MBSE for system to software engineering

Philippe GAST, David LESENS, Pierre MORO

Ariane Group, 51-61 route de Verneuil 78130 Les Mureaux France, <firstname>.<name>@ariane.group

This paper addresses the following objectives:

- O-1: Successful applications with emphasis on reporting perceived return-on-investment
- O-4: Leveraging MBSE results to improve the definition and development of other downstream applications and use cases (e.g. simulation, validation and verification, operations)
- O-5: Experience reports on sharing models and data (doing MBSE together): Across engineering domains (system and software)

1. INTRODUCTION

One of the main difficulties of the development of the Ariane 6 flight software was the maturity of the system requirements allocated to the software, which contained initially ambiguities and were subject to numerous evolutions.

In order to quicken the maturation of these requirements, a system / software co-engineering development method has been developed supported by an intensive usage of MBSE.

2. ARIANE 6 OBJECTIVES OF MBSE USAGE

MBSE was already used in Ariane Group past programs but has been deployed in a wider scale on Ariane 6. MBSE objectives for the system / software co-engineering were, by priority order:

1) To improve the consistency between the "system requirements allocated to the software" and the "software specification".
2) To generate documentation. MBSE is indeed a powerful tool, but does not allow yet replacing documents for several reasons: contractual reasons, reviews reasons (need of training)…
3) To perform early verification (including simulation) by taking advantage of a formalized model.
4) To generate code.
5) To generate tests.

3. MODEL CONTENT AND GENERATORS

3.1 Functional architecture

The Ariane 6 functional architecture is based on a centralised Mission and Launcher Management (MLM) ensuring the consistency between a set of components (see [RD01]). A component gathers hardware and the piece of software controlling the hardware.

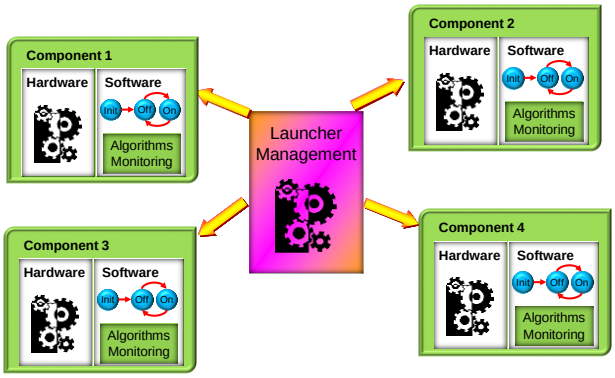

Figure 1: The system / software generic architecture

The software components are made up of

1) A finite state machine defining its behaviour in case of command reception and
2) A set of cyclic algorithms and monitoring depending on its state.

The consistency between the "system requirements allocated to the software" and the "software requirements" is ensured by sharing the same functional architecture between the system level and the software level. For instance, the finite state machine defined at system level representing the functional state of a sub component, is implemented as is in the flight software.

3.2 SysML modelling in practice

The past experiments on the deployment of SysML has shown that the engineering teams (especially the system engineering team) were not ready to use directly the Rhapsody tool currently deployed at ArianeGroup. Thus, a team dedicated to modelling is in place and works in co-engineering with the system team:

- From informal written exchanges or dedicated meetings, the modelling team captures the needs.
- The system team commits on the model, i.e. is fully responsible for its content.

In order to facilitate the modelling and prevent errors, a dedicated profile has been developed and the Rhapsody tool has been customised:

- All the SysML items not used in the frame of Ariane 6 has been removed from the Rhapsody GUI. This includes field in the properties, or diagram, or element in a diagram.
- All items have been specialized. For instance, monitoring and algorithms are specialization of SysML blocks with their own tags, colour and fields (containing information such as period or activation condition).
- All diagrams have been specialized. For instance two specialized kinds of activity diagram are proposed in the GUI, corresponding respectively to cyclic and acyclic behaviours. Each kind of activity diagram has its own allowed modelling items.

Around 150 automatic checks have been implemented to ensure the conformity of the model with the Ariane 6 profile. This profile and these checks have been formally documented in guidelines.

In complement, automated tools have been developed to ease the modelling, for instance for the automatic generation of the structure of a system or software item or the management of requirements.

All tools (described in this section and in the following ones) are accessible both in the Rhapsody GUI and in bash modes. Consistency checks and code generation are systematically and automatically executed before any commit in the configuration management system.

3.3 Complementary textual modelling

Some artefacts may take a great benefit from a graphical modelling (it is for instance the case for finite state machines or data-flow diagrams). However, other artefacts are not adapted to graphical representation and are more efficiently described in a textual representation. The SysML model has thus been completed by a Domain Specific Language (or DSL) describing:

- System mission plans.
- Multi-threading software architecture.
- Deployment of software monitorings and algorithms on this multithreading architecture.

The system mission plans are compatible with OBCP ("On Board Control Procedure") defined in ECSS-E-ST-70-01C "Space engineering – Spacecraft on-board control procedures". They can for instance describe:

- Requests for system actions (such as switch off of equipment or activation of a monitoring).
- Variables and numerical computation.
- Classical instructions of control.
- Monitoring of events.

```
plan Mission_Plan is
        -- Monitoring of an event
        wait event Tail_Off_Detection;
        -- Instruction of control
        if sqrt (A) > abs (B) then
                -- Numerical computation
                X := (5.3 - min (A, B)) / C;
        end if;
        -- Call of command
        Release_State;
end;
```

Table 1: Example of mission plan modelled in a DSL

3.4 Automatic documentation generation

The system detailed design documentation is composed by several sections, which are:

- Automatically generated from the SysML model for the "system requirements allocated to the software" section. This includes:
  - The finite state machines.
  - The definition of elementary system software functions (i.e. system functions which are implemented by software and which are not decomposed any more at system level, even if they can be further decomposed during the software design).
  - Some tables generated representing interfaces, activation conditions, frequencies, …
- Automatically generated from a Capella model for the functional analysis and the hardware architecture section.
- Manually written, mainly for the function's allocations either to hardware or software.

Once the model representing the "system requirements allocated to the software" for a component is completed, software requirements are added into the same model in order to generate the "software requirements" corresponding to the system component. The software requirements are captured in the SysML model either:

- Automatically: A requirement is defined as the instantiation of a generic requirements by retrieving instantiation parameters from the modelling artefacts. For instance, a software requirement is added to each finite state machine. The content of the requirement (transitions, initial state, etc.) is automatically generated using the information contained in the model.
- Manually, when the modelling is not adapted to automatic requirement generation.

The SysML model contains then information specific to the system documentation, specific to the software specification or common to both levels of documentation. In the latter case, the form of the generated documentation may be adapted to the users of this documentation while keeping a full consistency. For instance by:

- Hiding some modelling artefacts and by replacing them by tables or texts.
- Or by generating directly the graphical modelling artefacts.

The model itself becomes then the traceability matrix between the system and the software thanks to this 1 to 1 correspondence between the two levels of refinement.

3.5 Automatic code generation

A code generator has been developed to automatically generate a part of the flight software (see [RD03]):

- The software static architecture.
- The deployment of the static architecture on the real-time architecture (deployment of algorithms on the threads and safe data exchanges between threads).
- The mission and launcher management.
- The finite state machines.
- The data (interfaces with the hardware, constants and mission data).
- The instantiation of a generic mathematical library (e.g. for vectors, matrices, quaternions, filters and extrapolators).

The SysML semantics is neither enough formalised nor adapted to a launcher design to be used. So, even if SysML is used as a standardised graphical representation, ArianeGroup has defined and documented its own formal semantics.

The DSL used for the modelling of the mission plans and of the software real-time architecture has been extended to represent all the information of the SysML model needed to generate code. The code generator is classically designed with a front end able to extract all the needed information from a SysML model captured with the Rhapsody tool (the SysML tool currently used at ArianeGroup), a transformation engine, and then a back end with an Ada code generator. In case of obsolescence of the Rhapsody tool (implying an unavailability of the tool) in the future, this tool structure will allow adapting it easily to another SysML tool. This DSL has also been designed to be simple enough to be manually maintained.

The ECSS-Q-ST-80C – §6.2.8.3 stipulates “The required level of verification and validation of the automatic generation tool shall be at least the same as the one required for the generated code, if the tool is used to skip verification or testing activities on the target code”.

It was chosen to not qualify the code generator and to rely on classical verification and validation of the generated code, and especially on automatic test generation (see next section).

3.6 Automatic tests generation

Generating tests from a model may be used in two cases:

- To validate the product under tests which is developed from this model. The model is then used as the specification of the product under tests.
- To verify the internal consistency of the model.

The Ariane 6 verification and validation activities are supported by several test generators covering the following aspects:

- System tests for the validation of the mission and launcher management (see [RD02]).

  In this case, the product under tests is the model of the system detailed design. A model of the requirements is then needed to generate the tests.
- Software integration tests
  - Software / software integration (see [RD04]).
  - Hardware / software integration.

  In this case, the product under tests is the software code, generated partly automatically and partly manually.

4. BENEFITS & DIFFICULTIES

After several years of deployment of MBSE, this section provides a synthesis of the pros / cons of the approach.

4.1 Difficulties

One of the main interest of MBSE is its capacity to formalise descriptions, with a decrease of the ambiguities for the next level of refinement. But this advantage may also be a drawback by pushing the developers to provide too much details (such as making too much design choices at specification level). Sharing a model between two levels of refinements (here between the system detailed design and the software specification) increases this tendency.

Non-optimal choices of modelling rules have a strong impacts on the development costs.

A dedicated modelling team shall be put in place because the classical engineering teams (especially at system

level) have difficulties to be autonomous for the design of the model and for the use of the modelling tools.

The Ariane 6 project fails to select a single MBSE tool: Both the Rhapsody tool and Capella tool are used which may be considered as a source of over cost and a risk of errors.

4.2 Benefits

Despite of these difficulties, MBSE allows producing "system requirements allocated to the software" and "software requirements" with a high quality. An important decrease of the software specification ambiguities and a higher level of consistencies between the refinement levels has been observed (from the "system requirements allocated to the software" to the software code).

The automatic code generation allow a very quick taking into account of system requirements changes (at least the ones in the scope of the modelling!): from few months (without MBSE) to few days (with MBSE).

The simulation and the automatic test generation allows a very high level of coverage (for the mission and launcher management, the software / software integration and the hardware / software integration) never reached before and at very low costs see [RD02]

5. CONCLUSION

MBSE usage is considered as a great success for the Ariane 6 program when considering its complete eco-system:

- The model itself.
- The automatic checks.
- The automatic documentation generation.
- The automatic code generation.
- The automatic test generation.

6. REFERENCES

[RD01] Generic Software Architecture For Launchers, DASIA 2015.

[RD02] Automatic Test Generation An industrial Feedback, ERTS 2022.

[RD03] From functional definition to software code, ERTS 2016.

[RD04] Ghost code and optimisation, Ada Europe 2021.