Abstract—Design and Implementation of Safety-Critical Systems is becoming very difficult because it involves many requirements coming from different engineering domains. Due to the increase of complexity, software of such systems can no longer be produced with traditional methods, which show their limit over time. In that context, new development approaches have to be introduced to avoid actual development traps and pitfalls. Among them, the Model-Driven Engineering approach consists at representing system artifacts with models and auto-generate the code by refining them from high-level concepts down to the code. However, as for every new approach, it also brings new problems such as requirements consistency among the different notations (models) as well as integration issues (for example, making sure that implementation code from different models will behave correctly when merged on a single execution platform).

This article presents our experience for integrating Guidance and Navigation Control (GNC) algorithms designed with Application Models (Simulink) with Architecture Models (AADL). The process relies on code generator for both models and integrate it on a typical execution platform. In particular, we focus on the challenges of the integration, illustrating the practical problems we faced for producing a space system using a Model-Driven Engineering Approach.

Keywords-AADL, TASTE, Simulink, MDE

I. INTRODUCTION

A. Context

Safety-critical systems are getting more complex, collocating more functions on the same computing platform. As a consequence, their design becomes more complicated, leading to a long and potentially painful design process. Designers have to take into account requirements coming from different domains and specified with heterogeneous formalisms.

For that reason, producing the system using traditional methods is no longer feasible: checking impact between all requirements disseminated across different specifications and notations is impossible, especially when the process is not automated. For that reason, new approaches must be designed. In our context, the Model-Driven Engineering (MDE) approach aims at separating system concerns in models, let engineers focus on their part of the system while tools automate the integration and ensure consistencies between modelling artifacts.

B. Problem

One key aspect of the MDE methodology is the separation of concerns: each engineer focuses on designing and implementing his part of the system dedicated to his domain while specific tools process each implementation artifact, ensure their integration and preserve a semantic consistency between notations.

Despite having a clear separation between each domain, several problems remain when implementing a full MDE approach. First of all, because of the use of different notations, some requirements are sometimes captured twice in different models using heterogeneous notations. Then, tools are expected to ensure their consistency but engineers are also requested not to break them when manipulating models. Another issue is about simulation and implementation correctness: even when a system was intensively tested using simulation functions, its integration as an implementation code can generate a lot of errors. Most of the time, this is due to the heterogeneous nature of the execution platform, whose environment is different from the simulation. Thus, having a dedicated process that ensures a smooth integration of system functions by enforcing their behavior correctness is a must.

In that context, to improve MDE approaches, it is worthwhile to identify all traps and pitfalls in order to strengthen the overall approach. Outcome of such investigations would make tools more resilient to potential integration errors, the type of issue which is typically discovered just before completion of a project, when few resources are available.

C. Outline

The remainder of this paper is structured in two main sections. The first one is an overview of the tools we use: the TASTE toolset (Architecture modelling [1]), and Simulink [2] (Software modelling) and their integration. Second part of this paper presents our latest experiments to design a spacecraft system using our MDE toolset by designing GNC algorithms on top of a distributed architecture designed with TASTE. We provide a feedback about traditional traps and pitfalls of such integration, leading to an open discussion about potential improvements of the whole MDE development process.
II. BACKGROUND

A. Simulink

Simulink consists of a graphical modelling language and a set of tools for designing software (an example is shown in figure [2]). It focuses on the definition of functional concerns and is mostly used to design algorithms in specific engineering domain (power control, navigation, etc.).

Simulink is a well-known and established tool, providing convenient notation to abstract engineering concepts with a user-friendly simulation interface. Thus, it is a very efficient tool for engineers to prototype, design and implement their part of the system. In the context of the space industry, it is used in many engineering domains, from mechanical to robotics. In the present case-study, it was used to produce the GNC algorithms of a launcher.

B. TASTE

TASTE [3] is a project developed, maintained and supported by the European Space Agency. It aims at providing a MDE toolset for the production of safety-critical systems. For that purpose, it defines the system using three views:

1) The Data View (DaV) specifies data types and encoding functions used to communicate between system components using the ASN.1 [4] language (see listing [1] for an example). It corresponds to the external interfaces of the system, as specified in the ICD that specified all interfaces between the sub-systems (with their types, properties, etc.).

2) The Interface View (IV) enumerates system functions (what the system is doing), their requirements and constraints (timing, data protection, etc.) and interactions among them (communication channels) using the AADL [5] language. Function connections reference the Data View to specify types being used. As this model remains descriptive (a graphical sample is shown in figure [3]), it has to be associated with code that implements functions.

3) Deployment View (DeV) defines the execution platform (processors, buses, etc.) of the system, its configuration as well as deployment of functions (from the Interface View) into it. It also uses the AADL [5] for that purpose, an example is shown in figure [4].

These models are processed with their functional code to automatically produce system implementation (as shown in figure [1]) through the following steps:

- The Data View is translated into declarations (data types, functions) in the target code (C, Ada, etc.) so that we can use the same interfaces with different languages.
- The Interface and Deployment Views are processed to create an architecture code that supports the execution of the functional code. This aims at creating and configuring resources (tasks, mutexes, etc.) that reflect requirements specified in the interface view (period, deadline, etc.).
- The Integration Process compiles the functional code from the user with the architecture code generated from models, producing the program to be deployed on the execution target. This code is also automatically tailored to the target operating system. As for now, our toolchain supports several architecture (x86, SPARC, etc.) and different Operating System for safety-critical systems such as Linux or RTEMS [6].

![Diagram](image)

**FIGURE 1:** TASTE development process

C. Integration of Software Models into Architecture Models

Our process (figure [1]) automatically deploys application code (written by domain-specific users, such as electrical/mechanical engineers) on top of architecture code. The former can be designed using either regular (C, Ada) or modelling (Simulink, SDL, etc.) languages. Our toolchain automatically generates glue code that connects functional blocks, enabling communication between code written with different languages and executed on heterogeneous architectures.

However, when integrating application models (such as Simulink), algorithms requirements must comply with the architecture constraints (timing requirements of the Interface View, interfaces definition of the Data View, etc.). Using a traditional, manual integration, no check is performed, lack of compliance between models is discovered either after integration (at best), or during execution (at worst). The following section contains the description of a complete case study we conducted in order to experiment and validate the use of a model-based approach with the TASTE tools.

III. CASE- STUDY AND FEEDBACK

A. Overview

In order to validate our approach and tools, we have built a system to support the validation of the navigation algorithms


(GNC) of an onboard launcher software. In that context, we have developed (or reused) a large set of Simulink models that represent the environment of the software: the sensors and actuators on one side, and the flight dynamics on the other side. This way we have the means to generate realistic data at runtime to feed the control laws and run in closed loop.

On the other side, we have the flight code of the control laws in Ada language. The challenge is to put both pieces together, and make them run on different platforms: first natively on Linux to test the integration, and later on a mixed platforms (x86/Linux for the environment models, and Sparc/Leon for the control law running in real-time). At runtime, we want to observe data and plot the control laws’ main parameters. TASTE provide the means to achieve these goals.

B. Application modelling

Modelling the application with TASTE was done in three steps

1) specify the data types in ASN.1 to describe the messages exchanged between our functions (listing 1)
2) capture the logical architecture of the system (figure [3]) into an Interface View that references the Simulink model (shown in fig:simulink-model).
3) model the deployment of the application, by mapping the functions onto hardware components, and connect them with buses (figure [4]).

From these models, TASTE generate skeletons, that consist in empty code blocks that would contain the application (Simulink models). Once these code blocks have been filled by the user, tools automatically create all the glue code that is necessary to implement the system (communication, etc.) without having to manually tweak the interfaces. This auto-connects blocks from different implementation languages (Ada, C, Simulink, etc.) smoothly, without having to change the communication mechanisms. Using these tools gave us guarantee that there would be no inconsistencies in data representation for each block. This allows to start running simulations very quickly and make rapid prototyping of embedded application without having to tweak application code.

C. Feedback

The experience gives us the ability to produce a large scale launcher simulator that we are now running to cross-check the control laws of our launchers. Until now, we did most of these simulations entirely within the Matlab/Simulink environment. It is also a very powerful and effective approach; the drawback is that the execution is not representative of real targets, and that the integration of the code later on in the real onboard software can become much more difficult when done at a later stage. Using automated tools to make the models and code integration from the very early stages...
of the development brings significant added value and makes things simpler to integrate for non-software people.

However, we experienced two major issues when integrating Simulink code on the real target. First of all, the generated code from application models is not fully consistent with the models specification, leading to errors that were not seen during simulation. This brings us to debug the generated code by hand to discover the problem. This should be automatically detected by the tools. Also, another issue was the programming skills of engineers: the toolchain required adaptation of existing Simulink models to the TASTE interface. This job requires some model refactoring to fit TASTE and Simulink models and was not easy for non computer-scientist engineers. As a result, we spent a lot of time explaining the design process to engineers so that they can use our toolchain.

IV. CONCLUSION

This article presents our feedback about the issues of the deployment of software models with architecture, particularly regarding potential errors that are introduced during the integration phase, at the latest phases of system production. These experiments were done in the context of internal projects at the European Space Agency, while integrating GNC algorithms (designed and tested using simulation functions from Simulink) with an execution runtime.

These issues convince us to strengthen the overall development process and propose new functionalities that aim at checking system compliance between execution and simulation. For example, being able to monitor system interfaces and check correctness between simulation and execution would help developers but also support the overall development process, providing artifacts required for system validation.

A. Perspectives

Verification between models could also be introduced earlier in the development process, prior to system integration. For example, it would be possible to check compliance between heterogeneous models before generating or integrating code. This engineering effort, even if technically feasible, would require a static analysis of the model, which requires a huge maintainance effort due to the number of application languages supported by our toolchain.

sectionAknowledgment

The authors would like to thank the VEGA project and the TEC-ECN and TEC-SW laboratories for their support and contributions to this paper.

ACRONYMS

ASN  Abstract Syntax Notation
ICD  Interface Control Description
GNC  Guidance and Navigation Control
MDE  Model-Driven Engineering

TASTE  The ASSERT Set of Tools for Engineering

FIGURE 4: Deployment View of our case-study

REFERENCES


