# Chapter 6
# Formal Methods for Aerospace Systems

## Achievements and Challenges

**Marco Bozzano, Harold Bruintjes, Alessandro Cimatti, Joost-Pieter Katoen, Thomas Noll and Stefano Tonetta**

**Abstract** The size and complexity of control software in aerospace systems is rapidly increasing, and this development complicates its validation within the context of the overall spacecraft system. Classical validation methods are both labour intensive and error prone as they rely on manual analysis, review and inspection. Thus there is a growing trend to incorporate the use of automated formal methods. This chapter introduces the ESA-funded COMPASS project, which aims at an integrated system-software co-engineering approach focusing on a coherent set of specification and analysis techniques for evaluation of system-level correctness, safety, dependability and performability of on-board computer-based aerospace systems. Its modelling features and supporting toolset provide a unifying framework for system validation, employing state-of-the-art temporal-logic model checking techniques for infinite-state transition systems, both qualitative and probabilistic, with extensions to fault detection, identification and recovery (FDIR) and safety analysis. We provide an overview of the technology and of the results that have been achieved so far, and address several challenges for future developments. Current efforts of the

This work was supported by the European Space Agency through the COMPASS 3 project (ESTEC contract no. 4000115870).

M. Bozzano · A. Cimatti · S. Tonetta
Fondazione Bruno Kessler, Via Sommarive 18, 38123 Povo, Trento, Italy
e-mail: bozzano@fbk.eu

A. Cimatti
e-mail: cimatti@fbk.eu

S. Tonetta
e-mail: tonettas@fbk.eu

H. Bruintjes · J.-P. Katoen · T. Noll (✉)
Software Modeling and Verification Group, RWTH Aachen University, Ahornstraße 55, 52056 Aachen, Germany
e-mail: noll@cs.rwth-aachen.de

J.-P. Katoen
e-mail: katoen@cs.rwth-aachen.de

H. Bruintjes
e-mail: h.bruintjes@cs.rwth-aachen.de



S. Nakajima et al. (eds.), *Cyber-Physical System Design from an Architecture Analysis Viewpoint*, DOI 10.1007/978-981-10-4436-6_6

project consortium concentrate on improving and advancing both process as well as technology of the COMPASS approach, with the goal of bringing the methods to higher levels of technology readiness.

**Keywords** Safety and dependability analysis · Performance analysis · Model checking · AADL modelling language

## 6.1 Introduction

Verification and validation (V&V) are key processes in the engineering of safety-critical hardware and software systems. Their goal is to check whether the system under construction or its artefacts meet their requirements and its intended functions. The current industry practices for conducting V&V are rather labour intensive [4]. There are severe concerns on scaling these techniques to deal with the ever-growing complexity of systems and in particular of software. The trend is to incorporate the use of formal methods [44, 47, 62]. In particular, *automated verification techniques* are attractive for supporting more rigorous V&V. Formal methods, however, tend to require a high degree of expertise and specialised know-how. These incur substantial investments before their cost and efficiency benefits can be reaped.

To tackle this problem, the European Space Agency (ESA) has initiated an integrated system-software co-engineering approach focusing on a coherent set of specification and analysis techniques for the evaluation of system-level correctness, safety, dependability and performability of on-board computer-based aerospace systems. The work has been and is still being carried out in an ESA-funded project entitled COMPASS, which stands for *COrrectness, Modelling and Performance of AeroSpace Systems* [39].

This chapter gives an overview of the technology and of the results that have been achieved so far, and addresses several challenges for future developments. Current efforts of the project consortium concentrate on improving and advancing *process* as well as *technology* of the COMPASS approach, with the goal of bringing the methods to higher levels of technology readiness.

With regard to technology, several directions to be pursued have been identified, and corresponding methods and implementations are currently under development. Many of them are dealing with failure modelling and analysis. Originally, the COMPASS toolset supports *performability evaluation*: given an AADL model with associated error probabilities, probabilistic model-checking techniques are employed to determine the likelihood of a system failure occurring up to a given deadline. In many cases, however, the probabilities of basic faults are not (exactly) known. It would therefore be worthwhile to consider *parametric* error models, and to automatically compute the maximal tolerable fault probabilities such that the overall model satisfies its performability requirements. A related problem is *model repair*, where one tries to tune the error probabilities of a given model such that a given performability property holds. Moreover, there is increasing demand for verification techniques

that are able to cope with several (interdependent) performance measures, such as reliability and energy consumption. In this setting, *multi-objective model checking* is a promising approach.

Another error modelling concept to be investigated further are *Timed Failure Propagation Graphs* (TFPG), which enable a precise description of how and when failures originating in one part of a system affect other parts – a fundamental feature for successfully designing contingency mechanisms. The latter require the careful design and analysis of FDIR strategies, which in turn are based on the automated synthesis of observability requirements to ensure sufficient *diagnosability* of failure situations. Another safety-related concept is *Dynamic Fault Trees*, an expressive extension of standard Fault Trees that additionally cater for common dependability patterns. In COMPASS, their analysis relies on the extraction of an underlying stochastic model, which is a time-consuming process. This can be improved by reducing the size of this model prior to analysis using graph transformation techniques, and by accelerating the state space generation by leveraging reduction techniques from model checking.

Another direction which is currently under development, is the enhancement of the tool support to *formalise* the requirements into formal properties and to *validate* with formal techniques that the specification is correct and complete. Related to this, the specification of formal properties in terms of component assumptions and guarantees enables *contract-based design*, including the verification of contract-based refinement and contract-based compositional verification of the system behaviour.

The following section sketches a systematic space systems engineering approach as advocated by ESA and related institutions. In the subsequent two sections, we address both the results that have been achieved in the COMPASS project and some of the remaining challenges, and then conclude with a brief summary.

## 6.2 Space Systems Engineering

ESA and related institutions develop and maintain a series of standards for the management, engineering and product assurance in space projects and applications, known as European Cooperation for Space Standardization (ECSS). Among others, Standard ECSS-E-ST-10C [48] specifies the system engineering implementation requirements for space systems and space products development. More concretely, it states that

> Systems engineering is defined as an interdisciplinary approach governing the total technical effort to transform a requirement into a system solution. A system is defined as an integrated set of elements to accomplish a defined objective. These elements include hardware, software, firmware, human resources, information, techniques, facilities services, and other support elements.

Moreover, [48] partitions system engineering into the following activities:

**requirements engineering** which consists of requirement analysis and validation, requirement allocation, and requirement maintenance;

**analysis** which is performed for the purpose of resolving requirements conflicts, decomposing and allocating requirements during functional analysis, assessing system effectiveness (including analysing risk factors); and complementing testing evaluation and providing trade studies for assessing effectiveness, risk, cost and planning;

**design and configuration** which results in a physical architecture, and its complete system functional, physical and software characteristics;

**verification** whose objective is to demonstrate that the deliverables conform to the specified requirements, including qualification and acceptance;

**system engineering integration and control** which ensures the integration of the various engineering disciplines and participants throughout all the project phases.

The following section describes to what extent these activities are supported in our COMPASS approach.

## 6.3 Achievements

The COMPASS project funded by the European Space Agency (ESA) aims at an integrated system-software co-engineering approach focusing on a coherent set of specification and analysis techniques for evaluation of system-level correctness, safety, dependability and performability of on-board computer-based aerospace systems. Its main contributions are a tailored modelling language and a toolset for supporting (semi-)automated validation activities. The modelling language is a dialect of the Architecture Analysis and Design Language (AADL) and enables engineers to specify the system, the software, and their reliability aspects. The COMPASS toolset provides a unifying framework for validation, employing state-of-the-art temporal-logic model checking techniques for infinite-state transition systems, both qualitative and probabilistic, with extensions to fault detection, identification and recovery (FDIR) and safety analysis. Its applicability has been demonstrated in several case studies in the space domain, ranging from thermal regulation and mode management in satellites with associated FDIR strategies to an industrial-size satellite platform. Here we provide a brief overview of our framework. A more comprehensive description is given in [24, 65, 72].

### *6.3.1 The COMPASS Approach*

The COMPASS toolset addresses, in a coherent manner, different aspects that are relevant to the engineering of complex systems, such as co-engineering of hardware

and software, performability and dependability, reliability, availability, maintainability and safety engineering (RAMS). COMPASS offers a multi-disciplinary approach that supports the early design phases by considering systems at the architecture level. Thus it mainly targets the "requirements engineering" and "analysis" functions of system engineering, but also tackles the "design and configuration" and "verification" phases.

More concretely, COMPASS provides a specification language that offers convenient ways to describe nominal hardware and software operation, hybridity, (probabilistic) faults and their propagation, error recovery, and degraded modes of operation. This language is discussed in Sect. 6.3.2 in greater detail. It is equipped with a formal semantics that opens up the possibility to apply a wealth of formal methods for various kinds of verification and validation activities. Most of these are based on formal requirements as introduced in Sect. 6.3.3. V&V is supported by an integrated toolset, described in Sect. 6.3.4, that covers the following functionalities according to the ECSS standards.

**Requirements Validation** [48] In order to ensure the quality of requirements, they can be validated independently of the system. This includes both property consistency (i.e., checking that requirements do not exclude each other), property assertion (i.e., checking whether an assertion is a logical consequence of the requirements), and property possibility (i.e., checking whether a possibility is logically compatible with the requirements). Altogether these features allow the designer to explore the strictness and adequacy of the requirements. Expected benefits of this approach include traceability of the requirements and easier sharing between different actors involved in system design and safety assessment. Furthermore, high-quality requirements facilitate incremental system development and assessment, reuse and design change, and they can be useful for product certification.

**Functional Verification** [48] Analysing operational correctness is the first step to be performed during the system development lifecycle. It consists in verifying that the system will operate correctly with respect to a set of functional requirements, under the hypothesis of nominal conditions, that is, when software and hardware components are assumed to be fault-free. This can be accomplished by both simulation and exhaustive model checking techniques.

**Safety and Dependability Analysis** [49, 51–53] Analysing system safety and dependability is a fundamental step that is performed in parallel with system design and verification of functional correctness. The goal is to investigate the behaviour of a system in degraded conditions (that is, when some parts of the system are not working properly, due to malfunctions) and to ensure that the system meets the safety requirements that are required for its deployment and use.

**Performability Analysis** [50] To guarantee the required system performance in the presence of faults, integrated hardware and software models can be evaluated with respect to their performance behaviour in degraded modes of operation. In line with the approach for the functional correctness, again model checking techniques are employed for assessing this type of requirements.

**Fault Detection, Identification and Recovery (FDIR) Analysis** [49] System models can include a formal description of both the fault detection and identification sub-systems, and the recovery actions to be taken. Based on these models, tool facilities are provided to analyse the operational effectiveness of FDIR measures, and to investigate the observability requirements that make the system diagnosable.

In summary, the overall process of analysing system specifications in the COMPASS framework involves the following steps:

1. System specifications (describing the nominal and, if applicable, the error behaviour) are entered using a text editor, and loaded into the toolset (cf. Sect. 6.3.2).
2. Some of the subsequent analyses require writing properties. COMPASS offers several ways to specify such properties (cf. Sect. 6.3.3).
3. To interactively explore the dynamic behaviour of the system, the model simulation feature of the toolset can be employed (cf. Sect. 6.3.4).
4. Finally, depending on the type of the system, a plethora of analyses can be applied (cf. Sect. 6.3.4).

### *6.3.2 System Modelling*

AADL [84] is an industry standard for modelling safety-critical system architectures, which is developed and governed by SAE. This language provides a cohesive and uniform approach for modelling heterogeneous systems, consisting of software (e.g., processes and threads) and hardware (e.g., processors and buses) components, and their interactions. It enables analysis of system designs prior to implementation and supports a model-based and model-driven development approach throughout the system life cycle.

Our dialect of AADL was designed to meet the needs of the European space industry. The original language is mainly focused on the architectural organisation of a system under nominal and degraded modes of operation. The nominal modes indicate that the system is operating normally, whereas degraded modes typically signify that the system's functions are (partially) impaired due to some anomaly. Our goal was to extend AADL's scope on defining the architecture of a system by also allowing to analyse its dynamic behaviour, namely both its nominal and degraded modes of operation and their interweaving.

In particular, quantitative aspects such as the timing of operations and the likelihood of faults should be covered. To this end, we built on a core fragment of AADL Version 1 [83] and extended it, essentially by supporting the following features:

- Modelling both the system's *nominal* and *faulty* behaviour. To this aim, primitives are provided to describe software and hardware faults, error propagation (i.e., turning fault occurrences into failure events), sporadic (transient) and permanent faults, and degraded modes of operation (by mapping failures from architectural to service level).

- Modelling (*partial*) *observability* and the associated *observability requirements*. These notions are essential to analyse the effectiveness of fault management systems. These subsystems, being part of the overall system, monitor it, identifying when a fault has occurred, pinpointing the type of fault and its location, and finally recovering from it by, for example, switching to a backup system configuration.
- Specifying *timed* and *hybrid* behaviour. In particular, to analyse physical systems with non-discrete behaviour, such as mechanics and hydraulics, the modelling language supports continuous real-valued variables with (linear) time-dependent dynamics.
- Modelling *probabilistic* aspects, such as random faults and repairs, that are subject to stochastic timing.

A complete system specification consists of three parts, namely a description of the nominal behaviour, a description of the error behaviour, and a fault injection specification that describes how the error behaviour influences the nominal behaviour. This separation approach is different from the one taken in AADL and its Error Model Annex [81], which interacts through an explicit specification of mangling error and nominal events. In contrast, our dialect provides an automated mechanism (called *model extension*) that enables engineers to keep the nominal model completely separate from the error model. A comprehensive presentation of our specification language and its formal semantics is given in [25].

### *6.3.3 Requirements Specification*

An important aspect of V&V of requirements is the consistent and complete specification of formal properties associated with the requirements. As the (correct) specification of such properties requires a significant amount of technical expertise, the COMPASS toolset has striven to alleviate this burden from its users as much as possible.

The approach initially taken by COMPASS was to allow the user to specify properties by means of *patterns* [5, 46]. These provide a structured way of generating a formal property given a template with placeholders, where the user provides basic propositions (statements about the current state of the system) for each of these placeholders. The use of these patterns relies on the fact that the requirements themselves often use recurring shapes. For example, one pattern describes the *absence* of particular behaviour in the system, e.g., reaching a state of critical failure.

Various logics are supported by this approach, in particular qualitative, timed and probabilistic logics, such as LTL/CTL, MITL and CSL respectively. A formal property expressed by a pattern is converted into the appropriate logic depending on the pattern used, and given to a model checker for analysis. An appropriate model checker in the toolset will be provided the input model and one or more properties, and checks whether the property holds, thus providing the formal verification of the requirement associated with it.

Recently, support has been added to COMPASS for the *Catalogue of System and Software Properties* (CSSP) in the CATSY project [18]. The CSSP defines a set of design attributes which are used to automatically derive formal properties. A requirements taxonomy has been set up, with a focus on the space engineering domain, to allow various project requirements to be classified (e.g., requirements related to monitoring, protocols or availability). For each of these classes, associated design attributes have been identified. Such design attributes may refer to the presence of elements in the model (e.g., redundant components or mode transitions) or to properties associated with such elements (e.g., timing of events, or reactions to events). The latter are collected in the CSSP. This way, for appropriate requirement classes, a user can simply specify the value of the associated properties. A formal property is generated automatically for such model properties, allowing for the verification of the corresponding requirement.

Moreover, still within the CATSY project, the specification language has been enriched with the possibility to specify properties on components directly in temporal logics, thus without the help and limitation of the pattern-based approach. Finally, the properties attached to components (either specified directly in temporal logic or by means of patterns or the CSSP) can be structured into *contracts* [38], i.e., pairs of assumptions and guarantees, to enable the verification of contract-based refinement and contract-based compositional verification and safety analysis.

### *6.3.4 COMPASS Toolset*

The COMPASS toolset is the result of a significant implementation effort carried out by the COMPASS Consortium. The GUI and most subcomponents are implemented in Python, using the PyGTK library. Pre-existing components, such as the NuSMV and MRMC model checkers, are mostly written in C. Overall, the core of the toolset consists of about 100,000 lines of Python code. Figure 6.1 shows the functionality of the toolset. Its main features are introduced in the COMPASS Tutorial [41]. It is complemented by the COMPASS User Manual [40], which can be consulted for a more systematic reference.

COMPASS takes as input one or more AADL models, and a set of properties. As pointed out before, the latter are provided in the form of generic properties or instantiated property patterns, which are templates containing placeholders that have to be filled in by the user. The COMPASS toolset provides templates for the most frequently used patterns, that ease property specifications by non-experts through hiding the details of the underlying temporal logic. The tool generates several outputs, such as traces, Fault Trees and FMEA tables, diagnosability and performability measures.

The toolset builds upon the following main components. NuSMV (New Symbolic Model Verifier, [34, 74]) is a symbolic model checker that supports state-of-the-art verification techniques such as BDD-based and SAT-based verification for CTL and LTL [8]. nuXmv [32] is an extension of NuSMV for the SMT-based verification of

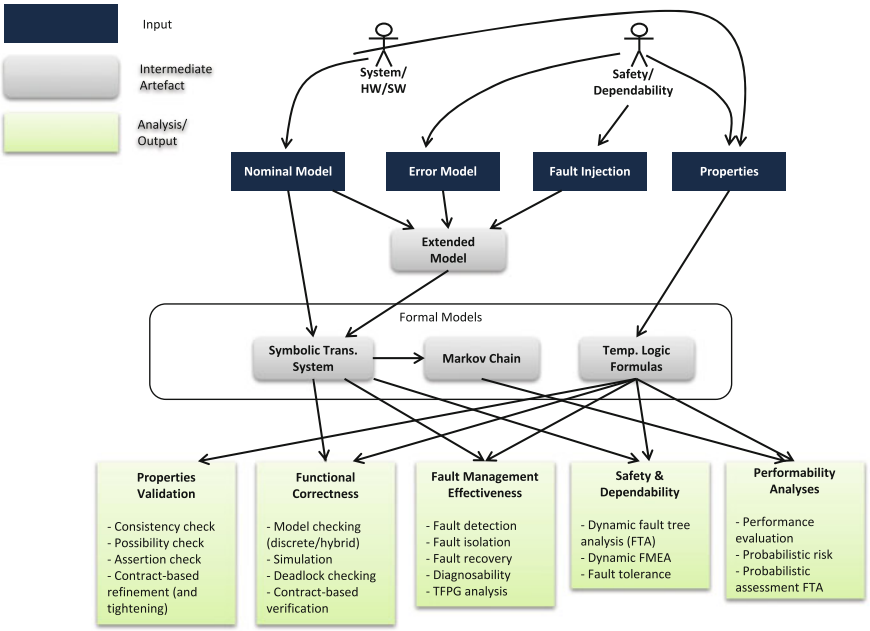


**Fig. 6.1** Functional view of the COMPASS platform

infinite-state systems. MRMC (Markov Reward Model Checker, [67, 71]) is a probabilistic model checker that enables the analysis of discrete-time and continuous-time Markov reward models. Specifications are written in PCTL (Probabilistic Computation Tree Logic) and CSL (Continuous Stochastic Logic [6], a probabilistic real-time version of CTL). SigRef [88] is used to minimise, amongst others, Interactive Markov Chains (IMC; [61]) based on various notions of bisimulation. It is a symbolic tool using multi-terminal BDD representations of IMCs and applies signature-based minimisation algorithms. xSAP [13] is a tool that supports model-based safety analysis including Fault-Tree Analysis, FMEA, and diagnosability. OCRA [36] takes in input a system architecture specification and allows to perform contract-based validation and verification.

The tool also supports a graphical notation of our AADL dialect, which is derived from the graphical notation of AADL [82]. We developed a graphical drawing editor enabling engineers to construct models visually using the adopted graphical notation. This editor is called the *COMPASS Graphical Modeller* and is part of the COMPASS toolset. Figure 6.2 shows the main window of the COMPASS toolset after loading a system model and performing a fault injection.

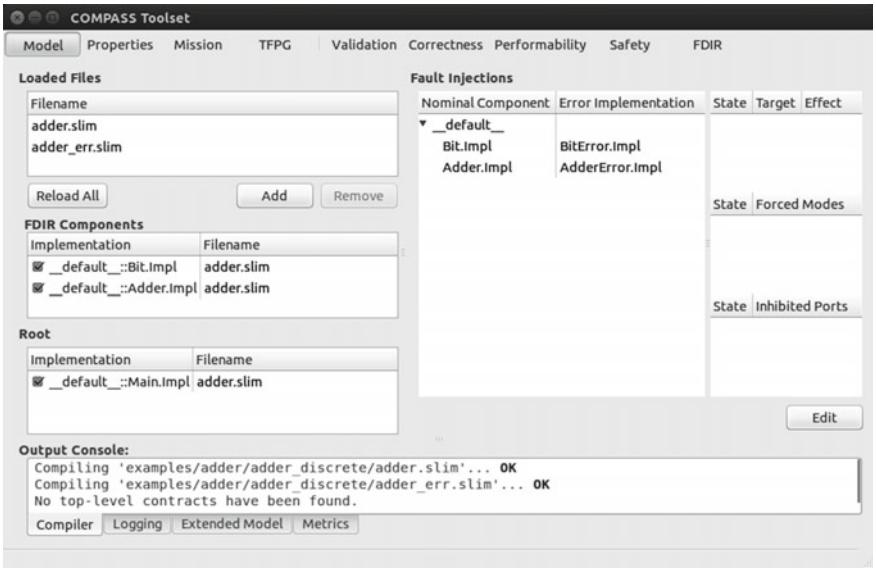


**Fig. 6.2** Main window of the COMPASS toolset

#### 6.3.4.1 Properties Validation

Figure 6.3 gives an example of a property specification by means of patterns. Before analysing the correctness of the behavioural model against the specified properties, the properties themselves can be validated to search for errors in the requirements or in their formalisation. COMPASS supports this activity of properties assurance [79] by allowing the user to specify and check *property validation problems*. These can be a simple check of consistency (i.e., logical satisfiability) of a set of properties or can consist of specifying a new property to be consistent with or entailed by a set of existing properties. In case of inconsistency or failed entailment, an execution trace is generated as a witness of the result. In case of proved inconsistency or entailment, a minimal subset of properties that are sufficient for the proof can be extracted.

When properties are structured into contracts, COMPASS supports the verification of their refinement as described in [38]. In *contract-based design*, the assumptions of a component are properties to be satisfied by the component environment, while the guarantees are properties to be satisfied by the implementation when the assumptions hold. A correct *contract refinement* ensures that any correct implementation of the subcomponents form a correct implementation of the composite component, and, together with an environment satisfying the assumptions, form a correct environment for each subcomponent. This is verified by generating and proving a set of proof obligations, which are validity problems for the underlying temporal logic. For every refined contract, there is a proof obligation to ensure that the guarantee of the composite component is entailed by the conjunction of the assumption of the

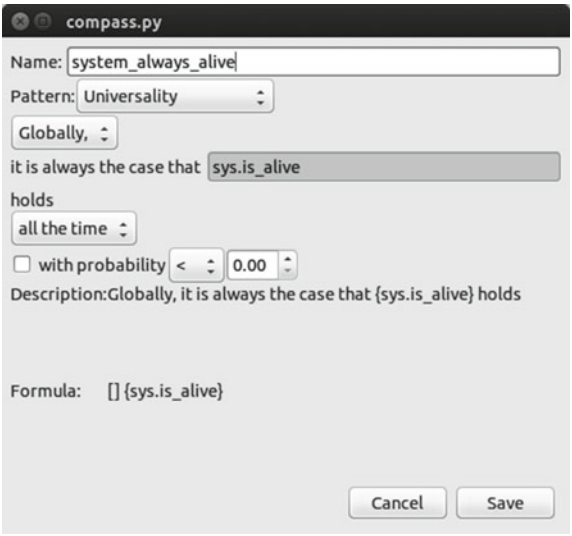


**Fig. 6.3** The property editor

composite component and the contracts of the subcomponents, and similarly for each assumption of each subcomponent.

If the contract refinement is not correct, the tool provides an execution trace for every invalid proof obligation. In case the refinement is correct, the tool can provide some feedback by presenting viable *tightenings* of the contract refinement, i.e. stronger/weaker versions of the assumptions/guarantees that still yield a correct contract refinement, as described in [35].

#### 6.3.4.2 Functional Correctness

COMPASS supports random and guided *model-based simulation* of AADL models. Guided simulation can be performed by choosing either the next transition to be taken, or a target value for one or more variables. The generated traces can be inspected using a trace manager that displays the values of the model variables of interest (filtering is possible) for each step.

*Property verification* is based on model checking [8], an automated technique that verifies whether a property expressed in temporal logic holds for a given model. Symbolic techniques [10, 11, 60] are used to tackle the problem of state space explosion. COMPASS relies on the NuSMV [34, 74] and nuXmv [32, 75] model checkers, which support both BDD-based and SAT-based verification for finite-state systems, and SMT-based verification techniques for timed and hybrid systems, based on the MathSAT solver [20, 69]. On refutation of a property, a counterexample is generated, showing an execution trace of the model violating the property. An example of this is shown in Fig. 6.4. It is also possible to run *deadlock checking*, in order to pinpoint deadlocks (i.e., states with no outgoing transitions) in the model.

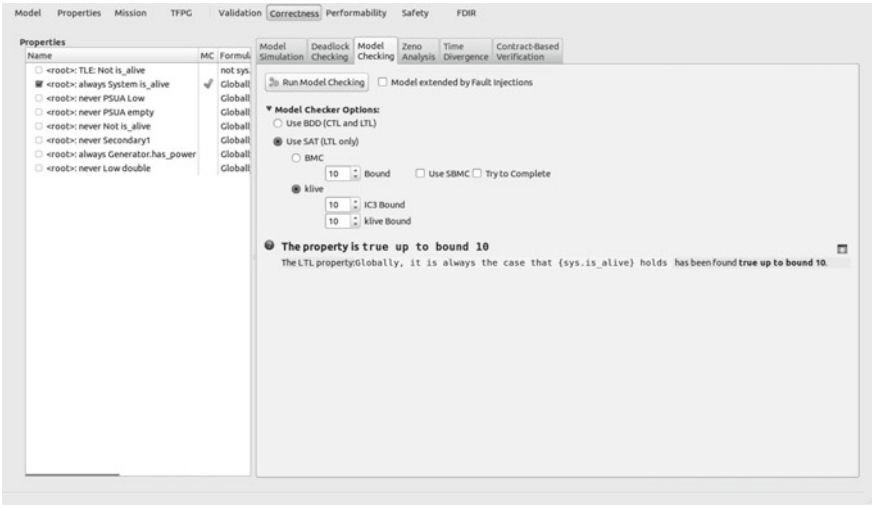


**Fig. 6.4** A model-checking counterexample

The verification of properties can be enhanced by contract-based specification to perform it in a compositional way. In this case, COMPASS interacts with OCRA [36], first, to check that the contract refinement is correct and, second, to individually verify each atomic component with respect to its local contract. These checks ensure the correctness of the overall system by compositional reasoning.

#### 6.3.4.3 Safety and Dependability Assessment

COMPASS implements model-based safety assessment techniques, based on symbolic model checking [26, 28], and supports traditional techniques such as *Failure Mode and Effects Analysis* (FMEA; [49]) and *Fault Tree Analysis* (FTA; [52]). FMEA is an inductive technique that starts by identifying a set of (combinations of) failure modes and, using forward reasoning, assesses their impact on a set of system properties. The results are summarised in an *FMEA table*. It is also possible to generate *dynamic* FMEA tables, i.e., to enforce an order of occurrence between failure modes. FTA is a deductive technique, which, given a *top-level event* (TLE), i.e., the specification of an undesired condition, constructs all possible chains of basic faults that contribute to its occurrence. Pictorially, these chains are organised in a *Fault Tree* with a two-layer logical structure, corresponding to the disjunction of its minimal cut sets (MCSs; [28]), where each MCS is a conjunction of basic faults. COMPASS also supports the generation of (a subset of) *Dynamic Fault Trees* [45], where ordering constraints between basic faults are represented using priority AND (PAND) gates. Figure 6.5 depicts a Fault Tree.

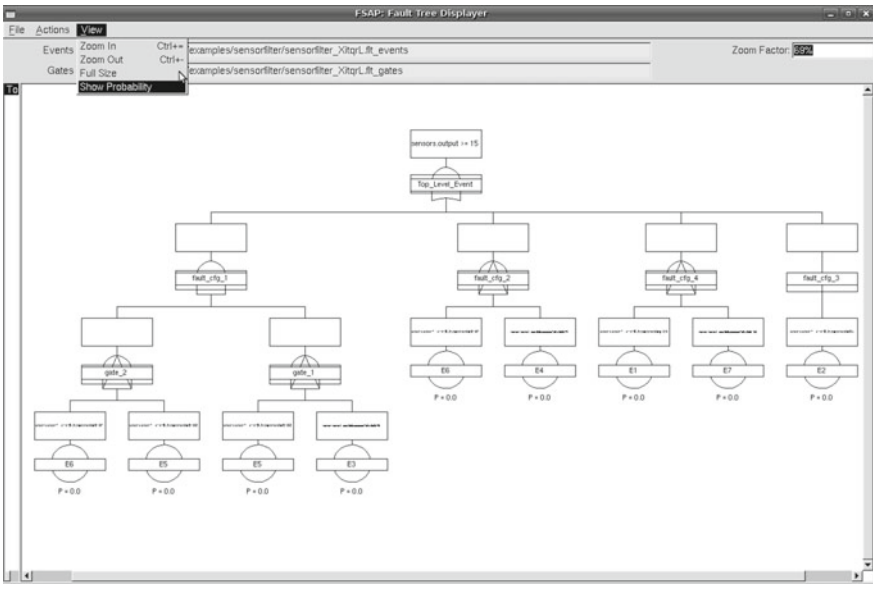


**Fig. 6.5** A generated fault tree

It is also possible to exploit a contract-based specification to obtain a fault tree that follows the hierarchical decomposition of the system [27]. In this case, COMPASS interacts with OCRA to compute a fault tree where each intermediate event represents the failure to satisfy either the guarantee or the assumption of a contract. The fault tree represents the dependency between such failure and the failures of other components. For example, the failure of a composite component can be caused by the combined failure of two of its subcomponents or by the failure of its assumption, which in turn can be caused by the failure of other components. Compared to the "monolithic" FTA described above, the contract-based FTA is more pessimistic with regard to the identification of possible system failures because it follows the conservative approximation given by the contract-based refinement. The resulting fault tree is however often more intuitive because it uses intermediate events corresponding to the components in the system architecture.

#### 6.3.4.4 Diagnosability and FDIR Analysis

The COMPASS toolset supports diagnosability and FDIR (Fault Detection, Isolation and Recovery) effectiveness analysis. These analyses work under the hypothesis of *partial observability*. Variables and ports in our AADL dialect can be declared to be observable.

*Diagnosability analysis* investigates the possibility for an ideal diagnosis system to infer accurate and sufficient run-time information on the behaviour of the observed

system. The COMPASS toolset follows the approach described in [37], where the violation of a diagnosability condition is reduced to the search of *critical pairs* in the so-called *twin plant* model, i.e., pairs of execution traces that are observationally equivalent but hide conditions that should be distinguished. Figure 6.6 shows such a pair of traces.

*FDIR effectiveness analysis* refers to a set of analyses carried out on an existing fault management subsystem. *Fault detection* is concerned with detecting whether a given system is malfunctioning, namely searching for observable signals such that every occurrence of the fault will eventually trigger them. *Fault isolation analysis* aims at identifying the specific cause of malfunctioning. It generates a Fault Tree that contains the minimal explanations that are compatible with the observable being triggered. Finally, *fault recovery analysis* is used to check whether a user-specified recoverability property holds.

#### 6.3.4.5 Timed Failure Propagation Graphs

COMPASS supports *Timed Failure Propagation Graphs* (TFPGs; [1, 70, 76]) as a means to model and analyse how failures originating in one part of a system affect other parts. Traditionally, TFPGs can be used for both diagnosis and prognosis. TFPGs describe the occurrence of failures and the temporal interrelationships between failures and their direct and indirect effects. They constitute a very rich formalism that can express Boolean combinations of basic failures, intermediate consequences, and transitions across them, labelled with propagation times and possibly dependent on the system's operational modes. TFPGs are increasingly used for

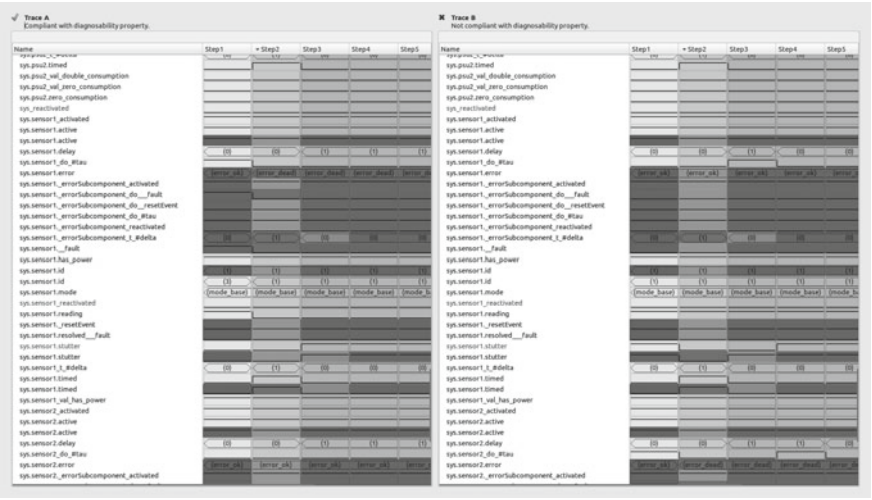


**Fig. 6.6** Diagnosability counterexample

the design of autonomous systems, in particular for the design of FDIR procedures. Compared to other techniques such as FTA and FMEA, TFPGs have substantial advantages. They present a more comprehensive and integrated picture than Fault Trees, as they focus on propagation paths in response to individual feared events. Moreover, in comparison to FMEA tables they provide additional and more precise information, such as timing information and AND/OR correlations between propagation causes and effects.

As shown in Fig. 6.7, COMPASS enables the modelling and analysis of TFPGs. The available analyses include *behavioural validation*, that is, verification that a TFPG is a complete representation of failure propagation with respect to a given system model, and *effectiveness validation*, that is, verification that a TFPG is a suitable model for diagnosis, i.e., contains sufficient information to carry out diagnosis, discriminating between different possible causes. Finally, COMPASS supports the automatic synthesis of a TFPG, given a set of nodes and a system model.

#### 6.3.4.6 Performability Analysis

We use *probabilistic model checking* techniques [7, 8] for analysing a model with respect to its performance. The COMPASS toolset in particular supports performance properties expressed by the probabilistic pattern system presented in [5]. It allows for the formal specification of steady-state, transient probabilities, timed reachability probabilities and more intricate performance measures such as combinations thereof. An example of a typical performance parameter is "the probability that the first battery dies within 100 h" or "the probability that both batteries die within the mission

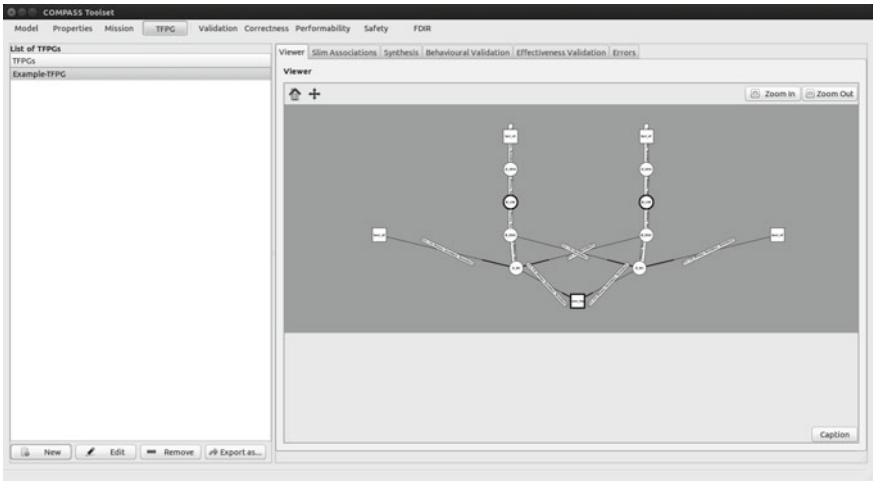


**Fig. 6.7** Example of a TFPG

duration". These properties have a direct mapping to Continuous Stochastic Logic (CSL; [6]) and are input to the underlying probabilistic model checker.

The probabilistic model checker furthermore requires a Markov model as input. This is obtained from the integrated nominal and error model through several steps. First, the extended model's reachable state space is generated through an exhaustive symbolic exploration. Second, the probabilistic rates as specified in the error models are interwoven through the state space by replacing the transition label with the associated probabilistic rate. The resulting state space is a symbolic representation of an Interactive Markov Chain (IMC), i.e., a Continuous-Time Markov Chain (CTMC) that may exhibit non-determinism [61]. This IMC is passed through the third phase, in which its size is reduced using weak bisimulation minimisation [43, 86]. In the final phase, CSL formulae are extracted from the performance requirements, and are fed together with the reduced IMC to a probabilistic model checker, to compute the desired probabilities. If the reduced IMC is a proper CTMC, the MRMC model checker [67] is used for this purpose, otherwise the IMCA model checker [58] is employed. As can be seen in Fig. 6.8, the result is a graph showing the cumulative distribution function over the time horizon specified in the performance requirement. Similar techniques are also used for *Fault Tree evaluation*, i.e., for computing the probability of the top-level event in Dynamic Fault Trees [19].

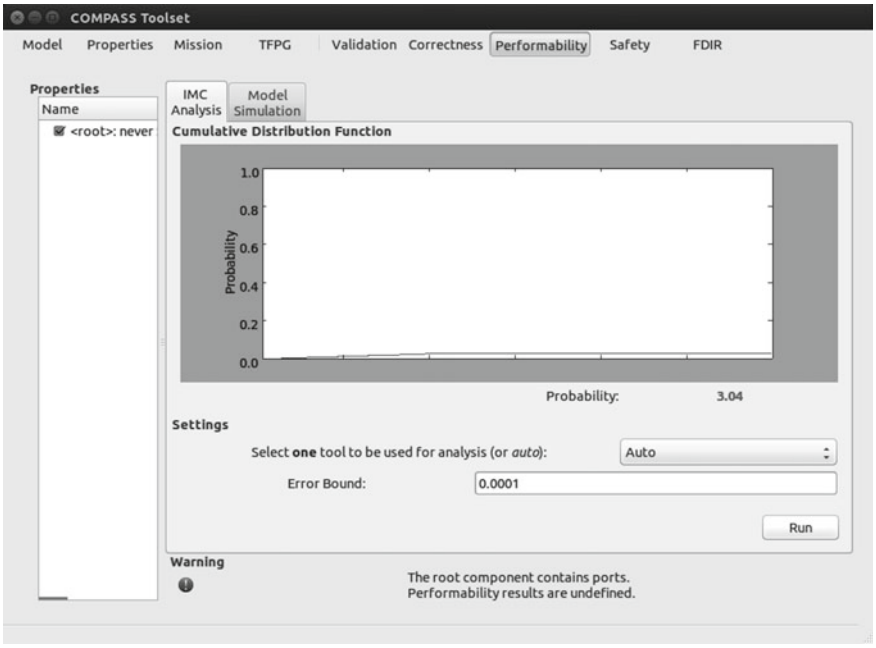


**Fig. 6.8** Performing performability analysis

### 6.3.5 Case Studies

The COMPASS methodology has been progressively assessed by several industrial case studies, steadily increasing in size and scope. Table 6.1 summarises the results, respectively giving appropriate references, the number of components of the system model ("#C"), the main aspect to be investigated, and the major technological challenges that have been solved and those that were left open (and will be addressed in Sect. 6.4).

Thales Alenia Space conducted the first evaluation. They developed two case studies of their satellite subsystems, respectively dealing with mode management and thermal regulation, and analysed them using the COMPASS toolset [65]. These subsystem case studies demonstrated the potential of understanding the subtle interactions between the system, software, and the fault management system. They furthermore raised follow-up questions: how would models with a greater level of detail be handled? In which phases of the systems engineering life cycle is the COMPASS methodology particularly suitable?

To address these issues, ESA conducted a laboratory project to model a full satellite platform using the COMPASS methodology. This was performed at phase B of the space systems engineering life cycle, the preliminary system design [24, 54]. A subsequent laboratory project was initiated afterwards to model a full satellite platform at phase C, the detailed system design. Here, special focus was put on diagnosability analysis, which in the phase B pilot was deemed intractable. This analysis increasingly gains importance in the engineering life-cycle as fault management designs become more involved to meet mission demands. Our experiences indicate a clear need for enhanced diagnosability analysis algorithms that also account for delayed diagnostic means. The outcomes of this study are discussed in detail in [24].

Moreover, [31] presents a case study performed together with Airbus Defense and Space, which was carried out to demonstrate the applicability of stochastic model checking (Monte Carlo methods) to analyse timed reachability properties of a simplified launcher system. The evaluation revealed the need to support more expressive kinds of stochastic logics.

Recently, a case study on the application of TFPGs for the on-going Solar Orbiter project [85] was performed internally at ESA. It shows the feasibility of using TFPG-

**Table 6.1** Overview of case studies

| Case study | #C | Aspect | Solved | Open |
|---|---|---|---|---|
| Mode mgmt. [65] | 3 | Fault mgmt. | Scalab. of analysis | — |
| Thermal reg. [65] | 12 | Hybrid behav. | Zeno paths | — |
| Platform B [24, 54] | 86 | FDIR | Fairness | — |
| Platform C [24] | 246 | Diagnosab. | Effic. diagnosab. anal. | Delayed diagnosis |
| Launcher [31] | 37 | Prob. reachab. | Effic. performab. anal. | Expressivity of logics |
| Solar Orbiter [12] | 15–39 | Failure propag. | TFPG analysis | Fault recov. synthesis |

based analyses to study time-critical failure propagation at a unit to subsystem level. When using focused modelling, also the analysis of timed failure propagation in detailed physical models is feasible. In general, TFPGs have been found to be a promising technology to formally integrate various key aspects of FDIR design, including discrete failure propagations across all levels of a system, time bounds on the delays, mode constraints, and monitors. This is very important, because informal analysis usually done with FMEA makes it difficult to demonstrate completeness and timing properties of the proposed design. Indeed, by application of TFPGs in the case study one case of a propagation link missing in the FMEA tables has been identified.

## 6.4 Challenges

With regard to technological challenges, several directions to be pursued have been identified, and corresponding methods and implementations are currently under development.

### *6.4.1 Formal Validation of Probabilistic Properties*

As explained in Sect. 6.3.3, system requirements are formalised by temporal logics. The toolset described in Sect. 6.3.4 supports the validation of such properties by means of satisfiability/validity checking, i.e., the problem of deciding whether a given formula is satisfied by at least one or, dually, by every system model. This is currently supported for qualitative logics like LTL and CTL, which only allow to describe the order of system actions but cannot express quantitative properties such as timing or probabilities. These logics possess the so-called finite model property, and the complexity of checking satisfiability has been explored for various fragments.

However, the satisfiability problem for probabilistic versions of CTL such as PCTL and CSL, which are employed to express quantitative properties of system models, is almost unexplored [33]. These logics are quite popular in the field of probabilistic verification as their model-checking problem is known to be decidable. When it comes to satisfiability, however, the analysis turns out to be a much more difficult endeavour. In fact, this is a long-standing open problem for PCTL. Results so far are restricted to logical fragments such as qualitative PCTL [29], or are obtained by considering variations of the satisfiability problem. One of the most recent results is given in [33], where the satisfiability problem for a bounded fragment of PCTL is shown to be decidable.

Therefore, there is strong demand for identifying richer fragments of PCTL and for studying their satisfiability, complexity, and finite/rational model properties. A promising direction is to start from the characterisation of safety and liveness fragments of this logic [66].

### 6.4.2 Contract-Based Fault Injection

As discussed in Sect. 6.3.4, Fault Tree Analysis can be performed either by means of a more traditional model-based approach, which computes the minimal cut set for a top-level event, or by means of contract-based safety analysis, which produces a hierarchical Fault Tree that follows the specified contract refinement (and thus the architectural decomposition). The two analyses are currently disconnected: while the model-based safety analysis exploits the error model specification to automatically inject faulty behaviours into the nominal model, the contract-based approach identifies a failure by the fact that the component implementation violates a guarantee or that the component environment does not satisfy an assumption. Thus, in the contract-based approach, in case of failure, any behaviour is possible. This may result in Fault Trees that describe combinations of failures which can never occur in the real system. An interesting research direction is to find an effective way to inject the faults in the contract specification in order to have degraded assumptions and guarantees in case of failures.

### 6.4.3 FDIR Design and Diagnosability

The area of diagnosability and fault detection, identification [and recovery] (FDI[R]) design is particularly challenging. Recent work [22, 23] has addressed the extension of diagnosis and FDI to incorporate the notion of *delay*, and to address cases where diagnosability cannot always be guaranteed for all system executions.

The diagnosis delay characterises situations where diagnosis requires a time delay in order to be carried out. The notion of *alarm condition* formalises the relation between the condition to be diagnosed (e.g., presence or absence of a fault, isolation between different faults) and the raising of an alarm by the diagnoser; an alarm condition may specify diagnosis with an exact delay (after exactly a given time), a bounded delay (within a given time) and finite delay (eventually). Moreover, the notion of *trace diagnosability* formalises cases where diagnosability cannot be guaranteed globally, but only locally, on a subset of traces, and the notion of *maximality* formalises the capability of a diagnoser to raise the alarm as soon as possible and as long as possible. Finally, FDI effectiveness properties state the correctness and completeness of an FDI design with respect to the FDI requirements. In [23], all of these properties of an FDI design can be specified using a general framework and language based on temporal epistemic logic. Verification can be performed using an epistemic model checker. Alternatively, the diagnosability check can be reduced to standard temporal logic model checking based on the twin-plant approach described in Sect. 6.3.4. The original version of the latter was introduced in [37] and is being employed in the current COMPASS toolset, while [22, 57] shows how to extend it to deal with epistemic logic. In addition, an algorithm for the automatic synthesis of correct-by-construction FDI components is available [23].

Another interesting research area concerns the analysis of observability requirements for diagnosis, and the synthesis of a set of observables that are sufficient to ensure diagnosability [16]. It is possible to rank configurations of observables based on cost, minimality, and diagnosability delay, thus helping designers in finding the most appropriate configuration.

The automatic *synthesis* of FDIR components has been considered in two projects related to COMPASS, namely AUTOGEF [2] and FAME [15], also funded by ESA. The problem is cast in the frame of discrete event systems and finite delay diagnosis, and is tackled by synthesizing the fault detection and fault recovery components separately, with the idea that fault recovery implements a plan (recovery strategy) to respond to the alarms generated by the fault detection component. FAME addresses the problem of FDIR synthesis for continuous-time systems, where the diagnoser communicates with the plant by sampling the values of the sensors at periodic time intervals. Another outcome of the FAME project is the definition of a general process for FDIR design that spans the different phases of system development (definition of mission and FDIR requirements, safety assessment, FDIR design and verification). This process aims at enabling a consistent and timely FDIR conception, development, verification and validation, overcoming several shortcomings of existing practices. FDIR synthesis, along with other functionality described in this section, is under consideration for inclusion in the COMPASS toolset.

### *6.4.4 Timed Failure Propagation Graphs*

Recent work has focused on techniques for validating TFPG models. In particular, [21] studies several validation problems using advanced techniques based on satisfiability modulo theory, namely possibility and necessity, refinement and diagnosability. Moreover, [17] addresses both the completeness of a TFPG with respect to a system model, and the problem of *tightness* of TFPG edges, that is, the possibility that certain parameters, specifically time bounds, of the TFPG can be reduced without breaking its completeness. Finally, the problem of automatic synthesis of a TFPG is thoroughly investigated in [14]. Automatic tightening of the TFPG nodes, coupled with the synthesis of the graph, may be used to automatically produce a complete and tight TFPG from a system model, given the definition of the TFPG nodes.

### *6.4.5 Parametric Error Models*

Originally, the COMPASS toolset supports performability evaluation (cf. Sect. 6.3.4): given an AADL model with associated error probabilities, the likelihood of a system failure occurring up to a given deadline is determined. The underlying technique is probabilistic model-checking. In many cases, however, the probabilities of basic faults are not known, or at best can be estimated by lower and upper bounds. It

would therefore be worthwhile to consider parametric error models, in which the probabilities of faults are (partially) left open.

*Parameter synthesis* focuses on automatically computing the maximal tolerable parameter values such that the resulting model satisfies its performability requirements. Although this problem is inherently harder than (probabilistic) model checking, first results indicate that for a limited number of parameters, solutions are feasible and scalable [42, 63]. They would allow to derive quality requirements for electronic [80] or mechanical parts [73] or software components [3] of a system to be developed.

A related problem is *model repair*, where one tries to tune the error probabilities of a given model such that the resulting model satisfies a given performability requirement. Current approaches only consider changes of the transition probabilities, whereas modifications of the underlying topological structure are not considered. Different methods exist, such as global repair [9] and the more recent technique to perform local repair operations in an iterative fashion [77].

### 6.4.6 Dynamic Fault Trees

Fault Tree Analysis (cf. Sect. 6.3.4) is a widespread industry standard for assessing system reliability [52]. Standard (static) Fault Trees model the failure behaviour of a system dependent on its component failures. To overcome their limited expressive power, *Dynamic Fault Trees* (DFT) have been introduced to model advanced dependability patterns, such as spare management, functional dependencies, and sequencing [45]. Currently, in addition to static Fault Trees the COMPASS toolset only supports sequencing, by representing ordering constraints between basic faults using priority AND (PAND) gates. However, there is strong demand for improving safety assessment by supporting more expressive constructs in DFTs. They often lead to fault models that are more succinct, and thus better comprehensible.

DFT analysis relies on the extraction of an underlying stochastic model, such as a Bayesian Network, a Continuous-Time Markov Chain, a Stochastic Petri Net, or an Interactive Markov Chain. This is a time-consuming process, in particular for more expressive dependency patterns, raising the need for approaches to make it simpler and cheaper (in terms of computational resources). A key technique is the *reduction of the state space* of DFTs prior to (and during) their analysis. Here, one technique is to consider DFTs as (typed) directed graphs and to manipulate them by graph transformation, a powerful technique to rewrite graphs via pattern matching. In [64], a catalogue of 28 (templates of) rules is presented that convert a given DFT into a smaller, equivalent one having the same system reliability and availability characteristics. Experiments with 170 DFTs, originating from standard examples from the literature as well as industrial case studies from aerospace and railway engineering, showed encouraging results. The rewriting approach enabled us to cope with 49 DFTs that could not be handled before. But also for static Fault Trees the processing pays off, rendering analysis much faster and more memory efficient, up to two orders of magnitude.

More state-space reductions can be obtained by tailoring two successful techniques from the field of model checking, namely, symmetry reduction and partial-order reduction [87]. In the DFT setting, this amounts to the detection of isomorphic sub-DFTs, of stochastic independencies, and of sub-DFTs that become obsolete after the occurrence of some faults. In addition, certain failure orderings arising from superfluous non-determinism can be ignored in the analysis. All this comes at no run-time penalty: as the results in [64, 87] indicate, structural transformations of DFTs operate very fast, and the stochastic model generation is significantly accelerated due to the reduction. This opens the possibility of supporting more expressive types of Fault Trees, and considering techniques on how to analyse them efficiently in the COMPASS toolset.

In addition, one can exploit that some stochastic analyses such as assessing the reliability of a system—how likely is it operational up to a certain point in time?—are *compositional*: the measure can directly be computed from its sub-DFTs' measures. This means that the analysis can be carried out in a modular way by considering only a part of the state space in each step, and by re-computing measures incrementally after local changes in DFTs.

Last but not least, the *parameter synthesis* techniques as sketched in Sect. 6.4.5 can also be applied to DFTs. Classical analyses require all component failure rates to be known, which often does not hold in practice. Thus, a relevant problem is to synthesise the allowed component failure rates ensuring, e.g., a given mean minimal time between failures. This is clearly an instance of the parameter synthesis problem as described earlier.

### *6.4.7 Multi-objective Verification*

Besides the correctness of their functional behaviour, systems are required to exhibit adequate *performance* characteristics. The latter can be measured by, e.g., its average and peak energy consumption, construction costs, and its availability and reliability. These measures are often contradictory: while using more power for data transmission typically increases the reliability level of communication, it also entails a higher energy consumption. But also less obvious mutual dependencies can emerge: optimising a system for (long-run) availability might reduce the (short-term) reliability.

In order to systematically investigate such effects, *multi-objective model checking* can be employed. This is a fully automatic technique by which, based on a model of the system under consideration and some measures-of-interest, a so-called Pareto curve is deduced [56]. The latter gives an (often graphical) representation of the optimal strategy for resolving non-deterministic choices in the system with respect to a given weighting of these measures. Currently, only Markov Decision Processes can be handled by this technique [55, 68]. While these support non-deterministic choices (to be optimised) and discrete probabilities, they lack continuously distributed random delays, which are typically used to describe, e.g., mechanical wear or other sources of failures.

*Markov Automata* [59] constitute a highly expressive formalism which extends Markov Decision Processes by such random delays. They are known to provide a suitable model for, e.g., Dynamic Fault Trees (cf. Sect. 6.4.6) or to define formal semantics of Stochastic Petri Nets. Previous work is only able to cope with optimising single measures on Markov Automata [59]. Our aim is therefore to extend the techniques that have been developed for Markov Decision Processes to this richer setting. Here, we will have to distinguish time-bounded analysis problems from others. With regard to the former, our idea is to employ the digitisation approach from [59] to derive upper and lower bounds for time-bounded reachability probabilities. In the unbounded case, it will be possible to completely abstract from the continuous behaviour of the Markov Automaton by instead considering the underlying Markov Decision Process.

## 6.5 Conclusion

To tackle the problem of correctness and reliability of control software in the aerospace domain, *formal methods* are increasingly being employed. They enable the exhaustive and mathematically founded analysis of all possible behaviours of a computer program and of its interaction with the overall system and the verification of properties such as functional correctness. They also allow to reduce the effort and, thus, the cost of testing activities [4]. Due to their benefits, they are increasingly becoming an integral part of the development cycle of safety-critical systems [44, 47, 62].

We have given a sketch of the ESA-funded COMPASS project, the related toolset and its underlying techniques. COMPASS provides an integrated approach to integrated system-software co-engineering covering modelling, analysis, and verification activities. While these methods turned out to be very useful in practical applications, there is still room for technological improvements. In the second part of this chapter, we have identified current bottlenecks and possible solutions. This comprises techniques that cover both the nominal and the error behaviour of systems, such as the formal validation of quantitative requirement specifications (Sect. 6.4.1) and multi-objective verification (Sect. 6.4.7).

Other methods focus on fault management, with the goal of improving the expressivity of error modelling and related analysis methods. In this category, we find approaches such as contract-based failure analysis (Sect. 6.4.2), Timed Failure Propagation Graphs (Sect. 6.4.4), parametric error models (Sect. 6.4.5), and Dynamic Fault Trees (Sect. 6.4.6). Last but not least, there is demand for better support for fault diagnosis and management in the form of FDIR design (Sect. 6.4.3).

To guarantee a smooth embedding of such technologies in the overall system development process, additional support by accompanying process-oriented measures is required. Here, the main concern is the integration of the modelling, analysis, and validation activities enabled by COMPASS with the design and implementation steps as provided by other tools supporting AADL (such as TASTE [78]) or other specification languages (such as Simulink). Moreover we note that our approach is completely model based. Thus, methods for generating code from AADL specifica-

tions and for checking the conformance of a hardware/software implementation with respect to the AADL model are required. For the latter, model-based testing [30] can be employed, which is an automated technique in which the test generation process is steered by the AADL model.

## References


1. S. Abdelwahed, G. Karsai, N. Mahadevan, S. Ofsthun, Practical implementation of diagnosis systems using timed failure propagation graph models. IEEE Trans. Instrum. Meas. **58**(2), 240–247 (2009)
2. E. Alaña, H. Naranjo, Y. Yushtein, M. Bozzano, A. Cimatti, M. Gario, R. de Ferluc, G. Garcia, Automated generation of FDIR for the COMPASS integrated toolset (AUTOGEF), in *Proceedings of DASIA 2012*, vol. ESA SP 701 (2012)
3. J. Alonso, M. Grottke, A.P. Nikora, K.S. Trivedi, An empirical investigation of fault repairs and mitigations in space mission system software, in *Proceedings of DSN 2013* (IEEE, 2013), pp. 1–8
4. P. Anderson, Detecting bugs in safety-critical code. Dr. Dobb's J. **33**(3), 22–27 (2008), http://www.drdobbs.com/tools/detecting-bugs-in-safety-critical-code/206104422
5. M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, A. Tang, Aligning qualitative, real-time, and probabilistic property specification patterns using a structured English grammar. IEEE Trans. Software Eng. **41**(7), 620–638 (2015)
6. C. Baier, B. Haverkort, H. Hermanns, J.P. Katoen, Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Software Eng. **29**(6), 524–541 (2003)
7. C. Baier, B.R. Haverkort, H. Hermanns, J.P. Katoen, Model checking meets performance evaluation. SIGMETRICS Perform. Eval. Rev. **32**(4), 10–15 (2005)
8. C. Baier, J.P. Katoen, *Principles of Model Checking* (MIT Press, New York, 2008)
9. E. Bartocci, R. Grosu, P. Katsaros, C.R. Ramakrishnan, S.A. Smolka, Model repair for probabilistic systems, in *Proceedings of TACAS 2011*. LNCS, vol. 6605 (Springer, 2011), pp. 326–340
10. A. Biere, A. Cimatti, E. Clarke, Y. Zhu, Symbolic model checking without BDDs, in *Proceedings of TACAS 1999*. LNCS, vol. 1579 (Springer, 1999), pp. 193–207
11. A. Biere, K. Heljanko, T.A. Junttila, T. Latvala, V. Schuppan, Linear encodings of bounded LTL model checking. Logical Methods Comput. Sci. **2**(5) (2006)
12. B. Bittner, Formal failure analyses for effective fault management: an aerospace perspective, Ph.D. thesis, University of Trento, 2016
13. B. Bittner, M. Bozzano, R. Cavada, A. Cimatti, M. Gario, A. Griggio, C. Mattarei, A. Micheli, G. Zampedri, The xSAP safety analysis platform, in *Proceedings of TACAS 2016*. LNCS, vol. 9636 (Springer, 2016), pp. 533–539
14. B. Bittner, M. Bozzano, A. Cimatti, Automated synthesis of timed failure propagation graphs, in *Proceedings of IJCAI 2016* (AAAI Press, 2016), pp. 972–978
15. B. Bittner, M. Bozzano, A. Cimatti, R. de Ferluc, M. Gario, A. Guiotto, Y. Yushtein, An integrated process for FDIR design in aerospace, in *Proceedings of IMBSA 2014*. LNCS, vol. 8822 (Springer, 2014), pp. 82–95
16. B. Bittner, M. Bozzano, A. Cimatti, X. Olive, Symbolic synthesis of observability requirements for diagnosability, in *Proceedings of AAAI-12* (2012)
17. B. Bittner, M. Bozzano, A. Cimatti, G. Zampedri, Automated verification and tightening of failure propagation models, in *Proceedings of AAAI 2016* (2016), pp. 3724–3730
18. V. Bos, H. Bruintjes, S. Tonetta, Catalogue of system and software properties, in *Proceedings of SAFECOMP 2016*. LNCS, vol. 9922 (Springer, 2016), pp. 88–101
19. H. Boudali, P. Crouzen, M. Stoelinga, A rigorous, compositional, and extensible framework for dynamic fault tree analysis. IEEE Trans. Dependable Secure Comput. **7**(2), 128–143 (2010)

20. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, R. Sebastiani, Mathsat: tight integration of SAT and mathematical decision procedures. J. Autom. Reason. **35**, 265–293 (2005)
21. M. Bozzano, A. Cimatti, M. Gario, A. Micheli, SMT-based validation of timed failure propagation graphs, in *Proceedings of AAAI 2015* (2015), pp. 3724–3730
22. M. Bozzano, A. Cimatti, M. Gario, S. Tonetta, Formal design of fault detection and identification components using temporal epistemic logic, in *Proceedings of TACAS 2014*. LNCS, vol. 8413 (Springer, 2014), pp. 46–61
23. M. Bozzano, A. Cimatti, M. Gario, S. Tonetta, Formal design of asynchronous fault detection and identification components using temporal epistemic logic. Logical Methods Comput. Sci. **11**(4), 1–33 (2015)
24. M. Bozzano, A. Cimatti, J.P. Katoen, P. Katsaros, K. Mokos, V.Y. Nguyen, T. Noll, B. Postma, M. Roveri, Spacecraft early design validation using formal methods. Reliab. Eng. Syst. Safety **132**, 20–35 (2014)
25. M. Bozzano, A. Cimatti, J.P. Katoen, V.Y. Nguyen, T. Noll, M. Roveri, Safety, dependability, and performance analysis of extended AADL models. Comput. J. **54**(5), 754–775 (2011)
26. M. Bozzano, A. Cimatti, C. Mattarei, A. Griggio, Efficient anytime techniques for model-based safety analysis, in *Proceedings of CAV 2015*. LNCS, vol. 9206 (Springer, 2015), pp. 603–621
27. M. Bozzano, A. Cimatti, C. Mattarei, S. Tonetta, Formal safety assessment via contract-based design, in *Proceedings of ATVA 2014* (2014), pp. 81–97
28. M. Bozzano, A. Cimatti, F. Tapparo, Symbolic fault tree analysis for reactive systems, in *Proceedings of ATVA 2007*. LNCS, vol. 4762 (Springer, 2007), pp. 162–176
29. T. Brázdil, V. Forejt, J. Kretínský, A. Kucera, The satisfiability problem for Probabilistic CTL, in *Proceedings of LICS 2008* (IEEE, 2008), pp. 391–402
30. M. Broy, B. Jonsson, J.P. Katoen, M. Leucker, A. Pretschner, (eds.), *Model-Based Testing of Reactive Systems: Advanced Lectures*. LNCS, Vol. 3472 (Springer, 2005)
31. H. Bruintjes, J.P. Katoen, D. Lesens, A statistical approach for timed reachability in AADL models, in *Proceedings of DSN 2015* (IEEE CS Press, 2015), pp. 81–88
32. R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, S. Tonetta, The nuXmv symbolic model checker. CAV **2014**, 334–342 (2014)
33. S. Chakraborty, J.P. Katoen, On the satisfiability of some simple probabilistic logics, in *Proceedings of LICS 2016* (ACM, 2016), pp. 56–66
34. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV 2: an open-source tool for symbolic model checking, in *Proceedings of CAV 2002*. LNCS, vol. 2404 (Springer, 2002), pp. 359–364
35. A. Cimatti, R. Demasi, S. Tonetta, Tightening a contract refinement, in *Proceedings of SEFM 2016* (2016), pp. 386–402
36. A. Cimatti, M. Dorigatti, S. Tonetta, OCRA: a tool for checking the refinement of temporal contracts, in *Proceedings of ASE 2013* (2013), pp. 702–705
37. A. Cimatti, C. Pecheur, R. Cavada, Formal verification of diagnosability via symbolic model checking, in *Proceedings of IJCAI 2003* (Morgan Kaufmann, 2003), pp. 363–369
38. A. Cimatti, S. Tonetta, Contracts-refinement proof system for component-based embedded systems. Sci. Comput. Program. **97**, 333–348 (2015)
39. The COMPASS project, http://www.compass-toolset.org/
40. COMPASS user manual. Technical Report. Version 3.0, COMPASS Consortium (2016), http://www.compass-toolset.org/docs/compass-manual.pdf
41. COMPASS tutorial. Technical Report Version 3.0, COMPASS Consortium (2016), http://www.compass-toolset.org/docs/compass-tutorial.pdf
42. C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.P. Katoen, E. Abraham, PROPhESY: a probabilistic parameter synthesis tool, in *Proceedings of CAV 2015*, LNCS, vol. 9206 (Springer, 2015), pp. 214–231
43. S. Derisavi, H. Hermanns, W.H. Sanders, Optimal state-space lumping in Markov chains. Inf. Process. Lett. **87**(6), 309–315 (2003)

44. Software considerations in airborne systems and equipment certification. Software Standard DO-178C/ED-12C, RTCA Inc. and EUROCAE (2011)
45. J.B. Dugan, S.J. Bavuso, M.A. Boyd, Dynamic fault-tree models for fault-tolerant computer systems. IEEE Trans. Reliab. **41**(3), 363–377 (1992)
46. M. Dwyer, G. Avrunin, J. Corbett, Patterns in property specifications for finite-state verification, in *Proceedings of ICSE 1999* (IEEE CS Press, 1999), pp. 411–420
47. Space engineering: Verification. ECSS Standard E-ST-10-02C, European Cooperation for Space Standardization (2009)
48. Space engineering: System engineering general requirements. ECSS Standard E-ST-10C, European Cooperation for Space Standardization (2009)
49. Space product assurance: Failure modes, effects (and criticality) analysis (FMEA/FMECA). ECSS Standard Q-ST-30-02C, European Cooperation for Space Standardization (2009)
50. Space product assurance: Availability analysis. ECSS Standard Q-ST-30-09C, European Cooperation for Space Standardization (2008)
51. Space product assurance: Dependability. ECSS Standard Q-ST-30C, European Cooperation for Space Standardization (2009)
52. Space product assurance: Fault tree analysis—adoption notice ECSS/IEC 61025. ECSS Standard Q-ST-40-12C, European Cooperation for Space Standardization (2008)
53. Space product assurance: Safety. ECSS Standard Q-ST-40C, European Cooperation for Space Standardization (2009)
54. M.A. Esteve, J.P. Katoen, V.Y. Nguyen, B. Postma, Y. Yushtein, Formal correctness, safety, dependability and performance analysis of a satellite, in *Proceedings of ICSE 2012* (ACM and IEEE CS Press, 2012), pp. 1022–1031
55. K. Etessami, M.Z. Kwiatkowska, M.Y. Vardi, M. Yannakakis, Multi-objective model checking of Markov decision processes. Logical Methods Comput. Sci. **4**(4) (2008)
56. V. Forejt, M. Kwiatkowska, D. Parker, Pareto curves for probabilistic model checking, in *Proceedings of ATVA 2012*. LNCS, vol. 7561 (Springer, 2012), pp. 317–332
57. M. Gario, A formal foundation of FDI design via temporal epistemic logic. Ph.D. thesis, Trento University, Italy (2016), https://marco.gario.org/phd/gario_phd.pdf
58. D. Guck, T. Han, J.P. Katoen, M.R. Neuhäußer, Quantitative timed analysis of interactive Markov chains, in *Proceedings of NFM 2012*. LNCS, vol. 7226 (Springer, 2012), pp. 8–23
59. D. Guck, H. Hatefi, H. Hermanns, J.P. Katoen, M. Timmer, Modelling, reduction and analysis of Markov automata, in *Proceedings of QEST 2013*. LNCS, vol. 8054 (Springer, 2013), pp. 55–71
60. K. Heljanko, T.A. Junttila, T. Latvala, Incremental and complete bounded model checking for full PLTL, in *Proceedings of CAV 2005*. LNCS, vol. 3576 (2005), pp. 98–111
61. H. Hermanns, *Interactive Markov Chains: The Quest for Quantified Quality*. LNCS, vol. 2428 (Springer, 2002)
62. G.J. Holzmann, The power of 10: rules for developing safety-critical code. Computer **39**(6), 95–99 (2006)
63. N. Jansen, F. Corzilius, M. Volk, R. Wimmer, E. Abraham, J.P. Katoen, B. Becker, Accelerating parametric probabilistic verification, in *Proceedings of QEST 2014*. LNCS, vol. 8657 (Springer, 2014), pp. 404–420
64. S. Junges, D. Guck, J.P. Katoen, A. Rensink, M. Stoelinga, Fault trees on a diet, in *Proceedings of SETTA 2015*. LNCS, vol. 9409 (Springer, 2015), pp. 3–18
65. J.P. Katoen, V.Y. Nguyen, T. Noll, Formal validation methods in model-based spacecraft systems engineering, in *Modeling and Simulation-Based Systems Engineering Handbook*, Chap. 14 (CRC Press, 2014), pp. 339–375
66. J.P. Katoen, L. Song, L. Zhang, Probably safe or live, in *Proceedings of CSL-LICS 2014* (ACM, 2014), pp. 55:1–55:10
67. J.P. Katoen, I.S. Zapreev, E.M. Hahn, H. Hermanns, D.N. Jansen, The ins and outs of the probabilistic model checker MRMC. Perform. Eval. **68**(2), 90–104 (2011)
68. M. Kwiatkowska, G. Norman, D. Parker, H. Qu, Compositional probabilistic verification through multi-objective model checking. Inf. Comput. **232**, 38–65 (2013)

69. MathSAT, http://mathsat.fbk.eu
70. A. Misra, J. Sztipanovits, A. Underbrink, R. Carnes, B. Purves, Diagnosability of dynamical systems, in *3rd International Workshop on Principles of Diagnosis* (1992), pp. 239–244
71. MRMC – Markov Reward Model Checker, http://www.mrmc-tool.org/
72. T. Noll, Safety, dependability and performance analysis of aerospace systems, in *Proceedings of FTSCS 2014*. CCIS, vol. 476 (Springer, 2015), pp. 17–31
73. Nonelectronic parts reliability data (NPRD-2016). Technical Report, Quanterion Solutions Inc. (2015), https://www.quanterion.com/product/publications/nonelectronic-parts-reliability-data-publication-nprd-2016/
74. The NuSMV model checker, http://nusmv.fbk.eu
75. The nuXmv model checker, https://nuxmv.fbk.eu/
76. S.C. Ofsthun, S. Abdelwahed, Practical applications of timed failure propagation graphs for vehicle diagnosis, in *Proceedings of Autotestcon 2007* (IEEE, 2007), pp. 250–259
77. S. Pathak, E. Abraham, N. Jansen, A. Tacchella, J.P. Katoen, A greedy approach for the efficient repair of stochastic models, in *Proceedings of NFM 2015*. LNCS, vol. 9058 (Springer, 2015), pp. 295–309
78. M. Perrotin, E. Conquet, J. Delange, A. Schiele, T. Tsiodras, TASTE: a real-time software engineering tool-chain overview, status, and future, in *Proceedings of SDL 2011*. LNCS, vol. 7083 (Springer, 2012), pp. 26–37
79. I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, A. Cimatti, Formal analysis of hardware requirements, in *Proceedings of DAC 2006* (2006), pp. 821–826
80. Reliability Prediction of Electronic Equipment. No. MIL-HDBK-217F in Military standardization handbook. Department of Defense, USA (1995), http://quicksearch.dla.mil/qsDocDetails.aspx?ident_number=53939
81. Architecture Analysis & Design Language (AADL) Annex, Volume 1, Annex E: Error Model Annex. SAE Standard AS5506/1A (International Society of Automotive Engineers, 2015)
82. Architecture Analysis and Design Language (AADL) Annex, Volume 1, Annex A: Graphical AADL Notation. SAE Standard AS5506/1 (International Society of Automotive Engineers, 2006)
83. Architecture Analysis & Design Language (AADL). SAE Standard AS5506 (International Society of Automotive Engineers, 2004)
84. Architecture Analysis & Design Language (AADL) (rev. B). SAE Standard AS5506B (International Society of Automotive Engineers, 2012)
85. Solar Orbiter, http://sci.esa.int/solar-orbiter/
86. A. Valmari, G. Franceschinis, Simple $O(m \log n)$ time Markov chain lumping, in *Proceedings of TACAS 2010*. LNCS, vol. 6015 (Springer, 2010), pp. 38–52
87. M. Volk, S. Junges, J.P. Katoen, Advancing dynamic fault tree analysis – get succinct state spaces fast and synthesise failure rates, in *Proceedings of SAFECOMP 2016*. LNCS, vol. 9922 (Springer, 2016), pp. 253–265
88. R. Wimmer, M. Herbstritt, H. Hermanns, K. Strampp, B. Becker, Sigref – a symbolic bisimulation tool box, in *Proceedings of ATVA 2006*. LNCS, vol. 4218 (Springer, 2006), pp. 477–492