32nd Annual INCOSE international symposium
hybrid event
Detroit, MI, USA
June 25 - 30, 2022

From Model-based to Model and Simulation-based Systems Architectures – achieving quality engineering through descriptive and analytical models

Pierre Nowodzienski
Thales Corporate Engineering
pierre.nowodzienski@thalesgroup.com

Juan Navas
Thales Corporate Engineering
juan.navas@thalesgroup.com

Abstract. Systems architecture design is a key activity that affect the overall systems engineering cost. Hence it is fundamental to ensure that the system architecture reaches a proper quality. In this paper, we leverage on MBSE approaches and complement them with simulation techniques, as a promising way to improve the quality of the system architecture definition, and to come up with innovative solutions while securing the systems engineering process.

Introduction

System architecture is a key engineering artefact in systems engineering. It permits engineers to reach a common comprehension of the expectations of their customers, to orchestrate the design of the subsystems and components to reach the system purpose, to compare alternative orchestrations, to propose solutions that are fitted to stakeholders' expectations, and to ensure that the engineering outcomes are compliant to these solutions.

The information contained in system architecture deliverables directly impacts (i.e. is an input for) a large part of other engineering activities and their related outcomes; and in complex systems engineering practice the system architecture is often considered as the backbone of a large part of the engineering process and beyond, covering the whole system life-cycle. Hence, ensuring a good architecture quality is strongly contributes to securing the further engineering activities.

Model-based Systems Engineering (MBSE) approaches have proven their value on improving the quality of system architectures. When properly set up, MBSE practices and tools can support the architects and guide them through systematic workflows that reduce the risk of inaccuracy, inconsistency and incompleteness of the design. However, our experience shows that in some contexts the current MBSE capabilities do not suffice to reduce these risks as required. This is due to both the increasing complexity of the systems under design, and to the increasing complexity of the engineering workflows and organizations put in place to develop our systems.

As systems engineering practitioners, we acknowledge the existence of extensions of MBSE approaches that make use of simulation to address these issues. However, we have identified a lack of formal, concrete and effective proposals regarding the extension of MBSE methodologies to embrace analytical models.

In this paper, we present how, and under which conditions, simulation techniques can be articulated with MBSE methodologies so to fill the gaps of current MBSE approaches in ensuring proper quality architecture designs. We start by providing a necessary background on architecture design, MBSE

for architecture design, and simulation of architecture design. Then we describe the objectives and the limitations of current MBSE approaches for architecture design, and present how simulation can contribute to overcome these limitations. A set of good practices for simulation is presented, along with an analysis of a case study that illustrates the proposals of this paper.

Background

Systems Architecture Design

A system architecture is defined as the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution [ISO/IEC/IEEE 42010]. The standard specifies the manner in which system architectures are organized and expressed.

An architecture is defined with regards to a set of *stakeholders*, which are individuals, teams, organizations or any other kind of entities having an interest in the system; e.g. if the system of interest is a residential building, examples of stakeholders are the promoter, the future homeowners, the future administrator, and the architect herself. A *concern* is a matter of relevance or importance regarding the system of interest to a stakeholder; e.g. one of the architect concerns would be to design the best possible atmosphere in which to live.

A (stakeholder) *perspective* is way of thinking about an entity, especially as it relates to concerns; e.g. how future homeowners may circulate across the residential building. An architecture *aspect* is a unit of modularization of concerns within an architecture description, capturing characteristics or features of the entity of interest; e.g. the organization of common areas in the building and in its green zones. An architecture *view* is an information item comprising part of the architecture description; e.g. a plan of the green zones and its circulation paths.

Model-Based Systems Architecture Design

Model-Based Systems Engineering (MBSE) is the formalized application of modelling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases. [INCOSE 2014]. In addition to providing an increased rigor in these engineering activities, one essential objective of a model-centric approach is to provide authoritative sources of truth that can be shared with all stakeholders.

Overview of the Arcadia method. Arcadia is a model-based method devoted to systems, software and hardware architecture design [Voirin 2017]. It describes the detailed reasoning to understand and capture the customer need, define and share the product architecture among all engineering stakeholders, and validate early and justify the design. Arcadia can be applied to complex systems, equipment, software or hardware architecture design, especially those dealing with strong constraints to be reconciled such as cost, performance, safety, security, reuse, consumption of resources, mass, etc. It is intended to be embraced by most stakeholders in system/product/software/hardware definition as their common engineering reference. It has been applied in a large variety of engineering contexts for more than 10 years.

Inspired on [ISO/IEC/IEEE 42010], the Arcadia method defines a set of perspectives that the system architect can adopt when designing the architecture. The four main perspectives are summarized in

Figure 1. Arcadia enforces a clear separation between the capture and analysis of the system context and needs (the operational analysis and system need analysis perspectives), also called here *need perspectives*, and the design of the solution (the conceptual and physical architectures) also called here *solution perspectives.*

	Perspective	Objective
NEED & CONTEXT PERSPECTIVES	Operational Analysis	What the stakeholders need to accomplish
	System Needs Analysis	What the system has to accomplish for the stakeholders
SOLUTION PERSPECTIVES	Conceptual Architecture	How the system will work to fulfill expectations
	Finalized Architecture	How the system will be developed and built

Figure 1: Arcadia engineering perspectives

When following the Arcadia method, the system architect is led to consider five main aspects during the process of defining the system architecture:

- The *purpose* – the reason to exist of the entities, including the system: the services it provides in different contexts of usage, so to fulfill stakeholders' expectations and provide them with valuable solutions.
- The *form* – the entities that are considered during the different contexts of use of the system, including the constituents of the system; here two sub-aspects may be considered: the *structure* of the architecture and the *interfaces* between its constituents.
- The *behavior* – the expected behavior of the entities in the context of usage, including the expected behavior of the system and of its constituents; here two sub-aspects can be considered: the *functions* that emerge from a functional analysis, and the *modes & states* of the constituents.

These aspects shall be considered for every perspective stated before. In addition to these aspects, Arcadia can be extended through viewpoints that address other aspects such as safety or cybersecurity, by defining new concepts, views and practices, and how those depend on or impact the Arcadia standard concepts, views and practices.

The maturity of a system architecture (or of a reference architecture [Cloutier 2010], especially in a Product Line context [Oster 2016]) can be evaluated by considering the aspects that have been addressed in the architecture design matrix shown in Figure 2.

As the lack of properly tailored tools has proven to be a major obstacle to the implementation of MBSE in industrial organizations [Bonnet 2015], Arcadia is recommended to be implemented using the open-source modelling workbench Capella, whose diagrams are inspired from SysML and that has proven suitable for systems engineers with diverse backgrounds and skills [Capella 2021a].

Overview of Arcadia concepts. In this paper we will use a subset of the concepts defined by Arcadia. For a sake of clarity, this section provides a brief definition of them. A c*apability* is an entity's

ability to supply a service. A system capability represents a usage context and is characterized by a set of *functional chains* and *scenarios* that describe the system behavior in a particular usage.

Both functional chains and scenarios reference *functions*, which are actions, operations or services performed by entities – the system, one of its components, or also by any other entity interacting with the system. They also reference *functional exchanges*, that express possible interactions between source and target functions, and which are further characterized by e*xchange items*, which reference the elements routed together during an interaction.

Entity's behavior can also be represented by *modes*, which are behaviors expected under chosen conditions; *states*, which are behaviors undergone by entities in conditions imposed by the environment; and *transitions*, which are changes from one mode/state to another.

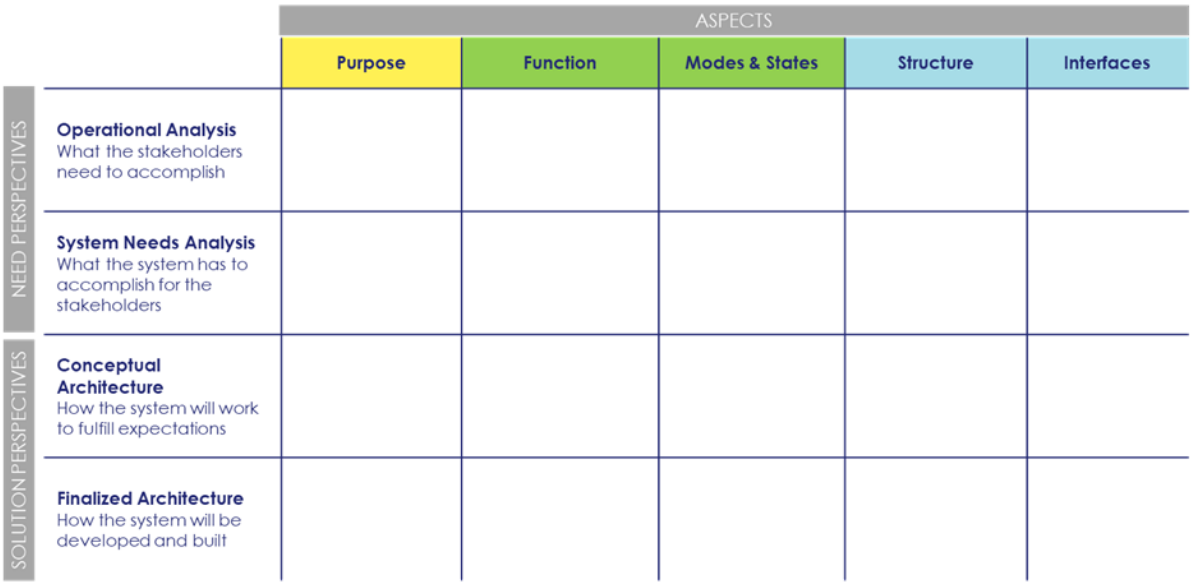

Figure 2: Arcadia architecture design matrix

Simulation of an architecture design

[Wippler 2018] provides a broad definition of simulation as a cognition process that predicts effects, including any form of deliberation based on a model, and not only the use of a model executable by computer means. This permits us to consider simulation as a component of a control mechanism and hence as a valuable means for reaching a system architecture design that satisfies the system objectives and the stakeholders' expectations. Figure 3 illustrates the role of the simulation in the architecture process in the control loop.

By considering simulation as part of the engineering control loop, we can identify the contribution of simulation in the architecture design process, both in terms of reaching the objective quality of the system, and of the time required to reach this objective quality.

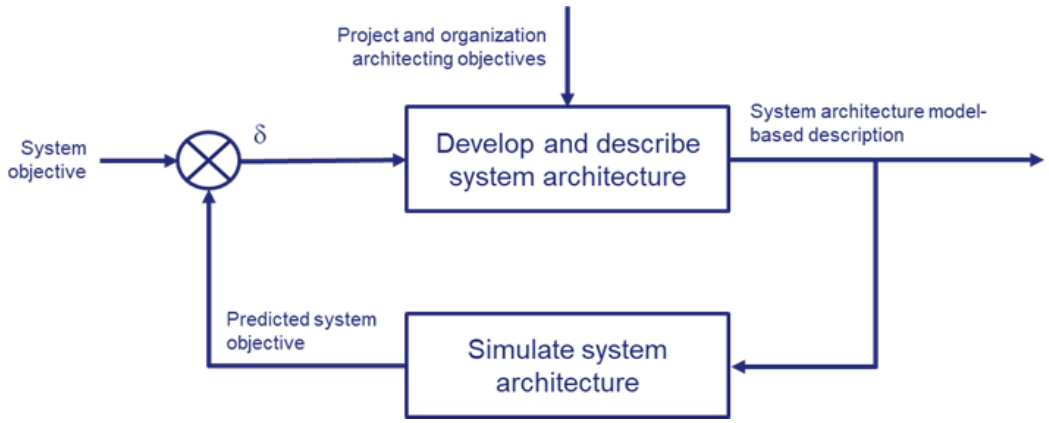

Figure 3: Regulation role of simulation in the architecture design process

Regarding reaching the objective quality of design, experience shows that 50% to 70% of the total of design errors made during system development lifecycle are introduced before implementation phase, mainly during Orientation and Design Phase. It means that we expect at least half of all the design errors already exists at the Preliminary Design Review (PDR). We can expect the quantity of errors to increase with the increase of system complexity the industry is facing, according to the metrics the Aerospace provides regarding the growth of embedded software complexity correlated with the number of Source Lines of Code (SLOC) [Filho 2018].

Regarding the time required to reach the objective quality, simulation contributes to reach it sooner, reducing the design error costs. As presented in [Stecklein 2004], design error fixing costs increase exponentially with project phase: if the cost of fixing a requirements error discovered during the requirements phase is defined to be 1 unit, the cost to fix that error if found during the design phase increases to 3 — 8 units; at the manufacturing/build phase, the cost to fix the error is 7 — 16 units; at the integration and test phase, the cost to fix the error becomes 21 — 78 units; and at the operations phase, the cost to fix the requirements error ranged from 29 to more than 1500 units.

Whereas the control loop is a useful pattern for identifying the role of simulation in architecture design, not all simulation mechanisms have the same effectiveness, and each mechanism requires different resources to be mobilized to ensure their effectiveness. Three existing mechanisms are:

- Workshops in which experts deliberate about the expected behavior of the system are relatively easy to perform, but its effectiveness strongly depend on factors that are difficult to assess, such as the skills of the experts and their ability to work together.
- Automatic validation rules, such as the ones provided by MBSE tools like Capella, are more reliable as they are based on the know-how of many experts and the return of experience of past projects. However, they only cover some quality aspects of the design, often its correctness and consistency.
- Computer-assisted simulation, in which executable models are created or generated from design models, is a powerful mechanism that may cover a large scope of quality aspects. However, the cost of putting in place and maintain simulation mechanisms is also higher than the previous ones.

In the rest of this paper, we will focus on this third mechanism. From now on, the term “simulation” alone refers to computer-assisted simulation.

Architecting objectives, current state and limitations

Architecture design objectives

As the architecture design strongly affects a large set of the engineering activities, both the quality of the system architecture artefact and the efficiency of the architecture design activity are key optimization levers for engineering organizations.

The usages of the architecture are manifold, and depend heavily on endogenous parameters such as the engineering organization, the engineering process and practices, the engineering teams, the industrial domain constraints and regulations, the expectations of the customers in terms of engineering activities, among many others. However, our experience shows that the objectives of engineering organizations with regards to the architecture design activities can be categorized in:

- *Share* – improve communication, reduce ambiguities and reach, as fast as possible, a common understanding of the system's purpose, form and behavior; e.g. use model-based architecture views to support discussions with customers.
- *Secure* - guarantee the quality of the engineering data, hopefully at any moment of the engineering process and beyond; e.g. use architecture models as the backbone for textual requirements elicitation, analysis and documentation [Bonnet 2019].
- *Automate* – automate the execution of the engineering processes to improve its efficiency, through (i) automatically generating engineering artefacts from architecture models, and (ii) automate architecture tasks; e.g. generate document deliverables from models, generate software modules from architecture models, automate the exploration of solution spaces.

The Share objective of architecture design deserves a few more words. This one is often despised and reduced to "having good-looking diagrams", ignoring that:

- Improving communications requires a common (architecture) language and to use a common vocabulary, which is not a trivial task.
- Fluency in (technical) communication drastically reduces the time required to understand what the other says, enables active listening attitudes, and ultimately improves the efficiency of collaboration in engineering.
- Clear, concise, properly organized and well documented architecture design models and views, can have a positive impact on the perceived quality of the information contained in them, and on the perception of quality of the system itself [Iandoli 2018] [Bateman 2010].
- Experience shows that the Share objective is often the first one that is set by engineering teams with few or no prior knowledge on MBSE practices and tools. This often means that the Share objective is their first action of a wider change management initiative in their organization (e.g. a Digital Engineering transformation). The results of these first change management actions often determine what happens next.

Current state and limitations

Architecture design descriptions, such as the ones that can be obtained using the Arcadia method and its associated MBSE tool Capella, allow engineers to address the objectives presented above. Indeed:

- The systematic use of Arcadia perspectives in Capella (cf. Fig. 2), from Operational Analysis to Physical Architectures and Product Breakdown Structure definitions, permit engineers to build a complete and consistent digital thread from the expectations of their stakeholders to the detailed definition of system components and expected behavior. This is particularly important both for business; to ensure that the expectations of future customers are properly addressed, and for certification concerns in mission-critical systems; to ensure that high-level safety and security concerns are properly addressed by lower-level components and their implementation, and that they can be validated.
- The systematic use of Arcadia aspects (cf. Fig. 2) permit engineers to ensure that each perspective is analyzed consistently and exhaustively. Arcadia aspects complement each other: e.g. by performing an analysis of components' modes and states, a previously developed functional analysis is completed through the addition of new functions or the clarification of the textual requirements attached to them.
- The kinds of model views proposed by Arcadia were defined in close collaboration with systems engineering practitioners. Several of them are based on or are the same as SysML ones – such as modes and states machines or sequence diagrams – while others correspond to diagrams that have been drawn by systems engineers for many years – such as views presenting multiple layers of functional decompositions, or views presenting the structure, functional and interfaces aspects together [Capella 2021b].

Nevertheless, experience on implementing MBSE in the field has identified a set of limitations of the current tooled-up practices:

Describing the expected behavior. One of the main goals of architecture design is to reach correct, complete and error-free descriptions of the system behavior that is agreed with the customer, and of components behaviors as designed by architects and agreed by components' engineering teams.

Arcadia concepts and methods, the fact that these methods are embedded in tools, and tool-specific features such as validation rules, all contribute to reaching correctness and completeness to a certain extent. However:

- Ensuring a full consistency between Arcadia aspects, and particularly between the functional analysis and the modes and states analysis, require the execution of cross-checks that in many cases (i) are specific to the project's needs, and (ii) cannot be specified and automated easily.
- In contexts such as Systems of Systems (SoS), or systems made mainly of subsystems, architecture models alone do not suffice to understand and master the behaviors that emerge from the composition.
- In contexts on which the system integrates components that are capable of learning and of adjusting their behavior to improve their efficiency (e.g. AI-powered components), current means for describing the system behavior need to be extended to take into account different ranges of operation derived from the integration of these components.

Ensuring the quality of the integration, verification and validation (IVV) strategy. In many of our engineering contexts, IVV strategies shall be defined as soon as possible, in order to secure the testability of the design, anticipate the means that will be required for IVV tests, and secure the execution of IVV campaigns. Therefore, architecture design is a major input of the definition of IVV strategies.

Arcadia concepts such as Capabilities, Functional Chains, Scenarios and Functions provide the backbone for defining IVV strategies. But when the engineering goals include securing the completeness of the design that will be implemented and of the IVV strategies by driving IVV campaigns earlier, current means such as peer reviews and validation rules do not longer suffice.

Making informed design decisions. Architecture design is well-suited to consider the multiple concerns that need to be addressed to ensure that a system satisfies the expectations of its stakeholders. Therefore, the architecture is in a privileged position to make trade-offs between multiple concerns and to optimize the design while considering these concerns together.

Although the existing mechanisms for extending the scope of models by developing concern-specific extensions [Langlois 2017] has permitted to integrate concerns to the architecture model such as safety [Bitetti 2019] and cybersecurity [Navas 2019] among others, trade-offs and optimizations remain tasks that rely strongly on the capacity of engineers to integrate non-formalized concerns into their analysis. Furthermore, trade-offs and optimizations often require frequent trial-and-error cycles that are poorly supported today.

Implementing Agility and DevOps. The advent of Agile and DevOps approaches in complex systems engineering, and the cultural change that comes with them, has had a strong impact on engineering practices. To name a few of them: the increasingly concurrent execution of ideation, design, implementation, IVV and operational feedback activities; shorter cycles of incremental and iterative development; and more frequent contact with customer, end-users and other stakeholders.

Agility and DevOps increase the need to respond to, as soon as possible, the impact that customers and end users' requests, or changes in operational contexts, may have on the system at operation. For the architecture design activity, this means evaluating how design modifications would change the systems behavior at execution and the way the systems are operated; and to be able to perform such evaluations frequently and incrementally.

Although architecture models have proven their contribution to the implementation of agile approaches on complex systems engineering [Navas 2020], current practices and tools do not suffice to (i) demonstrate to the customers the value that is acquired at each increment (e.g. by demonstrating the intended behavior of the system at operation), (ii) implement design approaches founded on e.g. Behavioral Driven Development (BDD) [Solis 2011].

Filling the gaps with Simulation

One major root cause of the limitations identified above is the fact that architecture models today are mostly descriptive, and that prediction capabilities are limited. Descriptive models are different in nature from predictive ones, as they retain “gray areas” of ambiguity, and do not need to perform as accurately as the predictive models. Although allowing designers to define their architecture while keeping some gray areas has proven useful at introducing model-based engineering approaches in large organizations, these gray areas may induce negative consequences in the engineering contexts presented above.

To overcome all these limitations, simulation offers capabilities to enable engineers to access the behaviors of the solution being designed. Simulation provides to an engineer an executable virtual object that can be exploited to enhance communication between all stakeholders, to further evaluate and analyze the solution definition as well as to expand and automate the exploration of the solution space.

Improved communication through simulation

Descriptive models are often used to illustrate how the system behaves, either with Modes & States machines or Scenarios and Functional Chains. The overall behavior of a system is then captured through a collection of such views, each one focusing on describing the behavior in a unique situation. Additionally for these types of views there is a compromise between completeness of the behavior description (completeness = unambiguous + full coverage of the scenario) and the easiness of readability.

The spreading of the information across views and the lack of readability or unambiguity (due to the above compromise) makes it hard to ensure a common understanding of the system behavior among stakeholders. Furthermore, the reader actually performs a model execution mentally to figure out what is the behavior described. In this case, the execution engine (i.e., the human brain) is different from one reader to another and thus leads to a variety of simulation results (i.e., interpretation) and not repeatable that dramatically increases the risk of interpretation divergence among stakeholders.

Simulation provides unambiguous means to figure out the system behavior such as time dependent curves, 2D and 3D rendering, quantitative results based on post-processing of simulation outputs. The use of one or another depends on the purpose of the communication, the type of information to be shared and the profile of the persons the information is shared with.

In addition to the simulation results themselves, the use of interactive simulations let the stakeholders (engineers, end-users, customers, managers...) experience the future system for themselves, confirm or refine the expected behavior, or to get a better understanding of the system behavior for detailed design. The use of simulation to improve communication is mainly facilitated by “Simulation as a Service” capabilities as it provides easy access to simulation means (execution and visualization) while complying with security constraints.

For engineers in charge of the detailed design, an executable specification provides extra-benefits by securing the understanding of the expected behavior. The simulation environment provides similar capabilities to the one provided by software IDE such as breakpoints with pause conditions, step forward and backward, model animation with highlighting or coloring, toggling data and much more. This set of debugging features is helpful in understanding the internal mechanisms and interactions between functions from which the system behavior emerges.

Another observed benefit is the reinforcement of concurrent engineering and collaboration. For system engineers, being able to share very early an executable specification that reflects certain aspects of the system with other engineering specialties is a powerful mean to enable the other specialties to start activities earlier and concurrently, to quickly iterate on the system definition as early as possible. Without the use of simulation these activities would have been performed far later or in a less complete and detailed way. For instance, an IVVQ engineer can start experimenting test cases with a virtual test bench and virtual object that mimics the system to test. This way the test campaign gets more mature and is verified before the execution of the test campaign on the real system. Actually, the test campaign elaboration and verification activity performed early in a virtual environment also offers opportunity to not only verify the test cases but also to provide feedback to Systems Engineers on defects detected during the test's execution. In this case we can see the simulation as a concrete enabler for co-engineering. Another example is for safety engineering. Although the use of descriptive models is already a proven way to perform safety analysis [R. de Ferluc 2018] early in the design process based on the architectural descriptive model produced by the System Engineering

team and to feed the architectural model back with the analysis results, the use of simulation offers more advanced analysis capabilities for reliability, availability, maintainability, testability and safety analysis [A. Garro 2012] that can be integrated in a feedback loop process on system architecture design activity (cf. Fig. 3). This feedback loop can occur at any stage of the system architecture design.

Secure the design through simulation

Leveraging simulation in the engineering process enables engineering project teams to early and continuously Integrate, Verify and Validate (IVV) the solution under design (i.e. through execution of a subset of the system definition, called "item under design"). Having the capability to simulate the item under design at any time enables engineers to perform unit-test and integration-test to verify and validate the item under design after each small design step.

This brings two major benefits. The first benefit is that it drastically reduces the overall solution cost reducing the time between the introduction of a design error and the fis, as (i) the earlier an error happens the more the error might cost; (ii) the majority of errors occur before starting the implementation, which is relatively early in the development process; and (iii) the situation gets worse as the complexity of the systems increases.

Given this situation, it is critical for engineers to have means to test their design before the implementation, hence before it is physically accessible. Having access to virtual test means including a virtual item under test and virtual test bench at any time through the development process is an answer to minimize the cost of errors.

A concrete example is a System Engineering team who captured the logical architecture of the system (structural and functional aspect) by following Arcadia methodology and language. The systems engineers wanted to be more precise and unambiguous regarding the dynamics of exchanges between functions, as there were many intricate feedback loops and parallel branches. One System Engineer started to build this simulation and it appeared the intellectual process required to build the simulation raised a lot of questions challenging the content of the logical architecture. He found ambiguities in the behavior description and missing data in interfaces. These errors have been fixed very early, avoiding later discovery by other engineering teams who would have spent a lot more time dealing with these errors before reaching out the Systems Engineering team.

The second benefit is that it enables wider solution space exploration and alternatives comparison. Due to project planning pressure, systems engineering teams rarely invest the time required to fully explore the solution and design space as well as comparing alternatives. These are activities that lead to producing designs that will not retained, and simulation reduce the time required to evaluate designs and determine which one is better than others, hence making the design space exploration and alternatives exploration more efficient. To further speed up the exploration, the simulation model can be parametrized and integrated into an optimization loop to automate this exploration and to converge towards the best solution.

As described in the previous chapter ("Improve communication with simulation"), leveraging simulation not only helps to secure the design with early and continuous IVV but also permits IVVQ engineering teams to secure the test campaign concurrently with the design activity. This opens up new methodologies that can be applied to complex systems engineering such as Test or Behavioral

Driven Development, provided the IVVQ team can provide comprehensive virtual test means for the foreseen solution to the system engineering team.

Automate tasks with simulation models

Working with models, be they descriptive or executable, means working on digital data. Hence, access to and transformation of this data can be automated with programming languages. MBSE tools often provide tasks automation such as document generation, code generation (for software component interfaces mainly) or descriptive models generation – for system to sub-system transition or to initialize dysfunctional models for instance. We can extend the range of tasks and enrich them based on simulation models. Maybe the more valuable capabilities are the design space exploration automation, design optimization automation, early test execution automation and model coverage analysis automation [Camus 2016] to achieve high quality system definition.

Based on simulation models we can also enrich the document generation with parts of simulation models that would be non-ambiguous or with simulation results to better express a desired behavior. And last but not least, as simulation models are executable by nature, we can generate both the detailed software component interfaces along with the internal behavior of the component [Fleischer 2009].

Good practices proposals

As discussed above, there are many reasons why we would want to rely on simulation. Hence, adopting a simulation-based engineering process, i.e. the use of simulation to support all engineering activities throughout the product lifecycle, leads to the creation of a multitude of simulations. As for any engineering activities, keeping consistency and coherency across all engineering artefacts including simulations is a challenge. To overcome this challenge there are good practices to apply.

Separation of concerns. An all-in-one model merging descriptive architectural models and simulation may be considered as a response to the need of coherency and consistency. We encourage separation of concerns to keep architecture definition and simulations as distinct, differentiable but still coherent objects. This is to avoid the overload of architectural models that support specific modeling objectives owned and defined by system architects. Actually, the system architect often requires a simple model to support the design thinking process. At some point, the overload of information in a single representation becomes counter-productive to achieve system architecture goals. This choice is similar to the one existing in Hardware Engineering where there is a clear separation between modeling and simulation respectively named Computer-Aided Design (CAD) and Computer-Aided Engineering (CAE).

Frame simulation design in your MBSE methodology. To help organize simulations and to ease the coherency and consistency of them with other engineering artefacts, simulation models should contribute to meet objectives defined in the applied MBSE methodology. For instance, if a simulation is built in the context of the System Analysis, then this simulation must contribute to the System Analysis objective, i.e. to define what the system must accomplish for the stakeholders. Furthermore, the simulation model should respect the same representation point of view as the one adopted in the Arcadia perspective. Again, if we consider a simulation built at the System Analysis perspective, the way the system behavior is captured must be considered as a “black-box” specification and not an implementation specification for the subsystem engineering teams.

One simulation per type of concern. Considering a single simulation to answer all the questions the engineers would have throughout the product lifecycle would mean this simulation is a perfect virtual representation of the system in all its aspects throughout all its history. This is neither realistic nor achievable. Hence, we must consider the worst case that we would have one simulation for each engineers' questions that will occur during the product lifecycle. To reduce the number of simulations one proposal is to define typology of concerns and to build one simulation per type of concern or at least reusable simulation components for each type of concern. To ease the management of these simulations a proper governance is needed.

Maximize reuse of simulation assets. There are several ways to reuse simulation assets: (i) reuse existing simulation components to build a new simulation, (ii) reuse an existing simulation to answer different questions from similar concerns, (iii) reuse test harnesses and test cases throughout the development lifecycle. For the first type of reuse, we need to define simulation modeling rules that ensure reusability of components (e.g. interoperability formats such as FMU, HLA..., shared interfaces repository, …), we must also minimize the number of simulation tools to keep the co-simulation constraints as low as possible, and define a standard component-based simulation architecture. For the last type of reuse, we need to ensure that the simulation facilities offer seamless support for Software-in-the-Loop (SIL), Processor-in-the-Loop (PIL) and Hardware-in-the-Loop (HIL) to progressively transition from a full virtual testing to real system testing.

Automate the transformation of architectural model element to simulation model element. In order to avoid human mistakes in the design of a simulation model based on existing architectural model, we strongly encourage automating this model transformation. This provides confidence in the coherency and consistency between architectural models and simulation models. The automation can also integrate the traceability links to ease the impact analysis and justification activity. A subsequent advantage of such automation is the standardization of the approach to building simulation models in the engineering organization.

Case Study, Analysis and Discussion

To test the proposed approach, we used a hypothetical, still realistic, drone-based system capable of fulfilling multiple missions including the evaluation of the health of crops. Its architecture has been formalized in Capella following the Arcadia method, exploiting the Arcadia perspectives. The following paragraph is an excerpt showing how an expectation of a stakeholder (evaluating crops in very large fields) drives to key capabilities (navigation) and key system and subsystem requirements (mass, integrity).

- Operational Analysis – by identifying Operational Entities (OE) and the analysis of their perceived value (e.g. pains & gains), operational activities and processes. For instance, Farmers (an OE) are interested on evaluating crops in very large fields
- System Needs Analysis – by defining (i) the System Capabilities and related Functional Chains and Scenarios, and how they provide value to the stakeholders (the OE); (ii) the non-functional key system requirements. For instance, the drone-based system shall provide functional capabilities such as Navigation, Acquisition of data or Mission planning; key system requirements include integrity – related to the capability of navigating while avoiding obstacles.

- Logical Architecture – by allocating Functions to the conceptual Logical Components (LC) of the solution. For instance, both the drone and the ground station LC contribute to the navigation capability, including the obstacle avoidance integrity feature. Key requirement for the drone subsystem is mass – as an enabler to navigate across large fields during the time required to do an evaluation.
- Physical Architecture – by performing a detailed functional analysis and allocation of Functions to the concrete components of the architecture. For instance, detailed requirements regarding integrity (including those related to obstacle avoidance) would be allocated to the navigation processors, and mass requirements resulting from mass vs fuel capacity tradeoffs may be allocated to drone physical components.

In the case study we built simulation models to support architecture objectives of Operational Analysis, System Analysis and Physical Architecture perspectives. On the Operational Analysis, simulations are used to verify and validate the operational processes and to optimize them. We have used a modeling language inspired from eFFBD to represent in a simulation tool the operational processes. The execution of the simulation is based on discrete event simulation which is a convenient type of simulation engine for processes that requires not much behavioral details to get ready to run. Hence it appears to be very well suited for early simulation of processes.

On the Physical Architecture, we performed simulation to verify the correctness of the functional analysis especially regarding the definition of the functional exchanges and to identify potential ambiguities in the architecture definition. We also performed architecture optimization leveraging a parametrized simulation model. The purpose was to optimize several characteristics of the system to reduce the overall mass of the drone given a collection of operational missions to accomplish.

Focusing in the System Needs Analysis, we leveraged simulation to (i) verify and validate the modes machine supervising the system, (ii) specify the expected behavior of a subset of functions in a functional chain context, (iii) integrate these functions to build the functional chain and verify and validate this functional chain, (iv) ensure a full consistency between the functional chain and the system modes as they interact each other, (v) and finally validate the emerging behavior coming from these interactions.

Fig. 4 (top) shows the modes machine used for the case-study, which is the one supervising navigation modes of the drone. Fig. 4 (bottom) shows the functional chain used for the case-study, which involves the functions in charge of manually pilot the drone with an obstacle avoidance feature to preserve the integrity of the system. The transitions of the modes machine are triggered by functional exchanges involved in the functional chain and the modes update function parameters to adapt the overall behavior of the system given the active mode.

It shall be noted that for all perspectives, simulation results were injected back into the architecture descriptive model, ensuring the regulation role of simulation depicted in Figure 3. Two cases were identified, they are illustrated below with examples of a simulation of the physical architecture:

- The simulation results modified the value of a property of an existing architecture element – for instance, the mass ranges of the concrete physical components of the architecture, which are key requirements of the subsystems directly impacting the customer satisfaction (been able to evaluate large fields)

- The simulation results induced stronger modifications of the architecture, such as new functions, different allocations or new operating modes, that often require specific co-engineering actions – for instance, the early simulation of the obstacle avoidance feature led to these kinds of modifications, done under the responsibility of the system architect and involving several subsystems' architects.

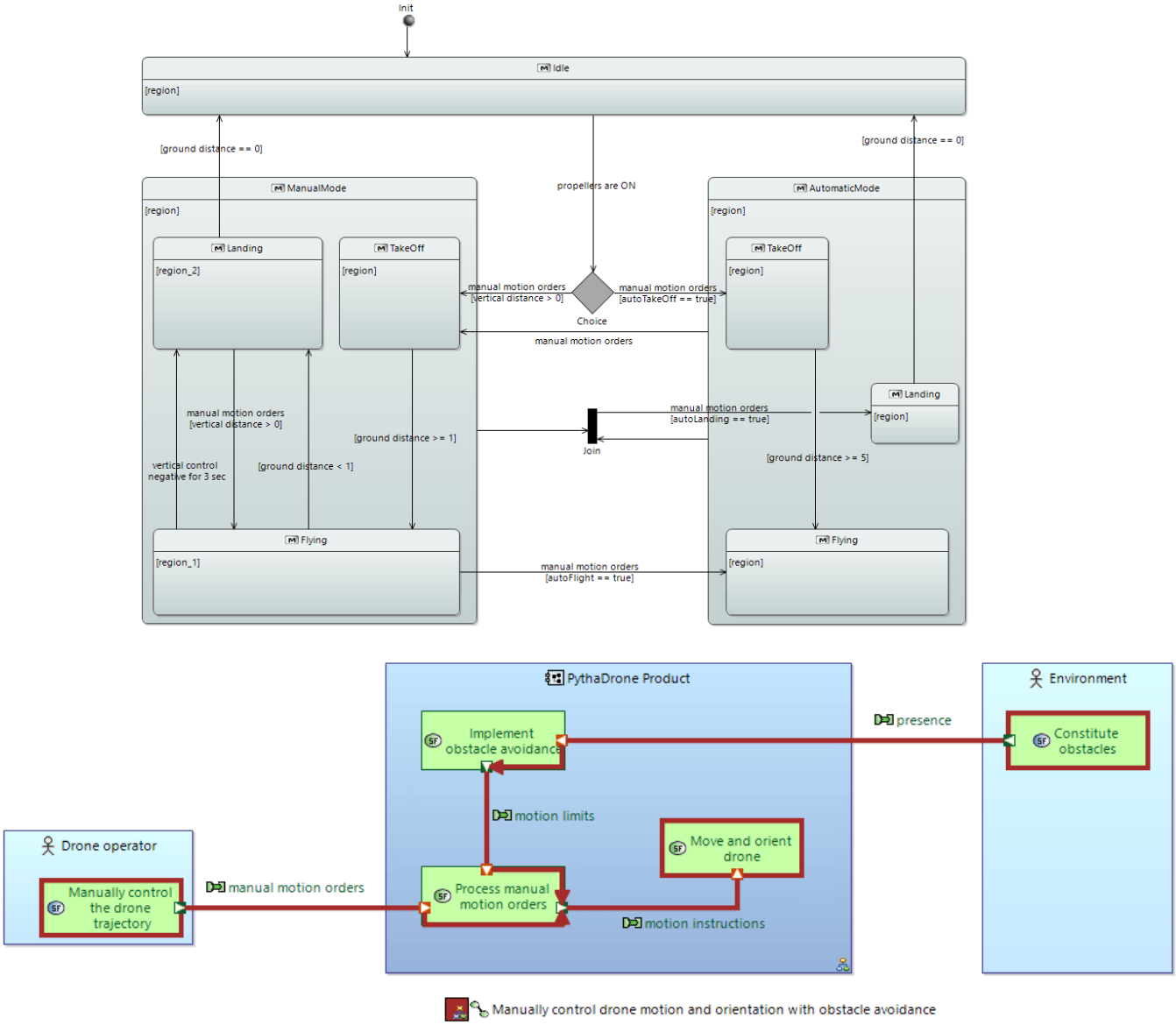

Figure 4: Drone-based system case study's mode machine (top) and functional chain (bottom)

Limitations of the descriptive architecture model

On this scope, the architecture model has limitations to completely capture the desired behavior and to ensure full consistency between modes analysis and functional analysis.

First, it is nearly impossible to specify with a descriptive model only the desired dynamic of the drone, i.e. how the drone reacts on an operator order. Similarly, it is hard to explain how aggressive

the response to an upcoming obstacle should be, or how the drone should behave when concurrent orders come from the operator and the obstacle avoidance functions.

Second, the content of modes and states machines must deal with a compromise between readability on one hand and formal expression and completeness on the other hand. By experience, the architecture model often favors the first aspect. Furthermore, even if the tool offers state machine simulation capability it is often at the price of detailed implementation effort. Hence, we must often deal with ambiguities and uncomplete specification.

Third, if the modes machine is growing in complexity, it becomes hard to verify it by mental simulation covering all the possible paths. Hence the potential is rapidly growing. Finally, it is hard to really capture the emerging behavior coming from the interactions between the functions, the supervising modes and the system states. It is also challenging to ensure full coherency across these three aspects of the system.

Simulation design workflow and benefits

In the context of the case-study, we use Simulink as the main simulation tool as this tool supports the simulation of discrete event systems, state charts, data flow (discrete or continuous time) and acausal systems (suited for multi-physics simulation). To support the simulation design, we have tooled-up the transition from Capella to Simulink. This tool (called Cap2SL) ensures the coherency, consistency and traceability of shared elements. To maximize reuse across simulation models, Cap2SL implements modeling rules favoring componentization and modularity.

In this workflow we consider two roles: the system architect is responsible of the architecture definition and generates a simulation request; the simulation engineer is in charge of building and running the simulation and providing data required by the request.

Case 1: Verify and Validate Modes machines. To build the simulation model from the modes machine defined in Capella, first we get an initialized version generated with Cap2SL. This simulation model captures the information stored in the Capella model. A substantial amount of data can be transformed applying a semantic mapping rule with no ambiguities. However other mapping rules might be applicable depending the modeling rules a company wants to apply and the meaning a company confers to modeling patterns. In case of potential ambiguities to map the semantics of the two tools, the information is transformed as textual information for the simulation engineer to have all existing information to build the simulation model. In our case, we must deal with:

- Transition trigger expressed in natural language such as “propellers are ON”. For such trigger, additional data manipulated in the system needs to be defined.
- Ambiguous guard expression: the data used to express the condition is part of several Exchange Items and these Exhange Items are carried by multiple Functional Exchanges. Hence, we don’t know if the data has a unique or several instance and if so, then which instance is to be tested in the condition expression.

For these two problems we already see value of simulation just in the process of building a simulation. It permits to challenge the system definition in front of a sort of a reality. Thus, to identify incomplete or too ambiguous specifications very early in the development lifecycle. Indeed, these issues would have been raised either by the subsystem engineering team or by the software team and

some of these issues are tightly connected with hardware because of the required data to capture by a sensor. Hence some of these errors would have led to costly rework and planning shift.

After some collaborative work sessions with system architect and simulation engineer the simulation engineer comes up with an executable model shown in Fig. 5. Concurrently, the IVVQ engineer develops a test harness (Fig. 6) reusing the interface definition generated by Cap2SL, and a test scenario based on the scenarios captured in Capella.

From this point, the early-testing activity can start. In the specific context of this case study, a lot more effort has been spent on fixing and improving the test scenario than on designing and fixing the item under test. We also used model coverage features offered by Simulink to identify missing scenarios. The early test raised a design error in the modes machine that was introduced in the Capella model and impacted the Functional Exchanges and Exchange Items definitions.

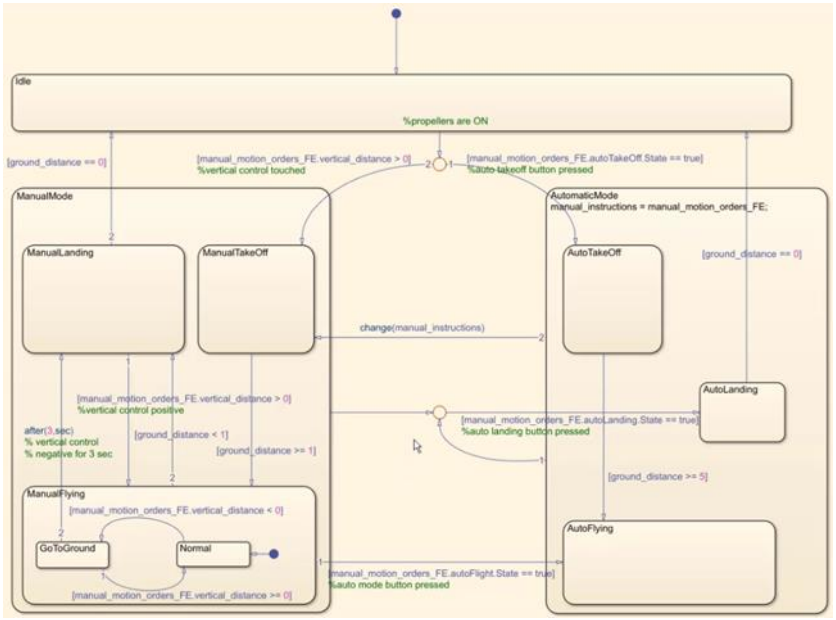

Figure 5: Drone-based system case study's mode machine executable model

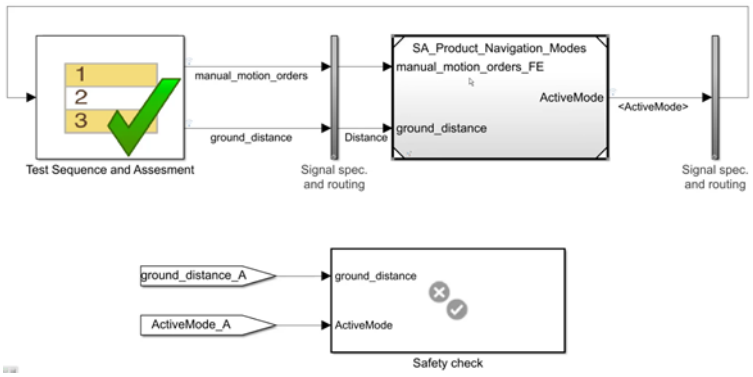

Figure 6: Test configuration

Case 2: Specify functions' behavior and verify functional chain. As for case 1, we first started by leveraging Cap2SL to initialize the functions models, the Functional Chain (FC) model and all the required data and interfaces. The FC model is made by aggregation of Function models individually

created and with their own lifecycle. At this point the FC model is compilation-ready. It means that the model does not contain design error that prevents to generate an executable software from the model. This already verifies the correctness of the Capella model on this scope.

From here we start work sessions with the system architect and simulation engineer to define the desired behavior of each function individually as the design model is not well suited to capturing this. As we are at the System Analysis perspective, we can keep things with a high level of abstraction leading to interactive and collaborative work session iterating quickly on the simulation model. The main challenge is to get a robust set of test scenarios to ensure we don't miss corner cases. Concretely, it is easier to specify expected behavior and corresponding scenarios at the FC boundaries than specifying the details of each function individually. Hence, we started by defining the test scenario for the FC and then working function by function in the same order than the sequence defined in the FC. For a given function involved in the FC, the system architect defines at a coarse grain level the expected behavior of the function. The test scenario is generated by executing the FC scenario and recording the outputs of the upstream functions. The key point is to quickly get a first executable FC model in order to leverage the better-defined FC expected behavior to identify errors in the individual functions' definition.

In our case we decided to develop an interactive control dashboard to let the system architect and other stakeholders somehow related to this functional chain to experience by themselves the system within the FC scope to provide feedback.

Case 3: Integrate Modes machine with Functional Chain. Once we get both the modes machine and the FC models verified, it becomes very easy to integrate them together, thanks to the componentization modeling rules we implemented in Cap2SL. The main concern we had was regarding how we should model the interactions between the modes machine and the functions since they do not concretely appear as interfaces in the design model. Either we decide to keep the Capella layout to make the system architect more comfortable to dig into the simulation model, or we make these interfaces visible in the diagram but by doing so we modify the interface definition of the functions. Since the beginning we favored keeping the layout of Capella for modeling in the simulation tool, so we continued with this approach for the last stage. However, this choice can be discussed. The simulation of this model did not raise any error; thus it contributes to increase our confidence in our system definition.

Feedback from the frontline

While this paper focuses on specific use-cases to illustrate the approach through concrete examples, the overall intention is to incorporate simulation activities in the daily system engineering work and to make it the new normal. According to return of experience from some company businesses already mature in this field, there is a measured project performance improvement when adopting simulation as the foundations for any design decision. Fig. 7 shows the correlation between the amount of system requirements covered by analytical models and the integration and flight tests qualification effort.

Fig. 7 also highlights an increase in the system quality with a drastically reduced technical debt. But this comes at the price of a multi-year journey transforming the organization, the culture, the skills, the processes and the tools. Last but not least, it is worth to note higher benefits happen when the engineering process is transformed as a whole instead of optimizing locally. In this project example,

a large part of the benefits come from the continuous reuse of test cases built at the early stage of the development lifecycle down to the ground field tests.

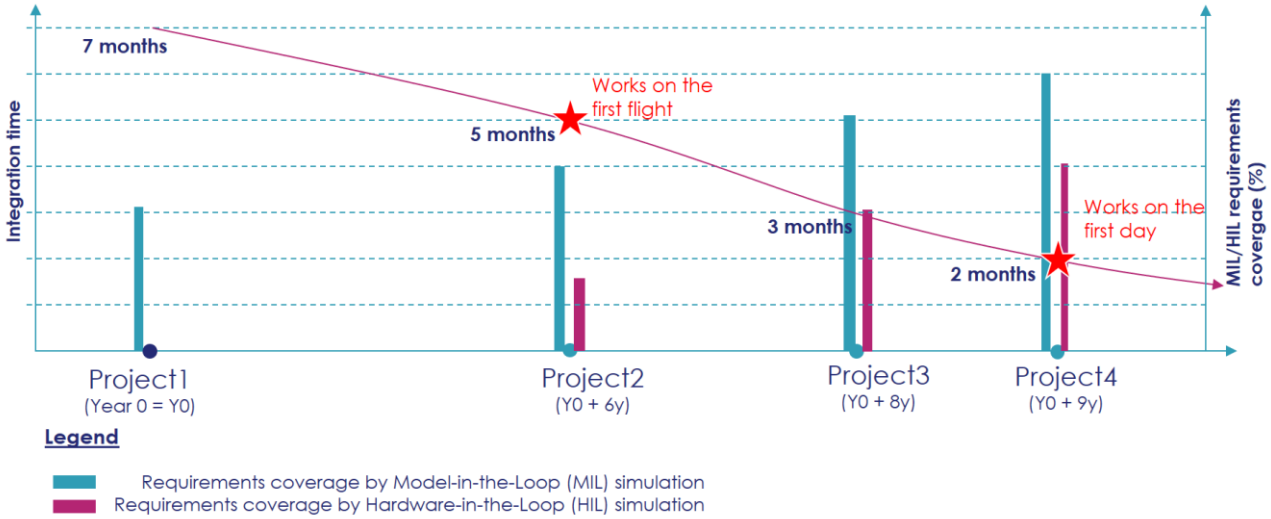

Figure 7: Evolution of integration and flight test qualification time over 4 projects in 9 years

Conclusion and perspectives

The use of simulation to support architecture and detailed design activities is an efficient means to improve quality, cost and planning while reducing risks. It is rare enough to be noticed as most of the time these three key project objectives cannot be improved together. This often comes at the price of an increase of the project expense on the upstream engineering activities, which is often perceived as a gamble from the project management perspective. However, this is not always true and some other experiments on other projects have demonstrated that the effort spent to specify the system with a simulation model is lower than doing it with textual requirements as a lot of effort is spent trying to identify all the missing cases, all the incoherencies and to make the textual requirements traced, coherent and consistent with the design model. This case-study has demonstrated clear benefits using simulation on top of design model. However, to be efficient, an automation tool is highly recommended as well as clear modeling rules that favor modularity and reusability.

One point of attention is when simulation is used as specification as the simulation can lead to over constrained specifications. Indeed, it is common to see an executable specification describing one nominal or ideal operating point. If the specification model is used as a scoreboard for equivalence testing, the probability of having the real system exactly match the behavior specified that way tends to zero. Hence it is critical as a system engineer to define margins in the expected behavior and to identify parameters' tolerance. The same way, as a simulation engineer it is as important to quantify the uncertainty of simulation parameters and the credibility level granted to simulation results.

Regarding the case study we intend to extend it to demonstrate other use cases. Among others, the most prevalent are:

- To handle the round-tripping. As simulation is used, in the context of this study, as a supporting activity of the architecture definition there is a need to go back and forth between the architecture definition and the simulation. This round-trip can be either simple and straightforward or more complex. The simple case is when simulation is leveraged for architecture

verification or for sizing and optimization. In these cases, the simulation request can clearly define what are the model elements of the architecture model to update (verification status, architecture parameters' value for instance). Hence, the update of these elements does not raise any particular issue. The complex case is when the simulation process raises some evolution requests on the architecture (ambiguities to resolve, missing data, incomplete definition...). This situation must be handled via a dedicated process to guarantee the integrity of the overall architecture definition. This process, when defined, can then be tooled-up to make easier the concrete update of the architecture model from simulation models.

- To perform simulation-based dependability analysis in accordance with both the architecture model and the Arcadia method.

Finally, this work shall be continued to confirm applicability and define good practices for use of simulation in different contexts, such as simulation for architecture optimization from the Orientation down to the detailed design, simulation in Product Line Engineering (PLE) contexts, and simulation in agile context.

References

Oster, C., Kaiser, M., Kruse, J., Wade, J. and Cloutier, R. 2016. "Applying Composable Architectures to the Design and Development of a Product Line of Complex Systems", Systems Engineering Journal, Vol. 19, No. 6, Wiley: NY.

Cloutier R, Muller G, Verma D, Nilchiani R, Hole E, and Bone M, 2010, 'The Concept of Reference Architectures', Systems Engineering Journal Vol. 13(1):14-27.

INCOSE, 2014, 'A World in Motion – Systems Engineering Vision 2025'.

ISO/IEC/IEEE 42010, 2011, 'Systems and software engineering — Architecture description'

Voirin J-L, 2017, 'Model-based System and Architecture Engineering with the Arcadia Method', ISTE Press, London & Elsevier, Oxford, 2017

Bonnet S, Voirin J-L, Normand V., Exertier D, 2015, 'Implementing the MBSE Cultural Change: Organization, Coaching and Lessons Learned', 25th INCOSE International Symposium

Voirin J-L, Bonnet S, Normand V., Exertier D, 2015, 'From initial investigations up to large-scale rollout of an MBSE method and its supporting workbench: the Thales experience', 25th Annual INCOSE International Symposium, 2015

Capella, 2022a, Capella Website: https://www.eclipse.org/capella/

Capella, 2021b, Equivalences and differences between SysML and Arcadia/Capella. Capella website: https://www.eclipse.org/capella/arcadia_capella_sysml_tool.html

Bonnet S, Voirin J-L, Navas J., 2019, 'Augmenting requirements with models to improve the articulation between system engineering levels and optimize V&V practices', 29th Annual INCOSE International Symposium, 2019.

ISO/IEC/IEEE 42010, 2011, 'Systems and software engineering — Architecture description'

Navas J., Bonnet S, Voirin J-L, Journaux G., 2020, 'Models as enablers of agility in complex systems engineering', 30th Annual INCOSE International Symposium, 2020.

Wippler J.L, 2018, 'Une approche paradigmatique de la conception architecturale des systèmes artificiels complexes, Université Paris-Saclay, 2018.

Iandoli, Luca, Letizia Piantedosi, Alejandro Salado and Giuseppe Zollo. 'Elegance as Complexity Reduction in Systems Design.' Complex. 2018 (2018): 5987249:1-5987249:10.

Scott Bateman, Regan L. Mandryk, Carl Gutwin, Aaron Genest, David McDine, and Christopher Brooks. 2010. Useful junk? the effects of visual embellishment on comprehension and

memorability of charts. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10).

Langlois B, Barata J, 2017. Extensibility of Capella with Capella Studio. Eclipse Newsletter. https://www.eclipse.org/community/eclipse_newsletter/2017/december/article4.php.

Navas, Juan & Paul, Stéphane & Bonnet, Stéphane. 2019. Towards a model-based approach to Systems and Cybersecurity co-engineering. INCOSE International Symposium 2019.

Lorenzo, Bitetti & Ferluc, Régis & Mailland, David & Gregoris, Guy & Capogna, Fulvio. (2019). Model Based Approach for RAMS Analyses in the Space Domain with Capella Open-Source Tool. 10.1007/978-3-030-32872-6_2.

C. Solis and X. Wang, "A Study of the Characteristics of Behaviour Driven Development," 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, 2011, pp. 383-387, doi: 10.1109/SEAA.2011.76.

Paulo Soares Oliveira Filho, "The growing level of aircraft systems complexity and software investigation", Published 2018

R. de Ferluc, F. Capogna, G. Garcia, O. Rigaud, D. Demarquilly, et al.. MODEL BASED SAFETY ASSESSMENT (MBSA) IN THE SPACE DOMAIN WITH CAPELLA OPEN-SOURCE TOOL. Oct 2018, Reims, France. ffhal-02064930

A. Garro, A. Tundis , "Modeling and simulation for system reliability analysis: The RAMSAS method", Published 16 July 2012, Computer Science, Engineering, 2012 7th International Conference on System of Systems Engineering (SoSE)

Jean-Louis Camus, Carole Haudebourg, Marc Schlickling, Joerg Barrho. Data Flow Model Coverage Analysis: Principles and Practice. 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Jan 2016, Toulouse, France. ffhal-01262411f

Fleischer, D., Beine, M., and Eisemann, U., "Applying Model-Based Design and Automatic Production Code Generation to Safety-Critical System Development," SAE Int. J. Passeng. Cars – Electron. Electr. Syst. 2(1):240-248, 2009

Stecklein, Jonette M., James B. Dabney, Brandon N. Dick, Bill R. Haskins, Randy Lovell and Gregory Moroney. "Error Cost Escalation Through the Project Life Cycle." (2004).

Biography

Juan Navas is a Systems Architect with +12-years' experience on performing and implementing Systems Engineering practices in multiple organizations. He is in charge of the Thales Corporate Modelling & Simulation coaching team and dedicates most of his time to training and other consulting activities worldwide, for Thales and other organizations. He accompanies systems engineering managers and systems architects implement MBSE approaches on agile operational projects, helping them define their engineering schemes, objectives, and guidelines. He holds a PhD on embedded software engineering (Brest, France), a MSc Degree on control and computer science from MINES ParisTech (Paris, France) and Electronics and Electrical Engineering Degrees from Universidad de Los Andes (Bogota, Colombia).

Pierre Nowodzienski is a Systems Architect with 10-years' experience on performing and implementing Systems Engineering practices in multiple organizations. He is a Thales Corporate MBSE coach and dedicates most of his time to training and other consulting activities worldwide, for Thales and other organizations. He accompanies systems engineering managers and systems architects implement MBSE and Simulation-Based Engineering approaches on agile operational projects, helping them define their engineering schemes, objectives, and guidelines. He holds a MSc Degree on Mechatronics and Complex Systems from ENSEA (Paris, France).