Workshop on Simulation and EGSE for Space Programmes (SESP)
26 - 28 March 2019

ESA-ESTEC, Noordwijk, The Netherlands

Charles Lumet[(1)], Régis de Ferluc[(2)], Frédéric Manon[(3)], Rachid Atori[(4)], Bobey Aurélien[(2)]

[(1)]SPACEBEL, Labège, France
Technoparc 8, Rue Jean Bart, 31 670 Labège, France
Email: first.lastname@spacebel.fr

[(2)] Thales Alenia Space, Cannes, France
5 allées des Gabians – BP 9, 06156 Cannes la Bocca Cedex - France
Email: first.lastname@thalesaleniaspace.com

[(2)] CNES, Toulouse, France
18 Avenue Edouard Belin, 31400 Toulouse, France
Email: first.lastname@cnes.fr

[(4)]SPACEBEL, Hoeilaart, Belgium
Ildefonse Vandammestraat 5-7, Hoeilaart Office Park - Building B, 1560 Hoeilaart, Belgium
Email: first.lastname@spacebel.be

INTRODUCTION

During the development lifecycle of satellites or other spacecraft, several engineering modelling levels are used along the way, each dedicated to a specific engineering phase. It goes from system engineering level, to sub-system level, and finally to component (HW & SW) level. In the early phases of a project, the functional analysis and associated logical architecture models are performed at high-level, in order to broadly define the characteristics of the mission. Such high level modelling is needed for instance for orbital mechanics simulators, thermic simulators, electrical systems ... These models are often descriptive and might use domain-related languages or concepts. As the project evolves, more detailed models appear, up to the so-called “operational simulators”, that are intended to be run to precisely mimic stations and satellite behaviour as seen from the control centre.

Since most of the early-phase models mentioned above are descriptive only, they cannot be 'run' to simulate the behaviour of the satellite or subpart they specify. This limitation unfortunately restricts the panel of possible uses for early feasibility analysis required to perform major trade-offs. Due to the use of discipline-specific tools to capture the design during these early phases, there have been no sufficient attempts to use these descriptive models as direct bases in a continuous process aiming at providing all the way a simulator that corresponds to the models.

In this paper, we introduce a practical example of how such a design can be achieved, through model-to-code transformation. Our starting point is a Capella model (Capella is an Open Source solution for model-based systems engineering - MBSE), which describes the high level design of a spacecraft system. This Capella model is extended with domain-specific models capturing the associated failures and their propagation, and the FDIR strategy and its high level objectives. Building on these models, we design a model-to-text transformation which produces a corresponding simulator that can run within BASILES, the CNES simulation platform. This transformation produces a set of SMP2 models containing functional high level code, SMDL artefacts defining required instances, connections, etc., and execution scenarios that reproduces the expected external conditions applied on the satellite. In our case, the model-to-text transformation must also provide the needed additional artefact files that allow running the SMP2 simulation with BASILES. Such a transformation now allows actually simulating the satellite system and therefore provides a quick way of assessing the suitability of an FDIR design with respect to the mission constraints, for instance the initial amount of propellant available or the time frame to observe. We successfully used this model-to-text transformation on a model of the CNES’ Microscope satellite.

By bridging the design model and the simulation model worlds, we open the door to multiple opportunities: on one hand, it allows to bring descriptive models coming from miscellaneous scientific fields and domains into one common and continuous framework. On the other hand, it provides the capability to actually simulate the described systems with less effort, leading to precise evaluation at a minimal cost.

OVERVIEW OF THE METHODOLOGY

The objective is to demonstrate the potential benefits of operational simulation tools to perform validation of the FDIR from very early phases (where FDIR concepts are sketched) to FDIR design and implementation validation.

The proposed methodology foresees the reuse of simulation artefacts across the whole lifecycle, while focusing on specific analysis depending on the development phase, as shown on Fig. 1.

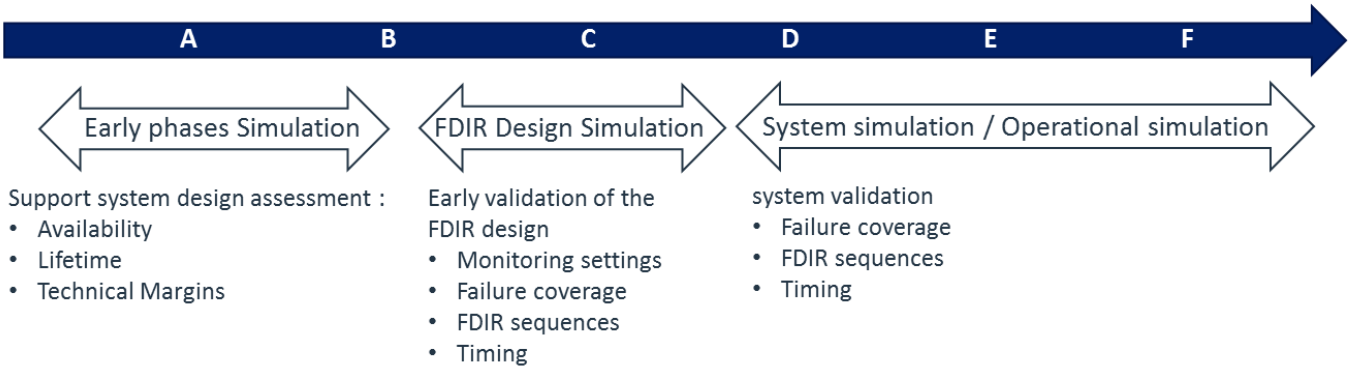

Fig. 1. FDIR simulation across the lifecycle

Reusing simulation artefacts requires managing the refinement of the engineering data across two dimensions, each one composed of several levels:

- Engineering levels:
  - The Functional analysis
  - is mapped on a Logical architecture
  - which is derived in a Physical architecture.
- Functional Abstraction level:
  - The Functional analysis elaborated during phase 0/A (advanced studies)
  - is refined at subsystems level during phase B1,
  - and finally at equipment levels during phase B2/C.

The ability to derive validation credits obtained early in the process until the end of the V-cycle depends on the traceability that can be defined between those levels in each dimension. Fig. 2 illustrates the different concepts and their mapping on the development process.

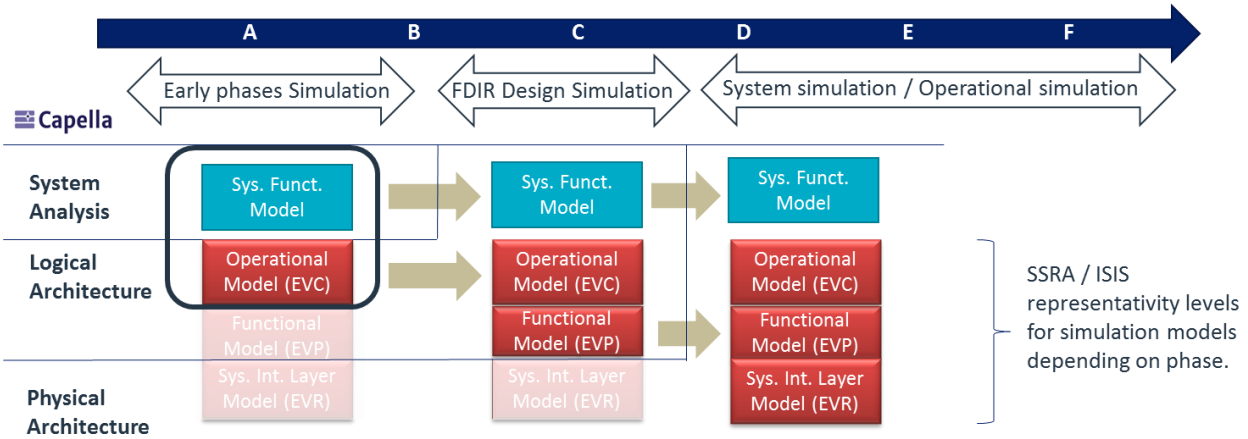

Fig. 2. Digital continuity according to the two dimensions in the development process

The study focuses on the simulation during early phases. The following sections details the model-based approach and the simulator generation activities.

Model-based approach from design models to simulation

During early phases, the different system design aspects that need to be captured by models for allowing simulation generation are the following:

- *Functional analysis model*: functional analysis performed at system level
- *System architecture model*: focuses on the interfaces of the system elements.
- *Erroneous behavioural model*: captures the failure modes and the impact on the nominal behaviour (which remains manually coded).

- *FDIR model*: captures both the strategy and the list of FDIR operational concepts
- *Mission Scenario model*: captures some operational scenarios to be tested.

Functional analysis and System architecture are captured into a Capella model. Capella is an Open Source Solution for Model-Based Systems Engineering [https://www.polarsys.org/capella].

As Capella is based on Eclipse EMF technology, it has been decided to capture other aspects (Erroneous behavioural model, FDIR model, and mission scenario model) thanks to EMF domain specific languages. The Capella model, extended with the external models can be used to produce the simulation tool, as depicted in Fig. 3.

Fig. 3. Proposed process at functional analysis level

This process aims at simulating the system functional analysis. Deployed in early phases, it allows assessing the availability and the reliability of the system by running a large number of scenarios in which failure events are randomly introduced.

This process can be complemented with the equivalent approach at logical engineering level, as depicted in Fig. 4.

Fig. 4. Proposed process at logical design level

APPLICATION OF THE MODEL BASE APPROACH ON THE USE CASE

Use-case description

To evaluate this approach, it was applied on the MICROSCOPE use-case. MICROSCOPE is a CNES mission with the challenging objective to perform at least N micro-gravity experiences during the mission time in order to test the universality of the equivalence principle. The mission duration is only limited by the initial amount of propellant.

Determinist events (eclipse, instrument illumination …) can prevent experiences to be performed, and a mission planning function has to find the optimum sequence for experience realization.

Failure events may also interrupt experiences which need to be planned again. This obviously implies a longer mission duration (rescheduling of experiences) but also increases the propellant consumption. The early analysis, based on simulation, aims at knowing the probability that the mission succeeds ($n > N_{min}$) before lacking of propellant.

Functional analysis

A very simple design model is built at the right abstraction level (see Fig. 5).

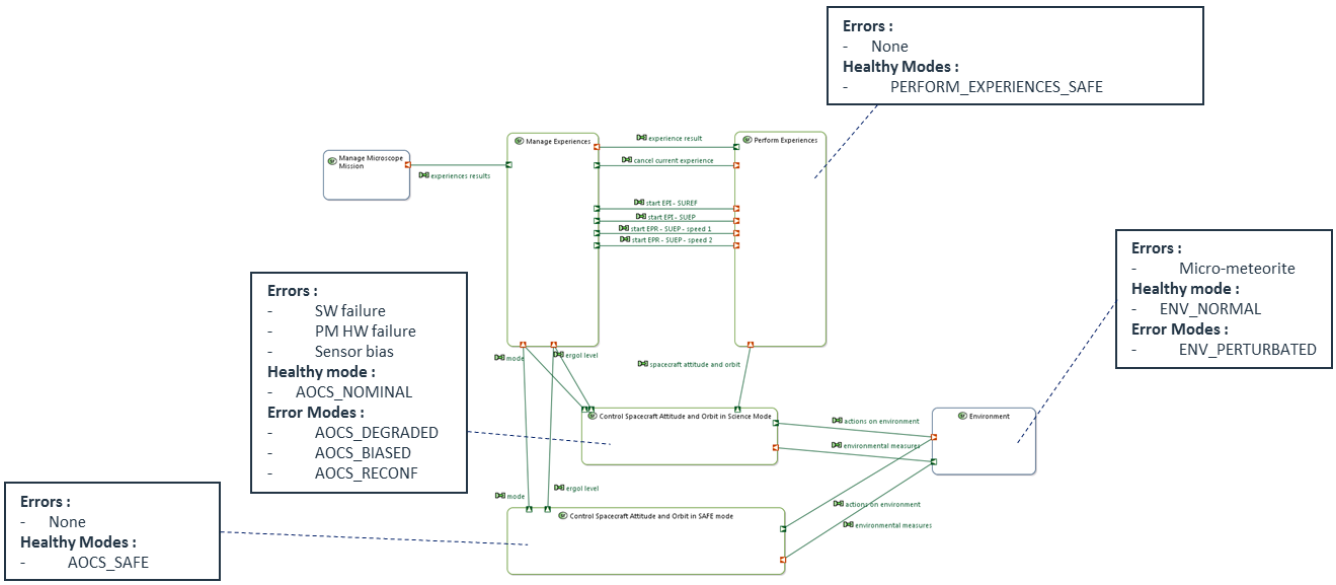

Fig. 5. Functional analysis of the Microscope mission.

Erroneous behavioural models:

In addition, for some system functions, the Erroneous behavioural models capture the failure modes and the impact on the nominal behaviour (which remains manually implemented).

Three aspects are defined:

- The failure modes (error modes), their probability, and the possible transitions between them.
- The effect on the data exchanged
- The propagation of the erroneous behaviour

FDIR model:

The FDIR design model captures, at functional analysis level, the FDIR strategy. First, the FDIR strategies specific to the mission are defined (e.g. FAIL'SAFE, FAIL'OP …). Secondly, the expected behaviour of the spacecraft when an error occurs (failure) in a given operational mode, and during a given mission phase, is specified.
This allows to generate a first simulation to analyse the chances to get a successful mission. This is described in the "SMP2 simulator generator overview" section.

Logical architecture

The logical model focuses on the interfaces of the system elements. It is performed in a two-step process: first, the functional analysis is refined, and secondly, refined functions are allocated to logical components.

One challenge is, during this refinement process, to keep the traceability between the first functional analysis and the refined one. This question is raised for functions and for functional exchanges. The Capella tool brings some support for this. Then, the work consists in specifying the logical component interfaces. The Capella toolset allows several ways to perform the description of the interfaces.

Capella also helps to map the Functional exchanges on the component exchanges: it is possible to specify the allocated functional exchanges for a given component exchange.

The logical model is complemented with an erroneous logical behaviour model:

When the functional analysis is refined at logical level, it is necessary to refine the safetybehaviour description for the refined functions. It can bring to the definition of new failure modes, and new failure propagations.
At logical architectural level, the failure propagation can be handled in two ways:

- Either thanks to the error model at logical level, which allows to describe the propagation in the same way as for the functional behavioural model.
- Or thanks to the nominal behaviour, not captured by the model, but rather implemented manually in the corresponding simulation model.

This flexibility allows to produce a simulator for the logical model even if the simulation nominal behaviour of every component is not available. The simulator can be incrementally improved by replacing the “logical” behaviour by the nominal behaviour.

FDIR model:

The FDIR design model at logical architecture level captures the list of monitorings and recovery actions related to the logical exchanged data.

Code generation for the logical level has not been explored during the study. The next section will present the results related to the simulator generation at functional level.

SMP2 SIMULATOR GENERATOR OVERVIEW

The goal of the generator is to produce an **executable simulator** that simulates the FDIR behaviour specified in the Capella descriptive models and dedicated extensions.

Expected end-use simulator overview

So as to allow reuse and easy sharing, the simulator was to comply with the ESA SMP2 standard, which has therefore been chosen as the target for the M2T transformation. The simulator is run with BASILES (CNES simulation platform).

Following the TOMS usage, the satellite deterministic behaviour, both in nominal and error cases, is entirely managed by the simulation models; external (environment events) and internal (random error events) influence is considered as scenario configuration and is thus handled, as well as all other configuration-related aspects, by the scenario script.

Nominal behaviour vs error behaviour

The generator purpose is to transform the descriptive model of the FDIR provided by Capella models into an SMP2 simulator. This generated simulator only handles the error behaviour of the satellite: since the nominal behaviour is not described in the Capella models, it will have to be to be added as an extension to the generated simulator (this is done manually in the case of this study).

It is important to note that what is here called the *error behaviour* is not just referring to the behaviour of an equipment that is currently not functioning properly: it rather stands for all the handling of the changes of statuses between mode(s) where the equipment is entirely functional to other mode(s) where the equipment might be in error or more generally in any degraded state.

Such a representation allows to handle error behaviour and nominal behaviour almost independently, which constitutes a major asset in the context of our study.

SMP2 GENERATOR DESCRIPTION AND MECHANISMS

Software

For the sake of this study, the model transformation is designed with Acceleo software. Since it accepts EMF models as input, Acceleo interfaces seamlessly with Capella, which was a major argument in favour of its use in our study.

Capella-to-SMP2 transformation

Systemarchitecture model:

The System architecture model elaborated with Capella is used as a basis to create the backbone of the simulator: models and data links. Each `Function` translates into one SMP2 model. One additional Container model is created to act as parent for all the `Function` models.

`Functional Exchanges` express data exchanges between `Functions`: each `Functional Exchange` translates into:

- **two SMP2 Fields**, one acting as output in the 'origin' model, the other as input in the 'destination' model
- **an Interface Link** between models ; Interface Links have been preferred to other SMP2 link types since on the one hand they can better scale to different types of data, and on the other hand, they are well suited to asynchronous event-driven data transfers. All needed interfaces are regrouped in a Connections model, which serves as a mere container.

The systemarchitecture model defines possible `Operational Modes` and `Mission Phases`. The current operational mode represents the satellite high level status; it is represented via a Field in the generated FdirController model (see below). The mission phase concept allow to divide a mission in separate parts, in which the satellite management policy might differ; it is also represented via a Field in the FdirController model.

Safetybehaviour model:

The Safety behaviour model describes the `Functions` state machines. Main objects are the following:

- `Error Model Event`: It represents an external event that applies to a `Function`. For each `ErrorModelEvent`, the generator creates an SMP2 EntryPoint and an associated event.
- `Error Mode`: It defines a state machine mode. In the generated code, the `Error Mode` is handled as a model SMP2 Field.
- `Error Mode Transition`: It describes a transition between modes and defines in response to which `Error Model Event` it must be triggered. The mode change is handled in the code of the `Error Model Event` associated handler function.
- `Functional Flows/Functional Flow Conditional Expressions/Safety Status Assignment`: It defines the value of an output input `Functional Exchange` depending on input `Functional Exchanges` and current `Error Mode`. This is handled by a function UpdateOuputs in each generated model. This function is called each time an input `Functional Exchange` value is updated or when the model `Error Mode` changes.

Fdirdesign model:

The FDIR design model describes the corrective actions of the OBSW on the satellite, which consist in triggering 'error' modes transitions within `Functions`. Main objects of the fdirdesign model are `Operational Mode Strategies`. An `Operation Mode Strategy` defines two Operational Modes (“considered” and “target”), two Error Model Modes (“considered” and “target”), and a delay. When the satellite is in the considered `Operational Mode` and a `Function` in the considered `Error Model Mode`, then the satellite and the `Function` switch, after a delay, to respectively the target `Operational Mode` and the target `Error Model Mode` (see Fig. 6).

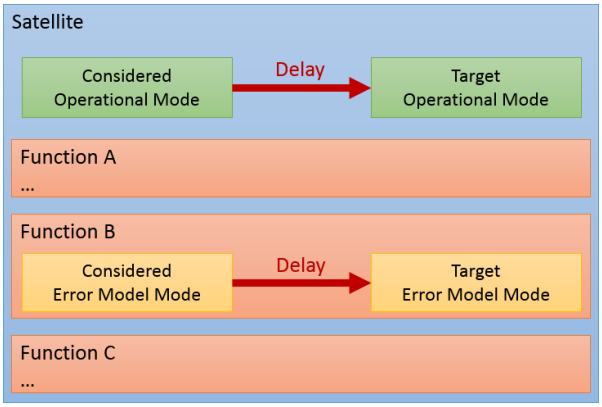

Fig. 6. Mode transition resulting from an operation mode strategy.

In the simulator, this whole behaviour is handled by a dedicated model named FdirController. This model is informed of each equipment mode changes (via Interface Links, with interface `OpeModeComputation`) and has the ability to force mode changes (also via Interface Links, with references `Ref<model>ModeManagement`), should an `Operational Mode Strategy` apply and require a mode transition. Some Entry Points are created to host the actual mode switches, since they must be scheduled with a delay.

As explained above, the generated code for each equipment handles the error behaviour of said equipment. The nominal behaviour, not described in the Capella models, is handled by additional hand-written code. In order to allow for interactions between hand-written and generated code, the latter provides some hooks, in the form of empty C++ functions. These functions are called by generated-code under specific conditions (on mode change, on data transfer via interface link, etc.), and are available to host nominal behaviour hand-written code.

Note that the link between error and nominal behaviour could have been done differently, for instance by dividing each `Function` into two separate SMP2 models (one handling the error behaviour, the other one the nominal behaviour) communicating for instance via Interface Links; however, in the context of our study, the added complexity of such a design led to choose the solution described above, which provides a less clear separation between the two behaviours, but offers the same coding context.

SIMULATION SCRIPT

The goal of the simulation script is to load the correct configuration for the simulation and to operate a *sweep*, i.e. a large number of successive simulations of missions. The script's algorithm can be summed-up as shown below. The backbone of this simulation script is created by the generator.

```
Initialize simulation infrastructure
Initialize models and nominal behaviour aspects
Save context
For a given number of loops
|  Restore context
|  Draw random error triggers
|  Run simulation #i
Exploit results
```

THE GENERATED SIMULATOR IN THE USE-CASE

The method has been tested with the Microscope mission as a practical use case (see **Use-case description** above).

Results

Here are two example of outputs generated by the simulator. The distribution of mission durations is represented in Fig. 7, with a different colour depending on mission success or failure (end due to propellant shortage).

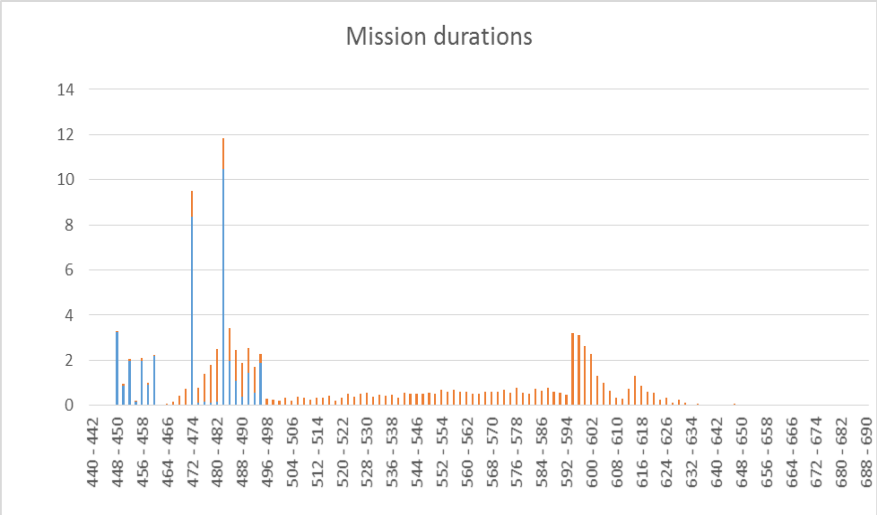

Fig. 7. Percentage of missions ending in the given time ranges (in days). Successful missions appear in blue, failed ones in orange.

The repartition of the additional propellant consumption per type of external error event is shown in Fig. 8.

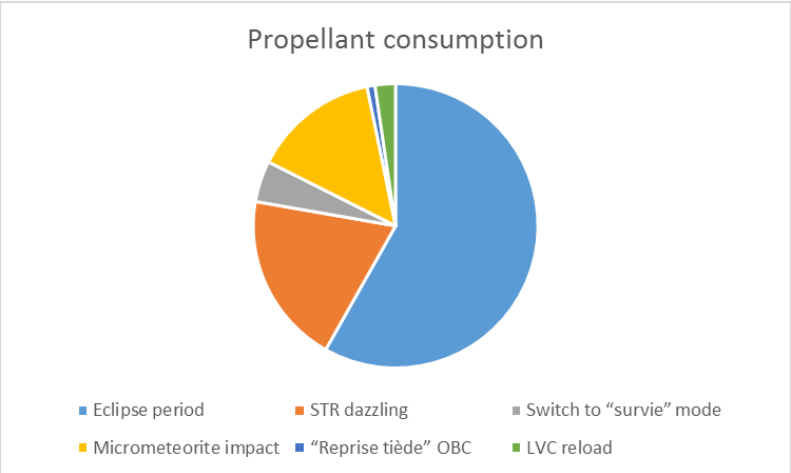

Fig. 8. Repartition of averaged additional propellant consumption per error event type

Code comparison

The code developed for this study uses a different philosophy than the reference one: it is completely based on the discrete-event simulation paradigm and it uses context storage/restore for each simulation of a sweep. This means that it makes the best possible use of all the SMP2 standard concepts, contrary to the reference code which used some hand-written code to take care of such aspects. This results in a more maintainable and customizable code, which moreover benefits from all the simulation infrastructure capabilities.

Another interesting comparison to make is about the number of lines of (hand-written) code in the reference implementation against the same number in the current study's implementation.

Table 1. Code comparison: number of lines for different code bases

Code base	Number of lines
Reference code (internship)	6703
Study code (all)	16298
Study generated code	15400
Study hand-written code	898

There is more than seven times less hand-written code using the generation from Capella models than in the reference code (entirely hand-written). Knowing moreover that SMP2 model coding style is much more verbose than pure BASILES (which is made clear by comparing the overall number of lines between the reference code and the study code in table 9) the actual gain is clearly over 7 times.

CONCLUSION

This study demonstrates the possibility of generating a SMP2 simulator from a Capella model extended with domain specific information, in order to successfully study the efficiency of FDIR strategy and associated FDIR design.

From a model based perspective, the use of Capella complemented with FDIR Domain Specific Models has demonstrated the capability to capture the essential concepts to achieve FDIR simulation (at functional level only). It is a promising work that will be integrated in further work dealing with model based reliability analysis, and with already deployed modelling tools covering the FDIR design (Thales Alenia Space FDIR editor).

As made clear in the last section, one major benefit of the method implemented is the reduced amount of hand-written code needed to obtain an executable simulator. It is worth noting that working with both generation templates and hand-written code does not add any major workload: this is made possible since the Eclipse Framework used to generate code, Acceleo, allows to easily update generated code without impacting hand-written code (located in protected sections) and, obviously, updating hand-written code has no impact on generation templates.

The study enabled two key areas for improvement and further work to be identified. Firstly, the separation between error and nominal behaviour, as explained in the Nominal behaviour vs error behaviour section above, although easy to understand, is hard to properly apply in practice. How should these two behaviours be separated? Should they lie in separate models? And more importantly, what information should they share?

Secondly, and more importantly, it is our feeling that it would be worth tackling a more complex use-case, so as to provide a truly generic solution. Lastly, it would be interesting to extend this process even more to other domains, in order to provide SMP2 simulators generated from several technical models (Simulink, Systema, etc.).