Timed Failure Propagation Analysis for Spacecraft Engineering: The ESA Solar Orbiter Case Study

Benjamin Bittner*, Marco Bozzano, and Alessandro Cimatti

Fondazione Bruno Kessler
{bittner,bozzano,cimatti}@fbk.eu

Abstract. Timed Failure Propagation Graphs (TFPGs) are used in the design of safety-critical systems as a way of modeling failure propagation, and to support the evaluation and implementation of functions for Fault Detection, Isolation, and Recovery (FDIR). TFPGs are a very rich formalism: they enable modeling Boolean combinations of faults and events, and quantitative delays between them. Several formal techniques have been recently developed to analyze them as stand-alone models or to compare them to models that describe the more detailed dynamics of the system of reference, specifically under faulty conditions.
In this paper we present several case studies that apply TFPGs to Solar Orbiter, an ESA deep-space probe under development by Airbus. The mission is characterized by high requirements on on-board autonomy and FDIR. We focus on three possible application areas: hardware-to-software propagations, system-level propagations, and propagations across architectural hierarchies. The case studies show the added value of TFPGs for safety analysis and FDIR validation, as well as the scalability of available analysis tools for non-trivial industrial problems.

1 Introduction

Modern complex engineering systems, such as satellites, airplanes and traffic control systems need to be able to handle faults. Faults may cause failures, i.e. conditions such that particular components or larger parts of a system are no longer able to perform their required function. As a consequence, faults can compromise system safety, creating a risk of damage to the system itself or to the surrounding infrastructure, or even a risk of harm to humans. Faults can also affect the availability of a system, for instance by causing service outages. In some applications such as telecom satellites or intelligence infrastructure, such outages might be unacceptable. For these reasons, complex system needs to tolerate faults – either passively, for instance through robust control laws, or actively, implementing a Fault Detection, Isolation and Recovery (FDIR) sub-system.

The first step in developing an FDIR architecture is to identify the faults and their effects on the system. Standard analyses employed for this task include Fault Tree Analysis (FTA) [16] and Failure Modes and Effects Analysis (FMEA) [10]. For a review on the state-of-the-art in modeling and tools we refer to [15]. These techniques however don't have a comprehensive support for timing of failure propagations and are specialized to specific discrete analyses that make it difficult to obtain a global integrated

* The first author was partially supported by ESA NPI contract No. 4000111815/14/NL/FE.

picture of the overall failure behavior of a system. This in turn makes it difficult to develop a coherent set of detailed FDIR requirements and to check whether a given FDIR architecture is able to handle all possible faults and their propagation effects.

To address these issues, *Timed Failure Propagation Graphs* (TFPGs) [12, 1] were recently investigated as an alternative failure analysis framework. TFPGs are labeled directed graphs that represent the propagation of failures in a system, including information on timing delays and mode constraints on propagation links. TFPGs can be seen as an abstract representation of a corresponding dynamic system of greater complexity, describing the occurrence of failures, their local effects, and the corresponding consequences over time on other parts of the system. TFPGs are a very rich formalism: they allow to model Boolean combinations of basic faults, intermediate events, and transitions across them, possibly dependent on system operational modes, and to express constraints over timing delays. In a nutshell, TFPGs integrate in a single artifact several features that are specific to either FMEA or FTA, enhanced with timing information.

TFPGs have been investigated in the frame of the FAME project [8, 9, 4], funded by the European Space Agency (ESA). Here, a novel, model-based, integrated process for FDIR design was proposed, which aims at enabling a consistent and timely FDIR conception, development, verification and validation. More recently, [6, 5, 3] have investigated TFPG-based validation and formal analyses. In particular, [6] focuses on the validation of TFPGs, seen as stand-alone models, using Satisfiability Modulo Theories (SMT) techniques; [5] addresses TFPG validation, and tightening of TFPG delay bounds, with respect to a system model of reference; finally, [3] develops algorithms for the automatic synthesis of a TFPG from a reference system model.

In this paper we present several case studies that apply TFPGs to an ESA deep-space probe design under development by Airbus, whose mission is characterized by high requirements on on-board autonomy and FDIR. These case studies were produced during a 10-month research stay at ESA-ESTEC, in which we studied the application of TFPGs for failure analysis in the "Solar Orbiter" (SOLO). We present three TFPG case studies for SOLO: one looking at error propagation from hardware to software, one studying failure propagation during detection and isolation activities on system-level, and one considering propagation across architectural layers. Based on the case studies, we make some observations on how TFPGs can improve our understanding of system failure dynamics and how they can be used to validate and tune FDIR design coverage. Finally, we describe how the analyses helped to raise five issues that were submitted to the FDIR critical design review. Four of them were classified as major, one as minor.

The case studies show the adequacy and added value of TFPGs for safety analysis and FDIR validation, as well as the scalability of available analysis tools for non-trivial industrial problems. Moreover, the issues we found led to an improvement of the available design documentation, and in one case triggered a modification of the design, as the analyses unveiled a missing consistency check.

The rest of the paper is structured as follows. Sect. 2 describes in detail the syntax and semantics of TFPGs. In Sect. 3 we present the main TFPG-related analyses available in the XSAP tool for model-based safety analysis and used for the case studies. Sect. 4 contains a detailed discussion of the case studies. We conclude with a summary and outlook on interesting directions in Sect. 5.

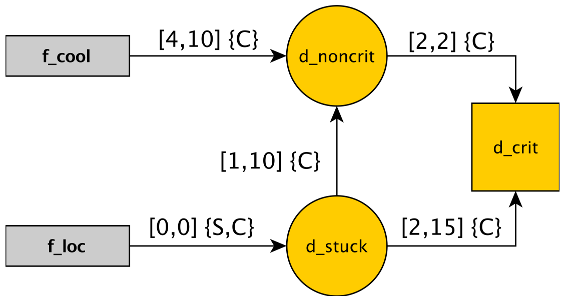

Fig. 1. TFPG for the ForgeRobot example. Rectangles are failure mode nodes, squares are `AND` nodes, and circles are `OR` nodes.

2 Timed Failure Propagation Graphs

TFPGs – first described in [12, 11] – are directed graph models where nodes represent *failure modes* (root events of failure propagations) and *discrepancies* (deviations from nominal behavior caused by failure modes). Edges model the temporal dependency between the nodes. They are labeled with propagation *delay bounds*, and *system modes* indicating the system configurations in which the propagation is possible. TFPGs are formally defined as follows.

Definition 1 (TFPG) *A* TFPG *is a structure $G = \langle F, D, E, M, ET, EM, DC\rangle$, where:*

- *F is a non-empty finite set of failure modes;*
- *D is a non-empty finite set of discrepancies;*
- *$E \subseteq V \times D$ is a non-empty set of edges connecting the set of nodes $V = F \cup D$;*
- *M is a non-empty set of system modes (we assume that at each time instant the system is in precisely one mode);*
- *$ET : E \rightarrow I$ is a map that associates every edge in E with a time interval $[t_{min}, t_{max}] \in I$ indicating the minimum and maximum propagation time on the edge, with $I \in \mathbb{R}_{\geq 0} \times (\mathbb{R}_{\geq 0} \cup \{+\infty\})$ and $t_{min} \leq t_{max}$;*
- *$EM : E \rightarrow 2^M$ is a map that associates to every edge in E a set of modes in M (we assume that $EM(e) \neq \emptyset$ for every edge $e \in E$);*
- *$DC : D \rightarrow \{\texttt{AND}, \texttt{OR}\}$ is a map defining the discrepancy type;*

Failure modes never have incoming edges. All discrepancies must have at least one incoming edge and be reachable from a failure mode node. Circular paths are possible, with the exception of self-loops or zero-delay loops.

As an example we consider *ForgeRobot*, a robot working in a hypothetical industrial forge. The robot is either in standby in a safe area, or performs work in a critical area that has high heat levels. It moves around using its locomotion facilities. To prevent overheating in the critical area, a cooling system is used. The TFPG in Fig. 1 shows possible failures of the robot and their effects over time. Two modes are used to differentiate the operational context: S for safe area, and C for critical area. The locomotion

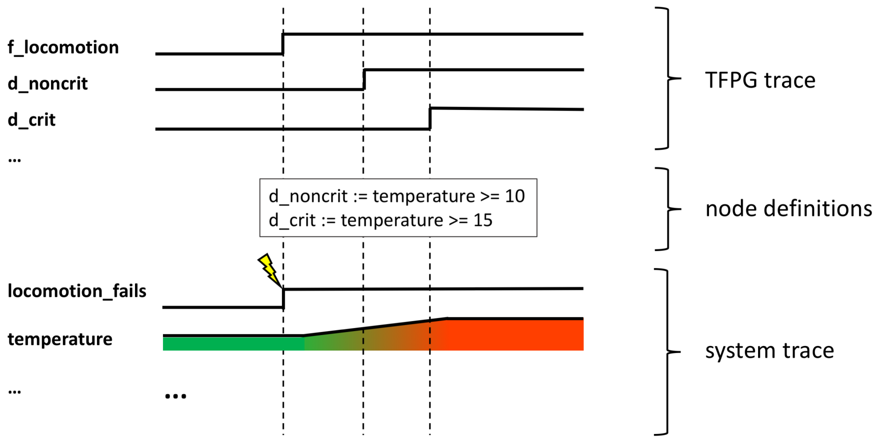

Fig. 2. Example of trace abstraction for ForgeRobot. Square signals are used to model Boolean values over time.

drive of the robot can fail (f_{loc}), causing the robot to be stuck (d_{stuck}). The cooling system can fail (f_{cool}), decreasing the performance of heat protection. f_{cool} and d_{stuck} can both independently cause a non-critical overheating of the robot ($d_{noncrit}$) in mode C. In case both happen, they cause a critical overheating (d_{crit}). The time ranges on the propagation edges represent the different propagation speeds, influenced by the variable amount of workload and of heat in the critical area.

According to the semantics of TFPGs [1], a TFPG node is activated when a failure propagation has reached it. An edge $e = (v, d)$ is active iff the source node v is active and $m \in EM(e)$, where m is the current system mode. A failure propagates through $e = (v, d)$ only if e is active throughout the propagation, that is, up to the time d activates. For an `OR` node d and an edge $e = (v, d)$, once e becomes active at time t, the propagation will activate d at time t', where $tmin(e) \leq t' - t \leq tmax(e)$, with $tmin(e)$ (resp. $tmax(e)$) representing the t_{min} (resp. t_{max}) parameter of edge e. Activation of an `AND` node d will occur at time t' if every edge $e = (v, d)$ has been activated at some time t, with $tmin(e) \leq t' - t$; for at least one such edge e we must also have $t' - t \leq tmax(e)$, i.e. the upper bound can be exceeded for all but one edge. If an edge is deactivated any time during the propagation, due to mode switching, the propagation stops. Links are assumed memory-less, thus failure propagations are independent of any (incomplete) previous propagation. A maximum propagation time of $t_{max} = +\infty$ indicates that the propagation across the respective edge can be delayed indefinitely, i.e. it might never occur at all. This is a useful over-approximation when the real t_{max} value is not available; it is also necessary when the propagation depends on some unconstrained input or other dynamics not captured by the TFPG.

3 TFPG Analysis with xSAP

Building a TFPG can be an error-prone and time-consuming activity. Just as it is difficult to get fault trees and failure mode and effect tables right, it is difficult to build TFPGs, possibly even more so as we combine several of their features and furthermore add timing information. There is therefore a clear need for a comprehensive framework to analyze and validate TFPGs. Such framework is provided by the xSAP safety analysis platform [2].

TFPG analyses implemented in xSAP cover validation of TFPGs as stand-alone models, or validation against a model of reference, i.e. a more detailed model representing the system's dynamic behavior. Validation of TFPGs as stand-alone models include analyses such as possibility, necessity, consistency and refinement checks [6].

Validation of TFPGs against a model of reference has the objective to make sure that no important failure behavior that is possible in the system is overlooked (i.e. not modeled) in the TFPG. Likewise, we want to make sure that the TFPG contains as few spurious behaviors as possible, even though it might be impossible to exclude all such behaviors, due to the approximate nature of the TFPGs. We define the following TFPG properties : *completeness* guarantees that all failure propagations possible in the system are captured by the TFPG; *edge tightness* guarantees that the time and mode constraints of propagations are as accurate as possible. The corresponding analyses in xSAP are called *behavioral validation* (completeness check) and *edge tightening* (generation of tighter bounds for TFPG edges, that preserve completeness) [5]. These analyses are implemented as verification problems in temporal logic model-checking. If the properties are violated, diagnostic information for debugging is provided.

In a second scenario, a TFPG can be automatically derived from the corresponding system model. This analysis is called *TFPG synthesis* in xSAP [3]. Synthesizing TFPGs may be preferable over manually creating them, in particular when it is not possible to leverage the results of previous safety analyses, or when the engineer performing the analysis does not have a sufficiently deep understanding of the system behavior under faults. Obviously the main burden then lies on the system modeler, but it is arguably easier to create a model that specifies the behavior of individual parts of the system, how they interact, and how they can fail locally, than it is to directly model the failure behavior that *emerges* from these local behaviors and interactions.

xSAP provides a set of algorithms to *automatically* derive TFPGs from the corresponding system models in a way that guarantees by construction a number of formal properties. The engineer needs to provide as input the system model, the set of failure modes, and the set of discrepancies and monitors that should be included in the end result. The link between the TFPG and the system of reference is defined by means of a trace-based semantics, which enables the comparison of TFPG behaviors (those compatible with the TFPG constraints) to behaviors that are possible in the system. Based on the traced-based semantics, TFPGs are formally defined as abstractions of system behavior. As an example, Fig. 2 shows a TFPG trace and the corresponding system trace in the *ForgeRobot* example. The algorithm for TFPG synthesis [3] is structured in three parts: generation of an initial verbose graph topology; simplification of the graph structure for improved readability; finally, tightening of the edge parameters for obtaining accurate propagation constraints.

4 Case Studies

Solar Orbiter (SOLO) is a Sun-observing satellite under development by the European Space Agency, planned to be launched in October 2018 [13]. It will orbit the Sun to perform various scientific observations which are very difficult or impossible to do from Earth. The FDIR requirements on SOLO are much more stringent than on typical Earth-observing satellites, especially due to the intensity of solar radiation. Many faults are highly time-critical as they can quickly cause considerable damage to the spacecraft. Detection, isolation, and recovery need thus to be performed in extremely short time frames.

Due to the mission's general complexity and time-critical faults, SOLO is an ideal context to evaluate the use of TFPGs in space systems engineering. Three case studies on timed failure propagation analysis were performed. To support system and TFPG modeling we referred to the project documentation related to FDIR (mostly FMECA and FDIR design coverage documents) and the general software/hardware architecture (for instance the Control Algorithm Specification). These documents were at a pre-CDR (Critical Design Review) level and thus contained considerable design details.

Collecting the necessary information for the case studies was very challenging especially due to the huge amount of documentation, which was several hundreds of pages just for the documents effectively used. Additionally we found that the information needed for the case studies was scattered throughout the documents, and collecting and interpreting everything required substantial work and several interactions with engineers. This gives an intuition of how difficult it is, for instance, to manually validate the FDIR design of the *whole* spacecraft and to verify that the overall FDIR design is coherent and covers all possible (and reasonably probable) effects of faults. A formal and structured approach to interpret this information, for instance by using TFPG modeling, is thus a clear benefit and increases the confidence in the completeness of the analysis.

The proprietary information that we used for our study is subject to non-disclosure, and therefore cannot be quoted literally. However, we remark that the models created for the case studies are generic; the problems the case studies deal with are quite universal and do not apply only to SOLO.

4.1 Hardware-to-Software Propagation

The first case study involved modeling and analysis of the gyroscope channel processing software function, which reads different types of raw gyroscope sensor data coming from the inertial measurement unit (IMU), retrieved over the bus and stored in the datapool. From these values the function computes the rotation rate around the axis on which the channel's gyroscope is positioned, along with health flags signaling data corruption. In total the function has 7 inputs and 13 outputs.

The function runs on the main computer and is called cyclically at fixed intervals. It is composed of several smaller subfunctions, some of which have internal state variables (in total 8) to store values computed in previous cycles for various purposes. Time elapses by one unit during tick events. After each tick, the values in the datapool are updated and the function uses them to compute, through various consecutive steps, the new output values.

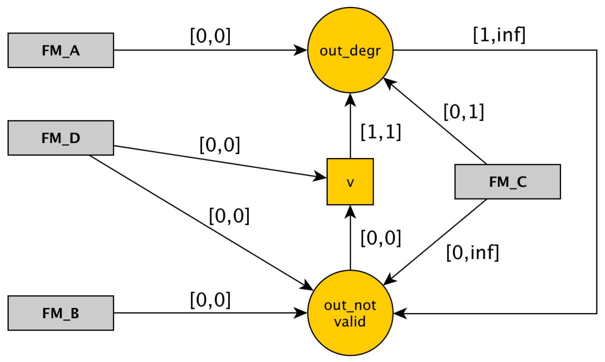

Fig. 3. Extract of TFPG for the gyroscope processing function. Multiple simultaneous faults and mode-switching are not considered.

For this function we created a model representing various computational steps on an abstract level. Variables with values in the reals are abstracted to discrete domains, such as “normal”, “degraded”, and “erroneous”. For degraded data readings we assume that the internal checks might or might not detect the corruption, thus including the possibility of detectable and undetectable levels of data corruption due to selected thresholds. Based on the IMU FMECA, 13 hardware faults where defined, which influence the values stored in the datapool. In the analysis we follow the single-fault assumption of FMECA. The faults are constrained to occur at the beginning of a cycle, such that we can analyze how many cycles (ticks) a fault needs to propagate to the function outputs. Failure modes are the 13 faults. We chose two discrepancies of interest expressed over the output values: degraded (and possibly undetectable) output measurements, and a Boolean data health flag indicating data corruption. The goal was to understand the temporal relationship between faults, the health flag, and degraded rate estimations.

The resulting model has in total 16 Boolean input variables and 84 Boolean state variables. The diameter of the corresponding reachable state-space is 105; this is the upper bound on the least number of transitions that need to be taken to reach any state from the set of initial states. In total 3488 states are reachable. Certain states can only be reached after executing the function several times, and since the function itself also consists of several steps, the overall execution can be quite long. This is important for the performance of model-checkers, which typically decreases with an increasing model depth. The complexity could be avoided by collapsing several computational steps into a single atomic transition, but this would also make modeling more difficult.

We ran the synthesis and tightening procedures on the problem instead of manually building the TFPG, as we didn’t have a clear expectation on the propagation behavior. Fig. 3 shows part of the synthesized TFPG. Most failure modes have the same edge as “FM_B”, and we don’t show them here for clarity. The following observations can be made based on the synthesized TFPG.

– “FM_B” and the failure modes not shown immediately trigger the health monitor and can thus be recognized and adequately handled by the overall IMU processing (if the flag is used correctly by subsequent functions); the fault doesn't lead to degraded output (but always erroneous output, not shown here).

- Also "FM_D" immediately triggers the health monitor; furthermore, after exactly one cycle, it will also reach the rate estimation (edge from the virtual AND node).
- "FM_C" will affect the rate estimation within one cycle; it might, depending on the fault magnitude, also trigger the monitor, but this is not guaranteed ($t_{max} = +\infty$).
- Finally, "FM_A" immediately results in degraded estimations; furthermore, the edge from "out_degr" to "out_not_valid" shows that, after at least one cycle, also the health monitor may trigger, but again this is not guaranteed.

These results concisely show the different propagations possible in the gyroscope channel processing function, and give formal support to the informal predictions made in FMECA and FDIR design coverage documents. The information shown in the TFPG goes beyond what can be described by FMECA tables and also fault trees, giving detailed insight into how faults evolve over time and affect possible monitored variables.

As for XSAP tool performance, we notice that, on an average desktop computer, synthesis and simplification was completed in 4 seconds by using the BDD-based synthesis engine, whereas tightening takes 43 minutes. The efficiency of the quantitative timing analysis thus clearly needs to be improved to enable faster iterations.

4.2 System-level Propagation

The second case study focused on a time-critical propagation scenario. The analysis scope is very different w.r.t. the first case study. We model the scenario shown in Fig. 4, where a thruster valve is stuck-open, causing a rotation of the spacecraft that might jeopardise the safe zone of the spacecraft attitude. In the case of Solar Orbiter this relates to the requirement to keep the heat-shield pointing towards the Sun. The analysis is focused on one axis only, which is a reasonable constraint. Note that for other missions similar requirements exist, i.e. keeping the high-gain antennas always pointing to Earth in order to maintain ground contact.

The goal of the case study was to formally validate a timing analysis done by hand, which is the basis for estimating the worst-case spacecraft off-pointing. The chosen scenario was well understood from a discrete perspective: it consists of fault occurrence, detection, and several isolation phases. We developed the TFPG shown in Fig. 6; the nodes A1 to A5 are different acceleration phases corresponding to different fault propagation stages, up to the point where fault isolation completely stops the propagation; the node M is a monitor that is used to trigger the fault isolation. The TFPG also contains delay bounds – derived from the documentation – whose precision is in tenths of milliseconds and whose values range from milliseconds to several seconds. For this TFPG we aimed at performing a completeness check, which would confirm the worst-case timing estimates made by engineers.

In Fig. 5 an abstract overview of the developed model is given. The physical state includes the real-valued spacecraft rotation rate, which develops according to an acceleration that is constant in each propagation phase. The software measures the rate via IMU and feeds it into the FDIR logic, which consists of several tasks. These tasks are scheduled together with nominal activities. When an off-nominal rate is detected,

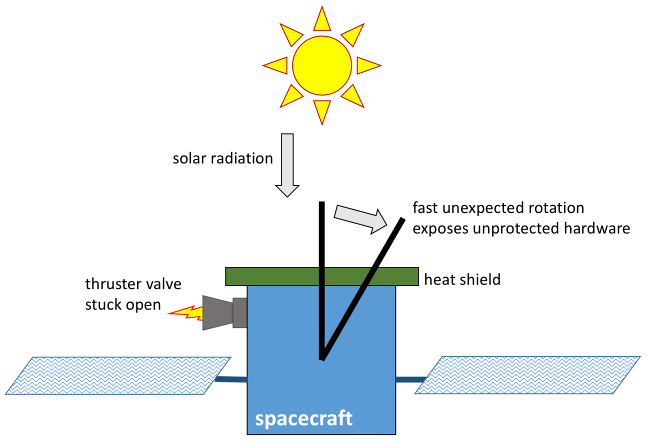

Fig. 4. Failure Scenario

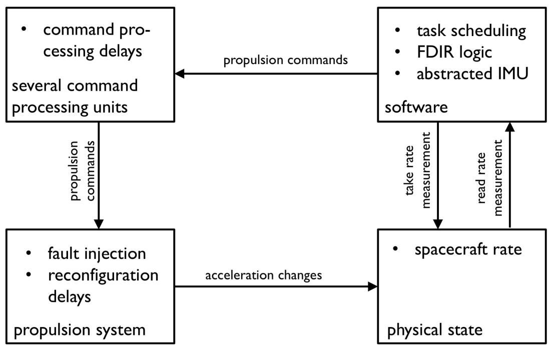

Fig. 5. System Model

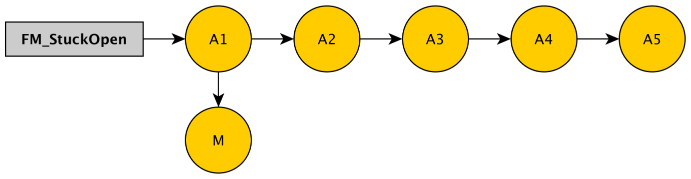

Fig. 6. TFPG Topology (M: monitor; A1-5: off-nominal acceleration phases).

an alarm triggers, and the FDIR sends several commands to the propulsion system and performs several software operations. The propulsion system includes several thruster valves, in one of which the fault can occur. The spacecraft acceleration is set according to the current configuration of the propulsion system. Delays incur in all parts of the model, from task scheduling to data transmission via communication infrastructure and propulsion system reconfiguration. All basic delays and acceleration constants are modeled in the same detail as found in the documentation, and the model is thus representative for the real physical behavior. We also remark that these modeling principles are applicable to many generic spacecraft designs that face similar challenges. Thruster-valve-stuck-open is a well-known failure mode that is critical in all missions where high-pointing accuracy is required. Overall the model consists of 7 Boolean and 1 real input variables, as well as 18 Boolean and 5 real state variables.

A first completeness check was run with bounded model-checking to have a quick feedback on the delay bound estimates. This check showed that they were not fully accurate with respect to the developed model (the completeness check failed), and the t_{max} bound on some segments needed to be increased. In other words this meant that the isolation phase took longer in the model than we expected.

As the automatic tightening procedure in XSAP assumes a TFPG that is complete to begin with we couldn't use it to identify valid over-approximations of the delay bounds. We performed a number of manual iterations based on bounded model-checking to identify time bounds that made the completeness test pass, proceeding in an ad-hoc manner until finding a solution that was precise down to millisecond level. This manual interaction with the model-checker took a couple of hours.

The completeness check on the final TFPG using the IC3 model-checking engine in NUXMV was able to prove the established bounds, with a runtime of 30 minutes on a high-end workstation. This is a relevant result not only from an application perspective, giving feedback on worst-case behavior in a critical scenario, but also from an analysis performance perspective. Indeed they show that it is feasible to analyse very focused but highly accurate propagation problems and prove respective TFPG properties.

The analysis made it possible to raise two issues at the FDIR critical design review of SOLO. The modeling of this scenario and the TFPG analysis showed that the documentation was not clear on whether the complete duration of the last propagation edge in the TFPG was considered in the worst-case analysis for spacecraft off-pointing. The issue was raised during CDR, and worst-case off-pointing estimates as reported in the documentation were confirmed as accurate. Furthermore, the spacecraft can be in several operational modes, and thus in principle the TFPG mode labels should cover all of them. The issue was raised whether mode-switching is of relevance to the propagation scenario. A corner case was identified by the review panel and confirmed not to influence propagation dynamics, and hence to be covered by FDIR.

4.3 Architectural Propagation

The objective of the third case study was to develop a TFPG model as a way to perform failure propagation analysis at various architectural levels, which is usually done with Failure Modes and Effects Analysis. FMEA (or FMECA, when also criticality is considered) is the primary type of failure analysis used in aerospace [7], and serves as a central point of reference in defining and validating FDIR architectures. Going beyond the scope of FMECA, we also wanted to integrate in the TFPG the set of monitors defined in the FDIR design, in order to assess their completeness w.r.t. possible failure propagation paths. We focused on propagations originating from the IMU, reaching subsystem (AOCS) and system (spacecraft) levels, and furthermore focused only on one system mode.

Identification of TFPG nodes The first issue was to decide what information from the FMEA should be imported in the TFPG as failure mode and discrepancy nodes. The natural candidates here are the failure modes and the corresponding failure effects. In cases where the failure mode is associated to a function, as opposed to a hardware or software component, the failure effect is usually identical to it, especially at unit level. In our case this was applicable to all unit-level failure modes, and for them we added a single node to the TFPG, declared as TFPG failure mode node.

At subsystem level the identification was more challenging. Each row in the subsystem table had one associated failure mode, but often more than one failure effect. While the failure modes represented a consolidated list of items (at all levels), the failure effects were less structured. Identifying propagation events thus required interpretation of the informal textual description of what effects a certain failure mode has, in order to extract a consolidated set of propagation events. A specific challenge here was that certain events were mentioned in various parts of different FMEA tables, but the textual description slightly differed, and thus unambiguous integration was not straightforward. Another challenge consisted in the fact that it seemed to be possible to derive distinctive propagation events from both failure mode and failure effect column entries. Whether a failure mode at subsystem level should be modeled in the TFPG as a separate event needed to be assessed by looking at the individual case.

A number of discrepancies were thus derived at subsystem level, declared as `OR` nodes due to the single-fault assumption. We assume here that all failure events at subsystem level can be traced back to events at unit level, and thus don't introduce dedicated TFPG failure mode nodes at subsystem level.

Finally we integrated also standard and functional monitors – SMON and FMON, respectively. The former are simple Boolean expressions over observable state variables, and the latter are Boolean combinations (OR/AND) of those standard monitors. Standard monitors can be seen as fault symptoms, and functional monitors, which are used to trigger recoveries, as slightly more complex diagnosers. Note how the semantics of the monitors matches the notion of TFPG discrepancies. They are conditional on the occurrence of a linked fault, otherwise false alarms would be possible.

Even though in this case study we didn't create a system model to compare the TFPG against, it became clear that a considerable difficulty would be in defining certain TFPG nodes. It seems to be pretty straightforward at the unit level, in our case with clear effects on the IMU hardware. However, at the subsystem level the FMEA uses terms such as "fast", "slow", and "high", without a formal definition being available in the project documentation. For a precise definition, which we would need for TFPG validation or synthesis, additional interaction with engineers would be necessary. Also the validation of the overall FDIR design would benefit from such a formal description.

Identification of TFPG edges The next question to consider was how to connect the nodes. Two approaches to link failure mode and discrepancy nodes were identified based on the FMEA tables: "forward linking" by considering the columns for failure effects at the higher architectural level (chosen for the case study), and "backward linking" via the possible-cause column. By "forward" we mean following the same direction as the propagation, and by "backward" the opposite direction as the propagation. For forward linking we focus on one FMEA table row and look at the prediction of failure effects at the next level. These should ideally match with the failure effects of some failure mode at the higher level. Relating table rows was possible this way but not fully straightforward, due to the less structured content of failure effect table cells and inconsistent use of effect names across levels. We were thus able to match rows of different FMEA levels. However, since the FMEA rows at subsystem level in our use case correspond to more than one node in the TFPG, additional interpretation of the nature of individual events and interaction with engineers were necessary to establish the exact temporal ordering via TFPG edges. This additional knowledge allowed us to create a clearer propagation model compared to how the FMEA tables represent propagation.

Backward linking of FMEA rows can in principle be done through the possible-cause column. However we found this to be conflicting with our choice of forward linking, because in the available tables this backward perspective had an implicit assumption of fault isolation. It indicated failures at a lower level that are not detectable or recoverable at that level, thus excluding all failure modes for which monitors and recoveries were defined there. It seems thus that two different implicit and possibly conflicting propagation models are present in the FMEA tables: one where FDIR fails or is not executed and the failure thus propagates further to the next level (forward linking), and one where FDIR cannot, by design, prevent a propagation (backward linking).

Edges towards monitor discrepancies were established based on "Failure Effect Summary List" (FESL) tables, which associate the standard and functional monitors to individual FMECA rows. Based on this and the precise definitions of the monitors it was possible to establish the edges from failure effect nodes to SMON nodes; edges from SMON to FMON nodes directly followed from the definition of FMON monitors.

We didn't model refined time bounds on the propagations, as this would have required more intense interaction with project engineers. Compared to FMEA the TFPG makes it clear that, without further information, we need to assume for all TFPG edges that propagation can be instantaneous ($t_{min} = 0$) or might never occur at all ($t_{max} = +\infty$), forcing engineers to be explicit about timing aspects. Note how instantaneous propagation is worst-case for propagations towards (unobservable) failure effects, as there is no time to react, and infinite delay is worst-case for propagations towards monitors, as the monitor will never trigger.

The TFPG resulting from our analysis is shown in Fig. 7.

Evaluation of FDIR Design Coverage The general problem that drove the case studies was the validation of FDIR design coverage. Typically based on FESL tables and supporting documentation, engineers try to investigate the following two questions: how are the monitors related to the failure mode row; what happens if FDIR at this level fails to detect and isolate the propagation. The experience with SOLO shows that both questions are not trivial when working with the classical approach.

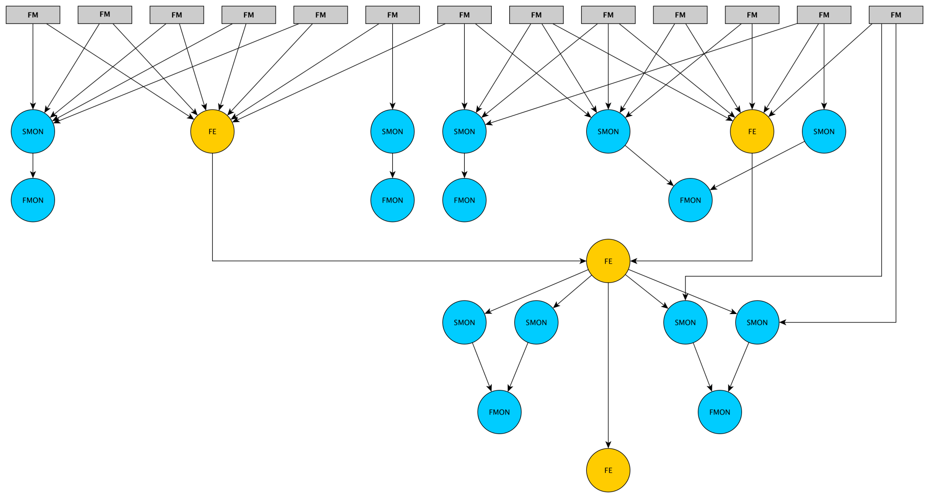

Fig. 7. TFPG of the IMU-to-AOCS case study. FM: failure mode; FE: (unobservable) failure effect; SMON: standard monitor; FMON: functional monitor.

Each failure mode row may contain more than one distinct event which have to be identified by interpreting the textual description in the row cells. Furthermore, if multiple monitors are assigned to the row, then there is no information in the table on which exact event each monitor is associated with. Furthermore, the structure of FMECA tables is driven by the failure modes, whereas the failure effects are typically presented more informally. FDIR monitors however, arguably, relate to the failure effects and not the more abstract concept of failure mode, which makes a comparison of the monitor and the related failure effects challenging.

TFPGs, instead, force the engineer to clearly describe what the events of interest are and how we assume them to be related in a temporal sense among themselves and w.r.t. monitors. From a purely qualitative point-of-view, coverage can thus be assessed by checking what monitors are reachable from every (unobserved) failure event. With the delay bounds TFPGs give also additional information not contained at all in FESL tables, and allow to compare fastest propagation time to the next event against the slowest propagation time towards the monitor (upper detection delay bound).

The second important question is also not straightforward to answer with FESL tables, being directly derived from FMECA tables: What will happen in terms of propagation when the FDIR fails to detect a failure mode or to recover from it, and is a fall-back monitor/response layer in place to capture the propagation? In the case study described here we showed that propagations between different architectural layers can be represented in TFPGs, thus connecting local information into a global propagation model. This then allows to precicely assess, by analyzing the graph, how many and which failure events with associated monitors a propagation has to go through before reaching a point where no further monitors (and recoveries) exist.

Contributions to FDIR Critical Design Review The modeling efforts described in this section raised several questions that were forwarded to the FDIR critical design review panel. A first question regarded the rationale behind the hierarchical placement of two failure modes at subsystem level, as their direct effects influence the whole system. This issue was identified as our modeling goal was to explicitly link various levels. Thus it was not fully clear how many monitor/response layers were in place to prevent propagation to system level. It was clarified during the review that while the failure effects influenced the whole system, the failure modes were placed at subsystem level because detection, isolation, and recovery was limited to that subsystem. This experience showed that TFPGs can help to visually analyze escalation levels and compare them to safety layers implemented in the architecture.

Furthermore, during our analysis we discovered one failure mode at unit-level that, according to the unit FMECA, was not detectable at unit-level. In the design coverage tables however, which group explicit failure modes into more abstract ones, all unit-level failures are detectable at unit-level with the proposed monitors. It was confirmed during the review that in fact detection of the identified failure mode is possible at unit-level, and that the unit FMECA table was incomplete. The documentation was updated accordingly, and the association with subsystem-level monitors was clarified.

Finally, by trying to model the IMU-to-AOCS TFPG we identified an ambiguity on the exact ordering of propagation events; this situation was clarified as well during the review. The problem is that propagation is supposed to connect failure effects at various levels. As shown in the case study, identifying those events and connecting them based on FMECA results and other documentation can be tricky.

5 Conclusion

Typically TFPGs are studied to *implement* fault management or FDIR with a model-based approach, see e.g. [14, 1]. Our experience in developing the case studies shows that TFPGs can be also a valuable *design-time* analysis tool. They provide formal rigor, native support of temporal propagation delays, the ability to unambiguously integrate local propagation patterns into a global model of failure behavior, as well as formally linking propagation effects to monitors used to trigger recoveries. These are clear advantages w.r.t. classical analysis tools such as FMEA and FTA, and the case studies show how these result in a better understanding of system-level failure behavior, which in turn makes deriving precise FDIR requirements or validating an FDIR design easier.

We conclude with a selection of interesting directions for future work in terms of evaluating the use of TFPGs for spacecraft systems engineering. It became clear while developing the case studies that, due to the fact that TFPGs integrate various FDIR-related information, it should also be possible to use them to support several common FDIR tuning problems.

A usual first step of recovery procedures on spacecraft is to disable, upon monitor triggering, a subset of other monitors to avoid execution of other potentially conflicting recoveries. We want to investigate the usability of TFPG models for deciding exactly what monitors need to be considered, which should in principle be possible with a mode-sensitive reachability analysis on the graph. Furthermore, a common principle in

FDIR designs is to execute the recovery of the first monitor that triggers. To guarantee that the desired recovery is triggered for each anticipated failure, filters can be used to delay the effective triggering. It would thus be interesting to see if the timing information in TFPGs can be exploited to derive globally consistent filter values. Finally we want to study the use of TFPGs to tune individual monitor thresholds and assess the impact on the detection delay in relation to other TFPG nodes. Not only should such an analysis show how the delay changes, but it might also show a change in reachability of failure effects due to its impact on the FDIR reactivity, in case the isolation and recovery logic is included as well in the system model.

References

1. Abdelwahed, S., Karsai, G., Mahadevan, N., Ofsthun, S.: Practical Implementation of Diagnosis Systems Using Timed Failure Propagation Graph Models. Instrumentation and Measurement, IEEE Transactions on 58(2), 240–247 (2009)
2. Bittner, B., Bozzano, M., Cavada, R., Cimatti, A., Gario, M., Griggio, A., Mattarei, C., Micheli, A., Zampedri, G.: The xSAP Safety Analysis Platform. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 533–539. Springer (2016)
3. Bittner, B., Bozzano, M., Cimatti, A.: Automated Synthesis of Timed Failure Propagation Graphs. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI 2016). pp. 972–978 (2016)
4. Bittner, B., Bozzano, M., Cimatti, A., De Ferluc, R., Gario, M., Guiotto, A., Yushtein, Y.: An Integrated Process for FDIR Design in Aerospace. In: Model-Based Safety and Assessment, pp. 82–95. Springer (2014)
5. Bittner, B., Bozzano, M., Cimatti, A., Zampedri, G.: Automated Verification and Tightening of Failure Propagation Models. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016) (2016)
6. Bozzano, M., Cimatti, A., Gario, M., Micheli, A.: SMT-based Validation of Timed Failure Propagation Graphs. In: Twenty-ninth AAAI Conference on Artificial Intelligence (2015)
7. ECSS-Q-ST-30-02C: Space product assurance; Failure modes, effects (and criticality) analysis (FMEA/FMECA). Tech. rep. (2009)
8. European Space Agency: Statement of Work: FDIR Development and Verification & Validation Process (2011), Appendix to ESTEC ITT AO/1-6992/11/NL/JK
9. FAME: FAME Project Web Page (2016), `http://es.fbk.eu/projects/fame`
10. McDermott, R., Mikulak, R.J., Beauregard, M.: The basics of FMEA. SteinerBooks (1996)
11. Misra, A.: Senor-based Diagnosis of Dynamical Systems. Ph.D. thesis, Vanderbilt University (1994)
12. Misra, A., Sztipanovits, J., Underbrink, A., Carnes, R., Purves, B.: Diagnosability of Dynamical Systems. In: Third International Workshop on Principles of Diagnosis (1992)
13. Müller, D., Marsden, R.G., Cyr, O.S., Gilbert, H.R., et al.: Solar orbiter. Solar Physics 285(1-2), 25–70 (2013)
14. Ofsthun, S.C., Abdelwahed, S.: Practical Applications of Timed Failure Propagation Graphs for Vehicle Diagnosis. In: Autotestcon, 2007 IEEE. pp. 250–259. IEEE (2007)
15. Ruijters, E., Stoelinga, M.: Fault Tree Analysis: A Survey of the State-of-the-art in Modeling, Analysis and Tools. Computer science review 15, 29–62 (2015)
16. Vesely, W., Goldberg, F., Roberts, N., Haasl, D.: Fault Tree Handbook (NUREG-0492). Washington, DC: Division of Systems and Reliability Research, Office of Nuclear Regulatory Research, US Nuclear Regulatory Commission (1981)