Integration of formal fault analysis in ASSERT: Case studies and lessons learnt

P. Bieber, J Blanquart, G. Durrieu, D Lesens, J Lucotte, F. Tardy, M Turin, C. Seguin, E. Conquet

Integration of formal fault analysis in ASSERT: Case studies and lessons learnt

P.Bieber[6], JP.Blanquart[1], G.Durrieu[6], D.Lesens[2], J.Lucotte[5], F.Tardy[3], M.Turin[4], C.Seguin[6], E.Conquet[7]

1: Astrium Satellites, 31 rue des cosmonautes, F-31402 Toulouse (jean-paul.blanquart@astrium.eads.net)
2: Astrium Space Transportation, route de Verneuil, F-78133 Les Mureaux (david.lesens@astrium.eads.net)
3: Dassault Aviation, 78 quai Marcel Dassault, F-92552 Saint-Cloud, (frederic.tardy@dassault-aviation.fr)
4: GTI6, 33 avenue Kennedy, F-91300 Massy, (michelturin@gti6.com)
5: INSA Lyon, 8 rue de la physique, F-69621 Villeurbanne, (jocelyn.lucotte@insa-lyon.fr)
6: ONERA Centre de Toulouse, 2 av. E.Belin, F-31055 Toulouse (pierre.bieber; guy.durrieu; christel.seguin@cert.fr)
7: ESA, Keplerlaan 1, AG Noordwijk, NL-2200, (eric.conquet@esa.int)

Abstract: The ASSERT European Integrated Project (Automated proof-based System and Software Engineering for Real-Time systems; EC FP6, IST-004033) has investigated, elaborated and experimented advanced methods based on the AltaRica language and support tool OCAS for architecture and fault approach propagation description analysis, and integrated in the complete ASSERT process. The paper describes lessons learnt from three case studies: safety critical spacecraft, autonomous deep exploration spacecraft, and civil aircraft.

Keywords: Dependability, Space, Formal Methods

1. Introduction

Dependability engineering is based on an appropriate combination of fault prevention, removal, tolerance and analysis techniques. The former three ones received a lot of attention in advanced engineering processes and methods. Conversely in the industrial practice, fault analysis is still largely based, with poor tool support, on tedious and error prone intellectual investigation of possible fault propagation paths. This is then used as input to detailed analyses (such as fault trees or quantitative dependability computation), generally with some tool support, or simply documented, for instance in failure modes, effects and criticality analysis reports.

The ASSERT European Integrated Project (Automated proof-based System and Software Engineering for Real-Time systems; EC FP6, IST-004033) has investigated, elaborated and experimented advanced methods based on the AltaRica language and support tool OCAS for architecture and fault approach propagation description analysis, and integrated in the complete ASSERT process.

The paper presents first the concepts studied when performing an early fault propagation analysis of an on board system. Then, it presents how the concept and analysis can be supported by AltaRica models and the associated tools. An illustration on the ATV system is given on the next part. Finally, the paper describes first lessons learnt from three case studies (safety critical spacecraft, autonomous deep exploration spacecraft, and civil aircraft). This sample of real cases of critical embedded systems enables to highlight a variety of architecture and dependability requirements. Moreover, they address different scenarios of use of fault propagation models: specification of fault detection isolation and recovery architectures, dependability assessment of architectures, comparison of architectures.

This last part discusses first the benefits and limitations of the AltaRica language and tools for each targeted scenario. Then it deals with process issues: clarification of the various dependability modelling levels (from functional system view to physical one); links with the other methods and tools of the complete process (links with engineering and validation models such as AADL or UML, and with means to ensure and maintain consistency such as model transformation techniques).

2. Fault propagation analysis for on board systems

At system level, dependability analysis is certainly one of the most difficult tasks required by customer according to the criticality of the system. The objectives are to identify the possible faults, their effects at subsystem and system level, specify the protection mechanisms, design a robust architecture fault tolerant, design the fault detection isolation and recovery (FDIR) mechanisms and implement them correctly in the on-board software, define operations with ground segment to cope with all identified failure cases and validate them. According to the criticality of the feared failure condition, the system shall be tolerant to 1 or 2 faults. The proof of this requirement is very difficult due to hidden common cause failure.

At development level the on-board FDIR function and ground operations are always defined very late in the development cycle, but they have often a large impact on software development and are the source of planning slippage and over-cost.

The trends are now to have vehicle more autonomous, with automatic functions: aircraft without manned control, spacecraft autonomous from ground station during long periods, spacecraft constellation, deep space mission without capability of ground control due to the round trip delay, launcher fully autonomous during the launch. These new missions in front of dependability problem are a real challenge which will need modern and efficient tools to provide proofs at system engineering level and reliable verification at implementation level.

Within the ASSERT project, solutions were investigated with tools already proven for aeronautic activity which requires a high level of safety including certification. But the problem of space missions is rather different: long mission time up to 15 years with a very high reliability, advance architecture for automated system with high safety during some mission phases, multi-platform cooperation with high autonomy.

Thus dependability requirements and on ground system architectures will be different for aeronautic and space domain. Nevertheless, early fault propagation analysis will, in both cases, cope with the set of concepts that is presented below.

2.1. Failure condition

In Aeronautic, it is defined the important concept of Failure Condition (FC) which could apply to space activity to analyse the dependability (reliability and safety of a system).

The failure condition refers to a combination of failures modes applied to functions of the system under study. We considered the following generic failure modes of functions: total loss, partial loss and erroneous behaviour. The FC may also include conditions that describe the current mode of the system. The FC may be permanent or transient.

A FC has several attributes:

- *SEVERITY* classification, for safety the categories are: Catastrophic, Hazardous, Major, Minor and No Safety Effect.
- *QUANTITATIVE OBJECTIVE* is a failure rate value that can be stated per mission hour, or for a given mission phase. Typical values are 10^{-9}/hour for Catastrophic FC, 10^{-7}/hour for Hazardous. Other categories are dependent on project, reliability objectives, etc.
- *QUALITATIVE OBJECTIVE* describes the number *N* of individual faults which are considered for a given FC: "*No combination of events with less than N individual faults shall lead to FC*" with N = 2 for Catastrophic FC, N=1 for Hazardous and Major FC.

2.2. Failure modes

We will then define the following failure types for propagating failures in the system:

Fail loss: the function is lost; it does not deliver its service, for electrical system its behaviour may be: no power, no signal, inactive…; for data, its behaviour may be: no answer, no valid data, erroneous data but associated with health status not OK.

Fail erroneous: the function is alive, but delivers an incorrect service: data is erroneous but no immediate mean permits to detect the failure or error state. Software error will be in this category except if an exception is raised with processing stop, in such case we will use the fail_loss type. Fail_loss is detected immediately, but Fail_erroneous can be detected if a failure detector is implemented (e.g. threshold detector, comparison with a prediction, voter when at least 3 instances are provided…). Failure detectors have false alarm rate and limited coverage ratio: if failure is detected, fail erroneous type is not propagated to other functions, either the failure is corrected else the fail_loss type will replace fail_erroneous type; if failure is not detected fail_erroneous type is propagated; but if false alarm is detected, the function is declared lost while it is correct.

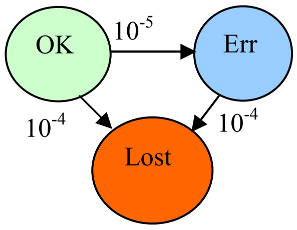
These kinds of failure modes are often identified when analysing fault propagation in computer based system and were more widely discussed in [1].

Fail permanent: the failure remains permanent if no action is taken: the possible actions may be restart power, reset system, or reconfigure; the actions are part of on board FDIR or maintenance operations.

Fail transient: the failure occurs from time to time but does not affect the behaviour of the system permanently; for space electronics, EMC susceptibility and SEU are the main cause of transient failures. For software, it may be a real time problem occurring with a low probability (overload, resource conflict…). Fail transient will propagate inside the system except if filtering mechanisms are implemented (repetition, protection, masking…), but if the failure affects remanent data, fail_transient will propagate as fail_permanent.

2.3. Failure transition and failure rate

Failures are introduced in the system as events which modify the state of the system. The natural state transitions of any item are:



State = OK | - fail_erroneous => State := Erroneous;

State = OK | - fail_loss => State := Lost;

State = Erroneous | - fail_loss => State := Lost;

Figure 1: Failure transition

This latter transition is introduced in order to show the possibility that an erroneous system be further lost, but the reverse transition is impossible: a system already lost cannot return alive without a corrective action as recovery.

For each failure event, we will have to provide the failure rate and the probability law (generally exponential); in this example fail_erroneous has a probability 1E-5 and fail_loss has a probability 1E-4.

2. 4. Failure analysis

The concept of FC can be used for safety analysis (see aeronautic standard): each feared FC is generally specified with a severity, a quantitative objective and a qualitative objective. The analysis will consist to search all failure cases which have the consequence to raise the FC. This search may be obtained by FMEA starting from the failure and going to the effects, or by fault trees starting from the feared FC and searching all failure cases. The severity category of each FC will identify all functions involved in each category, and may have impact on architecture for partitioning software between categories.

The concept of FC can be used for reliability analysis: a system, sub-system or equipment is specified with a reliability figure for a given time period: the whole mission, or part of it. The FC will be quantitatively evaluated taking into account failure rates, and architecture redundancies.

3. AltaRica implementation of the fault propagation analysis

AltaRica is a dependability language issued from LaBri (Laboratoire Bordelais de Recherche en Informatique) and industrial partners. AltaRica has been conceived to formally specify the behavior of critical systems in failure conditions and use these dependability specifications with tools.

Several Dependability tools can process AltaRica models: symbolic simulators, model-checkers, fault tree generators, sequence generators, AltaRica graphical modelers, etc. There are many of such tools. Beyond them, CECILIA workshop covers most of these functionalities through its modules like OCAS and ARBOR.

In this section, we briefly introduce the features of the language and of the tools that were used for ASSERT purpose. The interested reader may find more details in [2].

3.1 AltaRica models

An AltaRica model is a hierarchy of interconnected “nodes”. Each node has typed interfaces (“flows”), internal nominal or error “states” and a behaviour (a constraint automata) that results from “assertions” and “transitions” laws. The interface types usually record the failure modes that can be propagated by a flow. For instance, the node of the figure 2 has one output 0 and two inputs I1 and I2 that can propagate either correct or lost values as defined in section 2. The assertions indicate the computation formulae of output and local variables, which often consist in the description of flows values from states possible values. In the example of the figure 2, the assertion states that the output flow O has a correct value when the node status is correct and all its inputs are correct. Else, the output is lost. The transitions describe the node’s state changes under specific conditions, which follow event occurrences. In the example below, the loss event can trigger the transition only if the function is not already lost.

```
node COMPUTATION_Function2
  flow
    O:COMPUTATION_FailureType:out;
    I1:COMPUTATION_FailureType:in;
    I2:COMPUTATION_FailureType:in;
  state
    Status:COMPUTATION_FailureType;
  event
    Loss,
  trans
    not(Status = lost) |- Loss ->
Status:= lost;
  assert
    O = case {
      ((Status = correct) and
            (I1 = correct) and
            (I2 = correct)) : correct,
      else lost
    };
  init
    Status:= correct;
  extern
    law (<event Loss>) = "exp 1e-4";
edon.
```

Figure 2: Example of AltaRica node

By default, different event names depict independent events (as in fault-trees). One can nevertheless group events thank to the synchronisation feature.

Among other things, this feature enables the modelling of common cause failures.

3.2 Analysis of AltaRica models

During ASSERT, we mainly used three functions provided by OCAS: the interactive simulation, the fault-tree generation and the sequence generation.

The simulation mode of OCAS tool enables the injection of failures and then computes the resulting effects. It permitted to debug the models and give a graphical view of failure propagation in the system.

The fault tree and the sequence generations implement top down investigations. Starting from a FC, they extract respectively a tree or the set of sequences of events length that lead to the FC. The fault-tree generation is applicable only when the event ordering does not impact the FC reachability. The sequence generation is applicable in any case and enable to investigate some temporal aspect of failure propagation.

4. Illustration: ATV (Automated Transfer Vehicle)

4.1. ATV GNC architecture

The ATV is a spacecraft that is in charge of transporting cargo for the International Space Station (ISS).

The analysis described in this section is limited to the Rendez-Vous phase when the ATV docks on the ISS, and it addresses only GNC sub-system and software. The Altarica approach could also deal with interaction of the GNC sub-system with other subsystems as power, thermal, propulsion, docking…but the study is concentrated on software behaviour in presence of hardware failures and software errors.

The GNC architecture of ATV is described in figure 3. It is rather complex, it includes a pool of 3 fault tolerance computers (FTC) in charge of the nominal and survival GNC function running in the Flight Application Software (FAS: 3 instances of the same code in each computer) and a proximity flight safety function (PFS) limited to Collision Avoidance Manoeuver (CAM) triggered during the last Rendez-vous phases in case of GNC failure. The Propulsion Drive Electronics (PDE) are dual duplex (4 equipments), and a large number of redundant sensors are used:

- Gyrometers : 4 equipments able to measure angular speed along 2 axis
- Accelerometers : 3 equipments able to measure each acceleration along 3 axis
- GPS : 2 equipments able to provide vehicle position and speed
- Star tracker : 2 equipments able to provide accurate vehicle attitude
- Videometer : 2 equipments able to provide relative distance to the ISS
- Goniometer (RDV sensor) : 2 equipments able to provide angular deviation during rendez-vous

The FAS processes all these inputs data in the GNC function and delivers the PDE commands. But in order to insure the failure detection, it exists to other independent functions :

- the Flight Control Monitoring (FCM) which monitors the trajectory with videometers and goniometers
- the Gobal Measurement System (GMS) which insures the navigation update from star trackers and GPS

The FTC is a triplex system with votes provided in the FTC management layer (FML), if discrepancies are detected between the 3 FAS instances, the erroneous FTC is masked; in case of a second failure, the FTC pool is disabled, and the PFS takes control of GNC function to perform the CAM. A red button is provided to the ISS crew to start the CAM. The PFS is not completely independent of the nominal GNC, because it uses the same gyrometers but in coarse mode, with independent sun sensors for attitude update. The PFS is architectured with 2 computers (MSU), the first is master the second is slave, the master-slave switch will give control to the second one if the first is detected failed and the CAM continues. After completion of the CAM, the vehicle is pointed toward sun and the control is returned to the FTC which are restarted.

Communication between FTC, sensors and PDE is insured via 4 MIL 1553 bus on which are allocated the equipments. This communal resource will introduce common mode failures between functions, and allocation of equipments on the 1553 bus is not trivial. One FTC computer will be the master of the 1553 bus with possible reallocation in case of failure of this computer.

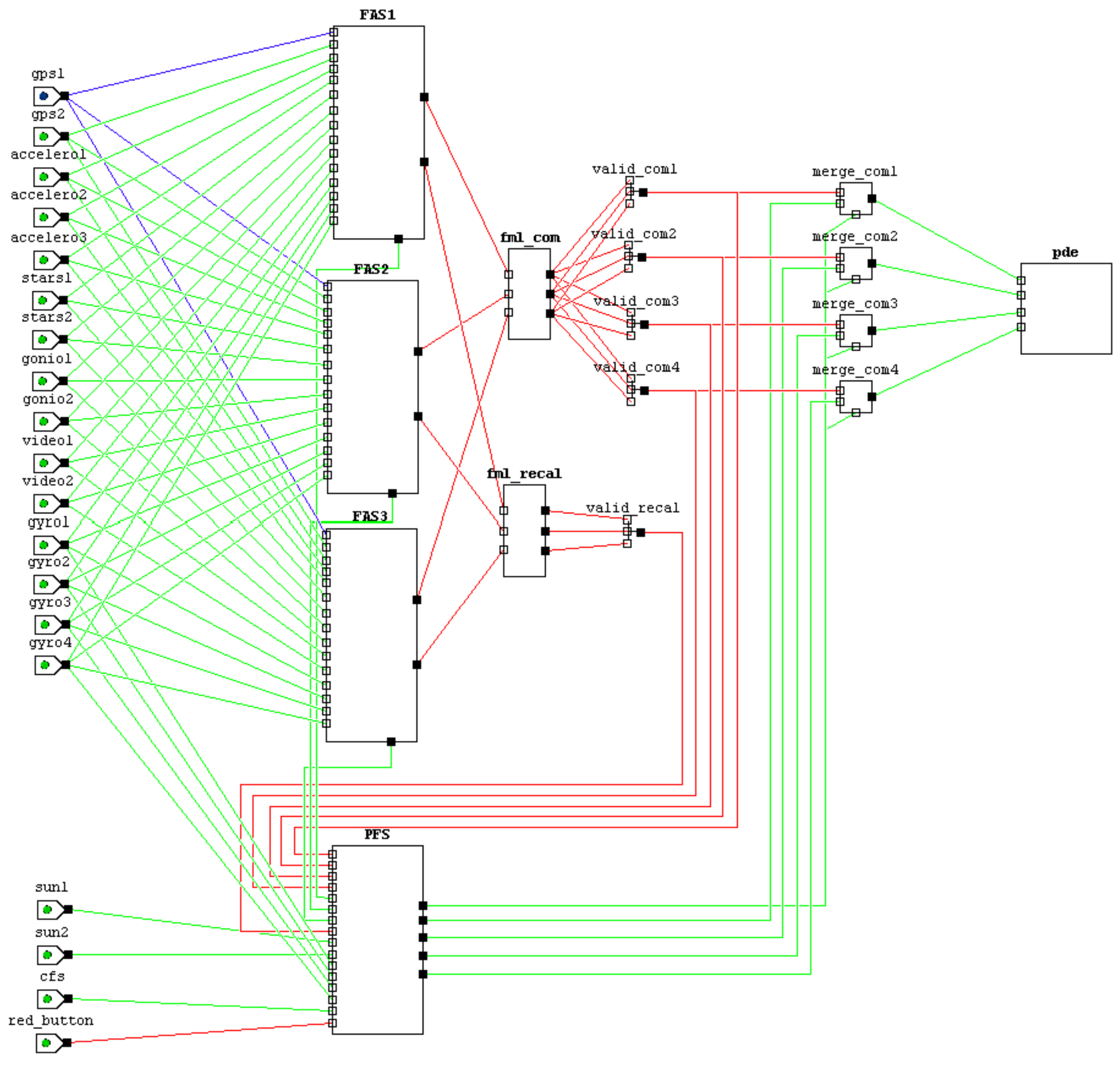


Figure 3: AltaRica model Top-level view of ATV architecture

4.2. ATV GNC Altarica model

The studied Altarica model is a simplification of this ATV architecture, and may include errors, the failure rates are not representative of the real ATV, it was provided only for the purpose of evaluation.

The higher level functional view (described in figure 3) is represented with external interfaces (sensors and commands); failure events are introduced for all external interfaces and each internal function. The links between functions are green in absence of error, red in case of loss of the function and blue in case of erroneous behaviour. Simulations can then be conducted by "injecting" failure events on inputs or functions.

For instance, we study failure modes related with the GPS system used during proximity phase. GPS loss is simple to detect via the GPS health status, and this failure mode will not be analysed in detail. GPS erroneous data is a possible failure mode to be considered and it is not detected internally by GPS receivers. It will be simulated by injecting the fail_erroneous event on one GPS receiver instance of the model. According to the model, the discrepancy between both GPS data is detected (by node valid_gps described in figure 4) and GPS system is declared lost. Then either this failure is blocked by GNC which will continue to control ATV without GPS data, or the failure propagates through GNC function (the output of GNC is equal to lost) and function FCM using independent sensors will detect the GPS failure and will invalidate all GNC commands. In the second case, the GNC health status (HS) is now incorrect and transmitted to PFS to start the CAM manoeuvre.

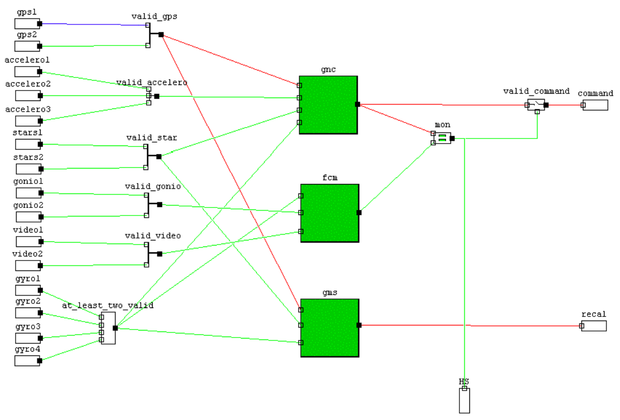


Figure 4: View of the FAS application

Given a Failure Condition, OCAS sequence generator or fault-tree generator are used to compute the set of minimal scenarios of failure events that lead to this FC. For instance, we looked at the following FC of the ATV GNC sub-system:

- "*Lost (resp. Erroneous) primary command* " named PrimLoss (resp. PrimErr), This FC is reached when the output of node valid_com is equal to `lost` (resp. `erroneous`).
- "*Lost (resp. Erroneous) primary and backup command* " names Prim&BLoss (resp. Prim&BErr). This FC is reached when the output of node merge_com is equal to `lost` (resp. `erroneous`).

Scenarios were computed in a few seconds by OCAS. The following table summarizes the results we have obtained.

Results	**Prim**	**Prim&B**	**Prim**	**Prim&B**
	Loss	**Loss**	**Err**	**Err**
1	8	0	0	0
2	13	28	33	0
3	70	622	142	510
4	0	2756	235	3120
proba	8. 1e-5	2.8 1e-9	8.7 1e-9	2.3 1e-12
time	0.03 s	4.37 s	0.21 s	5.11 s

Table 1: ATV Dependability Assessment Results

The first column indicates the safety result type : *1* for the number of single failure leading to the failure condition, *2* for double failures and so on, *proba* is the probability of occurrence of the failure condition, *time* is the amount of time needed by the tools to compute the minimal scenarios leading to the failure condition. The remaining columns provide the results for the four FC considered.

Using this table, we can check that no single failure leads to the Loss of primary and backup command and no double failure leads to erroneous primary and backup command. Thise model can also be used in order to identify the segregation requirements between the various functions, to associate failure rate objectives with component failure modes and to associate Development Assurance Levels with software implemented components.

The model is extended with a description of the hardware architecture and of the allocation of functional components onto this architecture. The hardware architecture is described using altarica nodes that model CPU, buses, ...

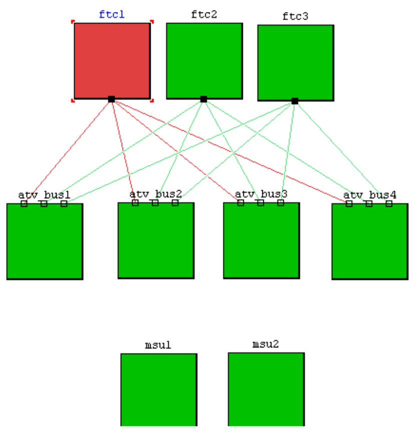


Figure 5: View of the hardware architecture

The allocation of functions on resources of the hardware architecture is described by grouping failure events of functions and resources. For each resource, a common cause failure group is added that simultaneously triggers the failure event of the resource and the failure events of all the functions supported by this resource. This model allows assessing the impact of hardware failures on functions and data flows in order to know whether the hardware architecture contains sufficient redundancy to support the functions, whether an allocation breaks segregation principles, …

5. Lessons Learnt

5.1. Case-studies Lessons Learnt

During the last year of the ASSERT project, the Altarica language and toolset was experimented on several case-studies called Pilot Projects:

- Astrium Space Transportation ATV GNC System
- Dassault Aviation Flight Control System
- Astrium Satellites Pleiades Spacecraft level FDIR System,
- GTI6 studied a temporal redundancy scheme proposed by CNES.

Pilot Projects felt that the Altarica language and associated tools were simple to understand and use. Onera organised for ASSERT industrial partners a one-day tutorial session on model based safety assessment with Altarica and associated tools. This limited training seemed to be sufficient for starting the dependability modelling and assessment activities.

Dassault Aviation case-study showed that once a model of a system under design is built it is easy to perform safety assessment and comparisons of various modifications of the initial design. Dassault studied 5 variants of the Flight Control System.

Astrium Satellites would have liked to be able to describe components that include some parameterization in order to build more generic descriptions of the systems. This is currently not supported by the Altarica language.

The temporal redundancy case-study showed that it was possible to model with Altarica transient failure modes that are much more frequent in the Space domain than permanent failures.

The interactive simulation tool is very interesting both to assess the failure propagation in a functional model and also to assess the impact of hardware components failures at the functional level. Fault tree and Sequence generation tools are efficient. But to be applicable, fault tree generation requires the model to be static (i.e. the chronology of failures has no importance). It could be possible to control the edition of Altarica nodes and models in order to be sure to produce models that can be analysed with the fault-tree generator.

Pilot projects also noticed that, in the space industry, the generation of fault-trees or sequences that combine a big number of failures is not necessarily required to obtain dependability results. In such a context, it might be more valuable to support the generation of FMECA (Failure Modes Effect and Criticality Assessment) tables that is a cumbersome task in industrial projects. These tables detail the effect at a component level and at the system level of all single failures. The table also defines the severity of the failures. Cecilia OCAS offers today a rudimentary tool that generates FMECA tables from an Altarica model. This tool is currently being improved by Dassault Aviation in order to consider its use on space industry projects.

5.2. Integration of failure propagation models into model based development process

The decomposition of failure propagation models into, on one hand, a functional model that is independent of the underlying hardware and, on the other hand, a hardware architecture and an allocation model is consistent with one of the principles of model-based design. One limitation of the tested approach is that services offered by the operating system or the middleware layer such as communication, memory or time management are not modelled. These services are particularly important for Space domain applications that tries to cope with transient failures. Onera has developed approach that includes these services in the scope of the models used to perform safety assessments but, due to the lack of time, the approach could not be tested on industrial case-studies.

A major concern raised by the Pilot Projects with respect to Altarica is related with the effort involved in building the failure propagation models and the possible inconsistencies between Altarica models and other models that are developed according to ASSERT process or, more generally, a model-based development process.

To limit the effort of building dependability models, Onera proposed to develop libraries of Altarica nodes that can be reused from one project to the other. This approach works well for families of systems that do not differ too much. But if a new technology appears and new associated failure modes have to be considered then the old library of components can no longer be used.

To avoid possible inconsistencies with other models developed in the context of ASSERT, Onera investigated a tool that generates an Altarica model from a model described in the AADL language [3]. The tool is based on model transformation technique. It extracts from the AADL model, the functional architecture, the hardware architecture and the allocation. The *property* declarations in the AADL model is used to associate an AADL process with the name of an Altarica node in the existing libraries, this node will be used to generate the Altarica model of the functional architecture.

To be able to generate the behaviour of Altarica nodes an alternative solution could consist into translating AADL code written following the Error Annex standard into Altarica. This seems feasible as LAAS investigated a similar approach during the first phase of the ASSERT project. In this case, the Altarica code might be much more complex than the one we have presented in this report.

Finally we would like to state that integration of dependability models in a model-based process is not limited to tools and languages interoperation. More fundamentally, the integration should define the role of the various kinds of analysis and their relationships.

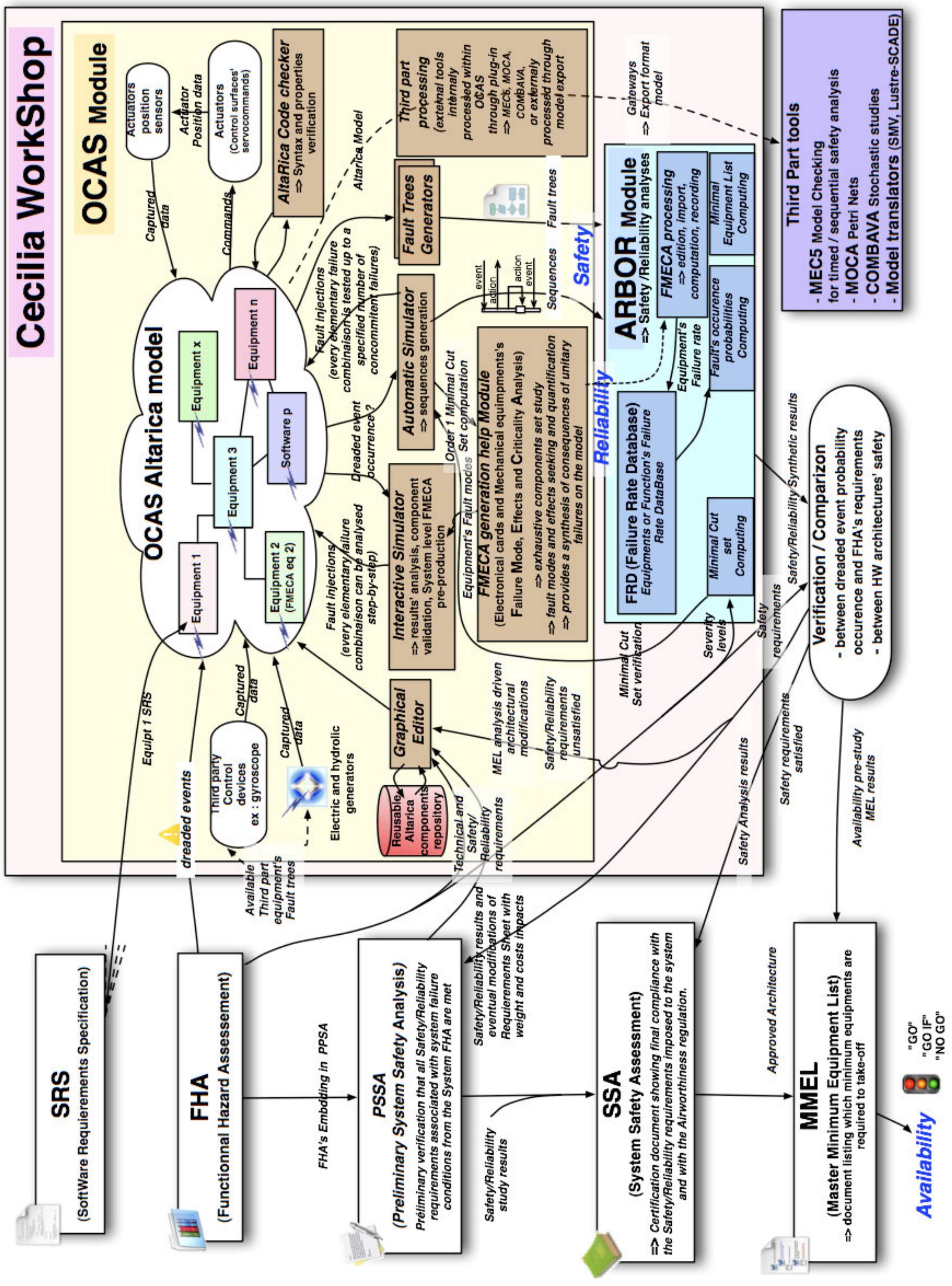
6. References

[1] J. McDermid, D. Pumfrey: "*A Development of Hazard Analysis to aid Software Design*" in Proceedings of COMPASS, 1994.

[2] A. Arnold., A. Griffault, G. Point, A. Rauzy: "*The AltaRica formalism for describing concurrent systems.*" In: Fundamenta Informaticae p109-124, 2000

[3] X. Dumas, C. Pagetti, L. Sagaspe, P. Bieber, P. Dhaussy, *Vers la Génération de Modèles de Sûreté de Fonctionnement*, to appear in proceedings of Conférence sur les Architectures Logicielles, Montréal, 2008

[4] A.E. Rugina AADL Dependability models



Cecilia WorkShop
OCAS Module
OCAS Altarica model
Equipment 1
Equipment 2 (FMECA eq 2)
Equipment 3
Equipment x
Equipment n
Software p
Actuators position sensors
Actuator Position data
Actuators (Control surfaces' servocommands)
Captured data
Commands
AltaRica Code checker => Syntax and properties verification
Altarica Model
Third part processing (external tools internaly processed within OCAS through plug-in => MEC5, MOCA, COMBAVA, or externaly processed through model export
Fault Trees Generators
Automatic Simulator => sequences generation
Interactive Simulator => results' analysis, component validation, System level FMECA pre-production
Fault injections (every elementary failure combinaison is tested up to a specified number of concommitent failures)
Fault injections (every elementary failure combinaison can be analysed step-by-step)
Dreaded event occurrence?
Order 1 Minimal Cut Set computation
Equipment's Fault modes
FMECA generation help Module (Electronical cards and Mechanical equipments's Failure Mode, Effects and Criticality Analysis) => exhaustive components set study => fault modes and effects seeking and quantification => provides a synthesis of consequences of unitary failures on the model
Graphical Editor
Reusable Altarica components repository
Third party Control devices ex : gyroscope
Electric and hydrolic generators
Available Third part equipment's Fault trees
Technical and Safety/ Reliability requirements
MEL analysis driven architectural modifications
Safety/Reliability requirements unsatisfied
Sequences
Fault trees
Safety
Reliability
Gateways => Export format model
ARBOR Module => Safety /Reliability analyses
FMECA processing => edition, import, computation, recording
Minimal Equipment List Computing
Equipment's Failure rate
Fault's occurence probabilities Computing
FRD (Failure Rate Database) Equipments or Function's Failure Rate DataBase
Minimal Cut set Computing
Minimal Cut Set verification
Severity levels
Safety requirements
Third Part tools
- MEC5 Model Checking for timed / sequential safety analysis
- MOCA Petri Nets
- COMBAVA Stochastic studies
- Model translators (SMV, Lustre-SCADE)
Safety/Reliability Synthetic results
Verification / Comparison
- between dreaded event probability occurence and FHA's requirements
- between HW architectures' safety
Safety Analysis results
Safety requirements satisfied
Availability pre-study MEL results
Equipt 1 SRS
dreaded events
SRS (SoftWare Requierements Specification)
FHA (Functionnal Hazard Assessement)
FHA's Embdding in PPSA
PSSA (Preliminary System Safety Analysis)
Préliminary verification that all Safety/Reliability requirements associated with system failure conditions from the System FHA are met
Safety/Reliability results and eventual modifications of Requirements Sheet with weight and costs impacts
Safety/Reliability study results
SSA (System Safety Assessment)
=> Certification document showing final compliance with the Safety/Reliability requirements imposed to the system and with the Airworthiness regulation.
Approved Architecture
MMEL (Master Minimum Equipment List)
=> document listing which minimum equipments are required to take-off
"GO" "GO IF" "NO GO"
Availability

Figure 5: Cecilia^{TM} OCAS integrated Dependability process