Engineering of complex avionics systems simulations using a model based approach

Patrice Thebault, Thierry Suquet, Michael Duffy, Benoit Viaud, Pascal Will, Julien Codron, Sébastien Lamoliate

Engineering of complex avionics systems simulations using a model based approach

Patrice THEBAULT
Thierry SUQUET
Michael DUFFY
Airbus Operation S.A.S

316, route de Bayonne
F-31060 Toulouse Cedex 3

Benoit VIAUD
Pascal WILL
Julien CODRON
ARTAL Technologies
227 Rue Pierre-Gilles de Gennes - BP 38138
31681 LABEGE CEDEX

Sébastien LAMOLIATE
SOPRA group
1 Avenue André Marie Ampère
BP 10134 - 31772 Colomiers Cedex

Abstract

With avionics system complexity increasing rapidly, the use of simulation throughout the system development lifecycle is a key enabler to achieve system maturity at Entry Into Service. However, simulations themselves are also complex systems whose development requires a high level of collaboration between multiple skills (IT, Plant modelling, Control, Embedded Software, etc). Therefore, having a functional simulation with the required level of fidelity and available at the right moment is always a tough challenge.

To tackle these challenges, the INtegrated SImulation into Design project (INSIDE) was initiated by Airbus in 2009 to organize this collaboration at early definition of simulation architectures and simulation model components.

The approach is based on a Model Driven Engineering using System Modelling Language SysML to support architecture definition. The introduction will describe the problem in detail and then the paper will explain the proposed solution and how it improves the performance of the simulation development process. Finally, the feedback and future improvements are presented.

1. Introduction

The following explains the engineering process of simulation products which are enabling tools used to perform verification and validation activities for a system of interest. In this case, the system of interest is considered to be an avionics system (cf. Figure 1).

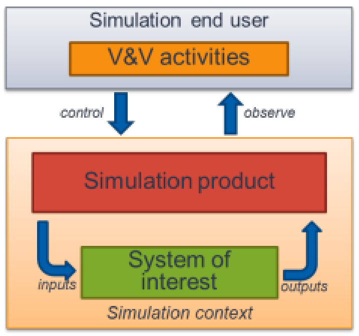


Figure 1 – High level simulation entities

The objective of the simulation

The overall objective of the simulation is to verify and validate the system of interest operating into simulated but realistic environment conditions.

Simulation can be used for:
- Requirements validation to ensure that the requirements for a product are sufficiently correct and complete to satisfy the needs of the customer, safety & program;
- Design Verification to provide evidence that the design, during the descending part of the V diagram is compliant with the requirements;
- Product Verification to ensure that the product meets the requirements;
- Product Validation to demonstrate that the product meets the needs of the customer.

The system of interest

In the simulation, the system of interest can be either virtualized or real.

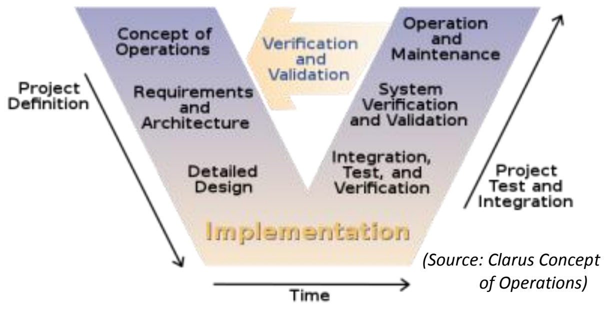

Concept of Operations
Verification and Validation
Operation and Maintenance
Project Definition
Requirements and Architecture
System Verification and Validation
Detailed Design
Integration, Test, and Verification
Project Test and Integration
Implementation
(Source: Clarus Concept of Operations)
Time

Figure 2 – V-cycle model

The system of interest is virtualized on the left hand side of the V development cycle (cf. *Figure 2*), when the system itself is under design. Simulation is then used to perform validation of the specification and verification of the design of the system of interest. At this stage, the system of interest integrated with the simulation product may be implemented by prototypes, which partially or totally cover the system of interest functional perimeter.

When the system of interest becomes a product, i.e. on the right hand side of the V development cycle, simulation is used to perform product verification and validation, particularly to explore the system of interest operational environment in extreme conditions (such as failure conditions, limits of the domain of use, …).

The simulation product

A simulation product in Airbus is generally described as a simulation application deployed on a simulation platform, and interfaced with the system of interest. When the System of Interest is virtualized, we generally consider it as part of the simulation application (cf. Figure 3).

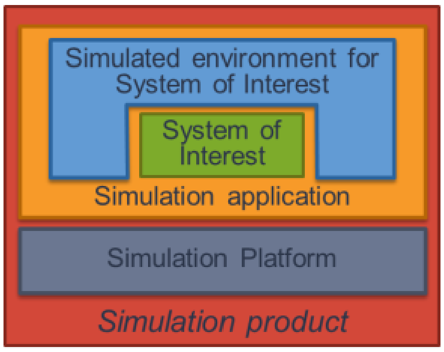


Figure 3 –Simulation product architecture

To enable such architecture representation, it is required to define a common understanding on how the simulator platform shall execute the simulation application. For a wide part of Airbus simulation products, this is defined by an internal Airbus standard.

The Airbus Simulation Model (ASM)

This standard was issued in early 2000 to support exchange and reuse of simulation components across Airbus simulation platforms. It defines the concept of an "Airbus Simulation Model" and how it shall be executed on a simulation platform.

The execution scheme of an ASM relies on a periodic call of a main entry point by the simulation platform, during which it processes outputs from given inputs. Specifying who is providing the inputs and who is using the outputs is the role of the simulation application. How the outputs from one ASM are provided to inputs of another one, is the role of the simulation platform.

From the delivery point of view, although the Airbus Procedure states that an ASM shall be provided as C or Fortran source code, they are increasingly delivered in tool specific modelling languages (such as Simulink, SCADE, …), or sometimes directly as binary executable files.

The simulation application

The role assigned to the simulation application is to simulate the operational environment of the system of interest.

The simulation application development is based on the knowledge of the operational environment of the system of interest, which is, in the avionics context, composed of equipment whose behaviour is governed by physical law (Aircraft motion, electrical, hydraulic system) and other avionic systems.

The simulation application comprises a set of ASM and associated configuration files which specify the connections between ASMs, and their scheduling properties.

The simulation platform

The simulation platform consists of an IT infrastructure and the simulation software. Three main objectives are assigned to the simulation platform. Firstly, it schedules and monitors the execution of the ASMs with respect to time constraints of logical or real time simulation. Secondly, it implements the communication between ASMs. Thirdly, it provides the end user with control and observation facilities to operate the simulation.

In addition, when system of interest is a real product, specific facilities are also included in the simulation platform to enable signal adaptation and transportation from real world to simulated world.

Statement of problem

The simulation product is used during the design of a system of interest. For this reason, it not only **needs to be updated continuously** to follow each design change, but also to offer to the simulation end user (i.e. the developer of the system of interest) new simulation capabilities required for V&V objectives.

It is mainly the simulation application which supports this required agility. As mentioned before, the simulation application relies on integration of ASMs. It is the first foundation of a component model, but it lacks certain aspects such as those concerning the composition of models into a simulation application.

To work around these drawbacks, the simulation application development process is performed by Simulation platform teams, but they still need to capture information from simulation component developers, which hinders overall efficiency.

The objective of INSIDE is to provide a solution to enable real autonomy to Simulation application providers and independence from Simulation platform teams.

Before presenting this proposed approach, Part 2 of this paper presents in detail the simulation application development process, and the challenges to make it reactive.

Part 3 and 4 expose the proposed approach based on MDE and SysML to capture the different aspects of the simulation application.

Part 5 explains the difficulties encountered to deploy this approach and the means to solve it using adequate tools.

Part 6 presents the initial feedback and the perspectives.

Part 7 concludes by considering potential extensions to cover the overall documentation needs (through models) of a simulation product.

2. Simulation application development process

The simulation application development process is the cornerstone of the simulation product development process (cf. *Figure 4*).

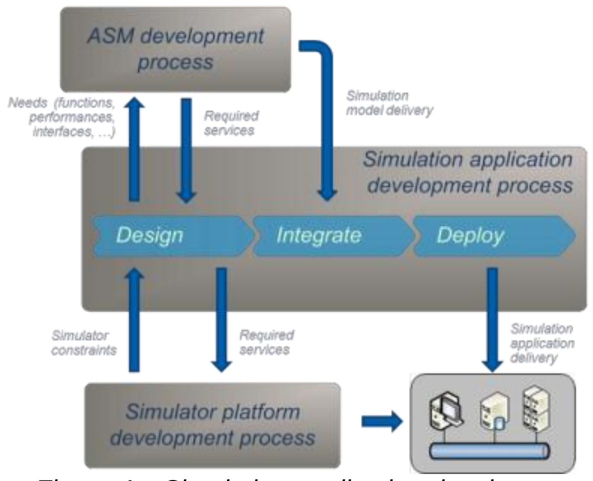

ASM development process
Needs (functions, performances, interfaces, …)
Required services
Simulation model delivery
Simulation application development process
Design
Integrate
Deploy
Simulator constraints
Required services
Simulation application delivery
Simulator platform development process

Figure 4 – Simulation application development process

The first objective of this process, “Design”, is to specify the simulation content and to define which simulation model can provide this content. This includes the definition of the functional and performance objectives of the simulation models and also the specification of their interfaces.

The second objective, “Integrate”, is to ensure the technical coherency between the simulation models and the simulation platform on which they will be deployed.

Many actors are involved in these processes and efficient collaboration is a major key to achieve the objective of reactivity required by the end user of the simulation product.

Design simulation applications

Several activities are necessary for the design of a simulation application. Firstly, end users must identify their objectives of used. This enables the required functions and performance of the simulation to be specified. The simulation application architecture is then defined as a set of ASMs connected together, and all required functions can be allocated to the ASMs.

A typical simulation application could contain up to 100 simulation models with a total of 200 000 inputs and outputs connected together. These are the key factors which lead to the complexity of the simulation application. This complexity could never be managed manually. It is therefore critical to structure information so that the correct design subset can be edited autonomously by the user.

Reuse simulation model

Obviously, each development of ASM takes time and costs money, therefore the simulation application designer should identify the reusable ones.

The reuse of ASM across simulation applications reduces cost and time, but it also brings constraints during the assembly phase when they have to be connected together. Only the component specialists have the knowledge to specify the connections, and several specialists are required to integrate a whole simulation application (Cf. *Figure 5*).

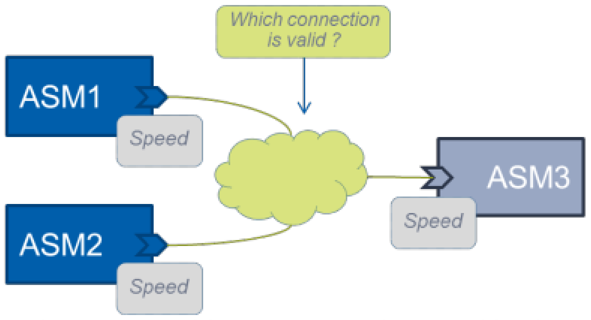


Figure 5 – Problem of connection specification

Furthermore, interfaces of models are not always compatible, which requires signal adaptation (cf. *Figure 6*).

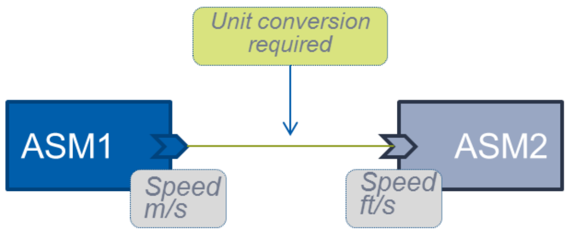


Figure 6 – Problem of signal adaptation

ASM implementation

Once the simulation application is logically designed, each ASM has to be developed within its own lifecycle and its specific methods.

Therefore, each ASM has its own definition, managed in a separated view from the one built inside the simulation application.

Traditionally, these two aspects are covered by separate documents: one is the Model Functional and Performance Requirement to describe the ASM from the simulation application designer point of view, the second is the Model Specification to describe the ASM from its developer point of view.

This distinction enables the acquirer of the ASM to keep an abstract view on it, while the developer can provides further details on its implementation, as for example its packaging properties which depend on the type of methodology and tools used during its development. The ASM implementation can then be delivered as a Simulink, Scade or C-Code model as long as they comply with the Airbus procedure execution semantics.

It is the simulation platform responsibility to finally take into account each ASM with adequate production and deployment procedures via model transformations.

3. MDE approach to support simulation application development

As mentioned above, many actors collaborate to develop a simulation application. Therefore, we need a shared language to support their communications.

SysML had been previously selected in another R&T Airbus project[2] as a language to support specification and design activities, and it was decided to adopt it. The contribution was then to demonstrate that it can be used efficiently to capture the design of complex simulation applications, and how we could make a bridge with legacy formalisms.

Let's first introduce the concepts that are needed for the formalization of the complete design of a simulation application architecture.

Simulation application domain model

In addition to "ASM" component, two other types of components have been considered.

The "PackageOfModels" component was added to support the architecture of the Simulation application, and to introduce a level of encapsulation, into the simulation application.

The concept of "SimulationDataBus" was added to enable capture of the connexions between different ASMs.

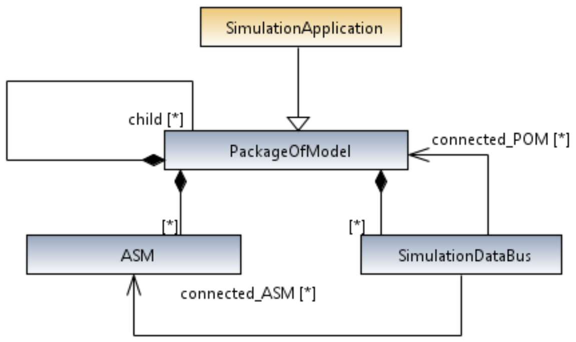


Figure 7– Type of simulation components

Each simulation component is characterized by a definition and an implementation. The concept of SimulationComponent was introduced to reuse these properties throughout the development process (cf. below).

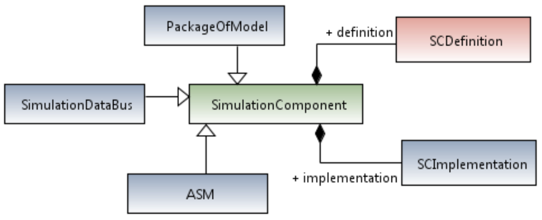


Figure 8 – Simulation component types

Each SimulationComponent includes a SCDefinition which contains the specification of the component from two points of view: the logical definition and the physical definition as already proposed in another Airbus R&T project[2] and following the EIA632[9] guidelines.

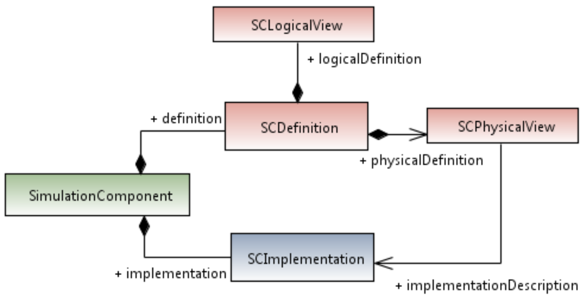


Figure 9 – Simulation Component definition

The logical definition is used to describe how the Simulation Component is breakdown into several sub simulation component and how they are connected together.

The physical definition allows connection of the definition of the component with its concrete implementation. For instance, it is possible to implement an ASM with Simulink, Scade or C Code but whatever the final technical choice, it is documented into the "PhysicalDefinition".

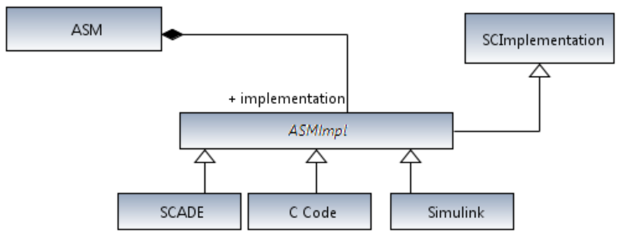


Figure 10 – Implementation for ASMs

Method to describe Simulation component

It was decided to capture the SCDefinition within the SysML model organized in several packages including one for the “SCLogicalDefinition” and another for the “SCPhysicalDefinition”.

Again, it is useful to present what concepts are intended to be captured prior to presenting how they are captured using SysML language.

At this stage of the project, capturing the structural aspect of the simulation was the primary focus while most of the behaviours are hidden inside the simulation components. There are both external and internal structures to capture.

External structure definition is used to describe the three types of Simulation component (cf. Figure 1) from a usage point of view. It provides the list of “SCDataPort” which corresponds to the interaction points of the simulation component. Each “SCDataPort” is specified with a “SCDataInterface” which provides the list of data that flows in or out of the simulation component via the SCDataPort.

Internal structure is only used to describe the internal organization of the PackageOfModels. It provides the list of instances of simulation component which composed the PackageOfModels, and how they are connected together through their SCDataPort. Each instance of simulation component is typed by a SCDefinition which corresponds to the definition of a required Simulation Component from the point of view of the PackageOfModels designer.

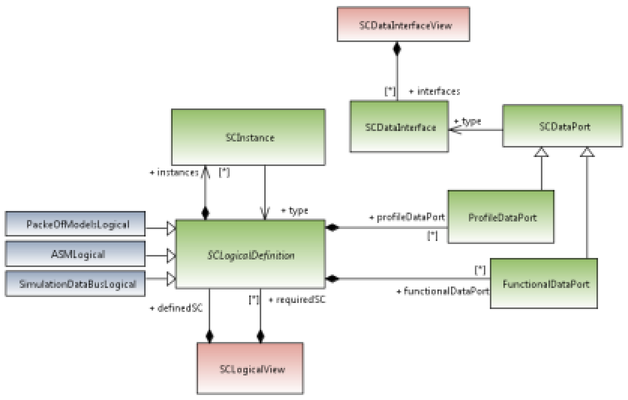


Figure 11 – Concept to be captured in SysML

Since ASMs and SimulationDataBus internal structure definition were already supported by specific formalism and legacy tools, it was preferred to integrate these specific formalisms behind the SCLogicalDefinition.

As mentioned in previous section, the main difficulties for the development of simulation applications relates to the capture of the connections specification between ASMs. For that, one of the critical elements are the interfaces. Because interface definition is a shared information between two connected components, defined at architecture level but also used also at implementation level, it cannot be managed within the SCLogicalDefinition. Therefore, it needs to be managed in a specific view which is imported by the logical one : the SCDataInterfaceView.

Mapping of concept with SysML language

In order to describe the simulation components within the SysML model, only a subset of the language is used.

First of all, it is required to support the organization of the information into different views such as the SCLogicalDefinition, SCPhysicalDefinition and SCDataInterfaceView. This is provided by the “uml::package” concept.

Then, for the description of the simulation components itself, a profile was developed to specialized the SysML language with the simulation domain concept.

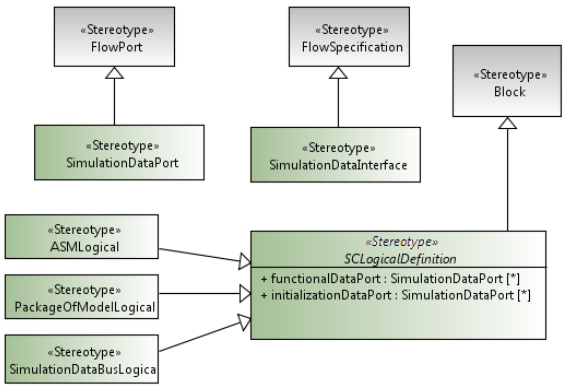


Figure 12 – SysML profile for simulation definition

The advantage of the profile solution was to be able to develop a checker on design rules for the different simulation components. For instance, an ASM component has two FunctionalDataPort with specific names.

While SCLogicalDefinition is a type definition, it has been decided to represent it with a “sysml::block”. Then, the Internal Block Diagram (IBD) is used to define the internal structure of the PackageOfModel, with “uml::part” used to represent the different SCInstances. Inside the IBD, the “uml::connector” is used to connect the port of the different instances of SimulationComponent.

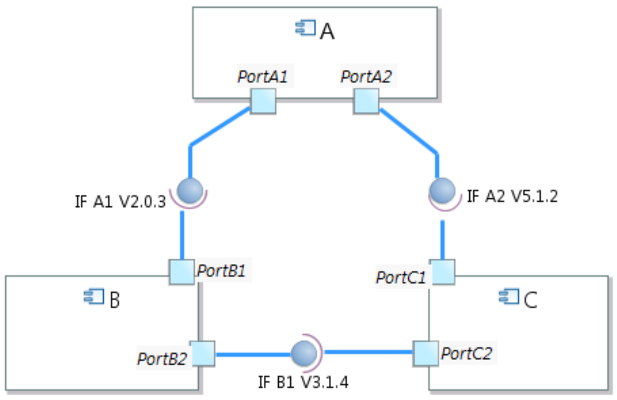
To specify the SCDataPort and SCInterface two modelling solutions were available. Using atomic flow ports or using flow ports typed by flow specification.

Both solutions require adding flow ports on the block which represents the simulation model; however the solution based on flow specification offers the major advantage in our case to separate the list of exchange flow from the structure of the simulation model, and then was manageable in a separated package. This package is then imported in the different SCLogicalDefinition.

The content of the SCInterface is detailed with SysML::FlowProperties which are typed by predefined SysML::ValueType, corresponding to primitive allowed type by the Airbus Procedure for ASM.

4. MDE and life cycle management

The design process described above reaches another level of complexity when it comes to managing the simulation evolution with time. Each component used in the simulation, starting with the ASM, will evolve over its life-cycle in order to integrate new functionalities, bug fixes, etc. There is a real need and challenge in supporting the simulation designer with the upgrade of components when building a new version of the simulation application.



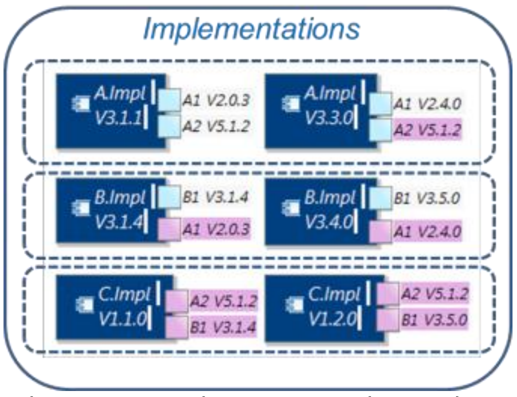


Figure 13 – Basic component interactions

In the situation illustrated above, the designer has started a simulation architecture definition involving three identified components; from this specification, the INSIDE framework is able to find and proposes existing implementations for these components that satisfy the versioned interface constraints. At some point, the version of interface A1 needs to be upgraded to version 2.4.0. In the example, INSIDE finds a matching implementation for component A, but not for component B: the implementation compliant with the upgraded version of A1 also provides an upgraded version 3.5 of its own interface B1. INSIDE then proposes an implementation solution to the simulation designer that implies another change of interface, in addition to the initial desired upgrade; the simulation designer has to decide whether to go with the proposed solution or to request the development of a new specific version of B satisfying the interface constraints.

To support this use case, INSIDE relies on:

- A clear distinction between simulation architecture needs and components implementations through versioned interfaces;
- A common repository where all elements are stored and referenced;
- A resolver function, triggered in the edition environment, which can browse the repository to find and propose the elements that satisfy the constraints of the architecture definition.

To give more autonomy to the resolver and minimize the number of requests to the designer during the process, INSIDE proposes to define the interface compliancy with a range of versions (e.g. [3.0.0; 4.0.0[). In the above example, the simulation designer could express that the interface B1 between B and C could be any version from V3.1 to V3.9. Given that level of freedom, the resolver can directly propose a matching solution.

More can also be done for the resolver by using semantic versioning [4]: the resolver can then infer a first level of compliancy between the elements from the name of the version (e.g. component C using version V3.1 of interface B1 shall also be able to use any 3.X version). This track has not yet been fully explored in INSIDE; considering the context, it will probably require an automatic versioning mechanism to support users when they publish their new component definition.

5. Tooling to support simulation application design

When the approach was experimented with real end users, a certain resistance was initially encountered .

Two types of difficulties were raised. Firstly, SysML was another language to learn and perceived as far from the business domain language.

Secondly, SysML model edition was perceived as a heavy process, with many operations required to specify simple properties.

A part of the solution came with the Eclipse technical framework [5] chosen to develop the project and especially with the Eclipse Modeling Framework. On one side there are existing implementations of SysML [6] on top of EMF and on the other side, there are several graphical technologies on top of EMF that allow to view and edit an EMF model more simply.

Considering the context, we needed more than just one graphical paradigm to view and edit our model: boxes and lines are well adapted for the overall architecture description, tabular editors are almost mandatory when it comes to managing interfaces with many thousands of signals and finally, mainly for the purposes of improving ergonomics, the main elements (ASMs, interfaces, …) shall be easily browsed from the project view.

Following the proven Model-View-Controller pattern, it is relatively easy with EMF to propose different views on parts of the same model. Obeo Designer **Erreur ! Source du renvoi introuvable.** was a first-class candidate to build consistent, visually pleasant and rich views; it was completed with lower level rendering technologies for large signals tables and management of the elements directly from the project representation tree.

In the Airbus context, there were legacy tools that could/should be used to edit some specific parts of the model; in particular, the edition of the connections between the signals in the databus was already provided by an existing tool. Using EMF, it was relatively easy to use Model-to-Text technologies such as Acceleo [8] to generate the required inputs for the legacy editors: e.g. the “edit databus” action in INSIDE transparently generates the configuration files and launches the external editor, allowing the user to refine the SysML model started in INSIDE.

6. Experimentation feedback

The approach has been deployed on a lab test environment only but experimentation was realized on a fully representative application case corresponding to a simulation product.

The simulation application deployed on this simulation product is composed of 109 ASMs which have more than 14 000 connections. The ASM providers work in five different organizations, belonging to the different Aircraft System design domains.

The objective of this application case is to perform multi system design validation, on Bleed, Engine and Warning system. For such objective, it is required to have high reactivity on the simulation application update after each system design change.

To enable such reactivity, the best way is to let the person who knows best, do what needs to be done: that is, the system designer, who is behind the design change.

But, when the changes are potentially originating from several design teams, it is required to ensure that on one hand they can collaborate together to update the whole simulation when the change impacts at a multi-system level and on the other hand that they can quickly iterate on their own part of the simulation when the change has no impact outside their system.

This is the typical use case for which our approach aim at providing a solution, and therefore, it was the perfect setting for verification. Five major criteria were chosen to assess the solution: autonomy, reactivity, workload, quality, ergonomics.

In order to perform the assessment, the simulation application was first designed (cf. Figure 14) by the INSIDE team, and then during a workshops with end users, realistic changes scenarios on the simulation application were executed so that end users can first understand the proposed principles, then project their uses into a future process, and finally assess if the project objectives were achieved.

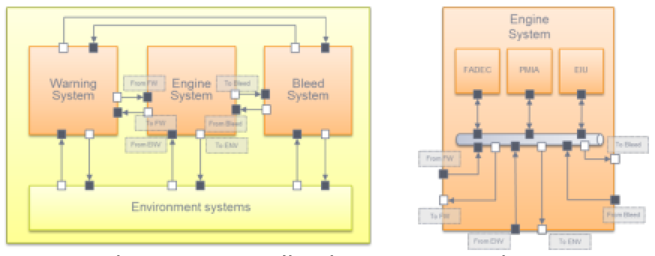


Figure 14 –Application case overview

Globally, the feedbacks from end users were very positive. They have assessed the added value of the solution in terms of improved autonomy for updating a simulation application.

On the side of reactivity, it was considered easy in terms of ability to easily integrate the update of a subset of the simulation application done by another team. In the legacy approach, this operation is always difficult, especially when there are concurrent modifications of the simulation application, which imply time consuming and error-prone merge operations. With the proposed approach, thanks to the ability to encapsulate ASM into PackageOfModels and also to the capability provided to easily collaborate on the definition of the interfaces, end users have confirmed that the operation would be far simpler.

For quality, we could not quantify objectively the level of improvement. Nevertheless, it was widely admitted that the autonomy given to the system designer, the primary owner of the knowledge, to update the simulation, would reduce the risk of introducing error.

Regarding ergonomics, since it was only a prototype approach, all the possible “shortcuts” were not implemented in the tool. But, there were no major questions or concerns from end users regarding possible difficulties to understand the key capabilities of the tool. However, the need for some fairly straightforward specific training has been identified.

Another very positive feedback was the ability to increase reusability of simulation application artefacts. Thank to the concept of PackageOfModel it becomes possible to reuse a larger subset of simulation applications, while before it was only possible at the ASM level. This opportunity was identified in particular for reuse from single-system simulation platforms to multi system platforms.

7. Conclusion

This paper presents the results of a project aimed at improving the global efficiency of a system simulation design in a context where the simulation itself has become a complex system, involving many different teams. A Model Driven approach has been set up which has enabled the formalization of the existing organisations, processes, tools and simulation models materials with a focus on:

- Collaboration: with a clear and formal separation of the simulation components specifications from their implementations, the simulation application designer can work on the definition and integration of the simulation components in a relatively independent way while the component providers can work with more formal specifications.
- Reuse: with the ability to compose simulation models into bigger models, it is possible to capitalize the definition and integration work related to these composite models.

Despite the fact that the project was built over the Airbus Procedure for ASM that gives guidelines on the structure and interfaces for an elementary simulation model, we believe that what has been designed to address the above mentioned issues is not specific to the Airbus environment and could be adapted to other contexts with the same benefits.

While the first steps in a transition towards more Model Driven Engineering are not always obvious - with a certain amount of energy spent just to describe what already exists -, it is then a strong basis on top of which it is possible to build high added-value features easily. To achieve even greater benefits, the framework started in this project should be extended with two additional MDE approaches: firstly, before the simulation design activities, to support capture of simulation needs (functional and performance), and secondly to specify the deployment of the simulation application on a simulation platform.

The final objective is to have a seamless process that enables rapid development of the simulation product, with high reactivity on the provision of updated simulation applications.

8. References

[1] OMG, Systems Modeling Language (OMG SysML), Version 1.1, 2008-11-01

[2] Mise en œuvre de SysML pour l’ingénierie des Produits Avioniques et de Simulation à Airbus - 16ème journée Thématique AFIS 27-09-2012

[3] Evaluation of Modeling Tools Adaptation, Amine El Kouhen, Cédric Dumoulin, Sébastien Gérard, Pierre Boulet, hal-00706701, version 2, http://hal.archives-ouvertes.fr/hal-00706701

[4] Tom Preston-Werner, “Semantic Versioning”, http://semver.org.

[5] Eclipse. http://www.eclipse.org

[6] TOPCASED: The Open Source Toolkit for Critical systems http://topcased.org

[7] Obeo Designer, http://www.obeodesigner.com

[8] Acceleo. http://www.acceleo.org

[9] ANSI/EIA 632 2003. Processes for Engineering a System. American National Standards Institute (ANSI)/Electronic Industries Association (EIA).