Towards an agile, model-based multidisciplinary process to improve operational diagnosis in complex systems

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

Nikolena Christofi[1], Xavier Pucel[2], Claude Baron[3], Marc Pantel[4], Sébastien Guilmeau[5], Christophe Ducamp[6]

Toulouse, France

Abstract

Systems' online diagnostics require multidisciplinary system knowledge and experience by their operators. When the complexity of the system rises, operational (in-service) diagnostics become a complex task. In an effort to improve the efficiency while better handling the complexity of diagnostics during operations, the authors propose a methodology aiming to increase the agility in complex systems' development processes. This paper introduces a new way to construct operational models early on the development cycle so as to improve the performance of monitoring activities and, ultimately, increase the confidence on the systems' resilience provided by its design.

Keywords: Satellite Systems, Health Monitoring, Online Diagnosis, Model-Based Systems Engineering (MBSE), Model-Based Safety Assessment (MBSA), Maintainability, Behaviour Trees, Model-Based Operations, Agile Techniques

1. Introduction

To this day, monitoring activities and associated tools used for Fault Detection and Diagnosis (FDD) [1] during system operation are limited to specific functions provided by certain subsystems, while largely relying on the operators' experience [2]. Past data i.e. Return of Operating Experience (ROE), also feed the algorithms used for FDD. However, the ever growing complexity of embedded systems demand that the tools used for system monitoring have an underlying knowledge of the whole system structure and behaviour, so as to provide a faster, and more reliable diagnosis, especially when time is a key operational constraint. What is more, without precise knowledge of the system design, it is ambitious for diagnostic tools to be able to provide a unique possible failure source, or the correct troubleshooting procedure, in case the monitoring data contain alarm inconsistencies.

Diagnostic tools shall thus be able to provide the operators with dedicated views of the system, which will help them perform diagnosis more efficiently. These views must incorporate the appropriate system information with the defined level of detail while keeping monitoring and diagnosis in the main focus. An overly detailed view of the system would make the operation model too heavy and long to exploit, while if the view is too coarse or stays only at high level, the model might lack crucial information to assist in diagnosis. To this end, one approach consists in collecting available information from the system architecture designed during Systems Engineering (SE), as well as from the Safety Analyses (SA) documents and models, the latter regarding the potential failures. The aim would then be to use the collection of data, information and models, to produce a tool that will support the operators in their monitoring task: interpreting housekeeping data, troubleshooting problems, and performing system maintenance.

Email address: `nikolena.christofi@irt-saintexupery.com` (Nikolena Christofi)
[1]IRT Saint Exupéry, LAAS-CNRS, Airbus Defence & Space, INSA Toulouse, Université de Toulouse
[2]ONERA, DTIS, Université de Toulouse
[3]INSA Toulouse, LAAS-CNRS, ISAE-SUPAERO, Quartz-Supméca, Université de Toulouse
[4]IRIT, Toulouse INP, Université de Toulouse
[5]IRT Saint Exupéry
[6]Airbus Defence & Space

Our goal is thus to use the system design information to construct operational diagnosis tools. We postulate that this is possible with the introduction of an Operations-Dedicated Model (ODM), which shall include both system design information –SE data, as well as the information produced by selected system dysfunctional analyses –SA data. The latter shall help in the diagnosis activities by suggesting a possible source for the error, and its location in the system.

We explore Behaviour Trees (BTs) as a solution towards the construction of ODMs to be used for operational diagnosis. Considering that BTs can provide an intuitive way to model and simulate system behaviour, they can also be used to meet the needs for a diagnostic model which includes the system's dynamic aspects and not only its structural information –as it is the case for the currently available diagnosis models. BTs have shown a lot of potential in the last decade, mainly with their application to robotics and Artificial Intelligence (AI). Initially developed within the gaming community to replace Finite State Machines (FSM) with more user friendly models, their use could be widened to the field of operational diagnosis to model, implement and monitor the state of complex systems.

This paper explores the use of BTs to create a system behavioural model –namely ODM, oriented towards system monitoring, which can be exploited during operations in order to increase efficiency in diagnosis. We show in place that the BT model can be built during the design phase and along the production of other design models and documents, in a co-design manner, thus contributing to an agile system development process. The co-design of system and BT models is a process within which documents and models of system function and dysfunction and BT models of system monitoring for operational diagnosis are mutually constituted.

The structure of this paper is as follows. Section 2 outlines the context of our research and the problematic addressed. Section 3 presents a literature review on the involved topics. The proposed approach is presented in Section 4, using an illustrating example in Section 4.1, and perspectives on the use of ODMs inside monitoring tools for diagnostic purposes in Section 4.2. Finally, Section 5 draws conclusions and discusses future work opportunities.

2. Context and Motivation

Despite the recent flourishing of model-based languages, methods and tools for system development –system design and Verification and Validation –V&V activities, diagnostic tools created for the same systems, have known a different path. Existing diagnosis tools are often designed after the system is built; hence the opportunities to make the system easy to operate –which would be the case if the tools were constructed during system design, are lost. Either under the form of data-based [3] or model-based [4], these tools are often targeting single system functions/subsystems –e.g formation flight [5], reaction wheels [6], or thrusters [7, 8, 9]. Consequently, their specificity to one function or subsystem makes them non-generic, and thus not reusable.

Moreover, *diagnostic models fail to include safety analyses methods' derived information*; methods such as Failure Modes and Criticality Effects Analysis (FMECA), Fault Tree Analysis (FTA), Failure Logic Modelling (FLM) [10, 11], Failure Propagation Modelling (FPM) [12], or Model-Based Safety Assessment [13, 12]. On the other hand, system diagnostic tools *can be very generic* i.e. agnostic of the domain usage: they use general diagnosis techniques, such as Machine Learning (ML) [14], *not specific to spacecraft operations*.

Moreover, operational diagnostics, as a core part of satellite maintenance, is limited by many constraints, should thus consist one of the major concerns in satellite design. Therefore, by integrating a diagnostic tool in the system development process and co-designing it along with the system itself, we can ensure that the operational diagnosis activities are well taken under consideration before the satellite deployment.

As mentioned in [15], this would mean that the system design and its monitoring models influence each other. The proposed process is depicted in Figure 1, where it is illustrated with a space satellite system application. Following the system's deployment, the operator, having at their disposal the housekeeping data the Operation Centre (OC) has received from on-board the satellite, can perform diagnosis with the help of the monitoring tool. The latter is using the ODM built during the system development process (concurrently with the system design models) in the form of BTs. This would add confidence in the diagnosis results as well to the operations model. For the needs of our project we assume that the ODM is representative of the actual system emitting the monitoring data.

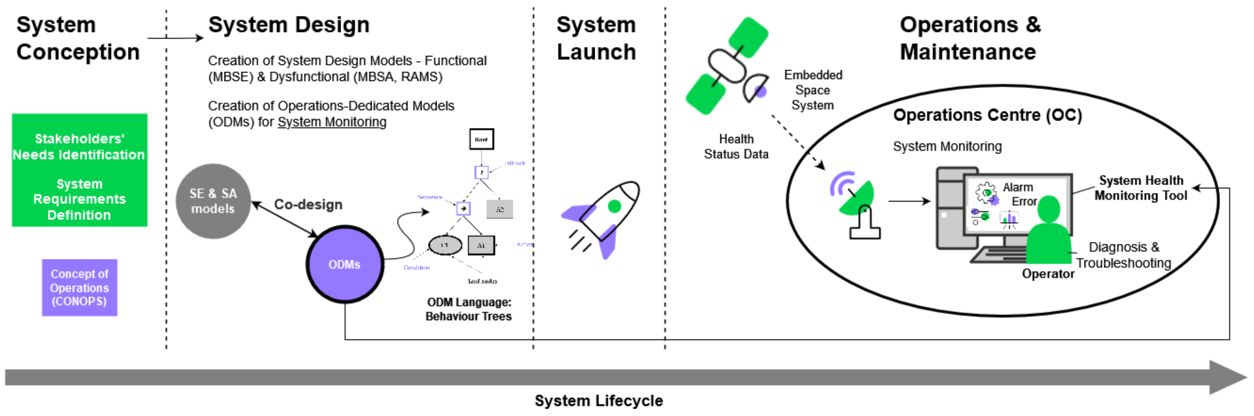

System Conception
Stakeholders' Needs Identification
System Requirements Definition
Concept of Operations (CONOPS)
System Design
Creation of System Design Models - Functional (MBSE) & Dysfunctional (MBSA, RAMS)
Creation of Operations-Dedicated Models (ODMs) for System Monitoring
SE & SA models
Co-design
ODMs
Root
Fallback
Sequence
A2
C1
A1
Action
Condition
Leaf nodes
ODM Language: Behaviour Trees
System Launch
Embedded Space System
Health Status Data
Operations & Maintenance
Operations Centre (OC)
System Monitoring
Alarm
Error
System Health Monitoring Tool
Diagnosis & Troubleshooting
Operator
System Lifecycle

Figure 1: Overview of the proposed approach: creating ODMs during system design phase (along with SE & SA models), to be used as a basis for the "System Health Monitoring Tool".

3. Literature Review

This section consists of a description of the current State of the Art (SoA) in the fields of MBSE –section 3.1, and MBSA, as a prominent System Safety Assessment (SSA) methodology –section 3.2. The aim is to introduce the reader to the topics involved in our approach. Finally, subsection 3.3 presents the basic semantics of BTs, which is the formalism used to create the ODMs.

3.1. Model-Based Systems Engineering (MBSE)

MBSE facilitates the fulfillment of the SE requirements through design –structural and behavioural system modelling in various decomposition levels. SE *models* allow the engineers to verify the solution's design, functions and envisaged operations, early on in the development process, resulting to the reduction of the overall project cost. MBSE allows design alterations to take place in the beginning of system development and not when the solution is furtherly developed.

By applying MBSE, one can create *architectural models* (express system structure and decomposition), *functional models* (express system functions), and *behavioural models* (express system behaviour in different operational scenarios). To represent these models, several types of formalism are used, along with their associated languages and methodologies. The most common languages used in MBSE [16] are SysML [17] –based on OMG's[7] UML [18] standard, and EAST-ADL [19]. ARCADIA [20], IDEF [21], and OPM [22] consist both MBSE languages and methodologies. In terms of tools, IBM's Rhapsody [23] and Magic Draw's Cameo [24]–both based on SysML, and Eclipses' open-source Capella [25]–based on the Arcadia method, are the most popular.

3.2. System Safety Assessment (SSA) and Model-Based Safety Assessment (MBSA)

System Safety Assessment (SSA) is an analysis which leads to the production of a series of documents reporting all the identified hazards for the system in question, and the complying with the safety requirements [26]. For each identified system Failure Condition (FC)–how the system could fail, characterised by their level of impact to the environment, humans and the system itself (for space systems: catastrophic, critical, major, minor/negligible) [27], an analysis shall be made, in order to identify the possible causes–and combination of, that can lead to these FCs.

A recently developed approach for performing SSA or Reliability, Availability, Maintainability and Safety (RAMS) analysis, is MBSA. It is a technique which models the system's structure and behavior in order to provide safety analysis results. In this approach, various SSA-related activities are based on formal models which contain system dysfunctional data. In order to perform dysfunctional analysis at system level, it is required to have a fundamental

[7]The Object Management Group® (OMG®) is an international, open membership, not-for-profit technology standards consortium, founded in 1989. OMG standards are driven by vendors, end-users, academic institutions and government agencies.

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 SE model, and (b) the various ways the failures can occur and propagate inside the system [28].

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 feared events, which affect operational availability (when a critical error occurs, the system is not available until the error is resolved). The resulting analytical models are manually built by safety engineers from the available SE information. These models can be used to compute quantitative and/or qualitative safety indicators. Several mathematical formalisms are used to support the computations, mainly Markov chains [29], Petri nets [30] or Finite State Machines (FSMs) [31], each with different variants. These types of underlying formalisms are important for the qualification of computation tools.

As in MBSE, a number of modelling languages supporting MBSA were developed, such as *AltaRica* (AltaRica LaBRI [32], AltaRica DataFlow [33], AltaRica 3.0 [34]), *Figaro* [35], *SAML* [36], *HiP-HOPS* [37], *Component Fault Trees* [38], *Generalized Stochastic Petri Nets* [39] and *Safety Architect* [40]. In our approach we use the AltaRica language, since it has has become a de-facto European industrial standard for MBSA [41]. The most mature industrial tools for MBSA based on AltaRica are Dassault's Cecilia OCAS [42] and Apsys' SimfiaNeo [43].

3.3. Behaviour Trees

BTs were first invented as a tool to build modular Artificial Intelligence (AI) in computer games. A known alternative to FSMs, BTs are meant to provide better modularity, scalability, extendibility, adaptability and reuse of code [44, 45, 46, 47, 48, 49, 50], and to be easier to understand for humans, thus allowing incremental functionality design and efficient testing. On the benefits of modularity, Bagnell et al. state that *"individual behaviors can be reused in the context of another higher-level behavior, without needing to specify how they relate to subsequent behaviors"* [51].

According to Colledanchise and Ögren, *"a Behavior 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"* [52]. According to García et al., a Behavior 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"* [53].

BTs are widely used in systems control since they provide an easy way to perform conditional state switching. However, the main reason we have chosen BTs as the means to construct the ODM is that they provide an answer to the question *"What is the system currently doing?"*, which is precisely the first question the operators ask themselves, both when the system is in nominal mode, as well as when an issue has been identified.

One of the benefits of BTs we have additionally identified is that their elements can be in the form of blackboxes and be furtherly developed when new information is available; that being during the design phase or the operations phase –in the case of telemetry data reception. In the following section we demonstrate that BTs are easy to integrate within the design process. Based on the MBSE and MBSA models, and with operational objectives, we can create a model that can then be used inside a monitoring tool, suited for diagnosis. In our modelling approach we use the

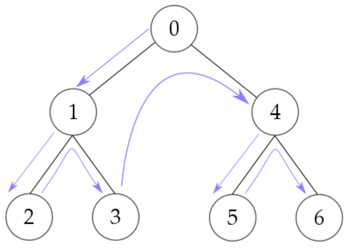


Figure 2: BTs' sequence execution.

classic formulation of BTs as described in [52], where BTs can be considered as a form of directed tree, where the flow amongst its nodes and edges is sequential in a left to right, depth first manner. The ticking execution sequence is illustrated in Figure 2.

BTs represent the possible system behaviours and their changes, in contrast to decision trees, which represent a logical formula, usually for automated decision purposes. BT nodes are visited at each time step (called "tick") during execution, starting from the root node. Parent nodes specify which children nodes must be visited, and in which order. Fallback and Sequence (parent) nodes define how the behaviour evolves when one of its child behaviour succeeds or fails. Action (child) nodes contain executable information. A classical BT formulation [52] is depicted in Figure 3.

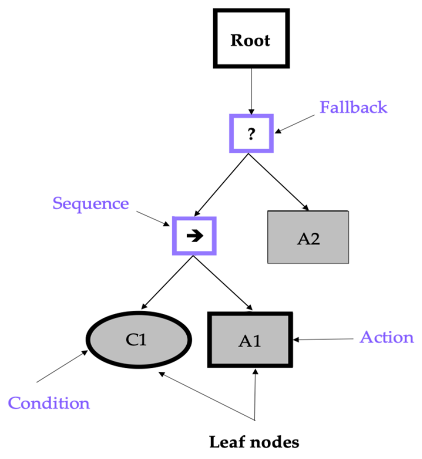


Figure 3: BTs' basic elements.

Regarding our BT implementation to construct the ODMs, we use the *PyTrees* (python implementation of BTs) library [54][55], which is relatively easy and intuitive to code with, so any engineer/scientist can build a BT model, with no particular programming background or developer skills. In our approach, BTs consist of *sequential tasks*: although PyTrees' ticking mechanism is compatible with parallel execution, we do not use this feature in our modelling approach, since no need has risen so far. Parallel execution in PyTrees is possible with the use of the the "parallel" composite node, which visits all its children simultaneously at each tick [56].

In terms of the trigger mechanism, we propose a *time-triggered* (vs event-driven [57] pg. 199) BT execution, in a way that the BT nodes are triggered in a time-increment manner and not by the order of the modelled events' occurrence. When representing structural system information, we often use time-triggered modelling –in contrast with dynamic system information, where event-driven modelling is preferred. Therefore, when dealing with *highly complex* systems, we prefer to represent them with time-triggered models that can periodically check for the occurrence (or not) of the modelled events, in an effort to reduce modelling complexity.

Regarding the *periodicity* of BTs, BT nodes are independent concurrent processes, that may or may not follow a mutual clock, e.g. they may terminate at any time. However, the BT is only active at every tick, which means BT events (starting a new action, mostly) only take place upon a tick. Ticks should be relatively fast with respect to the system's dynamics.

BTs can be modelled to be state-full i.e. having memory of all the visited nodes' state –state information can be either in the node type or the node instance (its source code), or state-less i.e. not "remembering" the state of multiple nodes, rather than saving only the state of the lastly triggered node. Clearly this option changes the behaviour of the composite nodes, while PyTrees can support both styles. By making the state-less modelling choice, one can affect the performance of the BT: state-less BTs offer less control over the workflow, thus this control must be implemented in the Action leaf nodes.

The BT implementation presented in this paper is using the default PyTrees' setting for Sequence and Fallback nodes, which is memory-full and memory-less, respectively. More specifically, the class syntax for the Sequence and Fallback nodes is respectively the following[56]:

```
class py_trees.composites.Sequence(name=Sequence, memory=True,  children=None)
class py_trees.composites.Selector(name=Selector, memory=False, children=None)
```

In regard to the *interruption* of BT behaviours, there may be many reasons to interrupt a node, some related to the BT, some completely unrelated. Action nodes are free to implement their internal interruption management mechanism: it integrates easily in the BT paradigm. However, many libraries (including PyTrees) also include an interruption mechanism for BT-related interruptions (in Fallback nodes mostly). In PyTrees, the memory parameter of Selector nodes inhibits the interruption mechanism, e.g. with memory=True the Selector node does not try to restart "high priority" terminated children, so there are no interruptions. With memory=False, the Selector node always queries the high priority children first, so they can "restart" and interrupt a currently running low priority child.

A simple example would be a system that tries to buy an item, but if it does not have enough money, works to earn it. The version *without interruption* could be:

- Acquire item (Sequence)
  - Work until enough money
  - Buy item

In this version, the "Work until enough money" must internally check whether there is enough money, and terminate or perform work accordingly. The version *with interruption* could be:

- Acquire item (Selector(memory=False))
  - Try to buy (Sequence)
    - Has enough money (Condition)
    - Buy item
  - Work

In this tree, the "Has enough money" and "Work" activities are separated. Because of "memory=False", even when the "Work" node is running, the BT still executes the "Try to buy" node. When the latter fails, it goes back to the "Work" node. When the "Try to buy" node finally succeeds, the BT automatically interrupts the "Work" node.

Existing BT libraries are built for control purposes, and rely almost exclusively on the past behaviour statuses "success", "failure" and "running" to decide to start or interrupt other behaviours. In our approach, we require that the status of each behaviour can be enriched with health indicators, such as alarms, specific to each behaviour. This enriches the ticking mechanism with an alarm gathering mechanism, that can be used for User-Interface (UI) or automated reasoning purposes. This mechanism can be flexibly implemented in the PyTrees library through the use of visitors. This feature is illustrated in the next section.

4. Proposal

In traditional satellite system development process, the SE, SA and Operations stages occur sequentially –each stage commences after the precedent's completion, and evolve linearly in time –no recurring loop is connecting the three stages. There is however an exception regarding the SE and SA stages, the activities of which interlock during system design. Nevertheless, SE and SA activities finish long before the system operation phase starts. Therefore, if any issue is detected at operation time (which could have been avoided by a change in design), there will be no modification in the SE or SA documents and models, since the latter would significantly augment the project cost. This modification would thus be dismissed, even if the operational impact would be beneficial in the long term.

In our proposal, SE and SA documents and models are being concurrently developed with the ODM. Hence operational aspects can be taken into account in the SE and SA activities. This would not only increase the confidence of the system design in regards to its representation of the actual system, but also to the foreseeing of possible operational issues, which can be avoided by modifying the system design, early on the system life cycle [58]. This would also decrease the system development cost, while optimising system design.

SE is defined by an iterative system development process. All teams involved in the system design (SE/MBSE, RAMS/SA) are striving for continued improvement through constant dialogue and joint meetings at each iteration step –concept of concurrent design [59]. Our proposal consists in *incorporating the ODMs' team in the feedback loop*, so as to *increase the agility* of the system design process. This shall imply the involvement of more stakeholders (operators / ODM architects) in the design phase.

The proposed methodology implies the constant amelioration of system design at each iteration loop, which increases the efficiency of the system design activities. The co-conception and co-design of system models means that any modifications are made early on the life cycle development process; in contrast to making changes at the end, during the V&V and testing phase. Hence an increase of efficiency is achieved, by reducing the total time and cost of the design phase. The proposal thus not only is agile, but also improves the agility of current methodologies, widely used in the aerospace industry.

4.1. Illustration of the ODM construction process

As mentioned in section 3.3, one of the benefits of BTs is that its elements can be in the form of black boxes and be furtherly developed when new information is available. This way, BT models can be easily integrated within the design process. Based on the SE and SA models, and with operational objectives, we can create a model that can then be used inside a monitoring tool, suitable to diagnosis. An illustration of our proposal is presented in Table 1, where a juxtaposition of SE and SA information and the implementation into a BT is depicted.

As shown in Table 1, information can be added in the BT models gradually, when new information concerning the system is available. In this example we tackle the case of an Earth Observation satellite that shall fulfill its primary mission, which is to take photos of the area(s) of interest on Earth and transmit them back to the OC, as we can see in the first model iteration. 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 model–as shown in the 3rd column, where a single action node "Observe Earth" is included in the tree, under the "Mission" node–here a Fallback node.

Throughout the system design development, the FMEA team can produce new system information, as shown in the second column, as for example, "Mission may fail". This information can be integrated in the BT model with the addition of a new action node ("Mission Fail"), on the right side of the "Observe Earth" node, which represents the system's nominal mode. This would mean that, if the "Observe Earth" node ticking returns "failure", the "Mission Fail" node will be ticked. If the "Observe Earth" node returns "success" or "running", the "Mission Fail" 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 model by adding a new action node between the "Observe Earth" and "Mission Fail" node, which represents the "Standby" mode of the system. Consequently, if the "Observe Earth" node ticking returns "failure", the "Standby" node will be ticked. If the "Standby" node also returns "failure", the "Mission Fail" node is ticked. If the "Observe Earth" node returns "failure" and the "Standby" node returns "success" or "running" the "Mission Fail" node will not be ticked. The latter would signify–in case of system monitoring, 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, send photos to Operations Centre. The "Observe Earth" action node can then be turned into a Sequence node in the BT model, and have three children, namely "Capture photos", "Save photos" and "Send photos". 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 BT will return "failure" as well.

Lastly, we assume that the FMEA teams communicate to the SE teams that a fault leading to instrument overheating thus causing the mission to fail can be detected by the operator, if able to monitor the instrument's health status data during the "Capture photos" phase. The SE team can then conclude that adding temperature monitoring capabilities in the picture capturing instrument can help prevent system failure, and implement it in the system design. This new design modification shall also be added in the BT; this way the BT model is up-to-date with the SE and SA system models.

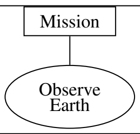
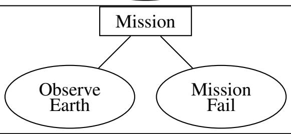
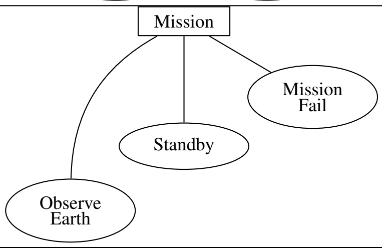
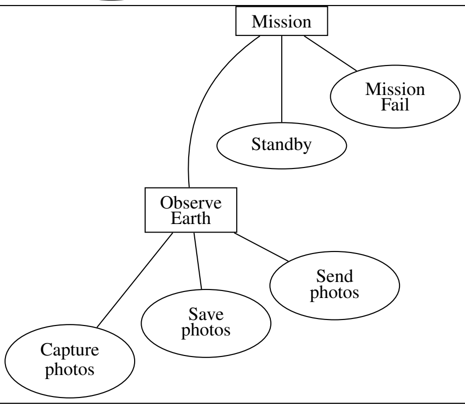
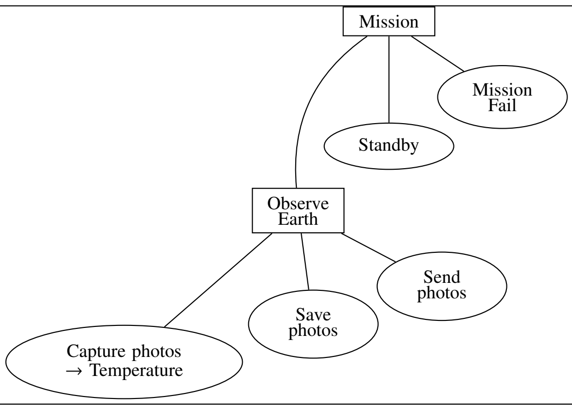
4.2. Exploitation of the ODM: Diagnostic Tools

The ODM is being created in parallel with the system functional and dysfunctional models and documents. How this activity ultimately improves the system's resilience is out of the scope of this paper, since there are many more steps in between. However, we can already point out that since the ODM is created at the design stage, it is possible to analyze it, identify design flaws that may impede operations, and derive new SE and SA requirements to address these flaws.

The most direct way to use the ODM is to build a UI component out of it, that displays the behaviour statuses in a hierarchical way, including possible alarms. The hierarchical nature of the model helps operators focus on relevant behaviours to quickly pinpoint problems and launch appropriate troubleshooting procedures.

Automated diagnosis techniques based on models [3] or data [4] can easily be integrated when they focus on particular components or functions of the system. Their output can be used to raise alarms in the associated behaviours,

Table 1: Illustration of the first iterations of a satellite design. At each iteration, the BT model can incorporate information coming from the system design process related activities, while remaining at each step a valid model.

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		 Mission Mission Fail Standby Observe Earth Send photos Save photos Capture photos
	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		 Mission Mission Fail Standby Observe Earth Send photos Save photos Capture photos → Temperature

or alternatively the automated diagnosis tool can be considered a behaviour in itself, that raises alarms when detecting abnormal situations.

The ODM is a model that can be used to support automated reasoning techniques. A pattern of alarms in some behaviours could be automatically associated to a diagnosis, or a troubleshooting procedure. There are many ways to implement such reasoning (case-based reasoning, decision rules, constraint programming), that depend on aspects specific to the system and its environment.

Taking the example illustrated in Table 1 as a use case, we created a tool to display the status of the BT nodes called at each execution i.e. the BT nodes' response to the embedded ticking mechanism, which can be one of the three: "Success", "Failure" or "Running". Each execution is defined by an incremental time step $T(t_n)$, where $t = t_0, ..., t_{n-1}$, $n \in \mathbb{Z}$, $t_0 = 0$. We also implemented an "Alarm" feature, so that raised alarms associated with a behaviour can also also appear in the tool interface.

Table 2 shows an example interface of a diagnostic tool using the BT model of the last row and column of Table 1 as its ODM, after 3 time executions. In this case scenario, we can observe that in the first execution (T0), the status of the root node "Mission" is "Running", since the status of its first node "Observe Earth" is also "Running". As a Fallback node, the "Mission" node "inherits" the status of its first successful or running child node. After another "Running" interval, the status of the satellite mission function returns "Sucess", indicating the mission's successful completion (with or without errors in between).

The Sequence node "Observe Earth" needs all of its children nodes to succeed in order for it to take the status "Sucess"; on the contrary, if at least one of its children nodes returns "Failure", the "Observe Earth" function would equally fail. In this case, its first child node "Capture Photos", along with the temperature check of its camera sensor has successfully completed its function. Then, we can see that the following function "Save Photos" is "Running", meaning that the satellite is in the process of saving the photos taken. Thus the function "Observe Earth" is in process, hence the satellite is currently undergoing its principal mission.

Table 2: Diagnostic tool UI example.

Type of Node	Behaviour Name	Status T0	Status T1	Status T2
Fallback	Mission	Running	Running	Success
Sequence	Observe Earth	Running	Running	Success
Action	Capture Photos / Check Temperature	Success	Success	Success
Action	Save Photos	Running !Alarm!	Success	Success
Action	Send Photos	–	Running	Success
Action	Standby	–	–	–
Action	Mission Fail	–	–	–

However, we can see that the "Save Photos" function has raised an alarm, which would indicate to the operator that there might be a problem related to the On-Board Computer (OBC), the memory e.g. over-warmed electronics, etc. In such case, the operator shall have a troubleshooting sequence related to each error or alarm raised, which they would follow in order to establish a road-map leading to the fault resolution. In the next time interval (T1), we can safely assume that the satellite is in the process of sending the photos to the OC, while in the third (T2), that the satellite has successfully completed its mission for the moment.

If the operator had access solely to the information presented in the last column (T2), which only displays the results of the "Observe Earth" function –and not the underlying process, they would not be able to "see" the alarm raised during the photos saving function. A model-based diagnostic tool which uses an ODM with a BT formalism can have several benefits over traditional diagnostic procedures, such as the one illustrated in this use case scenario. Effectively, a comparison with current monitoring methods can only be made by using a real-life model of a complex satellite system, and feedback from operators. In this paper we present only a proof-of-concept.

5. Conclusion

Monitoring tools for diagnosis during system operations are, to this day, still lacking important information needed for the operators to perform their supervision and diagnostic tasks efficiently. They mainly rely on knowledge acquired through previous experience as well as engineers' and operators' know-how, rather than system design elements. For this reason we propose a methodology for the concurrent construction of a model dedicated to system monitoring for diagnosis, along with the system design (construction of SE and SA models). Our methodology would not only increase the agility of the system development process, but also the confidence on the system design by creating a monitoring tool which is equally based on its design information as well as ROE. Eventually, our proposal shall be validated by real-life operators, after having tested the model's integration in a specific tool, as part of a representative case study. We plan to evaluate it through feedback from industrial partners, research labs, and space agencies.

References

[1] W. Kim, S. Katipamula, A review of fault detection and diagnostics methods for building systems, Science and Technology for the Built Environment 24 (2018) 3–21.

[2] K. Djebko, F. Puppe, H. Kayal, Model-based fault detection and diagnosis for spacecraft with an application for the sonate triple cube nano-satellite, Aerospace 6 (2019).

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

[4] Y. Lei, B. Yang, X. Jiang, F. Jia, N. Li, A. K. Nandi, Applications of machine learning to machine fault diagnosis: A review and roadmap, Mechanical Systems and Signal Processing 138 (2020) 106587.

[5] A. Barua, K. Khorasani, Hierarchical fault diagnosis and fuzzy rule-based reasoning for satellites formation flight, IEEE Transactions on Aerospace and Electronic Systems 47 (2011) 2435–2456.

[6] P. Baldi, M. Blanke, P. Castaldi, N. Mimmo, S. Simani, Combined geometric and neural network approach to generic fault diagnosis in satellite reaction wheels, IFAC-PapersOnLine 48 (2015) 194–199. 9th IFAC Symposium on Fault Detection, Supervision andSafety for Technical Processes SAFEPROCESS 2015.

[7] C. Pittet, A. Falcoz, D. Henry, A model-based diagnosis method for transient and multiple faults of aocs thrusters, IFAC-PapersOnLine 49 (2016) 82–87. 20th IFAC Symposium on Automatic Control in AerospaceACA 2016.

[8] R. J. Patton, F. J. Uppal, S. Simani, B. Polle, Robust fdi applied to thruster faults of a satellite system, IFAC Proceedings Volumes 40 (2007) 1–6. 17th IFAC Symposium on Automatic Control in Aerospace.

[9] A. Valdes, K. Khorasani, A pulsed plasma thruster fault detection and isolation strategy for formation flying of satellites, Applied Soft Computing 10 (2010) 746–758.

[10] O. Lisagor, L. Sun, T. Kelly, The illusion of method: Challenges of model-based safety assessment, in: 28th international system safety conference (ISSC), p. Num Pages: 10.

[11] Y. Papadopoulos, M. Walker, D. Parker, E. Rüde, R. Hamann, A. Uhlig, U. Grätz, R. Lien, Engineering failure analysis and design optimisation with hip-hops, Engineering Failure Analysis 18 (2011) 590–608.

[12] Society of Automotive Engineers (SAE) International, Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment, SAE International, p. 331.

[13] O. Lisagor, T. Kelly, R. 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, pp. 625–632.

[14] S. K. Ibrahim, A. Ahmed, M. A. E. Zeidan, I. E. Ziedan, Machine learning techniques for satellite fault diagnosis, Ain Shams Engineering Journal 11 (2020) 45–56.

[15] W. Wolf, Hardware-software co-design of embedded systems, Proceedings of the IEEE 82 (1994) 967–989.

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

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

[18] Object Management Group (OMG), Unified Modeling Language (UML), `http://www.omg.org/spec/UML/current`, 2021.

[19] EAST-ADL Association, EAST Architecture Description Language (EAST-ADL), `https://www.east-adl.info/Specification.html`, 2021.

[20] Thales, ARChitecture Analysis & Design Integrated Approach (ARCADIA), `https://www.eclipse.org/capella/arcadia.html`, 2021.

[21] Knowledge Based Systems, Inc. (KBSI), Integration DEFinition (IDEF), `https://www.idef.com/`, 2021.

[22] International Organisation for Standardization (ISO), Object Process Methodology (OPM), `https://www.iso.org/standard/62274.html`, 2021.

[23] International Business Machines Corporation (IBM), IBM®Rational®Rhapsody®Architect for Systems Engineers, `https://www.ibm.com/products/systems-design-rhapsody`, 2021.

[24] No Magic, Cameo Systems Modeler, `http://www.nomagic.com/products/cameo-systems-modeler.html`, 2021.

[25] Eclipse Foundation, Eclipse Capella^{TM}, `https://www.eclipse.org/capella/`, 2021.

[26] M. Verrastro, I. Dimino, Chapter 21 - morphing devices: Safety, reliability, and certification prospects, in: A. Concilio, I. Dimino, L. Lecce, R. Pecora (Eds.), Morphing Wing Technologies, Butterworth-Heinemann, 2018, pp. 647–682.

[27] European Cooperation for Space Standardization, ECSS-Q-ST-40C Rev.1: Space Product Assurance - Safety, 2017.

[28] M. Machin, E. Saez, P. Virelizier, X. de Bossoreille, Modeling Functional Allocation in AltaRica to Support MBSE/MBSA Consistency, in: Y. Papadopoulos, K. Aslansefat, P. Katsaros, M. Bozzano (Eds.), Model-Based Safety and Assessment, Springer International Publishing, Cham, 2019, pp. 3–17.

[29] P.-A. Brameret, J.-M. Roussel, A. Rauzy, Preliminary system safety analysis with limited markov chain generation, IFAC Proceedings Volumes 46 (2013) 13–18. 4th IFAC Workshop on Dependable Control of Discrete Systems.

[30] N. Leveson, J. Stolzy, Safety analysis using petri nets, IEEE Transactions on Software Engineering SE-13 (1987) 386–397.

[31] X. Chen, J. Jiao, A fault propagation modeling method based on a finite state machine, in: 2017 Annual Reliability and Maintainability Symposium (RAMS), pp. 1–7.

[32] Laboratoire Bordelais de Recherche en Informatique (LaBRI), AltaRica Project - MEthods and Tools for AltaRica Language, `https://altarica.labri.fr/wp/`, last accessed January 2022.

[33] M. Boiteau, Y. Dutuit, A. Rauzy, J.-P. Signoret, The altarica data-flow language in use: modeling of production availability of a multi-state system, Reliability Engineering & System Safety 91 (2006) 747–755.

[34] T. Prosvirnova, M. Batteux, P.-A. Brameret, A. Cherfi, T. Friedlhuber, J.-M. Roussel, A. Rauzy, The altarica 3.0 project for model-based safety assessment, IFAC Proceedings Volumes 46 (2013) 127–132. 4th IFAC Workshop on Dependable Control of Discrete Systems.

[35] M. Bouissou, H. Bouhadana, M. Bannelier, N. Villatte, Knowledge modelling and reliability processing: Presentation of the figaro language and associated tools, IFAC Proceedings Volumes 24 (1991) 69–75.

[36] M. Lipaczewski, S. Struck, F. Ortmeier, Using tool-supported model based safety analysis – progress and experiences in saml development, in: Proceedings of IEEE International Symposium on High Assurance Systems Engineering, pp. 159–166.

[37] Y. Papadopoulos, J. McDermid, Hierarchically performed hazard origin and propagation studies, in: Proceedings of the 18th SAFECOMP International Conference, volume 1698, pp. 139–152.

[38] B. Kaiser, D. Schneider, R. Adler, D. Domis, F. Möhrle, A. Berres, M. Zeller, K. Höfig, M. Rothfelder, Advances in component fault trees, CRC Press, p. Num Pages: 9.

[39] G. Balbo, Introduction to Generalized Stochastic Petri Nets, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 83–131.

[40] ALL4TEC, Safety Architect - Model Based Safety Assessment software, `https://www.all4tec.com/en/safety-architect-fmeca-fta-sofware/`, last accessed January 2022.

[41] M. Bozzano, A. Cimatti, O. Lisagor, C. Mattarei, S. Mover, M. Roveri, S. Tonetta, Safety assessment of AltaRica models via symbolic model checking, Science of Computer Programming 98 (2015) 464–483.

[42] P. Bieber, J. P. Blanquart, G. Durrieu, D. Lesens, J. Lucotte, F. Tardy, M. Turin, C. Seguin, E. Conquet, Integration of formal fault analysis in ASSERT: Case studies and lessons learnt, in: Embedded Real Time Software and Systems (ERTS2008), Toulouse, France, p. Num Pages: 9.

[43] M. Machin, L. Sagaspe, X. de Bossoreille, Simfianeo, complex systems, yet simple safety, in: Embedded Real Time Software and Systems - ERTS 2018, p. 4.

[44] M. Colledanchise, A. Marzinotto, D. V. Dimarogonas, P. Oegren, The Advantages of Using Behavior Trees in Mult-Robot Systems, in: Proceedings of ISR 2016: 47st International Symposium on Robotics, pp. 1–8.

[45] P. Ogren, Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees, in: AIAA Guidance, Navigation, and Control Conference, American Institute of Aeronautics and Astronautics, Minneapolis, Minnesota, 2012, pp. 1–8.

[46] M. Colledanchise, P. Ögren, How Behavior Trees modularize robustness and safety in hybrid systems, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1482–1488. ISSN: 2153-0866.

[47] M. Colledanchise, P. Ögren, How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees, IEEE Transactions on Robotics 33 (2017) 372–389. Conference Name: IEEE Transactions on Robotics.

[48] M. Colledanchise, R. Parasuraman, P. Ögren, Learning of Behavior Trees for Autonomous Agents, IEEE Transactions on Games 11 (2019) 183–189. Conference Name: IEEE Transactions on Games.

[49] A. 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, 2013, pp. 1–11.

[50] F. Rovida, B. Grossmann, V. Krüger, Extended behavior trees for quick definition of flexible robotic tasks, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 6793–6800. ISSN: 2153-0866.

[51] J. A. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. Pollard, M. Pivtoraiko, J.-S. Valois, R. Zhu, An integrated system for autonomous robotics manipulation, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2955–2962.

[52] M. Colledanchise, P. Ögren, Behavior Trees in Robotics and AI: An Introduction, arXiv:1709.00084 [cs] (2018). ArXiv: 1709.00084.

[53] S. García, P. Pelliccione, C. Menghi, T. Berger, T. Bures, High-level mission specification for multiple robots, in: Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2019, Association for Computing Machinery, New York, NY, USA, 2019, pp. 127–140.

[54] Google JAX, PyTrees Library Documentation, `https://py-trees.readthedocs.io/en/devel/`, last accessed January 2022.

[55] Google JAX, PyTrees Github space, `https://github.com/splintered-reality/py_trees`, last accessed January 2022.

[56] Google JAX, PyTrees Library Documentation - Composites, `https://py-trees.readthedocs.io/en/devel/composites.html`, last accessed January 2022.

[57] R. Ghzouli, T. Berger, E. B. Johnsen, S. Dragule, A. Wasowski, Behavior trees in action: a study of robotics applications, in: Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering, ACM, Virtual USA, November 2020, pp. 196–209.

[58] W. K. Vaneman, The system of systems engineering and integration "vee" model, in: 2016 Annual IEEE Systems Conference (SysCon), pp. 1–7.

[59] S. FINGER, M. S. FOX, F. B. PRINZ, J. R. RINDERLE, Concurrent design, Applied Artificial Intelligence 6 (1992) 257–283.