Timed Formal Model and Verification of Satellite FDIR in Early Design Phase

Alexandre Albore, Silvano Dal Zilio, Marie de Roquemaurel, Christel Seguin, Pierre Virelizier

Timed Formal Model and Verification of Satellite FDIR in Early Design Phase

Alexandre Albore[1,2,4], Silvano Dal Zilio[2], Marie de Roquemaurel[1,3], Christel Seguin[4], Pierre Virelizier[1,5]

1. Institute of Research and Technology (IRT) Saint-Exupéry, Toulouse, France
2. LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
3. AIRBUS Defence and Space, 31 rue des Cosmonautes, Toulouse, France
4. ONERA, 2 Avenue Edouard Belin, Toulouse, France
5. SAFRAN Tech, Rue des Jeunes Bois, Magny-Les-Hameaux, France

Regular Paper Abstract[1]

In a previous work, we proposed an extension of the AltaRica language and tools to deal with the modelling and analysis of failures propagation in presence of timed and temporal constraints. This need is crucial in the space industry, where safety functionalities raise new challenges for the early validation of systems during model conception. This paper focuses on the application of our approach to the Failure Detection Isolation and Recovery (FDIR) mechanisms of the Attitude and Orbit Control System (AOCS) of a satellite. We discuss the modelling methodology applied to this system and its properties, as well as the tractability of the model-checking analysis.

1 An Expression of Industrial Needs and Requirements

Failure Detection, Isolation and Recovery (FDIR) functions are implemented aboard satellites in order to detect the occurrence of failures and to prevent the failure from propagating in the whole system, which could cause critical events and thus jeopardize the mission.

The complexity of the FDIR verification and validation (V&V) increases with the system complexity. As systems include more and more interacting functions and functional modes, it becomes harder to evaluate at the overall system level the effects of local failures. That is even truer when we consider the effects of time. Indeed, unexpected behavior can arise from a bad timing of events. Timing constraints allow to represent the impact of computation times, delays on the propagation of failures, and the reaction time of the reconfigurations steps triggered in reaction to failure detection.

Thus, new models and tools are needed to assist early V&V of the on-board processes and FDIR design. Thomas and Blanquart [1] describe a process to model FDIR satellite functions. This process requires to model failure propagation, detection, and recovery times, which requires modelling languages expressive enough to support complex system modelling. Associated tools should provide automatic verification that on-board FDIR functions are correct despite latency in failure

We thank Jean-Paul Blanquart of AIRBUS Defence and Space for his kindness in discussing the FDIR model, and the evaluation of requirements for AOCS mode during the assessment of the scalability of the approach.

propagation or management. For instance, one should be able to verify whether failures are recovered before they are propagated further in the system.

2 Timed AltaRica

To match those needs, we describe a recent extension of the AltaRica modelling language, and an associated formal verification technique that supports model-checking verification of temporal functional properties [2]. Our approach is based on an extension of the AltaRica language where we can associate timing constraints with events.

AltaRica is a high level modelling language defined to ease the modelling of failure propagation in the systems and the computation of the possible combination of component failures that lead to an unwanted situation. In AltaRica, a system is expressed in terms of variables constrained by formulas and transitions. In this work, we consider the AltaRica 2.0 Dataflow language, which is sufficient to analyze the behavior of computer based systems [3]. An AltaRica model is made of interconnected *nodes*. A node can be essentially viewed as a mode automaton [4] extended with guards and actions on data variables.

We introduced the capacity to express complex timing constraints by extending the limited mechanism already present in AltaRica, which relies on the use of a *Dirac(x) distribution* that encodes a determinist transition that shall be triggered with the highest priority and with observable effect after x time units. On the opposite, the timing constraints introduced in our extension of the language, Timed AltaRica, allows to associate a time interval to a given guard, say $[a, b]$, meaning the transition can be (non-deterministically) triggered if it has been enabled for a duration δ with $\delta \in [a,b]$. A main difference with the original semantics of AltaRica is that the timing constraint on an event is not reinitialized unless its guard becomes false. We can use a real-time model-checker on these extended models by automatically translating Timed AltaRica specifications into the Fiacre pivot language [5]. The proposed Timed AltaRica extension matches well the Timed Petri Net formalism underlying Fiacre.

Fiacre is a high-level, formal specification language designed to represent both the behavioral and timing aspects of reactive systems. Fiacre programs are stratified in two main notions: *processes*, which are well-suited for modelling structured activities (like, for example, simple state machines), and *components*, which describe synchronization between a composition of processes. After compilation, the Fiacre code can be checked using Tina [6]. The core of the Tina toolset is an exploration engine that can be exploited by dedicated model-checking and transition analyzer tools. Tina offers several abstract state space constructions that preserve specific classes of properties like absence of deadlocks, reachability of markings, or linear and branching time temporal properties. These state space abstractions are vital when dealing with timed systems that generally have an infinite state space (due to the use of a dense time model). In our experiments, most of the requirements can be reduced to reachability properties, so we can use optimized on-the-fly model-checking techniques.

These tools are particularly interesting in the context of FDIR V&V. For instance, they enable the evaluation of the max convergence time of system elements after FDIR and the computation of "timed cuts" leading to undesired events.

3 AOCS case study

In order to assess our approach, we apply it to an industrial case study, namely the validation of the automatic mode management model of an AOCS (Attitude and Orbit Control System) of a satellite. In space missions, the control of the attitude and the orbit of the aircraft is delegated to the AOCS.

In the case study, the system relies on

- AOCS sensors:
  - Inertial Measurement Unit (IMU) implements accelerometers which provide the acceleration and gyros which give the estimated angular speed;
  - Star-TRackers (STR) gives the estimated attitude quaternion;
- AOCS actuators: Thrusters (ColdGasProp) permit the control of the spacecraft;
- The On-Board Computer unit, which manages all the spacecraft's activity and therefore the AOCS application software that acquires the information from the sensors and commands the actuators.

Several AOCS modes are designed for the different mission phases (Fig.1).

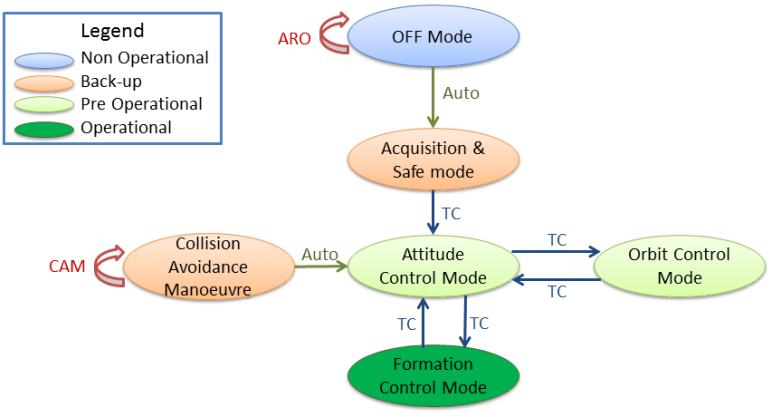

Figure 1. AOCS modes

Switching between a mode and another depends on three possible reasons: either the satellite receives a telecommand (TC) from Earth; or there is an automatic on-board transition (Auto); or an automatic FDIR reconfiguration order (ARO) is triggered on-board. Transition between modes is possible only when the involved equipment is available. In fact, some devices can take tenths of minutes to be ready for a mode switch. This is why it is very important to take into account timing constraints.

It is easy to understand that the planning and the reactivity to messages triggering a transition between modes is crucial, as failing to detect a message or reconfiguring in time would yield an unwanted mode. For instance, it could result in a transition into Safe Mode, which entails a heavy reboot and loss of scientific acquisition time. Many tasks of the mission depend on different inputs and are time-dependent; the temporal bounds of those tasks are the ones that we will analyze through formal V&V

in order to detect possible unwanted situations, or undetected failures through the system, in order to maintain the safety and mission autonomy objectives.

Figure 2 displays the AOCS mode automaton considered in the case study. This is a more precise version of the automaton of Fig. 1 based on a SysML activity diagram that represents the transitions between mode and, for each AOCS mode, its associated state-machine. Each AOCS mode has essentially three states: an initial; a nominal; and a "degraded" state.

Figure 3 also displays the state machine of the IMU, ColdGasProp and STR equipment which have similar state machines. Equipment which is started from an "OFF" state, can only reach an "ON" state by passing through a "Starting" state. When a failure happens, the equipment makes a transition to state "Failed". If the failure is permanent, the equipment passes on to the redundant equipment. If it is temporary, state "Starting" can be restored with no further consequences. The allocation of equipment on modes is given by Table 1. For example, in our study case, the STR is used in four modes.

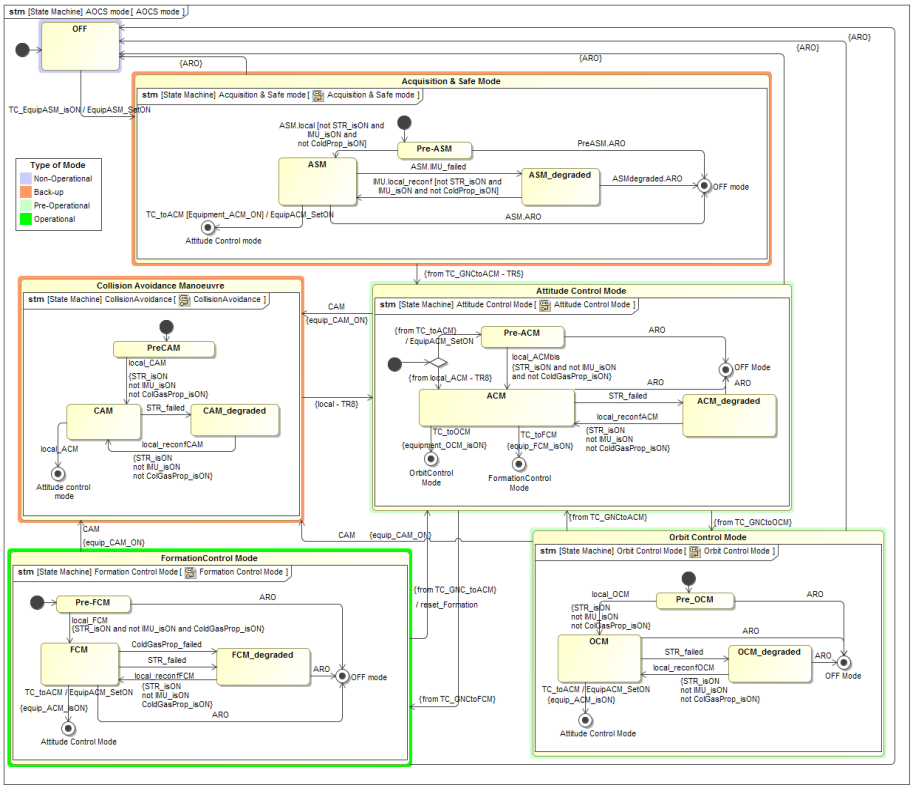

Figure 2. AOCS mode automaton

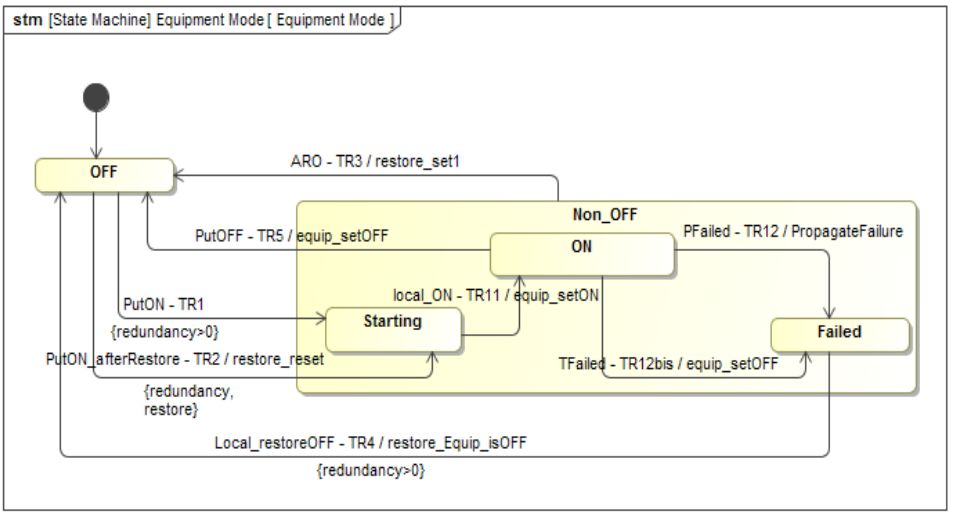

Figure 3. Equipment state machine

Legend ↗ Allocate	Acquisition & Safe mode	OFF	Attitude Control Mode	CollisionAvoidance	Formation Control Mode	Orbit Control Mode
1- Equipment	1		1	1	2	1
Cold Gas Propulsion					↗	
IMU	↗					
STR			↗	↗	↗	↗

Table 1: Allocation of the equipment on the modes.

The AOCS, as all satellite systems, presents an increasing complexity of interactions between functions and other equipment, often raising design and integration issues during the product final testing and verification phase. Besides, correcting these issues often generates a heavy rework and is a well-known cause for cost overruns and project delays. Moreover, the operation of space systems is traditionally informally expressed in design documents as a set of modes representing functions, equipments and monitoring mechanisms. Usually the mode dynamics of all the satellite systems and their interaction with the FDIR functions are validated by review, i.e. exhaustive manual (opposed to automatic) consistence checks and cross-reading. In case of formation flying, complex equipment and instruments are distributed over several spacecrafts. The human validation activities become practically impossible due to the high number of combinations to be analyzed. Powerful computer-aided analysis techniques are expected to help overcoming this issue.

FDIR functions validation is particularly difficult, especially with “traditional approaches” because of the large number of interactions and of situations; moreover these situations are not nominal, increasing the difficulty to analyse them; a large variety of points of view is necessary to fully analyse the behaviour of FDIR and its impact on dependability properties, including to be complete an explicit representation of the entities handled by FDIR: architecture, faults, time, etc.

We believe that formalizing and validating the specifications through animation/simulation and model-checking has several strong advantages. On the one hand, modelling during the specification phase forces the designer to formalise and clarify the specifications. Animation/simulation is useful for validating the model against the specifications and for identifying behaviour inconsistencies based on relevant user-defined scenarios. Such inconsistencies are difficult to identify in a classical purely paper-based specification process. Last, formal verification proves that none of the possible execution scenarios violates the system properties.

4 Case study modelling

The original model of the AOCS case study comes from the validation of the FDIR of formation flying satellites. Formation flying requires specific techniques to ensure flight coordination and the safety of the spacecraft in case of anomaly. This requires giving more autonomy and complex decision-making mechanisms to the On-Board Computer unit. The SPACIFY project [7] described a similar architecture using the Synoptic language, in order to identify the needs in terms of guaranteeing traceability, validation, and general analysis – possibly using formal techniques – of the V-cycle of development [8]. From this architecture, an AltaRica 1.0 version of the Synoptic model was generated automatically. We have used this initial AltaRica model and adapted it first to the AltaRica Dataflow syntax [3]. Then we can automatically translate this new model into Fiacre using our toolchain [2].

AltaRica modelling process

In order to validate the FDIR software, the AltaRica model shall represent both the FDIR logic and the failure propagation in the hardware platform that is monitored and reconfigured according to the FDIR logic. During earlier design phase, AltaRica models without timing constraints can be used to abstract the details of the failure propagation paths and to support a preliminary safety and dependability analysis. Then the model details can be refined and timing information can be introduced in timed Altarica. This will help system designers to verify critical properties when time-bounded reactions are required. On the application side, focusing on the interaction of FDIR mechanisms and AOCS modes is of an utmost importance when designing a space system.

In this case study, we considered a quite detailed view of the FDIR logic related to the management of the AOCS and a simplified view of the satellite hardware. Next study will consider more detailed view of the satellite hardware architecture.

Details of the model

Regarding the modelling activity, we focused first on the methodology applied to timed failure propagation described for a multi-phase system. In the starting architecture, described in Sect. 3, each satellite has three separate kinds of equipment: the STR (with a redundancy of 3), the ColdGasProp (with a redundancy of 3), and the IMU (with a redundancy of 2). We assume that mode transitions are initiated instantaneously (with a transition associated to a Dirac law), while events related to equipment “switching on” or rebooting are associated with a time law as follows:

- STR takes 30 minutes, it is associated with the interval [30,30]
- IMU takes less than 10 seconds, it is associated with the interval [0,0.1]
- ColdGasProp takes between 5 and 10 minutes, it is associated with [5,10]

We consider that these times include all physical delays. The notification of a detected failure from Surveillance is a timed event, ranging between 0 and 10 ms.

It is worth noting that the Synoptic (and later, the AltaRica 1.0) models that we have had access to, make practically no use of flow variables, which are variables that

represent the ports' input/output of components. This modelling style enforces that either communication are simultaneously performed both by sender and receiver or communication cannot occur at all. Moreover, this has a great infñluence on the empirical evaluation we performed, as undesired states are avoided by the modeller, while one of the interests of our approach resides in speeding-up the early design evaluation phase by outlining design errors and allowing an automated assessment of the model. As the design of the AOCS mode has no such flaws, we aimed at checking the invariants, in order to validate the model, and to evaluate the scalability of the approach on the former properties plus a temporal one.

5 Empirical evaluation

In general, real-time model-checking does not scale well on very detailed and large systems, especially when the system uses events that work on very different timescales. Nevertheless, failure propagation models can be large but not very detailed in early design phase. In this particular use case, we are able to check the smallest instances of the problem in less than 3 minutes on a typical laptop; while the most difficult problem can be checked in half an hour.

We propose different versions of the AOCS architecture in order to appraise the complexity of the case, and the scalability of our toolchain. We apply a series of simplifications to the model, generating different benchmarks of growing complexity. The main parameters in our experiments will be the number of replicas of each equipment; and the possibility, or not, to have transient failures.

A first simplification of the model is to consider permanent failures only. Benchmarks done in this condition are denoted "*Pfail only*" (see Table 2). This choice simplifies the model-checking problem since it discards loops in the behavior of the system that originate from the system rebooting through its "Starting" state. On the other hand, we can build instances that are more complex by increasing the number of equipment. For example, we can add a second kind of thruster. In Table 2, we label each experiment with the number of different kind of thrusters used.

For each configuration, we investigate two different kinds of properties.

A first set of properties contains invariants on the set of states reachable by the system, like the absence of deadlocks or the property that "*equipment STR is always OFF when the AOCS is in Acquisition & Safe Mode*". These are typically referred to as *logical safety properties*. Both examples of properties, when true, require enumerating all the possible states of the system. Therefore, they give a good estimate of the complexity and size of the problem. Since reachability properties do not require to compute the possible transitions between two states (to compute the *state graph*), it is possible to use very efficient on-the-fly techniques that are both space and time-efficient.

Then we check an example of timing property, namely we prove a bound on the maximal time it takes for the system to reach a safe mode after a particular event. More precisely, given a duration δ, we check whether the AOCS can reach its "Collision Avoidance" mode in less than δ minutes after activating surveillance. We can check this kind of properties by adding an "observer" component that monitors the time elapsed since an event occurred and that can raise an error after a timeout. The observer adds extra behaviors to the analysis of the initial system and can therefore significantly increase its state space, especially when there is a lot of non-determinism in the system or when there are long-running activities. We observed that our use case exhibit partially these two causes of state space explosion, limiting the verification of timed properties to the one thruster benchmarks (in 640s for the "Pfail only", and 44mn54s for the "Pfail and Tfail"), within the time-cut we set at 45mn. This can be explained by the fact that satellite systems are usually very deterministic. Indeed, operators need to plan the behavior of a satellite very precisely

when they compute TCs. This is an encouraging observation for the tractability of our approach in the aerospace domain.

Model	states	transitions	time (s)	size (MB)
1 thruster Pfail only	224 374	8 345 295	225	8
1 thruster PFail and Tfail	448 335	20 470 486	390	16
2 thrusters Pfail only	4 003 939	207 594 548	1 303	189
3 thrusters Pfail only	-	-	-	-

Table 2: Empirical evaluation of state space generation, Intel Xeon @ 2.33GHz

Table 2 gives the results obtained with our experiments on four different configurations. We record the size of the model (in number of states and number of transitions) as well as the execution time and the memory consumed. All these results where obtained on a typical laptop with 8 GB of memory and an Intel processor. We observe that the Tina model checker scales up well on these models, failing at the biggest instance, not producing results after 45mn of run-time. These numbers give a good estimate of the complexity of checking safety properties.

A little less than 10 years ago, a similar study [8] was performed using models expressed in AltaRica 1.0 and two different model-checkers, ARC [9] and MEC 5 [10]. These are two tools developped specifically for the AltaRica language. Like with the experiments reported in Table 2, ARC and MEC were used to compute the set of reachable states from an initial configuration of the system. These tools are based on symbolic methods for representing the sets of states and transitions of an AltaRica model. This is usually more efficient than enumerative techniques, such as those used by Tina. On the other hand, neither ARC or MEC take inherently into account timing constraints; whereas Tina relies on a “symbolic” representation of time constraints. Therefore they need to model time using an explicit clock (an integer variable) that should be updated in the model. ARC and MEC both bumped into serious limitations when scaling-up the models, even when simple invariants were checked [8].

It is worth noting that, in our experiments, the addition of timed transitions did not increased the number of reachable states and only increased the execution time by a factor of 4.

Even if in our case study no counterexamples were generated because of the correctness of the proposed models, our toolsuite permits, in an untimed model, to compute cutsets and countreexamples showing a timed scenario where the safety property checked is violated [2]. Such couterexamples are generally not easily readable and difficult to debug. In the case of an implementation with FIACRE, it is possible to exploit relations between models representing the information required by the user on the one hand, and information produced by the tools, on the other hand to visualize in a compact view both the outcome of the model-checker and the FIACRE model, which facilitates greatly the interpretation of the analysis process [11]. A possible future work would be of applying a similar approach to allow the representation of the analysis directly in the AltaRica initial model.

6 Related works

Several model-based approaches have been proposed, each with their associated tooling, in order to cope with the complexity of analyzing sophisticated safety

architectures and scenarios. However, rare are the languages that come equipped with tools that can perform formal analysis of timed models.

Figaro language [12] is an alternative language for failure propagation model which introduces time via stochastic events, while we express temporal constraints on the triggering of events. Figaro models can be analyzed with stochastic simulators which are relevant to assess performance of the system (e.g. an estimation of the false detection rate). As far as we know, there is no translation from Figaro to timed model checking tools in order to verify whether a software logic satisfies applicable determinist timed requirements.

Several works have combined model-checking and AltaRica. The archetypal example is the MEC tool [10] that was developed at the same time as the language. More recently, Bozzano et al. [13] have defined a transformation from AltaRica Dataflow to the symbolic model-checker NuSMV. While this tool does not support complex timing constraints, it offers some support for Dirac laws (and implicit priorities) by encoding an ad-hoc scheduler.

COMPASS uses the SLIM language, a subset of AADL, for modelling safety architectures. It can automatically generate fault trees, which can be evaluated to determine the probabilities of failures. The FDIR design process and analysis relies, for describing temporal events on fault propagation, on the so-called TFPM (Timed Failure Propagation Models) [14].

Validating a FDIR approach in satellite architecture has been done in project AGATA [15] by coupling simulation with model-checking, the latter to prove that some given logical or timing properties hold in all states of a scenario generated by simulation. In AGATA the choice went in the same direction as us: focusing significant part of the system and abstracted away the rest of it using UPAAL [16]. However, the model they performed model-checking was un-timed, with several variables used as clocks to track time, with the drawback that the size of the graph is exponential in the number of clocks [17], when Fiacre/Tina relies on a single clock.

7 Conclusion

The use of timed Altarica will help system designers to verify critical properties when time-bounded reactions are required. On the application side, it is central to use formal methods, e.g. model-checking, to validate properly formalized specifications. Forcing the designer to produce early well formed models during the specification phase yields to have the specifications formalised and pinned down for the next development phases. Then, the validation of those specifications in the model allows to identify behavioral inconsistencies. Espacially, such inconsistencies are difficult to identify in a classical, purely paper-based, specification process. Last, formal verification proves that none of the possible execution scenarios violates the system properties. Such approaches are useful not only for validating an architecture or FDIR strategy once defined, but also for tuning its parameters during the conception phase. Results from the early V&V of the on-board processes and FDIR design can be fed back into the requirements analysis phase. The contribution of proper (formal) analysis tools to the complex system designs helps in reducing the specification phase, and the early validation of requirements and models. Such extended V&V toolbox will eventually consent to reduce the time cost of large projects, helping the development of new spacecraft technologies.

References

[1] D. Thomas and J.-P. Blanquart, "Model-based RAMS & FDIR co-engineering at Astrium Satellites," in *Data System In Aerospace (DASIA)*, 2013.

[2] A. Albore, S. Dal Zilio, G. Infantes, C. Seguin and V. P., "A model-checking approach to analyse temporal failure propagation with AltaRica," *Model Based Safety and Assessment (IMBSA) - LNCS,* vol. 10437, pp. 147-162, 2017.

[3] A. Rauzy, " Altarica Dataflow language specification version 2.3," École Centrale de Paris, June 2013.

[4] A. Rauzy, "Mode automata and their compilation into fault trees," *Reliability Engineering and System Safety,* 2002.

[5] B. Berthomieu, J.-P. Bodeveix, P. Farail, H. Filali, M. Garavel, P. Gaufillet, F. Lang and F. Vernadat, "Fiacre: an intermediate language for model verification in the topcased environment," in *Embedded Real Time Software and Systems (ERTS)*, 2008.

[6] B. Berthomieu, P. Ribet and F. Vernadat, "The tool Tina – construction of abstract state spaces for Petri Nets and Time Petri Nets," *International Journal of Production Research,* vol. 42, no. 14, 2004.

[7] A. Cortier, L. Besnard, J. P. Bodeveix, J. Buisson, F. Dagnat, M. Filali, G. Garcia, J. Ouy, M. Pantel, A. Rugina, M. Strecker and J. P. Talpin, "Synoptic: A Domain-Specific Modeling Language for Space On-board Application Software," in *Synthesis of Embedded Software*, 2010.

[8] G. Sutre, F. Herbreteau and E. Fleury, "Traduction de Synoptic en AltaRica," LaBRI, 2009.

[9] G. Point, A. Griffault and A. Vincent, AltaRica Checker Handbook - A user-guide to ARC version 1, Talence: LaBRI - CNRS UMR 5800 - Univ. de Bordeaux, 2010.

[10] A. Griffault and A. Vincent, "The MEC 5 model-checker," in *International Conference on Computer Aided Verification*, 2004.

[11] F. Zalila, É. Jenn and M. Pantel, "Model Execution and Debugging - A Process to Leverage Existing Tools," in *5th International Conference on Model-Driven Engineering and Software Development*, 2017.

[12] M. Bouissou, "Automated dependability analysis of complex systems with the KB3 workbench: the experience of EDF R&D," in *International Conference on ENERGY and ENVIRONMENT (CIEM)*, 2005.

[13] M. Bozzano, A. Cimatti, O. Lisagor, C. Mattarei, S. Mover and M. Roveri, "Symbolic model-checking and safety assessment of AltaRica models," *Electronic Communications of the EASST,* vol. 46, 2012.

[14] B. Bittner, M. Bozzano, A. Cimatti, R. De Ferluc, M. Gario, A. Guiotto and Y. Yushtein, "An Integrated Process for FDIR Design in Aerospace," in *International Symposium on Model Based Safety and Assessment (IMBSA)*, Trento, 2014.

[15] A.-E. Rugina, . J.-P. Blanquart and R. Soumagne, "Validating Failure Detection Isolation and Recovery Strategies using Timed Automata," in *12th European Workshop on Dependable Computing (EWDC 2009)*, Toulouse, 2009.

[16] K. G. Larsen, P. Pettersson and W. Yi, "UPPAAL in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT),* vol. 1, no. 1, pp. 134-152, 1997.

[17] C. Daws and S. Yovine, "Reducing the Number of Clock Variables of Timed Automata," in *17th IEEE Real-Time Systems Symposium*, Washington, DC, 1996.