33RD CONGRESS
OF THE INTERNATIONAL COUNCIL
OF THE AERONAUTICAL SCIENCES
STOCKHOLM, SWEDEN, 4-9 SEPTEMBER, 2022

ICAS 2022 SWEDEN

Heterogeneous Systems Modelling in Support of Incremental Development

Robert Hällqvist, Johan Naeser, Johanna Wallén Axehill, Erik Herzog

Saab Aeronautics
Linköping
Sweden

Abstract

Today, virtually all heterogeneous systems are developed with the help of advanced modelling applications supported by a multitude of simulators and test stations. At the same time, complex systems are developed using an incremental approach where initial capabilities are validated prior to proceeding to more advanced ones. For a complex system it is a fundamental observation that it is virtually impossible to accurately predict how long time the development of a new capability will take. Consequently, a development organisation need to maintain flexibility, allowing it to integrate individual capabilities as they become available. This paper introduces a development model tailored for supporting incremental development of safety critical systems.

Under the proposed development model it is essential to separate models based on their temporal characteristics. Architecture views – representing a desired future state, must be separated from models capturing design views – a more concrete view on the system, which in turn must be managed separately from virtual and physical realisations. The proposed framework is illustrated by a detailed process and an actual example. Throughout the paper, the value of applying standards for information exchange, such as the Functional Mock-up Interface (FMI) and the System Structure and Parameterization (SSP) standards, are highlighted.

Keywords: Model-Based Systems Engineering (MBSE), Incremental development, Round-trip engineering, Functional Mock-up Interface (FMI), System Structure and Parameterization (SSP)

1. Introduction

There are many keys for mastering next generation of product development, such as model-based engineering, modelling and simulation, digital twins, agile and incremental development, to name a few of them. These keys are becoming mature, but individually they do not provide the solution to the challenges in systems development. Therefore, the question for an organisation developing complex systems is how they shall be combined for attaining the best possible results.

For example, Model Based Systems Engineering (MBSE) has for the last decades been heralded as the logical next step in Systems Engineering as defined in INCOSE 2014 [1]. Status for the development of complex systems, in terms of MBSE adoption, appears to be that models are used extensively in the systems development process, but formal product data management is still based on the traditional document paradigm. This partial adoption of MBSE shall not be viewed as a failure. It has proved beneficial in development programs, one example being the Gripen E/F fighter aircraft development performed by Saab Aeronautics as reported in Andersson et al. [2], Herzog et al. [3] and Lind and Andersson [4]. One identified future objective is to embed configuration information in the models themselves, i.e., modelling languages and associated development environments need to implement comprehensive configuration management support.

Another example is the emergence of various simulation platforms, allowing for simulation of multiple aspects of a system. Many simulation platforms, each with its own purpose and strengths, are typically utilised within a complex development project. For the development organisation, the task is then to ensure that the correct configuration of the right models is executed timely in the right simulation

platform to obtain the desired feedback. This also implies increased need of integrated configuration management support for all model artefacts and also new approaches to information structuring, such that the sum of all relevant models provide an adequate representation of the product under development. Similar needs are outlined and discussed in Fisher et al. [5]. Also, the diversity of models and analyses implies an increasing need to formulate clear principles on how to structure the development work, to give a clear work flow when developing architectures and models and performing various analyses.

For a complex system, the requirements and final target change over time. There is a huge amount of uncertainty built into the project, characterised by, e.g., an integration plan that is never constant, and a large need to coordinate across many disciplines. The development must be flexible and agile enough to handle changing circumstances. On the other hand, the models representing parts of or the whole product need to be updated over the product life cycle, and still be relevant within each specific development step taken on the way to the final target. Hence, product information must be managed with a separation between the design and integration of the short-term (present) functions from the long-term overall (future) view of the product. The association between these two levels need to be managed by configuration management tools.

All these aspects are evident in development of complex systems, such as the truly multi-disciplinary development of a fighter aircraft. The aircraft system is developed via a large and heterogeneous development organisation, where many specialty disciplines contribute, each of them having its own specialised tools for system simulation and analysis that need to interact with the surrounding disciplines. The complexity of an aircraft platform dictates an incremental development approach. On top of that, there are authority airworthiness regulations demanding a declaration of conformity to requirements and design for each specific product configuration, implying a need for a comprehensive configuration management support.

This paper outlines principles for structuring models of a product such that both the future and present can be maintained in a set of models, over time, as the product evolves throughout the development. The paper focuses on models and their structure as the first step in describing the principles and how they can be implemented in an instantiated example.

2. Large scale incremental development

This section introduces the 4-box development model applied for the development of complex systems within Saab Aeronautics, presented in Figure 1 below and described in detail in Herzog et al. [6]. This model represents both the more fixed and long-term perspective interface to the customer, and the more flexible and short-term perspective of the integration of the current configuration. This model is used as a basis for the understanding of different time perspectives, and how the models and development activities interact with each other over the four levels.

A key element in the 4-box development model also is the clear separation between development activities producing components for integration (be it hardware or software) and the actual integration of those components. This separation is made based on the insight that it is very difficult to predict when individual artefacts will be ready for integration into a product configuration, which is especially evident in large-scale development where a large number of development teams are active concurrently.

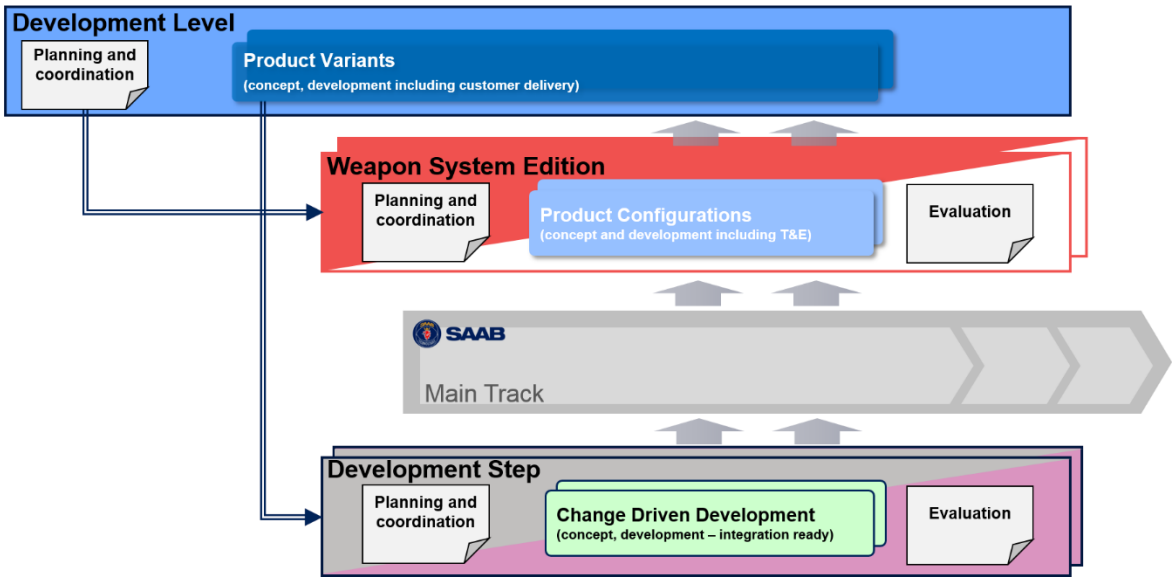

Figure 1. The 4-box development model

In the model there are four separate views, with their specific purpose and time perspective:

- **Development Level** represents a contractual obligation to the customer under which the organisation delivers weapon system configurations. This is the long-term, *future view*, where requirements analysis and architecting is performed for each product variant and the overall integration, verification and validation strategy is set. The Development Level also identifies groups of capabilities for realisation by the development teams as part of Development Steps.
- **Weapon System Edition** is used to establish a set of aircraft or weapon system configurations by integrating product changes from the Main Track in accordance with the overarching plan. Models representing a weapon system edition capture the *present state* of a product configuration. Weapon System Editions are defined for well-defined capability levels. This allows the integration organisation to seek a feasible integration sequence meeting the objectives of the Weapon System Edition.
- **Development Step** represents a defined set of capabilities in the development plan. Within the Development Step, the coarse development plan is broken down into small-scale tasks suitable for incremental development. Hence, the Development Step captures what will be realised in the *near future*. Delivery of the realised system elements to the Main Track is only allowed after successful integration testing in a complete system simulator.
- **Main Track** is the warehouse containing all versions and variants of all configuration items in the product under development available for integration.

Note that all boxes (except the Main Track) include fully instantiated technical processes, for instance in accordance with ISO 15288. The cadence and objectives of the activities are different in the boxes.

3. A framework for heterogeneous models

In the aspect of time, the levels represented in the 4-box development model in Figure 1 also represent different tenses:

- The Development Level represents the **future tense** – what shall be realised at the end.
- The Weapon System Edition is a representative for the **present tense** – what it is/becomes for each specific product configuration. This could be a realised product, or a model of it.
- The Development Step is a bridge between these tenses, **turning future to present** – moving towards the future by adding details required for realisation.

Therefore, during the development there is a need to manage different time aspects in the models:

- Model(s) how it will be in the long-term perspective – the architecture or **Definition**[1] of the future:
    - Captures the intended architecture.
    - Relatively undetailed in order to be robust to detailed variations in designs.
    - Serves as a long-term memory.
    - Input to change management/development planning.
    - For example, SysML as a common language.
- Model(s) describing detailed **Design** of, e.g., interfaces, Human-Machine Interaction (HMI) or wiring layout, capturing in detail what has or will be created now or in the near future (turning future to present). Design models may be created with multiple intents, i.e., at multiple fidelities and with interfaces described at different levels of granularity, for virtual or physical realisation or analyses. A design model:
    - Captures a system and its elements from a particular perspective.
    - Characteristic content could be overall or design discipline specific behaviour, interfaces or key properties.
    - Multiple Design models may be required to adequately represent the intent captured in a Definition model. Moreover, it is often necessary to create an adapted analysis architecture model to efficiently integrate individual design models for simulation.
    - Multiple languages and tools in multiple disciplines may be used, e.g., Simulink, Modelica, SysML, Computational Fluid Dynamics (CFD) and Fault trees.
- Executables, representing how a virtual **Realisation** (a representation of the product – in its present state at a selected structural and behavioural fidelity):
    - Multiple virtual Realisations with different fidelities and perspectives may be created.
    - Functional Mock-up Interface (FMI) and System Structure and Parameterisation (SSP) are standards preferred for definition of Realisation interfaces.
    - Realisation interconnection models – Composite models – required to adequately integrate individual children Realisation for simulation.
    - The Realisation can be virtual, e.g., a compiled model, but also a Realisation in a physical object (hardware component, software application, etc.).

Intuitively, the Definition will drive Design, which in turn drives Realisation, as illustrated in Figure 2. Both Design and Realisation will feed back knowledge to the Definition – which may prompt modifications. Due to the concurrency in development, the Definition, Design and Realisation will evolve concurrently and independently over time.

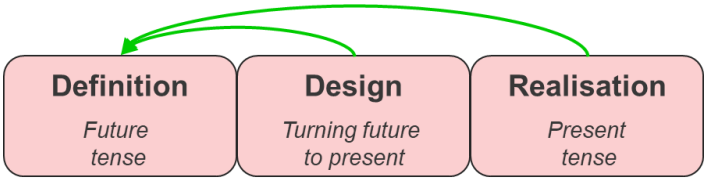

Figure 2. Tenses in models, with feedback of knowledge from the Design and Realisation to the Definition

[1] The term Architecture would be a better match against ISO 15288. However, we have found within our organisation that the term Architecture is highly ambiguous.

Hence, the overall conclusion for incremental development is that no model can represent *both* the present and the future at the same time. The tenses must be separated – the Definition and Design need to be captured in *different*, but *coordinated* models, just as the generated executables have to be coordinated with their corresponding design models.

3.1 Modelling architecture for mixing the tenses

In this section a modelling architecture framework is presented for how to manage the models on different system level for the different modelling tenses as described above.

Coupled to each system element, there are associated Definition, Design and Realisation information as illustrated in Figure 3. The Definition information will cover structure, key properties, behaviour definition and interfaces. Multiple Design and Realisation information on the top system sets detail the Definition each adding information with a dedicated purpose and at a desired level of fidelity. For efficiency in development, the objective is to iterate over this information and gradually add more detailed information. On the underlying elements, with multiple subsystems, Definition information for each subsystem is more detailed, as is also the case for its Design and Realisation information. The same pattern also applies to the (lowest) component elements.

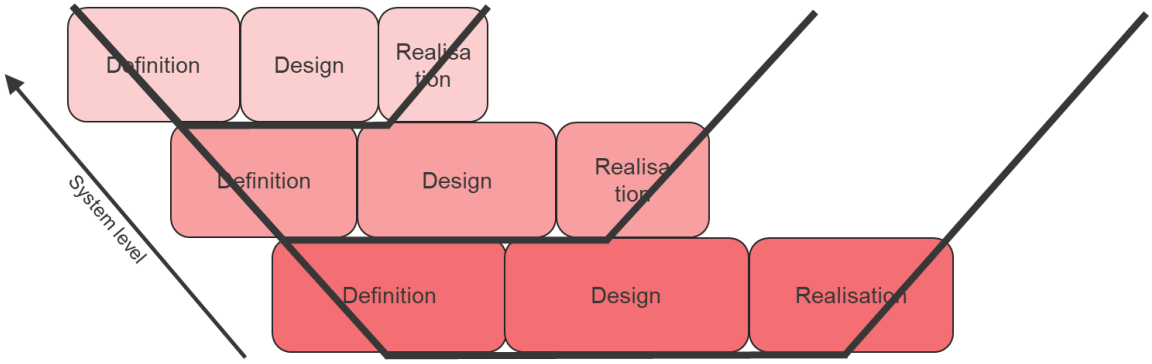

Figure 3. Modelling architecture, with the modelling tenses applied to each system level

Having the models on the top system level as an example, it can be seen from Figure 4 that we first have a leftmost Realisation, forming a virtual, coarse Realisation. When development progresses, Realisations on the underlying level capturing different aspects and disciplines will emerge. A combination of them then forms a virtual or physical Realisation on the parent system with intermediate details. This Realisation could, e.g., be a simulator representation of the overall system properties. The same pattern appears to the next underlying level as well, together forming detailed Realisations on the upper levels. These multiple virtual or physical Realisations are on different levels of detail and fidelity, and with different credibility. To summarise, there are multiple virtual or physical Realisations for a single Definition model version.

The principle of feedback is also of high importance. The feedback is iterative, and applied on each level, as illustrated in Figure 4, where the knowledge from the virtual and physical Realisations are used to improve and adjust both the Definition and the Design models. Since virtual Realisations can be created with relative ease, this feedback will prompt the generation of new versions of Definition and Design models. When automated, we label this feedback pattern Round-trip engineering as there will be multiple iterations where the Definition and Design models are refined based on the knowledge gained from virtual or physical realisations.

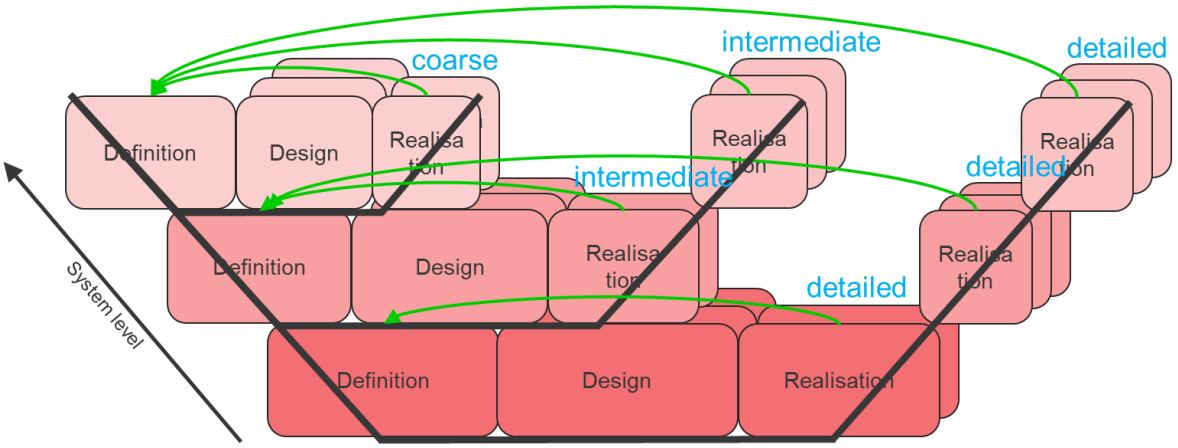

Figure 4. Modelling architecture, with multiple virtual and physical Realisations, covering multiple perspectives, with various fidelity and credibility. The Realisations capture different levels of details, and will feed back information to adjust Definition and Design models.

Configuration management becomes vital in the situation described above. The developing organisation needs to adequately keep track of the effective Definition information and the corresponding Design and Realisation items. This means the actual versions of individual items, as well as the effective traceability links between the items.

Figure 5 captures some potential evolutions of Definition model, Design models and Realisations. In the figure, the circles with a black outline that are labelled with a C represent the set of models that are valid and consistent at a specific time. Note that for the physical Realisation there are two valid representations indicating that two separate items have been produced.

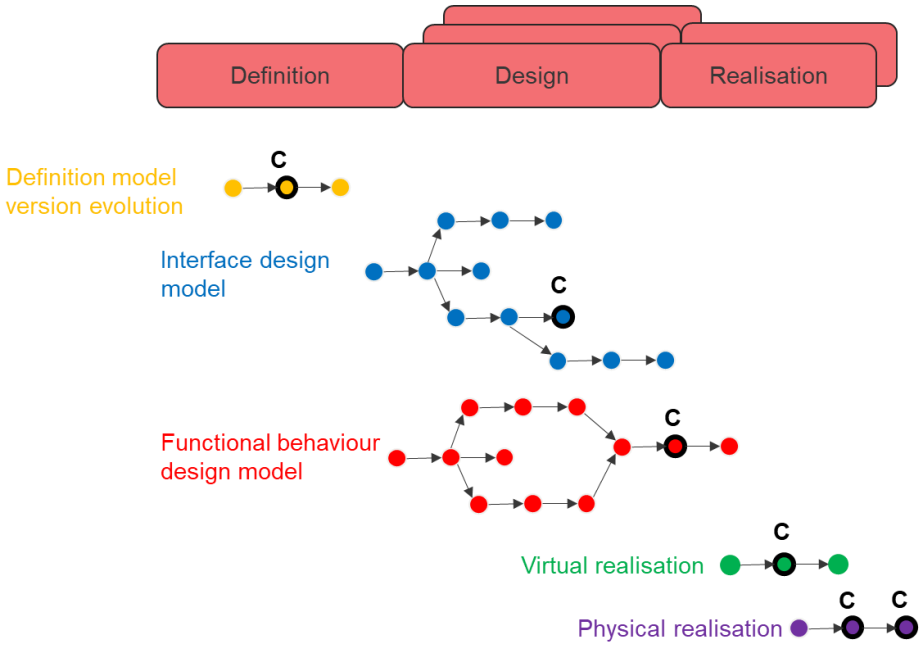

Figure 5. Model evolutions over time

4. Modelling languages and data exchange mechanisms

Saab Aeronautics has adopted an approach where the best-suited domain specific modelling language and tool is used for each development task. Such an approach benefits greatly from well-established and standardised lines of communication between the different engineering disciplines.

In this section, the framework presented in Section 3 is exemplified by means of deploying four different standards that together provide the basis to ensure efficient and sustainable model integration across all life cycle phases:

- Modelica [7]
- System Modeling Language, SysML [8]
- Functional Mock-up Interface, FMI [9]
- System Structure and Parameterization, SSP [10]

Generally, the two standardised modelling languages Modelica and SysML, with multiple tool sets adhering to each respective standard, ensure that developed models can be migrated at a low cost. Thus, avoiding tool vendor lock-in effects and enabling developers to freely exploit the benefits of specific Modelica and SysML tools. Furthermore, the FMI and SSP standards jointly address challenges associated with model integration and configuration management. Even though a significant portion of the physics-based aircraft vehicle system models used at Saab Aeronautics are expressed in the Modelica language, many hardware models and software are developed in non-Modelica compliant tools. In addition, architectural models are typically expressed in SysML. This introduces a significant challenge, as information needs to be exchanged between these modelling disciplines throughout development. This challenge is the primary focus of the FMI and SSP standards.

The exploited standards are here introduced at a level of detailed required to follow the Round-trip engineering workflow presented in Section 4.1:

- **Modelica** is a multi-domain, object-oriented, and equation-based modelling language. These characteristics make the Modelica language particularly suitable for modelling physics-based systems; however, the language does by no means exclusively target this discipline. Schamai et al. [11], for example, demonstrate how requirements can be formalized using Modelica within the field of requirements engineering. The Modelica language is standardised by the Modelica Association. Modelica models, that are fully compliant to the standard, can as a result be exchanged between different Modelica Modeling & Simulation (M&S) tools.
- **SysML** is just like Modelica a standardised modelling language, see for example Friedenthal et al. for a detailed description of the language and its use [12]. In contrast to Modelica, SysML targets the development of general purpose architectural models. In this context, SysML is chosen to model the relationships between executable entities, while providing a connection between the modelled physical system and its requirements.
- The **FMI** standard targets the exchange of domain specific executable models, so called Functional Mock-up Units (FMUs). In short, the FMI provides a standardised API that an exported model, source code or executable binary should support. In addition, FMI provides a standardized XML format for model interface descriptions. At the time of writing, more than 150 tools formally support the various flavours of the FMI standard [9], [13].
- The **SSP** standard aims to complement the FMI standard in terms of allowing for standardised specification of analysis architectures and variant handling of parameterised models. It provides a format for exchanging parameterised and executable analysis architectures, also known as simulators [14]. A parameterised simulator exported according to the SSP specification is packaged in a zip file format denoted as *.ssp. The zip file, once populated with all the required artefacts, contains everything that is needed to execute the simulator. In other words, the *.ssp includes both the structure in terms of an analysis architecture, as well as artefacts providing the mechanisms for variant handling and configuration management. As schematically visualised in Figure 6, an *.ssp file contains the analysis architecture description in the System Structure Description (SSD) format along with a resources folder. The resources folder can contain other *.ssp files as well as constituent executable models in, for example, the FMI format, i.e., executable analysis models exported from their original development tool. In addition, any System Structure Values (SSV) files, incorporating parameter values, are placed in the resources folder. These SSV files contain information rendered by non-executable analysis models. The parameter values in SSV files are bound to

the executable model parameters via included System Structure Mappings (SSM) files.

Figure 6. Schematic overview of a packaged simulator exported according to the SSP standard

4.1 Round-trip engineering for complex systems

This section introduces a Round-trip engineering pattern for complex systems. Round-trip engineering is a development method in which the activities are conducted concurrently and in synchronisation. Tool support enabling engineers to move freely between disciplines are a pre-requisite to Round-trip engineering [15], [16]. The workflow presented in Figure 7 is a detailed realisation of the evolution presented in Figure 2. This detailed representation is the result of a prolonged research collaboration on the topic conducted in the two successive ITEA initiatives: the *OpenCPS* [17] and *EMBrACE* [18] projects. The workflow visualises the evolution of primary artefacts, and their relationship to one and another, that are seen as relevant when exploiting M&S during aircraft system and subsystem development. Furthermore, Figure 7 aims to highlight the feedback necessary for evolving all the relevant artefacts. The exploited standards enable model based information exchange and, as a result, the flexibility required in Round-trip engineering. The considered artefacts are:

- Requirements
- System architecture
- Analysis architecture
- Analysis model(s)
- Composite system simulation

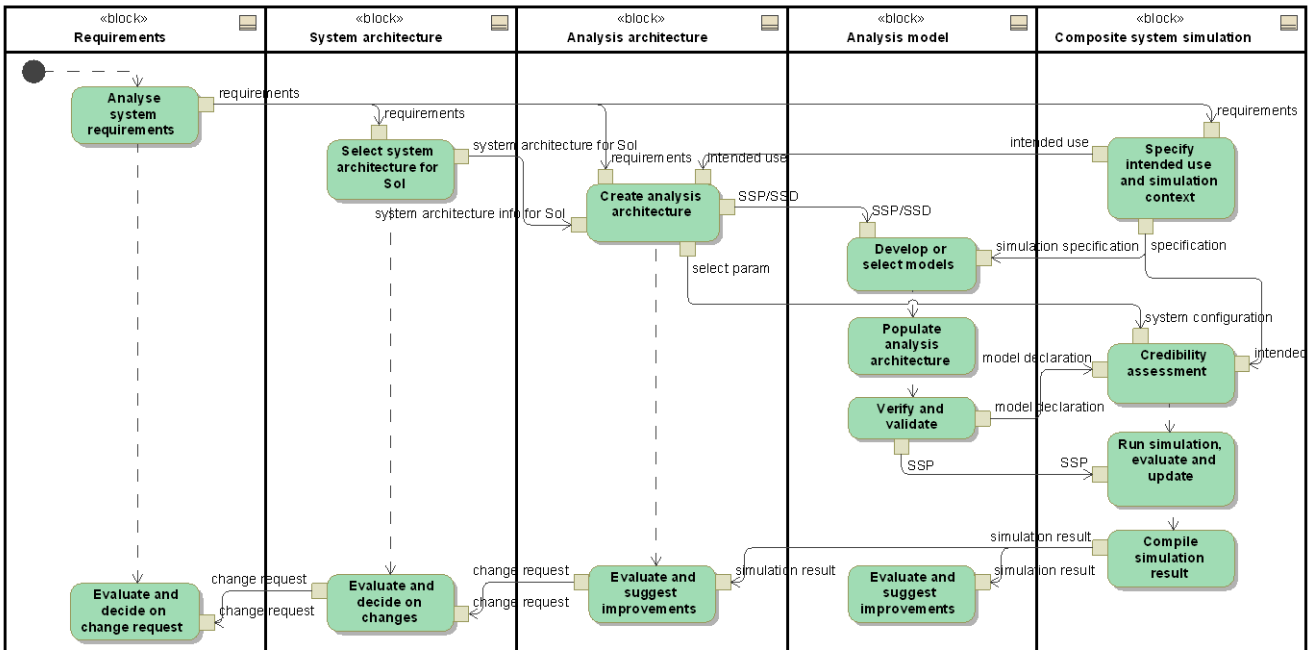

Figure 7. Round-trip engineering workflow. The swimlanes of the workflow SysML representation does not encompass all the constituent activites. A minimal subset of the activities are presented in order to convey the overall flow of information.

These artefacts relate to the Definition, Design, and Realisation models of Figure 2 as follows:

- The System architecture artefact maps to the Definition model as it captures what shall eventually be realised.
- Analysis models maps to Design models as they represent a current (virtual) state of product development. Analysis models can, for example, be parametrised executable models representing several potential realisations.
- For simulation purposes, it may be necessary to adapt the general system architecture model to adequately meet the identified intended use(s). Such an Analysis architecture is viewed as a special class of Design models.
- Realisations are executable models, either generated from a single Design model or a composite system simulation and they represent what is currently realised (virtually). In other words, an Analysis model becomes a Realisation model once it is compiled and coupled to the parameter set of a specific realisation.

All models described above will evolve independently of each other, i.e., there is no defined starting point in the workflow, but the end results must be kept under stringent configuration control.

4.2 Process descriptions

This section details the processes for developing and managing the artefacts identified in the previous section and in Figure 7.

4.2.1 Define requirements

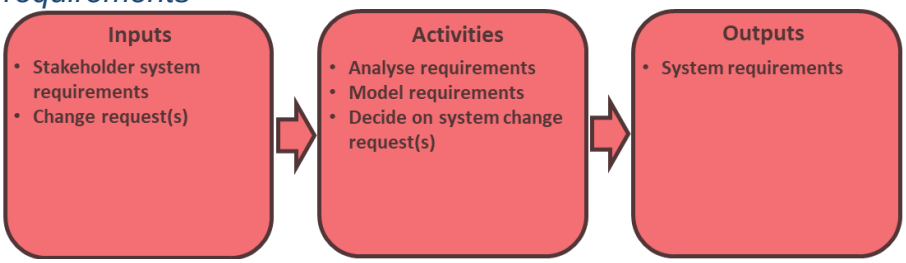

Figure 8. Requirements swimlane as a process in an Input-Process-Output diagram

The purpose of the activities within the *Define requirements* process in Figure 8 is to identify the requirements to be investigated, analysed, and verified related to the whole or part of the System of Interest. The activity to *Analyse system requirements,* initiated from various roles working within the process, aims to identify and select requirements relevant for the analysis based on the input *Stakeholder system requirements*.

Furthermore, the process includes an activity where the identified requirements are modelled. One way to *Model requirements* is to employ assumption and guarantees grouped in contracts [19], [20]. Such contracts enable a close integration between the requirements and the *System* and *Analysis architectures* if all artefacts are expressed in the same standardised format, for example SysML. Contracts, as schematically visualised in Figure 9, include the assumptions and guarantees of the engaged stakeholders. Such modelled contracts can be coupled to both the *System* and *Analysis architectures*.

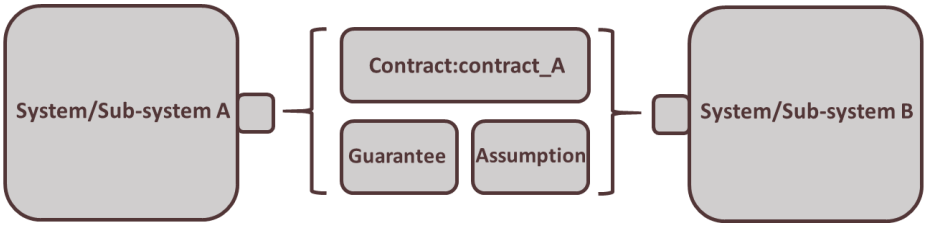

Figure 9. Schematic representation of a contract, including an assumption and a guarantee, between two stakeholders; Stakeholder A is engaged in activities related to System/Subsystem A whereas Stakeholder B is engaged in activities related to System/Subsystem B.

The *Modelling requirements* activity of Figure 8 should be applied to a degree deemed as useful and feasible. In some cases, expressing the *Stakeholder system requirements* in an executable format may be beneficial. When applied, executable requirements enable continuous verification, or monitoring of contracts, using M&S applications such as simulators. An implementation of modelling formal executable requirements in the Modelica language is provided by Otter et al. [21]. A more recent effort on the topic of modelling of executable requirements is the Common Requirements Modeling Language (CRML) currently under development within the frame of the EMBrACE project [18].

The last activity presented in Figure 8 is denoted *Evaluate and decide on changes.* This activity addresses any input *Change requests* related to the requirements of the system of interest.

4.2.2 Define System architecture

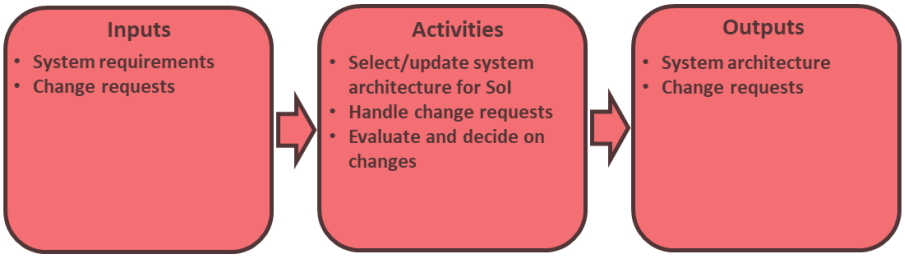

Figure 10. System architecture swimlane as a process in an Input-Process-Output diagram

The System architecture swimlane includes the process activities shown in Figure 10. One or more candidate architectures are created aiming at meeting identified *System requirements.* The *System architecture* is created/modified in *Select/updated system architecture for system of interest* activity based on architecting best practice and feedback from analysis and simulation activities. The *System architectures* can be expressed in multiple formats, for example, using SysML or in the xml formats of the SSP standard. Such formats facilitate a model-based communication, and consistency checks, with the subsequent *Create analysis architecture* activity of the *Define analysis architecture* process, see Figure 11. The *Evaluate and decide on changes activity* provides a mechanism for feedback just as in the *Define requirements* process.

4.2.3 Define analysis architecture

Figure 11. Analysis architecture swimlane as a process in an Input-Process-Output diagram

The *Define analysis architecture* process activities, see Figure 11, can be seen as the link between the *System architecture* view and the analysis view of the system of interest. This process provides an optimisation of the *System architecture* to match the needs of individual simulations. Therefore, it has to take into consideration the *System requirements*, the *Intended use* for each simulation, and the *System architecture*. Depending on where in the process the work starts, the *Define analysis architecture* process output artefacts will influence, or be influenced by, all of the process input artefacts.

The objects in the *System architecture* and the corresponding objects in the *Analysis architecture* can be either directly mapped, or lumped, to one and another. For example, a *System architecture* identifies the subsystem, but does not specify the number of *Analysis architecture* constituent

executables. For this reason, the *Check consistency* activity is essential to ensure traceability. However, performing such a consistency check can be a significant challenge, especially for large scale systems. One desirable approach, simplifying the activity, is to ensure that both architectures are expressed in the same standardised formats; such that objects in the two architectures can be directly mapped to each other.

In addition, the *Analysis architecture* provides information concerning the architecture constituent entities. Interface descriptions, evolving alongside the *Analysis architecture*, can be deduced for all the included subsystems. If the interface descriptions are expressed in the format provided by the FMI standard, then they can directly serve as templates when developing the corresponding analysis models in an M&S development tool; for example, the Modelica tool OpenModelica [22]. *Analysis architectures* are advantageously expressed using both the FMI and SSP format. Adding the SSD, SSV, and SSM formats of the SSP standard enable standardised descriptions of the flow of information between models from different engineering disciplines. Note that the *Analysis architecture*, independent of the level of abstraction, merely defines the pattern for interfacing models. Any needed executable simulation models are realised through the *Implement Analysis model(s)* process, see Section 4.2.4 below.

Furthermore, in close analogy with the *Define requirements* and *Define system architecture* processes, there is an activity providing a feedback mechanism within the *Define analysis architecture* process. This activity is denoted *Evaluate and suggest improvements.* However, *Evaluate and suggest improvements* is slightly different compared to the two corresponding activities in the *Requirements* and *System architecture* processes as it operates on input *Simulation results* compared to *input Change requests.* Simulation results may, however, raise *Change requests*. Changes in an *Analysis model* interface description may, for example, be prompted from the input *Simulation results* once these results are compared to the *Intended use*.

4.2.4 Implement Analysis model(s)

Figure 12. Analysis model swimlane as a process in an Input-Process-Output diagram

The *Develop model* activity is one of the key activities in the *Implement Analysis model(s)* process shown in Figure 12. The development of analysis models is based on the input model or simulator *Interface description* and the *Intended use*. These are analysed to form detailed model requirements. For tools supporting FMI or SSP *Analysis model* interfaces are generated based on the *Analysis architecture*, if such support is lacking the interface definition must be created manually. The model *Intended use(s)* drives the desired model fidelity and it thus serves as foundation for model development.

The *Implement analysis model* process activities may result in additional identified support *Analysis models* as a result of ownership, legacy, identified reuse potential, and separation of concerns. Any additional *Analysis models* are developed iterating the process described in Figure 12.

Moreover, the *Analysis model* process also includes the activity *Design parts for simulation.* The *Design parts for simulation* activity includes any development of support models providing the parameter values necessary to evolve an *Analysis model* from Design to Realisation model. This activity should be seen as its own instance of the *Implement analysis model(s)* process; however, it is shown as an activity in Figure 12 as its resulting artefacts typically provides information on which the executable *Analysis models* and simulators rely. Geometry models are examples of non-

executable models providing essential input to the corresponding executable models. The geometry of the System of Interest is modelled by the geometrical domain experts in their preferred tools. The geometry models are, just as *Analysis models* in general, founded on the input requirements of the system of interest and corresponding model *Intended use* which are analysed and translated to requirements on the geometry model. Once the model has been concretised, all the data relevant for analysing the system of interest is suggested to be converted to the standard SSP format. The SSV concept of the SSP standard is exploited to store the parameter values specified in the geometry modelling domain. Storing *Design parts for simulation* artefacts in the SSV format facilitate a number of beneficial features. For example, these supplemental artefacts can be coupled, with ease, to the dependent *Analysis models*. Furthermore, consistency checks in terms of, for example, regression testing against previous artefact versions can be automated and used to trigger communication between the different actors working within the Round-trip engineering workflow described in this paper.

The developed *Analysis model* needs to be verified before being used in simulation. The verification includes checking the model functions and interface against the specification. Assessing the representativeness of an executable model, with respect to its intended use, can be done as soon as an executable version of the developed model is available. The methods for assessing model validity depend on the life cycle stage of the system of interest. In early life cycle stages, sensitivity analysis and uncertainty quantification are applicable methods [23]. In later life cycle stages, these methods are complemented with comparisons to in-situ measurements in activities often referred to as predictive validation [14]. The representativeness of the emergent behaviour, on a system level, is assessed in the *Verify and Validate* activity of Figure 11. Eek et al. [24] propose packaging analysis model V&V and Uncertainty Quantification results in a meta-model, along with its corresponding *Analysis model*, such that the *Analysis model* carries information concerning its own credibility. Such an approach enables a solution where the assessment results can be provided to the end-user in a model-based format. The model is ready for export, for use outside its development environment, once enough evidence has been gathered indicating that the model fulfils the specification and its intended use. For tool independent exchange the FMI and SSP standards can be exploited.

The feedback mechanisms is, just as in the *Implement Analysis model(s)* process, represented by the *Evaluate and suggest improvements* activity of Figure 12.

4.2.5 Execute composite system simulation

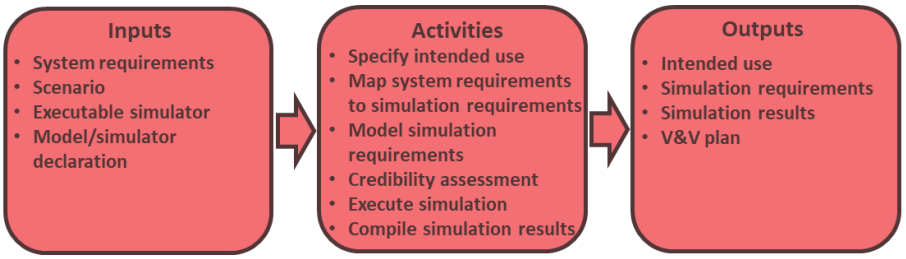

Figure 13. Composite system simulation swimlane as a process in an Input-Process-Output diagram

The *Execute Composite system simulations* process, see Figure 13, includes the activity *Specify intended use* in which the purpose of the executable model or simulator is expressed and related to the requirements of the system of interest. Once the expectations and purpose of the simulation is stated, be it for system validation, system verification or exploration, it is possible to formulate the *Simulation requirements*. Such requirements can be both functional and non-functional; for example, the model should predict the cooling performance with a maximum relative error of 5%, and the model should be provided to the end-user in the FMI format.

Subsequently, in the *Map system requirements to simulation requirements* activity, a mapping is established between the input *System requirements* and the generated *Simulation requirements*. This mapping aids to ensure relevance and traceability. Furthermore, just as in the *Define requirements*

process, the derived requirements should be modelled in a suitable language if deemed beneficial. The modelling of model requirements is conducted in the *Model simulation requirements* activity.

The *Model/Simulator declaration* process input serves as foundation in the *Credibility assessment* activity. This activity connects all evidence of representativeness, gathered in the *Implement analysis model process,* to the intended use of the Realisation exploited in the *Execute composite system simulation process*.

The fifth bullet listed in Figure 13 is the *Execute simulation* activity. This activity includes the steps of:

- Conducting a consistency check on the input *Analysis architecture* (provided in the *Executable simulator* process input) against previously exploited architectures (if any).
- Importing the input executable *Analysis model* or simulator in the simulation tool driving the simulation.
- Compiling, aggregating and analysing the credibility of the simulation.
- Executing simulation in accordance with the *Scenario* to be investigated.
- Compiling and communicating *Simulation results*.

Any required updates to the constituent *Analysis models* or *Analysis architecture* are communicated to all stakeholders, preferably via the FMI and SSP formats, so that all impacted artefacts can be updated as efficiently as possible with minimal risk of misinterpretations. If successful, the outcome of any executed simulation is communicated via the process output *Simulation results.*

5. Instantiated example

The formulation and development of an aircraft Environmental Control System (ECS) here serves as an example intended to demonstrate the workflow presented in Figure 7. This example is an expansion of the application example described by Hällqvist et al. [25]. The system of interest is an ECS, coupled to a coolant distributions system providing coolant to a consumer requiring cooling power. The end goal of this example is to facilitate a platform, a simulator, for model-based evaluation and comparison of two different Realisations of the coolant distribution system. This simulator needs to evolve as more information becomes available. Both Realisations will be refined during this evolution until a well-founded decision, on which Realisation to proceed with, can be made.

5.1 Use-case

The Round-trip-engineering workflow is exemplified via a use-case representing a realistic scenario during aircraft vehicle system development. The purpose of the use-case is to compare and evaluate two different physically feasible installations of an aircraft cooling system. The two alternatives require different routing solutions of the piping system transporting the coolant from the cooling system to the consumer. The evaluation is to account for transient operation of the aircraft where the inertia of the system of interest can be exploited to fulfill the subsystem requirements.

The actors involved in the use-case are engineers developing aircraft tactical subsystems, vehicle systems hardware and vehicle systems software.

5.1.1 Pre-requisites

A set of requirements for the system of interest is the primary pre-requisite to the use-case. The system of interest is the cooling system providing cooling to a radar deployed in a fighter aircraft. In summary, a radar with a specified cooling need is to be installed in a fighter aircraft. Identification of possible installation space enables two different variants of the cooling system. Which of the two options to proceed with is not obvious as emergent dynamic behavior needs to be considered. A central storage containing a large set of parameterised executable legacy models (Design models, in the terminology introduced in Section 3), including a modelled Environmental Control System, a modelled coolant power distribution system, and a modelled ECS controlling software is available for use, see Figure 14 and Figure 15. The coolant distribution system model is parameterized such that it can represent any feasible routing option. The controlling software ensures that the consumer of

coolant power can operate at temperatures specified as desirable. The controlling software set-point temperature is specified via the parameter denoted *Constant2* in Figure 15.

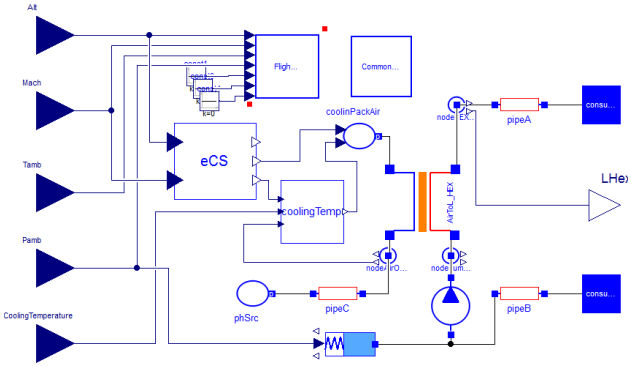

Figure 14. Legacy Modelica model of Environmental Control System (ECS) coupled to a liquid coolant distribution system

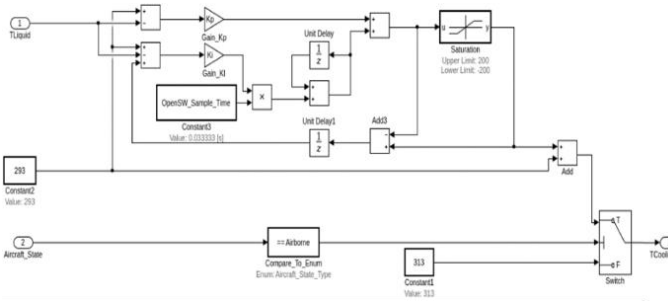

Figure 15. Legacy Simulink model of ECS controlling software

5.1.2 Main Scenario and Expected Outcome

1) Triggered by the purpose of the example, see Section 5, geometry models are developed of the two different routing options. Once these routing options are available, the geometry information needed to instantiate two executable Realisations is available. These two executables are realised by means of incorporating SSV files containing the geometry parameter values.
2) In parallel, available system requirements are identified and transformed to an intended-use in the *Specify intended use* activity of the *Execute composite system simulation* process. Included are identification of test cases and preparation of necessary stimulus.
3) The M&S engineer responsible for the coolant power distribution Analysis model imports the geometry information in the *Design parts for simulation* activity. The incorporation of geometry information evolves the Analysis model from a Design model to a Realisation. The two modeled solutions are validated and exported to later be integrated in the Analysis model representing the complete system.
4) An architect receives the legacy models and establishes an Analysis architecture that fulfills the composite system simulation intended use, as well as is consistent with the system requirements, and architecture. This Analysis architecture is populated with the required executable models; it is then exported for integration in a simulation tool.
5) An M&S engineer integrates the complete Analysis architecture, including executable Analysis models and their parameter values, in the best suited simulation tool. The actor then executes the simulation(s) and compiles the simulation results.
6) The simulation results are fed back for evaluation purposes in all the processes described in Section 4.2.

5.2 Results

The artefacts realising the use-case are developed and evolved by means of exploiting the processes of Figure 7. These artefacts, and their relationships, are described in this section.

The Analysis architecture of the application example is visualised in different views, with differently detailed interfaces, in Figure 17 and Figure 21. The corresponding System architecture is visualised in Figure 16 and Figure 20. In order to use any simulation results, it is necessary to understand how well the Realisation, used in the simulation, represents the actual system described in the system architecture. A consistency check between these two views can then be performed and possible differences can be identified and managed. This consistency check between the two architectures can be more or less trivial; the degree of difficulty depends on the status of the System architecture with respect to the intended use and with respect to the Analysis architecture,. In this example some of the Analysis models represents multiple systems found in a traditional airborne vehicle system structure, for example parts of the engine system is represented in the ECS_HW Analysis model. The

difference, with respect to the systems and elements within the System and Analysis architectures can be seen in Figure 16 and Figure 17.

A corresponding architecture with detailed interfaces, in a later stage of evolution, can be seen in Figure 20 and Figure 21. Knowledge about the models and what they represents are vital to assess the credibility of the complete simulator. In other words, the mapping between the presented architectures needs to be tracked and verified if to maximise the value of MBSE.

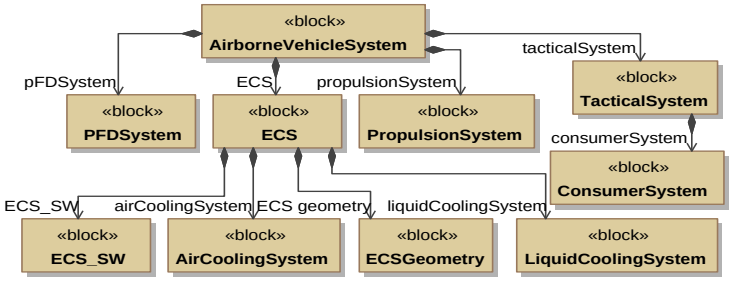

Figure 16. System architecture for parts of an airborne vehicle system

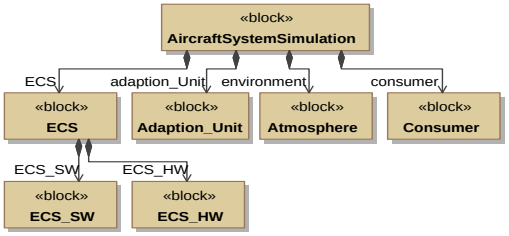

Figure 17. Initial analysis architecture based on available models

The initial Analysis architecture is created based on a set of legacy Analysis models from the systems simulation and software engineering domains, see Figure 14 and Figure 15 respectively. These legacy models are first combined, together with a model of a consumer of cooling power, to represent a closed loop radar cooling system. This expansion is a consequence of an interpretation of the, at this point in the workflow, informal and somewhat vague available intended use of the needed composite system simulation artefact. The intended use is at this point merely a qualitative statement that a simulator for concept evaluation during dynamic operation, accounting for transient emergent behaviour, is needed.

At this stage, the consumer model is merely a representation of a possible system, and this representation will later be populated with an executable model representing the actual components. Functionality to export interface descriptions from such a draft representation, in the standardised FMI format, was implemented in the *Papyrus* open-source UML/SysML modelling tool [26] within the frame of the OpenCPS project.

Once an initial version of the consumer Analysis model has been integrated into the overall Analysis model of the example, then the resulting Design model needs to be converted into a Realisation via two different sets of parameters. These two parameter sets are generated through the *Design parts for simulation* activity.

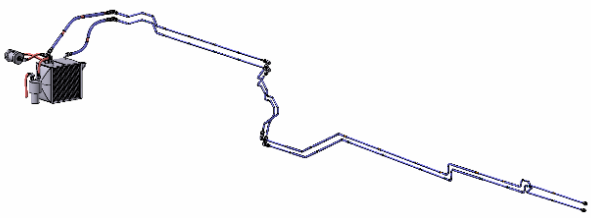

Figure 18. Application example geometry for configuration one

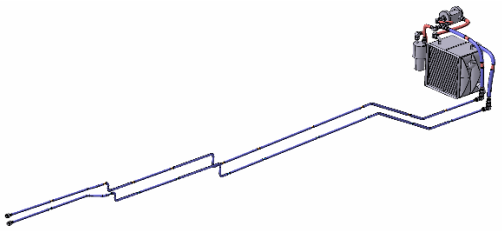

Figure 19. Application example geometry for configuration two

In this step of evolution, Analysis models from the geometry modelling domain are coupled to the application example via the SSV and SSM concepts of the SSP standard. The application example geometry configurations are shown in Figure 18 and Figure 19; extracts of the corresponding SSV files are listed in Listing 1 and Listing 2. The difference between the two configurations is made apparent if comparing the two SSV files one row at a time.

Listing 1. SSV file corresponding to application geometry for configuration one

```
<ssv:ParameterSet>
  <ssv:Parameters>
    <ssv:Parameter
name="part_'ret'.paramSet_'outParam'.param_'ZParamRL'">
      <ssv:Real unit="Unit Not Applicable" value="2.417386"/>
    </ssv:Parameter>
    <ssv:Parameter
name="part_'ret'.paramSet_'outParam'.param_'pipeLengthTotRL'">
      <ssv:Real unit="[m]" value="7.41248272578546"/>
    </ssv:Parameter>
  </ssv:Parameters>
</ssv:ParameterSet>
```

Listing 2. SSV file corresponding to application geometry for configuration two

```
<ssv:ParameterSet>
    <ssv:Parameters>
      <ssv:Parameter
name="part_'ret'.paramSet_'outParam'.param_'ZParamRL'">
        <ssv:Real unit="Unit Not Applicable" value="0.88024"/>
      </ssv:Parameter>
      <ssv:Parameter
name="part_'ret'.paramSet_'outParam'.param_'pipeLengthTotRL'">
        <ssv:Real unit="[m]" value="4.57083438288096"/>
      </ssv:Parameter>
    </ssv:Parameters>
</ssv:ParameterSet>
```

At this point, all the SSP artefacts that renders a complete Realisation, that can serve the *Execute composite system simulation* process, is available. This Realisation, in its current evolutionary state, is imported in a SSP supporting SysML tool such that the Analysis architecture can be refined and mapped to a corresponding System architecture.

A SysML block diagram of the System architecture is shown in Figure 20, the corresponding Analysis architecture is shown in Figure 21. A significant difference in the level of detail of the interfaces is made apparent if comparing these two views. Both architectures contain interfaces described at the level of detailed that is necessary for the purpose of the respective representations. This difference is fully natural; however, it needs to be tracked and checked for consistency just as in the earlier evolutionary state presented in Figure 16 and Figure 17.

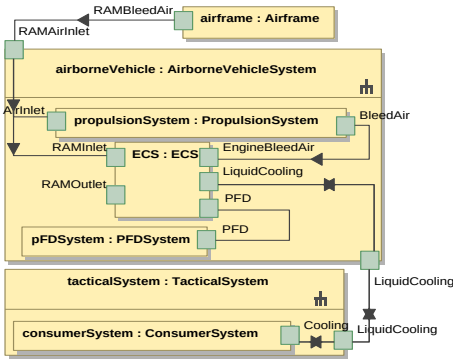

Figure 20. System architecture of the System of Interest developed through Round-trip-engineering

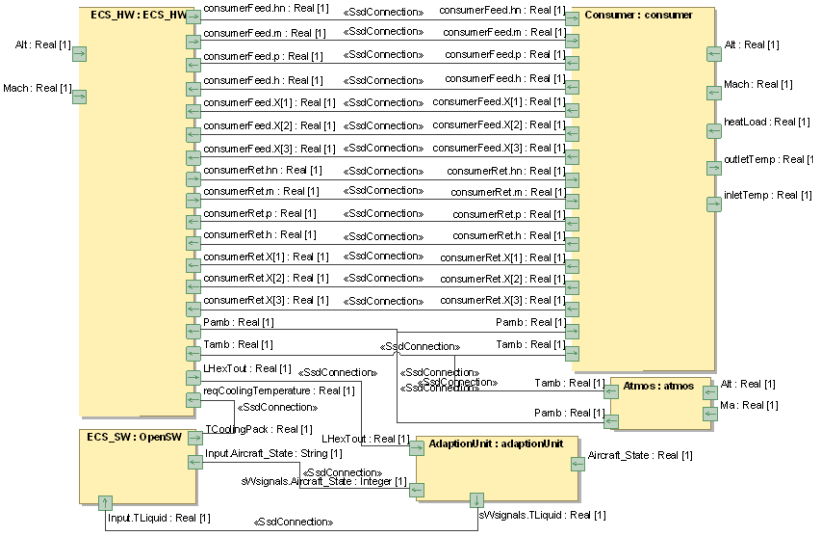

Figure 21. Analysis architecture of example simulator developed through Round-trip engineering

Application example results from the *Execute Composite system simulations* process are provide in Figure 22 and Figure 23. The application example contract selected for monitoring is presented in SysML format in Figure 24. The simulation boundary conditions, for a specific analysed flown mission, are provided in Figure 22 and Figure 23. The two different Realisations, one for each configuration, render different increases in liquid temperature across the consumer. Simulations of configuration one, the longer routing option visualised in Figure 18, results in higher temperature increases compared to that of configuration two. However, for this particular mission, the contract under investigation, see Figure 24, is fulfilled for both investigated configurations. No decision on preferred configuration can therefore be made based on these simulation results alone and the analysis needs to continue for other relevant missions.

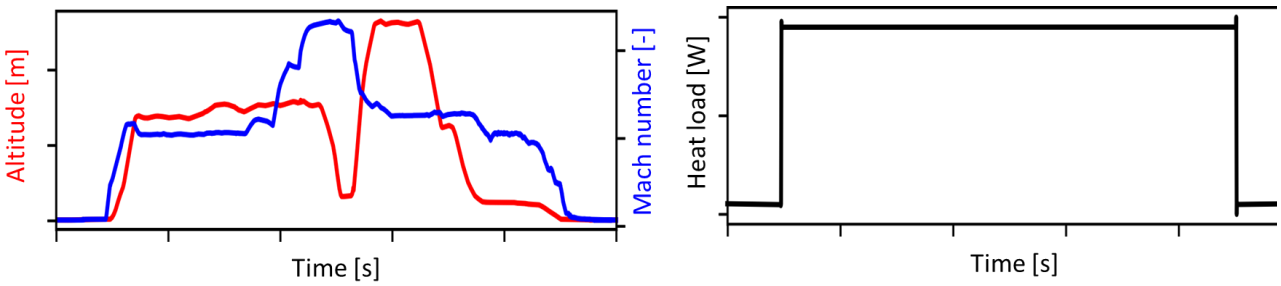

Figure 22. Application example composite simulation boundary conditions. The boundary conditions are generated from in-situ measurements conducted in an unspecified aircraft

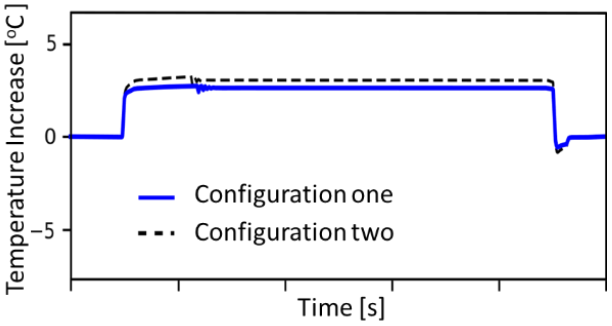

Figure 23. Liquid temperature increase for the consumer for the two different coolant distribution routing options

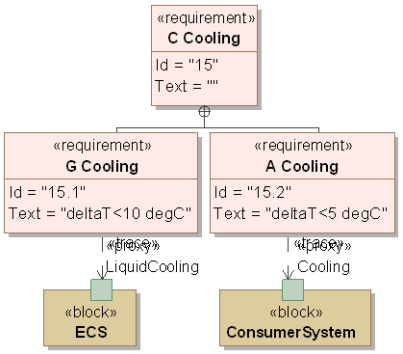

Figure 24. Application example contract including a “Guaranteed” and an “Assumed” liquid temperature increase

6. Discussion

In this paper, a new structuring principle is proposed for models created to support the incremental development of complex systems, in accordance with the Saab Aeronautics 4-box development model. The key principle promoted is that model elements with different perspectives (tenses) must not be managed in the same model. Model elements capturing a desired future state (Definition model) must not be mixed with the model elements capturing what is presently available for virtual integration (Realisation), and the development activities where the Realisation emerges (Design model).

The Round-trip engineering framework, described from both a process perspective and with an instantiated example in this paper, illustrates the “Definition-Design-Realisation” tenses introduced in Figure 2. The example clearly show how the strengths of different languages are utilised to form an integrated workflow. Moreover, it also highlights the value and importance on applying standards like FMI and SSP for automating the information flow across the tenses. The framework could be instantiated, to the degree necessary to any system level.

The principles proposed herein has the following advantages:

- It acknowledges that there is not a question of one model or one language to succeed in engineering supported by models. As an example, it is not a question of creating a single detailed, all encompassing, digital twin, but rather multiple twins with identified purposes, fidelities and credibility.
- It allows for any number of views using any number of languages that can be developed independently of each other.

When adopting the principles proposed, an organisation will have to place attention on realising a model configuration management capability where individual models (the correct version of each model) can be related to each other. Such a configuration management capability need to focus on

supporting and relating models captured at different abstraction levels. For a complex product there might exist a number of coarse Design models and Realisations, as well as an equally large number of medium granularity models built from subsystem models. Moreover, there will also exist a number of fine grain Design models and Realisations built from composed subsystem models.

It is not a trivial task for a development organisation to correctly manage and control the evolution of such a large modelling flora. However, the alternative, to consciously reduce the number of models used to represent a system under development will increase project risks and negate the advantages of introducing modelling and simulation.

In addition to separating the tenses of models, we see the following key enablers for the implementation of an integrated modelling and simulation capability:

- Integration of Realisations from different sources, by ensuring a common interface standard (e.g., FMI and SSP).
- Emerging standards, like SSP, facilitates management of heterogeneous yet related models, and are essential tools for facilitating the adoption of a truly model based approach to development.
- Single classification framework for coordinated integration of Realisations.
- Definition of a model credibility measure, to clarify to what extent simulation results based on the integrated models can be trusted to adequately represent an actual system. One such example, inspired by the NASA Credibility Assessment Scale [27], is the model credibility assessment approach proposed in Eek [24].

7. Conclusions and future work

For efficient development of next generation safety critical systems, organisations need to find approaches that combine the advantages of incremental development in terms of flexibility with the rigour provided by model based ways of working. The approach proposed in this paper – to partition models based on their respective tense – allowing the development organisation to develop models in parallel and combine them to adequately represent a relevant system configuration. Thus allowing for simulation services that can serve multiple purposes and be updated at a small cost.

Within Saab Aeronautics, parts of the outlined framework has been in operation for a long time and validated extensively. The contribution in this paper is the structure for integrating contributions from multiple sources. The principles presented have been evaluated on a small scale using paper examples and practical work within research projects. Results are encouraging and the presented solutions will be applied in next generation development projects. There is also a need for additional work in developing and promoting standard-based interoperability mechanisms and for further standardising frameworks for Analysis model classification and credibility assessment.

8. Contact Author Email Address

mailto: Robert.hallqvist@saabgroup.com, Erik.Herzog@saabgroup.com

9. Copyright Statement

References

[1] INCOSE, “A World in Motion - Systems Engineering Vision 2025,” 2014.

[2] H. Andersson, E. Herzog, G. Johansson and O. Johansson, “Experience from introducing Unified Modeling Language/System Modeling Language at Saab Aerosystems,” *Systems Engineering 13,* 2010.

[3] E. Herzog, J. Hallonquist and J. Naeser, “Systems Modeling with SysML - an experience report,” in *INCOSE International Symposium*, Rome, Italy, 2012.

[4] I. Lind and H. Andersson, “Model Based Systems Engineering for Aircraft Systems - How does Modelica Based Tools Fit?,” in *Modelica Conference*, Dresden, Germany, 2011.

[5] A. Fisher, M. Nolan, S. Friedenthal, M. Sampson, M. Bajaj, L. VanZandt, K. Hovey, J. Palmer and L. Hart, “Model Lifecycle Managment for MBSE,” in *Proceedings of the INCOSE International Symposium, 24*, 2014.

[6] E. Herzog, A. Forsgren Goman, O. Sundin and Å. Nordling Larsson, “A 4-Box Development Model for Complex Systems Engineering,” in *Submitted to the 32nd INCOSE International Symposium*, 2022.

[7] P. Fritzson, Principles of Object Oriented Modeling and Simualation with Modelica 2.1, Wiley-IEEE Press, 2004.

[8] OMG, “OMG Systems Modeling Language, Version 1.6,” OMG Systems Modeling Language, 2019.

[9] Modelica Association Project "FMI", “Functional Mock-up Interface for Model Exchange and Co-Simulation, version 2.0.3,” Modelica Association, 2021.

[10] Modelica Association Project "SSP", “System Structure and Parameterization Report 1.0,” Modelica Association, 2019.

[11] W. Schamai, L. Rogovchenko-Buffoni, N. Albarello, P. Miranda and P. Fritzson, “An Aeronautic Case Study for Requirement Formalization and Automated Model Composition in Modelica,” in *Proceedings of the 11th International Modelica Conference*, Linköping, 2019.

[12] S. Friedenthal, A. Moore and R. Steiner, A Practical Guide to SysML: The Systems Modeling Language, ELSIVIER, 2014.

[13] Modelica Association, “FMI (Functional Mock-up Interface)-Tools,” Modelica Association, [Online]. Available: https://fmi-standard.org/tools/. [Accessed 04 05 2022].

[14] R. Hällqvist, On Standardized Model Integration: Automated Validation in Aircraft System Simulation, Linköping: Linköping University Electronic Press, 2019.

[15] D. Rosca and L. Domingues, “A systematic Comparison of Roundtrip Software Engineering Approaches applied to UML Class Diagrams,” *Procedia Computer Science,* pp. 861-868, 2021.

[16] S. Sendall and J. Kuster, “Taming Model Round-Trip Engineering,” in *Proceedings of Workshop on Best Practices for Model Driven Software Development*, Vancover, 2004.

[17] OpenCPS project partners, “The official OpenCPS project website,” 08 02 2022. [Online]. Available: https://www.opencps.eu/.

[18] EMBrACE project partners, “Environment for model-based rigorous adaptive co-design and operation of CPS,” [Online]. Available: https://www.embrace-project.org/. [Accessed 08 02 2022].

[19] P. Nuzzo, A. L. Sangiovanni-Vincentelli and R. M. Murray, “Methodology and Tools for Next Generation Cyber-Physical Systems: The iCyPhy Approach,” *INCOSE Incternational Symposium,* vol. 25, no. 1, pp. 235-249, 2015.

[20] E. Herzog and H. Andersson, “Initial Experience in Contracts Based Systems Engineering,” *INCOSE International Symposium,* vol. 19, no. 1, pp. 1510-1520, 2009.

[21] M. Otter, T. Nguyen, D. Bouskela, L. Buffoni, H. Elmqvist, P. Fritzson, A. Garro, A. Jardin, H. Olsson, M. Payelleville, W. Schamai, W. Thomas and A. Tundis, “Formal Requirements Modeling for Simulation-Based Verifcation,” in *Proceedings of the 11th International Modelica Conference*, Linköping , 2015.

[22] P. Fritzson, A. Pop, A. K., A. Asghar, B. Bachman, W. Schamai, M. Sjölund, B. Theile, J. Tinnerholm and P. Östlund, “The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development,” *Modeling, Identification and Control: A Norwegian Research Bulletin,* vol. 41, no. 4, pp. 241-295, 2020.

[23] W. Oberkampf and C. Roy, “A comprehensive framework for, verification, validation, and uncertainty quantification in scientific computing,” *Computer methods in applied mechanics and engineering,* pp. 2131-2144, 2011.

[24] M. Eek, On Credibility Assessment in Aircraft Systems Simulation, Linköping: Linköping University Electronic Press, 2016.

[25] R. Hällqvist, R. Munjulury, R. Braun, M. Eek and P. Krus, “Engineering Domain Interoperability Using the System Structure and Parameterization (SSP) Standard,” in *Proceedings of 14th Modelica Conference 2021*, Linköping, 2021.

[26] F. Bordeleau, “Model-Based Engineering: A new Era Based on Papyrus and Open Source Tooling,” in *OSS4MDE@MoDELS*, Valencia, 2014.

[27] B. Maria, J. B. William, L. G. Lawrence, P. Joseph, E. M. Gary, J. S. Martin and W. Jody, “NASA Standard for Models and Simulations: Credibility Assessment Scale,” in *AIAA Aerospace Sciences Meeting*, Orlando, FL, 2009.

[28] M. Eek, R. Hällqvist, H. Gavel and J. Ölvander, “A Concept for Credibility Assessment of Aircraft System Simulators,” *AIAA Journal of Aerospace Information Systems 13.6,* p. 219–233, 2016.