AIRCRAFT SYSTEM DESIGN: A MODEL-BASED AND COLLABORATIVE APPROACH

Viola Voth[1] & Oliver Bertram[1]

[1]German Aerospace Center (DLR), Institute of Flight Systems, 38108 Braunschweig, Germany

Abstract

This paper examines challenges faced by aviation industry stakeholders in developing complex systems within a decentralized environment. As aircraft, engine, and equipment manufacturers grapple with multinational structures and geographically dispersed teams, traditional document-centric approaches prove cumbersome. This paper explores potentials of Model-Based Systems Engineering (MBSE) to address these challenges. MBSE minimizes document reliance, enhances collaboration, and offers a centralized system model to ensure consistency. The approach not only promises efficiency gains but also a less segmented development process in the intricate landscape of the aerospace industry. The central research question investigates the successful integration of aircraft design and MBSE to navigate the complexities of distributed system development. This inquiry sets the stage for introducing a model-based multidisciplinary method and illustrating its application through a use case.

Keywords: Aircraft System Design, Model-Based Systems Engineering (MBSE)

1. Introduction

The aviation industry is faced with the challenge of developing complex systems which fulfill the requirements of safety, efficiency and performance [1]. Complex systems consist of a large number of elements that interact through diverse and non-linear relationships both with and independently of each other [2]. The development process of such systems requires a finely balanced collaboration between different departments and development teams in order to ensure the frictionless interaction and extremely important coordination of the numerous interfaces.

Until now, the development of aircraft systems has been managed through the use of documents [3]. This means that separate documents such as detailed technical specifications, design descriptions or test plans are created in each discipline, which serve as input for the downstream development departments. However, this approach carries a number of risks, particularly inefficiencies and errors, which can also affect development time and costs. The greatest difficulty is coordinating and managing the interdependencies between the documents and making the current development status available to all of their participants [4, 5].

One approach that has recently emerged as promising for the development of complex systems is Model-Based Systems Engineering (MBSE). MBSE aims to use a central system model [6] to minimize the documentation effort required in the classical development process and to reduce the effort required to maintain consistency at the same time. These models are created using the modeling language SysML. SysML stands for Systems Modeling Language and is a graphical modeling language. The use of these formal modeling languages minimizes misinterpretations, simplifies simulation and analysis, supports model reuse, and ensures development consistency. At the same time, the model enables efficient management of different solution variants and improves the traceability of changes.

Above all, the model is intended as an effective knowledge repository that captures the knowledge of all participants in a standardized, machine-readable form [7]. The ultimate goal is to store all information in one central location [1, 4, 6–10].

However, conventional MBSE methodologies are not convenient for aircraft development. One central issue that is missing, is an approach for task assignment and distribution. The parallel development of different subsystems requires seamless coordination and information transfer, taking into account the dependencies and interfaces between the systems. For example, requirements must be continuously exchanged between teams and subsystems. The goal is a consistent development process. Without this assignment process, model-based aircraft design is not sufficiently feasible.

Since model-based development is to be retained due to its many advantages, an extension of the MBSE methods is required. The goal of this paper is to develop such an extension, especially for task assignment. The central research question is therefore: How can an aircraft development be successfully established using the MBSE approach in order to meet the challenge of decentralized system development in the aerospace industry?

2. State of the art

Aerospace companies face the difficult challenge of continuously optimizing the development of new aircraft and their systems to meet increasing customer requirements and tough competitive conditions. Reducing development and production cycles as well as the associated costs over the entire lifecycle of an aircraft is a constant challenge. The completion of an airplane is the result of a complex compromise that reflects the knowledge and experience of many engineers in an aircraft manufacturer's multidisciplinary development groups [11]. The design and construction of aircraft and their systems requires the integration of a common product and process model distributed across multiple sites and/or suppliers [12].

2.1 Aircraft Development Process

The aircraft development process is a complex and iterative process that goes through several phases to build a functional and safe aircraft. In the specification phase, aircraft configuration studies are conducted to determine appropriate technologies and technical requirements. This is followed by the iterative concept phase in which conventional, innovative and revolutionary aircraft concepts are evaluated for technical and economic feasibility. The selected aircraft configuration is then further developed in the preliminary aircraft design. This involves several specialized departments that design and optimize the airframe and aircraft systems grouped by ATA chapters [13]. This process is iterative, allowing for continuous improvements and adjustments. In the detailed aircraft design, the individual assemblies are finally designed in detail, whereby development tasks can also be outsourced to suppliers. Close cooperation between the various development teams and suppliers is essential to achieve an optimal result. MBSE is becoming increasingly important as an interdisciplinary approach to managing this complexity.

2.2 Model-Based Systems Engineering

The International Council on Systems Engineering (INCOSE) defines MBSE as „the formalized application of modeling to support system requirements, design, analysis, verification and validation activities that begin in the concept phase and extend throughout development and later lifecycle phases“ [8]. Unlike the document-based SE approach, which has a number of challenges such as managing document dependencies, the goal of MBSE is to minimize written documentation [4, 5]. At the same time, a central system model should reduce the effort required to maintain consistency. All information relevant to the development of the system is contained in this model. This visionary system model, see Figure 1, would contain all information so that it could be considered as "single source of truth". Due to several factors, such as the size of the model and usability, such a visionary system model is practically impossible to implement [7]. In practice, there is an interaction between the SysML system model and the discipline-specific models, see Figure 1. The different discipline-specific models describe different aspects of the system in detail. The SysML system model, on the

other hand, incorporates certain parts of these discipline-specific models. The goal is to represent the underlying understanding of the system. Engineers view the model from their discipline-specific perspective, which gives them all the relevant data about the system. Avoiding media disruption and centralizing information relevant to the development of the system in a system model makes it easier to maintain consistency and enables efficient cross-referencing between different aspects of the model.

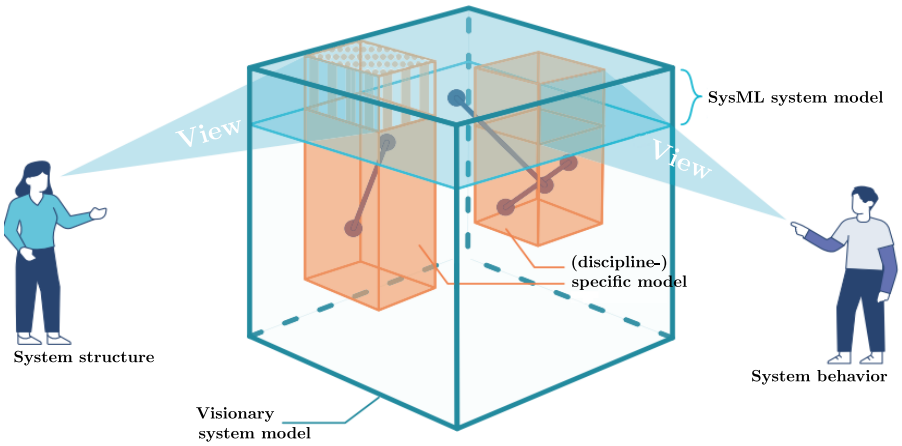

Figure 1 – Abstract representation of the system model, adapted from [7]

The modeling triad of language, tool and method plays a central role in MBSE. Language standards define which elements should be used and how their syntax and semantics are designed, while methods provide guidelines and recommendations for the modeling process. Tools are used to design and efficiently manage the complex system models. They provide the practical basis for applying the model-based approach. Well-known tools include Cameo Systems Modeler from Dassault Systèmes, Enterprise Architect from SparxSystems, Rational Rhapsody from IBM, Visual Paradigm from Visual Paradigm, Papyrus from Eclipse and Capella from PolarSys. The advantage of these tools is that they allow developers to explicitly represent their abstract and conceptual models and to exchange information on this basis. Furthermore, they are designed to actively support developers in the correct use of the modeling language. The effective coordination and interaction of these three pillars is important for an effective work in the field of MBSE [14].

In recent years, several methods have been established in the field of systematic system development, as well as some methodologies that are defined as a summary of related processes, methods and tools according to [14]. These methods and methodologies include SYSMOD [15], OOSEM [14], Harmony SE [14], Vitech MBSE Methodology [14, 16], State Analysis [14] and ARCADIA [17]. These methodologies can be divided into two categories, as described by [5]. The first category includes methodologies that provide concrete assistance in building a system model, such as MagicGrid and SYSMOD. They have a strong focus on structured modeling. The second category includes methodologies that are applicable not only to MBSE, but also to SE in general, such as IBM's Harmony SE methodology. These methodologies provide tools and guidelines that support the entire development process.

Despite their diversity, most methods share common steps that correspond to the left branch of the V-model. These include the identification and analysis of requirements, the identification of functions and the derivation of architectures from functional and non-functional requirements that describe the system structure of the system from both a logical and physical perspective. The only exception is the OOSEM methodology, which skips the functional development step.

2.3 V-Model

The V-Model is the most established and widely used process model for the development of complex systems in the aerospace industry. Basically, V-models serve as a framework for the interaction of development processes and offer a targeted linking of the necessary tasks in the SE [7]. The characteristics of the V-model basically comprise the three central steps: Requirements Engineering, Decomposition and Integration and Verification and Validation. The *requirements development* includes the systematic elicitation of requirements and the management of requirements changes. *Decomposition and Integration* are necessary to capture the high complexity of today's systems and make it manageable. This process takes into account the system environment and the desired interaction of the system of interest (SoI) to be developed within the higher-level system of systems (SoS). The complexity is reduced by breaking it down into manageable elements, with the subsequent activities focusing on the definition of solutions.

The RFLP concept is often used in the context of the V-model. The abbreviations stand for the following as can be seen schematically in Figure 2:

1. **Requirements:** Requirements are precise descriptions of the functions, services and constraints that a system must satisfy. They can be divided into functional requirements, which define the necessary system functions and non-functional requirements, which define quality characteristics such as performance, reliability and usability.

2. **Functions:** The identification of the functions that the system must perform to meet the defined requirements is a detailed specification of the system's functionality.

3. **Logical System Architecture:** The logical system architecture abstractly describes the structure of the system by defining the logical elements and their interactions. It determines how the functions of the system are distributed among the different elements.

4. **Physical System Architecture:** In contrast, the physical system architecture describes the actual implementation of the system and refers to the physical distribution of components.

If a solution element is too complex, it is treated as a subsystem and divided into subordinate elements. *Verification and Validation* monitors that the properties of the system are continuously checked. This ensures that the developed system meets the requirements and satisfies the needs of the stakeholders.

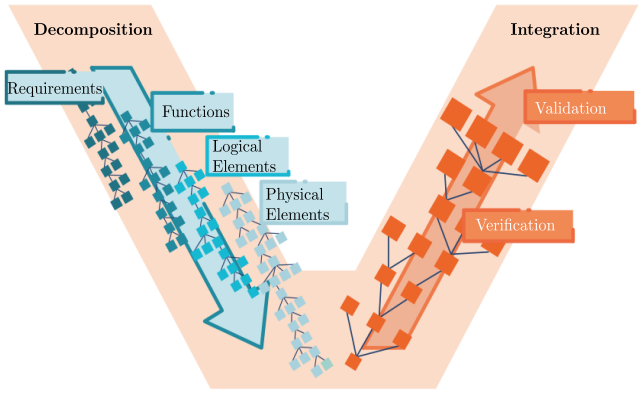

Figure 2 – RFLP Concept within the V-Model, adapted from [7]

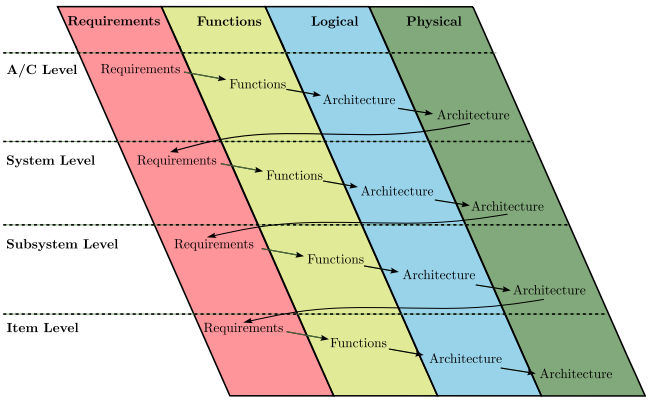

Figure 3 – V-Model for general aircraft development, adapted from [18]

For model-based aircraft development, the V-model of the RFLP approach has been extended by Dollinger et al. [19] as well as Esdras and Liscouët-Hanke [18] to meet the requirements of aircraft design. Due to the complexity of an aircraft, it is divided into smaller subsystems. At each stage of development, the methodology goes through the four RFLP steps. The approach of Esdras and

Liscouët-Hanke [18] emphasizes the step-by-step definition of solutions, starting with functions, which are then realized through logical and finally physical solution elements. New requirements may arise at the next system level as the physical architecture evolves.

Furthermore, the V-model is very well known in the document-centered approach in the aviation industry. The CS-25 [20] refers several times to ARP 4754B [21] in the development of safety-critical aircraft systems. The guideline is also based on a V-model, which shows the integration of development and safety. This V-model is relatively similar to the concepts already presented, but not completely congruent [22]. The ARP approach has also not yet been fully implemented on a “model-based“ basis. As we are concentrating on model-based development in this paper, we will only deal with the RFLP approach and leave the ARP process out of this paper.

3. Scientific Problem and Research Question

In this chapter, the research problems are explained and the scientific questions will be formulated. There are two separate problems, which will be independently addressed in Section 3.1 and Section 3.2.

3.1 Tool Adaptation - Customizing the Connections

The Cameo Systems Modeler tool from Dassault Systèmes was used for model-based development in this paper. This tool is one of the most prominent and widely used on the market and has the advantage of being flexible and offering a great deal of freedom. During the model-based development of a new system according to the RFLP concept, we found that the traceability between the various elements (requirements, functions, logical and physical elements) is lost most of the time. In a final system architecture, it is difficult to trace which requirement, function or logical element led to which component in the system architecture. It was often not possible to understand why a particular component has been selected in the final system architecture. The reason for this is that Cameo, like most other tools, is based on the SysML language. This language consists of a limited number of standard connections that can be set between elements such as requirements, etc. So-called "trace" connections are often used to indicate the origin of an element. However, by using only one type of connection, information that is important for traceability is lost. To prevent this, the connections in the Cameo tool must be adjusted. By adjusting the relationships between the elements, a higher level of granularity is achieved. This allows a more precise assignment of requirements, functions and logical elements to the corresponding components. By implementing these enhancements, we aim to improve the traceability and transparency of the modeling process, thereby increasing the efficiency and quality of aircraft development.

The customization of the connections not only makes the origin information transparent, but also makes it possible to make automated queries to the model. For example, it is possible to check whether all requirements are assigned to at least one element. This automation helps to ensure the completeness and consistency of the model. Given the potential benefits of automation and improved modeling efficiency, the following scientific question arises: **How can the Cameo tool be adapted to effectively implement the RFLP approach in practice?** Section 4 covers this topic in detail.

3.2 Assignment Process

When examining the RFLP concept in the literature, it was found that this concept is typically only demonstrated on monolithic systems. A system is called monolithic if it is considered as a unit without a clear division into individual modules and levels. However, such a simplified view is not sufficient in the aviation industry, where complex systems with multiple hierarchical levels are common. This means that different teams work in parallel on the design and implementation of different subsystems that together form the overall system (aircraft). These parallel activities save time and speed up the development process, but also require careful coordination and integration to ensure that all systems interact smoothly. These challenges are not limited to the aerospace industry, but can also be found in other industries where complex systems are developed by different teams. It is therefore obvious

that a methodological extension of RFLP concepts is needed to meet the challenges of multi-level systems, especially in contexts such as aircraft development.

The extension of the RFLP concept to the requirements of multi-level systems is accompanied by an allocation of elements between different systems and/or development teams. Assignment is critical because the complexity of the systems and the multinational organizational structures of the companies involved require a clear and seamless transfer of elements for a consistent development process. The assignment of elements is therefore a crucial key point for effective collaboration and integration in the development of highly integrated systems.

Against this background, the following scientific question arises: **How can an effective assignment of elements (requirements, functions, logical and physical elements) between different systems in model-based aircraft development be defined and implemented to ensure a smooth workflow, clear responsibilities and improved coordination?** This question is thoroughly addressed in Section 5.

4. Tool Adaptation

This chapter deals with the first scientific question formulated in the previous chapter. It elaborates the customizations within the Cameo tool to address the challenge outlined in Section 3.1.

4.1 Method

As explained in detail in Section 2.3, the development of a (sub)system requires various elements such as requirements, functions, logical and technical elements. These elements often arise during the development process due to various reasons. In order to be able to understand which source led to the creation of the individual elements, connections are made between the elements. To answer the first scientific question, the following section first evaluates what connections are needed between the various elements. It then explains how to integrate the new connections into the Cameo model.

4.1.1 Types of elements and their relations

The development of a new system (level n) usually starts with a list of **requirements**. The requirements can either be derived from a superordinate system (level n-1), from a system at the same level (level n) or a subordinated system (level n+1). Different reasons can lead to a requirement, see Figure 4. They can be:

1. describe a customer need (use case),
2. describe an intended system behavior or its performance
3. describe a functionality required for the operation of a specific system element
4. reflect a design constraint or guideline imposed by regulations, standards or design guidelines (Regulation)
5. reflect the result of a safety analysis (see "Safety Analysis"), or
6. specify another requirement (see "Requirement").

Tool support is essential to make the exact origin of requirements transparent during development. Requirements are never created out of the blue, but are always derived from one of the above causes. Therefore, the origin of each requirement must be clearly stored in the tool. As most requirements are derived from one of the above causes, the "derived" link between the origin and the requirement is set in the system model. The only exception is when a requirement specifies or refines another requirement. In this case, the "refines" relationship is used. Relationships are always set from the requirement to its origin, see Figure 4. By setting the relationship from the requirement to its origin, traceability is facilitated and the question of "why" is stored in the model.
Requirements can in turn be fulfilled directly via **functions**, see Figure 5. A function describes a system task in an abstract and solution-neutral way. In addition to the direct fulfillment of a requirement, functions can also be created indirectly.

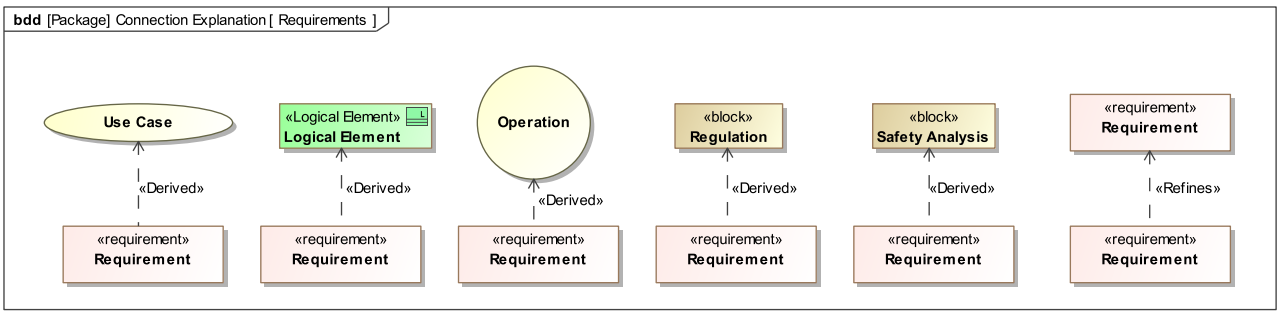

Figure 4 – Possible reasons that lead to a requirement

They can be either

1. decompose another function, or
2. be derived from a logical or technical element.

In order to trace the origin of a function in the model, different relationships can be used depending on the origin. If a function directly fulfills a requirement, the "implements" connection is selected, see Figure 5. However, if an abstract function is further decomposed, the "decomposition" connection is used and if the function is derived from a logical or technical element, the "derived" connection is used as with the requirements. A function derived from a logical or technical element always occurs when a design decision has been made. For example, the decision that a fuel cell system (logical element) will provide the necessary electrical power in future aircraft can lead to a function called "Manage Fuel Cell Membrane Temperature" being derived.

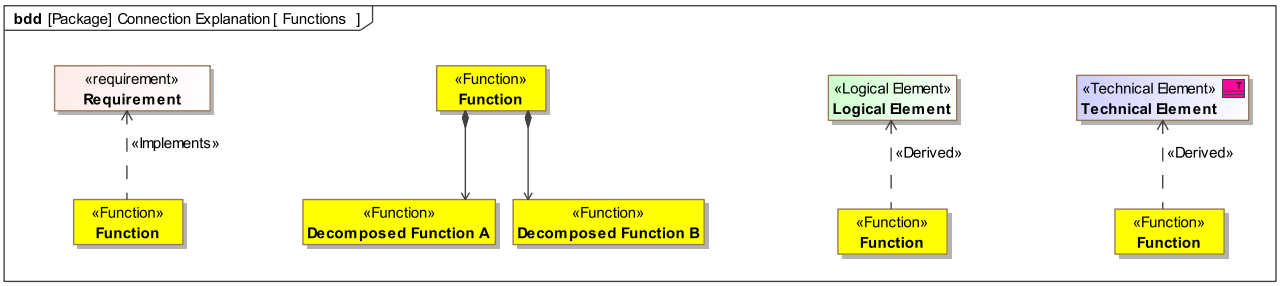

Figure 5 – Possible reasons that lead to a function

Logical Elements describe the system components in an abstract and conceptual way and can be combined to form a logical system architecture that represents the structure of the system to be developed. This architecture consists of an arrangement of logical elements that perform the functions of the system.
As shown in Figure 6, logical elements can occur in many ways:

1. They can fulfill a requirement directly,
2. can be allocated from a function, or
3. can be derived from another logical element.

Depending on the origin of a logical element, different links are used. If the element directly implements a requirement, the "implements" association is used. If the logical element is intended to perform a specific function, the "allocate" link is used. However, if the element is derived from another logical element, the "decompose" link is taken, same as for functions.

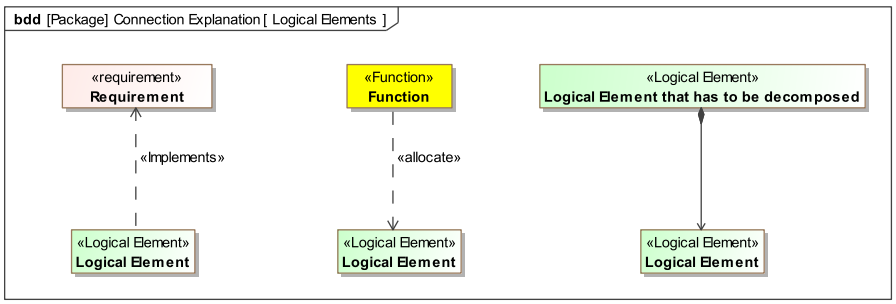

Figure 6 – Possible reasons that lead to a logical element

Technical Elements represent the actual components of the system. The elements are used to develop the physical architecture of the system. The architecture in turn influences the underlying layer. The elements can occur in two different ways. Either

1. they arise directly from a requirement, or
2. they realize a logical element

Depending on the cause of a technical element, different links are used. If a requirement is directly satisfied by a technical element, the "implements" link is used. If, on the other hand, a technical element realizes a logical element, the "realizes" relationship is used.

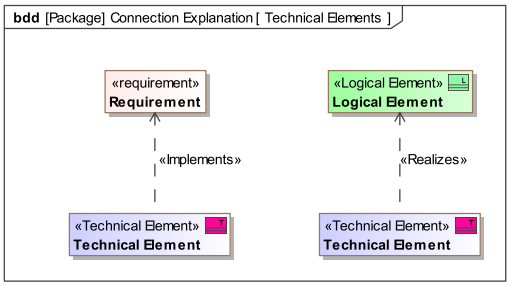

Figure 7 – Possible reasons that lead to a technical element

4.1.2 Possible paths from a requirement to a technical element

As just described in detail, there are several ways that can lead to a component. The different ways can be seen schematically in Figure 8. A basic distinction can be made between the following:

- **Option 1:** Describes the "ideal" path from requirements definition through functional derivation to logical and then physical element development.
- **Option 2:** A requirement can be implemented directly through a logical element. This is done by setting the "implements" relationship.
- **Option 3:** A requirement can be implemented directly through a technical element. As with option 2, this is done by setting the "implements" relationship.
- **Option 4:** A requirement can lead to a technical element through several intermediate steps. For example, further functions can be derived from logical elements, which in turn refer to a logical element.
- **Option 5:** A requirement can be further specified by a new requirement using the "refines" link.
- **Option 6:** A function is decomposed into at least one further sub-function, which in turn is assigned to a logical element
- **Option 7:** A logical element is divided into at least one other logical element

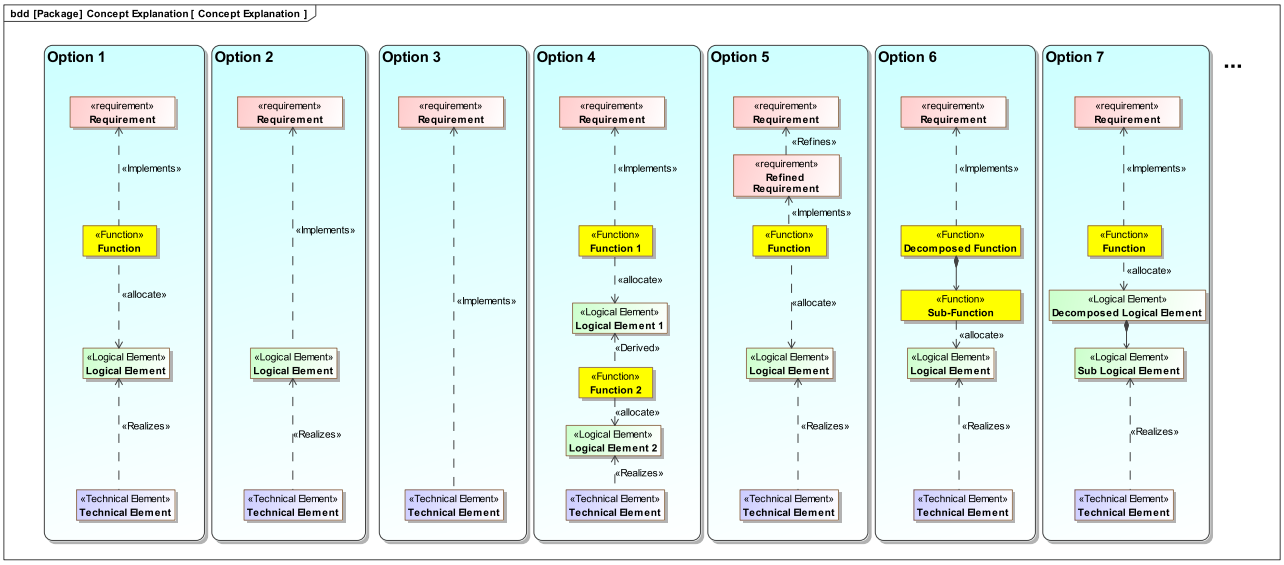

Figure 8 – Possible paths from a requirement to a technical element

In addition to the options described above, there are numerous other possibilities resulting from the various combinations. The matrix shown in Figure 9 provides a clear overview of how the various elements can be combined. The matrix can be read as follows: The source element is shown in the rows and the target element in the columns. The connections point from the source element to the target element. For example, a requirement (column) can be refined and derived from other logical elements, processes, regulations, safety analyses and use cases (row), as shown in Figure 4 and 9. This flexibility in the relationship allows for a wide range of potential development paths.

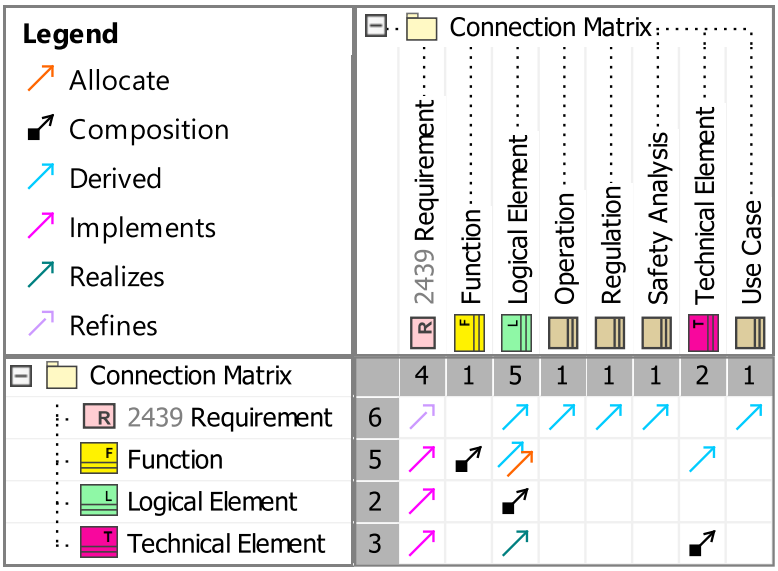

Figure 9 – Matrix with the possible connections between the different elements

4.1.3 Adaptations to SysML

In order to improve the comprehensibility and usability of the tool, specific adaptations to SysML have been made as part of this work. For example, both element stereotypes and connection stereotypes have been renamed to more clearly define their respective functions. These changes can be found in Table 1, which shows the different element types as well as the different connections and their definitions.

Table 1 – Terminology

Terminology	Definition
Element Types	
Requirement	Requirements are formal, structured specifications that clearly describe what the system to be developed must be able to do, which boundary conditions apply and which system properties are desired [7]
Function	Describe the purpose of the system in an abstract and solution-neutral way
Logical Element	Describe the components of the system in an abstract way
Technical Element	Represent the actual components of the system
Connection Types	
Implements	Refers from a requirement to an element that fulfills this requirement
Allocate	Refers from a function to a logical element
Realizes	Refers from a technical to a logical element
Decomposition	Breaks an element down into sub-elements
Derived	Refers from a requirement to its origin
Refines	Refines a requirement
Assign	Describes the assignment of an element from one system to another
Accept	Accepting the assignment of an element
Reject	Rejecting the assignment of an element

A potential disadvantage of this adaptation could be the large number of new connections. The more options, the more difficult it becomes to manage them. It is also possible that unwanted connections are inserted. To prevent this, the tool has been modified in a way that only allowed connections between two elements can be set.
As mentioned in Section 3.1, another benefit of this clear naming is the ability to generate automated queries directly in the model. The specific naming of links makes it possible to search for specific relationships between elements. For example, requirements can have only "derived" or "refines" connections to show the origin of the requirement. This unambiguous mapping makes it possible to formulate queries that search the entire model and check whether each requirement has at least one "derived" or "refines" relationship. If this is not the case, the affected requirements can be automatically displayed in a table. The new element types can also be used to create specific tables that display, for example, all the functions in the model. This automation significantly reduces the system architect's workload, as potential errors or inconsistencies can be quickly identified and resolved. This not only contributes to modeling efficiency, but also increases the reliability and quality of the overall system.

4.2 Example of application

To illustrate the benefits of the tool adaptation, this chapter presents an example of a possible development path. The thermal management system (TMS) of a fuel cell system (FuCS) is used as an example. Assuming that the FuCS provides the electrical power for the propulsion in future aircraft, it is part of the thrust generation system (TGS). A schematic representation of the FuCS can be found in Figure 10.
Polymer electrolyte membrane (PEM) fuel cells have proven ideal for mobile applications. These cells efficiently convert chemical energy from hydrogen and oxygen into electrical energy and heat without producing pollutants or noise [23]. The functionality of a PEM fuel cell is described in detail in [24, 25]. In order to provide sufficient electrical power to the systems on board an airliner, voltages in excess of 200 V are required. Since a fuel cell generates voltages between 0.6 - 0.7 V on average

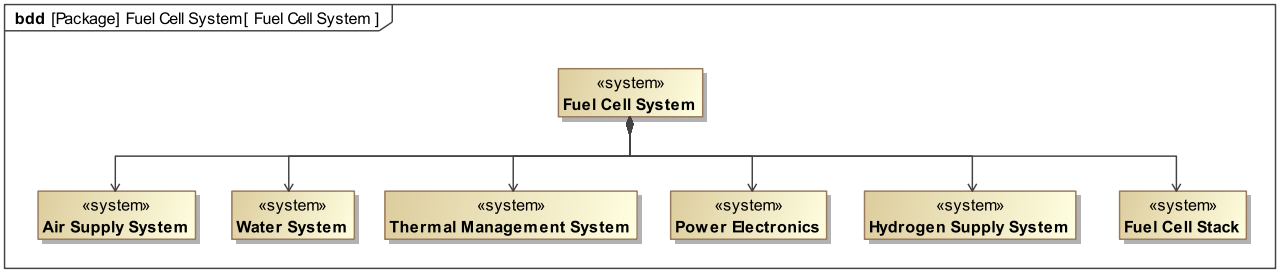

Figure 10 – Schematic structure of the FuCS

[25], the cells are connected in series, whereby the individual voltages add up and the total output increases [26, 27]. These cells are called a fuel cell stack.
In addition to the fuel cell stack itself, other subsystems are required to generate power, collectively referred to as the FuCS, see Figure 10. These include the hydrogen and oxygen supply, the water module, the cooling system and the power electronics. To better understand the complex decision and the resulting requirements and functions, an example from the development of a TGS for a future aircraft is considered below. Let us assume that there is a requirement for thrust generation to be CO_2 and N_2O free. This requirement could then lead to the decision to use a PEM fuel cell to generate the electrical power, see Figure 11.

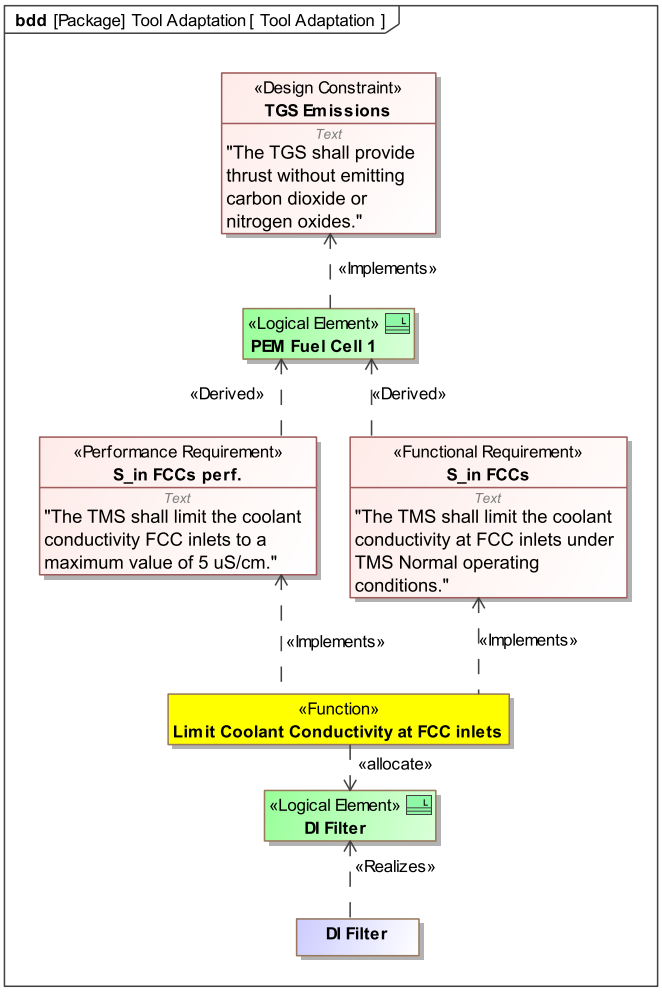

Figure 11 – Example that shows how a requirement is derived into a technical element (option 6)

The choice of a PEM fuel cell places additional requirements on the TMS. For example, excessive conductivity of the coolant would cause a short circuit within the fuel cell stack. Therefore, there are requirements to limit the conductivity of the coolant. This leads to the implementation of a specific function, such as "Limit Coolant Conductivity at Fuel Cell Inlets". This function, in turn, may require the assignment of a logical element, such as a deionisation (DI) filter, to ensure that the requirements are met. This illustrates the complex interweaving of requirements, decisions and functions in the development process of a highly integrated system.

5. Assignment Process

Another central problem, which was described in Section 3.2 is the missing assignment of elements within the development process. Since no approach to this process has yet been described in the literature, this chapter presents a possible approach. This approach must meet several requirements. These include a seamless and unambiguous exchange of elements between systems/development departments to ensure a consistent development flow from one development phase to the next. In addition, the approach should include the need for traceability to ensure smooth interaction between (sub)systems. Avoiding information loss throughout the design process is critical to achieving a highly integrated and efficient aircraft architecture. The goal is to allow the different teams to work separately on their respective (sub)systems, but to develop in a single model. This comprehensive approach is designed to improve collaboration between different interdisciplinary teams to meet the complexity of the aerospace industry.

5.1 Method

Due to the complexity of the overall aircraft system, the methodology includes a hierarchical subdivision into different levels, each level contributing to the development of the overall system. An advantage of this hierarchical structuring is that it provides a comprehensive framework for systematically dealing with the complexity of the overall system, thus promoting a more detailed and structured approach to system development. The hierarchical structure shown in Figure 12 illustrates the relationships between the different levels and shows the general development process along the left branch of the V-model.

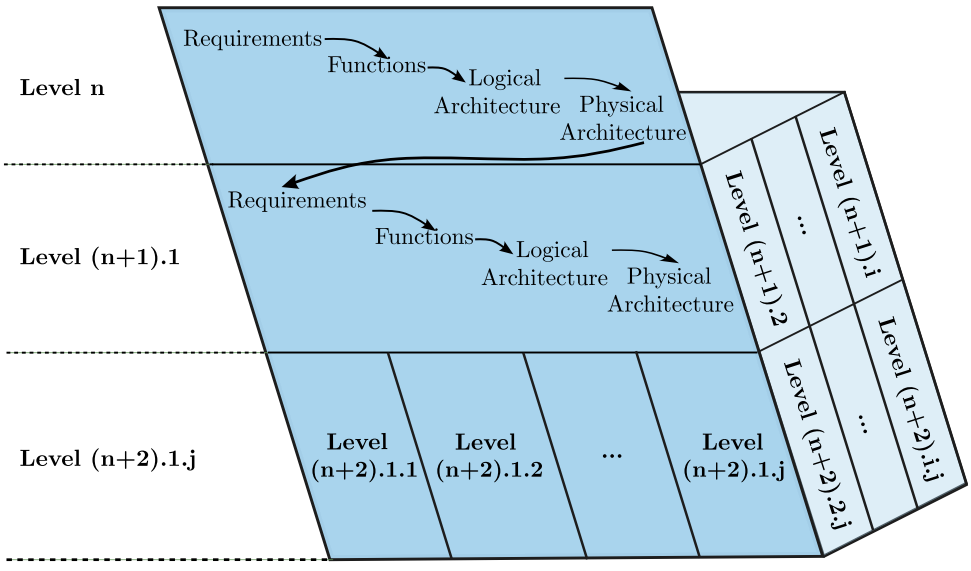

Figure 12 – RFLP specific V-Model for multi-level system development

5.1.1 Hierarchy Levels

The number of levels may vary from project to project. One option is the following structure:

- **Level 0 (Aircraft Level):** Represents the highest level, which includes the entire aircraft.
- **Level 1 (System Level):** Represents a system within the airplane.

- **Level 2 (Subsystem Level):** Represents an individual subsystem within the system.
- **Level 3 (Equipment):** Represents the equipment systems or individual components within a specific subsystem.

In this subdivision, level 0 is the top level and represents the entire aircraft. A decision made at this level has a significant impact on subsequent levels. The aircraft systems defined in the physical architecture are developed in parallel at the next level (level 1). This implies that the various development steps, such as requirements analysis, functional development and derivation of the logical and physical system architectures, are repeated according to the number of systems. From this level on, the V-model expands into the third dimension, see Figure 12. Many of these systems also have different interfaces to each other.
If a system is too complex, it can be broken down into further subsystems. A concrete example is the TGS with the assumption that it is the level n system, see Figure 13. Assuming that a FuCS provides the electrical power for the propulsion in future aircraft, the following levels result. The FuCS is assigned to level n+1 (level (n+1).1) and provides electrical power to an electric motor (level (n+1).2). This in turn drives a propeller (level (n+1).3). Due to its complexity, the FuCS (level (n+1).1) is divided into further subsystems (level (n+2).1.j). These include the fuel cell stack (level (n+2).1.1), the air supply system (ASP) (level (n+2).1.2), the hydrogen supply system (level (n+2).1.3), etc.

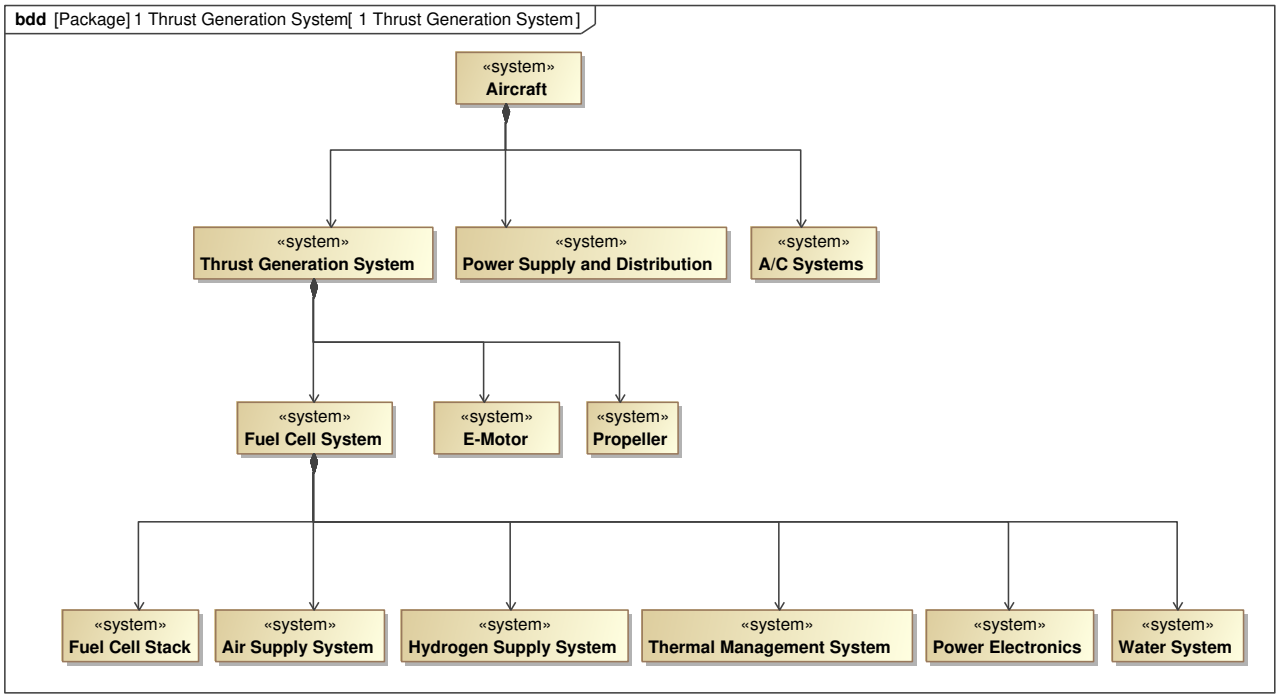

Figure 13 – Schematic structure of the aircraft

5.1.2 Assignment Process

As explained in Section 4.1.2, during the development of (sub)systems, new functions or requirements may arise, often as a result of design decisions. If these cannot be fulfilled by the (sub)system under consideration, they can be assigned to other (sub)systems or components. The same applies to requirements that are placed on a system to be developed. These come either from the higher-level system (level n-1), from a system at the same level (level n) or a subsystem (level n+1). For the purpose of clarity, the process is explained using a function assignment. However, it also applies to the assignment of requirements.

The development of a system (level n) begins with a set of functions that are assigned to the system. These functions are called "Assigned Functions" (see Figure 14) and are reviewed by the respon-

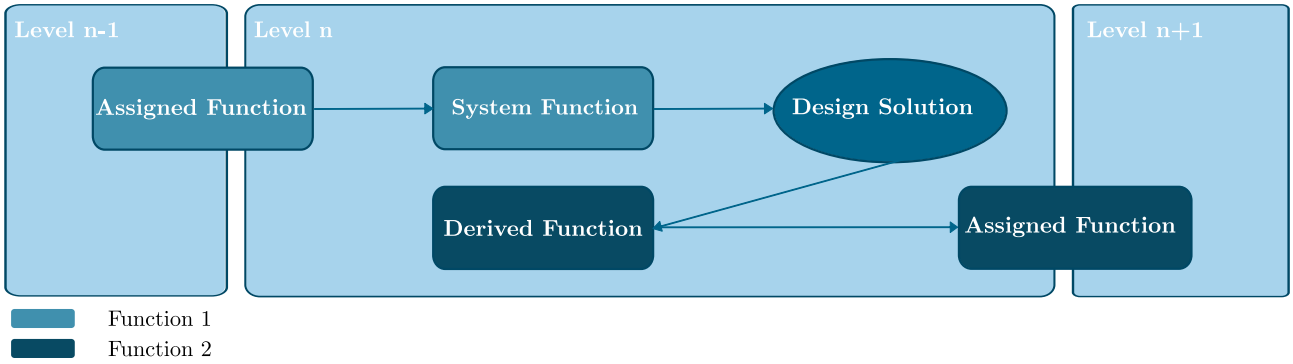

Figure 14 – General assignment process of a function

sible system architect. If a function is accepted and/or refined, it automatically becomes a "System Function" of system n. All these system functions must be considered and fulfilled during the system design phase. During system development, new functions may also be derived, often as a result of design decisions ("Derived Functions"). If these functions cannot be performed by the system under consideration, they can be assigned to other (sub)systems or components ("Assigned Function").

To illustrate this, we assume that a function of a subsystem (level n) is to be assigned to another (sub)system, see Figure 15. The assignment can be done in different ways, depending on the function and the system structure:

- **Direct assignment to a lower level (level n → level n+1)**
  If an assigned function describes the function of a lower level (n+1), it can be assigned directly, see Figure 14.

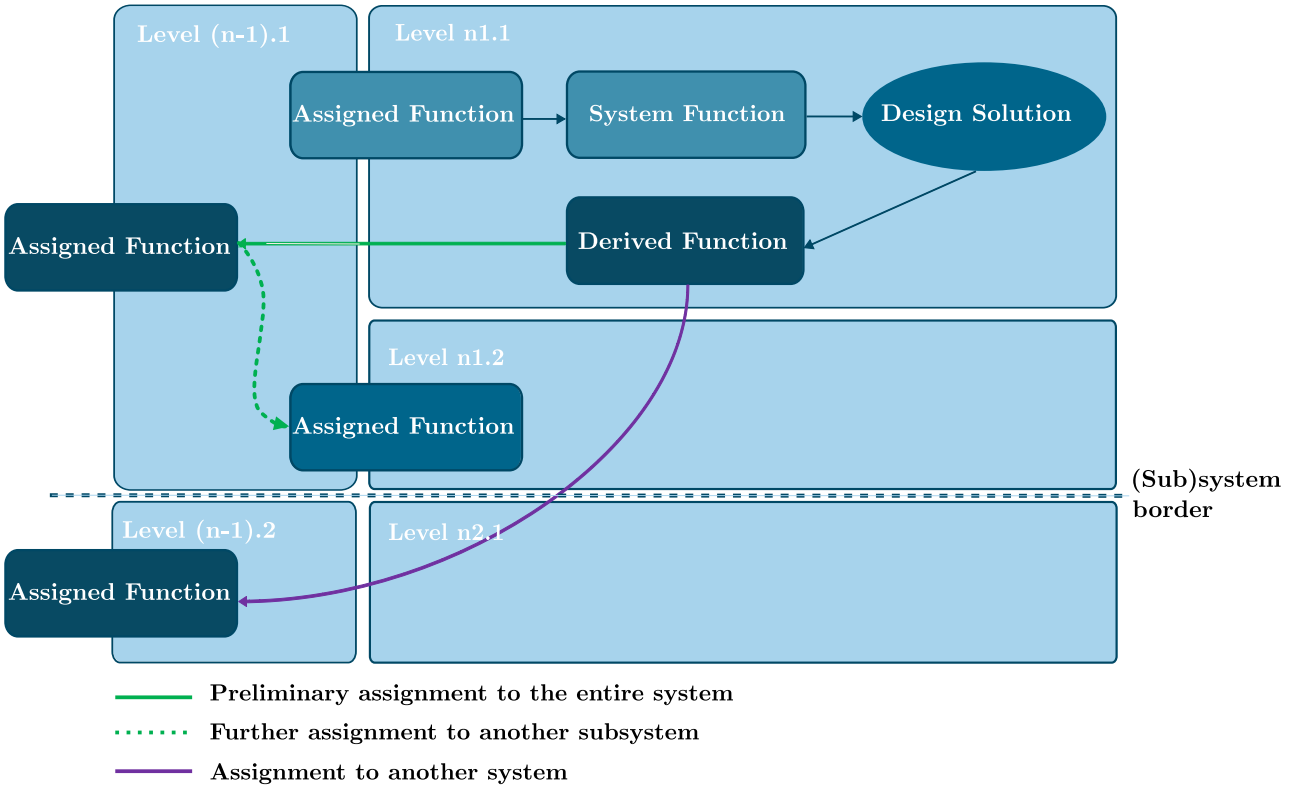

Figure 15 – Extended assignment process

- **Preliminary assignment to the overall system (level n → level n-1)**
  If the function is to be performed by another subsystem within the same system, but it is not yet clear which subsystem this will be, the function can be provisionally assigned to the overall system (level n-1), see Figure 15. In the initial phase, it is up to the system architect to determine which subsystem should fulfill the function. This preliminary assignment allows flexibility in decision making before a final assignment is made.

- **Assignment to another system (level n → level n-1)**
  If the function is to be performed by another system, the function can be assigned directly to that system. Cross-system assignment allows a clear definition of responsibilities between different systems, as shown in Figure 15.

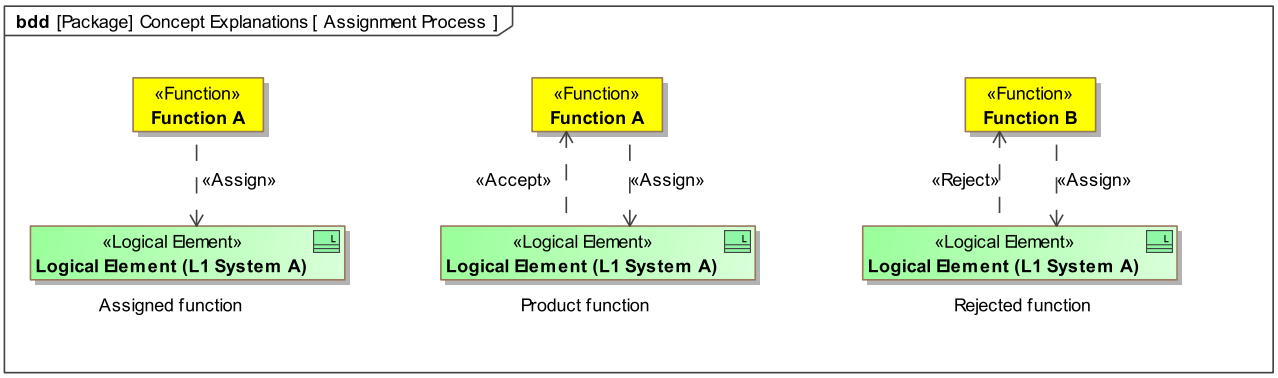

Figure 16 – Assigning a function to a logical element

The assignment process is implemented in the Cameo tool as follows. If a function is to be assigned to another system, the first step is to set up an "assign" relationship between the function and the logical element of the assigned (sub)system, see Figure 16. Using a table automatically generated in Cameo, the assigned function is displayed to the system architectures of the assigned system (L1 system A). The system architect can now decide whether to accept the function, making it a product function, or reject it. The rejected function, together with a reason for the rejection, is displayed to the system architect of the original system via a table, see Figure 17.

#	Name	Status	Reason for rejecting the function
1	Function A	Accepted	
2	Function B	Not Accepted	Because the function is not within our area of responsibility

Figure 17 – Function assignment overview for the system architect

5.2 Example of Application

In order to better illustrate the assignment process and to deepen the understanding, the assignment process is explained in the following chapter using an example. The assignment of functions within the FuCS to be integrated into the aircraft architecture of future aircraft is used as an example. The schematic structure can be taken from Figure 10 or Figure 13.
The oxygen supply on the cathode side is considered below as an example of function assignment. A compressor is used to compress the ambient air to the required pressure level. Since the increase in pressure is accompanied by an increase in temperature, the air must be cooled after compression [28].

During the system design of the oxygen supply system, the system architects discuss and decide with the FuCS architects that the temperature of the oxygen mass flow will be limited. The "Limit Air

Flow Temperature" function is created. In this example, it is assumed that the architects decide to use a liquid-cooled heat exchanger to cool the air mass flow. Based on this decision, other functions are required. On the one hand, the coolant must be both supplied to and removed from the heat exchanger ("Supply Coolant to Air Supply" and "Remove Coolant from Air Supply"). On the other hand, the solution includes the limitation of the minimum cooling mass flow and the maximum inlet temperature ("Limit Coolant Mass Flow" and "Limit Coolant Temperature at Air Supply Inlet"), see Figure 18.

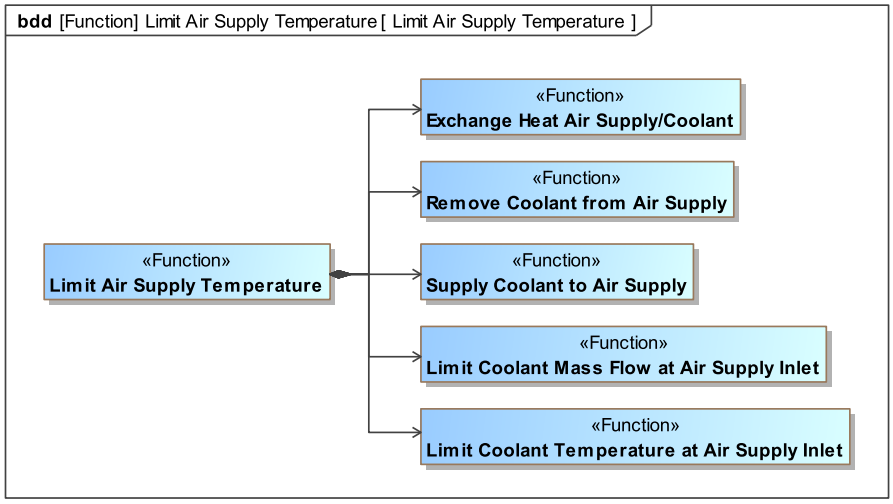

Figure 18 – Decomposition of the function "Limit Air Supply Temperature"

Due to the existence of a specialized subsystem, the TMS, within the fuel cell system, the oxygen system architects decide not to develop these subfunctions themselves, but to hand them over to the TMS team.

To do this, the functions described are assigned to the higher-level system, the FuCS. There, the FuCS architect decides whether the functions should be forwarded to the TMS. This step ensures that the responsible system architect at level n-1 remains informed about which functions are assigned to which subsystem. This procedure prevents the overall system architect from losing the overview.

Once the functions have been assigned to the TMS team, it is up to the responsible TMS architect to decide whether to accept or reject the functions. To make this process transparent, a table like the one in Figure 19 can be used. By setting the relationship to "Accept", the TMS Architect accepts the feature and is responsible for its implementation and fulfillment. As you can see in Figure 19 most of the functions are accepted. Only the "Supply Coolant to Air Supply" function has not yet been accepted. It is currently assigned to the TMS system, therefore its status is set to "Proposed". The table also shows the original system in the "Origin" column. This allows the TMS system architect to immediately see where the function originally came from.

#	Name	Status	Origin
1	Limit Coolant Mass Flow at Air Supply Inlet	Accepted	ASP
2	Limit Coolant Temperature at Air Supply Inlet	Accepted	ASP
3	Remove Coolant from Air Supply	Accepted	ASP
4	Supply Coolant to Air Supply	Proposed	ASP

Figure 19 – Example of a possible table in Cameo with all the details

6. Summary and Conclusion

The aerospace industry is faced with the challenge of developing complex systems. These systems consist of a large number of interacting elements developed by different departments. Coordinated collaboration is therefore essential to ensure the smooth interaction and coordination of the numerous interfaces. MBSE has proven to be a promising approach by using a central system model that minimizes documentation effort and increases consistency. The development is carried out according to the RFLP concept.

During the model-based development of a new system using this concept within the Cameo Systems Modeler tool, it was found that traceability between the various elements (requirements, functions, logical and physical elements) can be lost. It is often not possible to trace which requirement, function or logical element led to which component in the system architecture. This is probably due to the fact that the tool is based on the SysML language, which is based on a limited number of standard connections. Since the same connection type was often chosen between the elements, the information is lost in terms of traceability. For this reason, the standard connections have been adapted in this work, which increases the granularity. This adaptation significantly improves traceability and transparency within the development process, thereby increasing the efficiency and quality of aircraft development. In addition, automated queries can be made to the model to detect potential errors and inconsistencies.

We found that the RFLP concept is typically demonstrated only on monolithic systems, while complex systems with multiple levels of hierarchy are common in aerospace development. Therefore, a methodological extension of RFLP concepts is necessary to meet the challenges of multi-level systems. In particular, the effective assignment of elements between different systems and/or development teams is crucial for efficient collaboration and integration in the development of highly integrated systems. The extension presented here therefore primarily enables the effective assignment of elements between the different departments of the company. This increases transparency and minimizes the risk of misunderstandings and miscommunication. In addition, the continuous transfer of elements enables an efficient exchange of knowledge between the different project phases and thus facilitates the development of solutions. By clearly assigning tasks, duplication and inconsistencies are avoided, coordination is improved and responsibilities are clearly defined. This results in a more efficient way of working, better collaboration and a smoother development process.

So far, the RFLP extension presented in this paper has proven to be very successful. In the future, our approach could help optimize the development of complex aircraft systems in the aerospace industry and the challenges of distributed systems development.

7. Outlook

In addition to the tool adaptations and assignment process described in this paper, the consideration of design decisions is another important aspect of the development process that is often overlooked. During development, many decisions are made that are not currently considered in the current methods, but which have a significant impact on the final system architecture. The integration of design decisions into a suitable method offers a promising perspective for the future. This allows decisions made during the development process to be anchored in the model and stored permanently. The benefits of such integration are many: it improves the documentation and traceability of decisions, promotes a better understanding of the system, and facilitates future changes. These aspects will be discussed in detail in a forthcoming paper.

8. Contact Author Email Address

viola.voth@dlr.de

9. Copyright Statement

References

[1] Thomas Vosgien. *Model-based system engineering enabling design-analysis data integration in digital design environments: application to collaborative aeronautics simulation-based design process and turbojet integration studies*. Dissertation, Ecole Centrale Paris, 2015.

[2] Edward Crawley, Bruce Cameron, and Daniel Selva. *System Architecture: Strategy and Product Development for Complex Systems*. Pearson Education Limited, Harlow, 1. edition, 2015.

[3] Oliver Alt. *Modellbasierte Systementwicklung mit SysML*. Hanser, München, 2012.

[4] Helmut Scherer. *Modellbasierte Methoden zur Modellierung des Zielsystems und des Funktions-Gestalt-Zusammenhangs zur Unterstützung der Serienentwicklung von Baukästen am Beispiel von Hybrid-Triebstrangsystemen*. Dissertation, Karlsruher Institut für Technologie, Karlsruhe, 2016.

[5] Alexander Ahlbrecht. *Erweiterung von MBSE Prozessen bei der Entwicklung sicherheitskritischer Systemarchitekturen durch die Nutzung Formaler Methoden*. Masterarbeit, Technische Universität Braunschweig, Braunschweig, 2021.

[6] Jonas Andreas Powelske. *MBSE-gestützte Methoden zur Strukturierung und Anwendung von Baukästen in der Frühen Phase der PGE - Produktgenerationsentwicklung mechatronischer Steuergeräte im Einsatz in Kleinantrieben im Automobilbereich*. Dissertation, Karlsruher Institut für Technologie, Karlsruhe, 2023.

[7] Iris Gräßler and Christian Oleff. *Systems Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2022.

[8] Tim Weilkiens, Jesko G. Lamm, Stephan Roth, and Markus Walker. *Model–Based System Architecture*. Wiley, 2016.

[9] Melanie Hoffmann. *Model Based Systems Engineering as an Interdisciplinary Methodology for Energy System Development*. Dissertation, Technische Universität Braunschweig, Braunschweig, 2021.

[10] Florian Schummer and Maximillian Hyba. An approach for system analysis with model-based systems engineering and graph data engineering. *Data-Centric Engineering*, 3, 2022.

[11] Thierry Pardessus. Concurrent engineering development and practices for aircraft design at Airbus. *24th International Congress of the Aeronautical Sciences*, 2004.

[12] Martin Eigner, Daniil Roubanov, and Radoslav Zafirov. *Modellbasierte virtuelle Produktentwicklung*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[13] Cord-Christian Rossow, Klaus Wolf, and Peter Horst, editors. *Handbuch der Luftfahrzeugtechnik*. Carl Hanser Verlag, München, 1. edition, 2014.

[14] Jeff A. Estefan. Survey of model-based systems engineering (MBSE) methodologies. *Incose MBSE Focus Group*, 25(8):1–12, 2008.

[15] Tim Weilkiens. *Systems Engineering mit SysML/UML: Anforderungen, Analyse, Architektur*. dpunkt.verl., Heidelberg, 3. edition, 2014.

[16] omgwiki.org. Vitech model-based systems engineering (MBSE) methodology.

[17] eclipse.org. Arcadia: A tooled method to define, analyse, design & validate system, software, hardware architectures.

[18] Gustavo Franco Esdras and Susan Liscouët-Hanke. Development of core functions for aircraft conceptual design methodology and results. *Bombardier Product Development Engineering*, 2015.

[19] Daniel Dollinger, Julian Rhein, and Kevin Schmiechen and Florian Holzapfel. Be lean - how to fit a model-based system architecture development process based on ARP4754 into an agile environment. *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, 2021.

[20] EASA. Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes (CS-25), 2021.

[21] Aerospace, S.A.E. ARP4754B. Guidelines for Development of Civil Aircraft and Systems, 2024.

[22] S. Lübbe, M. Schäfer, and O. Bertram. Coupling of model-based systems engineering and safety analysis in conceptual aircraft system design. In *33rd Congress of the International Council of the Aeronautical Sciences*. 2022.

[23] H. Lüdders. *Modellbasierter Entwurf und Bewertung von multifunktionalen Brennstoffzellensystemen auf Flugzeugebene*. Dissertation, Technische Universität Hamburg-Harburg, Hamburg, 2014.

[24] Peter Kurzweil. *Brennstoffzellentechnik*. Springer Fachmedien Wiesbaden, 2013.

[25] Ryan O'Hayre, Sku-Won Cha, Whitney G. Colella, and Fritz B. Prinz. *Fuel Cell Fundamentals*. John Wiley & Sons, 3. edition, 2016.

[26] Karl-Heinz Schmid. *Wärmemanagement von Brennstoffzellen-Elektrofahrzeugen*. Cuvillier, Göttingen, 1. edition, 2009.

[27] Ralf Peters, editor. *Brennstoffzellensysteme in der Luftfahrt*. Springer-Verlag, 2015.

[28] Tim Burschyk. *Entwicklung und Bewertung eines brennstoffbasierten Klimatisierungssystems für luftfahrttechnische Anwendungen*. Masterarbeit, Technische Universität Hamburg-Harburg, Hamburg, 2020.