission to fail can be detected by operator monitoring health status data during “Capture photos” phase	
Add instrument temperature monitor for operator		

Table 3.2: Illustration of the first iterations of a satellite design. At each iteration, the ODM can incorporate information coming from the system design process related activities, while remaining at each step a valid model.

This would mean that if the "Capture photos" activity is successful, then the "Save photos" activity can be performed. Similarly, if the "Save photos" activity is successful, the "Send photos" activity is able to be performed. If one of the activities cannot be performed, the "Observe Earth" activity will fail, and so the satellite will try to go into Standby mode.

Lastly, we assume that the FMECA teams communicate to the SE teams that the operator should be able to detect a fault leading to instrument overheating, thus causing the mission to fail, if able to monitor the instrument's health status during the "Capture photos" phase.

The SE team can then conclude that adding embedded temperature monitoring capabilities (i.e. temperature sensors) in the picture capturing instrument (onboard the satellite), can help prevent system failure, and implement it in the system design. This new modification shall also be added in the BT. This way the ODM is up-to-date with the SE and SA system models.

3.3.2 Proof of Concept 2: translation of SE and SA models into BTs

In this section a satellite subsystem use case is used to verify whether it is possible to represent MBSE and MBSA artifacts with BTs. A simulation of the architecture definition of a Guidance, Navigation and Control (GNC) satellite subsystem from the functional model design, to the dysfunctional model creation and analysis is illustrated. At the end of the two activities, an ODM is constructed, using the available SE and SA data.

In this proof of concept we use an old SysML model (made in Cameo) for the system description[1], available from a past project [170]. For the needs of our case study, we assume that this model is provided to the ODM modellers at some point throughout the system life cycle.

Figure 3.2 presents an overview of a GNC subsystem of a satellite's on-board FDIR module, consisted of State Machine Diagrams (SMDs). For simplification purposes, the internal modes (and their transitions) of each GNC mode were kept here in the form of blackboxes. The detailed diagram can be found in its entirety in Figure 5.1, Annex 5, while the model itself in [171].

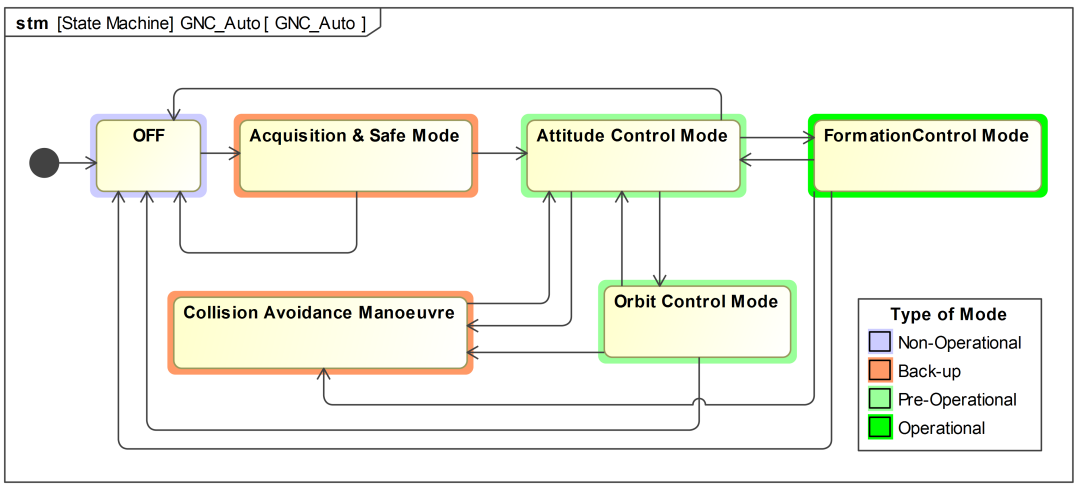


Figure 3.2: GNC Subsystem of a satellite's FDIR module modelled in SysML using Cameo Systems Modeler™.

[1]Created by M. Roquemaruel (Airbus Defence and Space), courtesy of MOISE project.

SMDs are used for modelling state transitions between different system modes. Here we can see the six GNC subsystem modes. The GNC subsystem can transition from Standby (OFF) mode to Acquisition & Safe Mode (ASM) and back. In Figure 3.3, where we illustrate the Acquisition & Safe Mode Internal SMD, we can see that the ASM mode has 3 internal modes: pre, normal and degraded mode. From its normal mode it can switch to Attitude Control Mode (ACM), which has the same internal states as ASM. ACM can switch from its Nominal mode to Orbit Control Mode (OCM) or Formation Control Model (FCM). OCM can switch back to ACM and equally has pre, nominal and degraded internal modes.

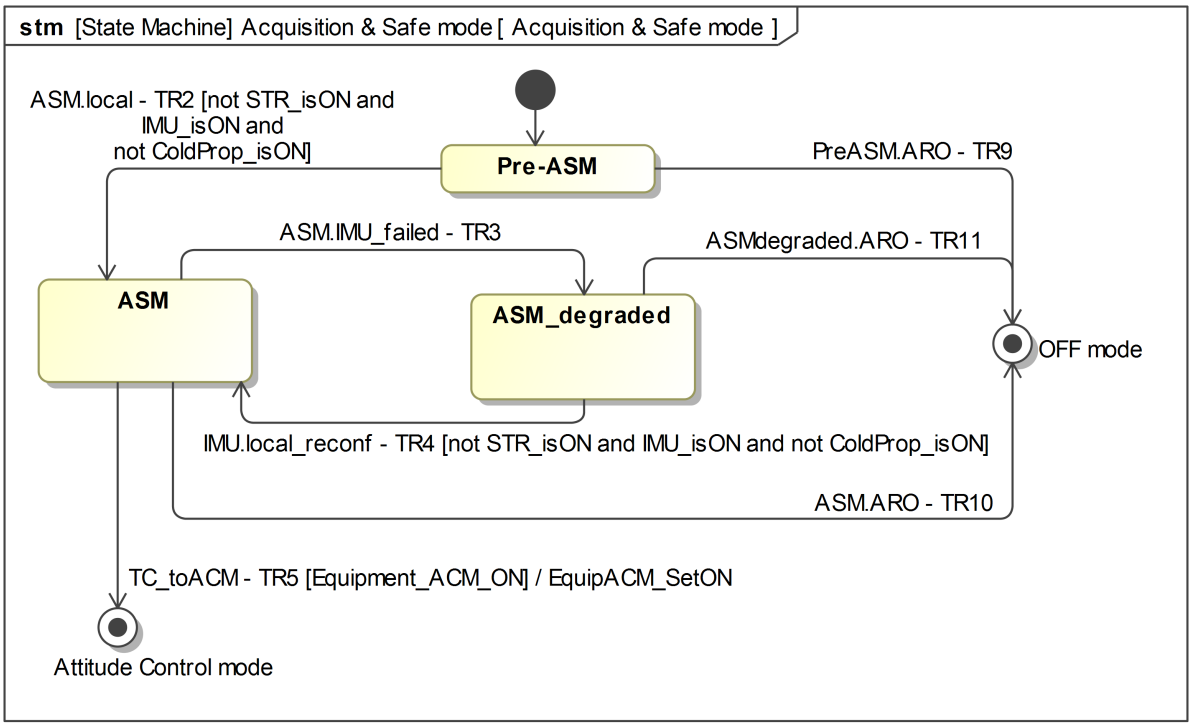


Figure 3.3: Acquisition & Safe Mode Internal State Machine Diagram.

FCM has the same internal modes and mode switches. In the case of an object being detected in the satellite' orbit, ACM, OCM and FCM switch to Collision Avoidance Manoeuvre (CAM) Mode. All modes can switch at any time from pre, normal or degraded, to standby mode.

Using SMDs for the construction of the BT model is not accidental. It is convenient to choose an SMD as a source for the BT model since SMDs are based on FSMs' logic, which is similar to BTs' logic. As a matter of fact, Colledanchise and Ögren even propose design patterns for translating FSMs to BTs [45].

Regarding the SA model, I built it using the SimfiaNeo tool and the AltaRica DataFlow language. This model must contain all the information regarding the satellite subsystem components' reliability. The purpose is to calculate the availability of its primary function i.e. "Observe Earth". As an Earth Observation (EO) satellite, it must be able to complete its mission, which is take photos of specific, predefined areas on Earth, and successfully transmit them (*non-degraded image; minimum noise and blur*) back to the Operations Centre (OC).

In the contrary case, and if the failure is permanent, we have *"loss of mission"*, which would be associated to a *Critical FC*, according to ECSS-Q-30-02A [172]. The mission was also divided into several phases, In regard to the system information included in Table 3.2, we have divided the model in several mission phases, as seen in Figure 3.4.

EO Satellite - Phases

	Variable	Standby	Battery Recharging	Image Capturing	Send Images / Exchange TM&TC	Save Photos / Clear Memory
1	Phase name	Standby	Battery Recharging	Image Capturing	Send Images / Exchange TM&TC	Save Photos / Clear Memory
2	Mission time	10	10	10	10	10
3	Inactive events	0	0	0	0	0
4	Satellite_System.Actuators.Thrusters_Cold_Gas_Propulsion.state_actuator	off	off	off	off	off
5	Satellite_System.Command_Decision.IMU_Command.command_state	idle	idle	idle	idle	idle
6	Satellite_System.Command_Decision.STR_Command.command_state	idle	idle	idle	idle	idle
7	Satellite_System.Command_Decision.Thrust_Command.command_state	idle	idle	idle	idle	idle
8	Satellite_System.OBC.state_obc	nominal	nominal	nominal	nominal	nominal
9	Satellite_System.OC_Antenna.oc_antenna	nominal	nominal	nominal	nominal	nominal
10	Satellite_System.Operator.operator_status	nominal	nominal	nominal	nominal	nominal
11	Satellite_System.Payload.Camera_Temperature_Monitoring.cam_temp_monitor	nominal	nominal	nominal	nominal	nominal
12	Satellite_System.Payload.Payload_Camera.state_payload_camera	nominal	nominal	nominal	nominal	nominal
13	Satellite_System.Power_Source.state_power_source	nominal	nominal	nominal	nominal	nominal
14	Satellite_System.Sensors.Inertial_Mesaurement_Unit_IMU.state_sensor	off	off	on	on	on
15	Satellite_System.Sensors.Star_Tracker_STR.Camera_FPGA.str_cam_fpga_state	nominal	nominal	nominal	nominal	nominal
16	Satellite_System.Sensors.Star_Tracker_STR.Camera_Lens.lense_state	nominal	nominal	nominal	nominal	nominal
17	Satellite_System.Sensors.Star_Tracker_STR.Lense_Sensor.lens_sensor_state	nominal	nominal	nominal	nominal	nominal
18	Satellite_System.Subsystems.Attitude_and_Orbit_Control_Subsystem_AOCS.Acquisition_and_S...	off	nominal	nominal	nominal	nominal
19	Satellite_System.Subsystems.Attitude_and_Orbit_Control_Subsystem_AOCS.Attitude_Control_...	off	nominal	nominal	nominal	nominal
20	Satellite_System.Subsystems.Attitude_and_Orbit_Control_Subsystem_AOCS.Collision_Avoidan...	off	nominal	nominal	nominal	nominal
21	Satellite_System.Subsystems.Attitude_and_Orbit_Control_Subsystem_AOCS.Formation_Contro...	off	nominal	nominal	nominal	nominal
22	Satellite_System.Subsystems.Attitude_and_Orbit_Control_Subsystem_AOCS.OFF_Mode.state_a...	nominal	off	off	off	off
23	Satellite_System.Subsystems.Attitude_and_Orbit_Control_Subsystem_AOCS.Orbit_Control_Mo...	off	nominal	nominal	nominal	nominal
24	Satellite_System.Subsystems.Communication_and_Data_Handling_Subsystem_CDHS.Antenna....	nominal	nominal	nominal	nominal	nominal
25	Satellite_System.Subsystems.Communication_and_Data_Handling_Subsystem_CDHS.Data_Tra...	nominal	nominal	nominal	nominal	nominal
26	Satellite_System.Subsystems.Communication_and_Data_Handling_Subsystem_CDHS.Demodu...	nominal	nominal	nominal	nominal	nominal
27	Satellite_System.Subsystems.Communication_and_Data_Handling_Subsystem_CDHS.Modulat...	nominal	nominal	nominal	nominal	nominal
28	Satellite_System.Subsystems.Communication_and_Data_Handling_Subsystem_CDHS.Receiver....	nominal	nominal	nominal	nominal	nominal
29	Satellite_System.Subsystems.Communication_and_Data_Handling_Subsystem_CDHS.Transmit...	nominal	nominal	nominal	nominal	nominal
30	Satellite_System.Subsystems.Electrical_Power_Subsystem_EPS.state1	nominal	nominal	nominal	nominal	nominal
31	Satellite_System.Subsystems.Environmental_Control_Subsystem.state1	nominal	nominal	nominal	nominal	nominal
32	Satellite_System.Subsystems.Microcosm_Autonomous_Navigation_Subsystem_MANS.state1	nominal	nominal	nominal	nominal	nominal
33	Satellite_System.Subsystems.Propulsion_Subsystem.state1	nominal	nominal	nominal	nominal	nominal

Figure 3.4: Mission phases of the EO satellite in SimfiaNeo.

Then, with the "Image Capturing" phase as the critical phase, we need to create an elaborate SA model of the satellite, so as to be able to consider all the possible ffailures' propagation that can take place throughout the system. Figure 3.5 shows the EO satellite SA model, developed in SimfiaNeo. For demonstration purposes, some elements' modelling is only high level (they were modelled as blackboxes), while others e.g. Telemetry (TM) subsystem, in more detail.

If we take the information from Table 3.2 "add camera temperature monitor for operator", we understand that the payload subsystem, which includes the image capturing instruments i.e. the camera, must include a monitoring system. As it can be seen in Figure 3.6, the output node of the monitoring block will give a signal to raise an alarm if the camera has failed, or if the monitoring system itself has failed. The totality of the model can be found in [173], and the respective AltaRica code in [174].

After completing the model we are able to make calculations related to the Failure Condition (FC) "Loss of Mission", where we added an *observer*. By performing a sequences computation, we can construct the FT depicted in Figure 3.7. In this special case, the MCs of the FT consist of single failures, which is normally a forbidden design specification: if a single component failure can lead to a catastrophic FC, our system is not reliable.

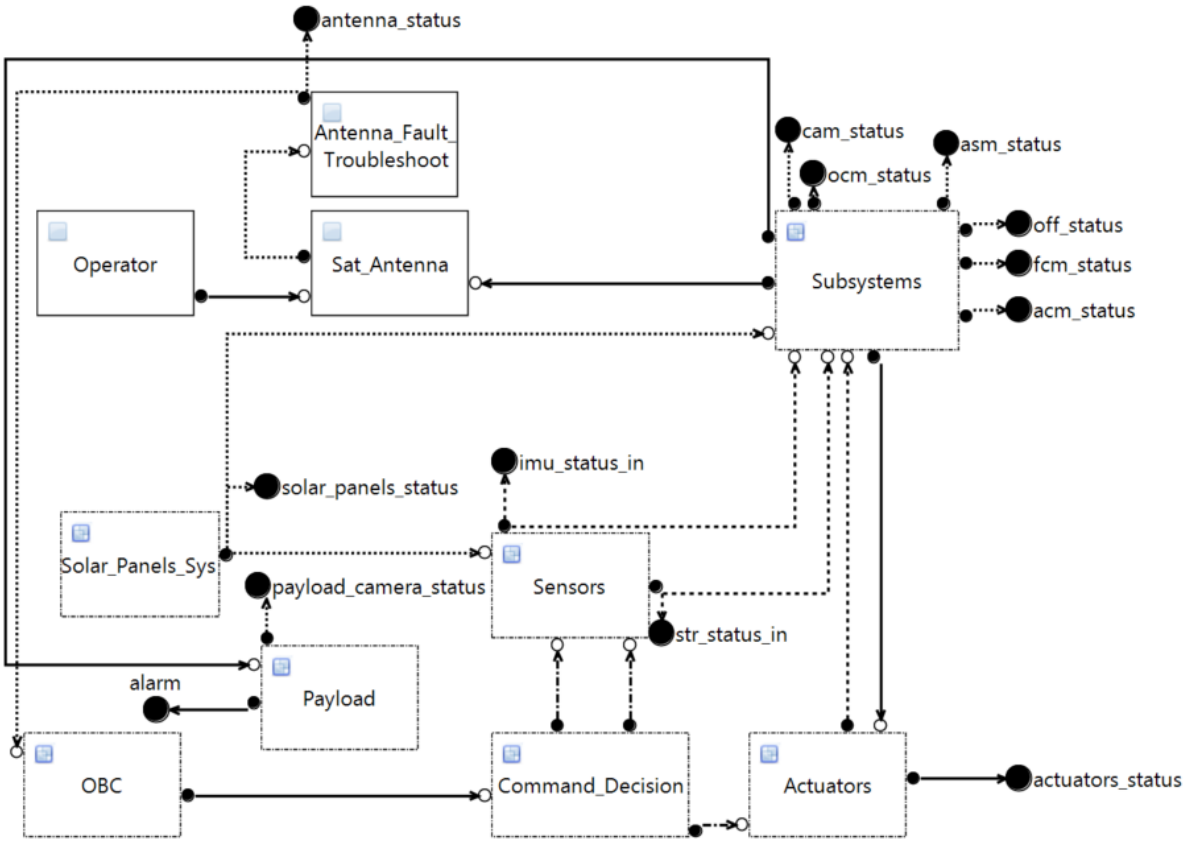


Figure 3.5: EO Satellite system in SimfiaNeo.

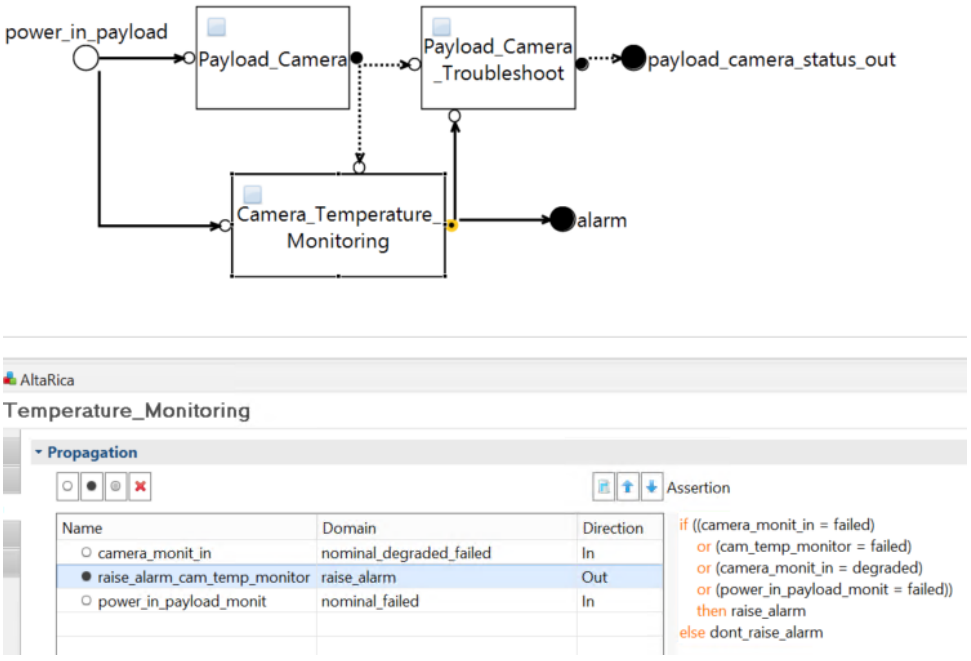


Figure 3.6: The EO camera, its monitoring system and the assertion for the output connector, in SimfiaNeo.

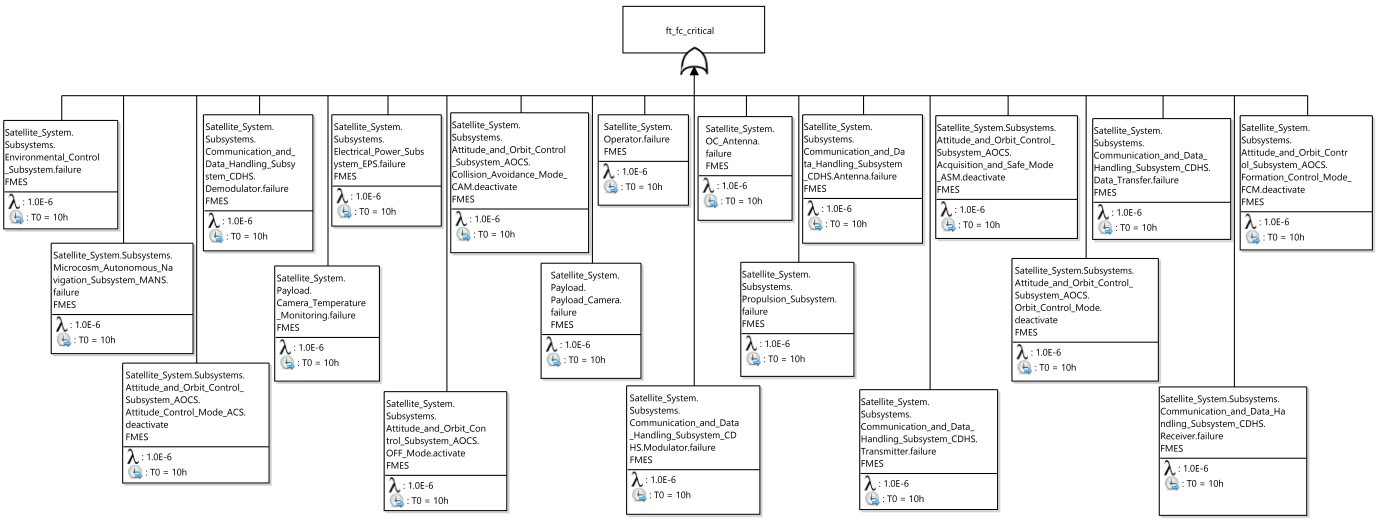


Figure 3.7: Generated FT for critical FC.

If not already defined by the system design team, components whose "single" failure (not in combination with other components' failure) can lead to a catastrophic FC, shall have at least a redundant element on-board. Both active and passive redundancy [175] can be modelled in SimfiaNeo, as shown in Figure 3.8. Here the passive redundancy of reaction wheels is presented.

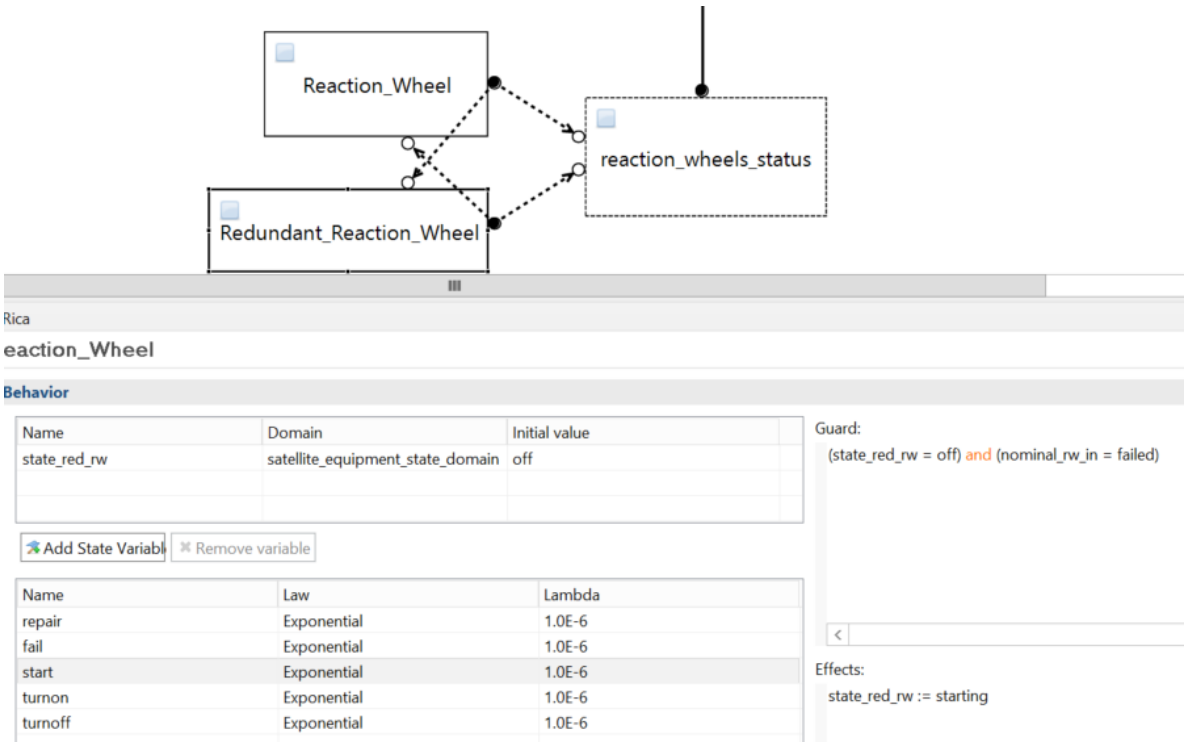


Figure 3.8: Redundant reaction wheels modelling in SimfiaNeo.

From the "guard" and "effect" dialog for the "start" event of the redundant reaction wheel, we can see that the latter can only change status from "off" to "starting" if the primary reaction wheel has failed. Further analysis can be performed, in the case of any components exhibiting

behaviours not conforming with the desired reliability and availability margins. The analysis can result to the proposal of modifications in terms of the system architecture, or the system supervision, in order to meet the anticipated performance criteria.

Moving on to the ODM, it must be mentioned that BTs are usually made for control purposes. In that case, the modelling objective is to have a realistic representation of the system behaviour, and in as-much detail as possible. Our objective however, is to construct a model which allows the system's health-status monitoring, to facilitate and generate its diagnosis, in case of failure.

A detailed representation of the system structure and functions is too large and too complex to read. Especially since the operators have very little time available, within the limited time-frame during which the satellite is passing over the OC i.e. communication window. This applies to all systems with high time constraints, and hence low availability.

For this reason, we need a model able to transmit health-status information in a clear way, while including both functional (SE) and dysfunctional (SA) elements–as described in section 3.1. We hence need a dysfunctional analysis of the whole, integrated system (and not only the GNC subsystem module), as well as the available system representation from the SE activities. The latter can be found in the SMDs of the SE SysML model that describes the internal mode switching of the GNC module (Figure 3.2). We are therefore able to get a model similar to the one depicted in Figure 3.9.

As shown in Figure 3.9, for the "Observe Earth" function to succeed, all the tasks "Check Temperature", "Capture Photos", "Save Photos" and "Send Photos" must succeed, otherwise the function fails. When the satellite fails to fulfill its primary mission, which is to observe the Earth and send the captured photos back to the OC, the satellite goes into "Guidance, Navigation and Control (GNC)" mode, so as to correct its attitude, in order to be able to go back to its Earth observation task. When in GNC, the module could be in "Standby", "Back-Up", "Pre-Operational" or "Operational" mode. Respectively, it can be either in "ASM", "CAM", "ACM", "OCM", or "FCM". If the "Attitude Control" mode fails too, then the "Ground Navigation Control" fails, thus the mission fails (node "Mission Fail").

The way this BT is modeled, the GNC node only gets ticked when the "Observe Earth" node has failed. In reality this is not the case, since some GNC functions can be running in parallel with the principal mission-related functions (capture, save & send photos). More specifically, the satellite goes in backup mode (ASM, CAM) in case of emergencies, as for example, when an object is entering the satellite trajectory, or if the satellite localisation lost.

This would cause all the other functions go on standby mode until the GNC backup modes have performed their task successfully (avoided object, localisation re-established etc.). On the other hand, operational modes ACM, OCM and FCM can be running simultaneously with the "Observe Earth" node, since some of their functions might be essential for the mission tasks, such as changing the satellite's attitude in order to get a photo from a better angle.

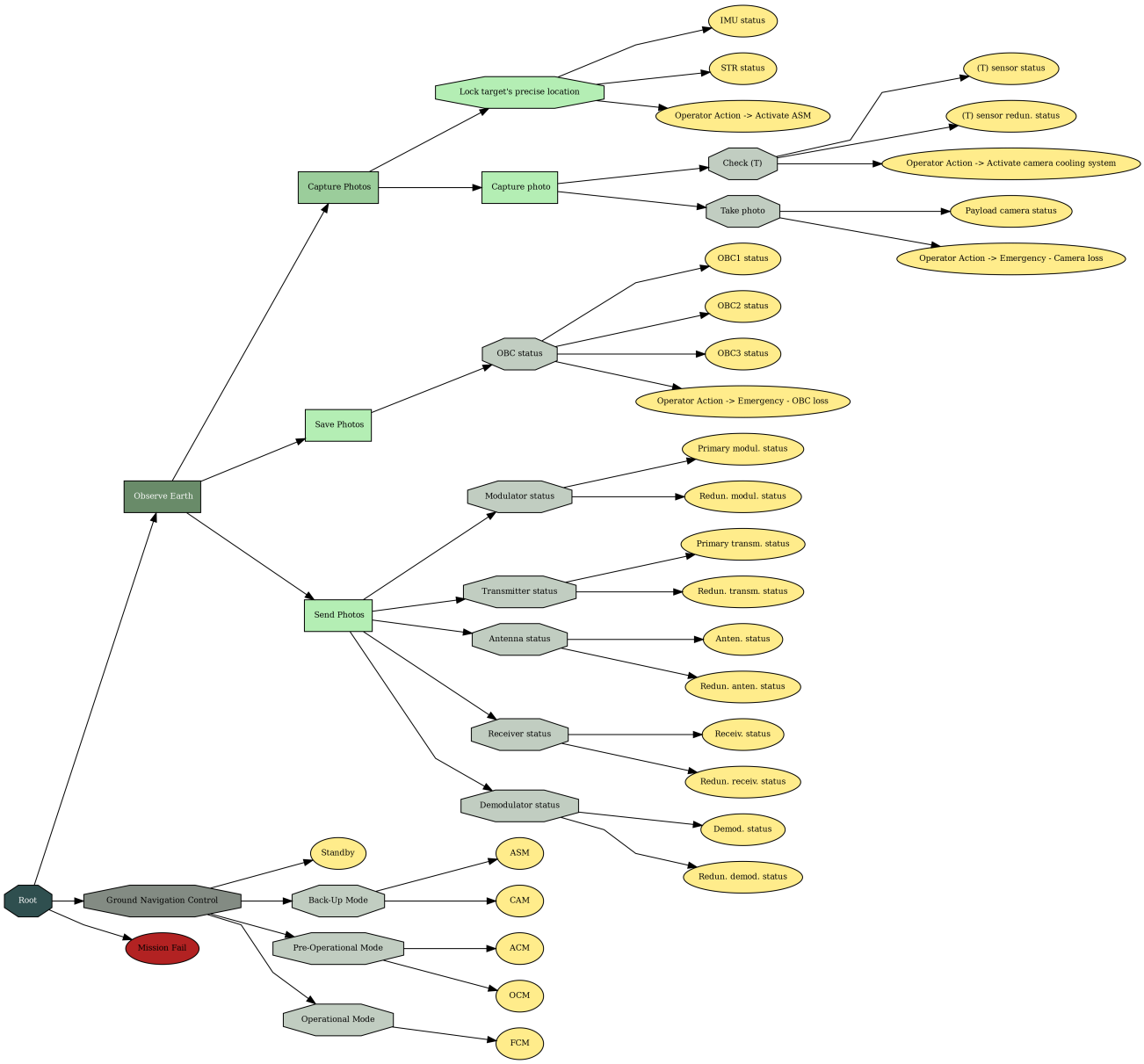


Figure 3.9: A BT model created with gradually collected information when made available by the SE and SA teams.

This shows the limitation of using only Fallback and Sequence nodes. In the beginning of our experimentation with BTs we only explored the use of Fallback and Sequence nodes, as they are the most commonly used in current literature. However, we quickly realised that the use of Parallel nodes is imperative to describe realistic complex systems' processes, as explained above. So we not only established the use of Parallel nodes in our methodology –see ParallelAll in section 3.2, but we also took BT semantics a step further. As explained in section 3.4.1, we went on to define an additional Parallel node (ParallelAny), in an effort to extend the descriptiveness of BTs, so as to be able to fully represent the structure and behaviour of complex systems.

This simplified BT model can be useful for the operator to observe the current state of the satellite system and be able to trace-back the appeared faults, alarms or errors, since they have an overview of the status of each system element that led to the situation at the moment of

the received TM. Moreover, the model does not include unnecessary system information which would make the model too large and difficult to read and exploit. Moreover, each mode (ASM, CAM, ACM, OCM, FCM) can be expanded, taking individual elements from other BT models. This way the operator can choose which elements will be visible from the model, depending the current needs. Moreover, "Operator Action" nodes are present in this model, in order to introduce some mitigation techniques (along with redundancy) in the ODM. The purpose of this node is to give control over the satellite to the operator, in case a series of certain failures occur, and to direct the operator to a specific troubleshooting procedure–according to the failure propagation that took place.

As mentioned above, for illustration purposes, only the TM function was modelled up to component level. As shown in Figure 3.9, when the primary component fails, the redundant component is activated. If we take the example of the signal modulator, the operator will be able to view its current health-status–whether the pinging of the task node "Primary modulator status" returns success or failure. If its the latter, the operator will be able to activate the redundant modulator. In the next BT execution, the operator will be able to see the status of the redundant modulator, which they tried to activate just before. In the case of success, the operator will run the troubleshooting procedure for restoring the primary modulator or understanding the fault propagation that caused its failure. In the case of failure, the operator knows that both the modulator components are out of reach and will thus launch the troubleshooting procedure for that situation.

3.3.3 Proof of Concept 3: Using BTs for system supervision

To take the example a step further, we use a tool we built, which takes as input BT models and upon execution (python script) prompts a UI that displays the current state of the BT nodes (the ones that were ticked). If we use as input the ODM introduced earlier (illustrated in Figure 3.9), after five time intervals, we see the prompt shown in Figure 3.3.

In this prompt we can see the behaviour nodes' name and their response in five consecutive tree executions–corresponding to five consecutive data packages sent from the satellite. The letter "S" stands for "Success" and "F" for "Failure". The exclamation mark ("!") stands for "Alarm", while "R" for running. As we can see, many different combinations (success/failure of functions and components) can result to the same performance, which allows the mission to be undertaken by the satellite. If our interest is the modulator, based on the current execution status, as well as the history available, we can speculate a fault in the On-Board Computer (OBC) propagating to other components. In the last execution, since both the primary and redundant modulators have failed, the satellite goes on standby mode and in the control of the operator. The latter will then run a troubleshooting procedure, drawing information from the current situation and provided history, as well as his experience and speculated assumptions.

Node Type	Behaviour	T=T0	T=T1	T=T2	T=T3	T=T4	T=T5	T=T6
Sequence	Root	S	S	R	S	R	S	R
Fallback	Observe Earth	S	S	R	S	R	S	F
Fallback	Capture Photos	S	S	R	S	S		S
Sequence	Lock target's precise location	S	S	S		S		S
Leaf	IMU status	S	S	S		S		S
Leaf	STR status							
Leaf	Operator Action -> Activate ASM							
Fallback	Capture Photo	S	S	R	S	S		S
Sequence	Check (T)	S	S	S		S		S
Leaf	(T) sensor status	S	S	S		S		S
Leaf	(T) sensor redun. status							
Leaf	Operator Action -> Activate camera cooling system							
Fallback	Take photo	S	S	R	S	S		S
Leaf	Payload Camera Status	S	S	R	S	S		S
Leaf	Operator Action -> Emergency Camera Loss Troubleshoot Procedure							
Fallback	Save Photos	S	S		S	R	S	S
Fallback	OBC Status	S	!S		S	R	S	S
Leaf	OBC1 Status	S	F		F	F	S	S
Leaf	OBC2 Status		S		F	F		
Leaf	OBC3 Status				S	R		
Leaf	Operator Action -> Emergency OBC Loss Troubleshoot Procedure							
Sequence	Send Photos	S	S		S		S	F
Fallback	Modulator Status	S	S		S		S	F
Leaf	Primary Modulator Status	S	S		S		S	F
Leaf	Redundant Modulator Status							F
Fallback	Transmitter Status	S	S		S		S	
Leaf	Primary Transmitter Status	S	S		S		S	
Leaf	Redundant Transmitter Status							
Fallback	Antenna Status	S	S		S		S	
Leaf	Primary Antenna Status	!S	S		S		S	
Leaf	Redundant Antenna Status							
Fallback	Receiver Status	S	S		S		S	
Leaf	Primary Receiver Status	S	S		S		S	
Leaf	Redundant Receiver Status							
Fallback	Demodulator Status	S	S		S		S	
Leaf	Primary Demodulator Status	S	S		F		S	
Leaf	Redundant Demodulator Status				S			
Sequence	Ground Navigation Control							R
Leaf	Quit Mission							

Table 3.3: BT tool (developed in python). Displays the returned status of the ODM's ticked nodes.

3.3.4 Outcomes

From the previous three examples, we can safely conclude that representing simple MBSE and MBSA artifacts using BT semantics to construct ODMs is not only possible but also presents great advantages. What is more, alarms –serving as an example of modelling system fault detection mechanisms, were also implemented. Moreover, we have demonstrated the feasibility of integrating ODMs in a system supervision tool, equipped with dedicated UIs, that can display the status of each BT node upon execution. This tool can serve as being:

- the final monitoring tool, or a prototype;
- used to inspect / validate the BT behaviour so as to integrate it with other monitoring tools or frameworks.

Note that so far, we have only investigated how to transport existing information from SE and SA models to the language semantics of BTs. However BTs should also be able to contain information specific to operations, provided not only by SE and SA experts, but directly by operators. The next section proposes a methodology that enables this.

3.4 Methodological proposal for building ODMs

The ODM methodological approach consists in defining a model that describes the system's operational procedures, with particular emphasis on fault diagnosis activities. In this section, a step-by-step description of the proposed methodology is provided. Moreover, a variant of the BT language is defined, created for the needs of the ODM method. The aim is to provide BT language semantics with the tools to express ODMs using FTs as input artifacts.

As illustrated in Figure 3.10, ODMs serve a double purpose. On the one hand, since they are concurrently created with the SE & SA/RAMS models, they provide feedback to the system designers, specific to health monitoring and FDIR aspects. This consequently contributes to improving the system's structural and behavioural design.

The left side of Figure 3.10 portrays these interactions, between system architects, safety experts, the ODM design team and the operators, on the right side. Note that, the left part of Figure 3.10 portrays a group of *recursive processes*, which is initiated by the preliminary system architecture proposal and is followed by a series of iterative activities, that eventually lead to the final system design definition.

On the other hand, following system deployment, the ODM can immediately be used for system supervision and diagnosis. BTs are executable, hence the ODM can facilitate the design of test scenarios, distribute diagnosis objectives across the various activities, and ensure that operators are given realistic tasks. This process is illustrated at the right side of Figure 3.10.

Since the whole system lifecycle could not be treated within the PhD timeline, the scope had to be reduced to one of the two phases –the creation or exploitation of the ODM. Hence the ODM

creation phase was chosen to be the focus for the PhD work. The ODM exploitation phase is thus considered future work.

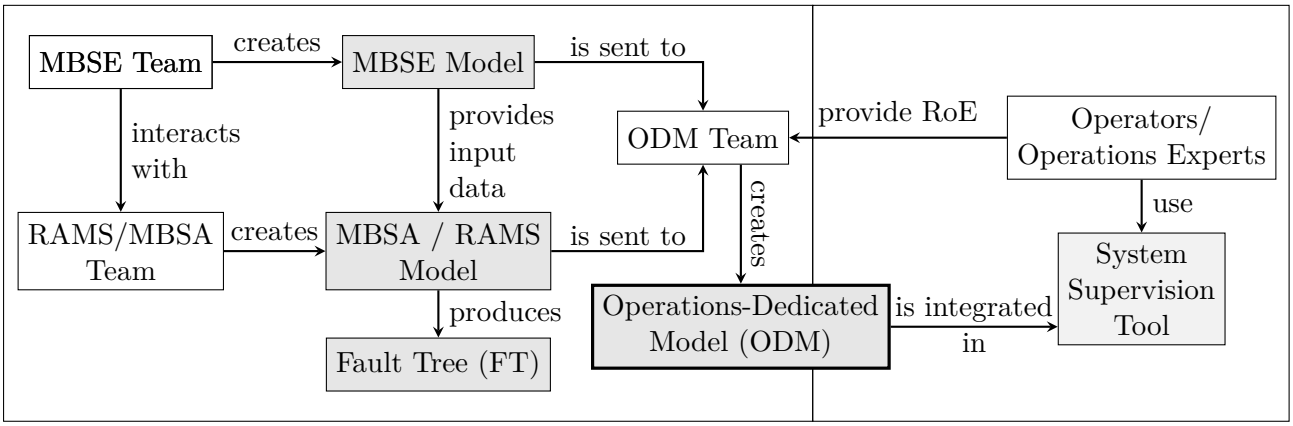


Figure 3.10: Overview of the proposed methodology for the construction and exploitation of an Operations-Dedicated Model (ODM)

Before we move onto the ODM methodology presentation, we will go through the new BT nodes we defined for this purpose. As mentioned above, we had to extend some BT semantics to meet the ODM requirements. Below, in section 3.4.1, are provided the definitions of these extended behaviour nodes, followed by a detailed description of the ODM methodology, in section 3.4.2.

3.4.1 BT Language Semantics - Extended Nodes

Here are provided three new BT node types, which we defined throughout the thesis work. They are an extension of the standard BT behaviours, as introduced in section 3.2. These new behaviour nodes were created so as to achieve the representation of redundant system behaviours –i.e. ParallelAny, as well as fault detection and avoidance system mechanisms in the BT language. They also facilitate the automatic model transformation from FTs to BTs. The returned behaviour status of each node, based on the status behaviour of their children nodes is summarised in Table 3.4. The latter includes both standard and extended BT nodes, as defined throughout our study.

Definition 14 (ParallelAny behaviour) *When a* ParallelAny *behaviour starts, it starts all its children in parallel. It fails once all its children have failed. If one child succeeds, ParallelAny behaviour succeeds and interrupts the rest of the children. We represent ParallelAny behaviours by* $\vee$*-shaped gray trapezia* $\bigvee$*.*

In addition to composite behaviours, we also introduce standard behaviours, related to fault diagnosis.

Definition 15 (Fault detection behaviour) *A* Detect *behaviour is an atomic behaviour dedicated to detecting a fault. In this behaviour, success means that it has detected the fault. Otherwise, it stays in the running mode and never fails. Fault detection behaviours are represented by diamonds* $\diamond$*.*

	Sequence	Fallback	ParallelAll	ParallelAny	Inverter	Fault Detection	Fault Avoidance
On tick	Ticks current child	Ticks current child	Ticks all children	Ticks all children	Ticks child	Ticks child	Ticks child
One child returns "Success"	Ticks next child	Returns "Success"	Waits all children	Interrupts other children, Returns "Success"	Returns "Failure"	Returns "Success"	Behaviour never succeeds
Last child returns "Success"	Returns "Success"	Returns "Success"	Returns "Success"*	Returns "Success"*	–	–	–
One child returns "Failure"	Returns "Failure"	Ticks next child	Interrupts other children, returns "Failure"	Waits all children	Returns "Success"	Behaviour never fails	Returns "Failure"
Last child returns "Failure"	Returns "Failure"	Returns "Failure"	Returns "Failure"*	Returns "Failure"*	–	–	–
One child returns "Running"	Returns "Running"	Returns "Running"	Returns "Running"	Returns "Running"	Returns "Running"	Returns "Running"	Returns "Running"

Table 3.4: Description Summary of BT Standard & Extended Nodes Execution: Sequence, Fallback, ParallelAll, ParallelAny, Inverter, Fault Detection & Fault Avoidance. *Note: here "last child" refers to the last non-terminated child; the child which has not yet returned its status to its parent —applicable only to nodes that tick several children in parallel.

The semantics of the BT depicted in Figure 3.1 are as follows. The top-most behaviour is a fallback, i.e. it tries to run its first child, and in case of failure, falls back to its next child. In this instance, the first executed behaviour is "Operate system (nominal mode)". The nominal mode is implemented by a ParallelAll behaviour, that runs the inverted "Detect fault" and "Control system" behaviours in parallel. When a fault is detected, the "Detect fault" succeeds, so its inverter fails, and thus the whole nominal mode behaviour fails. This interrupts the "Control system" behaviour. Similarly, if the "Control system" fails for some internal reason, the nominal mode fails and interrupts the "Detect fault" behaviour as a result. When the nominal mode fails, the root fallback behaviour starts the "Operate system (degraded mode)" behaviour. Finally, the ODM methodology uses a type of behaviours that can be either atomic or composite, but their semantic is defined with respect to a specific fault event.

Definition 16 (Fault event avoidance behaviour) *A* Fault event avoidance *behaviour is a behaviour that should fail when the fault event occurs, and never succeed. It can be atomic or composite. All behaviours whose names start with "Avoid" are Fault event avoidance behaviours.*

Fault event avoidance behaviours are always associated with a fault event from a FT. These fault events represent the failure of a function or of a fault detection mechanism. Therefore they do not always make sense from an operational point of view. Fault event avoidance behaviours are merely an artifact used as temporary translation between FTs and ODMs. Moreover, in a composite Fault event avoidance behaviour, its children must be compatible with the fault avoidance specification, otherwise the BT is invalid, and its semantics hence undefined.

3.4.2 ODM Methodology Description

Our methodology for obtaining an ODM from a FT is composed of *two steps*:

1. Translate each FT into a BT (*BT-v.1*). This step is automated; the purpose is to provide a first draft which accounts for all the faults that can affect the system.

2. Elicit all BTs into a single BT (*BT-v.2*), in which each behaviour represents an actual activity in operations. This step is done manually by a person with modelling skills, but more importantly operational experience.

For illustrative purposes, we assume that all fault events can be detected or mitigated, hence be included in the ODM as their corresponding operational activities. However, in general FTs may contain faults that cannot be detected nor mitigated (e.g. automatic monitoring not implemented), and therefore cannot be translated into any behaviour. In this sense, FTs can contain information irrelevant to operations. This is why we propose to construct ODMss from BTs in several steps, since: (i) an FT cannot consist an ODM as is, while (ii) additional new information must be elicited –in combination with FT data, to create a coherent and useful ODM.

During the *first step*, the FT is transformed into a BT as follows.

Definition 17 (Fault tree transform *FT2BT*) *The transform of a FT into a BT is implemented by the function FT2BT defined on FT nodes as follows.*

1. *If FTN is a basic event named "Fault event X", then:*
   FT2BT(FTN) is an atomic fault avoidance behaviour named "Avoid fault event X".

2. *If FTN is an AND gate labelled "Fault event X" with children nodes FTN_1, FTN_2, ..., then FT2BT(FTN) is a ParallelAny behaviour named "Avoid fault event X", with children behaviours $FT2BT(FTN_1)$, $FT2BT(FTN_2)$, ...*

3. *If FTN is an OR gate labelled "Fault event X" with children nodes FTN_1, FTN_2, ..., then FT2BT(FTN) is a ParallelAll behaviour named "Avoid fault event X", with children behaviours $FT2BT(FTN_1)$, $FT2BT(FTN_2)$, ...*

A pseudo-code for the *FT2BT* function is proposed below.

Definition 18 (Fault tree transform *FT2BT* pseudo-code)

```
def FT2BT(ft_node) {
    bt_node_name = "Avoid " + ft_node.name

    if (ft_node.is_basic_event())
        return AtomicBehaviour(bt_node_name)

    bt_children = list()
```

```
    for (ft_child in ft_node.get_children())
        bt_children.append(FT2BT(ft_child))

    if (ft_node.is_AND_node())
        return ParallelAny(bt_node_name, bt_children)
    if (ft_node.is_OR_node())
        return ParallelAll(bt_node_name, bt_children)

    raise Error("Unknown FT node type")
}
```

During the *second step*, firstly, *Fault avoidance* behaviours are elicited to *"Operate"*-type behaviours. That is because BT-v.2 represents an executable system model. Then, *critical* Fault avoidance behaviours are elicited to a parent Sequence *"Operate without critical faults"* behaviour, with children an Inverted *Fault detection* i.e. *"Detect critical faults"* behaviour, and an *"Operate with degraded faults"* behaviour.

The second step of the ODM methodology is performed manually, whilst following several general guidelines. In many instances, a behaviour named "Avoid something negative" does not represent a real activity in the system. The purpose of this step is to replace these unrealistic behaviours with other behaviours which account for:

- Fault tolerance and robust control.
- Fault mitigation activities.
- The fact that some activities occur in a predetermined sequence.
- The fact that some faults may have different observable effects depending on the system configuration, or its operational phase.

One important aspect of this step is that every transformation is documented and justified. This guarantees that every fault event considered in the FT is either directly accounted for in the BT, or handled by one or several precisely identified behaviours.

Regarding the expertise required for the building of the BT monitoring models, we propose that a dedicated team with operations background shall build the ODMs rather than the system architects or the safety analysts, so as to allow a distinct point of view. That would also ensure that the models will contain the necessary information for **supervision and diagnosis only**. Moreover, the construction of BTs with the PyTrees library is relatively easy and intuitive to code with. We hence believe that most system modelling engineers with no particular coding background, provided with sufficient training, guidance and documentation, could create ODMs. By not requiring particularly sophisticated skills prior to the initial training –other than system modelling and knowledge of operations, we can ensure that the monitoring model is easy to maintain throughout the years.

In the following chapter the methodology is thoroughly illustrated, through its application in several use case scenarios. More specifically, in sections 4.2 and 4.3.

3.5 Conclusion

This chapter addresses the question of which language semantics are the most appropriate for representing ODMs. The choice of BTs is presented and justified by using several examples to demonstrate that Behaviour Trees can be used to represent system design information, and to incrementally integrate new system data when they become available throughout architecture definition, in section 3.3.1. Moreover, BTs can be directly used to translate SysML model artifacts, in 3.3.2. Finally, that BTs can be integrated inside a system supervision tool, in section 3.3.3.

The standard and extended BT semantics are also provided, essential for building ODMs. It is demonstrated that BTs are able to represent SE/MBSE and SA/MBSA artifacts, and be developed concurrently with the rest of system design models, in a co-design manner. The ODM creation proposal from FTs is provided and will be illustrated through its application in various case studies, in the following chapter.

Proposal Validation with Case Studies

In the previous chapter the proposed methodology for the creation of an Operations-Dedicated Model (ODM) from Fault Tree (FT) data was described. In this chapter the methodology is illustrated through its application to three different use case studies. First, on a Satellite's AOCS subsystem, in section 4.1, followed by a UAV structure inspection mission, in section 4.2, and lastly, a Satellite Ground Station (GS) system, in section 4.3. Finally, section 4.4 discusses the collective outcomes of the performed case studies, while section 4.5 presents an overview of the totality of studies presented in this manuscript and their contribution to the thesis work.

The three case studies are presented in chronological order. First, we used the AOCS use case to evaluate the capability of BTs to integrate produced artifacts from SE and SA activities. Secondly, we used the UAV use case to attempt a methodology implementation on a realistic case study, from an FT obtained by a formal Fault Tree Analysis (FTA). Hence we could demonstrate our approach capabilities in the case when "ideal" (formal) input data are available. Lastly, we used the GS case study as an *a posteriori* application to our approach. The use case was related to an already existing system, with the absence of any formal input data (design data not tailored to our approach). This way we could evaluate the flexibility and versatility of the ODM creation approach.

4.1 AOCS Use Case

ODMs can be extended with the appropriate level of detail at each system development stage, by using the available information from the SE and SA teams. We consider the example illustrated in Table 3.2, which is constructed in the very beginning of the system development process, during its mission definition, to demonstrate the evolution capabilities of ODMs to integrate new information, and replace blackboxes with newly available system behavioural elements. For the needs of this study, we created the below diagram –Figure 4.1, and assume that it is provided to the ODM modellers at some point throughout the system design phase.

Figure 4.1 presents the operational phases of a spacecraft's generic Attitude and Orbit Control System (AOCS). It consists of a State Machine Diagram (SMD), and was modeled in SysML language. It was created using Airbus' MOFLT (see section 2.1.5) SysML profile. SMDs are used for illustrating state transitions within different system modes. They constitute an implementation of FSMs, as discussed in Section 3.1.

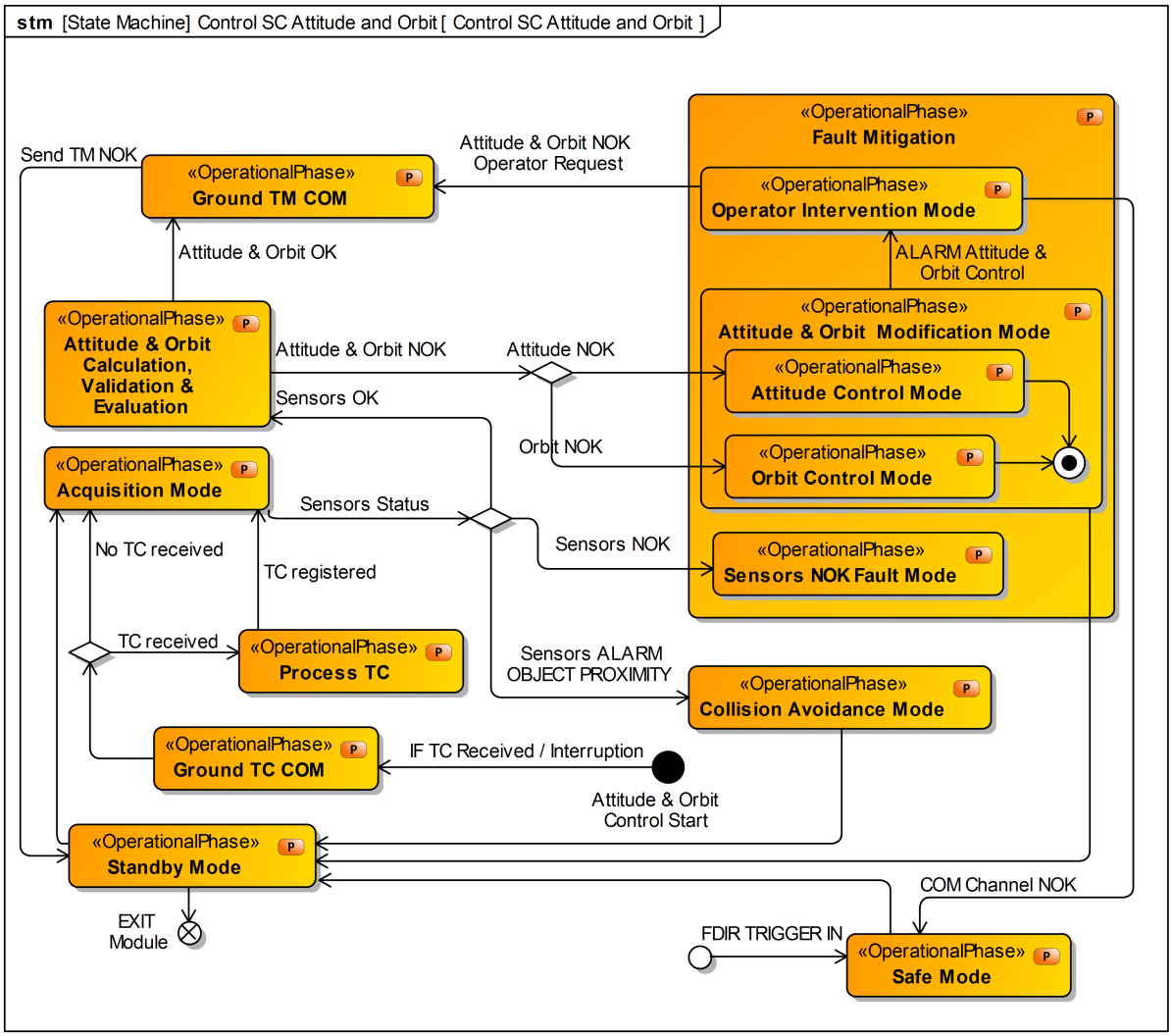


Figure 4.1: State Machine Diagram illustrating the internal modes' transition of an AOCS subsystem. Created using SysML in Cameo Systems Modeler.

In this example are modeled the operational phases –or modes, of the AOCS , both in functional and dysfunctional operational scenarios. As we can see, "Safe Mode", "Attitude & Orbit Modification Mode" and "Collision Avoidance Mode" can transition back to "Standby Mode". Unless a Telemetry (TM) is received, which interrupts the internal loop, "Standby Mode" can switch to "Acquisition Mode", which, according to a series of decisions, can switch to other modes.

The AOCS onboard module is responsible for:

- correctly calculating the SpaceCraft's (SC) Attitude & Orbit, by using sensor measurements e.g. star trackers or Internal Measurement Units (IMUs),
- keeping the SpaceCraft in correct Attitude & Orbit, and
- re-establishing the correct Attitude & Orbit in case of Attitude & Orbit drifting.

Hence the operational phases "Acquisition Mode" (receiving sensors' measurements) and "Atti-

tude & Orbit Calculation, Validation & Evaluation" (calculating the SpaceCraft's Attitude & Orbit values, and validating their correctness).

The AOCS module then (unless an error is detected) communicates these values to the operators i.e. to "Ground TM COM". If the calculated values fail to be validated, or if the sensors send a NOK status (NOK: Not OK), AOCS transitions to a "Failure Operational Scenario", where internal mitigation procedures start being implemented. Note that "Safe Mode" can also be triggered "externally", by an FDIR-module originated signal. The failure scenario starts putting in place mitigation mechanisms based on the type of detected fault, as for example the modelled fault modes "Sensors NOK Fault Mode" and "Attitude & Orbit NOK Mode". These modules will in turn trigger the activation of "Safe Mode" and "Operational Intervention Mode". Each of these operational phases are associated to other activities, implemented in a lower abstraction level (not visible in the diagram and not in scope for this example).

Using SMDs for the construction of the ODM is not accidental. It is convenient to choose an SMD as a source for the ODM since, as mentioned earlier, SMDs are based on FSMs' logic, which is similar to BTs' logic. As a matter of fact, Colledanchise and Ögren even propose design patterns for translating FSMs to BTs [45]. If we wanted to represent the SysML SE model as an ODM, we would have the model (or equivalent one) represented by Figure 4.2.

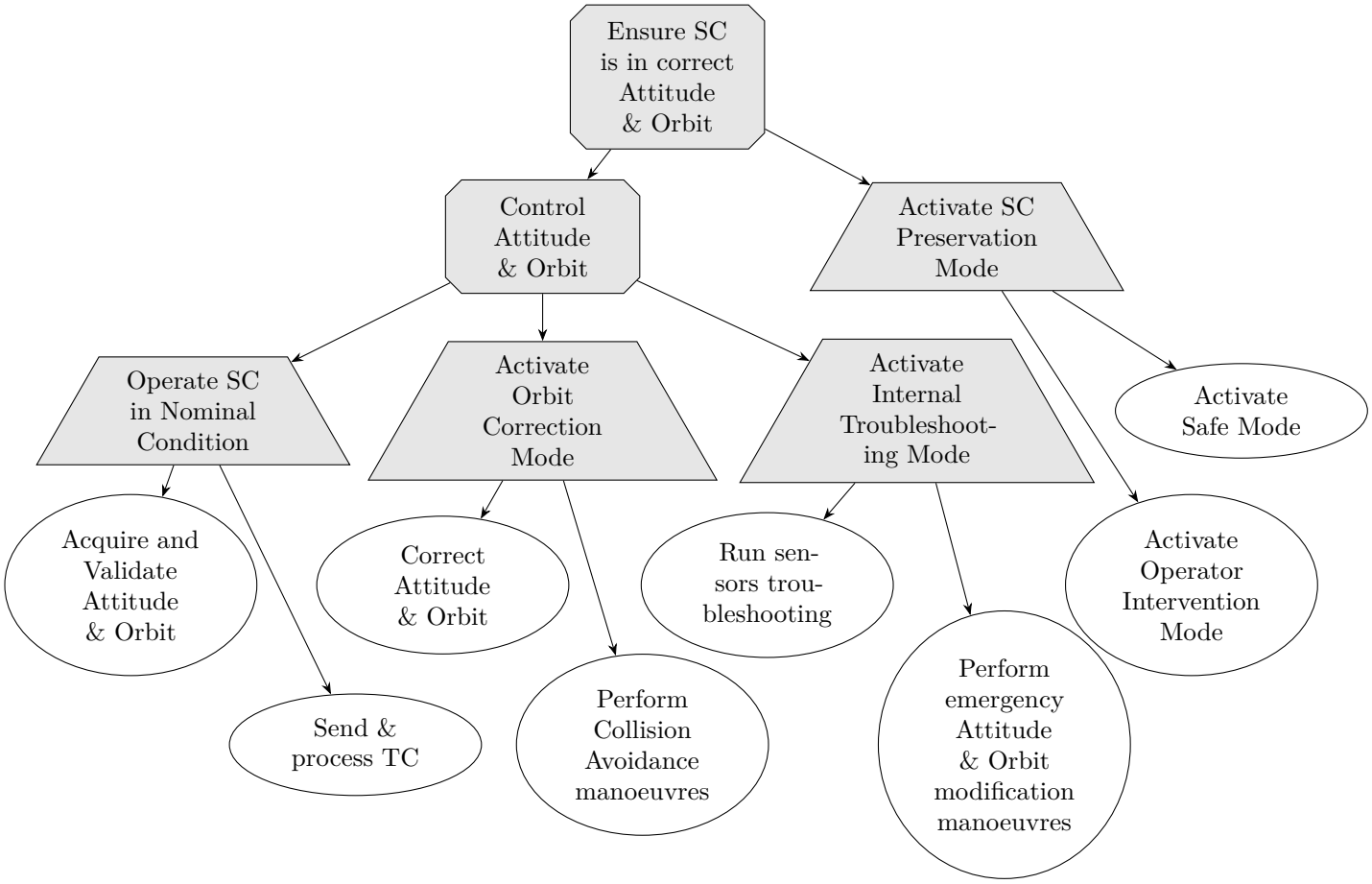


Figure 4.2: ODM of the GNC Satellite subsystem; created with information from the SE SysML model of Figure 4.1.

As shown in Figure 4.2, the Root node of the ODM is a Fallback node, with the associated behaviour "Ensure SC is in correct Attitude & Orbit". That is because the Root node represents the tree's main objective –the task the behaviour tree is expected to achieve. In this case, the tree is expected to ensure that the spacecraft is at all times in the correct Attitude & Orbit, as is the principal purpose of an AOCS module.

At each BT execution, the Parallel node "Control SC's Attitude & Orbit" is ticked first. If this node returns Failure, the ParallelAll "Activate SC Preservation Mode" node is ticked. This means that if the module fails to control the SpaceCraft's Attitude & Orbit, it will automatically call the operator to intervene, while attempting to go into Safe Mode. If, at any moment, any of the two (or both) "Activate Operator Intervention Mode" or "Activate Safe Mode" is successful or running, we managed to avoid the SC going off orbit, or failing. In reality, Safe Mode is activated following specific monitoring information e.g. power supply failure, in order to preserve the satellite, while waiting for human intervention with the goal to recover the nominal or redundant configuration.

Otherwise, if the "Activate Operator Intervention Mode" node is ticked, it means that the module fails to control the SpaceCraft's Attitude & Orbit, it will automatically call the operator to intervene. If this behaviour fails too, the module has no other choice than to "Activate Safe Mode". In this mode the SpaceCraft will go into idle. If this Action fails too, the Root behaviour will have failed. For a reminder of BT nodes' execution behaviour, see Table 2.2.

Coming back to the "Control Attitude & Orbit" node, which also has three ParallelAll behaviour children. The main AOCS functions when the SpaceCraft is in nominal condition are "Acquire and Validate Attitude & Orbit" and "Send & process TC". The respective functions for operating under major faults' occurrence are "Correct Attitude & Orbit" and "Perform Collision Avoidance manoeuvres". When the SpaceCraft is in critical condition, the AOCS attempts to "Run sensors troubleshooting" and "Perform emergency Attitude & Orbit modification manoeuvres".

By observing the diagram and tree illustrated in Figures 4.1 and 4.2 respectively, we can see that, although they were constructed using the same system information, they seem –and are, fundamentally different. That is because they serve an entirely different purpose. On the one hand, SysML SMDs are used during early system development, for high level design. The system's concept of operations, as well as its structural and behavioural architecture are determined. The relationships between the system's components and the various stakeholders are also delineated. At any moment throughout the system conception phase, design models must reflect the system's defined requirements, which are always at the epicentre. On the other hand, ODMs are built with an operational perspective, for system monitoring, diagnosis and troubleshooting. ODMs' end-users are the system operators, and not the modellers. Design models cannot represent this kind of operational diagnosis-related information, simply because they are built with disparate scope and objectives.

This simplified ODM can be useful for operators to observe the current state of the Satellite system and trace-back the appeared faults, alarms or errors. ODMs provide operators an

overview of the status of each system function, which led to a specific symptom in the received TM. Moreover, ODMs do not include unnecessary system information, causing it to be too large to read and exploit. Most importantly, *each tree node can be expanded, by incorporating the respective sub-trees in the current view.* This way the level of detail in the tree structure can be manually determined by operators themselves. Thus, based on the state in which the system is, the operator can be prompted to what they can or must do (the actions may be different depending on the system). The FDIR link can also be accounted for in this model.

4.2 UAV Mission Use Case

In the previous section we showed that BTs can indeed be built from system design data. We also argued that BTs are more suited to operations than their most related SE models. In the following example we ensured to have formal input artifacts, so as to illustrate the implementation of our methodology in system development processes where formal SE and SA activities take place.

Here we illustrate our approach with an FT produced for an Unmanned Aerial Vehicle (UAV) mission. It addresses the Feared Event (FE) of crash landing due to a fault occurred within the Power Supply (PS) subsystem. Regarding the UAV mission use case, we know the following:

- The UAV is used for infrastructure (power lines, canals etc.) inspection missions, and operates beyond visual line of sight (range over 100km);
- A complete FTA was performed, but we focus on a single FE, and a limited set of root causes; this selection was performed by an SA expert (Kevin Delmas, ONERA).

The UAV has several procedures for mitigating faults. First, it can perform a contingency landing, which is a controlled landing at a predefined location. Secondly, it can use the Flight Termination System (FTS), which deploys a parachute that slows down the UAV, so that it crashes with an as limited ground speed as possible.

Based on the preceding fault analysis, there exist various faults that can lead to a UAV crash landing. They are all modelled in this example as a degradation of the PS, with varying severity. Plenty of detection capabilities are associated to these faults, both on-board and at the control station, from where the pilot operates the UAV.

The UAV implements a fault escalation strategy as follows. A low-criticality fault occurring but staying undetected or unmitigated, transforms into a higher-criticality event. The *highest-criticality event is the FE itself*, i.e. the crash landing. The FT we take as input for our ODM creation methodology is depicted in Figure 4.3.

As we can see in Figure 4.3, the FT has six basic events. The rest of its nodes represent how the combination of the individual faults can escalate to the top FE.

The FT's basic six events are the following.

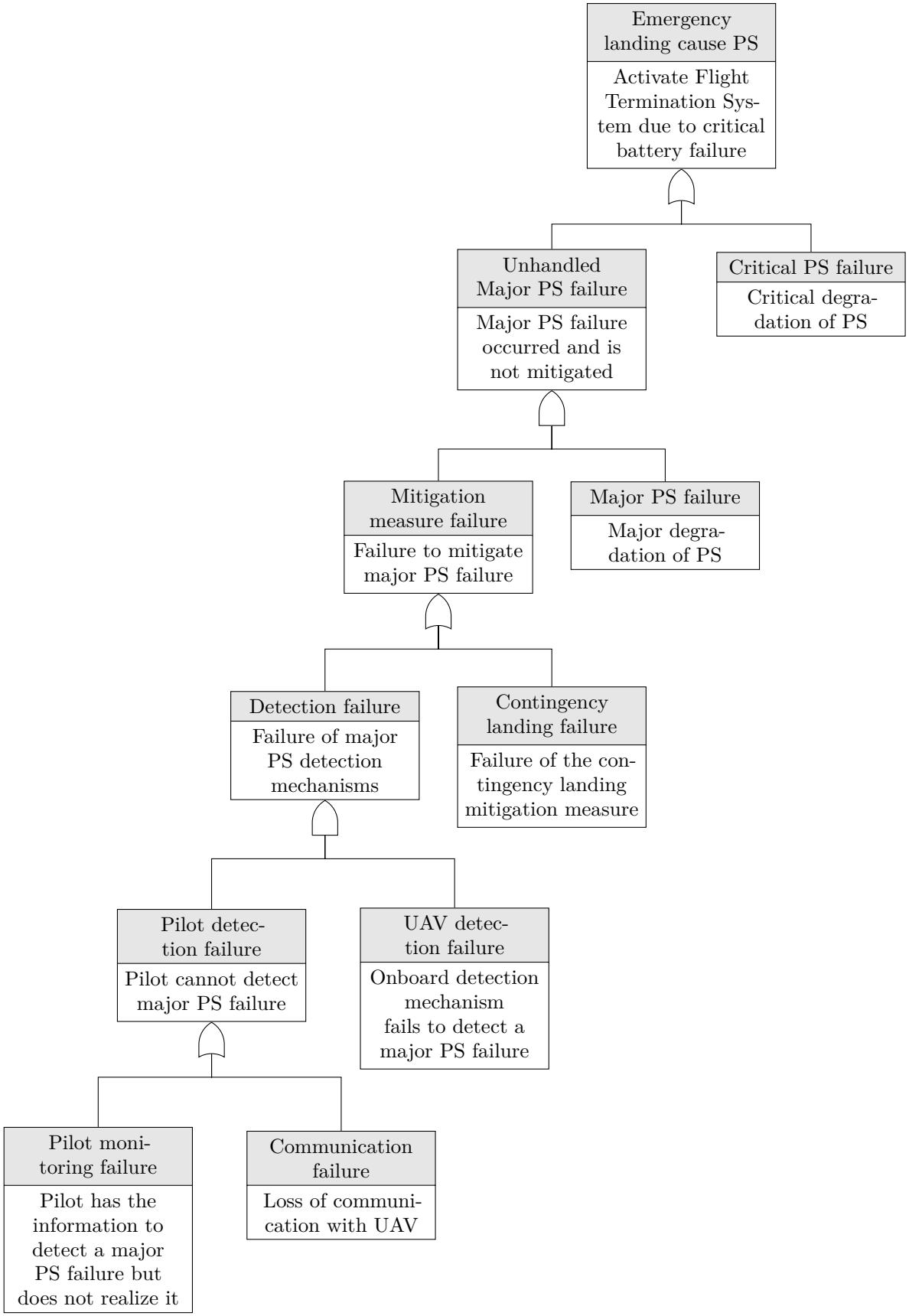


Figure 4.3: FT leading to the emergency landing of a UAV due to an unmitigated PS failure (feared event).

- *Critical PS failure*: a fault occurs in the PS in a way that completely shuts down its function.
- *Major PS failure*: a fault occurs in the PS in a way that significantly alters its function.
- *Contingency landing failure*: something goes wrong during the contingency landing procedure, e.g. an additional degradation of the PS, or an actuator failure, etc. This makes it impossible to carry out the procedure.
- *UAV detection failure*: the automatic fault detection mechanism on-board the UAV does not detect that the PS is faulty.
- *Communication failure*: the information that would let the pilot know that the PS is faulty is not communicated from the UAV to the pilot.
- *Pilot monitoring failure*: the pilot fails to realise something is wrong with the PS, even though they have all necessary information in order to detect the fault.

Based on the first methodological step (described in section 3.4.2), BT-v.1 (Figure 4.4 below) is obtained by applying the transformation *FT2BT* (3.4.2) to our FT (depicted in Figure 4.3).

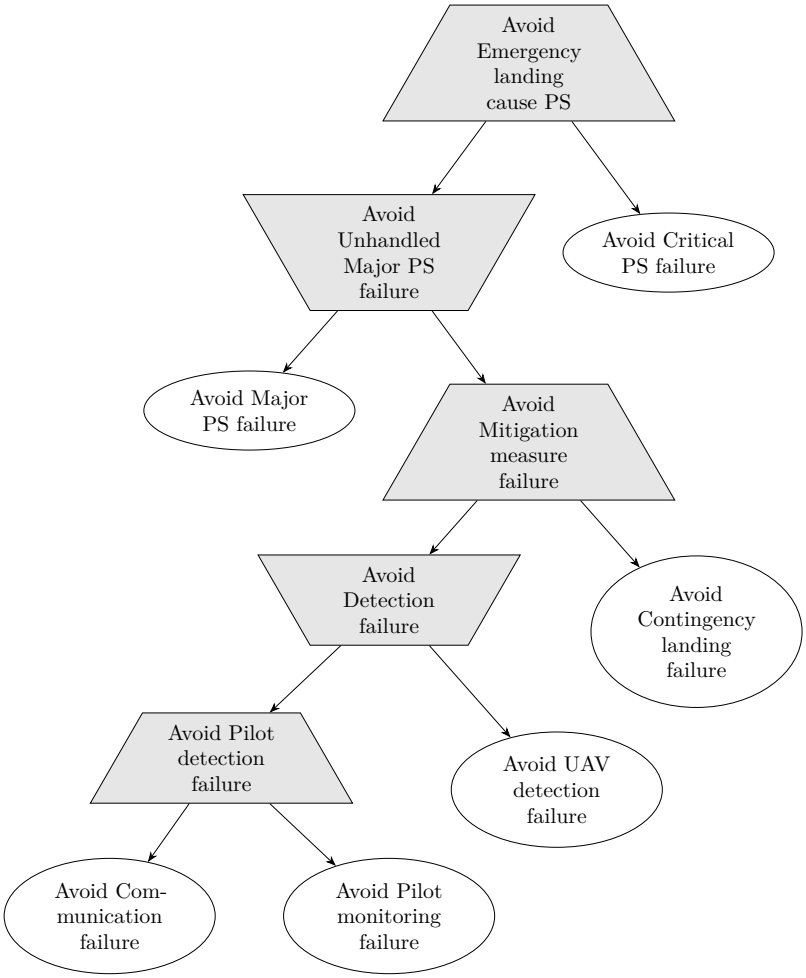


Figure 4.4: First version of the BT, derived automatically from the FT diagram (Figure 4.3).

One can observe that the result is a straightforward reformulation of the FT into a hierarchical fault management behaviour. Moreover, that:

1. BT-v.1 decomposes a global dysfunctional requirement into more specific dysfunctional requirements;

2. the transformation conceptually follows the "De Morgan's Law" principle [176], where: *not (A and B) = (not A) or (not B).*

Following the creation of BT-v.1 (not usable in this state), we describe the second model transformation step, where we use informal SE information to manually elicit the elements constituting BT-v.2. We hence depict in Figure 4.5 what BT-v.2 would look like if the operator applied the following elicitation steps to BT-v.1.

1. Behaviour "Avoid Emergency landing cause PS" involves a mitigation measure (the FTS). So we replace it with the fault mitigation pattern depicted in Figure 3.1. The topmost behaviour is called "Operate UAV with Critical PS faults"; the nominal mode is called "Operate UAV without Critical PS faults"; the degraded mode is "Activate FTS".

2. Behaviour "Avoid Critical PS failure" is replaced with the fault detection behaviour from the critical fault mitigation pattern, called "Detect critical PS faults".

3. Behaviour "Avoid Unhandled Major PS failure" takes place of the control behaviour is the critical fault mitigation pattern.

4. In addition, behaviour "Avoid Unhandled Major PS failure" also involves a mitigation measure (contingency landing). So we also replace it with the mitigation pattern of Figure 3.1. The topmost behaviour is called "Operate UAV with major PS faults", the nominal mode is named "Operate UAV without major PS faults", and the degraded mode is called "Perform Contingency landing". The command behaviour is named "Control UAV".

5. Behaviour "Avoid Mitigation measure failure" is part of the major fault mitigation pattern, and is deleted.

6. Behaviour "Avoid Detection failure" is the most delicate one: fault detection mechanisms may fail themselves, but the question is whether there exists a monitoring activity for these mechanisms. In this instance, we decide that in the event where we lose communication, or detect that the UAV cannot detect major PS faults anymore, or suspect the pilot in incapacitated, then we try a contingency landing. So "Avoid Detection failure" is transformed into a "Detect faults in detection system" that is a sibling (and thus has the same effect) as "Detect major PS failure".

7. Behaviour "Avoid UAV detection failure" is transformed into a fault detection behaviour, and renamed as "Detect faults in UAV detection".

8. Behaviour “Avoid Pilot detection failure” is transformed into a fault detection behaviour, and renamed “Detect faults in pilot detection”, and is left as a ParallelAll behaviour.

9. Communications failures cannot be prevented nor mitigated, we can only detect them. So behaviour “Avoid Communication failure” becomes a detection behaviour “Detect communication fault”.

10. Behaviour “Avoid Pilot monitoring failure” consists in validating the orders sent from the pilot. It is transformed into an atomic behaviour named “Validate pilot inputs”. Note that if the pilot sends absurd orders, the “Validate pilot inputs” fails. However, since its parent is a fault detection behaviour, fault occurrences should be a success. Thus, we invert the outcome of the “Validate pilot inputs” behaviour. An alternative would be to use a “Detect invalid pilot inputs” fault detection node.

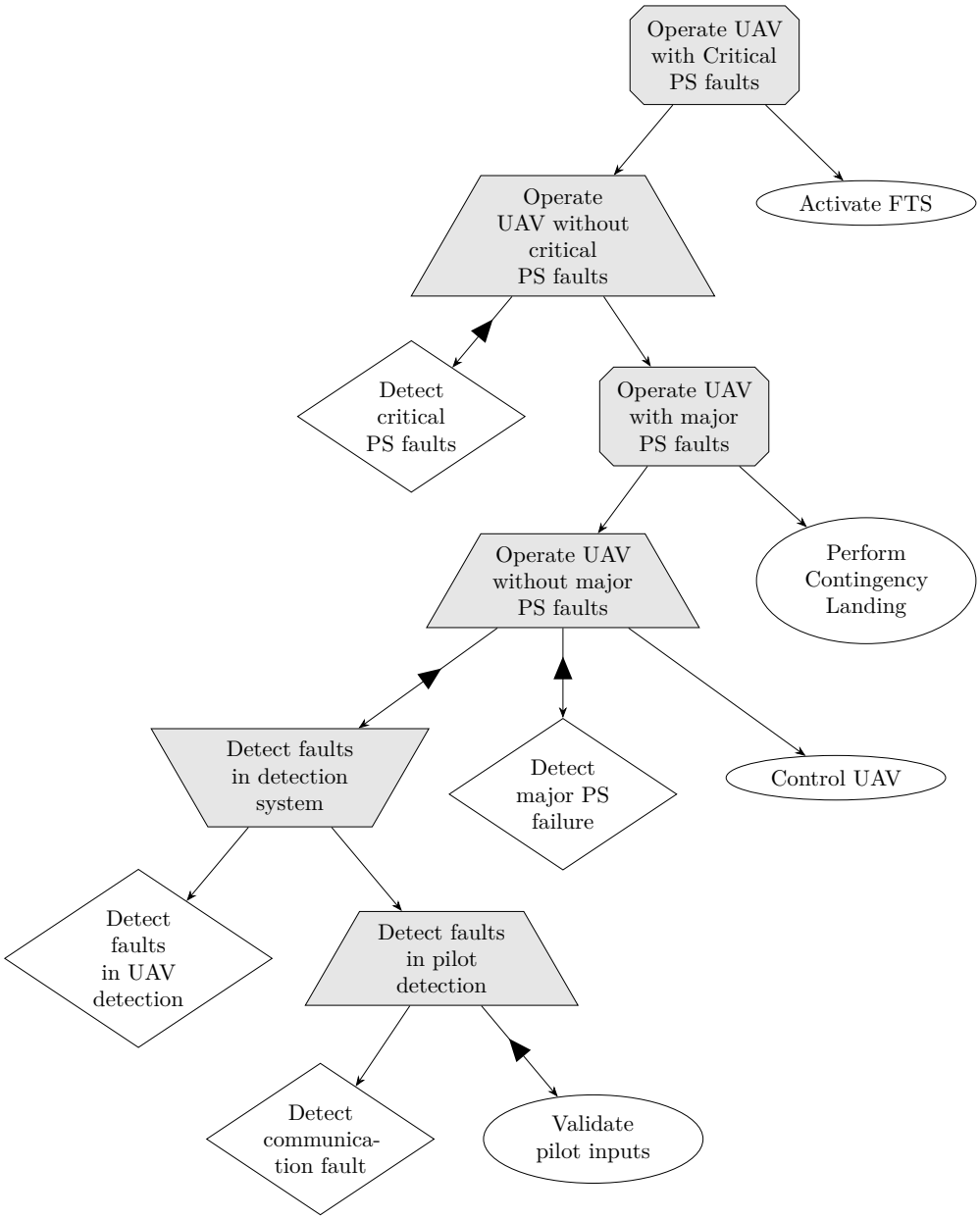


Figure 4.5: Second version of the BT, derived from eliciting the first version (Figure 4.4).

Note that at step 6, our methodology led us to discover that some cases were not covered by the system functional specification, not detailed here. Namely, in case of a failure in the fault detection mechanisms, it was not specified whether we can detect it, and what the corrective action should be. This demonstrates that the use of an ODM can help improve the design of the whole system. With this example we have achieved the following:

- we showed that our methodology works well with an FT as input;
- we decomposed a high level dysfunctional objective into smaller dysfunctional objectives, until they could be translated in functional requirements;
- we specified a set of diagnosis related activities, that can be fed to SE activities for implementation;
- there is still work to be done from the ODM to an actual diagnosis system.

4.3 Ground Station Use Case

In this subsection we illustrate our methodological approach proposal with the help of a space Ground Station (GS) use case (provided by Airbus Defence & Space (ADS)). This Satellite Telemetry Image (TMI) receiving station includes the *computing and control* part of the GS. Using the receiving plan information, the station's tasks include the following:

- acquisition/tracking phase initialisation;
- satellite tracking and data acquisition launching;
- metrics and log files generation, that reflect the TMI acquisition status;
- GS Antenna physical piloting, before and after the acquisition phase.

This receiving station contains its own scheduler so as to manage all the programmed passes. It also includes it own Antenna, which serves as the communication device and link between the Ground and the Satellite, to gather the TMI.

The information regarding the possible faults that can occur within the GS system was provided through a faults' propagation mind map, which can be found in its entirety in Annex 5. This mind map concerns the loss of the Telemetry (TM) data from the Satellite and is a collection of Return of EXperience (REX) from Satellite operators. As seen in Figure 5.3 of Annex 5, the loss of TM data can be due to four main causes:

1. Image files processing failure;
2. Data reception failure;
3. Site power distribution failure;
4. Safe Torque Off (STO).

Each main failure can be traced back to one or several failure modes, through fault propagation sequences. Note that, although the notion of hierarchy between the faults' occurrence does exist, the analysis does not take into account the order in which each fault occurs and how that could impact differently the system as a whole. This means that the hierarchical relation is not the same as the classical FT parent/child relation. In an FT, a child "contributes to" its parent, and occurs before. Here, it is more that a child "is a more specific diagnosis" than its parent. The reason is that this analysis is based on feedback from the operators, describing the faults often encountered while operating the Antennas, and not on *formal* safety analysis calculations.

Moreover, the analysis does not take into account combinatory faults' induced failures. This insinuates that the occurrence of a sole fault leads to another fault, but never a combination of two or more faults at the same time (simultaneously or with time difference). This will reflect in the respective FT and BT.

In order to better understand the GS's architecture, another schema was provided, illustrating the structural elements of the RF subsystem of the GS. This information is represented by Figure 4.6.

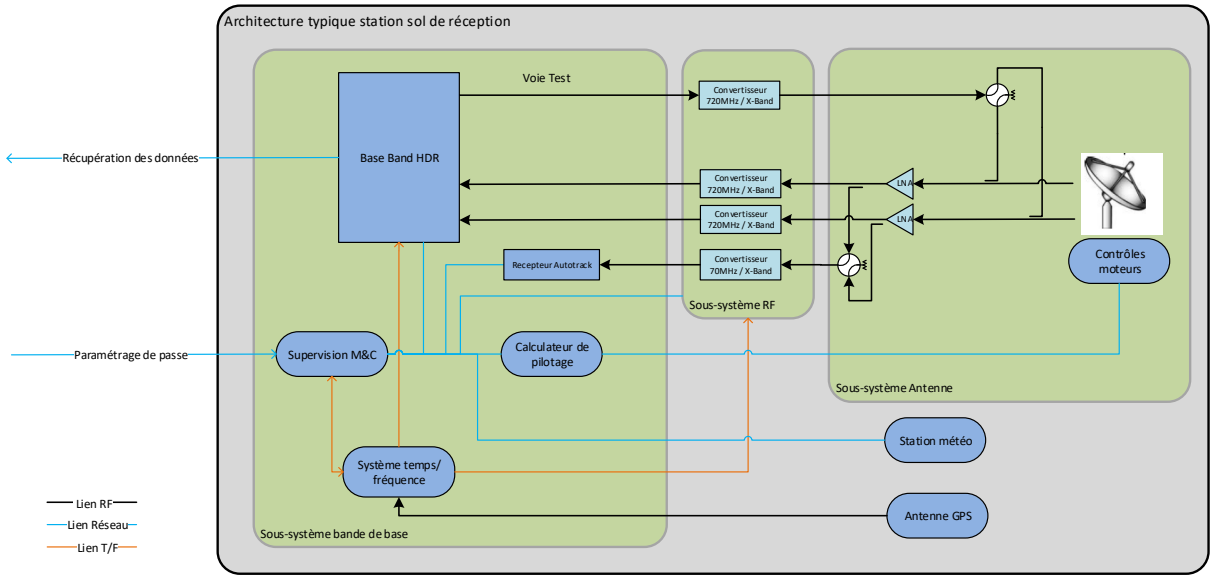


Figure 4.6: Synoptic Diagram of RF subsystem architecture. A larger view of the diagram is provided in Figure 5.2, Annex 5. Courtesy of Airbus Defence and Space.

There we can see the consisting elements of the data receiving part of the GS, which includes three main subsystems, namely are the following:

- Base-band subsystem,
- Radio-Frequency (RF) subsystem,
- Antenna subsystem.

Starting from the base-band subsystem, it consists of the M&C supervision, HDR base-band, and Time/Frequency subsystems, as well as the control computer and auto-track receiver.

Regarding the RF subsystem, it contains four frequency converters, between 720MHz and X-Band frequencies. The Antenna subsystem includes the Antenna motor controllers, two Low Noise Amplifiers (LNAs) and two transistors.

In the legend on the lower left side we can see the line colors meaning: black for RF links, cyan blue for network links and orange for T/F links. We can also see their interactions with the Weather Station and with the GPS of the GS's Antenna.

For illustration purposes we choose to tackle a small and representative part of the GS fault propagation mapping of Annex 5. For the methodology application demonstration, we selected the Radio-Frequency (RF) subsystem of the GS. Following discussions with operations experts from ADS, we concluded that the RF subsystem is one of the most interesting subsystems to tackle. We were then able to produce the mindmap shown in Figure 4.7.

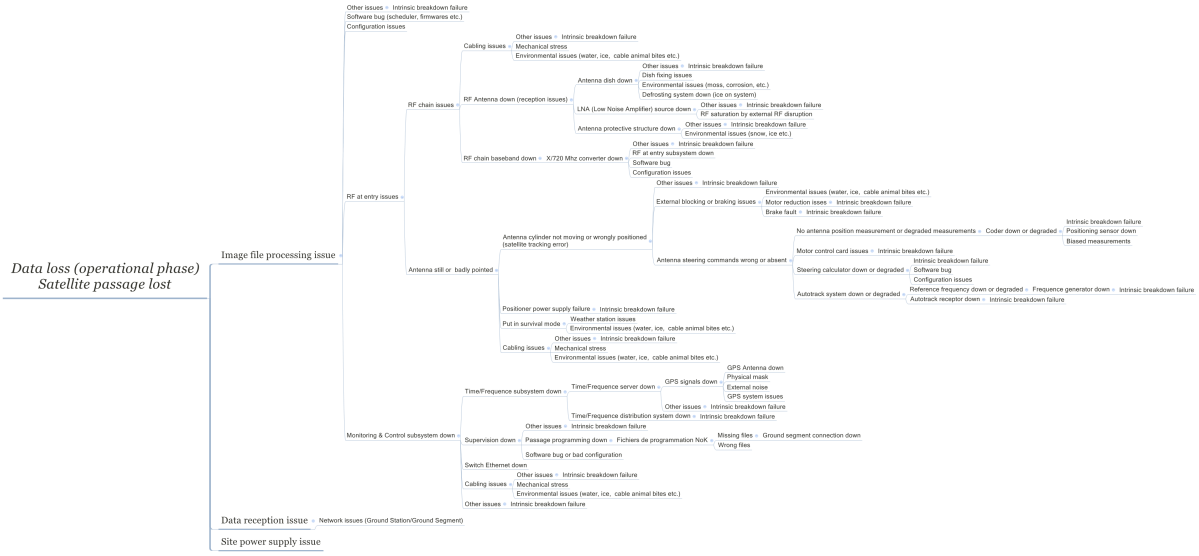


Figure 4.7: Extracted Mind Map with experimental information regarding data loss of Satellite TM data in the RF subsystem of a GS. Translated from French. A larger view of the diagram is provided in Figure 5.9, Annex 5.

We used the information regarding the GS' functional and dysfunctional architecture provided by Figures 4.7 and 4.6 to build a SysML Activity Diagram. This diagram illustrates the GS' sequence of functions, from the moment the station receives the pass planning forecast for the day, until it successfully transmits the TMI it received from the passing Satellite. This process is divided into several operational phases, hence defining its overall Concept of Operations (ConOPS).

This activity diagram depicts the functional scenario we chose for our case study, and is shown in Figure 4.8. Figures 5.5 to 5.8 illustrate how we interpreted the given data from the use case we were provided with, to develop a GS system description in SysML. The activity diagram of Figure 4.8 is a resulting part of this SysML model, which can be found in [177] for download.

The FT contemplates the Feared Event (FE) of loss of Satellite TM data due to a malfunction of the RF subsystem. Here we consider the RF subsystem as isolated from the rest of the GS system. The faults consisting the FT of Figure 4.9 represent only critical single faults. That is

because, as mentioned above, the information source for the analysis was solely the anomalies' DataBase, as collected from GS operators throughout the years. The second reason is that, other than a few means of self-testing, there is little point in investigating during the Satellite pass. That is because, even if an RF problem is identified, the pass is already lost (without capability of repairing). A fault mitigation solution to consider is a network of Ground Stations, as a System of Systems (SoS).

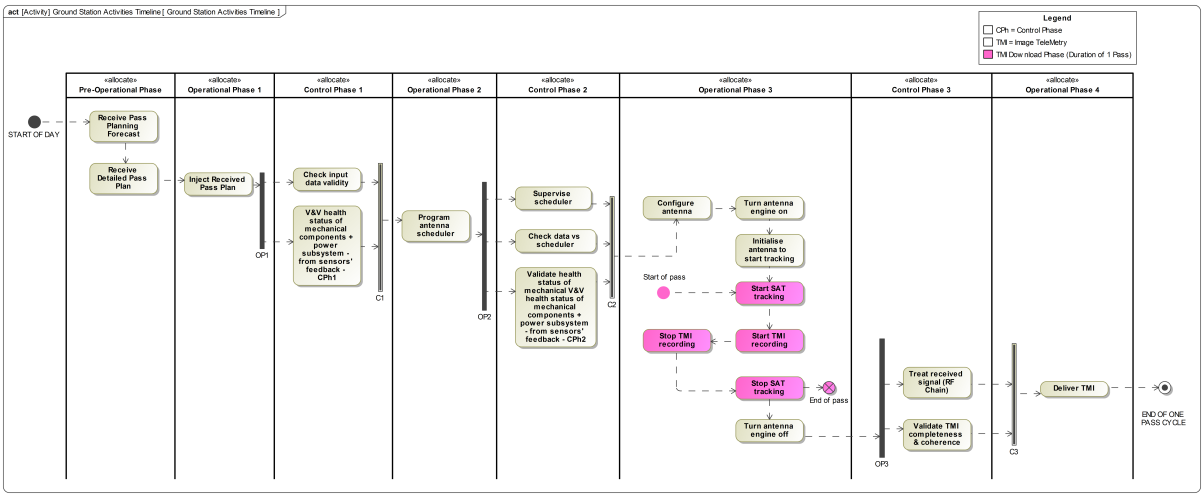


Figure 4.8: SysML Activity Diagram representing the operational scenario (sequence of operational activities) of an image telemetry data receiving Ground Station in *one* satellite pass (single satellite pass cycle). This is the operational scenario we tackle in our case study. A larger view of the diagram is provided by Figure 5.10, Annex 5.

Therefore, only OR gates are included in the FT, as we can see in Figure 4.9, where the represented faults are single (single event occurrence leads to the higher level fault expression), and not combinatory (a combination of faults leads to the higher level fault). The latter would need an AND gate for their expression.

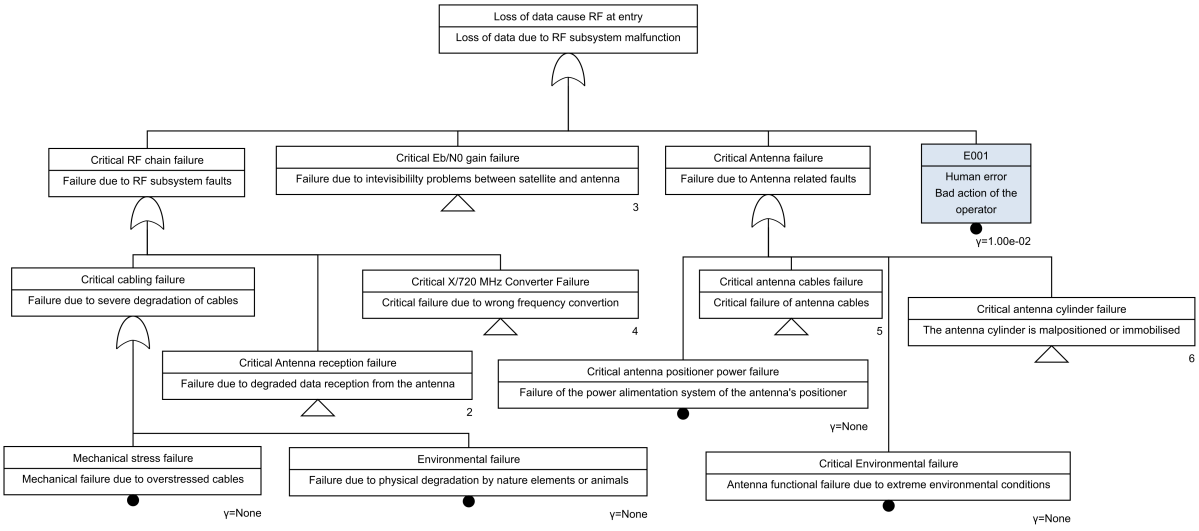


Figure 4.9: Fault Tree (FT) for a Ground Station (GS) system related to Satellite Telemetry (TM) data loss. Focus on the RF subsystem. Feared Event: Loss of TM data due to wrong RF at the GS TM receptor. A larger view of the diagram is provided in Figure 5.11, Annex 5.

In Figure 4.9 we can see that the loss of data due to the RF subsystem malfunction could be

caused by a critical *(i)* RF chain, *(ii)* Eb/N0 gain, or *(iii)* Antenna failure. Human error is out of context for this study. Failure *(i)* could be caused by a critical *(i.a)* cabling, *(i.b)* Antenna reception, or *(i.c)* frequency converter failure. Critical failure of the RF subsystem cables could be caused by a mechanical failure due to overloaded stress on the cables, or by physical degradation of the cables due to unforeseen environmental conditions and animals.

Failure *(iii)* could be caused by a critical failure of the Antenna *(iii.a)* positioner (rotator of the Antenna parabola), (iii.b) cables, *(iii.c)* cylinder (Antenna base body), or due to *(iii.d)* natural degradation of the Antenna (environmental phenomena and wildlife).

The first step in our methodology is to transform the FT into a BT, based on *4.9 - I, II & II.* The result is shown in Figure 4.10.

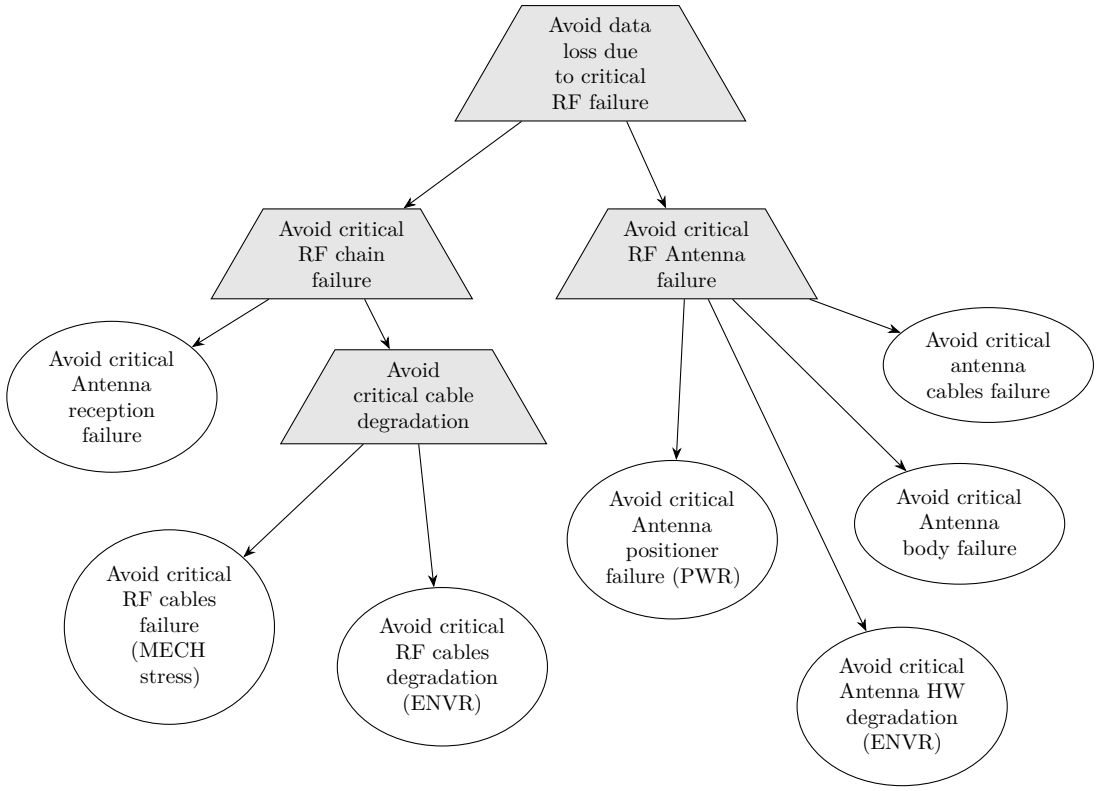


Figure 4.10: First version of the BT, derived automatically from the FT depicted in Figure 4.9.

Based on rule *no.I*, the FE "Loss of data cause RF at entry" becomes an *Avoid* type of behaviour. This results to the Root ParallelAll behaviour "Avoid data loss due to critical RF failure". Similarly, the rest FT nodes –still based on rule *no.I*, are translated into respective *"Avoid"* behaviours. Since all the gates present in the FT of Figure 4.9 are *OR* gates, following rule *no.III*, the respective parent BT nodes are represented by ParallelAll behaviours.

According to the second step of the method, BT-v.1 is elicited to BT-v.2, as illustrated in Figure 4.11. Thus all "Avoid", behaviours become "Operate" behaviours. The Root behaviour is represented by the Fallback node "Operate RF Sub-System (SS)". Its children consist of the ParallelAll behaviour "Operate RF SS without critical faults" and the leaf behaviours "Operate RF SS in degraded mode" and "Terminate RF SS's Functions". If the subsystem fails to operate

nominally, it tries to enter a degraded mode. There the subsystem is still in operation but with degraded functions. If operating the subsystem in degraded mode fails too, the subsystem executes a fail-safe action and terminates all subsystem's functions.

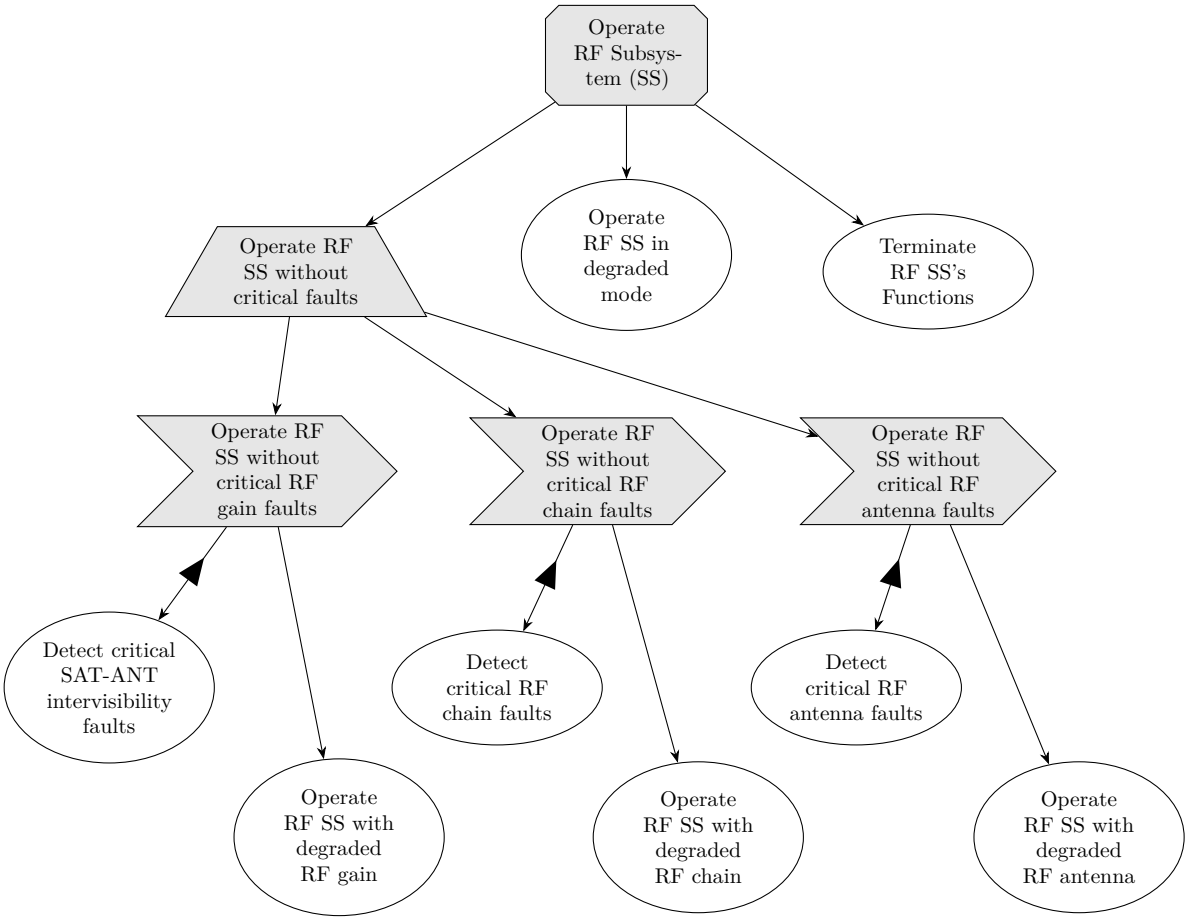


Figure 4.11: Second version of the BT, derived from eliciting the first BT version (Figure 4.10); follows format of Figure 3.1.

As instructed by the second step of the method, each ParallelAll *"Avoid critical faults"* behaviour becomes an *"Operate without critical faults"* Sequence behaviour with an Inverted *"Detect critical faults"* and an *"Operate system in degraded mode"* behaviours as children. That is because, if the system fails to detect critical faults, a "Success" status will be sent to the parent node –because of the Inverter. This will make the Sequence parent node tick the second child, where the system will attempt to operate under non-nominal conditions. We can call this a fail-safe design.

In this last case study we demonstrated the application of the proposed ODM creation methodology in a GS use case, where formal input data were not available, and in general, the use case was not optimal for a formal method application. First of all, the use case was based on an already existing, operational system. Hence, the SysML model was built posterior to the system deployment and operation (not the other way around). Moreover, the FT was not produced as a result of formal FTA activities. Secondly, for the current system, diagnosis is made a posteriori, in the sense that, we don't mitigate faults, but instead try to fix them quickly.

Finally, the ODM's scope is limited to the timeline of one Satellite passage and the consecutive data download, but the actual troubleshooting activities occur between downloads. And this is a limiting contradiction. Nevertheless, we have identified ways of how to improve the system, which is has a crucial added value and potential to system design activities.

4.4 Case Studies Outcomes

In this section we presented the application of the ODM creation methodology in three different use cases. The first uses an SE overloaded model (SysML activity diagram) as input, the second formal SA produced artifacts (FTs), and the third real input data, based on operators' experience of past failures and how they were dealt with, in a form of a hierarchical mind-map.

From the implementation of the method in these three case studies, we can safely draw the following conclusions:

- the first use case, since the lack of failure events and fault propagation data, could only serve as the base of a preliminary case study, to show that BTs can be created from SysML model artifacts; the full proposed ODM creation methodology could not be applied;
- the second use case consisted the easiest case for the model transformation, since formal FT data were available; if we also had formal system design data in our disposal, we could show the full potential of the methodology;
- the third use case presented several drawbacks, and mainly the fact that the available faults' mindmap was not created during system design but during its operation, while combinatory faults and the severity of each fault occurrence was not taken into consideration; we showed that the methodology is still applicable, but some intermediate work is necessary, for the input data to be adapted and be used as inputs for the ODM;
- the application of our methodology led to the formulation of additional functional requirements i.e. "Detect fault behaviours". These requirements, if implemented in the form of functional behaviours, will help detect and mitigate the faults during data download from the Satellite, and prevent downtime;
- this method will help the system architects to better implement the system behaviours, manually, or automatically;
- the proposed methodology can be easily applied and can improve the system architecture design solution.

4.5 Results Summary Discussion

As presented in sections 3.3.1, 3.3.2, 3.3.3, 4.1, 4.2 and 4.3 respectively, each of the performed proof-of-concept and case studies contributed to the gradual evolution and verification of the ODM building approach. In the beginning, the use of BTs as the ODM language semantics was

explored. We thus first needed to investigate whether system design artifacts can indeed be represented by the standard BT language semantics. That is because BTs have never before been used in the domain of model-based design. When we demonstrated this, we could then proceed to defining and developing the ODM creation methodology.

The proof-of-concept studies were used as a step-by-step plan for the feasibility verification of translating SE and SA activities' result artifacts into models that use the language semantics of BTs –as suggested by our ODM proposal, while preserving their original meaning, as imposed by their initial model semantics. Therefore, we used the first proof-of-concept example –see section 3.3.1, to demonstrate the feasibility of translating SE and SA textual artifacts into BTs. Moreover, we were able to show that BT modelling can be dynamic and adaptable to new information coming from different teams, as new nodes can be added –or removed, easily.

In the second proof-of-concept example –see section 3.3.2, we used an AOCS SMD to create corresponding dysfunctional models in Altarica, using the SimfiaNeo Tool. This way we showed how one can express the combination of information derived from SysML and AltaRica visual models to build reciprocal BT models, which represent the same information about the system, and at the same level.

Then, knowing that BT models are executable, we wanted to demonstrate their potential in terms of their exploitation, as much as their creation. For this reason we created a minimal visual BT exploration tool –see section 3.3.3, that can indicate the return status of each BT node at every tick. Moreover, the history of execution was available, while we were also able to implement alarms, as a complementary node health status information. With this example we were able to demonstrate the potential of BTs' exploitation during operations, to help with system supervision and troubleshooting.

We then used the example of the second proof-of-concept exercise to create a more elaborate case study, where we could build sysml models from scratch, and translate them into BT models. With this first case study –see section 4.1, we intended to demonstrate the facility of translation of SysML state machine objects into BT nodes. We did not take this route in our research, this case study however suggests the perspective of creating an algorithm for direct translation of SysML state machine artifacts into BT behaviour nodes.

When we first introduced the idea for automatically translating FT to BT semantics, hence directly deriving operational objectives from system safety analyses' results, it was imperative that we had access to a *formal* and complete FT, which we could use to illustrate our proposed approach. Naturally, the FT was resulting from the safety analysis activities of an existing system scenario i.e. the UAV mission dysfunctional scenario for battery failure, which was used to develop our second case study, as presented in section 4.2.

The last case study was used to demonstrate that the proposed approach is adaptable to any kind of input data, even if not formal. More specifically, the available data were provided in the form of a faults' diagram from ADS. They consisted the result of REX activities from actual

GS operations, and was provided to us by ADS. Having in mind that formal data are rarely available from the system design phase, we wanted to show the realistic aspect of our proposal, as well as the fact that it meets the current needs of the industry. Moreover, we were able to confirm the industrial need for an ODM that was expressed from ADS at the beginning of the thesis, while being able to propose a solution to a pressing matter which is the lack of models oriented to system monitoring and troubleshooting, to help with operational diagnosis. Last, we were able to provide another application field to our methodology –other than space and aerial system, that of ground stations.

A summary of all the six studies performed towards the validation of the ODM creation methodology approach are illustrated in Figure 4.12.

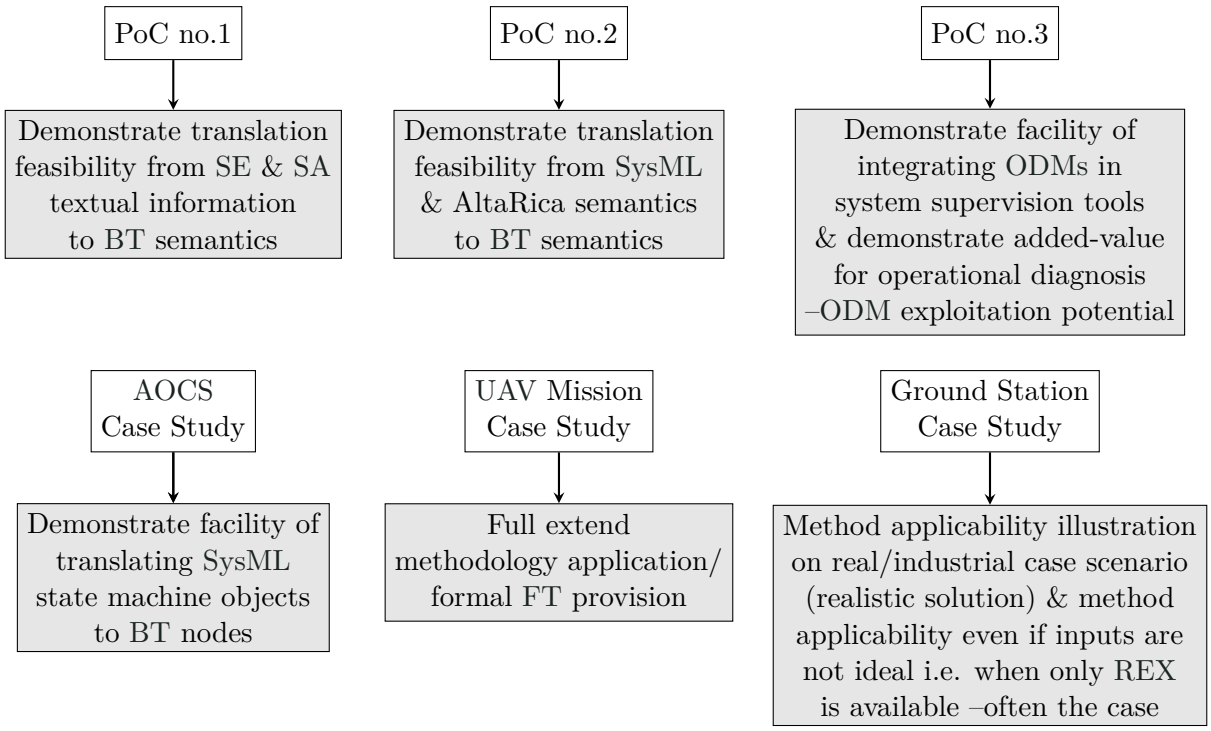


Figure 4.12: Proof-of-Concept and Case studies' contribution to ODM creation methodology's evolution and verification.

Conclusion

This manuscript summarises the work of a PhD thesis which tackles the definition of a new type of model, dedicated to system operations, aiming to improve the efficiency of maintenance activities by its operators so as to reduce their response time to failure, hence accelerating the system recuperation. The topic is motivated by ongoing limitations of the state of the art in the domain of complex systems' development and operation, as well as of the current industrial practise. The initial need for launching this project were expressed by Airbus Defence and Space, who first identified this research gap.

Current limitations in academic research and industrial practice include the following. First, we identified the lack of dedicated models for the operators, that represent the system structural and behavioural architecture, to help them with their diagnostic tasks. The only models available are system design models, which do not provide a holistic view of the system, especially when it comes to faults' occurrence and fault mitigation scenarios. That is partly because dysfunctional models are not made available to operators for system maintenance; even if they did have access, they posses neither the time, nor the skills to jointly exploit system functional and dysfunctional models.

What is more, no practise is in place so far, to ensure the artifact continuance between system design, and operations and maintenance; we recognised the fact that design models are not always synchronised with the final system implementation and operations. System architecture is subject to modifications, following the validation and testing phase, which violates the traceability principle that every part of the implementation should be related to a requirement. Hence design models are no longer valid, nor representative of the system architecture and behaviour, by the time these systems are deployed. We were also able to confirm the last point during the extended bibliographical research undertaken for the needs of this thesis, as well as from the inquiries of industrial partners.

Therefore operators often trust their experience (and not models), to perform diagnosis. This long-gained knowledge and expertise are however not capitalised upon. Little formal practise is in place, to allow the virtual loop between occurring operational incidents and system design. This would allow the continuous improvement of the system's fault mitigation capabilities in the future. By this, we could establish a digital maintenance continuity, to ensure the consistency between the possible faults identified during the system dysfunctional analysis activities and

their actual occurrence in operations (validation of system dysfunctional behaviour as envisaged through dysfunctional analyses).

On the one hand, prior attempts exist to establish this continuity as a-posteriori (to system deployment) solutions, with models able to simulate operational scenarios. These solutions do not generally deal with the issues related to the fault diagnosis and mitigation. On the other hand, models created during system design, that might be available to operators, are far too complex to be understood and exploited by non-modeling experts, such as the operators, which adds a significant difficulty and delay in operators' daily tasks. The approach presented in this manuscript offers a solution that fills the gap in operational modelling and simulation, and proposes a concurrent engineering approach, while being feasible for application within both the academic and industrial community, where lay different needs and skills of both the model designers and their users.

Our work implements a digital continuity strategy, where artifacts produced from system functional and dysfunctional models are utilised for the creation of an Operations-Dedicated Model (ODM). These models are created during the system architecture definition phase. Artifacts might be derived by MBSE and/or MBSA activities (model products), or, for example, RAMS, and/or FMEA/FMECA activities (table/document products), or classical FTA, where Fault Trees (FTs) are built. For the creation of the new type of operational diagnosis-oriented model, we used the language semantics of BT.

We first defined the ODM needs. This helped us determine which are the most suitable language semantics to use for the ODM implementation/representation, namely BTs. This decision was the result of a thorough comparison between all the available candidate language semantics and formal methods. It was then validated by testing its relevance on several "proof of concept" modelling experiments. We then constructed a methodological proposal for the ODM creation, which is using FTs as input data.

The proposed methodology aims to transform system information on faults' occurrence probabilities and propagation, combined with system architecture and operational scenarios, into an executable system model, that includes all the functions that must be in place to ensure the system's nominal operation, or the system's low-risk fault mitigation behaviours. The methodology incorporates two steps, and is defined by a function called FT2BT. The first step consists in an automatic transformation from a FT to a first BT version, which represents the system monitoring objectives. The second step consists in an intuitive elicitation of the first BT version to a second BT version, which represents the system behavioural execution/operational activities, by incorporating system functional information.

We evaluated our proposed methodology on three different use case, to illustrate the method's capabilities, opportunities and limitations. Notably, the first case study was used to test the feasibility of data transformation from SysML model artifacts, to BTs. With the second case study we were able to fully illustrate the application of our methodology to a semi-autonomous complex system mission scenario, using formal FT artifacts as input data. In the third case

study we demonstrated the flexibility and wide applicability scope of our method, by using non-formal dysfunctional data as input, that were provided by a space industrial actor and which were derived by operational REX.

Through these three case studies we manifested that our approach overcomes the main obstacle of current solutions in operations where the available models are too large and not specific to troubleshooting i.e. system simulators. These models include overwhelming and not always useful information, leading to increased complexity for their understanding and simulation hence a waste of time and effort by the operators. The proposed ODM solution provides models that can incrementally reveal information based on the current need (blackbox modelling), while they are easily, directly and quickly executable. More importantly, they are characterised by both dynamic and temporal properties, while they provide an overview of the overall current system status by displaying all individual system states. Furthermore, we showed that no other language semantics or type of model can offer this capabilities' ensemble. Our industrial partners have embraced our solution, and look forward to future work.

Although we produced a solution responding to the given problematic, other departure points could have led to different results. To start with, the gap in current practise was provided since the beginning by a space industrial actor. This meant that we took their need for granted, and transformed it into a research problem. From the beginning we knew that our goal was to create a new type of model –or meta-model, of one sort. This kept us from investigating other possible solution avenues. In particular, within the project context, we took for granted that both system functional and dysfunctional models are available for the system in question, which is, in reality, not often the case. Moreover, the fact that we used FTs as an input artifact for our method introduces many limitations, since FTs are not always produced during the architecture definition phase of a new system, or they are not formally produced. This reduces the added value of the produced BT model, while it reduces the method's applicability scope only to companies which are using specific system development approaches.

Hence other research pathways could have been considered, such as the following. First of all, another approach could propose the definition of generic FT and BT data models, before proposing a transformation methodology, from FTs to BTs. Thus a generic methodology that takes into consideration all possible FT structures (e.g. all gate types, multi-feared events) could have been proposed. However, this would require the provision of a variety of FT examples, from both the academia and the industry, to better understand how FTs are used in real complex systems' development projects, and to which extend. Otherwise, one could consider other system design artifacts (than FTs, or solely FTs) for the ODM creation methodology. This way a more complete approach could have been proposed. An example would be to define the same kind of methodology for the automatic transformation of SysML artifacts into BTs.

Finally, in light of recent advances in literature, we could have shifted our focus to another perspective, such as to connect the ODM methodology to the construction of complex systems' Digital Twin (DT) models. Since the ODM is operator-centric, it can serve as the starting point

for the DT design. Or, since ODMs describe the entire FDIR activities, they can hence serve as the starting point of the FDIR design. Next steps would include the allocation of behaviour to components, and refinement of their interactions. This would enable the automation of some FDIR functions, for example with AI techniques.

Regarding future work opportunities, and in spite of the thesis proposal's contributions to knowledge, in improving the quality and relevance of available data to operators, there are still some perspectives to be considered. Within the impending research avenues, some examples are: refinement of the method, ODM integration in a system monitoring/diagnosis tool, operators' training for the ODM building and exploitation, as well as a deployment of the method in the ongoing process of the development of real systems. This would also contribute in the evaluation of the method's scalability (how large of a system the ODM can handle, and where the benefits of its usage remain significant). Another option would be the application of the ODM creation methodology in an existing, operational system, so as to compare the added value of the use of ODMs for FDD, compared to current practise.

In a broader application perspective, the results of this work could additionally contribute to the field of Asset Management (ISO5500). The ODM creation methodology could notably provide support to the Integrated Management Systems Approach (clause 2.6) [178].

Concerning the validation perspective of the methodological proposal described in this manuscript, one can deduce an approach consisting of the following steps:

1. choose a complex system for which related FTs exist (produced by safety analysis activities);
2. choose the experts who will consist the newly formed ODM team – these experts must fulfill the following criteria:
   (a) have operational experience (have been operators or have worked in the field of operations);
   (b) have diagnosis experience or expertise/skills/knowledge (theoretical, not applied), but ideally operational diagnosis;
   (c) have a basic knowledge of Python;
   (d) be familiar with Fault Trees and Safety/RAMS analyses;
3. have the ODM team apply the ODM creation methodology;
4. construct a visual diagnosis tool for BT exploration & exploitation;
5. have the operators manipulate ODMs with the new diagnostic tool – these operators must:
   (a) have already been responsible for the system's operational maintenance for some months;
   (b) not have a long experience, so they are not already too much used to current practise;

(c) have sufficiently long experience as to be familiar with current practises and their potential drawbacks / possibilities for improvement;

(d) have an open mind for / are receptive towards new ideas / improvement proposals;

6. compare the operators' efficiency in performing troubleshooting with and without the use of the ODM diagnostic tool, through feedback from the operators and from external observation;

7. draw conclusions & define future steps.

We can see this validation roadmap proposal from two different angles. On the one hand, it would allow us to evaluate and improve the ODM creation methodology. On the other, we would be able to evaluate the impact of the use of ODMs for operational diagnosis and its potential influence on the efficiency of operators' daily tasks.

To conclude, the work presented in the manuscript, which was the result of N. Christofi's PhD thesis, consists the first step towards the definition of an Operations-Dedicated Model (ODM). We have provided the limitations of the current research and practise, and proposed a solution that meets the needs expressed by the industry, while filling the existing gap between system design and operational maintenance. Prospective implications for research are at hand. We hope to have partially contributed to future progress in the MBD and operational diagnosis world.

Publications

Throughout the undertaking of the PhD we have published the following journal article(s).

- Nikolena Christofi, Xavier Pucel, Claude Baron, Marc Pantel, Sebastien Guilmeau, Christophe Ducamp. Toward an Operations-Dedicated Model for Space Systems. *Journal of Aerospace Information Systems (JAIS), American Institute of Aeronautics and Astronautics*, `http://arc.aiaa.org/doi/abs/10.2514/1.I011093`, March 2023.
- Under publication: Journal of the International Council on Systems Engineering – Wiley.

The following papers have been presented in international conferences and published in the respective proceedings:

- Nikolena Christofi, Xavier Pucel, David Canu, Jerome Golenzer, Christophe Ducamp. Introducing Operational Diagnosis Models for Ground Station Architectures using Behaviour Trees. *17th International Conference on Space Operations – SpaceOps 2023*, March 2023, Dubai, United Arab Emirates.
- Nikolena Christofi and Xavier Pucel. A Digital Twins Modelling Methodology for System Operations using Fault Trees and Behaviour Trees. *2nd International Workshop on Model-Driven Engineering of Digital Twins – ModDiT'22 / MODELS '22 Companion*, October 2022, Montreal, QC, Canada.
- Nikolena Christofi and Xavier Pucel. From Safety Assessment Models to Operational Diagnosis Models. *33rd International Workshop on Principles of Diagnosis – DX 2022*, September 2022, Toulouse, France.
- Nikolena Christofi, Xavier Pucel, Claude Baron, Marc Pantel, Sébastien Guilmeau, Christophe Ducamp. Towards an agile, model-based multidisciplinary process to improve operational diagnosis in complex systems. *11th European Congress on Embedded Real Time Systems – ERTS 2022*, June 2022, Toulouse, France.
- Nikolena Christofi, Claude Baron, Xavier Pucel, Marc Pantel, Mathilde Machin, Christophe Ducamp. Adopting a model-based approach for satellite operations' diagnosis. *13ème Conférence Internationale de Modélisation, Optimisation et Simulation (MOSIM 2020)*, November 2020, Agadir, Morocco.

Bibliography

[1] Anton Strahilov and Holger Hämmerle. Engineering workflow and software tool chains of automated production systems. In Stefan Biffl, Arndt Lüder, and Detlef Gerhard, editors, *Multi-Disciplinary Engineering for Cyber-Physical Production Systems: Data Models and Software Solutions for Handling Complex Engineering Projects*, pages 207–234. Springer International Publishing, Cham, 2017.

[2] Stefan Biffl, Juergen Musil, Angelika Musil, Kristof Meixner, Arndt Lüder, Felix Rinker, Danny Weyns, and Dietmar Winkler. An industry 4.0 asset-based coordination artifact for production systems engineering. In *2021 IEEE 23rd Conference on Business Informatics (CBI)*, volume 01, pages 92–101, 2021.

[3] Alexander Egyed, Klaus Zeman, Peter Hehenberger, and Andreas Demuth. Maintaining consistency across engineering artifacts. *Computer*, 51(2):28–35, 2018.

[4] European Cooperation for Space Standardization. ECSS-S-ST-00-01C: Glossary of terms, 2012.

[5] Louise Travé-Massuyès, Teresa Escobet, and Xavier Olive. Diagnosability Analysis Based on Component-Supported Analytical Redundancy Relations. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 36:1146–1160, December 2006.

[6] Institute of Electrical International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) and Electronics Engineers. *ISO/IEC/IEEE 15288:2015 - Systems and Software Engineering - System Life Cycle Processes.* International Standards Organization (ISO), Geneva, Switzerland, 2015.

[7] J. C. Laprie. *Dependability: Basic Concepts and Terminology*, pages 3–245. Springer Vienna, 1992.

[8] Dieter Fensel. Relating ontology languages and web standards. *Informatik und Wirtschaftsinformatik. Modellierung*, pages 92–71, 2000.

[9] Alexander Maedche, Hans-Peter Schnurr, Steffen Staab, and Rudi Studer. Representation language-neutral modeling of ontologies. In *Proceedings of the German Workshop" Modellierung-2000". Koblenz, Germany*, pages 129–142. Citeseer, 2000.

[10] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[11] US Military. Mil-std 882b system safety program requirement. *Washington: US Department of Defense*, pages 2465–562, 1984.

[12] Institute of Electrical International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) and Electronics Engineers. *ISO/IEC/IEEE 21839: 2019 - Systems and software engineering–System of systems (SoS) considerations in life cycle stages of a system*. International Standards Organization (ISO), Geneva, Switzerland, 2019.

[13] Mark W. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284, 1998.

[14] National Aeronautics and Space Administration (NASA). NASA 2022 Strategic Plan. https://www.nasa.gov/sites/default/files/atoms/files/fy_22_strategic_plan.pdf, 2022. Last accessed October 2022.

[15] National Aeronautics and Space Administration (NASA). NASA's 2018 Strategic Plan. https://www.nasa.gov/sites/default/files/atoms/files/nasa_2018_strategic_plan.pdf, 2018.

[16] European Space Agency (ESA). ESA's Technology Strategy for Space19+. https://esamultimedia.esa.int/docs/technology/ESA_Technology_Strategy_Version_1_0.pdf, 2019.

[17] Narayan Prasad. An overview of on-board computer (OBC) systems available on the global space marketplace. https://blog.satsearch.co/2020-03-11-an-overview-of-on-board-computer-obc-systems-available-on-the-global-space-marketplace, 2021. Last accessed November 2021.

[18] Xavier Olive. FDI(R) for satellites: How to deal with high availability and robustness in the space domain? *International Journal of Applied Mathematics and Computer Science*, 22(1):99–107, March 2012.

[19] Martin S Feather and Lawrence Z Markosian. Towards Certification of a Space System Application of Fault Detection and Isolation. page 4, 2008.

[20] Bob Ferrell, Mark Lewis, Rebecca Oostdyk, Jesse Goerz, Jose Perotti, and Barbara Brown. Lessons Learned on Implementing Fault Detection, Isolation, and Recovery (FDIR) in a Ground Launch Environment. In *AIAA Infotech@Aerospace 2010*, Atlanta, Georgia, April 2010. American Institute of Aeronautics and Astronautics.

[21] Joost-Pieter Katoen. Towards Trustworthy Aerospace Systems: An Experience Report. In Gwen Salaün and Bernhard Schätz, editors, *Formal Methods for Industrial Critical Systems*, volume 6959, pages 1–4. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[22] Christopher Edwards, Thomas Lombaerts, Hafid Smaili, et al. Fault tolerant flight control. *Lecture Notes in Control and Information Sciences*, 399:1–560, 2010.

[23] Joost-Pieter Katoen and Thomas Noll. Trustworthy Aerospace Systems. *Public service review*, 11:204–205, 2011.

[24] Azad M. Madni, Marcus Nance, Michael Richey, William Hubbard, and Leroy Hanneman. Toward an experiential design language: Augmenting model-based systems engineering with technical storytelling in virtual worlds. *Procedia Computer Science*, 28:848–856, 2014. 2014 Conference on Systems Engineering Research.

[25] German Aerospace Center (DLR). Virtual Satellite 4. `https://github.com/virtualsatellite/VirtualSatellite4-Core`, 2019. Last accessed September 2022.

[26] Holger Schumann, Andy Braukhane, A Gernd, JT Grundmann, R Hempel, B Kazeminejad, O Romberg, and M Sippel. Overview of the new concurrent engineering facility at dlr, 2008.

[27] Concurrent Design Facility (CDF). `https://technology.esa.int/lab/concurrent-design-facility`, 2022. Last accessed October 2022.

[28] Reinhold Bertranda, Jennifer Hoffmannb, Nieves S. Moralc, and Marcus Wallumd. Concurrent engineering for ground segment and operations conceptual design – use cases, methods and tools on the way to digitalisation. In *73rd International Astronautical Congress (IAC)*. International Astronautical Federation (IAF), 2022.

[29] Todd J. Bayer, Seung Chung, Bjorn Cole, Brian Cooke, Frank Dekens, Chris Delp, I. Gontijo, Kari Lewis, Mehrdad Moshir, Robert Rasmussen, and Dave Wagner. Model based systems engineering on the europa mission concept study. In *2012 IEEE Aerospace Conference*, pages 1–18, 2012.

[30] Elyse Fosse, Ann Devereaux, Corey Harmon, and Mallory Lefland. Inheriting curiosity: Leveraging mbse to build mars2020. In *AIAA SPACE 2015 Conference and Exposition*, 2015.

[31] Shoumen Palit Austin Datta. Emergence of digital twins. *Journal of Information Management*, 5:14–34, 2017.

[32] Tsega Y. Melesse, Valentina Di Pasquale, and Stefano Riemma. Digital twin models in industrial operations: State-of-the-art and future research directions. *IET Collaborative Intelligent Manufacturing*, 3(1):37–47, 2021.

[33] Michael Schluse, Linus Atorf, and Juergen Rossmann. Experimentable digital twins for model-based systems engineering and simulation-based development. In *2017 Annual IEEE International Systems Conference (SysCon)*, pages 1–8, 2017.

[34] Azad M. Madni and Michael Sievers. Model-based systems engineering: Motivation, current status, and research opportunities. *Systems Engineering*, 21(3):172–190, 2018.

[35] Maria Paola Cabasino, Alessandro Giua, Laura Marcias, and Carla Seatzu. A comparison among tools for the diagnosability of discrete event systems. In *2012 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 218–223, 2012.

[36] S. Hashtrudi Zad, R.H. Kwong, and W.M. Wonham. Fault diagnosis in discrete-event systems: framework and model reduction. In *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171)*, volume 4, pages 3769–3774 vol.4, 1998.

[37] Stavros Tripakis. Fault diagnosis for timed automata. In Werner Damm and Ernst Rüdiger Olderog, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 205–221, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[38] K.B. Ramkumar, P. Philips, H.A. Presig, W.K. Ho, and K.W. Lim. Structured fault-detection and diagnosis using finite-state automaton. In *IECON '98. Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society (Cat. No.98CH36200)*, volume 3, pages 1667–1672 vol.3, 1998.

[39] Ferdinand Settele, Alexander Weber, and Alexander Knoll. Plant model-based fault detection during aircraft takeoff using non-deterministic finite-state automata. *Aerospace*, 7(8), 2020.

[40] Francesco Basile. Overview of fault diagnosis methods based on petri net models. In *2014 European Control Conference (ECC)*, pages 2636–2642, 2014.

[41] Maria Pia Fanti and Carla Seatzu. Fault diagnosis and identification of discrete event systems using petri nets. In *2008 9th International Workshop on Discrete Event Systems*, pages 432–435, 2008.

[42] Janan Zaytoon and Stéphane Lafortune. Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2):308–320, 2013.

[43] Thushara Ekanayake, Devapriya Dewasurendra, Sunil Abeyratne, Lin Ma, and Prasad Yarlagadda. Model-based fault diagnosis and prognosis of dynamic systems: a review. *Procedia Manufacturing*, 30:435–442, 2019. Digital Manufacturing Transforming Industry Towards Sustainable Growth.

[44] Erik Frisk and Lars Nielsen. Robust residual generation for diagnosis including a reference model for residual behavior. *Automatica*, 42(3):437–445, 2006.

[45] Michele Colledanchise and Petter Ögren. Behavior Trees in Robotics and AI: An Introduction. *arXiv:1709.00084 [cs]*, july 2018. arXiv: 1709.00084.

[46] Alex Champandard. Understanding behavior trees. *AiGameDev.com*, 6(328):19, 2007.

[47] Damian Isla. Handling complexity in the halo 2 ai. In *Game Developers Conference (GDC), San Francisco*, 2005.

[48] Damian Isla. Building a better battle. In *Game Developers Conference (GDC), San Francisco*, volume 32, 2008.

[49] Ramiro A. Agis, Sebastian Gottifredi, and Alejandro J. García. An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games. *Expert Systems with Applications*, 155:113457, October 2020.

[50] Petter Ogren. Increasing modularity of uav control systems using computer game behavior trees. In *AIAA guidance, navigation, and control conference*, page 4458, 2012.

[51] Andreas Klöckner. Interfacing Behavior Trees with the World Using Description Logic. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, Guidance, Navigation, and Control and Co-located Conferences. American Institute of Aeronautics and Astronautics, August 2013.

[52] Andreas Klöckner. Behavior Trees for UAV Mission Management. In Matthias Horbach, editor, *INFORMATIK 2013: Informatik angepasst an Mensch, Organisation und Umwelt*, volume P-220, pages 57–68, Koblenz, Germany, September 2013. Köllen Druck + Verlag GmbH, Bonn. ISSN: 1617-5468 Meeting Name: INFORMATIK 2013: Informatik angepasst an Mensch, Organisation und Umwelt.

[53] Andreas Klöckner. The Modelica BehaviorTrees Library: Mission Planning in Continuous-Time for Unmanned Aircraft. In Hubertus Tummescheit and Karl-Erik Arzén, editors, *Proceedings of the 10th International Modelica Conference - Lund, Sweden - Mar 10-12, 2014*, pages 727–736, Lund, Sweden, December 2014. Linköping University Electronic Press. ISSN: 1650-3686.

[54] Alejandro Marzinotto, Michele Colledanchise, Christian Smith, and Petter Ögren. Towards a Unified BTs Framework for Robot Control. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, page 8, 2014.

[55] J. Andrew Bagnell, Felipe Cavalcanti, Lei Cui, Thomas Galluzzo, Martial Hebert, Moslem Kazemi, Matthew Klingensmith, Jacqueline Libby, Tian Yu Liu, Nancy Pollard, Mihail Pivtoraiko, Jean-Sebastien Valois, and Ranqi Zhu. An integrated system for autonomous robotics manipulation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2955–2962, 2012.

[56] Aleksandras Melnikovas. Towards an explicit research methodology: Adapting research onion model for futures studies. *Journal of Futures Studies*, 23(2):29–44, 2018.

[57] Mark Saunders, Philip Lewis, Adrian Thornhill, and Alex Bristow. *"Research Methods for Business Students" - Chapter 4: Understanding research philosophy and approaches to theory development*, pages 128–171. 03 2019.

[58] Dr Mike Hinchey, Caroline Wang, and Josh McNeil. Formal Methods for System/Software Engineering: NASA & Army Experiences. *Formal Methods*, page 22, 2011.

[59] C. Seidner and O.H. Roux. Formal Methods for Systems Engineering Behavior Models. *IEEE Transactions on Industrial Informatics*, 4(4):280–291, november 2008.

[60] ECSS-E-ST-10C Rev.1 Standard. System engineering general requirements, 15 February 2017.

[61] INCOSE. Systems Engineering Handbook, version 2a, 2004.

[62] NASA/SP-2007-6105 Rev1. NASA Systems Engineering Handbook, 1995. SP-610S.

[63] INCOSE. International Council on Systems Engineering. https://www.incose.org/. Last accessed Apr. 17, 2022.

[64] R. F. Schmidt. Ieee p122 - standard for application and management of the systems engineering process. In *[1993 Proceedings] AIAA/IEEE Digital Avionics Systems Conference*, pages 6–11, 1993.

[65] Airbus Group. Digital Design, Manufacturing & Services: Transforming Airbus through digital continuity. https://www.airbus.com/en/innovation/disruptive-concepts/digital-design-manufacturing-services, 2022. Last accessed November 2022.

[66] Howard Cotterman, Kevin Forsberg, and Hal Mooz. *Visualizing project management: models and frameworks for mastering complex systems*. John Wiley & Sons, 2005.

[67] Kevin Forsberg and Harold Mooz. The relationship of system engineering to the project cycle. *INCOSE International Symposium*, 1(1):57–65, 1991.

[68] David D. Walden, Garry J. Roedler, and Kevin Forsberg. Incose systems engineering handbook version 4: Updating the reference for practitioners. *INCOSE International Symposium*, 25(1):678–686, 2015.

[69] Louis Wheatcraft, Carl Svensson, and Mike Ryan. Developing a systems engineering capability that meets the needs of your organization. *INCOSE International Symposium*, 27:1438–1455, 07 2017.

[70] Alessandro Landi and Mark Nicholson. Arp4754a/ed-79a-guidelines for development of civil aircraft and systems-enhancements, novelties and key topics. *SAE International Journal of Aerospace*, 4(2011-01-2564):871–879, 2011.

[71] S-18 Aircraft & System Develoment and Safety Assessment Committee. *ARP4761 - Guidelines and Methods for conducting the Safety assessment process on civil airborne systems and equipment*. SAE International, December 1996.

[72] International Council on Systems Engineering (INCOSE). Incose systems engineering vision 2025 – a world in motion, 2014.

[73] INCOSE-TP-2004-004-02. INCOSE Technical proceedings / INCOSE SE Vision 2020, September 2007.

[74] Object Management Group (OMG). `https://www.omg.org/about/index.htm`. Last accessed October 2022.

[75] A. Wayne Wymore. *Model-based systems engineering : an introduction to the mathematical theory of discrete systems and to the tricotyledon theory of system design*. CRC press, 1993.

[76] UML-SysML. `https://sysml.org/`. Last accessed October 2022.

[77] OMG-UPDM. `https://www.omg.org/spec/UPDM`. Last accessed October 2022.

[78] The Modelica Association. `https://www.modelica.org/`. Last accessed October 2022.

[79] Hyde, Randall. *Write Great Code, Vol. 2: Thinking Low-Level, Writing High-Level*, volume 2. No Starch Press, 2004.

[80] Model Based Systems Engineering (MBSE) Initiative. INCOSE / OMG. `http://www.omgwiki.org/MBSE/doku.php`. Last accessed October 2022.

[81] Object Management Group (OMG). Systems Modeling Language (SysML). `https://www.omg.org/spec/SysML/Current`, 2021. Last accessed November 2021.

[82] Albert Albers and Christian Zingel. Challenges of model-based systems engineering: A study towards unified term understanding and the state of usage of sysml. In *Proceedings of the CIRP Design Conference*. Springer, 01 2013.

[83] JPL Estefan, Jeff A. Survey of Model-Based Systems Engineering (MBSE) Methodologies. *An INCOSE Initiative*, page 70, 2008.

[84] Lenny Delligatti. *SysML distilled: A brief guide to the systems modeling language*. Addison-Wesley, 2013.

[85] J. Martin. *Systems Engineering Guidebook: A Process for Developing Systems and Products*. CRC Press, 1996.

[86] Shashank P. Alai. *Evaluating ARCADIA/Capella vs. OOSEM/SysML for System Architecture Development*. PhD thesis, Purdue University, Indianapolis, Indiana, 8 2019.

[87] Mary Bone and Robert Cloutier. The current state of model based systems engineering: Results from the omg™ sysml request for information 2009. In *Proceedings of the 8th conference on systems engineering research*, 2010.

[88] Mark Sampson. Guiding principals for systems engineering tool deployment. In *INCOSE International Symposium*, volume 11, pages 308–315. Wiley Online Library, 2001.

[89] Daniele Gianni, Andrea D'Ambrogio, and Andreas Tolk, editors. *Modeling and Simulation-Based Systems Engineering Handbook*. CRC Press, 0 edition, October 2018.

[90] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[91] Paolo Bocciarelli, Alessandra Pieroni, Daniele Gianni, and Andrea D'Ambrogio. A model-driven method for building distributed simulation systems from business process models. In *Proceedings of the 2012 winter simulation conference (WSC)*, pages 1–12. IEEE, 2012.

[92] Fabien Bouffaron. Airbus MBSE Framework : Model Execution of System Architectures (MOFLT). In *MBSE Cyber Experience Symposium 2021 - JAPAN*, ONLINE, Japan, September 2021.

[93] Thales. ARChitecture Analysis & Design Integrated Approach (ARCADIA). https://www.eclipse.org/capella/arcadia.html. Last accessed Apr. 17, 2022.

[94] Jean-Luc Voirin. *Model-based System and Architecture Engineering with the Arcadia Method*. Elsevier, 2017.

[95] Pascal Roques. *Systems Architecture Modeling with the Arcadia Method: A Practical Guide to Capella*. Elsevier, 2017.

[96] Pascal Roques. Mbse with the arcadia method and the capella tool. In *8th European Congress on Embedded Real Time Software and Systems (ERTS)*, 2016.

[97] Object-Oriented SE Method. https://www.incose.org/incose-member-resources/working-groups/transformational/object-oriented-se-method. Last accessed October 2022.

[98] IBM Telelogic Harmony-SE. https://www.omgwiki.org/MBSE/doku.php?id=mbse:harmonyse. Last accessed October 2022.

[99] Pattern-Based Systems Engineering (PBSE) / SystematicaTM. https://www.omgwiki.org/MBSE/doku.php?id=mbse:pbse. Last accessed October 2022.

[100] Michel D Ingham, Robert D Rasmussen, Matthew B Bennett, and Alex C Moncada. Engineering complex embedded systems with state analysis and the mission data system. *Journal of Aerospace Computing, Information, and Communication*, 2(12):507–536, 2005.

[101] No Magic. Cameo Systems Modeler. http://www.nomagic.com/products/cameo-systems-modeler.html. Last accessed Apr. 17, 2022.

[102] A. Joshi, S.P. Miller, M. Whalen, and M.P.E. Heimdahl. A proposal for model-based safety analysis. In *24th Digital Avionics Systems Conference*, volume 2, pages 13 pp. Vol. 2–, 2005.

[103] Yiannis Papadopoulos and John A McDermid. Hierarchically performed hazard origin and propagation studies. In *International conference on computer safety, reliability, and security*, pages 139–152. Springer, 1999.

[104] Oleg Lisagor, Linling Sun, and Tim Kelly. The illusion of method: Challenges of model-based safety assessment. In *28th international system safety conference (ISSC)*, 2010.

[105] Yiannis Papadopoulos, Martin Walker, David Parker, Erich Rüde, Rainer Hamann, Andreas Uhlig, Uwe Grätz, and Rune Lien. Engineering failure analysis and design optimisation with hip-hops. *Engineering Failure Analysis*, 18(2):590–608, 2011.

[106] Duane Kritzinger. Fault tree analysis. In Duane Kritzinger, editor, *Aircraft System Safety*, chapter 4, pages 59–99. Woodhead Publishing, 2017.

[107] Bernhard Kaiser, Peter Liggesmeyer, and Oliver Mäckel. A new component concept for fault trees. In *Proceedings of the 8th Australian workshop on Safety critical systems and software-Volume 33*, pages 37–46, 2003.

[108] Oleg Lisagor, Tim Kelly, and Ru Niu. Model-based safety assessment: Review of the discipline and its challenges. In *The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*, pages 625–632, 2011.

[109] Rodgers William P. *Introduction to system safety engineering.* Wiley series on systems engineering and analysis. Wiley, New York, 1971.

[110] Roland Harold E. 1924. *System safety engineering and management.* Wiley, New York, 2e edition edition, 1990.

[111] Nancy Leveson. Software safety: why, what, and how. *ACM Computing Surveys (CSUR)*, 18(2):125–163, 1986.

[112] John E. Hosford. Measures of dependability. *Operations Research*, 8(1):53–64, 1960.

[113] D Prasad, J Mcdermid, and I Wand. Dependability terminology: similarities and differences. *IEEE Aerospace and Electronic Systems Magazine*, 11(1):14–21, 1996.

[114] Jean-Claude Laprie and Alain Costes. Dependable computing and fault tolerance at laas: a summary. In Algirdas Avižienis, Hermann Kopetz, and Jean-Claude Laprie, editors, *The Evolution of Fault-Tolerant Computing*, pages 193–213, Vienna, 1987. Springer Vienna.

[115] J. C. Laprie. *Dependability: Basic Concepts and Terminology*, pages 3–245. Springer Vienna, Vienna, 1992.

[116] INCOSE - International Council on Systems Engineering. Guide to the Systems Engineering Body of Knowledge (SEBoK). https://www.sebokwiki.org/, 2021. Last accessed November 2021.

[117] Étienne André, Didier Lime, Mathias Ramparison, and Mariëlle Stoelinga. Parametric Analyses of Attack-fault Trees. *Fundamenta Informaticae*, 182(1):69 – 94, September 2021. This manuscript is the author version of the manuscript of the same name published in Fundamenta Informatica 182(1).This manuscript is an extended version of the manuscript of the same name published in the proceedings of the 19th International Conference on Application of Concurrency to System Design (ACSD 2019).

[118] Rausand, Marvin. *Reliability Quantification*, chapter 5, pages 91–164. John Wiley & Sons, Ltd, 2014.

[119] Yong Bai and Qiang Bai. Subsea risk and reliability. In Yong Bai and Qiang Bai, editors, *Subsea Engineering Handbook (Second Edition)*, chapter 10, pages 239–261. Gulf Professional Publishing, Boston, second edition edition, 2019.

[120] Frank Wabnitz and Houston Netherland. Use of reliability engineering tools to enhance subsea system reliability. In *Offshore Technology Conference*. OnePetro, 2001.

[121] Denning, Richard. Applied R&M Manual for Defence Systems, Part C - Techniques, Chapter - 29 Fault/Success Tree Analysis. *Ministry of Defence*, 2012.

[122] Sohag Kabir. An overview of fault tree analysis and its application in model based dependability analysis. *Expert Systems with Applications*, 77:114–135, July 2017.

[123] Mathilde Machin, Estelle Saez, Pierre Virelizier, and Xavier de Bossoreille. Modeling Functional Allocation in AltaRica to Support MBSE/MBSA Consistency. In Yiannis Papadopoulos, Koorosh Aslansefat, Panagiotis Katsaros, and Marco Bozzano, editors, *Model-Based Safety and Assessment*, pages 3–17, Cham, 2019. Springer International Publishing.

[124] Michel Batteux, Tatiana Prosvirnova, Antoine Rauzy, and Leïla Kloul. The AltaRica 3.0 project for model-based safety assessment. In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 741–746. IEEE, 2013.

[125] Anjali Joshi, Mats PE Heimdahl, Steven P Miller, and Mike W Whalen. Model-based safety analysis. *NASA/CR-2006-213953*, 2006.

[126] Peter Fenelon and John A McDermid. An integrated tool set for software safety analysis. *Journal of Systems and Software*, 21(3):279–290, 1993. Applying Specification, Verification, and Validation Techniques to Industrial Software Systems.

[127] Yiannis Papadopoulos, Martin Walker, David Parker, Septavera Sharvia, Leonardo Bottaci, Sohag Kabir, Luis Azevedo, and Ioannis Sorokos. A synthesis of logic and bio-inspired techniques in the design of dependable systems. *Annual Reviews in Control*, 41:170–182, 2016.

[128] André Arnold, Gérald Point, Alain Griffault, and Antoine Rauzy. The altarica formalism for describing concurrent systems. *Fundamenta Informaticae*, 40(2-3):109–124, 1999.

[129] Pierre Bieber, Charles Castel, and Christel Seguin. Combination of fault tree analysis and model checking for safety assessment of complex system. In Andrea Bondavalli and Pascale Thevenod-Fosse, editors, *Dependable Computing EDCC-4*, pages 19–31, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[130] Mathilde Machin, Laurent Sagaspe, and Xavier de Bossoreille. Simfianeo, complex systems, yet simple safety. *Embedded Real Time Software and Systems (ERTS)*, 2018.

[131] Pierre Bieber, Jean-Paul Blanquart, Guy Durrieu, David Lesens, J Lucotte, Frederic Tardy, Michel Turin, Christel Seguin, and Eric Conquet. Integration of formal fault analysis in assert: Case studies and lessons learnt. 12 2022.

[132] Marco Bozzano and Adolfo Villafiorita. The fsap/nusmv-sa safety analysis platform. *International Journal on Software Tools for Technology Transfer*, 9(1):5–24, 2007.

[133] Peter Feiler, David Gluch, and John Hudak. The architecture analysis & design language (aadl): An introduction. page 145, 02 2006.

[134] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, September 1995.

[135] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57 – 95, 1987.

[136] Johan Kleer and James Kurien. Fundamentals of model-based diagnosis. *IFAC Proceedings Volumes*, 36, 06 2003.

[137] Mark Schwabacher and Kai Goebel. A survey of artificial intelligence for prognostics. In *AAAI Fall Symposium: Artificial Intelligence for Prognostics*, pages 108–115, 2007.

[138] Johan de Kleer and James Kurien. Fundamentals of model-based diagnosis. *IFAC Proceedings Volumes*, 36(5):25–36, 2003. 5th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes 2003, Washington DC, 9-11 June 1997.

[139] Yaguo Lei, Bin Yang, Xinwei Jiang, Feng Jia, Naipeng Li, and Asoke K. Nandi. Applications of machine learning to machine fault diagnosis: A review and roadmap. *Mechanical Systems and Signal Processing*, 138:106587, 2020.

[140] Yannick Pencolé, Elodie Chanthery, and Thierry Peynot. Definition of Model-based diagnosis problems with Altarica. In *27th International Workshop on Principles of Diagnosis (DX-2016)*, page 8p., Denver, CO, United States, October 2016.

[141] Francesco Basile, Pasquale Chiacchio, and Gianmaria De Tommasi. An Efficient Approach for Online Diagnosis of Discrete Event Systems. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, 54(4):12, 2009.

[142] Michele Colledanchise and Petter Ögren. How Behavior Trees modularize robustness and safety in hybrid systems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1482–1488, September 2014. ISSN: 2153-0866.

[143] Michele Colledanchise, Alejandro Marzinotto, Dimos V. Dimarogonas, and Petter Oegren. The Advantages of Using Behavior Trees in Mult-Robot Systems. In *Proceedings of ISR 2016: 47st International Symposium on Robotics*, pages 1–8, June 2016.

[144] Francesco Rovida, Bjarne Grossmann, and Volker Krüger. Extended behavior trees for quick definition of flexible robotic tasks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6793–6800, September 2017. ISSN: 2153-0866.

[145] Michele Colledanchise, Ramviyas Parasuraman, and Petter Ögren. Learning of Behavior Trees for Autonomous Agents. *IEEE Transactions on Games*, 11(2):183–189, June 2019. Conference Name: IEEE Transactions on Games.

[146] Michele Colledanchise and Petter Ögren. How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees. *IEEE Transactions on Robotics*, 33(2):372–389, April 2017. Conference Name: IEEE Transactions on Robotics.

[147] Sergio García, Patrizio Pelliccione, Claudio Menghi, Thorsten Berger, and Tomas Bures. High-level mission specification for multiple robots. In *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2019, pages 127–140, New York, NY, USA, October 2019. Association for Computing Machinery.

[148] PyTrees Library Documentation. `https://py-trees.readthedocs.io/en/devel/`. Last accessed Apr. 17, 2022.

[149] BehaviorTree.CPP - Behavior Trees Library in C++. `https://github.com/BehaviorTree/BehaviorTree.CPP`. Last accessed Apr. 17, 2022.

[150] Groot - Graphical Editor to create Behavior Trees. Compliant with BehaviorTree.CPP. `https://github.com/BehaviorTree/Groot`. Last accessed Apr. 17, 2022.

[151] Harald Van der Werff and Freek Van der Meer. Sentinel-2a msi and landsat 8 oli provide data continuity for geological remote sensing. *Remote Sensing*, 8(11), 2016.

[152] James R. Irons, John L. Dwyer, and Julia A. Barsi. The next landsat satellite: The landsat data continuity mission. *Remote Sensing of Environment*, 122:11–21, 2012. Landsat Legacy Special Issue.

[153] Nikolai Mansourov and Djenana Campara. Chapter 3 - how to build confidence. In Nikolai Mansourov and Djenana Campara, editors, *System Assurance*, The MK/OMG Press, pages 49–80. Morgan Kaufmann, Boston, 2011.

[154] Andrew Waterson and Alun Preece. Verifying ontological commitment in knowledge-based systems. *Knowledge-Based Systems*, 12:45–54, April 2000.

[155] Stephen Cranefield and Martin K. Purvis. Uml as an ontology modelling language. In *Intelligent Information Integration*, 1999.

[156] Egidio Astesiano and Gianna Reggio. Formalism and method. In Michel Bidoit and Max Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development*, pages 93–114, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[157] Stefan Farfeleder, Thomas Moser, Andreas Krall, Tor Stålhane, Herbert Zojer, and Christian Panis. Dodt: Increasing requirements formalism using domain ontologies for improved embedded systems development. In *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 271–274, 2011.

[158] Joerg Evermann and Yair Wand. Ontology based object-oriented domain modelling: fundamental concepts. *Requirements engineering*, 10(2):146–160, 2005.

[159] Nikolena Christofi, Claude Baron, Xavier Pucel, Marc Pantel, Mathilde Machin, and Christophe Ducamp. Adopting a model-based approach for satellite operations' diagnosis. In *13ème Conférence Internationale de Modélisation, Optimisation et Simulation (MOSIM 2020)*, 2020.

[160] Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. On the need for megamodels. In *OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 1-9*, 2004.

[161] Ruth Sara Aguilar-Savén. Business process modelling: Review and framework. *International Journal of Production Economics*, 90(2):129–149, 2004. Production Planning and Control.

[162] Thomas Allweyer. *BPMN 2.0: introduction to the standard for business process modeling.* BoD–Books on Demand, 2016.

[163] Cristina Venera Geambaşu. Bpmn vs. uml activity diagram for business process modeling. In *Proceedings of the 7th International Conference Accounting and Management Information Systems AMIS*, pages 934–945, 2012.

[164] Wolfgang Herzner, Sven Sieverding, Omar Kacimi, Eckard Böde, Thomas Bauer, and Brian Nielsen. Expressing best practices in (risk) analysis and testing of safety-critical systems using patterns. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, pages 299–304, 2014.

[165] Morayo Adedjouma and Nataliya Yakymets. A framework for model-based dependability analysis of cyber-physical systems. In *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, pages 82–89, 2019.

[166] Xi Chen and Jian Jiao. A fault propagation modeling method based on a finite state machine. In *2017 Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–7, 2017.

[167] Michelle Zhu and R. Brooks. Comparison of petri net and finite state machine discrete event control of distributed surveillance network. *IJDSN*, 5:480–501, 09 2009.

[168] Michele Colledanchise and Petter Ögren. How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Transactions on Robotics*, 33(2):372–389, 2017.

[169] Marc Bouissou and Jean-Louis Bon. A new formalism that combines advantages of fault-trees and markov models: Boolean logic driven markov processes. *Reliability Engineering & System Safety*, 82(2):149–163, 2003.

[170] Alexandre Albore, Silvano Dal Zilio, Guillaume Infantes, Christel Seguin, and Pierre Virelizier. A Model-Checking Approach to Analyse Temporal Failure Propagation with AltaRica. In *Model-Based Safety and Assessment. IMBSA 2017*, volume 10437 of *Lecture Notes in Computer Science*, page 15p., Trento, Italy, September 2017.

[171] Marie de Roquemaruel. Cameo "GNC Auto" Model, courtesy of MOISE Project. `https://se.pf.irt-saintexupery.com/-/ide/project/nikolena.christofi/phd/tree/master/-/satellite/cameo_model/gnc_auto.mdzip/`, 2022. Last accessed December 2022.

[172] European Cooperation for Space Standardization (ECSS). ECSS-Q-ST-30-02C: Failure modes, effects (and criticality) analysis (FMEA/FMECA), 2009.

[173] Nikolena Christofi. SimfiaNeo Satellite Model. `https://se.pf.irt-saintexupery.com/-/ide/project/nikolena.christofi/phd/tree/master/-/satellite/simfia_neo_model/`, 2022. Last accessed December 2022.

[174] Nikolena Christofi. Simfia Neo Satellite Model AltaRica Code. `https://se.pf.irt-saintexupery.com/-/ide/project/nikolena.christofi/phd/t`

ree/master/-/satellite/simfia_neo_model/sat.alt/, 2022. Last accessed December 2022.

[175] European Cooperation for Space Standardization (ECSS). ECSS-Q-ST-30-09C: Space product assurance: Availability Analysis, 2008.

[176] Augustus De Morgan. *Formal logic: or, the calculus of inference, necessary and probable.* Taylor and Walton, 1847.

[177] Nikolena Christofi. Ground Station SysML Model. https://se.pf.irt-saintexupery.com/-/ide/project/nikolena.christofi/phd/tree/master/-/ground_station/cameo_model/ground_station.mdzip/, 2022. Last accessed December 2022.

[178] International Standards Organization (ISO). ISO 55000: Asset Management – Overview, principles and terminology, 2.6 - Integrated management systems approach, 2014.

Annex A - Proof of Concept 2

GNC State Machine Diagram

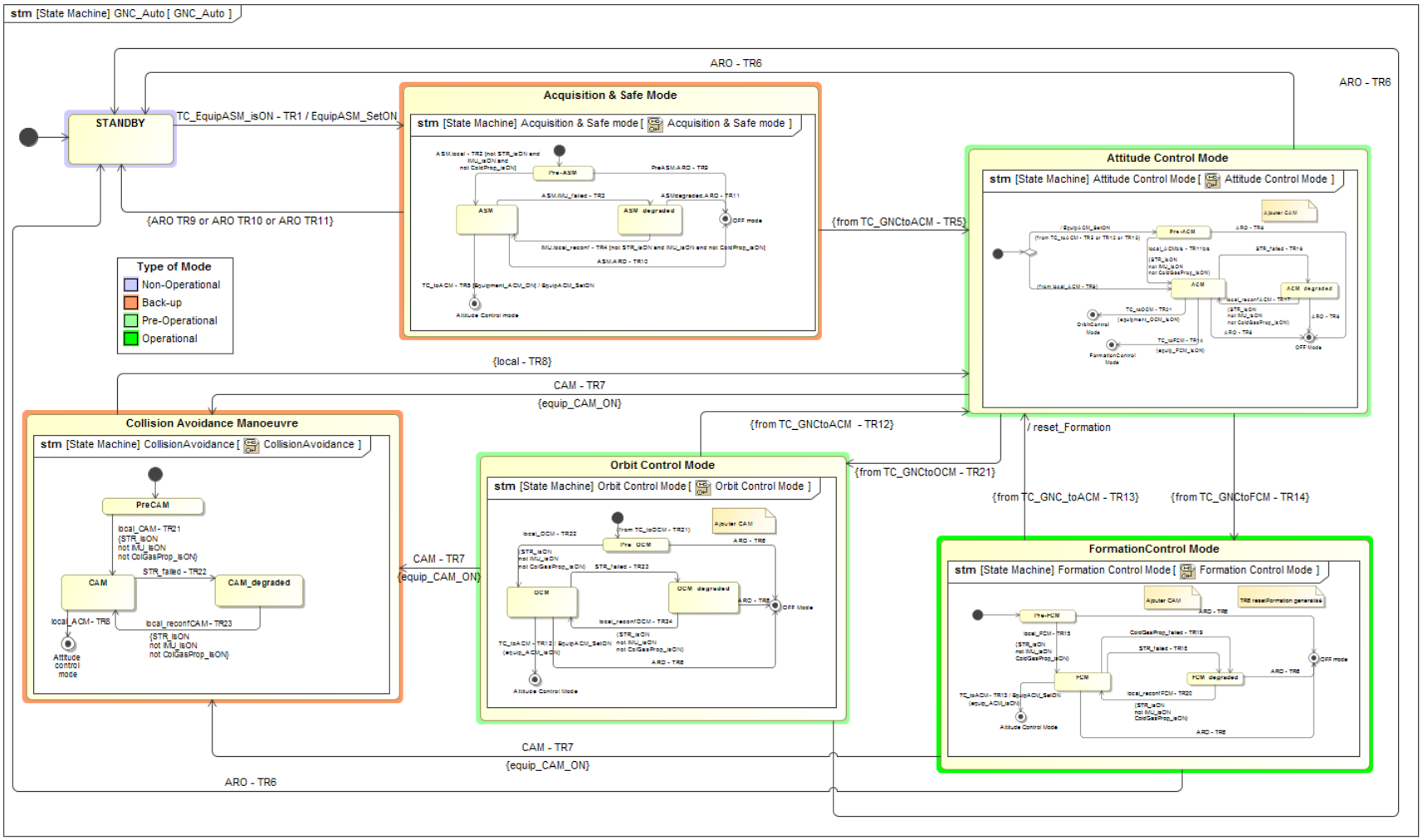


Figure 5.1: GNC State Machine Diagram (WhiteBox View). Courtesy of the MOISE project.

Annex B - Ground Station Case Study

RF Subsystem Architecture &
Full Fault Propagation Tree Map

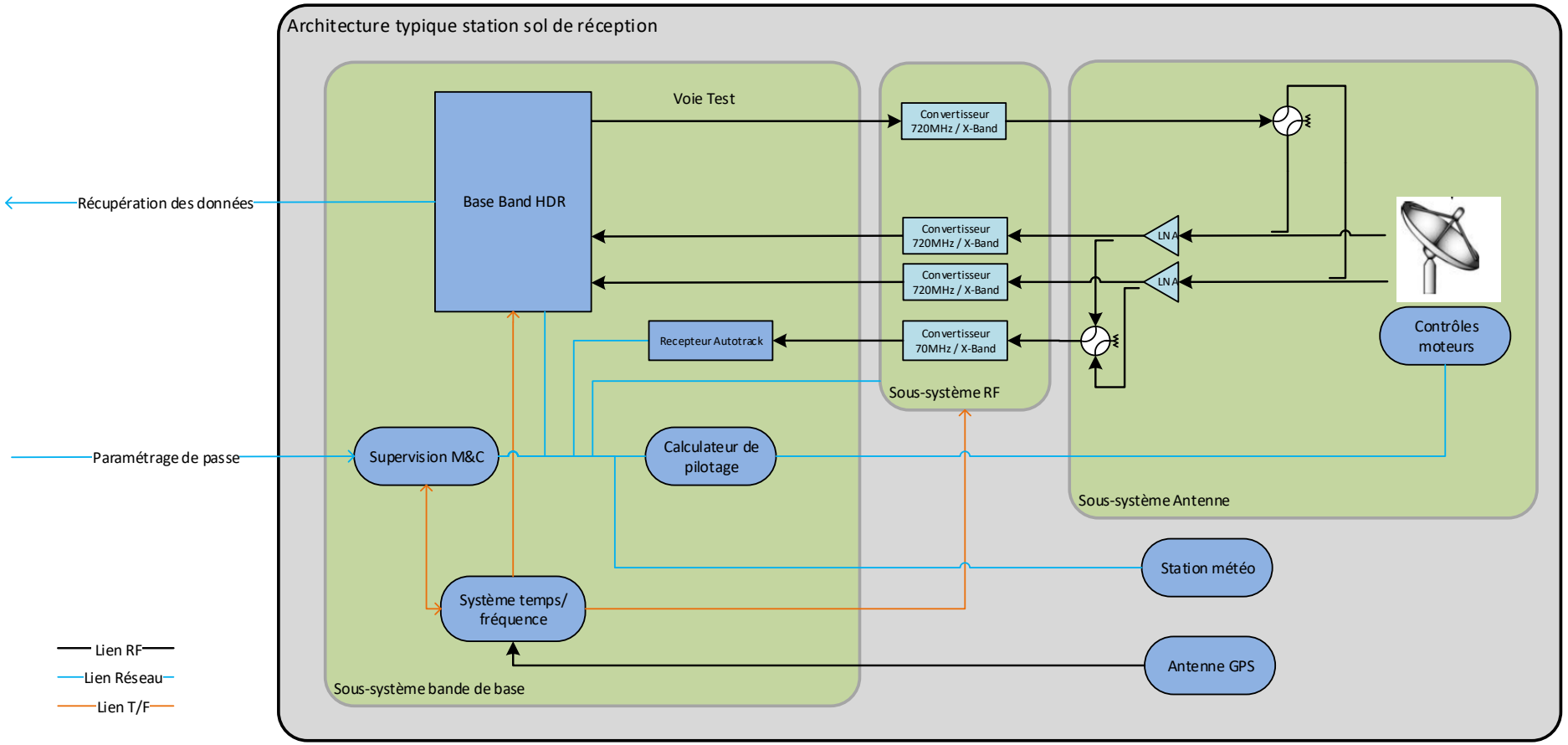


Figure 5.2: Synoptic Diagram of RF subsystem architecture. Copy of Figure 4.6 - a larger view. Courtesy of Airbus Defence and Space.

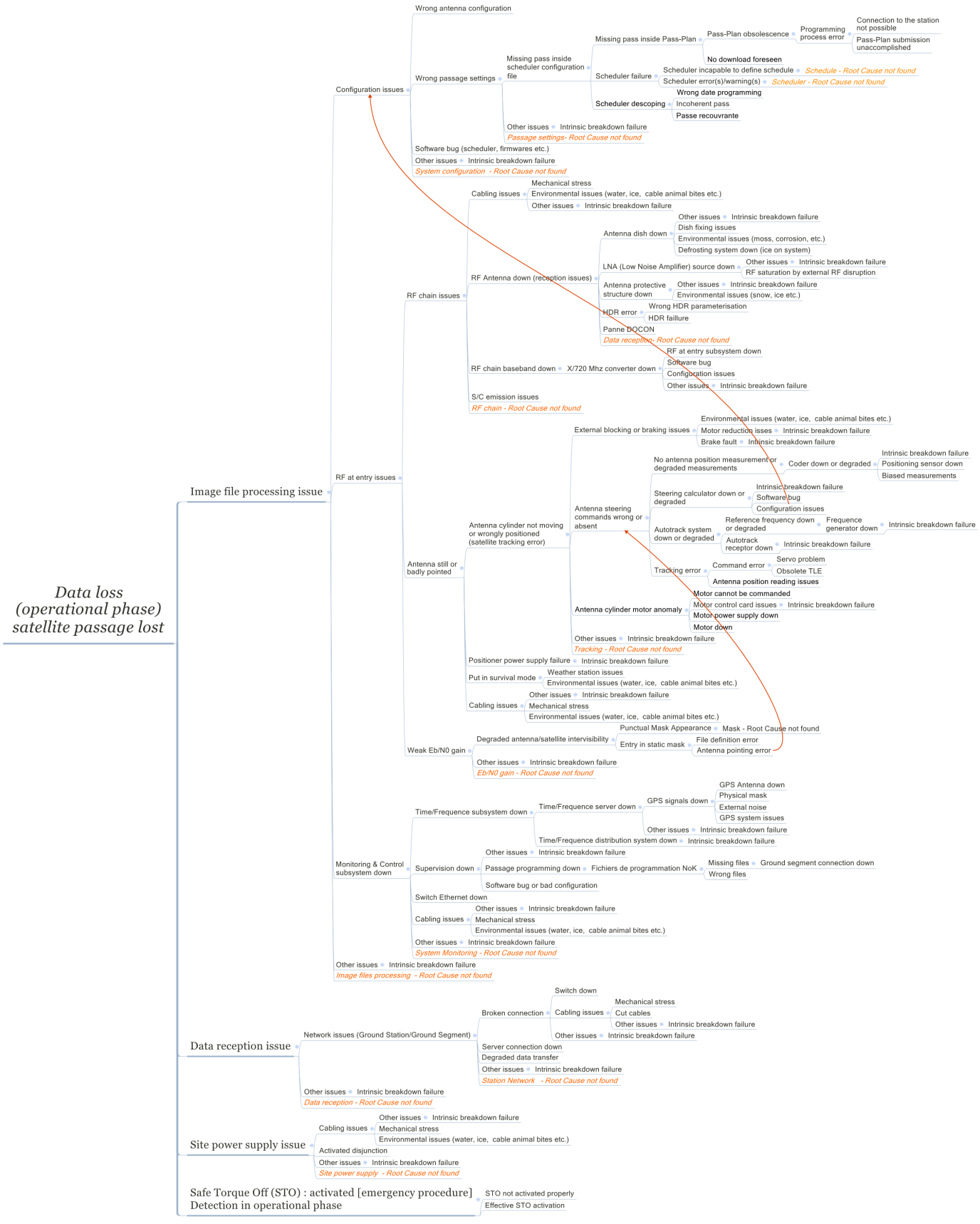


Figure 5.3: Mind Map illustrating fault propagation sequences leading to the top feared events for the GS use case. Translated from French. Courtesy of Airbus Defence and Space.

Annex C - Ground Station SysML Diagrams

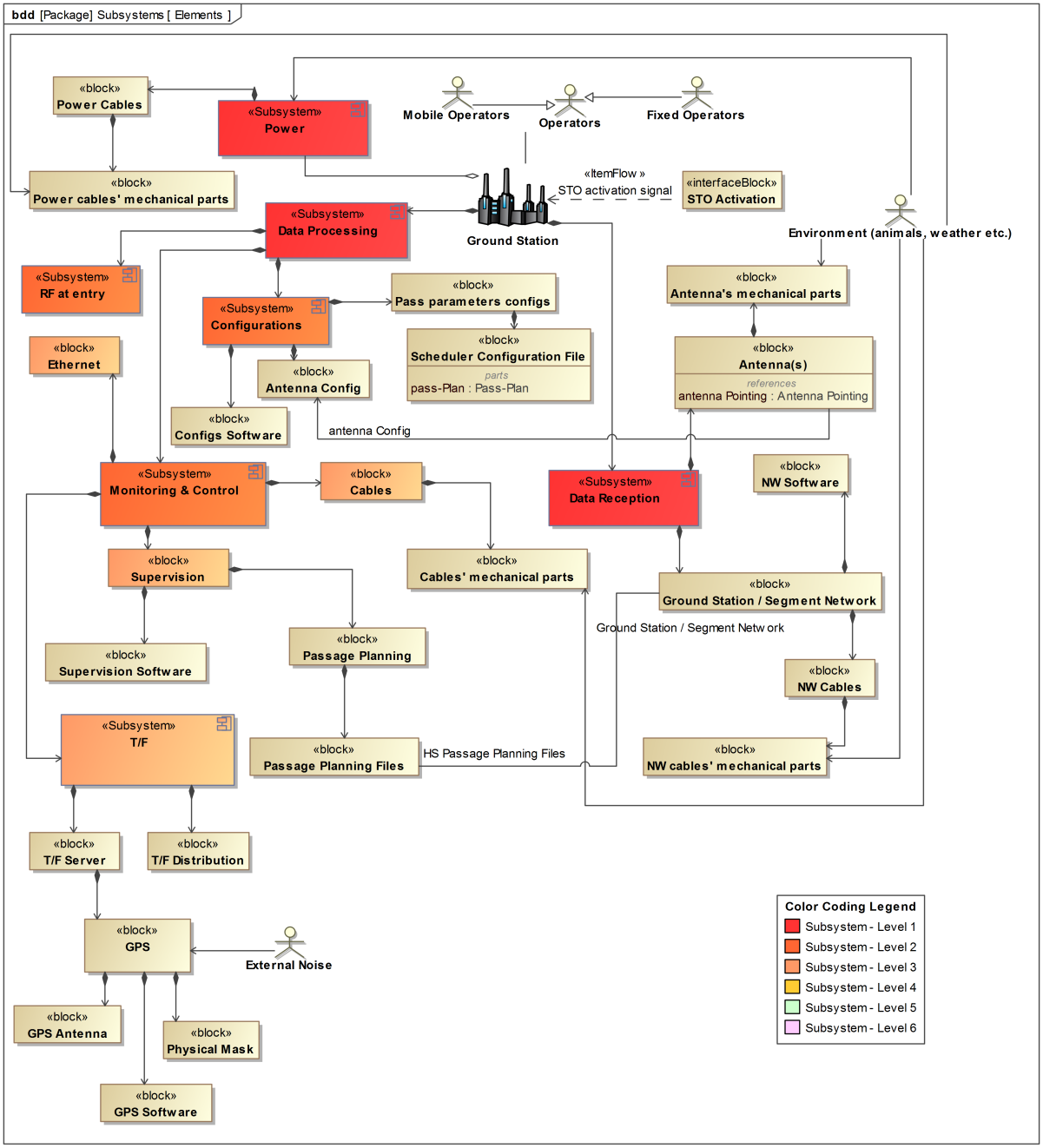


Figure 5.4: Cameo BDD representing the GS's structural elements.

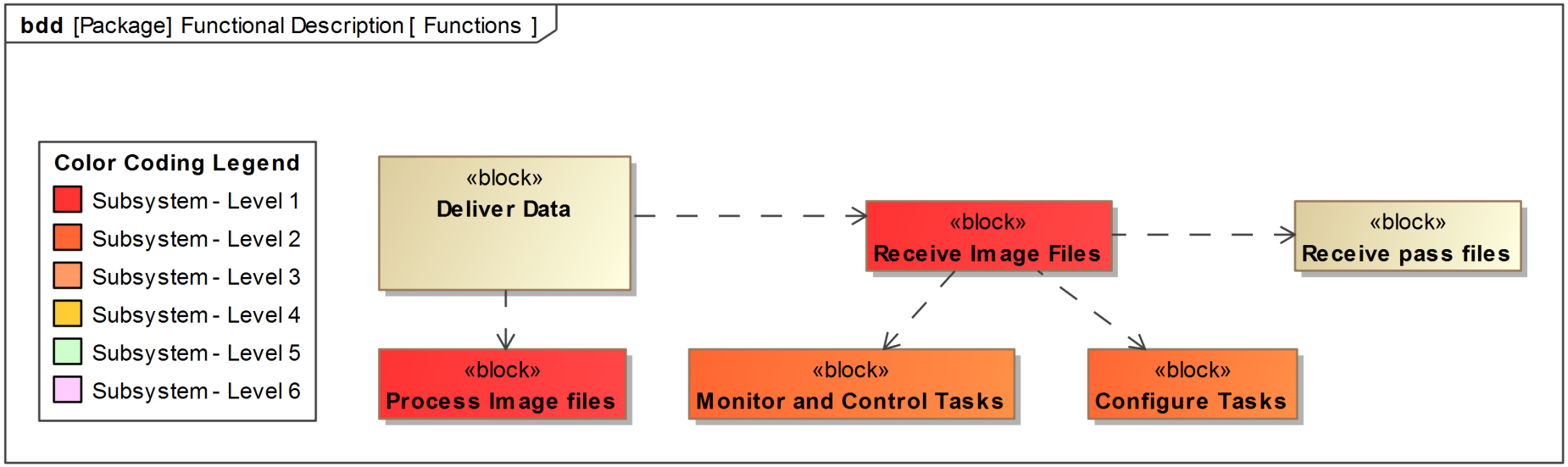


Figure 5.5: Cameo BDD demonstrating the GS's basic functions.

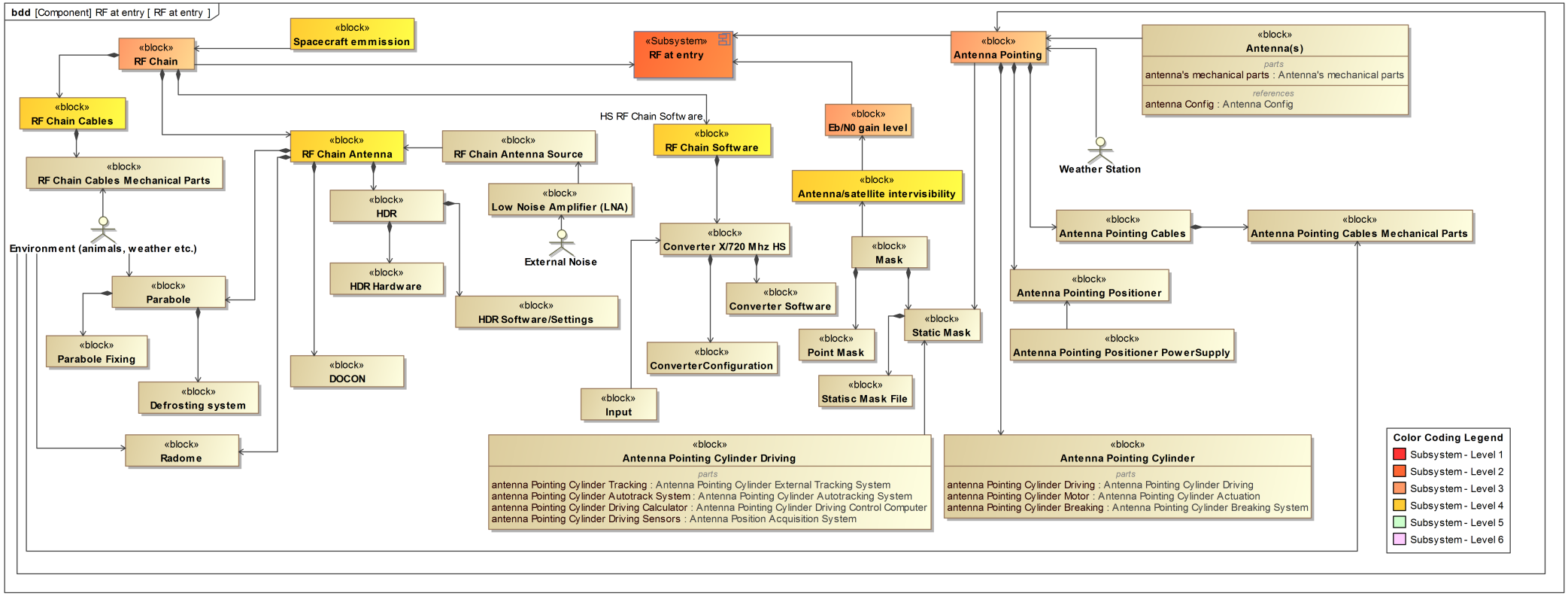


Figure 5.6: Cameo BDD representing the architecture of the GS's RF subsystem.

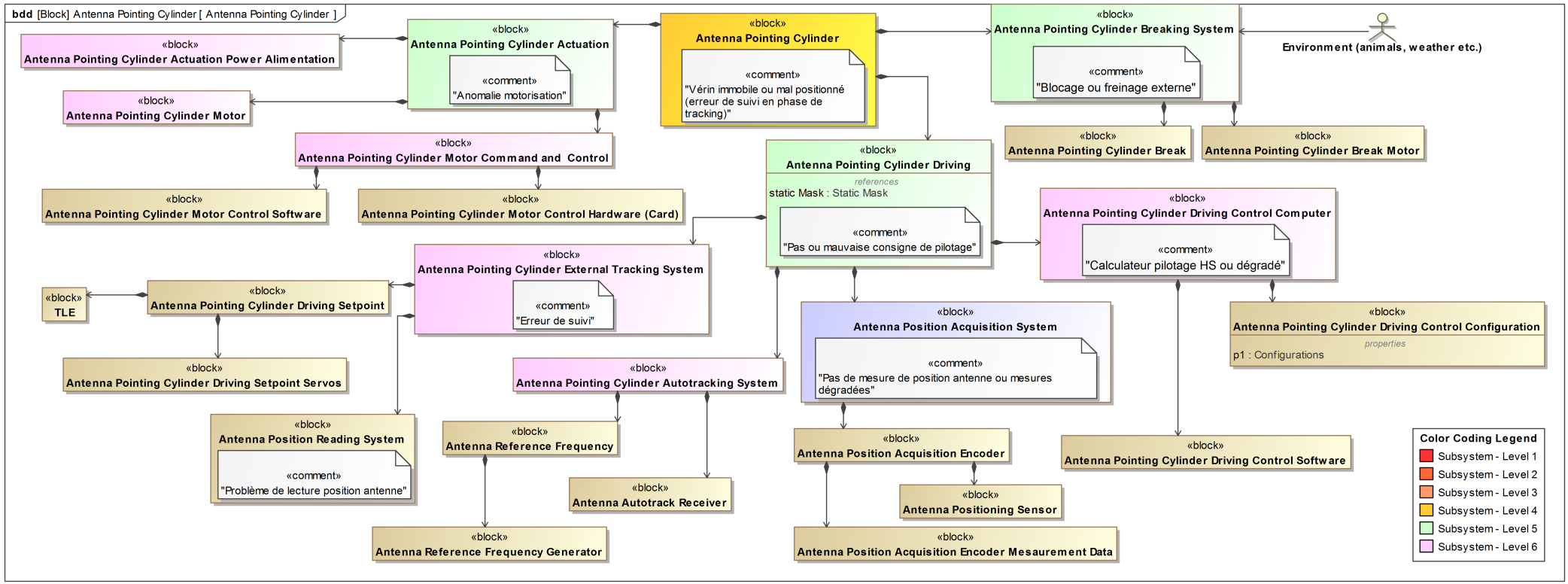


Figure 5.7: Cameo BDD representing the architecture of the GS's Antenna pointing cylinder.

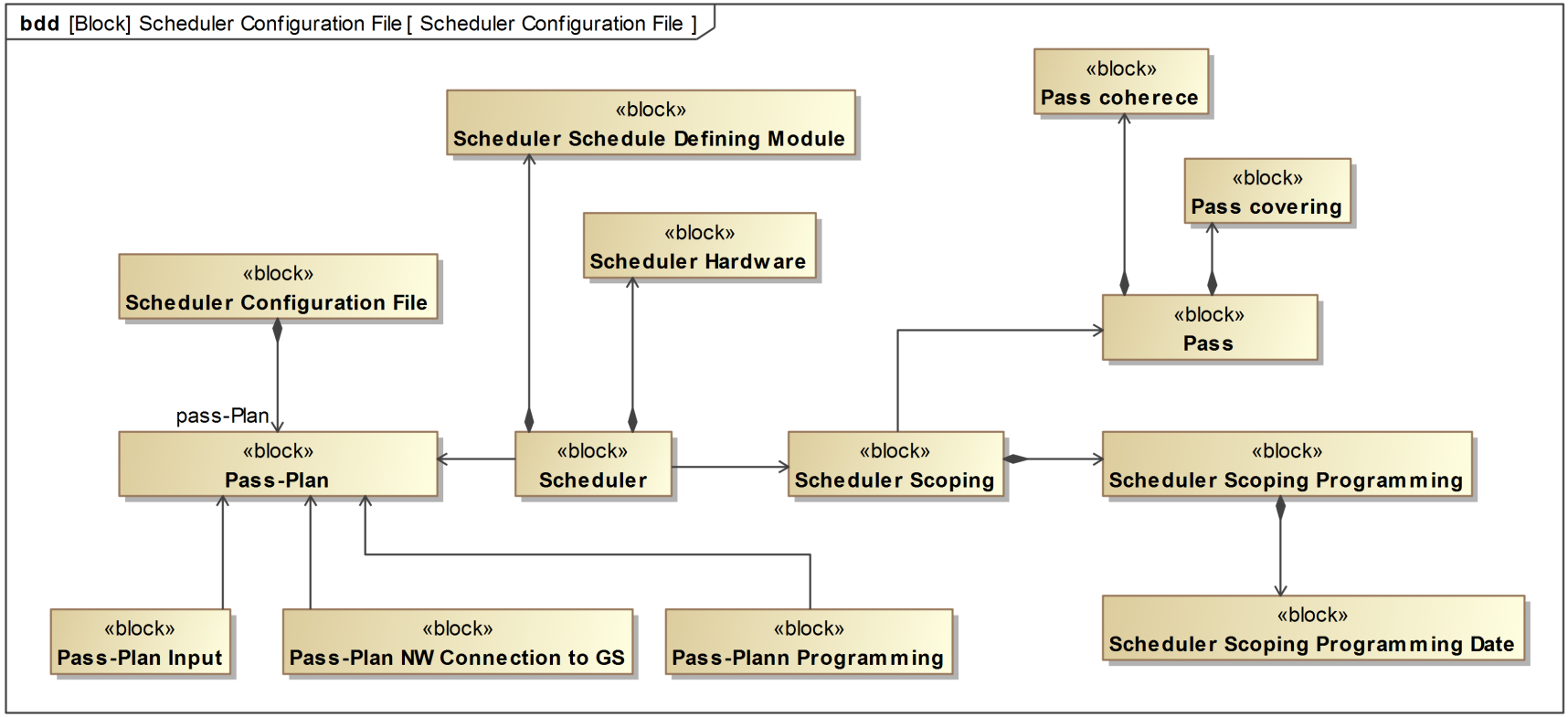


Figure 5.8: Cameo BDD representing the file interactions of the GS's Configuration Scheduler.

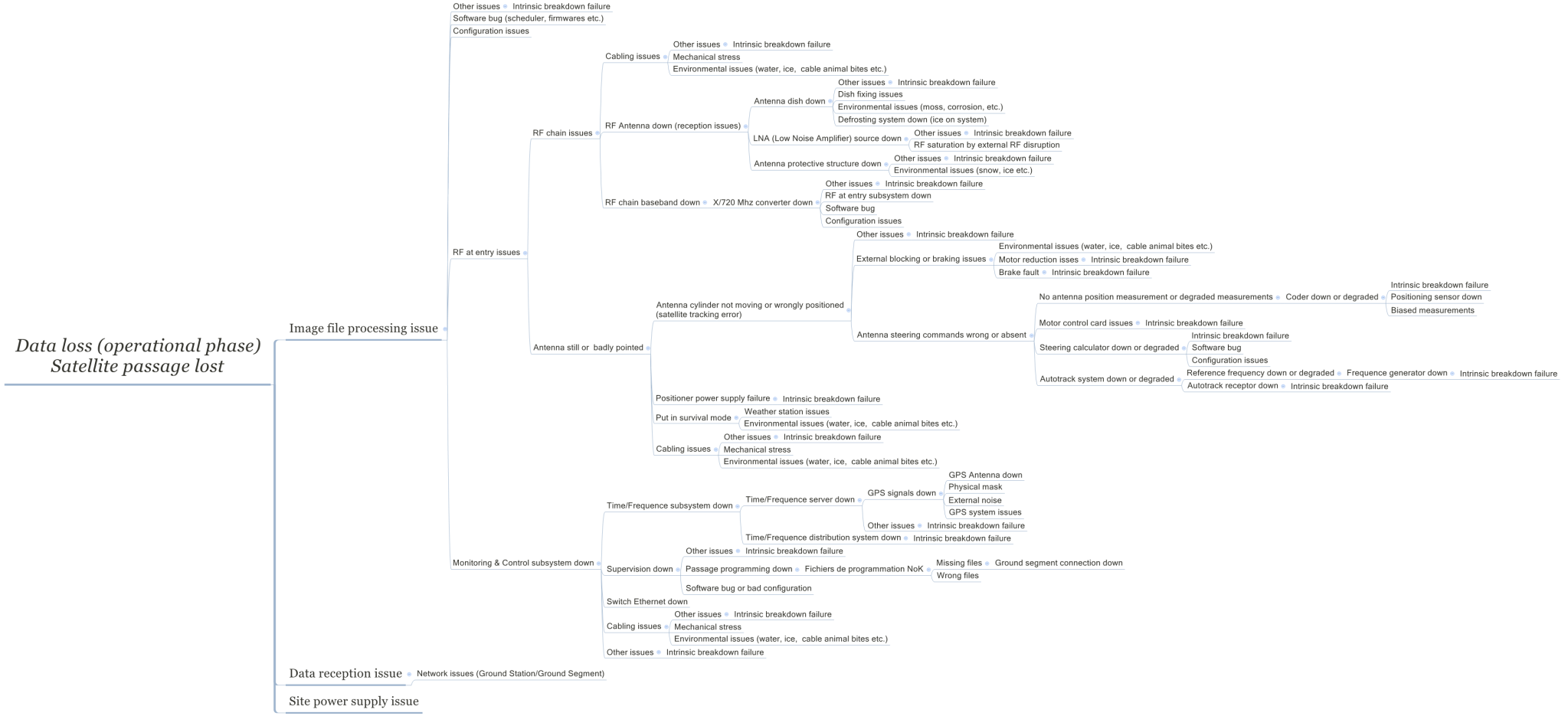


Figure 5.9: Extracted Mind Map with experimental information regarding data loss of Satellite TM data in a GS system. Focus on the RF subsystem. Copy of Figure 4.7 - a larger view.

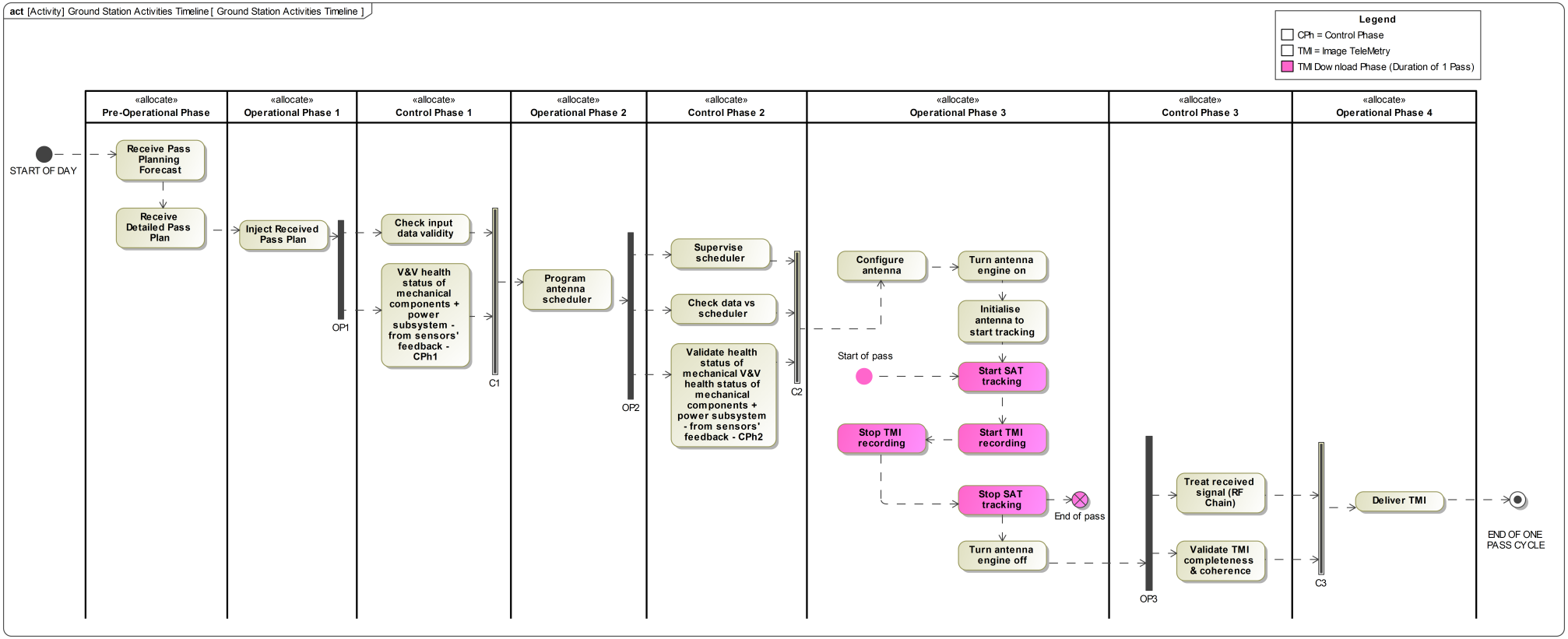


Figure 5.10: SysML Activity Diagram representing the operational scenario (sequence of operational activities) of an image telemetry data receiving Ground Station in *one* satellite pass (single satellite pass cycle). This is the operational scenario we tackle in our case study. Copy of Figure 4.8 - a larger view.

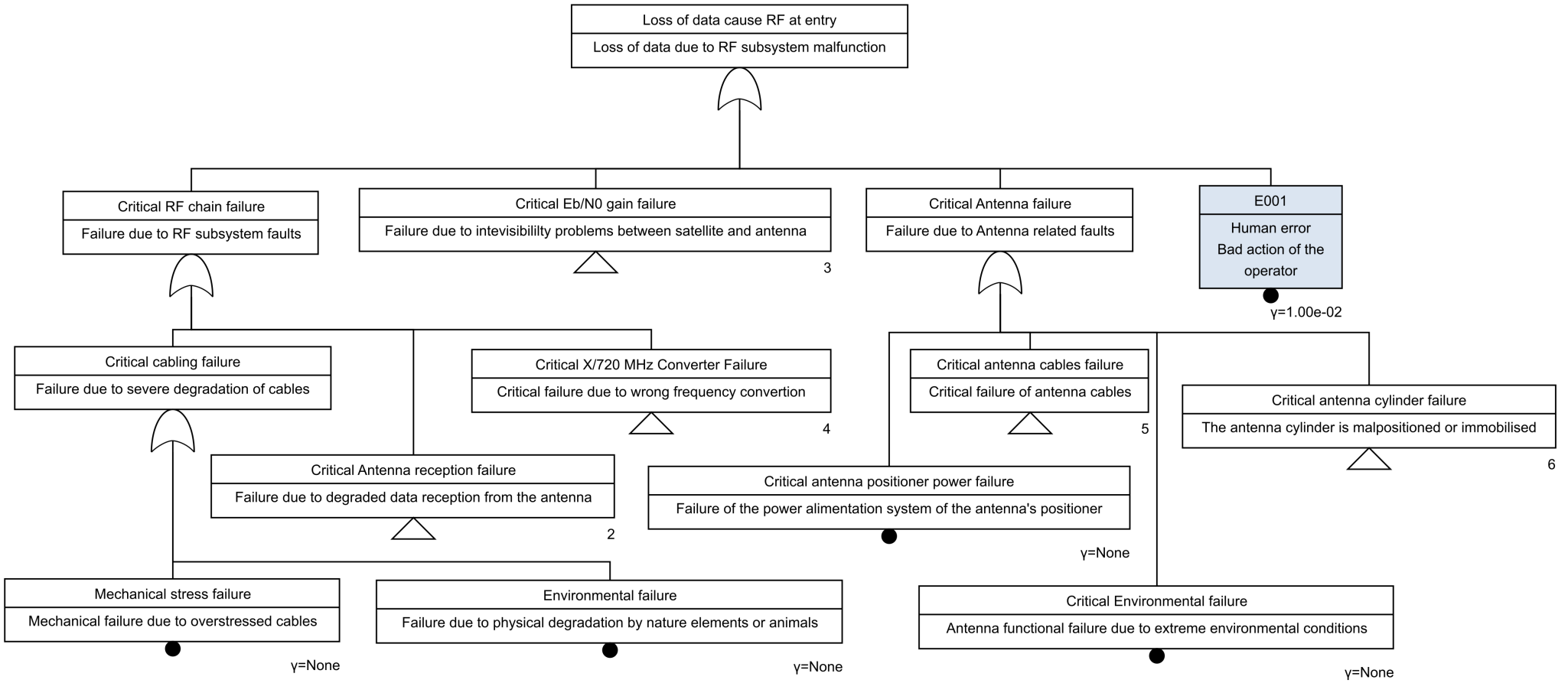


Figure 5.11: Fault Tree (FT) for a Ground Station (GS) system related to Satellite Telemetry (TM) data loss. Focus on the RF subsystem. Feared Event: Loss of TM data due to wrong RF at the GS TM receptor. Copy of Figure 4.9 - a larger view.