A Novel Methodology to Construct Digital Twin Models for Spacecraft Operations Using Fault and Behaviour Trees

Nikolena Christofi
IRT Saint Exupéry, LAAS-CNRS, INSA Toulouse, Airbus Defence & Space, Federal University of Toulouse
Toulouse, France
nikolena.christofi@irt-saintexupery.com

Xavier Pucel
DTIS-ONERA, ANITI,
Federal University of Toulouse
Toulouse, France
xavier.pucel@onera.fr

ABSTRACT

Successful satellite data reception requires the nominal operation of the ground stations in charge of their health monitoring as much as the spacecrafts themselves. Although the concept of Model-Based Diagnosis (MBD) in the field of autonomous systems –such as satellites, has long been researched and developed, that is not the case for their ground systems. Both satellites and ground stations operate autonomously. The latter however, are not equipped with the advanced Fault Detection, Isolation and Recovery (FDIR) capabilities one finds today on-board all orbiting spacecrafts. The aim of the study presented in this paper is the improvement of ground stations' operational diagnostics by providing the operators with ad-hock, Operations-Dedicated Models (ODMs). The latter serve as a basis for the construction of the system's Digital Twin (DT) models. These models allow the operators react more quickly and more precisely to alarms raised by the station. By helping the operators identify the malfunction and correct it in the quickest delays, they can avoid loosing the next satellite telemetry (TM) data, thus saving precious time and costs. This would increase both the availability and maintainability of the system. In a larger framework, ODMs are ideally concurrently built and connected with the engineering and safety models of the system, in a sort of virtual continuous improvement loop. While the utter purpose of ODMs is their usage as the system's DTs during operations, they also contribute to the stations' architecture and robustness continuous improvement, through increasing its fault detection and mitigation capabilities.

CCS CONCEPTS

• **Software and its engineering → System modeling languages**; **Design languages**; **Architecture description languages**; **Fault tree analysis**; **Operational analysis**.

KEYWORDS

MBSE, MBSA, fault trees (FTs), behaviour trees (BTs), operational diagnosis, model-based diagnosis (MBD), model-based operations (MBO), digital twins (DTs), ground systems operations.

MODELS '22 Companion, October 23–28, 2022, Montreal, QC, Canada
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9467-3/22/10…$15.00
https://doi.org/10.1145/3550356.3561550

ACM Reference Format:
Nikolena Christofi and Xavier Pucel. 2022. A Novel Methodology to Construct Digital Twin Models for Spacecraft Operations Using Fault and Behaviour Trees. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3550356.3561550

1 INTRODUCTION

In the era of digital communications, models are becoming prevalent for the development of complex systems, especially when time and safety are critical. Nevertheless, the use of models is not extended to the field of operations. When operating a satellite system, operators need to detect and isolate –or even mitigate, any occurred fault in a *very limited time-frame*. However, satellite operators have no exhaustive knowledge of the system architecture, nor its underlying design. They are only expected to monitor the health state of the system and detect any deviations (called *symptoms*) of it components in the so-called housekeeping data.

Operators' troubleshooting tasks vary depending 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 foresee its occurrence, operators are expected to *diagnose* the system, in order to eliminate the fault, and restore functions to their nominal state (or the least harmful possible). Thus, if the anomaly is not included inside the symptoms' database (no troubleshooting procedures associated), the operators must take individual action.

These actions are usually based on the operators' knowledge and experience. If this knowledge were to be combined with knowledge of the system itself (architectural and behavioural, as well as functional and dysfunctional), operational diagnosis could be improved significantly. For this reason the authors propose the creation of an Operations-Dedicated Model (ODM) [5], representing a Digital Twin of the Operations and Maintenance system tasks, for Fault Detection and Diagnosis (FDD).

With the ODM's integration inside a monitoring tool, the operators will be provided a dedicated User Interface (UI) to help them identify anomalies more efficiently (in less time, and with less effort). This will allow them to restore the system more quickly –than currently, where no operations-model is available to them. The authors propose a methodology for the creation of ODMs using the formalism of Behaviour Trees (BTs), taking as input system design information, found inside Fault Trees (FTs). The system health-monitoring data received at each satellite passage and lessons learnt from operating and maintaining the system i.e. Return of Experience (RoE), are used to feed the Digital Twin model.

FTs are produced from system Reliability, Availability and Safety (RAMS) analyses, or Model-Based Safety Assessment (MBSA) models. They thus incorporate dysfunctional system information, but also, indirectly, functional information, since they use as input Systems Engineering (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. We use a Satellite Ground Station (GS) architecture to demonstrate our proposal.

The structure of the paper is as follows. Related work on current operational diagnosis approaches and BTs is provided in Section 2. Section 3 contemplates the usage of the ODM as the system's DT. Section 4 describes the proposed method; basic terms are formulated and methodological steps are provided in detail. The method is illustrated with a use case example. Finally, Section 5 discusses the proposed method's limitations and Section 6 draws conclusions.

2 LITERATURE REVIEW

Recently the practise of integrating experimental design in system modelling is being explored. Virtual design in combination with operational data consist the future of the SE practise [15]. Originally introduced as a means to break information silos and reduce verification and 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 [3] and Mars 2020 [12] missions. The enabling of MBSE in combination with the development of Industry 4.0 inspired the development of the 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 [8].

A Digital Twin (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 [17]. 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 [19]. It is used throughout a system's life cycle for evaluating keymetrics and compliances [16]. In the past few years, the use of DTs in a variety of industrial operations is being explored exhaustively. Nevertheless, and as stated [17], "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), FDD has made significant advancements in the last decades [4] [13]. The most popular solutions consist of Finite State Automata (FSA) [21] [18] [20] or Petri Nets (PNs) [2] [11] 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.

More specifically, in petri-nets or automata-based methods (PNs 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 [9]. For this and other reasons –explained below, the authors have investigated other possible solutions, resulting to the choice of Behaviour Trees (BTs).

BTs belong to the DES family. They were first invented as a tool to build modular Artificial Intelligence (AI) in computer games. A known alternative to Finite State Machines (FSMs), BTs are meant to provide better modularity, scalability, extendibility, adaptability and reuse of code [6], and to be easier to understand for humans, thus allowing incremental functionality design and efficient testing.

The choice of BTs for the ODM construction was inspired by the fact that BTs can easily provide a detailed view of the system's current state –in different decomposition levels. This is an indispensable information for the system's operators. More specifically, BTs represent (and implement) a way to control the execution of a set of concurrent processes. Each concurrent process is represented by a leaf node (action; behaviour/function to be executed). Parent nodes (Sequence, Fallback/Selector, Parallel) specify which children nodes must be visited, and in which order. BTs define how the behaviour evolves when one (or all) of its children behaviour succeeds or fails.

More precisely, 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 $children : B \rightarrow B^*$ 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.

In the next section we further justify our choice of BTs as the language semantics for building the ODM. Moreover, we explain why BTs also meet the DTs requirements; where $DT = ODM_{executed} + (RoE)$. More information is provided below.

3 ODM EXPLOITATION AS DIGITAL TWIN

Figure 1 portrays the interactions between the system architects and safety experts, the ODM design team and the operators, with the ODM and the ODM DT. The left side boxes represent pre-system-deployment interactions. The right side boxes illustrate post-system-deployment activities.

As illustrated in Figure 1, ODMs serve a double purpose. On the one hand, since they are concurrently created with the SE & SA/RAMS models, they can provide feedback to the designers, specific to the system's health monitoring and FDIR aspects. This allows the improvement of the system's structural and behavioural design.

On the other hand, following the system deployment, the ODM can immediately be used as a prototype for the system's DT. That is because BTs are executable. The ODM can facilitate the design of test scenarios, distribute diagnosis objectives across the various activities, and ensure that operators are given realistic tasks. The DT model is then periodically updated with any new information supplied by the received system operational data.

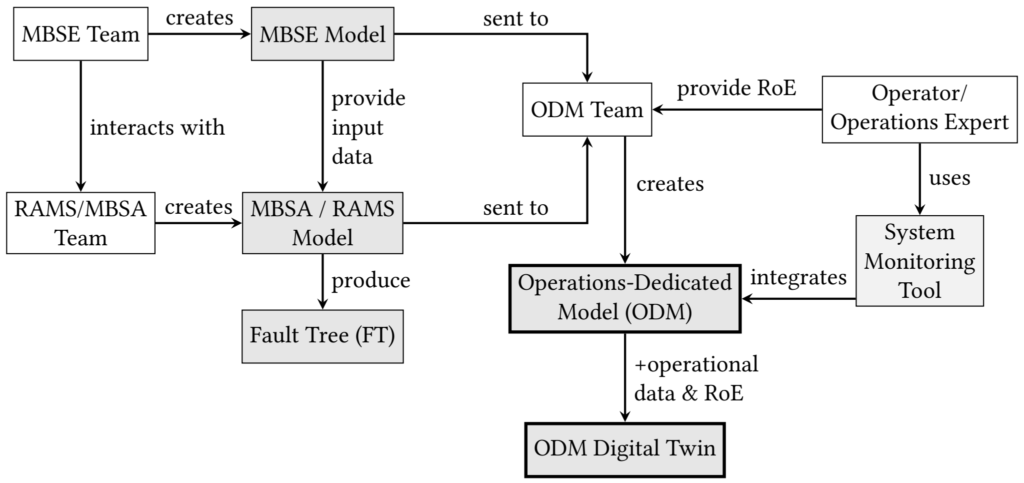

Figure 1: Overview of the proposed methodology for the construction and usage of an operations-dedicated model, used as the base to construct a digital twin monitoring model of the system, with the integration of operational data feedback © N. Christofi.

Eventually, the DT model will be used by operators for FDD purposes. For this an additional intuitive User Interface (UI) is needed, for the operators to be able to execute and exploit the models. This is possible by integrating the DT inside monitoring tools equipped with UI capabilities. BT libraries already provide similar tools, such as Groot for the BehaviourTree.CPP library [10]. What is more, operational feedback can eventually contribute to the amelioration of the original system, by initiating modifications to the existing architecture while providing compelling information for future systems' design.

Regarding our BT implementation to construct the ODMs, we use the *PyTrees* (python implementation of BTs) library [7]. We found PyTrees relatively easy and intuitive to code with –in comparison to the C-based library alternative we tested, namely BehaviourTree.CPP [10]. This way any engineer/scientist can build a BT model, with no particular programming background or developer skills. In the subsection below follows a more elaborated description of BTs, including a definition of its basic constituting elements.

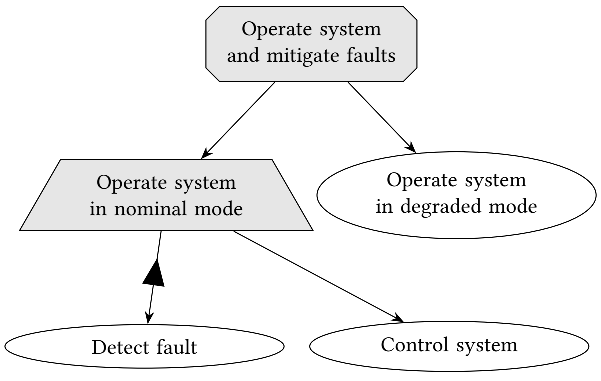
Operate system and mitigate faults
Operate system in nominal mode
Operate system in degraded mode
Detect fault
Control system

Figure 2: The BT for system operation with fault detection and mitigation; children are ordered from left to right © N. Christofi.

To establish ODM's operational diagnosis capabilities, its language semantics must enable the modelling of FDD related elements. That is, the system's fault detection, automatic diagnosis and recovery features, as well as its fault mitigation mechanisms. Moreover, these semantics must allow associating health indicators to system elements. To that end, we extended the PyTrees library with three entirely new types of nodes, so as to enhance ODMs' system monitoring features. Regarding health indicators, we have have so far enriched the status of each BT behaviour (other than the standard "Success", "Failure" and "Running") with associated *alarms*. A visual tool demonstrating the feasibility of our approach has also been implemented.

In the following section we present basic BT definitions, as well as our contribution as regards to the extension of the BT library to meet the ODM needs. For that we provide a detailed description of each newly defined node. We then outline the associated developed methodology for the ODM creation from system dysfunctional models.

4 METHODOLOGICAL DESCRIPTION FOR THE ODM CREATION

Throughout system development, ODMs must gather all system monitoring relevant information, for performing automatic diagnosis and hinting to a single failure candidate. During system operations, ODMs must be easily readable and usable by operators, so as to aid them in their troubleshooting tasks. BT semantics can fulfill both these causes, due to the facility in which one can model complex system behaviours, as well as the intuitiveness of their construction and format. Below we provide basic definitions on BTs as well as the definitions we have introduced within the language semantics, to better suit the requirements for the building and exploitation of ODMs.

4.1 ODM Semantics Description: BTs

Figure 2 illustrates a BT, in which the root behaviour is named "Operate system and mitigate faults". Its children are named "Operate system in nominal mode" and "Operate system in degraded mode". The former has children behaviours, while the latter is a leaf behaviour. Each behaviour *is composed of* its children.

Definition 4.1 (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 $state : \mathcal{B} \rightarrow \mathcal{S}$, 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 also 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 and ParallelAll.

Definition 4.2 (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. In this paper Sequence behaviours are drawn with gray signal shapes ⊳.

Definition 4.3 (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. In this paper Fallback behaviours are drawn with gray octagon shapes ⯃.

Sequence and Fallback behaviours have at most one running child at each instant.

Definition 4.4 (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 —◀—→.

Definition 4.5 (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, ParallelAll behaviour fails and interrupts the rest of the children. In current literature ParallelAll behaviours are simply called *Parallel*. We chose to modify its standard name for distinction reasons with the supplementary parallel behaviour we introduced, as defined shortly below. In this paper, ParallelAll behaviours are represented by gray $\wedge$-shaped trapezia ⏢ .

In addition to the standard composite behaviours presented above, we introduced three new BT behaviours, related to semantic translations from FTs and to fault diagnosis. They are respectively called ParallelAny and Fault Detection and Fault Avoidance behaviours.

Definition 4.6 (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 ⏢.

Definition 4.7 (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 ◇.

Finally, our methodology uses a type of behaviour that can be either atomic or composite, but their semantic is defined with respect to a specific fault event.

Definition 4.8 (Fault avoidance behaviour). A *Fault avoidance* behaviour is a behaviour that should fail when the fault event occurs, and never succeed. It can be atomic or composite. In this paper, all behaviours whose names start with "Avoid" are Fault avoidance behaviours.

Fault avoidance behaviours are always associated with a fault event from a FT. They do not always make sense from an operational point of view, and are mostly an artifact used as a temporary translation between the FT and the ODM. Note that in a composite Fault avoidance behaviour, its children must be compatible with the Fault avoidance specification, otherwise the BT is invalid, and its semantic hence undefined.

In Figure 2, the semantics of the BT are as follows. The topmost 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 behaviour executed is "Operate system in 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, which 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 in degraded mode" behaviour.

4.2 ODM creation

As mentioned earlier, we use FTs as an input source for ODM construction. FTs are the produced result of Fault Tree Analysis (FTA). 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 undesirable event –viz. *Feared Event (FE)*, can be predicted in the context of its environment and operation [14] [1]. 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). The FT segments leading to a FE define all of the things that could go wrong (faults) to cause the FE e.g. loss of telemetry (TM) image data.

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.

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

Definition 4.9 (Fault tree transform FT2BT). The transform of a FT into a BT is implemented by the function *FT2BT* defined on FT nodes as follows.

I. If FTN is a basic event named "Fault event X", then: *FT2BT*(FTN) is an atomic fault avoidance behaviour named "Avoid fault event X".
II. 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*(FT_1), *FT2BT*(FT_2), … .
III. If FTN is an OR gate labelled "Fault event X" with children nodes FT_1, FT_2, …, then *FT2BT*(FTN) is a ParallelAll behaviour named "Avoid fault event X", with children behaviours *FT2BT*(FT_1), *FT2BT*(FT_2), … .

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.

4.3 Example

In this subsection we illustrate our methodological approach proposal with the help of a simplified Ground Station (GS) use case. For the needs of the paper, we only demonstrate a small part of the dysfunctional analysis that took place in order to produce the FT shown in Figure 3. The Radio-frequency (RF) subsystem of the GS system was chosen for the methodology application demonstration.

The case study contemplates the 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 3 represent only critical single faults. That is because the information source for the analysis was solely the Return of Experience (RoE) from GS operators. Hence only OR gates are included in the FT: the represented fault events 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. Only critical faults were of interest for the specific study.

In Figure 3 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.

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 3 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 5. Thus all "Avoid", behaviours become "Operate" behaviours. The Root behaviour is represented by the Fallback node "Operate RF Subsystem (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.

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.

5 DISCUSSION

The first BT version was automatically created from the provided FT. BT-v.1 did not include any supplementary system information, as for example on how to avoid certain faults. In the second BT version we can see that certain elements, which were included neither in the FT nor BT-v.1, are useful for system operation and specifically, fault mitigation.

BT-v.2 shows that by being able to detect a specific behaviour, we can manage to operate the system *even under non-nominal conditions*. For example, by adding additional sensors, redundant components or further surveillance on the Antenna body, the risk for its associated fault occurrence can be greatly reduced. The latter contributes to increasing the GS's Availability and Maintainability features.

Surely, the method implementation on a real-life complex system is necessary, to order to evaluate its scalability and facility

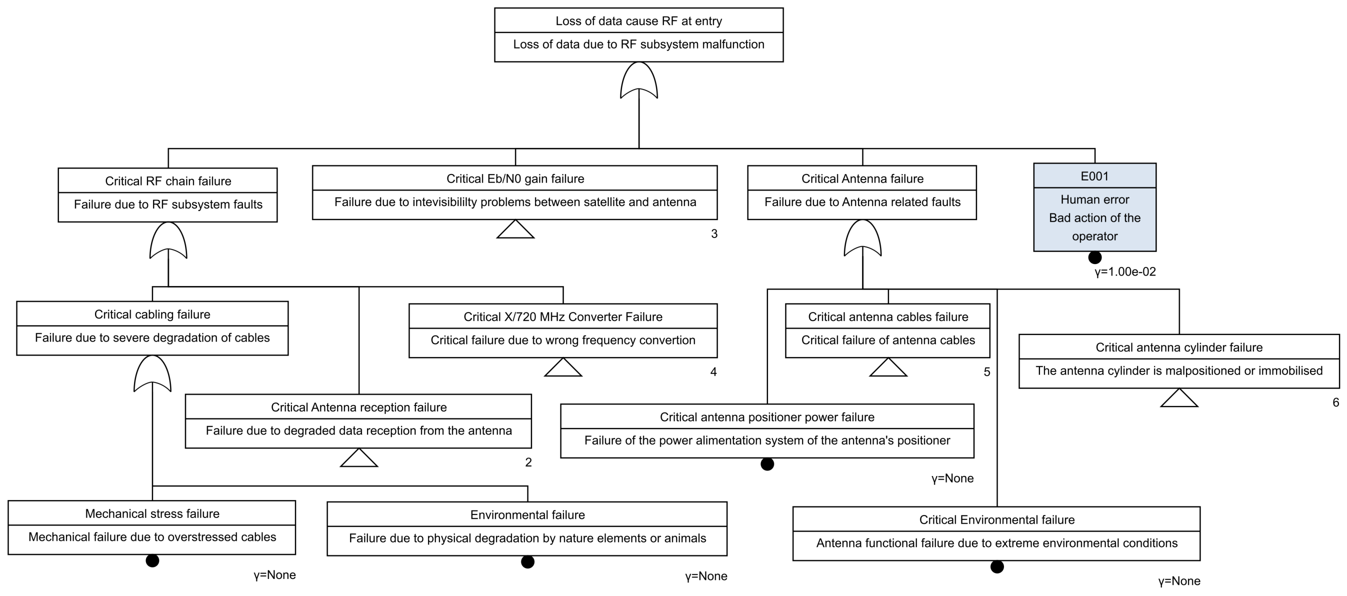
Loss of data cause RF at entry
Loss of data due to RF subsystem malfunction
Critical RF chain failure
Failure due to RF subsystem faults
Critical Eb/N0 gain failure
Failure due to intevisibililty problems between satellite and antenna
3
Critical Antenna failure
Failure due to Antenna related faults
E001
Human error
Bad action of the operator
γ=1.00e-02
Critical cabling failure
Failure due to severe degradation of cables
Critical X/720 MHz Converter Failure
Critical failure due to wrong frequency convertion
4
Critical antenna cables failure
Critical failure of antenna cables
5
Critical antenna cylinder failure
The antenna cylinder is malpositioned or immobilised
6
Critical Antenna reception failure
Failure due to degraded data reception from the antenna
2
Critical antenna positioner power failure
Failure of the power alimentation system of the antenna's positioner
γ=None
Mechanical stress failure
Mechanical failure due to overstressed cables
γ=None
Environmental failure
Failure due to physical degradation by nature elements or animals
γ=None
Critical Environmental failure
Antenna functional failure due to extreme environmental conditions
γ=None

Figure 3: Extracted Fault Tree for a Ground Station (GS) System regarding data loss of satellite telemetry (TM) data. Focus on the RF Subsystem. Feared Event: Loss of TM data due to wrong RF at the GS TM receptor.

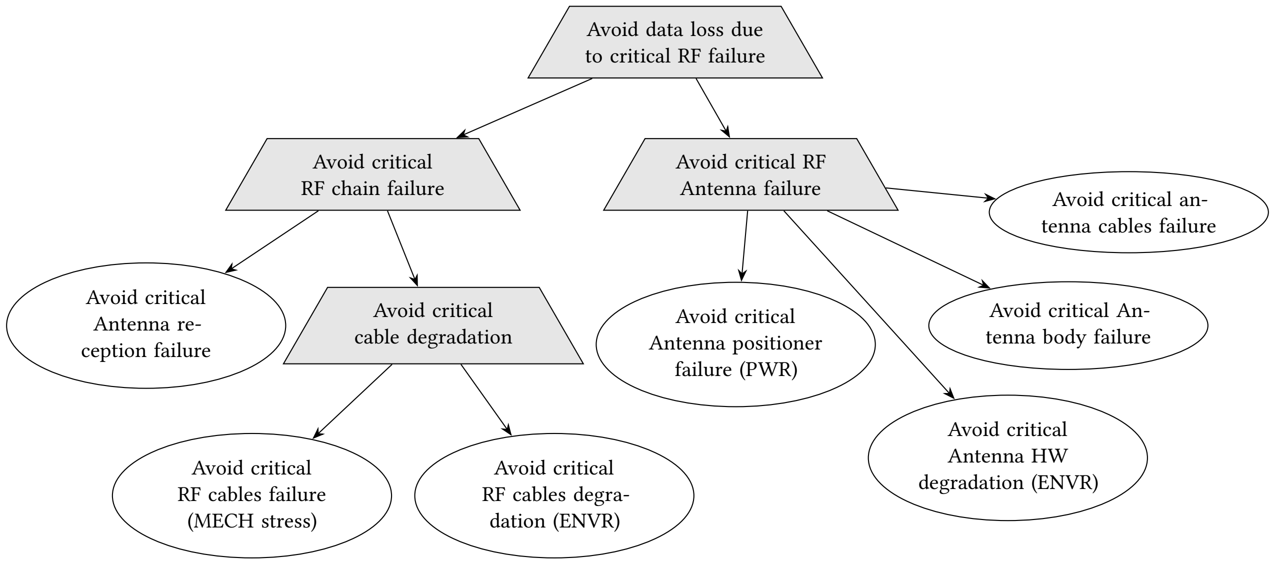
Avoid data loss due to critical RF failure
Avoid critical RF chain failure
Avoid critical RF Antenna failure
Avoid critical antenna cables failure
Avoid critical Antenna reception failure
Avoid critical cable degradation
Avoid critical Antenna positioner failure (PWR)
Avoid critical Antenna body failure
Avoid critical Antenna HW degradation (ENVR)
Avoid critical RF cables failure (MECH stress)
Avoid critical RF cables degradation (ENVR)

Figure 4: First version of the BT, derived automatically from the FT depicted in Figure 3.

of application. In addition to that, it is imperative that operators provide feedback on the methodology and the intuitiveness of ODM exploitation.

6 CONCLUSION

In this paper the authors propose a novel methodological approach on how to construct Operations-Dedicated Models from Fault Trees, using the Behaviour Tree formalism. The method is illustrated with a Ground Station use case example. The limitations of the proposal are discussed, as well as future work opportunities. Finally, the authors contemplate the methodology validation by planned interactions with system operators and modellers, and extended dissemination to DevOps experts, through conference presentations.

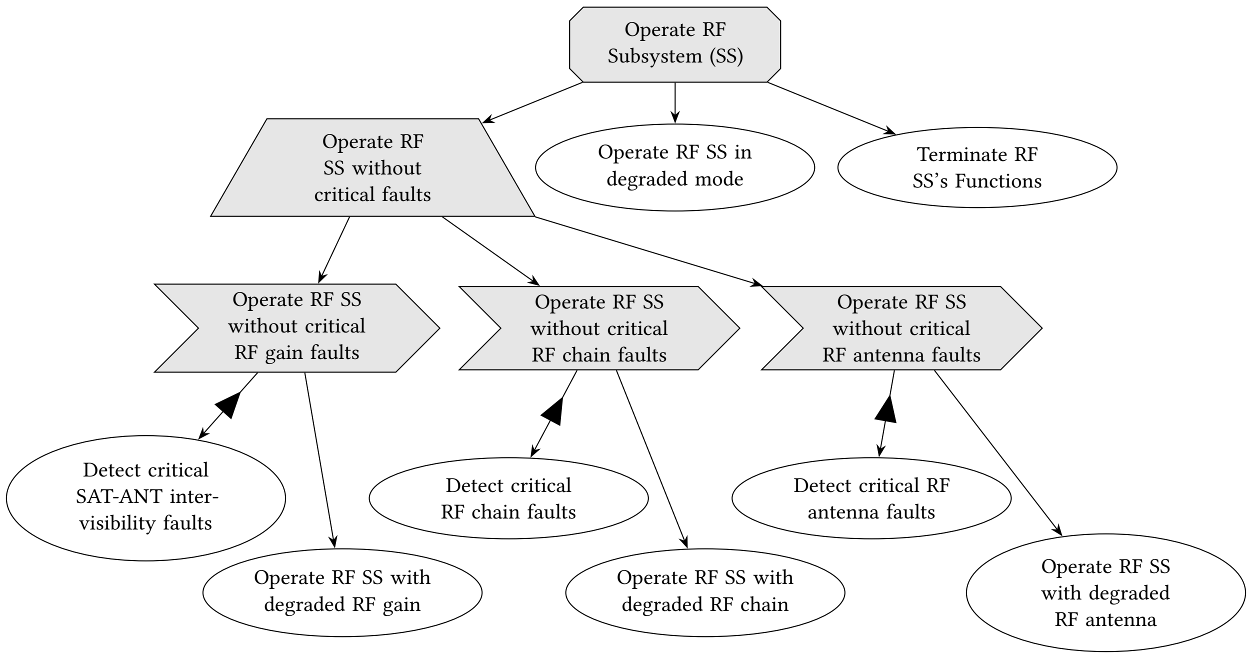
Operate RF Subsystem (SS)
Operate RF SS without critical faults
Operate RF SS in degraded mode
Terminate RF SS's Functions
Operate RF SS without critical RF gain faults
Operate RF SS without critical RF chain faults
Operate RF SS without critical RF antenna faults
Detect critical SAT-ANT inter-visibility faults
Operate RF SS with degraded RF gain
Detect critical RF chain faults
Operate RF SS with degraded RF chain
Detect critical RF antenna faults
Operate RF SS with degraded RF antenna

Figure 5: Second version of the BT, derived from eliciting the first BT version (Figure 4); follows format of Figure 2.

ACKNOWLEDGMENTS

The authors would like to thank all individuals, companies and research institutes involved in the S2C project led by the IRT Saint Exupéry and in particular Airbus Defence & Space for proposing and funding the research topic. The work presented in this paper results from the PhD thesis undertaken by N. Christofi and is equally supported by the French National Research Agency (ANR). Special thanks to Claude Baron, Marc Pantel, Sebastien Guilmeau and Christophe Ducamp for co-supervising the thesis work, as well as to Jerome Golenzer and Paul Albessard for their contribution on Ground Segment Operations. Most importantly, we thank David Canu, for providing us the Ground Station Use Case.

REFERENCES

[1] Yong Bai and Qiang Bai. 2019. Subsea Risk and Reliability. In *Subsea Engineering Handbook (Second Edition)*, Yong Bai and Qiang Bai (Eds.). Gulf Professional Publishing, Boston, Chapter 10, 239–261. https://doi.org/10.1016/B978-0-12-812622-6.00010-5

[2] Francesco Basile. 2014. Overview of fault diagnosis methods based on Petri net models. In *2014 European Control Conference (ECC)*. 2636–2642. https://doi.org/10.1109/ECC.2014.6862631

[3] Todd J. Bayer, Seung Chung, Bjorn Cole, Brian Cooke, Frank Dekens, Chris Delp, I. Gontijo, Kari Lewis, Mehrdad Moshir, Robert Rasmussen, and Dave Wagner. 2012. Model Based Systems Engineering on the Europa mission concept study. In *2012 IEEE Aerospace Conference*. 1–18. https://doi.org/10.1109/AERO.2012.6187337

[4] Maria Paola Cabasino, Alessandro Giua, Laura Marcias, and Carla Seatzu. 2012. A comparison among tools for the diagnosability of discrete event systems. In *2012 IEEE International Conference on Automation Science and Engineering (CASE)*. 218–223. https://doi.org/10.1109/CoASE.2012.6386425

[5] Nikolena Christofi, Xavier Pucel, Claude Baron, Marc Pantel, Sébastien Guilmeau, and Christophe Ducamp. 2022. Towards an agile, model-based multidisciplinary process to improve operational diagnosis in complex systems. In *11th European Congress on Embedded real time systems (ERTS 2022)*. https://www.erts2022.org/, Toulouse, France. https://hal.laas.fr/hal-03699979

[6] Michele Colledanchise and Petter Ögren. 2018. Behavior Trees in Robotics and AI: An Introduction. *arXiv:1709.00084 [cs]* (June 2018). https://doi.org/10.1201/9780429489105 arXiv: 1709.00084.

[7] Pytrees Development Community. 2022. *Pytrees Library Documentation*. Retrieved July 11, 2022 from https://py-trees.readthedocs.io/en/devel/

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

[9] Thushara Ekanayake, Devapriya Dewasurendra, Sunil Abeyratne, Lin Ma, and Prasad Yarlagadda. 2019. Model-based fault diagnosis and prognosis of dynamic systems: a review. *Procedia Manufacturing* 30 (2019), 435–442. https://doi.org/10.1016/j.promfg.2019.02.060 Digital Manufacturing Transforming Industry Towards Sustainable Growth.

[10] Davide Faconti Eurecat. 2022. *BehaviorTree.CPP*. Retrieved July 11, 2022 from https://www.behaviortree.dev/

[11] Maria Pia Fanti and Carla Seatzu. 2008. Fault diagnosis and identification of discrete event systems using Petri nets. In *2008 9th International Workshop on Discrete Event Systems*. 432–435. https://doi.org/10.1109/WODES.2008.4605985

[12] Elyse Fosse, Ann Devereaux, Corey Harmon, and Mallory Lefland. 2015. Inheriting Curiosity: Leveraging MBSE to Build Mars2020. In *AIAA SPACE 2015 Conference and Exposition*. https://doi.org/10.2514/6.2015-4617

[13] S. Hashtrudi Zad, R.H. Kwong, and W.M. Wonham. 1998. Fault diagnosis in discrete-event systems: framework and model reduction. In *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171)*, Vol. 4. 3769–3774 vol.4. https://doi.org/10.1109/CDC.1998.761808

[14] Duane Kritzinger. 2017. Fault Tree Analysis. In *Aircraft System Safety*, Duane Kritzinger (Ed.). Woodhead Publishing, Chapter 4, 59–99. https://doi.org/10.1016/B978-0-08-100889-8.00004-0

[15] Azad M. Madni, Marcus Nance, Michael Richey, William Hubbard, and Leroy Hanneman. 2014. Toward an Experiential Design Language: Augmenting Model-based Systems Engineering with Technical Storytelling in Virtual Worlds. *Procedia Computer Science* 28 (2014), 848–856. https://doi.org/10.1016/j.procs.2014.03.101 2014 Conference on Systems Engineering Research.

[16] Azad M. Madni and Michael Sievers. 2018. Model-based systems engineering: Motivation, current status, and research opportunities. *Systems Engineering* 21, 3 (2018), 172–190. https://doi.org/10.1002/sys.21438 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/sys.21438

[17] Tsega Y. Melesse, Valentina Di Pasquale, and Stefano Riemma. 2021. Digital Twin models in industrial operations: State-of-the-art and future research directions. *IET Collaborative Intelligent Manufacturing* 3, 1 (2021), 37–47. https://doi.org/10.1049/cim2.12010 arXiv:https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/cim2.12010

[18] K.B. Ramkumar, P. Philips, H.A. Presig, W.K. Ho, and K.W. Lim. 1998. 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)*, Vol. 3. 1667–1672 vol.3. https://doi.org/10.1109/IECON.

1998.722923

[19] Michael Schluse, Linus Atorf, and Juergen Rossmann. 2017. Experimentable digital twins for model-based systems engineering and simulation-based development. In *2017 Annual IEEE International Systems Conference (SysCon)*. 1–8. https://doi.org/10.1109/SYSCON.2017.7934796

[20] Ferdinand Settele, Alexander Weber, and Alexander Knoll. 2020. Plant Model-Based Fault Detection during Aircraft Takeoff Using Non-Deterministic Finite-State Automata. *Aerospace* 7, 8 (2020). https://doi.org/10.3390/aerospace7080109

[21] Stavros Tripakis. 2002. Fault Diagnosis for Timed Automata. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Werner Damm and Ernst Rüdiger Olderog (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 205–221.