Formal Verification and Validation of AADL Models

M. Bozzano[2], R. Cavada[2], A. Cimatti[2], J.-P. Katoen[1], V.Y. Nguyen[1], T. Noll[1], X. Olive[3]

[1] Software Modeling and Verification Group, RWTH Aachen University, Germany

[2] Embedded Systems Group, Fondazione Bruno Kessler, Italy

[3] Research & Technology, Thales Alenia Space, France

Topics: Hardware/Software Co-Design, Model Driven Engineering, Verification and Validation
Domains: Satellite and Space Exploration

1 Introduction

Safety-critical systems are increasingly difficult to comprehend due to their rising complexity. Methodologies, tools and modeling formalisms have been developed to overcome this. Component-based design is an important paradigm that is shared by many of them. It helps to master the overall complexity while in addition allowing for reusability. Furthermore, it easily supports the common issues in the engineering disciplines, like hardware/software (i.e., co-engineering), performability, dependability, reliability, availability, maintainability and safety engineering (RAMS). Model artifacts that are typical for a discipline can be encapsulated in the affected components, while staying imperceptible for non-affected components. This leads to different views of the system under development, which subsequently entails the natural distinction in different system abstractions, formalisms and tools. Nonetheless, there is only one system under development. With the current methodologies, there is no single view of this system that links all aspects relevant to all involved engineering disciplines in a coherent manner.

To overcome this problem, the European Space Agency (ESA) has set up the COMPASS project[1] (Correctness, Modeling, and Performance of Aerospace Systems). Its goal is to develop a coherent and multidisciplinary approach towards developing systems at an architecture (i.e., systems engineering) level. The Architecture and Analysis Design Language (AADL) [1] and its Error Model Annex [2] is adopted as the key formalism to *fully* capture model the system under development (see § 2). Furthermore, we base a collection of verification and validation (V&V) activities over AADL models on state-of-the-art model checking techniques, like symbolic and bounded SAT-based model checking, and also probabilistic variants thereof. A mature graphical-driven toolset has been developed to support this approach (see § 3). The toolset and its potential have been evaluated by Thales Alenia Space using several case studies from the satellite domain (see § 4).

2 Architecture and Analysis Design Language

The AADL modeling language is standardized by several major industrial and academic partners since 2004 [1]. Models in this language can capture real-time, performance critical and distributed aspects at an architectural level. It supports annexes to extend the language, among which the Error Model Annex [2] was particularly interesting for our application. We have defined a formal semantics for its core features and added additional elements needed to enable formal validation and verification. The resulting input language has the following design features:

- Modeling both the system's nominal and faulty behavior. To this aim, our input language provides primitives to describe software and hardware faults, error propagation (that is, turning fault occurrences into failure events), sporadic (transient) and permanent faults, and degraded modes of operation (by mapping failures from architectural to service level).
- Modeling (partial) observability and the associated observability requirements. These notions are essential to deal with diagnosability and Fault Detection, Isolation and Recovery (FDIR) analyses.
- Specifying timed and hybrid behavior. In particular, in order to analyze continuous physical systems such as mechanics and hydraulics, our input language supports continuous real-valued variables with (linear) time-dependent dynamics.
- Modeling probabilistic aspects, such as random faults, repairs, and stochastic timing.

A complete specification consists of three parts, namely a description of the nominal behavior, a description of the error behavior, and a fault injection specification that describes how the error behavior influences the nominal behavior.

The *nominal model* describes the system under normal operation. It is a system decomposition of interacting components in which system detail can be abstracted by defining a hierarchy among components.

[1] http://compass.informatik.rwth-aachen.de

The interaction interfaces are specified using port connections of which there are three types. Data port connections expose component variables to other components, flow port connections are evaluatable functions based on incoming data ports and event port connections are used to define (multi-way) hand-shaking communication between components. These port interfaces are complemented by a *mode transition system*, which describe changes over the ports, and thus essentially capture a component's behavior. The transition system can be annotated with linear differential equations and timing constraints to model the evolution of physical aspects, like temperature, pressure and scheduling of tasks. Modes can also be used to represent degradation of the system. Transitions between these modes can lead to dynamic reconfigurations by (de)activating components and port connections.

An *error model* as defined by AADL's Error Model Annex [2] expresses in which ways the system can fail, i.e., it models how faults may affect normal operation, how failure effects propagate throughout the system and may lead to a degraded mode of operation. It is modeled as a *probabilistic finite-state automaton*. Transitions occur due to spontaneous events or propagations. Events may be annotated with a rate that indicates the expected number of occurrences per time unit. Propagations occur between components when they are in a super-subcomponent relation or when they access a common bus. This implicit linking reflects the oblivious and pervasive nature of error models.

As error models bear no relation with nominal models, an error model does not influence the nominal model unless they are linked through *fault injection*. An injection describes how a failure event lead to data failures in the nominal model, thereby changing the system's operation. There is an automatic procedure, called *model extension*, that integrates nominal and errors models with the given fault injections. The principal idea is that the nominal and error models are running concurrently. Using the fault injections, error transitions from the error models are interwoven in the nominal model via an interleaving semantics. The specified data failures override the nominal data effects. Failures hold as long as the error model is in an error state. They can be made transient by allowing the error model to leave erroneous states. The resulting model after model extension is an integrated AADL model, also called the *extended model*.

3 Toolset for Verification & Validation

A graphical toolset, named COMPASS Toolset, has been developed to conduct verification and validation analyses over AADL models. Its development has reached its final phase, and has been accepted by the ESA during its Acceptance Review meeting of March 2010. A public release is planned for May 2010.

The toolset leverages existing mature tools like the NuSMV symbolic model checker [3] and the MRMC probabilistic model checker [4]. The inputs of these tools are obtained by fully automated semantic-preserving model transformations from AADL and the requirements specification patterns. These patterns act as parameterized templates [5, 6] for a comprehensible and easy-to-use method of capturing safety, correctness, performance and dependability requirements. Several analyses have been defined using these inputs. Their outputs are transformed back to a graphical and engineer-friendly form.

The next subsections introduce the analyses and their applicability in the system design process.

3.1 Requirements Validation

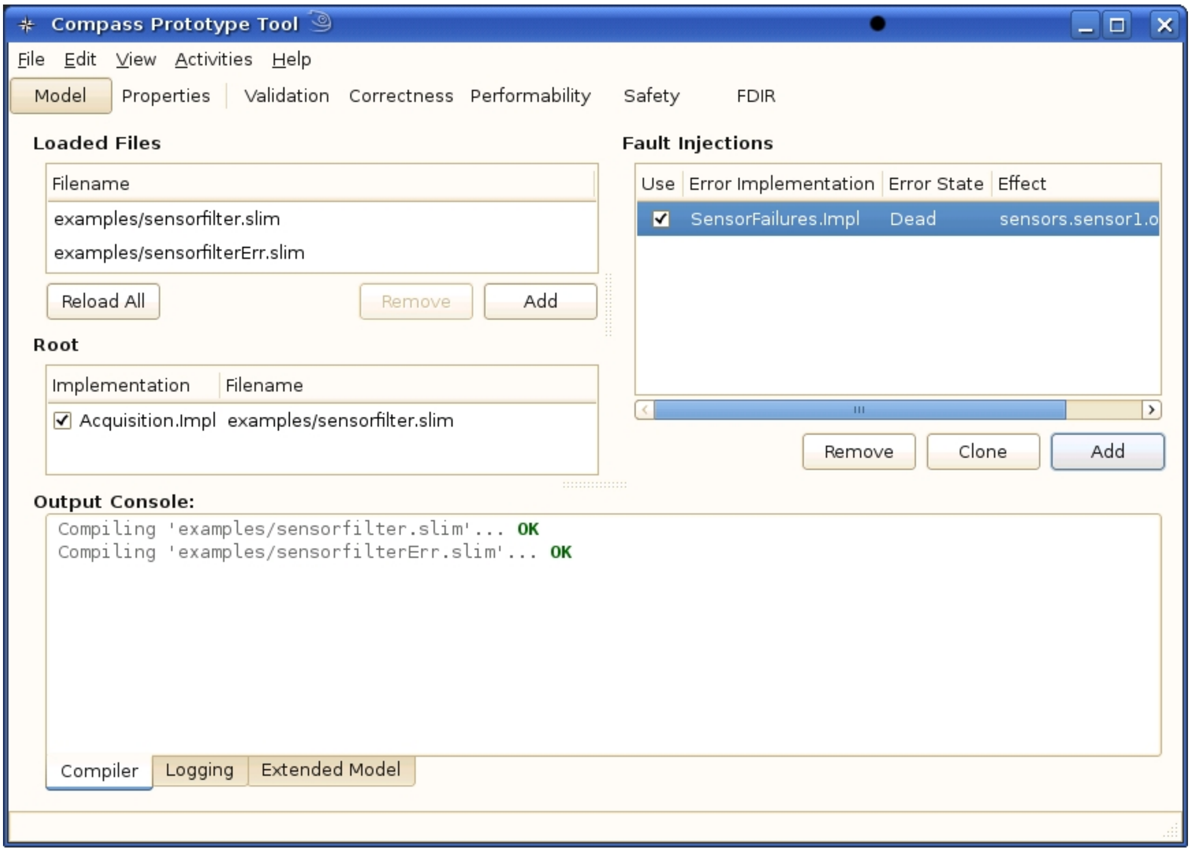
The validation activities are centered around the requirements. In order to ensure the quality of requirements, they can be validated independently of the system. For this, the requirements are converted automatically to their underlying formal logics (like Linear Temporal Logic and Computational Tree Logic), after which the analyses can be run. 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) [7]. 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.

3.2 Functional Verification

The formalized requirement and the AADL model are the input for analyzing operational correctness, which is the first step to be performed during the system development life cycle. 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. One particular instance of this general model-checking problem that is specifically supported by the toolset is deadlock checking, i.e., ensuring that the system does not give rise to terminating computations. This is usually required for reactive systems. Moreover the toolset offers the feature to interactively simulate the execution of the system.

3.3 Safety and Dependability

Analyzing system safety and dependability is a fundamental step that is performed in parallel with system



Filename
examples/sensorfilter.slim
examples/sensorfilterErr.slim

Implementation	Filename
☑ Acquisition.Impl	examples/sensorfilter.slim

Use	Error Implementation	Error State	Effect
☑	SensorFailures.Impl	Dead	sensors.sensor1.o

```
Compiling 'examples/sensorfilter.slim'... OK
Compiling 'examples/sensorfilterErr.slim'... OK
```

Figure 1: Main screen of the COMPASS toolset with AADL models loaded.

design and verification of functional correctness. The goal is to investigate the behavior 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 mandatory for its deployment and use. For this the toolset generates common-practice artifacts, namely (dynamic) fault trees, (dynamic) Failure Modes and Effects Analysis tables (FMEA), fault tolerances, and criticality numbers [8].

3.4 Performability

To guarantee the required system performance in the presence of faults, integrated hardware and software models can be evaluated with respect to their performance behavior in degraded modes of operation. The measures resulting from this analysis can be used to evaluate design decisions for costs and risks. The analysis is based on the input AADL model extended with its error models and fault injections. The resulting extended model is transformed to its underlying Markov model, which enables MRMC to compute the measures of interest like reliability, maintainability and survivability.

3.5 Fault Detection, Isolation and Recovery

System models can include a formal description of both the fault detection and isolation sub-systems, and the recovery actions to be taken. Based on these models, tool facilities are provided to analyze the operational effectiveness of the FDIR system, especially on how faults are detected, whether observability requirements suffice for system diagnosability [9], how the FDIR system isolates faults and whether recovery actions, in presence of failures, lead to an operational system state.

4 Case Studies

Two distinct case studies have been used to evaluate the AADL modeling language and the applicability of the COMPASS toolset in an industrial context. The first case study is related to the definition of satellite mode management and its associated FDIR strategy. The second case study models the thermal regulation function, where two sub cases have been derived at functional and behavioral levels. Both case studies are described in the next subsections

4.1 Global FDIR Strategy

The global FDIR strategy case study deals with the definition of satellite mode management and with the FDIR strategy management. The case study models the satellite mode management, the AOCS (Attitude and Orbital Control System) mode management, the configuration and reconfiguration sequences and an abstracted model of the AOCS equipments and other functional subsystems.

The satellite mode management consists of three modes and seven elementary transitions. The AOCS mode management has six modes and about 20 transitions. Five AOCS units have been considered. The model of the equipment is based on a six-state automaton, which can include timed transitions. The two functional chains are represented in an time-abstracted way. No behavioral part is modeled, only flags have been used to inject the faults, and emit the system alarm. The fault flag represents the level 1 faults (detected by software), which occur on the equipments. Examples of such faults are transmission error, electrical default and data inconsistency. The system level alarm represents several hardware detected faults (loss of pointing Earth or Sun). They are used as last barrier before losing the satellite and a direct reconfiguration to the satellite's safe mode.

The configuration and reconfiguration sequences represents sequences of commands, which are send to the AOCS units. Each AOCS mode has a configuration sequence to set up the right configuration of the AOCS units. The reconfiguration sequences are used in case of a detected failure. Both sequences used the same mechanism to initiate. An event is sent by the monitor (reconfiguration) or by the mode transition between two AOCS modes. The event is caught by a sequence manager, which emits an event for starting the procedure. Initially all the procedures are in an `Idle` state, waiting for the start event. This mechanism is similar to the one provided by the ECSS Packet Utilization Standard (PUS) for communication between ground and onboard application (see Figure 2). Service 5 is used to raise events, service 12 is dedicated to on-board monitoring, and service 19 allows one to launch an action on the reception of an event. These services are chained as such to achieve a reconfiguration that starts from a monitoring failure.

4.2 Verification

The COMPASS toolset was used to analyze the three following items:

1. Identification of all the failures leading to a given level of FDIR. It should allow to validate that the classification of failures by each level is well done. For instance, for level 1 (equipment failure), only the monitor can be the root cause. Level 4 (ultimate one) can only be reached by a system alarm or a combination of failures from lower levels

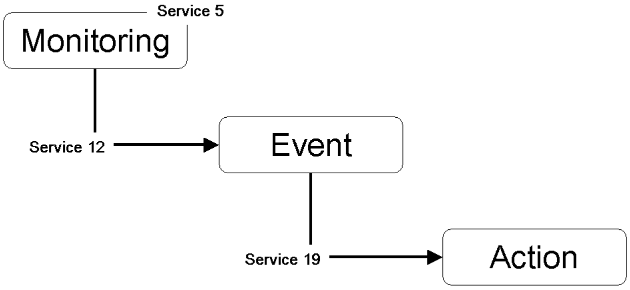


Figure 2: Synopsis of PUS mechanism used for reconfiguration.

2. Identification of the failures, which can lead to the activation of a reconfiguration sequence. The information is then used to check the absence of border effects on the design.

3. Impact of a reconfiguration sequence on the satellite mode and AOCS mode. It allows one to verify the correctness and completeness of the satellite and AOCS modes automata with regard to the equipment configuration.

The analyses for this case study have been conducted on a standard computer (2GHz processor and 2GB memory). The duration of analysis is dependent on the model. All the analyses completed in less than one hour.

4.2.1 Simulation

Before the verification of the above items, the AADL model was checked for deadlocks. This analysis checks for any dead ends in the system, and it is also needed to ensure the correctness of the verification results. Simulations were then used to generate traces of the model. Both random (where the evolution of the trace is determined by a randomly choosing successor transitions) and guided (where the user interactively chooses successor transitions) have been used. Also traces have been generated with and without fault injections. The main goal of the simulations was to check the sanity and fidelity of the model.

4.2.2 Fault Tree Generation

The first two analysis items (identification of failures leading to a given level of FDIR and the activation of reconfiguration sequence) have been verified by generating fault trees. For each FDIR level, one fault tree has been generated. Each leaf represents a fault, which can lead to this level. In this case, the FDIR level is considered as the feared event. Figure 3 shows the resulting fault tree for the most critical event (level 4). From the requirements, level 4 can only be reached,

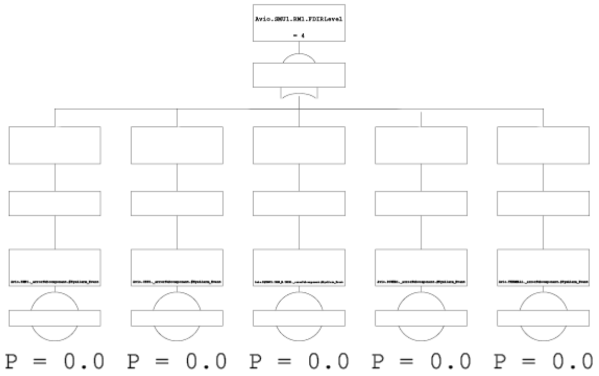


Figure 3: Generated fault tree for FDIR level 4.

when one of five system alarms occurs. The expected result is manually verified using the fault tree.

The second item has been verified similarly. The feared event is the occurrence of the first step of the reconfiguration sequence. Moving to this first step means that the fault occurrence is related to this sequence. One fault tree has been generated for each reconfiguration sequence. Each fault set leading to the reconfiguration sequence has been identified. Then we checked manually, that the FDIR level of the reconfiguration sequence and of the faults set are the same. Each time we identified the same FDIR level for the faults and the sequence. This analysis validates the FDIR level classification of the faults and reconfiguration sequence, by crossing both.

4.2.3 Model Checking

The third item, the impact of the reconfiguration sequence on the satellite, was verified using model checking. We wanted to check that the AOCS units will be in the right status (from requirements), depending on the AOCS mode. The requirement was formalized using the global universality requirement pattern. This pattern must be used for system invariants. Informally, the requirement states that *being in mode M implies that unit U has states S*. Twenty properties (five units times four modes) have been model checked this way.

4.3 Thermal Regulation System

The second case study is a thermal subsystem that regulates the satellite's temperature. It is a co-designed system and performs both active and passive regulation. Passive regulation is achieved by optical solar reflectors, shields, and heat pipes. They have no behavior and are therefore not modeled. Active regulation relies on sensors and heaters at dedicated positions in the satellite. Their position is not taken into account as no requirement uses that information.

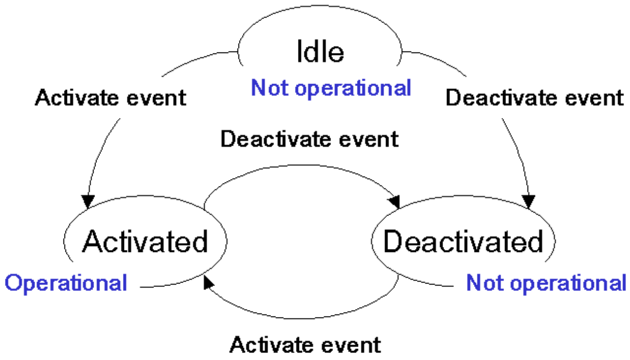


Figure 4: Automaton pattern of a function.

4.3.1 Functional Model

The model at functional level covers the whole perimeter of the thermal regulation function. It consists of five subfunctions and twelve elements (five passive and seven active). As the functional representation is represented by the tree decomposition, we have defined a formal pattern used for all functions. The pattern for a function was defined only once (see Figure 4). The AADL code of that pattern is shown in Figure 5.

Safety analyses have been performed to identify the critical path on the function tree. The fault trees allow one to identify the single failure point. This kind of analysis can be used at system level to trade-off different architectures. FMEA tables have been generated too. Due to the simplicity of the model, the results of the analyses were obtained in less than one second.

This functional model's fidelity is interesting at system level. It permits to take into account the passive component and to have a global representation of satellite. For further analysis, this functional model was completed by a more detailed logical model. It takes into account the behaviors of the thermal lines.

4.3.2 Logical Model

The thermal subsystem consists of thermal lines, heaters and sensors. A thermal line consists of two heater lines (nominal and redundant), two safety switches and three thermistors in hot redundancy. If a failure occurs, the software switches the nominal heater line off and enables the redundant one to continue temperature regulation. Figure 6 shows the decomposition of a thermal line. The behavior of the line depends on the computation of the median of the three thermistor's measurements. Based on this value, it switches the heaters on or off, according to the thresholds of the measured temperature.

AADL Model We only present the interface of the AADL model, which is shown in Figure 7. The descriptions of the interface is presented in Table 1.

```
system Function
features
    Activate: in event port;
    Deactivate : in event port;
    Operational : out data port bool
                  default false;
end Function;

system implementation Function.Impl
modes
  Idle: initial mode;
  Activated : mode;
  Deactivated : mode;
transitions
  Idle -[ Activate
    then Operational := true] ->
    Activated;
  Idle -[ Deactivate
    then Operational := false]->
    Deactivated;
  Deactivated -[ Activate
    then Operational := true] ->
    Activated;
  Activated -[ Deactivate
    then Operational := false]->
    Deactivated;
end Function.Impl;
```

Figure 5: AADL code pattern of a function.

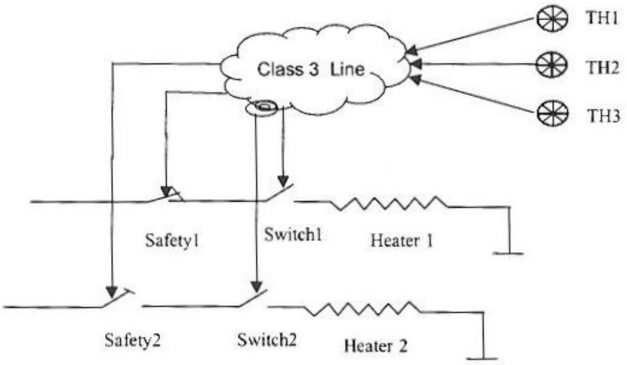


Figure 6: Structure of a thermal line.

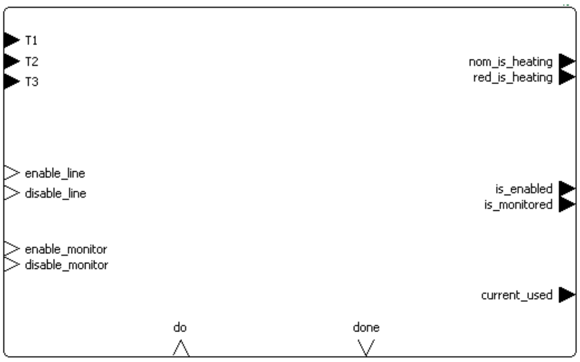


Figure 7: Interface of thermal line component.

Input Port	Type	Description
T1, T2, T3	data	measured values from thermistors
enable_line	event	externally switch line on
disable_line	event	externally switch line off
enable_monitor	event	switch monitoring on
disable_monitor	event	switch monitoring off
do	event	used for simulation
Output Port	**Type**	**Description**
nom_is_heating	data	status of nominal heater
red_is_heating	data	status of redundant heater
is_enabled	data	status of thermal line
is_monitored	data	status of monitoring
current_used	data	currently used heater (nominal/redundant)
done	event	used for simulation

Table 1: Meaning of thermal line component ports.

The AADL model contains the seven active components from the functional model. Each function has been mapped logically on one or several of these components. Two error states have been considered for each unit of the model: (i) for temperature sensor, stuck at min value and stuck at max value, (ii) for the heater, no heating and always heating and (iii) for the switch, stuck at ON and stuck at OFF. An environment model has been used to describe the evolution of temperature. Several reconfiguration sequences have been modeled to move from nominal to redundant switches and to move from nominal to redundant heaters. Three models, describing alternative system designs, have been developed: (M1) without any reconfiguration, (M2) with a reconfiguration from nominal to redundant, and (M2) from nominal to redundant and back (case of transient failure based on ground expertise). The models were analyzed for correctness, reliability, safety and diagnosability capabilities.

4.3.3 Verification

All models have been checked for deadlock-freedom. Model checking was used to verify that the temperature is regulated between the lower and upper bounds of the safe range under nominal operation. Several simulations has been performed for sanity-checking the model. Figure 8 shows a simulation with an injected fault on a heater and the switch to the redundant one.

4.3.4 Safety & Dependability

Fault trees and FMEA tables have been generated. They conform to the expectations. The general form of the fault tree shows the absence of a single point of failure. All the fault sets have at least cardinality two. Fault detection analysis has been used to verify that there exists a means of detecting each feared event. Fault isolation analysis has been conducted too. It provides for each observable the list of faults set to which the observable is sensitive. The two analyses are sufficient to

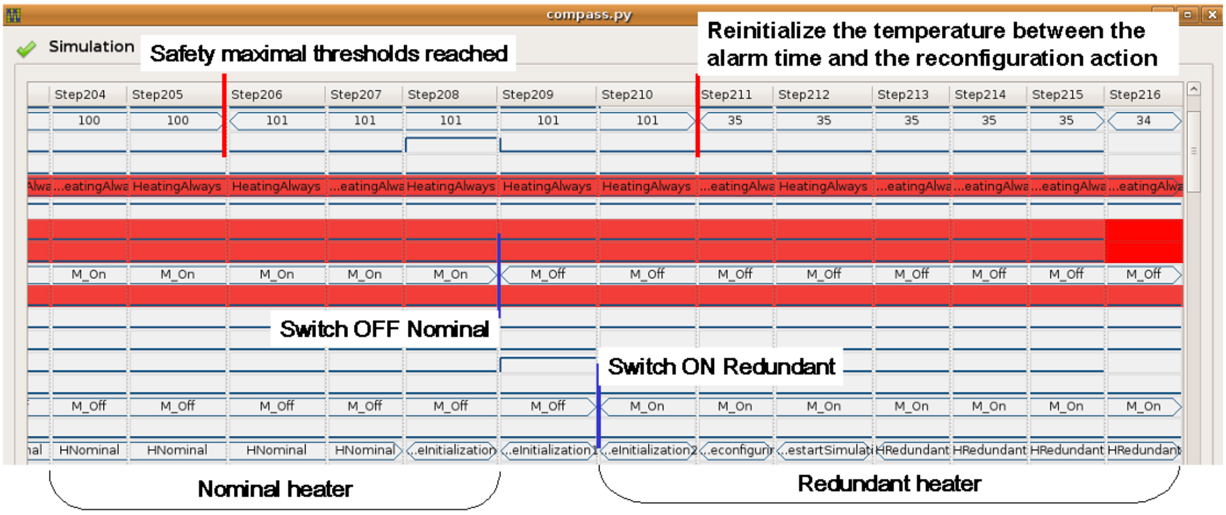


Figure 8: Simulation with fault injection.

validate the observability of the modeled system. The collected data have been used to improve the dependability of the thermal regulation line from a hardware and software point of view.

4.3.5 Recovery Analysis

The main objective was to determine which designs were able to sustain multiple faults by recovering from them. The three configurations of the model (without reconfiguration, with simple reconfiguration and with double reconfiguration) have been checked with the fault recovery analysis. The results are outlined in Figure 9. The columns names indicate failure configurations for the nominal heater (named H1 in Figure 9) and the redundant heater (named H2 in Figure 9).

5 Scalability

Two synthetic benchmarks have been crafted to measure the scalability of the toolset. The adder benchmark is parametric in the number of input modeled. Higher number of inputs increase the model's size. The sensorfilter benchmark is parametric in the degree of redundancy. A higher level of redundancy increases the model's size. The three analysis engines fundamental to all the supported analyses were benchmarked using these two synthetic benchmarks.

The BDD-engine is used for time-abstract models. The results for this engine (see Figure 10) shows that an increasing number of bits (indicating the model's complexity) leads to a higher computation time. As the figure shows, the increase grows exponentially.

The SAT-engine can be used for time-abstract models and is mandatory for timed and hybrid models. The results from the synthetic benchmarks (see Figure 11) shows a similar trend akin to the BDD-engine although this engine can cope with larger models easier.

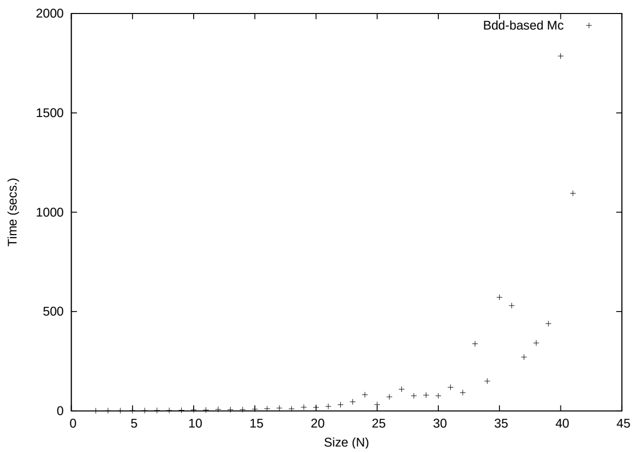


Figure 10: Scalability of the BDD-based engine. The size N means the number of Boolean variables needed to hold a model's state.

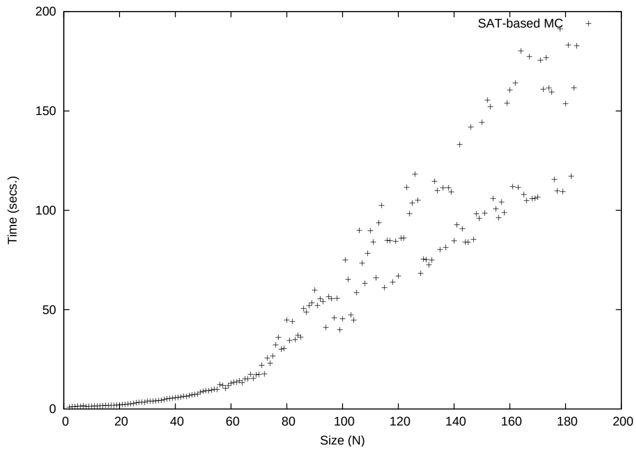
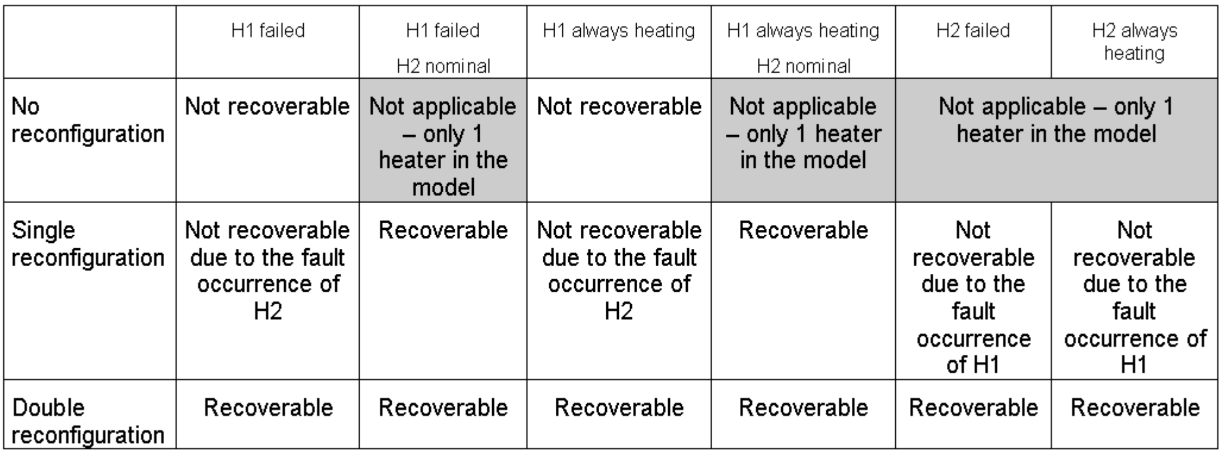


Figure 11: Scalability of the SAT-based engine. The size N means the number of Boolean variables needed to hold a model's state.



	H1 failed	H1 failed H2 nominal	H1 always heating	H1 always heating H2 nominal	H2 failed	H2 always heating
No reconfiguration	Not recoverable	Not applicable – only 1 heater in the model	Not recoverable	Not applicable – only 1 heater in the model	Not applicable – only 1 heater in the model	
Single reconfiguration	Not recoverable due to the fault occurrence of H2	Recoverable	Not recoverable due to the fault occurrence of H2	Recoverable	Not recoverable due to the fault occurrence of H1	Not recoverable due to the fault occurrence of H1
Double reconfiguration	Recoverable	Recoverable	Recoverable	Recoverable	Recoverable	Recoverable

Figure 9: Results for fault recovery analysis.

Model	Complexity	Time
sensorfilter_2	2	4.05
sensorfilter_3	3	16.96
sensorfilter_4	4	74.21
sensorfilter_5	5	273.43
sensorfilter_6	6	861.69
sensorfilter_7	7	2677.01

Table 2: Scalability of the probabilistic engine. The complexity is measured as the degree of redundancy.

The probabilistic engine is computationally heavier. Our results with the sensorfilter benchmark (see Table 2) have shown that the computation time increase exponentially with the model's complexity.

The nature of the algorithms used by the toolset show that the computation time increases exponentially with the model's size. The evaluation by the industrial partner has however shown that the current engines are fast enough to allow for mildly complex models, provided they are modeled with reasonable care.

6 Conclusion

We introduced a comprehensive and coherent methodology for the design of complex safety-critical systems. Our approach is applicable to any domain where, e.g., timing, system performance and safety are at stake. Examples are avionics, transportation, including railways and automotive, power and the medical domain. The methodology is based on a formal specification language that is derived from AADL and its Error Model Annex. It supports a variety of V&V activities such as consistency analysis, simulation, correctness verification, performability evaluation, dynamic fault tree generation, FMEA table generation, FDIR, and diagnosability analysis. These are implemented by a graphical toolset which integrates state-of-the-art analysis and verification tools.

An industrial partner evaluated the methodology using their own case studies has taken place as part of the COMPASS project. They have shown that AADL provides enough expressiveness to model all the hardware and software subsystems in satellite avionics. They found that the hierarchical structure and the component-based paradigm enabled them to reuse models from one satellite to another. Also the approach supported their incremental modeling approach. The results provided by the RAMS analyses allowed the evaluation of design alternatives. An issue was however scalability. It is foreseen that larger models come with increased computation time. A follow-up research project has been approved by the European Space Agency to tackle this issue.

A comprehensive report of the COMPASS project can be found in [10]. It describes the modeling language, its semantics, the toolset and the supported analyses in more detail.

References

[1] Architecture Analysis and Design Language (AADL) V2. SAE Draft Standard AS5506 V2, International Society of Automotive Engineers, March 2008.

[2] Architecture Analysis and Design Language Annex (AADL), Volume 1, Annex E: Error Model Annex. SAE Standard AS5506/1, International Society of Automotive Engineers, June 2006.

[3] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *Int. J. on Software Tools for Technology Transfer*, 2(4):410–425, 2000.

[4] J.-P. Katoen, I.S. Zapreev, E. M. Hahn, H. Hermanns, and D.N. Jansen. The ins and outs of the

probabilistic model checker MRMC. In *Quantitative Evaluation of Systems (QEST)*, pages 167–176. IEEE CS Press, 2009.

[5] Lars Grunske. Specification patterns for probabilistic quality properties. In Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors, *ICSE*, pages 31–40. ACM, 2008.

[6] M. Dwyer, G.S. Avrunin, and J.C. Corbett. Patterns in property specifications for finite-state verification. In *Int. Conf. on Software Engineering (ICSE)*, pages 411–420. IEEE CS Press, 1999.

[7] I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In *Design Automation Conference (DAC)*, pages 821–826. ACM, 2006.

[8] M. Bozzano and A. Villafiorita. The FSAP/NuSMV-SA Safety Analysis Platform. *Int. J. on Software Tools for Technology Transfer*, 9(1):5–24, 2007.

[9] A. Cimatti, C. Pecheur, and R. Cavada. Formal verification of diagnosability via symbolic model checking. In *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 363–369. Morgan Kaufmann, 2003.

[10] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. Safety, dependability, and performance analysis of extended AADL models. *The Computer Journal*, doi: 10.1093/com, March 2010. `http://comjnl.oxfordjournals.org/cgi/reprint/bxq024?ijkey=JJ1GiN0GnkxUCGV&keytype=ref`.