



From Safety Assessment Models to Operational Diagnosis Models

Nikolena Christofi, Xavier Pucel

► To cite this version:

Nikolena Christofi, Xavier Pucel. From Safety Assessment Models to Operational Diagnosis Models. 33rd International Workshop on Principle of Diagnosis – DX 2022, LAAS-CNRS-ANITI, Sep 2022, Toulouse, France. hal-03773708

HAL Id: hal-03773708

<https://hal.science/hal-03773708>

Submitted on 9 Sep 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

From Safety Assessment Models to Operational Diagnosis Models

Nikolena Christofi^{1,4} and Xavier Pucel^{2,3,4}

¹IRT Saint Exupéry, LAAS-CNRS, INSA Toulouse

²ONERA, DTIS, Toulouse, France

³Artificial and Natural Intelligence Toulouse Institute (ANITI)

⁴Université Fédérale de Toulouse, France

e-mail: nikolena.christofi@irt-saintexupery.com, xavier.pucel@onera.fr

Abstract

The operation of industrial systems can involve complex procedures that must be applied within narrow time constraints –in particular when dealing with faults. The use of formal models can help the design of these procedures and their validation, thus to assist operators. We propose to introduce a new type of Operations Dedicated Model (ODM) in the system design process, and suggest a formalism from the Behaviour Tree family. We assume that Safety Analysis (SA) models describe dysfunctional aspects of the system, notably via Fault Tree Analysis (FTA), and propose a methodology for creating the ODM from a Fault Tree, so as to account for all possible fault events considered by FTA. We demonstrate this methodology on an Unmanned Aerial Vehicle (UAV) example, and discuss how this model improves the system's operations.

1 Introduction

When operating complex systems with low tolerance for faults, such as satellites or Unmanned Aerial Vehicles (UAVs), Fault Detection and Diagnosis (FDD) activities must take place within a short time-frame. It is therefore imperative that the information used by the operators to perform diagnosis and troubleshooting activities is well organized. Additionally, the tools used for system monitoring should take advantage of any knowledge concerning the entire system's structure and behaviour.

System monitoring tools can be divided into two categories. On the one hand, some tools completely *automate* diagnosis for some parts of the system, such as a component, or a function. These tools can be implemented using a wide range of diagnosis techniques [1–4]. On the other hand, some tools aim at assisting operators in their troubleshooting tasks, by aggregating health indicators, or providing quick access to relevant documentation. Both types of tools, even those built on so-called data-based approaches, can only be effective when used for specific tasks, and when the proper input information is available –that being model(s) or data. In both cases, specifying these diagnostic tools require dedicated views of the system. These views shall incorporate the appropriate system information with the defined level of detail, while keeping monitoring and diagnosis as the main focus. An overly detailed, or unstructured, view of the system would make the operation

model too heavy and long to exploit; if the view is too coarse or stays only at high level, the model might lack crucial information for diagnosis.

Within the model-based trend for system development, the authors question the current techniques used in diagnosis during operations. We believe that the use of models shall become imperative for system monitoring, allowing, on the one hand, to *improve operational diagnosis*, and on the other, to *align diagnosis activities with formal data continuity practices* put forward by space companies and agencies worldwide [5, 6]; notably Model-Based Systems Engineering (MBSE) and Model-Based Safety Assessment (MBSA). The former contains a system's functional description –how the system is expected to behave under nominal conditions, and the latter dysfunctional –how the system is expected to behave in case of fault occurrence(s). Hence they can both produce significant knowledge to aid operators in their FDD tasks. Within these lines, we propose a model-based approach which uses the information included in SE/MBSE and SA/MBSA artifact instances (i.e. data models, requirements documents, software or system specifications, architecture descriptions, program modules, test cases, quality requirements) [7], to co-design a monitoring model.

In this paper, we address the problem of collecting available information from MBSE and MBSA and/or Reliability, Availability Maintainability and Safety (RAMS) activities –the latter regarding potential failures, and produce an Operations-Dedicated Model (ODM), which gathers the knowledge required for diagnostics during system operation. As shown in Figure 1, we propose that ODMs are constructed concurrently with SE and SA models and/or documents, during the system design phase. Throughout a classic system development cycle, system architects start designing a system (collection of components in a defined architecture), with the purpose that the system's main function is accomplished. In the meantime, and based on each released MBSE model version, the safety analysts perform studies to ensure that the system conforms with safety constraints and regulations. According to our proposal, ODMs are produced throughout the system architecture development cycle, so that, at each design iteration, new information is integrated in the ODMs, while, at the same time, feedback is provided back to the system architects and safety experts. Hence the system's architecture –including fault mitigation techniques, is eventually improved. ODMs shall be created by a dedicated team of operators and/or operations experts.

Essentially, our approach focuses on *fault management*

There have been a few attempts to use SA information as input to Model-Based Diagnosis (MDD); some work deals with safety models [20, 21]. In our experience, the purpose of SA models does not align with supporting the operators, and as a consequence, SA models are not directly usable as a source for diagnosis activities –whether automated or performed by a human operators. In our selected use case, where we demonstrate the application of our proposed method using a UAV FT as an input, we demonstrate SA models’ inadequacy to meet operational diagnosis objectives –in contrast to ODMs.

3 ODM approach and methodology

Our approach consists in building an ODM that describes the system’s operational procedures, with particular emphasis on fault diagnosis activities. In this section, we elaborate on the purpose of this model, its requirements, propose a variant of Behaviour Tree languages to express it, and present a methodology to create it from a Fault Tree.

3.1 Requirements

ODMs shall contain both functional and dysfunctional system information, necessary for operational diagnostics. In specific, the ODM formalism shall be able to:

- combine *structural* (component breakdown and functional description) and *behavioural* (functional implementation) system information;
- combine *nominal* and *non-nominal* (dysfunctional) system information;
- *integrate* information using *different types of data* as input;
- model operational activities;
- represent system monitoring information –account for feared events, faults, and mitigation means such as troubleshooting and repair.

Additional requirements which fall out of the scope of this paper, but are considered for future work include:

- support of functional & dysfunctional system analyses, for the amelioration of system design;
- integration within a diagnostic tool and implementation of a dedicated User Interface (UI).

This ODM shall serve a double purpose. On the one hand, it shall include relevant system monitoring information in order to perform automatic diagnosis and hint to a single failure candidate. On the other, it must be easily readable and usable by operators, in order to aid them in their troubleshooting tasks. We believe that BT semantics can fulfill both these causes; we have therefore chosen to construct ODMs using BTs.

3.2 Behaviour Trees (BTs)

BTs offer a way to model discrete event systems. A BT represents a set of concurrent processes, in which one process (the tree) orchestrates the execution of a set of other processes, which are represented by the tree leaves.

BTs have shown a lot of potential in the last decade [22, 23], mainly with their application to robotics and Artificial Intelligence (AI) [24–26]. Initially developed within the gaming community to replace Finite State Machines (FSM) with more user friendly models [27], their use could

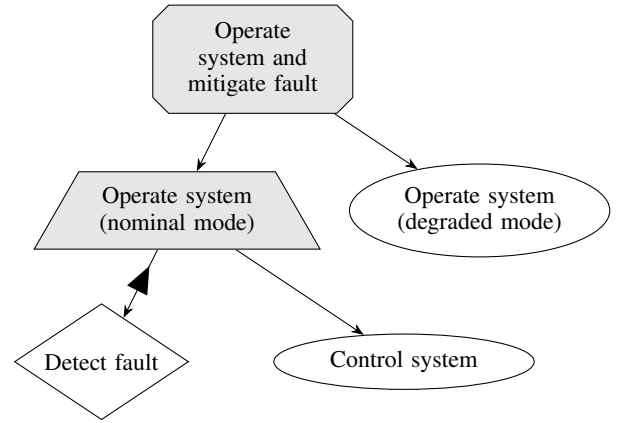


Figure 2: The BT for a system operation with fault detection and mitigation. Children are ordered from left to right.

be widened to the field of operational diagnosis to model, implement and monitor the state of complex systems.

Definition 1 (Behaviour Tree (BT)). A Behaviour Tree (BT) is a tuple $\langle \mathcal{B}, \text{root}, \text{children} \rangle$ where \mathcal{B} is a set of behaviours, $\text{root} \in \mathcal{B}$ is the root behaviour, and $\text{children} : \mathcal{B} \rightarrow \mathcal{B}^*$ 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 2 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.

Definition 2 (Behaviour status). The set of possible statuses for behaviours is the finite set $\mathcal{S} = \{\text{IDLE}, \text{RUNNING}, \text{SUCCESS}, \text{FAILED}\}$.

At each instant in the execution of a BT, the state of the BT is a function $\text{state} : \mathcal{B} \rightarrow \mathcal{S}$, 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, PARALLELALL and PARALLELANY.


Definition 3 (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 \triangleright .


Definition 4 (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 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 this paper, ParallelAll behaviours are represented by gray \wedge -shaped trapezia .

Definition 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 7 (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 .

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

Definition 8 (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 .

In Figure 2, the semantics of the BT 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 behaviour executed 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, 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 (degraded mode)” behaviour.

Finally, our 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 9 (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. In this paper, 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 failure detection mechanism. Thus they do not always make sense from an operational point of view. Fault event avoidance behaviours are merely

an artifact used as a temporary translation between the FT and the ODM. Moreover, 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 this paper, for illustrative purposes, we assume that all fault events can be detected or mitigated, hence included in the ODM as their corresponding operational activities. However, in general FTs may contain fault events that cannot be detected nor mitigated, 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 ODMs from BTs in several steps, since (i) an FT cannot consist an ODM as is, and (ii) new information must be elicited –in combination with FT data, so as to create a coherent and useful ODM.

3.3 Methodology

Our methodology for obtaining a BT –which constitutes our ODM, is composed of two steps:

1. Translate each FT into a BT. 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 the BTs to create a single BT, in which each behaviour represents an actual activity in operations. This step i.e. merging BTs into a single BT manually to then improve it based on one’s knowledge and experience, is performed manually by a person with modelling skills, but more importantly, with operational experience e.g. operator/operations expert.

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

Definition 10 (Fault tree transform $FT2BT$). The transform of a FT into a BT is implemented by the function $FT2BT$ defined on FT nodes as follows:

- If FTN is a basic event named “Fault event X ”, then $FT2BT(FTN)$ is an atomic fault avoidance behaviour named “Avoid fault event X ”.
- If FTN is an AND gate labelled “Fault event X ” with children nodes FTN_1, FTN_2, \dots , then $FT2BT(FTN)$ is a ParallelAny behaviour named “Avoid fault event X ”, with children behaviours $FT2BT(FT_1), FT2BT(FT_2), \dots$.
- If FTN is an OR gate labelled “Fault event X ” with children nodes FT_1, FT_2, \dots , then $FT2BT(FTN)$ is a ParallelAll behaviour named “Avoid fault event X ”, with children behaviours $FT2BT(FT_1), FT2BT(FT_2), \dots$.

The second step of our 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.

3.4 Example

We illustrate our approach with a FT produced for a UAV mission. It addresses the FE of crash landing due to a fault occurred within the Power Supply (PS) subsystem.

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 a ground speed as limited as possible.

There exist various faults that can lead to a crash landing. They are all modelled in this paper as a degradation of the PS, with varying severity. Plenty 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: 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 3. It has 6 basic events:

- *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.

The other nodes in the FT represent how the individual faults can be combined to escalate into the FE.

The first draft of the ODM, shown in Figure 4, is obtained by applying the transformation $FT2BT$ to the FT depicted in Figure 3. One can observe that the result is a straightforward reformulation of the FT into a hierarchical fault management behaviour.

We depict in Figure 5 what the ODM would look like if the operator applied the following elicitation steps to the first BT:

1. Behaviour “Avoid Emergency landing cause PS” involves a mitigation measure (the FTS). So we replace

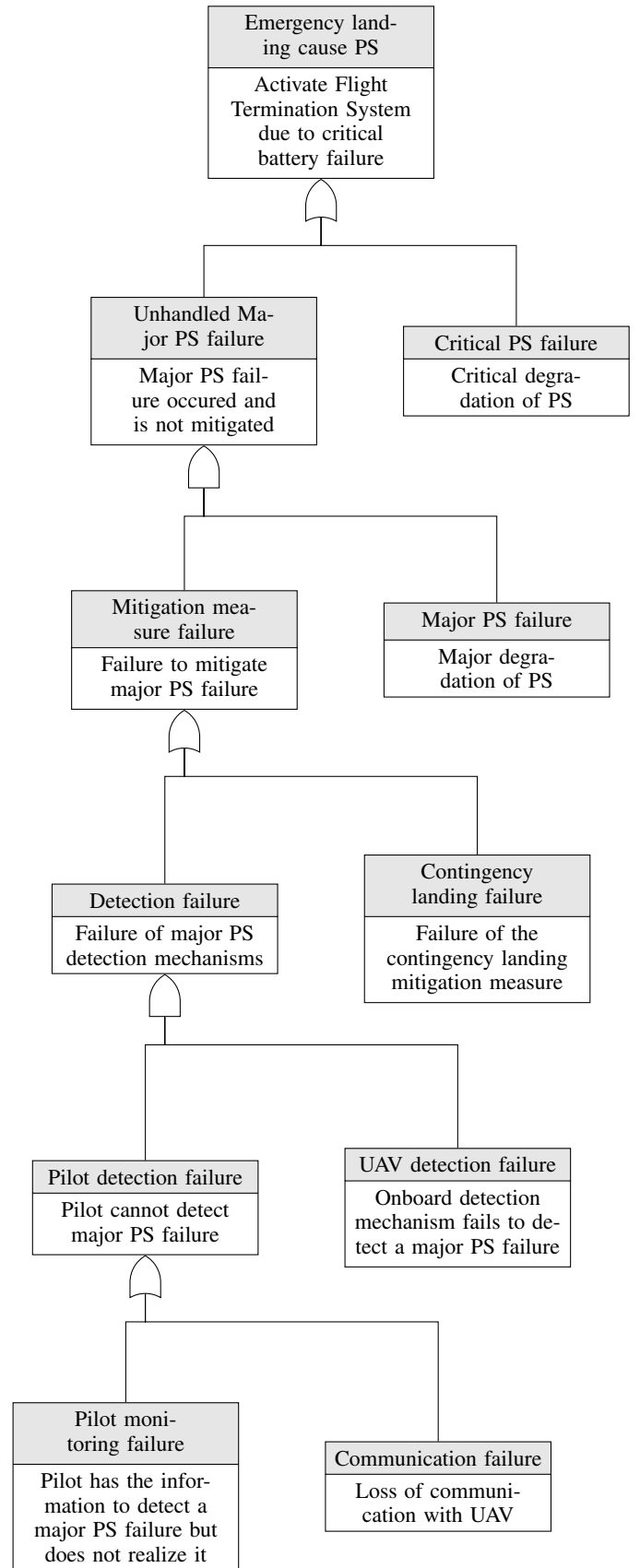


Figure 3: FT leading to the emergency landing of a UAV due to an unmitigated PS failure (feared event).

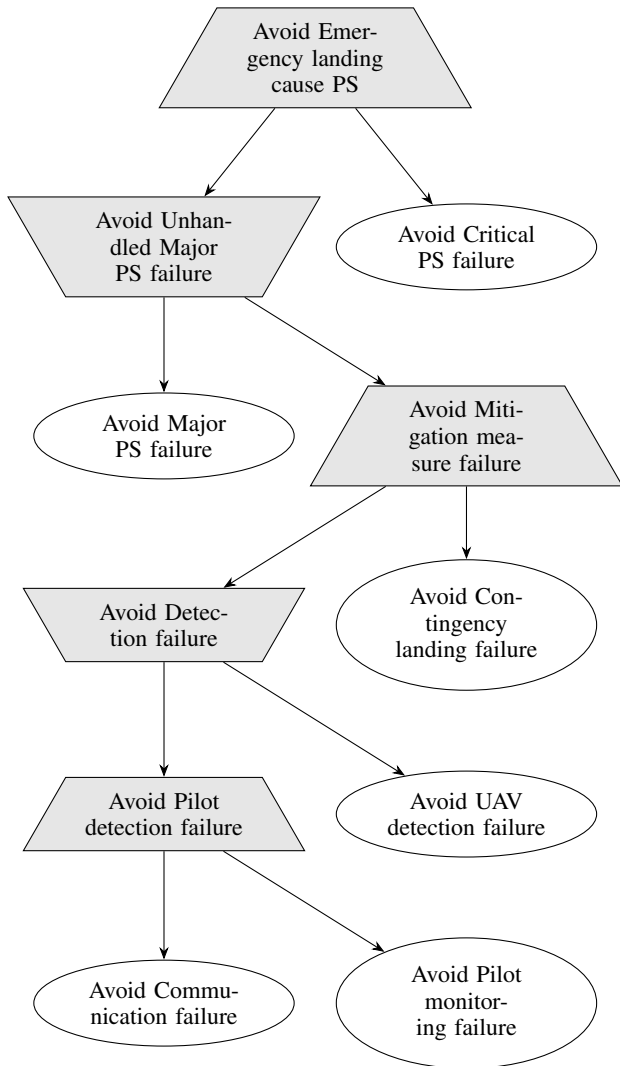


Figure 4: First version of the BT, derived automatically from the FT diagram (Figure 3).

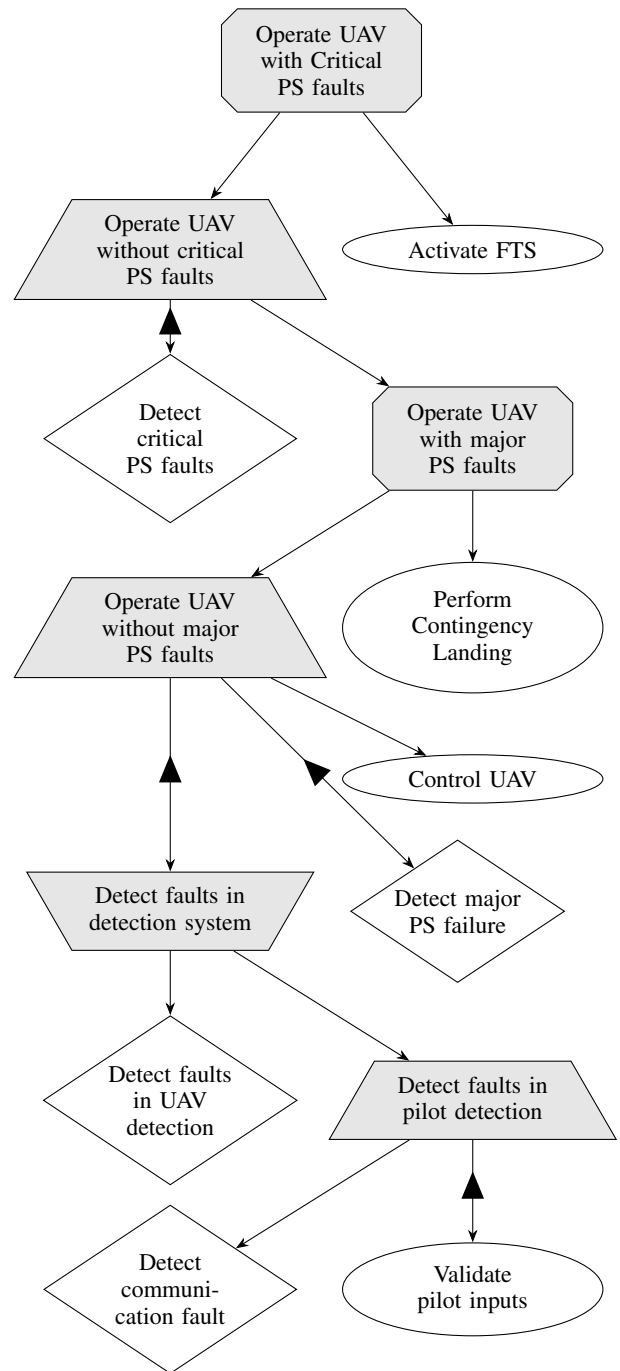


Figure 5: Second version of the BT, derived from eliciting the first version (Figure 4).

it with the fault mitigation pattern depicted in Figure 2. 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 2. 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 if we lose communication, detect that the UAV cannot detect major PS faults anymore, or suspect the pilot is incapacitated, 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.

Note that at step 6, our methodology led us to discover that some cases were not covered by the system specification. 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.

4 Model exploitation

In the sections above we discussed the steps towards the construction of an ODM. The purpose of this model is to account for systems’ operations as early as possible during the system design.

There are several ways to exploit the ODM in order to improve the final system. First, the ODM can be used to feed a visual BT monitoring tool. This is useful for systems such as satellites that can be operated from various ground stations. The status of each behaviour in the BT indicates in which operational mode the system is –and why, and which operations are currently ongoing. Secondly, verification and validation techniques –including model checking, can be applied to the ODM, as well as cross-validation with other MBSE and MBSA models.

5 Conclusion

In this paper we address the need to account for operations during a system’s design. We propose an approach based on an Operations-Dedicated Model (ODM), and suggest Behaviour Trees (BTs) to express this model. We presented formal semantics for the BT nodes, including nodes which are not present in classical BT libraries. We also introduce a novel methodology to create this ODM from a Fault Tree (FT), obtained through standard Safety Assessment (SA) techniques.

To that end, we demonstrate our approach on a UAV use case, focusing on faults in the power supply subsystem. We demonstrate that our methodology is applicable and that it leads to a BT which represents the system’s operation in a way that it accounts for all fault events mentioned in the FT. We noted that the failure of fault detection mechanisms can be more difficult to integrate in operations, than the failure of the functions themselves.

Our belief that the ODM can actually improve the design of an industrial system remains to be demonstrated, for example through interviews with system operators.

6 Acknowledgements

The authors would like to thank all individuals, companies and research institutes involved in the S2C project of IRT Saint Exupéry and in particular, Airbus Defence and Space for proposing and funding this research topic. This work is supported by the French Research Agency (ANR).

The authors also warmly thank Kevin Delmas (ONERA, DTIS) for providing a perfectly sized and realistic fault tree extracted from a UAV infrastructure inspection project.

References

- [1] Zhiwei Gao, Carlo Cecati, and Steven X Ding. A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches. *IEEE transactions on industrial electronics*, 62(6):3757–3767, 2015.
- [2] M-O Cordier, Philippe Dague, François Lévy, Jacky Montmain, Marcel Staroswiecki, and Louise Travé-Massuyès. Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2163–2177, 2004.

- [3] 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.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] Manfred Broy. A logical approach to systems engineering artifacts: semantic relationships and dependencies beyond traceability—from requirements to functional and architectural views. *Software & Systems Modeling*, 17(2):365–393, may 2018.
- [8] Dr Mike Hinchey, Caroline Wang, and Josh McNeil. Formal Methods for System/Software Engineering: NASA & Army Experiences. *Formal Methods*, page 22, 2011.
- [9] C. Seidner and O.H. Roux. Formal Methods for Systems Engineering Behavior Models. *IEEE Transactions on Industrial Informatics*, 4(4):280–291, november 2008.
- [10] John W. Evans, Steven L. Cornford, David Kotsifakis, Tim Crumbley, and Martin S. Feather. Enabling assurance in the mbse environment. In *2020 Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–7, 2020.
- [11] 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.
- [12] Oleg Lisagor. *Failure logic modelling: a pragmatic approach*. PhD thesis, University of York, 2010.
- [13] 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.
- [14] Yiannis Papadopoulos, Martin Walker, David Parker, Erich Rude, 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.
- [15] S-18 Aircraft, Sys Dev, and Safety Assessment Committee. *Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment*, page 331. SAE International, 1996.
- [16] 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.
- [17] Duane Kritzing. Fault tree analysis. In Duane Kritzing, editor, *Aircraft System Safety*, chapter 4, pages 59–99. Woodhead Publishing, 2017.
- [18] 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.
- [19] Frank Wabnitz and Houston Netherland. Use of reliability engineering tools to enhance subsea system reliability. In *Offshore Technology Conference*. OnePetro, 2001.
- [20] Yannick Pencolé, Elodie Chanthery, and Thierry Peynot. Definition of model-based diagnosis problems with altaraica. In *27th International Workshop on Principles of Diagnosis (DX-2016)*, page 8p, 2016.
- [21] Fabien Kuntz, Stéphanie Gaudan, Christian Sanino, Éric Laurent, Alain Griffault, and Gérald Point. Model-based diagnosis for avionics systems using minimal cuts. In *DX 2011*, pages 138–145, 2011.
- [22] 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.
- [23] Michele Colledanchise, Alejandro Marzinotto, Dimos V. Dimarogonas, and Petter Oegren. The Advantages of Using Behavior Trees in Multi-Robot Systems. In *Proceedings of ISR 2016: 47st International Symposium on Robotics*, pages 1–8, June 2016.
- [24] 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.
- [25] 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.
- [26] 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.
- [27] 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.