Chair of Aircraft Design
Lehrstuhl für Luftfahrtsysteme

Multidisciplinary Optimization of Hydrogen Fuel System Architectures for Conceptual Aircraft Design

Master Thesis

Submitted in partial fulfillment of the requirements for the degree M.Sc. Aerospace Engineering at the TUM School of Engineering and Design of the Technical University of Munich.

Supervised by	Prof. Dr.-Ing. Mirko Hornung
	Kristina Mazur, M.Sc.
	Chair of Aircraft Design
	Sparsh Garg, M.Sc.
	German Aerospace Center (DLR)
Submitted by	Diogo Neves
	Student ID: 03767257
Submitted on	June 10, 2025
Sequential number	LS-MA 24/07-EX

Abstract

The pursuit of sustainable aviation has promoted the evaluation of alternative aircraft propulsion technologies, such as hydrogen fuel cells. In light of this, this study attempts to reduce the existing knowledge gap related to the fuel system of such aircraft, in order to assist decision making in the conceptual and preliminary design phases.

For this purpose, system architecture optimization (SAO) is used to study the design space of the hydrogen fuel system of a fuel-cell-based regional airliner. The employed framework consists mainly of the ADORE, MDAx and RCE platforms. Together, these platforms enable the joint use of MBSE and MDO for the automatic exploration of architecture design spaces. In addition, CPACS is used as a central data schema.

Using system mass and electric power consumption as performance metrics, the results show that the fuel system's performance is strongly dependent on the type of heater used for the various heating functions of the system. Given this stark influence, other architectural decisions become less prominent. Among them are the type of insulation, type of pump, and the choice between a pressure driven and a pump driven system.

Two different optimization approaches are compared, with surrogate-based optimization showing superior performance to that of the NSGA-II algorithm. The benefits of using an automated approach in the system architecting phase are quantified by comparing it to more conventional approaches. Significant time savings are achieved, as well as a more comprehensive design space exploration.

This research demonstrates the use of system architecture optimization in a realistic engineering application. It also creates a foundation for the study of hydrogen fuel system architectures, which can be further developed to expand the system boundaries and conduct more extensive studies.

Keywords: system architecture optimization; hydrogen fuel systems; hydrogen aircraft design; multidisciplinary design optimization.

Table of Contents

List of Figures

List of Tables

Abbreviations

ADORE Architecture Design and Optimization Reasoning Environment

ADSG Architecture Design Space Graph

BOP Balance of Plant

CDS Central Data Schema

CFRP Carbon-Fibre-Reinforced Plastics

CPACS Common Parametric Aircraft Configuration Schema

DOE Design of Experiments

DSM Design Structure Matrix

FC Fuel Cell

GUI Graphical User Interface

INCOSE International Council on Systems Engineering

MBSE Model-Based Systems Engineering

MDA Multidisciplinary Analysis

MDAO Multidisciplinary Design Analysis and Optimization

MDAx MDAO Workflow Design Accelerator

MDF Multidisciplinary Feasible

MDO Multidisciplinary Design Optimization

MTOW Maximum Take-off Weight

NASA National Aeronautics and Space Administration

NFE Node Factory Evaluator

OAD Overall Aircraft Design

PEM Proton Exchange Membrane

PEMFC Proton Exchange Membrane Fuel Cell

PIDO Process Integration and Design Optimization

RCE Remote Component Environment

RFLP Requirements - Functional - Logical - Physical

SAF Sustainable Aviation Fuel

SAO System Architecture Optimization

SBO Surrogate-Based Optimization

SOFC Solid Oxide Fuel Cell

TET Turbine Entry Temperature

TLAR Top-Level Aircraft Requirement

XDSM Extended Design Structure Matrix

1 Introduction

Over the past several decades, climate change has become an increasingly pressing global concern, with anthropogenic emissions having a relevant impact on ecosystems, human health, and socio-economic systems (IPCC 2023). Among the most important contributors to global greenhouse gas emissions is the transportation sector, representing an estimated 15% of their atmospheric release (US EPA 2025). Within this realm, aviation emissions have been growing faster than rail, road or shipping, accounting for 2.5% of global energy-related CO_2 emissions (IEA 2025).

The wish to revert this trend has sparked a societal push towards sustainable aviation. *Flightpath 2050*, published by the European Commission, portrays the vision for the future of European aviation. It sets ambitious environmental protection goals which ought to be met by 2050. Among them are a 75% reduction in CO_2 emissions per passenger kilometer, a 90% reduction in NOx emissions and a 65% reduction in noise pollution (European Union 2011).

To achieve these challenging goals, significant efforts have been made over the past years to develop technologies that drive sustainability. The EXACT project, an internal project from the German Aerospace Center (DLR), is one of the largest European research projects on sustainable aviation (DLR 2025). The goal is to "identify how future aircraft should be designed, by which technologies they should be powered and how they should be operated to achieve the EU's ambitious climate goals".

An important part of the EXACT project consists of evaluating the feasibility of the use of hydrogen as an aviation fuel. The most common types of propulsion system for hydrogen aircraft are gas turbine engines and fuel cells (Wallington et al. 2025). Between the two, fuel-cell-based aircraft are less well-established and thus pose a higher level of uncertainty. This then warrants significant research efforts to better understand the intricacies of such configurations.

Fuel-cell-based aircraft seem to show the most promise in the regional, short-range sector. The reasons for this are numerous and related to the limited maximum power rating of modern electrical machines, the lower specific power of fuel cell configurations, and heat dissipation issues at the larger scales (Adler & Martins 2023). Thus, within EXACT, fuel-cell-based aircraft developments were limited to regional applications given their current infeasibility for longer flights.

In the context of overall aircraft design (OAD), significant differences arise in the design of a fuel cell aircraft in comparison to more conventional configurations. These are naturally related to the novel propulsion system and fuel type. Because of this, detailed analyses have been done on tank design and integration, as well as on the fuel cells themselves (DLR 2025). However, what is still lacking is more comprehensive information on the hydrogen fuel system, which acts as a connecting link between tank and fuel cell. The work presented in this thesis attempts to take initial steps in closing this knowledge gap and developing related capabilities at the DLR.

Typically in OAD, aircraft systems are accounted for on the basis of statistical data (Torenbeek 1982). Essentially, values for mass and power consumption, for example, are taken from previous aircraft designs as initial estimates deemed accurate enough for this early stage. The lack of experience with hydrogen technology makes this approach highly uncertain (Burschyk et al. 2024). In order to accurately evaluate the potential of hydrogen powered aircraft, it would be useful to integrate a more detailed, physics-based analysis of the aircraft fuel system into the conceptual design activities.

In early design stages, one of the most important activities is that of system architecting. It comprises the definition of the architecture of the system, which has a significant influence on the system performance (Bussemaker & Boggero 2022). This in fact applies to the design of a hydrogen fuel system. However, architecture design spaces expand drastically as a function of the number of architectural decisions (Iacobucci 2012). For complex systems, this makes it infeasible to exhaustively and manually probe for the best solutions (Bussemaker et al. 2020). The traditional workaround to this problem has been the filtering of select architectures for further development based on expert opinion (Roelofs & Vos 2018), exposing the design process to bias, subjectivity and conservatism, hindering disruptive innovation (Bussemaker et al. 2020).

System architecture optimization (SAO) (Bussemaker & Ciampa 2022) can be used to mitigate the aforementioned issues, by converting system architecting into a numerical optimization problem. Model-based systems engineering (MBSE) methods are used to generate candidate architectures and multidisciplinary design optimization (MDO) techniques are employed to evaluate the architectures and search for optimal configurations. Therefore, SAO constitutes a systematized, automated and thus unbiased method of design space exploration

(Bussemaker & Ciampa 2022).

Architecture optimization constitutes a very particular type of MDO problem. Succinctly, SAO problems can be described as *mixed-discrete*, *hierarchical*, *multi-objective*, *black-box* optimization problems (Bussemaker et al. 2024a). This poses significant constraints on the possible optimization algorithms that can be employed, essentially excluding gradient-based optimization.

Furthermore, the platforms used to formulate and execute these problems, called MDO platforms, have to face significant challenges. The main challenge is related to the fact that different architectures have different components with different connections between them. Because of this, design variables, constraints and even the disciplines involved may vary depending on the architecture. Thus, MDO platforms meant to be used in SAO problems should be capable of automatically readjusting the MDO problem during the optimization process, depending on the architecture being analyzed.

One platform capable of fulfilling all the requirements for use in SAO is the joint framework of MDAx (Garg et al. 2024a) and RCE (Boden et al. 2021). Recent adaptations to MDAx have allowed the framework to accommodate the automatic adjustment of the optimization problem, through the concept of dynamic MDAO (Garg et al. 2024b).

1.1 Thesis Structure and Objectives

Dynamic MDAO enables the execution of system architecture optimization studies. It consists of the set of capabilities that allow for the dynamic and automatic formulation of MDAO workflows to accommodate for different architectures within an optimization study (Garg et al. 2024b).

Recently, these capabilities have been implemented in the back-end of MDAx. Developed by DLR, MDAx is an application used for collaborative MDAO workflow modeling. It also enables the export of said workflows for operation in execution environments such as RCE (Garg et al. 2024a).

This master thesis aims to make use of these newly available capabilities to study the design space of a hydrogen fuel system for a fuel-cell-based regional airliner. Synergistically, it aims to demonstrate that the employed framework can in fact be used for realistic application cases, while also identifying potential room for improvement in the implementation.

Chapters 2 and 3 provide the necessary background on the two main topics of the work.

Chapter 2 presents the fundamentals of the relevant engineering design methods, starting with a general introduction to systems engineering and MDO. Then, SAO is discussed in detail, from its main characteristics to the requirements on MDO platforms for their use in such problems. The most relevant of these is the capacity for dynamic MDAO, a topic addressed at the end of the chapter. Chapter 3 relates to the other main topic, that of hydrogen aviation. Its structure progressively narrows the scope, starting from the implications of hydrogen on the overall aircraft design and ending with a detailed overview of the fuel system. Chapter 4 describes the methodology employed to implement the SAO problem. The employed framework is introduced and its application exemplified through a benchmark problem. The architecture design space is defined, including the establishment of the reference aircraft. Additionally, the disciplines used to design the various components of the fuel system are introduced. Furthermore, the optimization problem is formulated and the workflow is generated in MDAx. Finally, the complete optimization workflow is imported and implemented in RCE. Having conducted the design space exploration, the results are presented in Chapter 5. The influence of the various architectural decisions is discussed and the optimal architecture is presented. Additionally, different optimization strategies are compared, assessing the capacity to reach the global optimum and required computational effort. Lastly, the benefits of the use of an automated approach are presented. In Chapter 6, the outcomes of the study are summarized and the limitations of the current implementation are highlighted. Based on this, recommendations are made for further work.

In essence, this work pursues the following objectives:

- Study of optimal hydrogen fuel system architectures for a fuel-cell-based regional airliner, through the use of system architecture optimization.
- Demonstration of the dynamic MDAO capabilities implemented in MDAx in a realistic application case.
- Identification of shortcomings and potential for improvement in the current implementation of the Dynamic MDAO methodology.

2 Engineering Methods for Digital Development

The current chapter presents the engineering design methods which are central to the work of this thesis. First, the broader fields of systems engineering and multidisciplinary design optimization are introduced. Afterwards, focus is narrowed into system architecture optimization and the concept of dynamic MDAO, which enables the use of MDO platforms for the analysis and optimization of system architectures.

2.1 Systems Engineering and Multidisciplinary Design Optimization

The rapid technological developments witnessed over the 20th century have led to an exponential increase in the complexity of the technical systems developed in industries such as automotive, aerospace or microelectronics (Aerospace Industries Association 2016). Figure 2.1 illustrates this, in the case of software complexity in commercial aircraft.

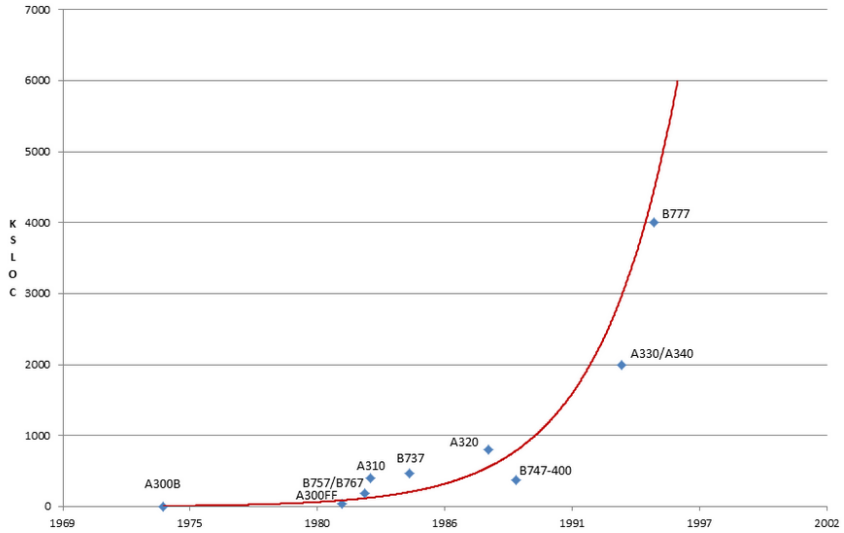


Figure 2.1: Growth of software complexity in commercial aircraft expressed in thousands of source lines of code (KSLOC) used in specific aircraft over time. Adapted from Aerospace Vehicle Systems Institute (2024).

This increase in complexity has been, particularly in the aerospace industry, accompanied by an equally steep increase in both development and unit costs. The United States Department of Defense's Joint Strike Fighter program is an example of this. The development and acquisition costs of the program's chosen aircraft, the Lockheed Martin F-35, are currently estimated to be at around $450 billion (Losey 2024). Each aircraft is said to have a unit price of more than $150 million (Capaccio 2017). In contrast, the Lockheed P-80 Shooting Star, a first generation fighter jet, is estimated to have had a unit cost of $110,000 in the mid 1940's (Aero Corner 2024, Living Warbirds 2024).

The level of convolution and the progressively higher cost of these highly advanced systems also means that their development has become associated with an elevated level of risk. In extreme cases, a failed development program can mean the bankruptcy of a company given the magnitude of the required investment.

Thus, with competitive markets and a small margin for error, it eventually became necessary to develop an organized and systematized approach to the product development process. One such approach is the discipline of Systems Engineering.

2.1.1 Systems Engineering

The National Aeronautics and Space Administration (NASA) defines systems engineering as a "methodical, multi-disciplinary approach for the design, realization, technical management, operations, and retirement of a system" (Hirshorn et al. 2017, p. 3). It represents an effective means to manage complexity throughout every phase of the product life cycle. Some of its primary goals are the reduction of risk associated with the development of novel systems, as well as the reduction of time to market (Haskins 2006).

The activities or processes associated with systems engineering can be categorized into four main groups (Hirshorn et al. 2017):

- **System design**: definition of requirements and technical solutions.
- **Technical management**: technical controlling such as configuration, interface and requirements management.
- **Product realization**: product implementation and evaluation through processes such as system integration, verification and validation.
- **Project management**: managing programmatic risk, the project team, costs and schedule.

If a system is developed using the aforementioned systems engineering methodologies, it is considered and *engineered system*. An engineered system is "designed or adapted to interact with an anticipated operational environment to achieve one or more intended purposes while complying with applicable constraints" (Sillitto et al. 2019).

In the design of such systems, decisions made in very early stages (namely conceptual design) are some of the most important and have a tremendous impact on the performance of the final product (Maier & Rechtin 2009). They also have a great influence on the total costs

of a program. As can be conferred in Figure 2.2, although the costs expended in the early phases are not particularly high when compared to later stages, by the end of preliminary design most of the life cycle costs will be allocated and committed. Design alterations then become more expensive as the project matures (Hirshorn et al. 2017). The use of systems engineering aims to facilitate the discovery of potential issues sooner, helping to prevent cost and schedule overruns (INCOSE 2023).

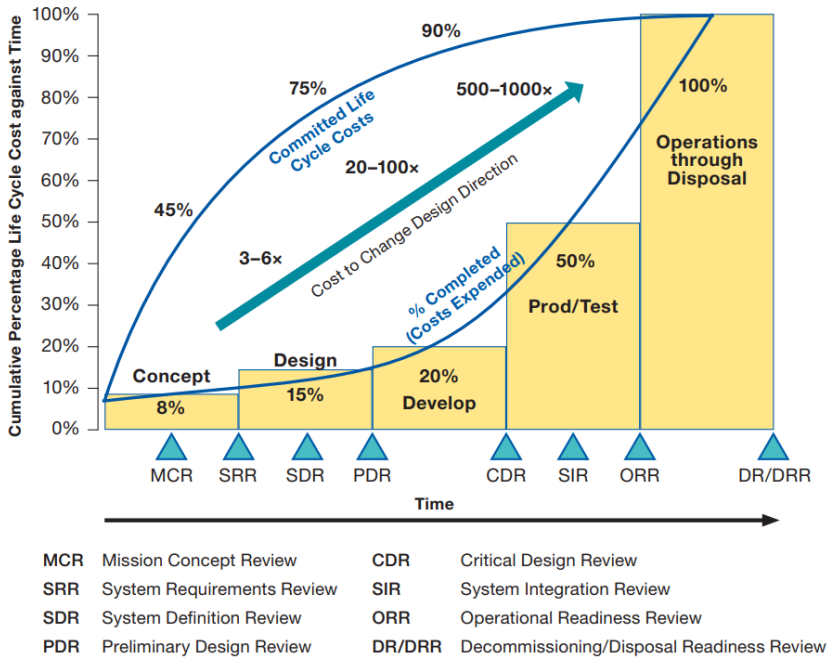


Figure 2.2: Life-Cycle cost impacts from early phase decision-making. Reproduced from Hirshorn et al. (2017).

Many of these key early decisions influence what is referred to as the architecture of the system. They must be made at a point where not much information exists on what the final product will actually look like and how it will perform. Despite this uncertainty, these choices should be thoroughly analyzed from a holistic viewpoint in order to try to reach the optimal architecture (Crawley et al. 2016, p 17-18). This constitutes the concept of system architecting.

2.1.2 System Architecting

System architecting is a part of the systems engineering process and consists of the set of activities that lead to the establishment of the system architecture. This includes the review of functional and performance requirements, with many trade studies being performed as to assess the best way to meet said requirements in a cost-effective manner (Hirshorn et al. 2017, p 24-25).

In turn, the system architecture itself can be defined as "the embodiment of concept, the allocation of physical/informational function to the elements of form, and the definition of relationships among the elements and with the surrounding context". In other words, it is the abstract description of the entities (components) of a system and the relationship between those entities (Crawley et al. 2016, p 124).

The definition of the architecture is done by decomposing the system into manageable elements and interfaces. This can be carried out in a number of different ways, of which the disciplinary, physical, perimeter-based, and functional decompositions are examples (Mavris et al. 2008). Figure 2.3 attempts to visually illustrate the difference between the approaches.

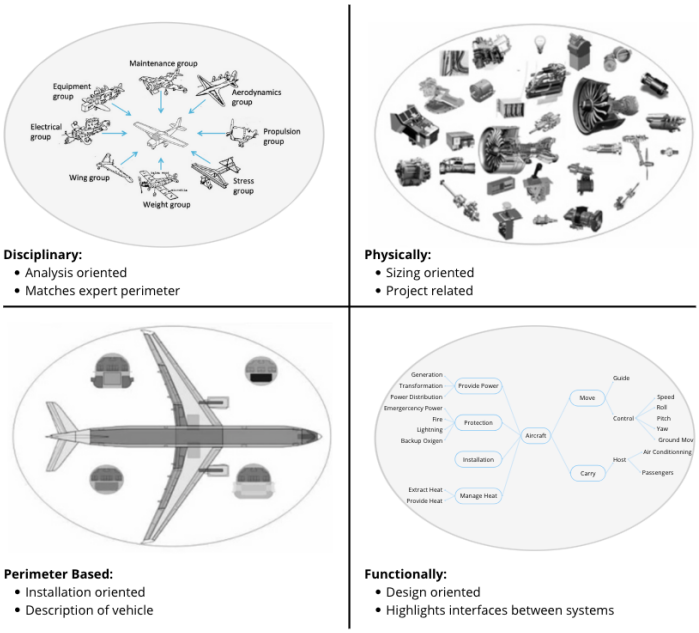


Figure 2.3: Approaches to architecture breakdown. Adapted from Mavris et al. (2008).

In functional decomposition, elements or components are grouped based on the functions they enable. This function-based breakdown is popular in aerospace because the functional requirements of aircraft and aircraft systems are much more stable throughout development than their physical counterparts, facilitating the comparison of architecture proposals (Mavris et al. 2008).

This type of function-based architecture breakdown and definition is generally carried out using the RFLP approach (Requirements - Functional - Logical - Physical) (Kleiner & Kramer 2013). In this method, one starts by uncovering the requirements on the system and defines functions that it must perform in order to fulfill those requirements. Next comes the attribution

of physical components capable of carrying out the functions. This is known as function-form mapping (Crawley et al. 2016). The decision on a type of solution, in the form of a component, can then induce a number of new required functions. In turn, these functions require new components to fulfill them. Once this cascade is resolved, the complete logical architecture is found.

Lastly, the physical architecture is established by defining how the elements of form (components) are interconnected, potentially through the exchange of mass, energy, information, etc. It builds on the logical architecture by providing the additional necessary information on the implementation of the final system (Bussemaker et al. 2020).

There are numerous ways of visually representing and documenting both the architecture design space and instantiated architecture alternatives. Something most have in common is that they rely on the use of models to portray this information. The application of models to facilitate the typical activities associated with the broader field of systems engineering is what constitutes Model-Based Systems Engineering (MBSE), presented next.

2.1.3 Model-Based Systems Engineering (MBSE)

Traditionally, systems engineering has been document-centric, meaning strongly reliant on documentation as the primary method for storage and tracking of official decisions and product information. Although models have been used for many decades, they have historically been seen as a secondary means of technical communication. Currently, a transition in favor of model-based processes is underway, with the aerospace industry at its forefront (Aerospace Industries Association 2016).

The International Council on Systems Engineering (INCOSE) defines model-based systems engineering (MBSE) as "the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases" (INCOSE 2007).

At the foundation of MBSE are a few essential governing principles (Madni et al. 2023):

- It represents a unified, interdisciplinary model that covers the entire system life cycle, constituting a single source of truth and primary means of technical communication.
- Configuration management is integrated into the models, with the objective of mitigating maintenance issues.

- Methodology-neutrality is an intrinsic characteristic, allowing for the application of different modeling constructs based on complexity and the required level of detail.

Many types of models are used, with different levels of abstraction and fidelity. Some examples are functional flow diagrams, object-oriented models and block diagrams, among many others. The choice of which to use depends on the application. Associated with these models are different modeling languages and architecture frameworks, which allow for the description of relationships and dependencies between elements. When composed effectively, models allow for higher autonomy, precision and accuracy when compared to their document-based counterparts (Aerospace Industries Association 2016).

Ultimately, MBSE is used with the intent of reducing the risk, time-to-market and cost of a development program (Madni et al. 2023). This is achieved by means of, but not limited to, more accurate and stable requirements generation, physics-based modeling and earlier verification and validation, leading to earlier fault detection (Aerospace Industries Association 2016).

The employment of these approaches in aeronautics is especially well suited. In aircraft development programs, the early stages are characterized by high design space dimensionality, the evaluation of numerous competing conceptual designs and high volatility in requirements. Thus, the increased transparency, efficiency and traceability of design decisions attainable through model-based processes is of significant value (Ciampa et al. 2020).

Whereas MBSE spans the entire product life cycle, Multidisciplinary Design Optimization (MDO) is another technology whose popularity has steadily grown. It is increasingly being integrated and implemented specifically in the product design phases (conceptual to detailed) to reduce development time and achieve optimal technical solutions. Analogously, MDO has its roots in aerospace research and is particularly suited for such applications, given the notable interdisciplinarity and dependency between disciplines. Said interrelations cannot be neglected if one aims to achieve an overall optimized total system.

2.1.4 Multidisciplinary Design Optimization (MDO)

The conventional design process used in industry is based on manual iteration between design modification and evaluation. Even when optimization is used, it is commonly carried out in a sequential manner, with each discipline being optimized independently, one after the other. In many cases, this can lead to suboptimal results. The solution to this problem is the main

motivation for the use of multidisciplinary design optimization (MDO).

MDO consists essentially of the concurrent optimization of the system as a whole, made possible by considering all necessary interactions between disciplines through coupled models. This then allows for reaching a globally optimal design. Its application in the design process has the potential to increase system performance, decrease design time, reduce total cost and reduce uncertainty sooner, as illustrated in Figure 2.4 (Martins & Ning 2021).

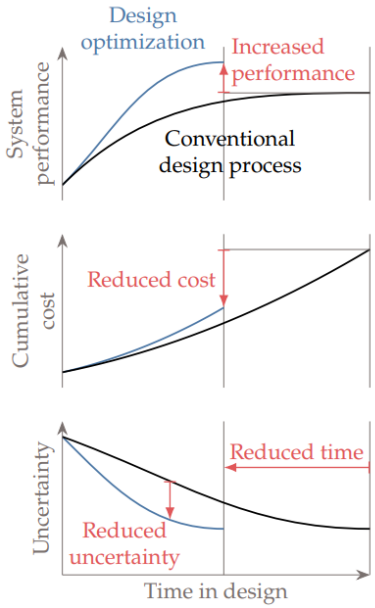


Figure 2.4: Advantages of the use of MDO. Reproduced from Martins & Ning (2021).

The same MDO problem can be formulated, and thus solved, in a variety of different ways. The combination of problem formulation and organization of both the analysis models and the optimization method is called an MDO architecture. This architecture can be of a monolithic or distributed nature. Monolithic architectures solve a single optimization problem, whereas distributed approaches split the same problem into multiple smaller ones, containing only part of the total design vector and constraints. The choice of most appropriate architecture for a given situation is important to reduce solution time. Especially when dealing with complex workflows involving many disciplines and variables. A comprehensive survey of MDO architectures can be found in Martins & Lambe (2013).

To represent the coupling of multidisciplinary systems through their data dependencies, a design structure matrix (DSM) or directed graphs can be used. One kind of graph exists, however, that was developed specifically for the visual representation of the solution process

of MDO architectures. It's called the eXtended Design Structure Matrix (XDSM) and its main advantage is that it shows process flows in addition to data flows (Lambe & Martins 2012).

Figure 2.5 shows the XDSM of a standard multidisciplinary feasible (MDF) architecture, which is one of the most common types of monolithic architectures. A multidisciplinary analysis (MDA) is performed, in this case with 3 coupled disciplines, where the values of the coupling variables are iteratively computed until consistency is achieved across all models. Afterwards, the objective function(s) and potential constraints are computed and fed back to the optimization driver, which checks for convergence and feasibility, and suggests a new design vector to be analyzed. This loop continues until an optimal solution is found, defined by the optimizer's convergence settings.

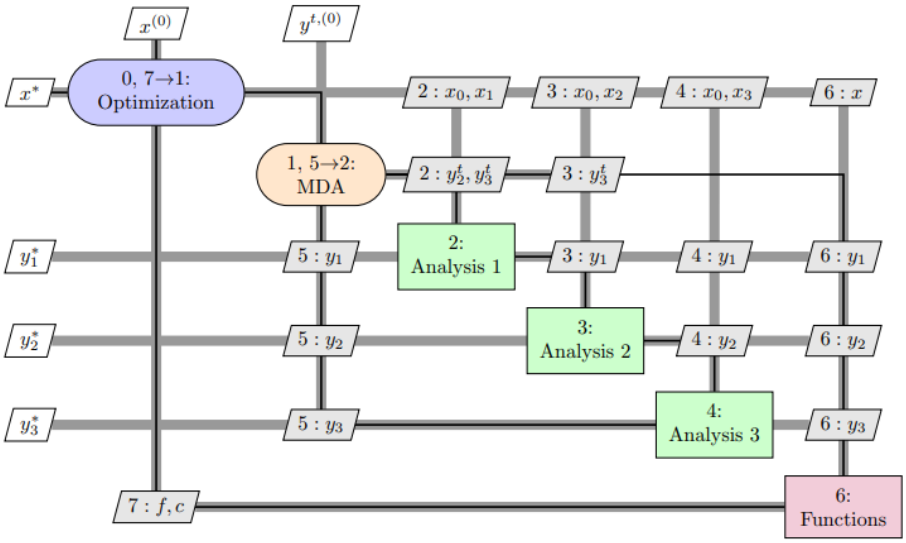


Figure 2.5: Example XDSM of an MDO problem formulation in the form of a multidisciplinary feasible (MDF) architecture. Reproduced from Lambe & Martins (2012).

MDO has the potential to allow for higher-fidelity analysis in earlier stages of design (Liu et al. 2024), making it feasible to assess a higher number of possible configurations in more detail before settling on a concept to pursue further. Its multidisciplinary nature, however, contrasts with the typical working method of the aeronautical industry, where work is divided by areas of expertise. This and the lack of an established framework for collaborative design may explain, at least in part, the relatively slow introduction of MDO practices in industrial application.

Collaborative MDO

There are a number of technical as well as non-technical challenges that currently prevent large-scale MDO from being the norm in industrial application. Ciampa & Nagel (2020) enumerate the main challenges, some of which include:

- **Many languages**: different disciplines use different nomenclature, making the assurance of consistency among models an extremely cumbersome activity.
- **Workflow integration and alteration**: the design of complex products requires large workflows with many data dependencies. The setup of the MDO architecture, in addition to the assurance of data availability and compatibility between all models, is a time-consuming task. Moreover, design iteration typically leads to the need for reconfiguration of the workflow, significantly increasing the workload associated with process generation and maintenance.
- **Inter-organizational collaboration**: in some cases, different companies might contribute to the same problem with models from their area of competence. Thus, data accessibility becomes a concern, as proprietary information should be protected.
- **Mindset**: most engineers working on a project will not be experts in MDO, instilling a certain inertia in accepting different working methodologies.

Collaborative MDO aims to combat the aforementioned challenges and thus break the typical barriers that exist between different disciplinary departments within an organization or even among partner organizations. Figure 2.6 illustrates the current state-of-the-art in collaborative design environments for the case of Overall Aircraft Design (OAD). The main tenet lies on the distributed contribution from the various specialist competences to a design environment where the total system is developed and optimized. This design environment is called a Process Integration and Design Optimization (PIDO) environment.

This type of framework makes use of decomposition methods such as Concurrent Engineering to ensure that the distribution among specialists is not limited to the respective analysis capabilities, but also encompasses the design task itself (Ciampa & Nagel 2016). To enable this, a Central Data Schema (CDS) is used. A CDS is a data exchange format based on common semantics which ensures that all stakeholders speak the same language while simultaneously reducing the amount of possible data interfaces. The most well established CDS in aircraft design is the Common Parametric Aircraft Configuration Schema (CPACS) (Alder et al. 2020).

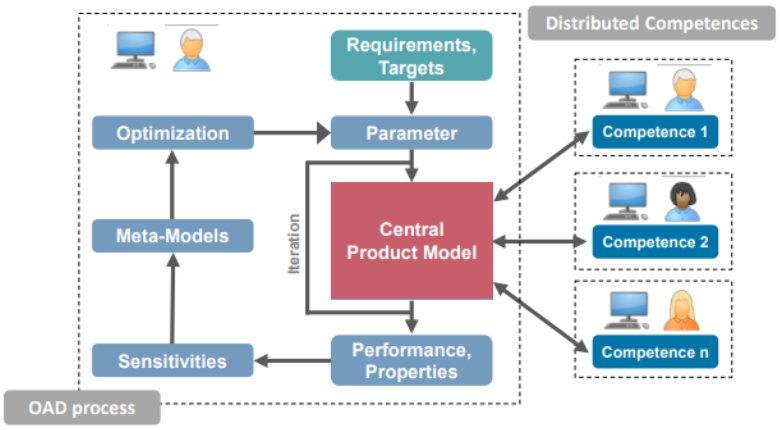


Figure 2.6: Third generation collaborative MDO environment applied to OAD. In comparison to its predecessors, this generation does not only represent the different competences through analysis models, but by distributing design authority, thus enabling concurrent engineering. Reproduced from Ciampa & Nagel (2016).

The challenge of inter-organizational collaboration is addressed by ensuring that the used PIDO environment is able to remotely execute workflow tools with the respective authorization. It thus restricts access to the said tools from other stakeholders. Furthermore, the mindset problem can be combated through basic training and a clear work distribution. This includes the assignment of a workflow responsible, passing to the disciplinary expert only the responsibility for their own analysis competence.

Collaborative MDO therefore has the potential not only to allow for an optimized solution in which interdisciplinary dependencies are considered, but also to significantly accelerate the design process (Ciampa & Nagel 2016).

2.2 System Architecture Optimization: Bridging the Gap Between MBSE and MDO

As defined by Ciampa & Nagel (2021), a modern complex system development framework, illustrated in Figure 2.7, can be divided into an upstream architecting phase, associated with systems engineering and specifically MBSE approaches, and a downstream product design phase, executed in a multidisciplinary design analysis and optimization (MDAO) context.

In the upstream phase, stakeholder needs and system capabilities are defined and translated into requirements, culminating in the outlining of the system architecture. The downstream phase consists of design space exploration for a given architecture and selection of the optimal solution for a product design that satisfies the requirements.

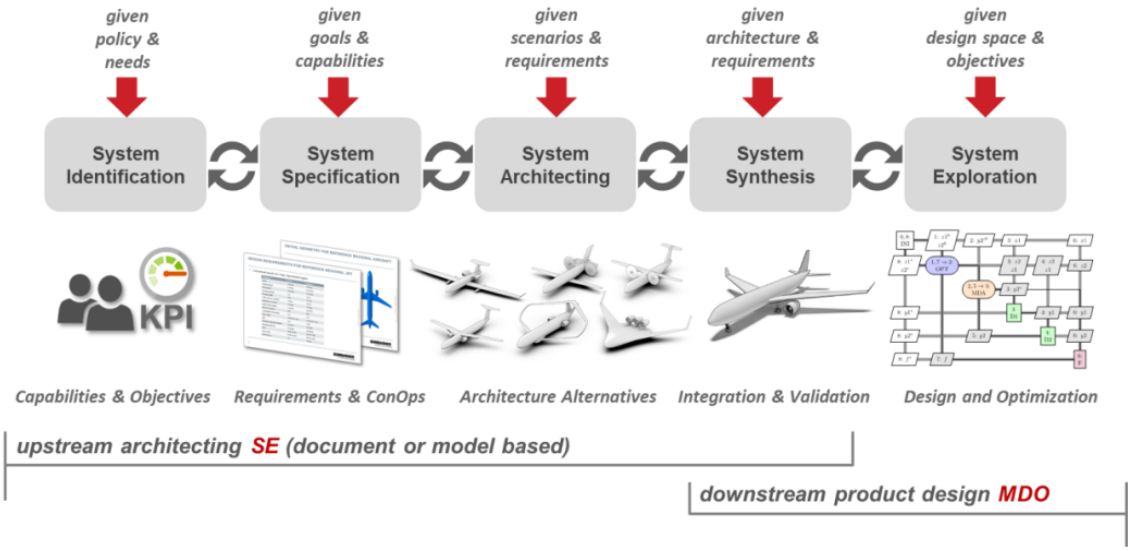


Figure 2.7: Complex System Development Framework. The different stages are showcased, as well as the relation to the main methodologies used. Reproduced from Ciampa & Nagel (2021).

The execution of the phases depicted in Figure 2.7 in a unidirectional and sequential form seriously hinders the capacity for design iteration and exploration. If MDO activities are only performed once an architecture is defined and frozen, the possible trade-offs from then on become much more limited (Ciampa et al. 2020). Thus, it is essential for a coupling to exist between the MBSE and MDO phases, enabling coherence between the two.

One of the ways in which this coupling takes form is in the relation between the system architecture, more specifically the constituent components, and the MDAO process (Ciampa & Nagel 2021). This bridge allows for verification on the fulfillment of requirements for designed solutions, enabling feedback between the architecting and design optimization steps. Therefore, the exploration of architecture design spaces is made possible (Bussemaker, Boggero & Ciampa 2022).

Architecture design spaces expand drastically as a function of the number of architectural decisions (Iacobucci 2012). For complex systems, this makes it infeasible to exhaustively and manually probe for the best solutions (Bussemaker et al. 2020). The traditional workaround to this problem has been the filtering of select architectures for further development based on expert opinion (Roelofs & Vos 2018), exposing the design process to bias, subjectivity and conservatism, hindering disruptive innovation (Bussemaker et al. 2020).

By converting system architecting into a numerical optimization problem and integrating

physics-based evaluation, system architecture optimization (SAO) mitigates the aforementioned issues by providing a systematized, automated and thus unbiased method of design space exploration (Bussemaker & Ciampa 2022).

The concept of system architecture optimization is depicted in Figure 2.8. Initially, the design space is formalized and modeled in an MBSE environment through the mapping of function to form (components) and identification of architectural choices. Once it is fully defined, architecture instances are automatically generated by an exploration algorithm and quantitatively evaluated, typically using MDAO techniques and employing physics-based models. Then, the performance metrics resulting from the evaluation can be used to compare architectures and make an informed decision on the most suitable solution (Bussemaker, Garg & Boggero 2022).

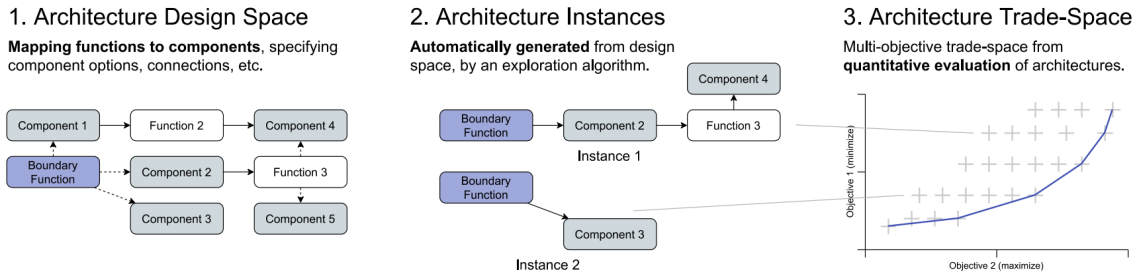


Figure 2.8: Systematic architecture design space exploration concept: Architecture instances are automatically generated from a previously modeled and formalized architecture design space and quantitatively evaluated through multidisciplinary analysis methods, the results of which are used to discern optimal architectures with respect to objective functions. Reproduced from Bussemaker, Garg & Boggero (2022).

In the design of complex systems, many objectives may be taken into account, that are often in conflict with each other. For example, maximization of performance and minimization of cost are habitually in opposition. In these cases, a single optimal architecture cannot be obtained, but rather a set of optimized solutions emerges called the Pareto set. The evaluation of this set results in a Pareto front, represented by the blue line in Figure 2.8. The Pareto set constitutes the set of all *non-dominated* designs. A design is said to be *non-dominated* if there are no other designs among those evaluated that are superior to it with respect to all objectives (Martins & Ning 2021). These points are said to be Pareto optimal and it is then up to the engineer to decide on the best trade-off between objectives, defining exactly where along the Pareto front the chosen solution will lie.

The two main elements of an SAO problem have already been implicitly mentioned, but shall be formally defined as (Bussemaker et al. 2020):

1. An **architecture generator**, suggesting architecture instances to be evaluated.
2. An **architecture evaluator**, evaluating the performance of a given architecture instance.

The following sections will go into further detail on these two fundamental pillars of SAO.

2.2.1 Architecture Generation

The architecture generator has the task of generating architecture instances from an architecture design space. This design space models architectural choices and constraints. It is then through the resolution of these choices that architecture instantiation is carried out.

Architectural choices, or decisions, are high-level design decisions that lead to different system architectures (Crawley et al. 2016). These decisions are not all identical in nature. For example, the choice on the type of engine for an aircraft is a different type of decision than the number of engines. While in the first case one selects an alternative over another, the second choice affects the number of component instances. Selva et al. (2016) propose a set of six different patterns that can be abstracted from architectural decisions. Each decision is thus expressed through one or a combination of these patterns. For the sake of succinctness, these patterns will not be discussed further and the interested reader is referred to the aforementioned literature.

The architecture design space should be sufficiently formalized in order to enable automatic computer processing, while simultaneously allowing its definition in typical system architecting terminology, such as function-to-form mapping (Menshenin et al. 2020). The architecture design space graph (ADSG), one of the existing methods for design space modeling, satisfies these two requirements. It can be encoded as design variables and is compatible with the use of optimization algorithms, which allows users to model the architecture design space using common MBSE processes while also enabling systematic exploration of the architecture design space (Bussemaker et al. 2024a).

Architecture Design Space Graph (ADSG)

The ADSG (Bussemaker et al. 2020) is a directed graph used to model the architecture design space, generate architecture alternatives and formulate optimization problems (Bussemaker & Ciampa 2022). It is based on the functional decomposition approach, as defined by Mavris

et al. (2008), and establishes all possible components of the system architecture and how they are interconnected (Bussemaker et al. 2020).

The graph consists mainly of nodes and edges. Nodes represent elements of the architecture, like functions, components or ports. Edges, on the other hand, are directed from one node to another and can denote either a connection or a derivation. A comprehensive list of all node types in the ADSG can be found in Bussemaker et al. (2020).

The method lies on the basis that design choices in system architecting are mainly of three types:

- **Function fulfillment**: closely related to functional decomposition, establishing a derivational relationship between architecture elements. Starting with one or more boundary functions, which act at the system boundary and provide the main value to the system, components fulfill functions and induce or derive new ones. When a function can be performed by more than one possible component, the selection of one of them constitutes a function fulfillment decision. Figure 2.9 shows an example.
- **Component Characterization**: component-level choices that provide additional information, exemplified in Figure 2.10. The required amount of a certain component can vary depending on the architecture. For that reason, the number of instances is an architectural decision. In addition, components can be assigned attributes (another type of node), and the selection of values for these attributes constitutes a design decision.
- **Component connections**: interactions between components are modeled through ports, and decisions are related to the mapping of sources to targets. Connections can represent physical links, information or energy flows, among other things.

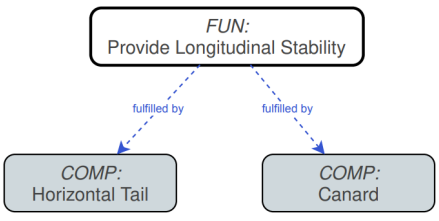


Figure 2.9: Example of a function fulfillment architectural decision. Two different configuration options are given for the provision of longitudinal stability of an aircraft: a horizontal tail and a canard. To instantiate an architecture, a decision between the two must be made. Reproduced from García Sánchez (2024).

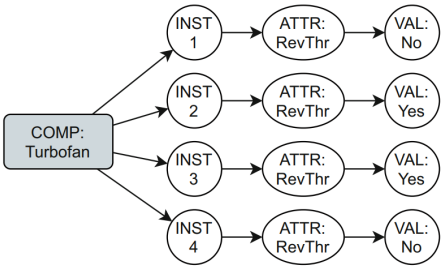


Figure 2.10: Example of component characterization decisions. There are four instances of a turbofan engine, where two are capable of thrust reversal. Each instance has associated to it an attribute node which points to its value node. Reproduced from Bussemaker & Ciampa (2022).

Apart from the ones already mentioned, a number of other nodes exist that help manage complexity. For example, concept nodes make possible the mapping of solution-neutral to solution-specific functions. Decomposition nodes are used to break down a function into a number of sub-functions. Non-fulfillment nodes allow for the conditional removal of a function from the architecture, whereas multi-fulfillment nodes can associate multiple components to one function (Bussemaker et al. 2024a).

Performance metrics can also be defined, representing important indicators through which different architectures can be compared. They can quantify the performance of the total system or be associated with specific functions or components. The inclusion of performance metrics is determinant to the applicability of the ADSG to optimization problems, as they can be construed as objectives or constraints (Bussemaker et al. 2020).

Through this formalized method, an architectural design space can be generated and represented in computational environments. One software tool capable of this is the Architecture Design and Optimization Reasoning Environment (ADORE) (Bussemaker et al. 2024a). ADORE is defined as "a platform to support the implementation of System Architecture Optimization (SAO) problems. To do so, it provides an architecture design space modeling GUI, various interfaces for connecting to evaluation code, and interfaces for connecting to optimization algorithms" (Bussemaker 2024).

2.2.2 Architecture Evaluation

Apart from the formalized representation of the design space and automatic creation of architectures, the other big enabler of system architecture optimization is the automatic quantitative evaluation of proposed designs. This quantitative information can be fed back to an optimizer

in the form of performance metrics, such that architectures can be compared and their feasibility (violation of constraints) assessed.

Figure 2.11 illustrates the functioning of an SAO loop. The architecture evaluation step is typically performed in a multidisciplinary analysis (MDA) context, which in turn makes the total loop a multidisciplinary design optimization problem. However, architecture optimization constitutes a very particular type of MDO problem, the singular challenges of which are the topic of the next section.

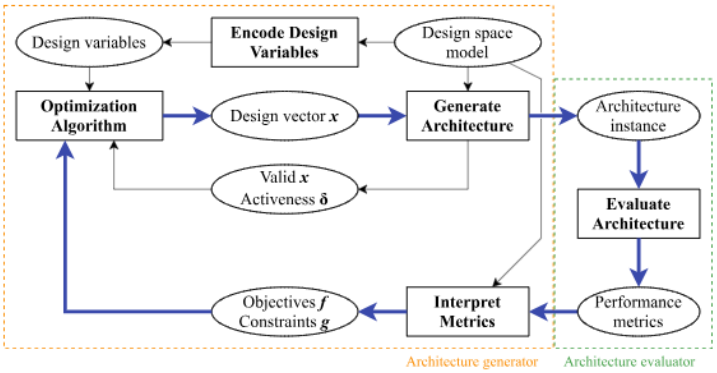


Figure 2.11: The SAO loop. An optimizer proposes a design to the architecture generator and receives feedback on the validity of said architecture. If an architecture is valid, it gets evaluated and performance metrics are extracted and fed back to the optimization algorithm. Reproduced from Bussemaker (2024).

Characteristics of System Architecture Optimization Problems

Whereas component design variables are typically continuous (e.g.: wingspan, sweep, aspect ratio, etc.), architectural decisions are encoded as discrete design variables (Bussemaker et al. 2020), meaning that they only take certain specific values. So, it is a characteristic of such optimization problems to involve a combination of both types. The set of all variables is then called a set of mixed-discrete variables.

Since architectural decisions are hierarchical in nature, as in they may or may not exist depending on an upstream, higher-level decision, the same is the case for their corresponding encoded design variables. These variables, however, always exist in the design vector, requiring a method to deal with them when not active. One such method is imputation, where design variables are assigned a predefined value when inactive (Zaefferer & Horn 2018). This hierarchical structure is the reason for the need of a feedback between the architecture generator and the optimization algorithm, as seen in Figure 2.11. When the optimizer suggests

an infeasible architecture, the inactive variables are set to their imputation value, creating the corresponding feasible architecture.

It is also common in architecture optimization problems that more than one objective is defined, often reflecting conflicting stakeholder needs. It is thus considered a multi-objective problem.

Additionally, the evaluation framework may contain physics-based analysis models, the solution of which is based on numerical simulation techniques. These act like black box functions, where only the inputs and outputs are known, with no information available on the internal functioning. This subjects the optimization problem to hidden constraints, which are constrains unknown to the optimizer (Müller & Day 2019) and that can compromise the validity of an analysis or simulation (Bussemaker et al. 2021).

In summary, it can be concluded that, in general, system architecture optimization problems are *mixed-discrete*, *hierarchical*, *multi-objective*, *black-box* optimization problems. This poses significant constraints on the possible optimization algorithms that can be employed, since they must be able to deal with all these features.

In problems like this, gradient-based optimizers are not an option since they cannot handle discrete variables. They also assume that the functions being evaluated are continuous, which is what allows gradient computation. The usage of gradient-free optimization is in fact a necessity, since such algorithms do not assume function continuity, can handle discrete variables and are better suited for multi-objective optimization (Martins & Ning 2021). Evolutionary algorithms are among the most prominent gradient-free algorithms, an example of which is the Nondominated Sorting Genetic Algorithm II (NSGA-II) (Deb et al. 2002).

Gradient-free algorithms do possess a significant disadvantage when compared to gradient-based algorithms, which is their far inferior efficiency, requiring a large number of function evaluations (Martins & Ning 2021). Figure 2.12 shows the number of function evaluations required for the optimization of the n-dimensional Rosenbrock function for an increasing number of design variables. Although its a specific example, it illustrates the general disparity between the two approaches. As can also be inferred from the graph, this inefficiency of gradient-free algorithms is particularly problematic for a large number of design variables.

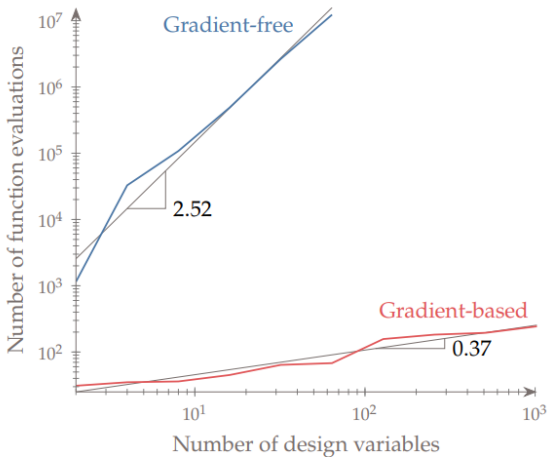


Figure 2.12: Number of function evaluations for an increasing number of design variables in the optimization of the *n*-dimensional Rosenbrock function, comparing a gradient-based and a gradient-free algorithm. The gradient-based algorithm is much better suited for large design problems. Reproduced from Martins & Ning (2021).

To make large system architecture optimization problems feasible, surrogate-based optimization (SBO) can be used, significantly reducing the number of function evaluations needed. In surrogate-based optimization, a surrogate model of the expensive evaluation framework is created and used to search the design space (Bussemaker et al. 2021).

2.2.3 MDO Platforms for System Architecture Optimization

Having presented a comprehensive overview of the SAO process, what is now left on this matter is an outline of the current state of the art in the computational implementation of such problems. García Sánchez (2024) performs a relatively detailed analysis and comparison of different MDO platforms in the aerospace industry and evaluates their suitability for use as architecture evaluators.

One of the mentioned frameworks is OpenMDAO (Gray et al. 2019), developed as a collaboration between the MDO Lab of the University of Michigan and NASA. Its main drawbacks are stated to be: the lack of a graphical user interface (GUI), increasing the time and effort necessary for the formulation of an MDO problem; the lack of flexibility when adapting the problem, such as addition of disciplines or alteration of connections between disciplines; and the inability to automatically adjust the problem depending on the system architecture to be analyzed. This last topic will be further addressed in Section 2.3 as a part of "Dynamic MDAO".

Another platform worthy of naming is GEMSEO (Gallard et al. 2018), developed by the MDO

Competence Center of IRT Saint Exupéry. It is stated to have many similarities with OpenMDAO, sharing the same advantages and drawbacks.

WhatsOpt (Lafage et al. 2019), a web application developed by ONERA, is said to be able to mitigate some of the issues of OpenMDAO or GEMSEO when used in conjunction with them. Described as a GUI for the formulation of MDO problems, it decreases the time, effort and expertise required to create such problems.

Finally, KADMOS (van Gent & La Rocca 2019) is a graph-based MDO problem formulation platform developed at the TU Delft. It does not execute the problem itself, but the exported formulation can be executed in other platforms such as OpenMDAO and RCE (presented next). García Sánchez (2024) considers it to have many of the characteristics necessary for use as an architecture evaluator, such as dealing with mixed-discrete variables, facilitating collaborative MDO through the inclusion of a GUI and the ability to automatically adjust the problem based on the proposed architecture. However, the GUI is said to lack usability due to the platform's focus on a methodological use of a graph-theoretic workflow representation. This highly formal and mathematical representation is also said to involve a large exposure to technical terminology and a lack of flexibility in workflow modeling (Page Risueño et al. 2020).

Attention is now routed to the framework most relevant to the current work: RCE and MDAx.

RCE and MDAx

Developed at the German Aerospace Center (DLR), Remote Component Environment (RCE) is an open-source application that enables the setup and execution of MDA and MDO problems (Boden et al. 2021). Disciplinary tools are integrated as standalone components and viewed as black boxes, with defined inputs and outputs. Tool integration is made easy by an intuitive GUI and a graphical wizard that guides the user.

Components can also be executed in a distributed manner, with different tools of the same workflow located in different computers, promoting collaboration between different experts and even between different institutions through the protection of intellectual property.

While very useful as an execution platform, the formulation and, especially, alteration of large MDO problems can still be quite cumbersome. This is why it is best used in conjunction with platforms specifically designed for the creation of such problems, like KADMOS or MDAx (García Sánchez 2024).

The MDAO Workflow Design Accelerator (MDAx) is an application developed by DLR for collaborative MDAO workflow modeling that enables the creation, inspection and exploration of components and their relationships. It also enables the export of workflows for execution in Process Integration and Design Optimization (PIDO) environments (Garg et al. 2024a).

At the foundation of MDAx is the graph-based approach to formulation of MDO problems created by Pate et al. (2014) and later developed by van Gent (2019). This approach has the advantage that the problem does not need to be fully defined beforehand, allowing for an iterative exploration in search of feasible problem architectures.

The usage of MDAx is made simple through a GUI (shown in Figure 2.13), where the problem is formulated in the form of an XDSM. Users can easily drag and connect different components, making it simple and fast to create and modify existing XDSM's with no coding necessary. This is considered to be the main advantage of MDAx in comparison to other similar platforms (García Sánchez 2024).

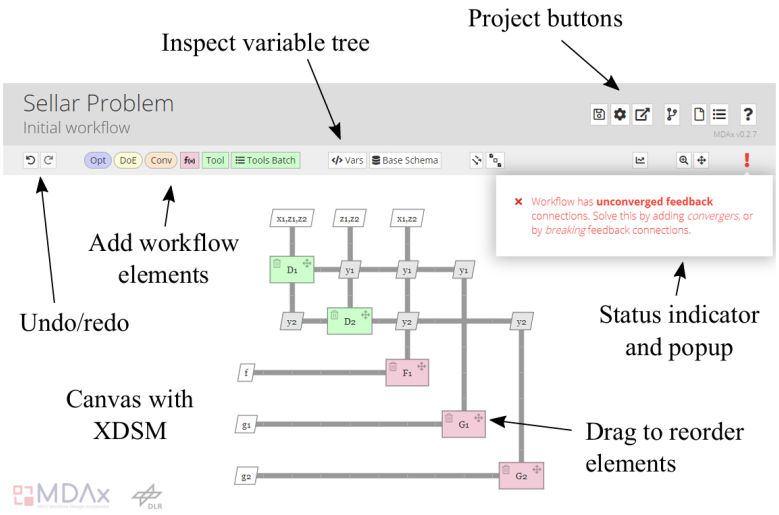


Figure 2.13: MDAx GUI. The XDSM in the canvas is the object of the work. The toolbar offers many features, such as the adding of components (tools, optimizers, convergers, etc.) and the inspection and definition of the variable tree based on the central data schema. Reproduced from Page Risueño et al. (2020).

To formulate an MDO problem in MDAx, one first includes the components, also denominated as blocks, in the XDSM. Subsequently, the inputs and outputs of said components are defined. For this, MDAx operates on the basis of a central data schema (CDS), which serves as a universal language spoken by all the tools. This allows for the automatic mapping of output variables to input variables of the same name, between different components.

The used central data schema can be of any origin, provided that it is XML-based. This schema then defines the structure of the variable tree, as seen in Figure 2.14. This tree contains the variables involved in the problem and displays them as a list where indentation is used to denote hierarchy. The definition of the CDS can be done manually in the variable tree, by creating the variables and child variables one by one, or automatically through the upload of an XML file which follows the desired schema.

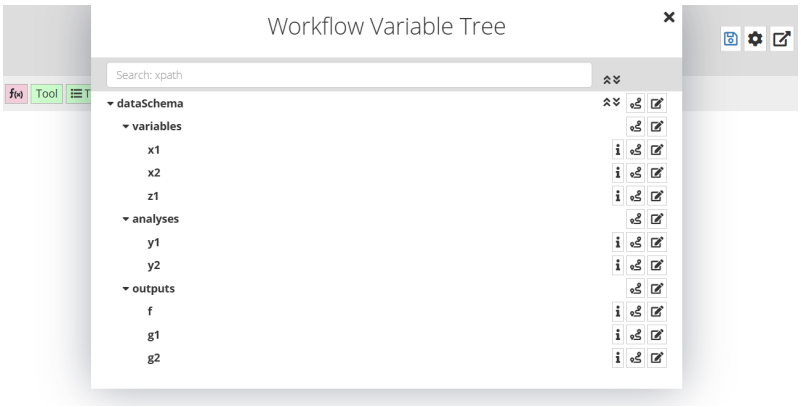


Figure 2.14: Popup of the workflow variable tree in MDAx. Indented nested levels can be collapsed to help display large trees. Adapted from Page Risueño et al. (2020).

Analogously, tool inputs and outputs can be defined manually variable by variable, or individual input/output XML files can be uploaded which abide by the data schema and contain the relevant variables. For further information on the XML markup language and file format, the reader is referred to the corresponding available literature.

Although any XML-based data schema can be used, the use of the Common Parametric Aircraft Configuration Schema (CPACS) is the standard in aircraft design research. CPACS is a central data exchange format that uses XML as its modeling language, developed by DLR specifically for aircraft design activities. It is easy to read and interpret by humans, and just like any other central data format, it drastically reduces the number of possible interconnections between stakeholders (tools in an MDO workflow) (Alder et al. 2020), as illustrated in Figure 2.15.

Throughout the process of creating a workflow model, the status indicator, depicted in Figure 2.13, alerts to issues in the workflow. The two main sources of issues are collisions and unconverged feedback. A collision arises, for example, when multiple tools output the same variable, which in turn is an input to another tool, generating ambiguity in the data coupling between components (Page Risueño et al. 2020). Feedback, on the other hand, occurs when

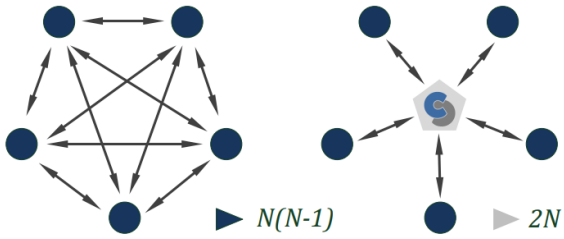


Figure 2.15: The possible amount of data interfaces reduces form N(N - 1) to 2N by using a central data format such as CPACS. Reproduced from Alder et al. (2020).

a tool requires as input an output of a tool that is downstream in the workflow. In some cases, this can be resolved by applying methods of tool rearrangement to remove feedback. When feedback is unavoidable, it must be dealt with through the addition of a converger. The various methods offered by MDAx for the resolution of collisions and feedback are presented in detail by Page Risueño et al. (2020).

When a valid and executable workflow is created, it can then be exported to a PIDO environment such as RCE. The export requires almost no further work in RCE for execution, except for the integration of the tools to be executed (if used for the first time) and the definition of the files to be used as workflow input. An example export of a one-tool workflow is shown in Figure 2.16.

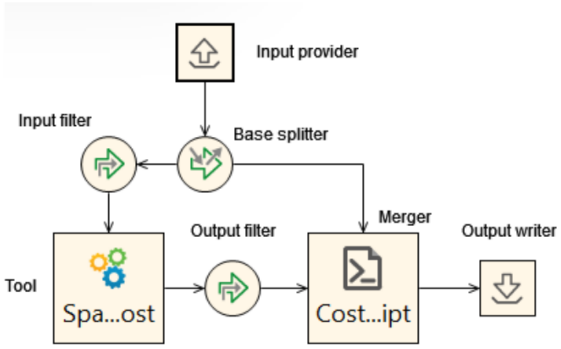


Figure 2.16: Example workflow created in MDAx and exported to RCE. The base splitter generates two copies of the input XML file. The input filter removes anything that is not an input to the tool. The output filter extracts the necessary outputs after executing the tool. The outputs are added to the original file in the merger block, and the results are then stored in the output writer. Reproduced from García Sánchez (2024).

In recent years, efforts have been made to adapt MDAx such that the MDAx + RCE framework is able to fulfill the last remaining open requirement for use as an architecture evaluator in SAO problems compatible with collaborative MDO: automatic readjustment of the MDO problem

during the optimization process, depending on the architecture being analyzed. This new functionality is called Dynamic MDAO and is the topic of the next section.

2.3 Dynamic MDAO

Within the design space of a system architecture, very different architectures can exist, possibly being composed of components with completely distinct working principles and thus inducing different interactions between the system constituents. The particular architecture to be analyzed has influence on the design variables, disciplines and data connections present in the MDAO problem formulation (Garg et al. 2024b). For a relatively small design space of ten architectures, for example, it already becomes impractical to manually formulate different problems for each candidate solution. This calls for the automation of the process, which is the essence of *dynamic MDAO*, a concept introduced by Garg et al. (2024b).

Dynamic MDAO consists of the set of capabilities that allow for the dynamic and automatic formulation of MDAO workflows to accommodate for different architectures within an optimization loop.

In order for an MDO platform to have the capability for dynamic MDAO, it must be able to accommodate a few different factors that affect the formulation and execution of the problem called *architectural influences*.

2.3.1 Architectural Influences

Analyzing different system architectures requires modifications to the MDAO problem formulation and execution on the basis of those architectures. Each modification is induced by one or more of the overarching factors termed architectural influences.

García Sánchez (2024) identifies a total of four architectural influences, indicated next. This conclusion is reached by analyzing a sample of MDAO problems from literature and using the work of Bussemaker, Garg & Boggero (2022) as a foundation.

Conditional Variables

The design variables that make up an MDO problem can vary depending on the architecture to be evaluated. This is because different components are defined by different variables and each system architecture comprises different components. Any variable whose existence in the problem is dependent on an architectural decision (not always present independently of the architecture) is denominated a conditional variable.

The existence of conditional variables implies alteration of the set of inputs and outputs of design disciplines. Therefore, these should be readjusted automatically for each architecture. Other than simple variables, more complex data structures, such as arrays, can also be used in these problems as a means of grouping information. The information carried may also vary depending on the architecture, causing an array, for example, to not always have the same length. Thus, MDO platforms should also be flexible in dealing with structures such as variable-length arrays.

As an example from aircraft design, the choice between a simple and a double tapered wing could be an architectural decision. If a double tapered wing is chosen, additional variables are needed to fully describe the wing geometry, such as the kink position (y_k) and the local chord at kink (c_k), as shown in Figure 2.17. Thus, the inclusion of these variables in the problem would be conditional on a certain architectural decision.

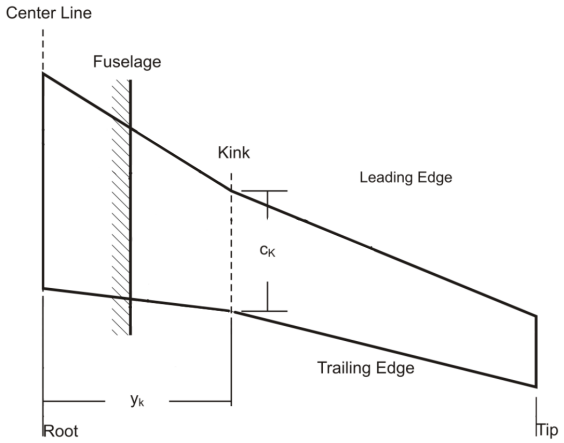
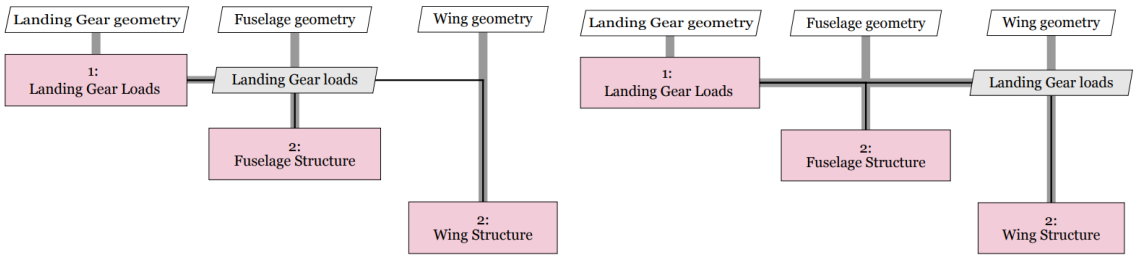


Figure 2.17: The double tapered wing. If chosen in detriment of a simple tapered wing, additional variables are needed to describe the wing geometry, such as the kink position (y_k) and the local chord at kink (c_k). Adapted from Scholz (2015).

Data Connection

In some cases, the routing of variables can change as a result of an architectural decision. In other words, it is possible that the set of disciplines involved in the exchange of a certain variable is different according to the architecture at hand. Thus, MDO platforms used for system architecture optimization should be able to dynamically reroute variable connections between disciplines. An example is shown in Figure 2.18, where landing gear loads are passed to different disciplines depending on the landing gear configuration of the aircraft.



(a) Landing gear attached to the fuselage

(b) Landing gear attached to the wing

Figure 2.18: Example of a data connection architectural influence. Depending on the aircraft landing gear configuration (attached to fuselage or wing), the landing gear loads are passed on to the corresponding structural tool. Reproduced from García Sánchez (2024).

Discipline Repetition

It can occur that a discipline must be repeated. However, the number of repetitions can be the consequence of an architectural decision. This should be adjusted automatically, as each iteration of the discipline involves different inputs and outputs.

An example could be the case of a hydraulic pump sizing discipline. If the number of pumps varies for different hydraulic system architectures, the number of repetitions of the discipline will depend on the architecture.

Discipline Activation

The inclusion of a discipline in the MDO problem may be subject to an architectural decision. This occurs when two or more technologies are available to perform a certain function, which require different computational tools for their assessment.

In early stage aircraft design, an example of discipline activation could be the choice of type of engine. If more than one type of engine is analyzed, and assuming different tools are used for their sizing and analysis, the required tool is dependent on the composition of each specific architecture. For each case, the appropriate tool will be activated, and the others will be deactivated. In these cases, the connection between the rest of the workflow and the active tool change depending on the architecture. Also, coupling variables could be different, given that the different engine disciplines might require different inputs and provide different outputs.

2.3.2 Enabling MDAO in System Architecture Optimization

In the previous section, the different types of architectural influences were presented. Now, the focus will be on the methodologies employed to deal with these influences, enabling the

use of multidisciplinary optimization for architecture evaluation. Special emphasis is placed on the strategies implemented in MDAx.

Garg et al. (2024b) present four high-level strategies for the integration of architecture influences, of which the *single dynamic* approach (illustrated in Figure 2.19) is considered to be the most useful for collaborative MDO out of the three that are capable of being implemented in current software solutions.

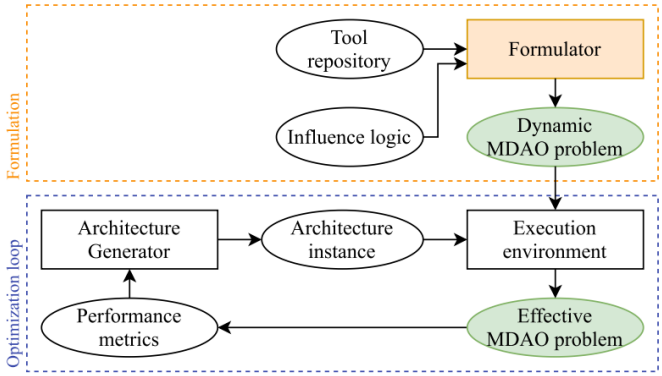


Figure 2.19: The "Single dynamic" strategy: a high-level strategy for dealing with architectural influences in MDAO problems. The formulation phase is executed before the optimization loop, which is carried out without user interaction. MDAx is an example of a "formulator", whereas RCE can be used as the execution environment. Adapted from Garg et al. (2024b).

In a single dynamic MDAO problem, tools are selected from a previously defined repository and the influence logic is specified. The influence logic consists of the problem's architectural influences and information on how the workflow behavior should be altered based on these influences. Contingent on these two things, a formulator (a platform like MDAx, for example) generates a dynamic MDAO problem that, when later executed in an appropriate environment, automatically modifies its behavior based on the given architecture instance. The result is an effective problem for each architecture assessed.

It is notable that the formulation phase is carried out prior to, and thus outside the optimization loop. The automatic formulation of static MDAO problems for each architecture within the optimization loop constitutes the "*on-demand*" strategy (Garg et al. 2024b), which is currently beyond the state of the art.

The extension of MDAx to support architectural influences was done based on the single dynamic strategy. In this way, the exported RCE workflow contains all the logic necessary for the dynamic modification of its execution behavior. The way in which each architectural

influence is dealt with in MDAx is presented next.

Discipline Activation

If a discipline's inclusion in the MDO problem is dependent on an architectural decision, it is associated (in MDAx) with the condition(s) which determines this inclusion. This condition is called an *activation assertion* (García Sánchez 2024). In such cases, the RCE export includes a script known as the *activation logic script*, which is executed prior to the tool and checks if the assertion is satisfied. If it is, the discipline is executed, being bypassed otherwise (Figure 2.20).

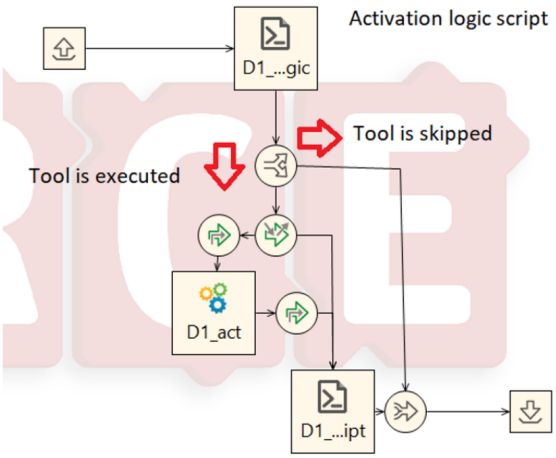


Figure 2.20: Example of a tool with activation logic in RCE. The workflow XML file is parsed by the activation logic script, which checks if the activation assertion is true or false. If true, the XML is passed to the tool and the tool is executed. Otherwise, the workflow XML is passed to the next discipline. Reproduced from García Sánchez (2024).

This check, executed by the activation logic script, is done by assessing an XML file called the workflow XML. This is the file passed from tool to tool in RCE which contains all information on the evaluated architecture. The condition being evaluated can be associated with the existence of a certain node in the XML file, the number of times a certain node is instantiated or the contents of the node. Table A.1, included as an appendix, lists all currently implemented ways of checking activation assertions.

Discipline Repetition

The number of repetitions of a discipline can be set as a predefined number or be associated with the number of instances of a certain component in the workflow XML file. The RCE export

of disciplines with possible repetition comes with a few additional scripts, as can be observed in Figure 2.21.

The "Global to Local" script determines the number of repetitions to be made based on the predefined assertion condition and selects the appropriate inputs and outputs for each iteration. The latter task is done by converting the workflow XML file (global data) into a somewhat altered version of it, adequate for each iteration (local data). Then, the tool outputs are written back into their correct form (global) in the workflow XML by the "Local to Global" script. Finally, the "Iterator" block checks the number of already executed iterations and, based on that, determines wether further loops are required or if the execution can proceed to the next discipline.

Data Connection

In MDAx, all possible data connections are included in the formulation and exported to RCE. Thus, when certain connections are not present for a specific architecture, the variables involved are simply removed from the input of the receiving discipline. This is possible due to the fact that MDAx uses a central data schema (García Sánchez 2024).

The conditions for deactivation of variables are stated in conditional assertions related to XML nodes, much like in the case of activation and repetition assertions. This information is stored and exported in the "Global to Local" script, which checks these conditions and, if true, the nodes are deleted from the XML file passed to the tool. When data connection and repetition exist for the same discipline, the "Global to Local" script will include the necessary tasks for both.

Conditional Variables

Before the execution of any tool, an input filter parses the workflow XML file and selects only the parts of the file containing variables which are defined as inputs for the tool (see Figure 2.21). When a certain variable is not found by the filter, it is simply ignored. This straightforward behavior deals with conditional variables. It doesn't, however, remove them from the MDO problem.

To remove variables from the problem entirely, an approach based on the deletion of said variables from the tool output by the "Local to Global" script is proposed, but not yet implemented (García Sánchez 2024).

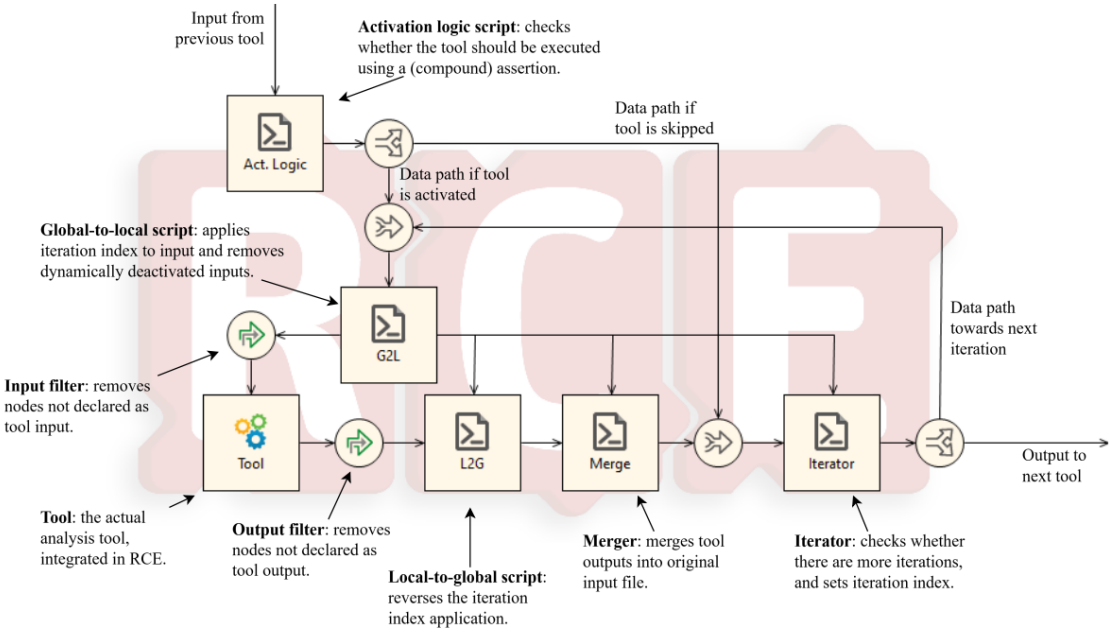


Figure 2.21: Overview of dynamic MDAO implementation, generated in MDAx and exported to RCE. In this example, the tool is at least subject to the discipline activation and repetition architectural influences. The data connection and conditional variables influences may also be present, but are not directly discernible in the workflow. Reproduced from Garg et al. (2024b).

3 Hydrogen Fueled Aircraft

The strong push towards sustainable aviation and carbon-neutrality has spurred a strong interest in hydrogen as an aviation fuel. The fact that its consumption does not emit CO_2 and lowers NO_x emissions (in the case of fuel cells, eliminated altogether) makes hydrogen an important research object. The focus is on understanding whether it could replace kerosene in the long run, with sustainable aviation fuels (SAFs) acting as a transitory solution. This chapter attempts to provide insight on the challenges and singular characteristics of hydrogen aircraft, with particular emphasis on the propulsion and fuel systems.

3.1 Aircraft Design Considerations

Aircraft designs that use hydrogen as a fuel have significant distinctions compared to their kerosene counterparts. These differences naturally arise from the different properties of the two fuel types. Table 3.1 compares select properties of the two fuels.

Hydrogen has a specific energy (energy per unit mass) that is roughly three times higher than that of Jet A-1 fuel. On the other hand, its energy density (energy per unit volume) can be as much as an order of magnitude lower, depending on the type of storage.

Table 3.1: Select properties of kerosene, liquid (LH_2) and gaseous (GH_2) hydrogen. Adapted from Adler & Martins (2023), Peschka (1992) and Brewer (1991).

Properties	Kerosene	LH_2	GH_2 (350 bar)	GH_2 (700 bar)
Specific energy (KJ/g)	42.8	120	120	120
Energy density (MJ/L)	34.9	8.5	2.9	4.8
Boiling point (K)	440-539	20.27	20.27	20.27
Storage temperature (K)	ambient	20	ambient	ambient
Lower hating value (KWh/kg)	9.5	2.36	2.36	2.36

The two most viable options for hydrogen storage are cryogenic liquid and compressed gas. Liquid storage offers the best energy density, being still approximately four times lower than that of kerosene. It also requires maintenance of the fuel below its boiling point of around 20 Kelvin. This substantial difference compared to ambient temperature means that fuel tanks must be designed to minimize heat transfer to the fuel, also referred to as *heat leak*. Excessive *heat leak* leads to evaporation of the fuel. This phenomenon is called *boil-off* and causes a pressure buildup in the tank, requiring it to be vented if a critical pressure is reached (Brewer 1991).

Compressed gas storage does not have this *boil-off* issue, as the hydrogen is stored in a gaseous state at ambient temperature. However, in order to attain even remotely competitive energy densities, hydrogen must be stored at high pressures, requiring significantly heavier and more robust tanks capable of withstanding said pressures (Nash et al. 2012). Nevertheless, even at 700 bar, the energy density is just over half that of liquid hydrogen, as per Table 3.1.

There seems to be a consensus among research and industry that liquid storage is the most viable solution, the low energy densities of gaseous storage being simply prohibitive. The NASA-funded Lockheed investigations from the 1970's (Brewer & Morris 1978), the European Community-funded CRYOPLANE project (Airbus Deutschland GmbH 2003) and GKN's H2GEAR program (Wood et al. 2023) are just some examples of cases where liquid storage was opted for.

Given its higher specific energy compared to kerosene, for a given amount of energy required for a mission, the fuel weight is significantly lower. This weight improvement is offset, however, in various different ways. The low energy density of hydrogen leads to a higher volume of fuel required for a given mission. Thus, hydrogen tanks are notably heavier, not only because of their larger size, but stemming also from the thermal insulation and structural soundness requirements mentioned earlier (Verstraete 2013).

Additionally, the minimization of the heat leak of a tank is related to the minimization of its surface area-to-volume ratio, meaning that the optimal shape for a liquid hydrogen tank is a sphere (Adler & Martins 2023). This makes it infeasible to store fuel in the wings, as spherical tanks of the required size do not fit inside a wing. In fact, most hydrogen aircraft concepts are associated with the so-called "dry-wing" configuration, alluding to the fact that these do not carry fuel on the wings. Figure 3.1 illustrates various tank placement concepts for a regional airliner, with the notable fact that all options integrate the tanks somewhere along the fuselage. Because of this, the typical structural load alleviation provided by the fuel weight on the wings is not present, calling for higher strength requirements and thus an increased structural weight of the wing (Verstraete 2013).

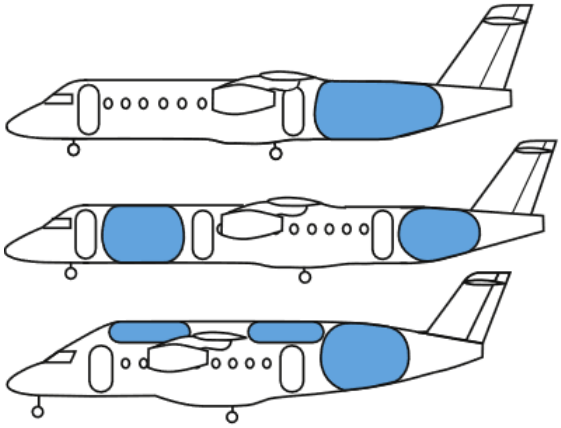


Figure 3.1: Potential tank placement configurations for a regional airliner. Each configuration presents advantages and disadvantages in relation to mass, center of gravity position, cabin access from the cockpit, among others. Reproduced from Verstraete et al. (2010).

3.2 Propulsion Systems

The most common types of propulsion system for hydrogen aircraft are gas turbine engines and fuel cells (Wallington et al. 2025). There are also many different possible hybrid configurations between the two, where batteries may play a role in energy storage.

Hydrogen-burning gas turbine engines are more suited to larger medium to long-range aircraft, whereas fuel-cell-based propulsion is better suited to smaller, regional short-range aircraft. This is related to a few key points, one of which being that turbofan or turboprop engines allow for significantly higher specific power when compared to a fuel cell arrangement with an electric motor and propeller. At the larger scales associated with long-range, heat dissipation becomes a major bottleneck in fuel-cell-based systems, requiring a large and heavy thermal management system. Furthermore, the current state-of-the-art in electrical machines simply does not provide sufficient power capable of directly substituting modern turbofan engines (Adler & Martins 2023).

As the scale reduces, on the other hand, so does the efficiency of gas turbine engines. This contrasts with fuel cells, the efficiency of which is much less sensitive to size. Furthermore, the efficiency of fuel cells is higher than that of the most efficient gas turbines, allowing for a more fuel-efficient solution (Wallington et al. 2025). At lower power requirements, the thermal management issues of fuel cells are more easily mitigated, making them a strong contender for smaller short-range applications.

3.2.1 Hydrogen Combustion

Combustion engines made for the use of hydrogen as fuel are very similar to the currently available kerosene-based engines. In fact, most components require little to no modification whatsoever. Only the combustion chamber needs significant analysis and design work to adapt to the combustion properties of the distinct fuel (Brand et al. 2003).

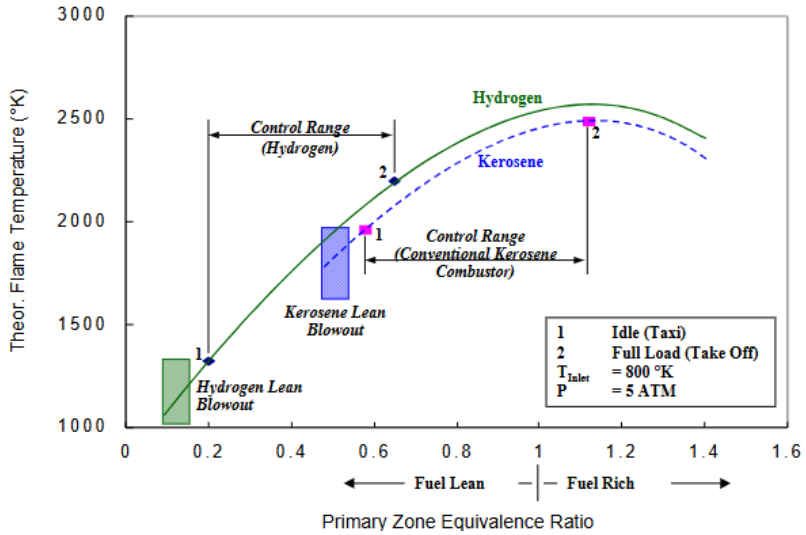


Figure 3.2: Temperature characteristics of hydrogen and kerosene combustion. Hydrogen has a wider range of flammability, allowing a shift into leaner fuel mixture regions. Reproduced from Brand et al. (2003).

Figure 3.2 shows that, compared to kerosene, hydrogen has a larger range of flammability, allowing it to be burned at leaner mixtures and lower temperatures, which in turn increases turbine life by decreasing turbine entry temperature (TET) (Khandelwal et al. 2013). Although premixing of hydrogen with air is seen as potentially dangerous, gaseous fuel injection enables more efficient diffusion than liquid droplets, potentially eliminating the need for this altogether. Hydrogen also possesses higher flame speed and reactivity, which leads to shorter residence times and smaller combustion chamber designs. In addition, since NO_x formations are strongly dependent on flame temperature and residence time, both of which are lower in hydrogen combustion, emissions of these gases also tend to be lower (Tiwari et al. 2024).

3.2.2 Fuel Cell Propulsion Architectures

Hales et al. (2023) present the typical fuel-cell-based hydrogen electric propulsion system architecture, illustrated in Figure 3.3. Both the options of gaseous and liquid fuel storage are included, the conclusion having already been reached in this chapter that liquid storage is

considered to be the most feasible.

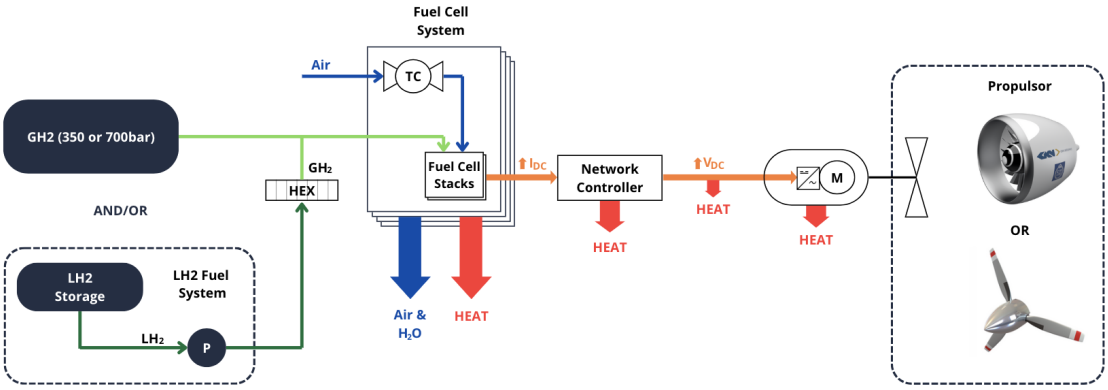


Figure 3.3: Conventional fuel-cell-based hydrogen electric propulsion system architecture. Adapted from Hales et al. (2023).

In the case of liquid storage, the fuel is pumped through the feed lines and evaporated in a heat exchanger, which brings it to the desired thermodynamic state for entry into the fuel cell system. This system is composed of a fuel cell stack and the so-called balance of plant (BOP) (the fuel cell system is analyzed in more detail in a later subsection).

In the fuel cell stacks, hydrogen and oxygen react to produce electricity, the only by-products of this reaction being saturated air and condensed water. Heat is also released, originating from the inefficiencies of the process.

A DC bus typically regulates the voltage of the generated DC power and distributes it not only throughout the propulsion system, but also to secondary loads such as the environmental control system, avionics and flight control systems.

Finally, AC power is obtained through an inverter and fed to an electric motor, which in turn powers a propeller or ducted fan. A gearbox may also be employed to decouple the rotational frequencies of motor and propeller.

Significant improvements in mass, volume and efficiency can be obtained, in comparison to the conventional architecture, with the application of active cooling, also known as cryogenic cooling (Wood et al. 2023). In this technology, a cryogenic coolant loop (typically helium) keeps the temperature of the power supply lines and electrical machines low enough to take advantage of the concept of superconductivity. This significantly reduces heat losses in the cables and motor, allowing these to be of smaller size and thus have lower mass. Synergis-

tically, the heat extracted by the coolant can be used in the heat exchanger to evaporate the liquid fuel. Cryogenic cooling plays an integral part in GKN's H2GEAR program (Wood et al. 2023), as well as the Airbus Cryoprop demonstrator (Airbus 2024).

Hybrid Electric Architectures

When it comes to degradation and durability, the optimal scenario for fuel cell stacks consists of continuous operation at constant power levels (Wood et al. 2023). Additionally, the dynamic response of fuel cells is considered slow (Jiang & Fahimi 2010). For these reasons, the inclusion of battery systems in the propulsion architecture as a means to react to power demand oscillations (particularly at take-off or highly dynamic accelerations) is appealing. This joint architecture constitutes a hybrid fuel cell electric powertrain.

There are two main possible architectures for hybrid electric systems: series and parallel configuration (Soleymani et al. 2024). In the series configuration, the fuel cell acts mainly as an auxiliary power source by charging the batteries, which in turn are responsible for powering the electric motors that drive the propellers. In parallel configuration, depicted schematically in Figure 3.4, both the fuel cell and the battery work in tandem to provide the required power. In a study on fuel cell powered unmanned aerial vehicles, Xu et al. (2022) conclude that although higher in complexity, parallel systems outperform series configurations in efficiency, fault tolerance, dynamic response and total mass.

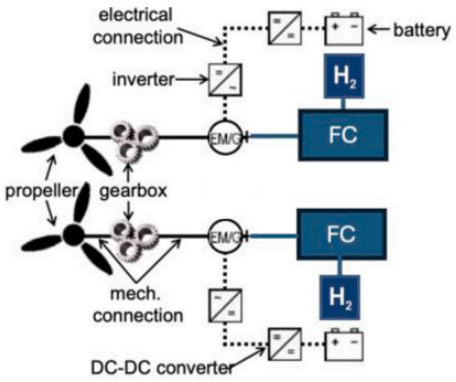


Figure 3.4: Parallel hybrid electric configuration for hydrogen fuel cell aircraft powertrain. Adapted from Soleymani et al. (2024).

In turn, parallel configurations themselves can be built in many different architectures, all of which can generally be categorized as either of passive or active type (Soleymani et al. 2024). Example schematics of each type of architecture are given in Figure 3.5. Their naming refers to power distribution and control. In the active configuration, controlled converters allow for

the variation of the power share between fuel cell and battery, helping to maintain close to optimal operating conditions for both components, thus increasing their lifetime. In the passive configuration, the power sources are directly connected to the bus and the fuel cell operates at the voltage set by the batteries. Comparatively, passive configurations are generally lighter, simpler, cheaper and have less conversion losses. The major disadvantage is again that active power control is not possible, which significantly jeopardizes the longevity of the components (Xu et al. 2022).

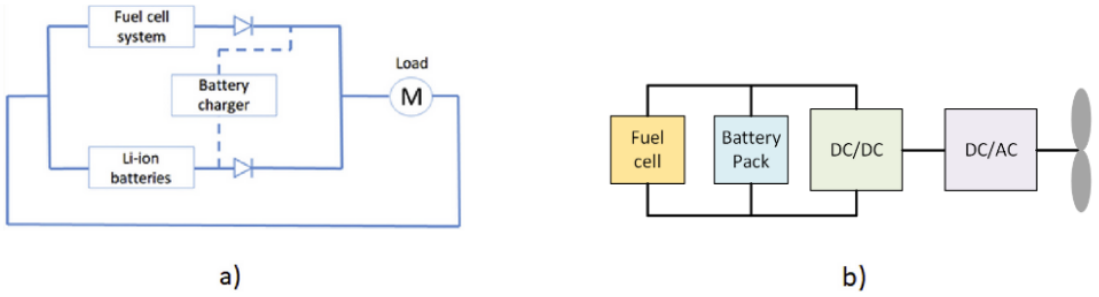


Figure 3.5: Parallel hybrid fuel cell/batteries power system configuration examples. a) A battery charger is employed and a diode is present for each power source. b) The battery is directly connected to the fuel cell and a dc–dc converter is placed between the dc bus and the load. Adapted from López González et al. (2019) and Xu et al. (2022).

Fuel Cell System

The fuel cell system, representatively depicted in Figure 3.3 as a part of the total powertrain architecture, is made up of the fuel cell stack and the balance of plant (BOP).

The two types of fuel cell considered most promising for aeronautical applications are proton exchange membrane fuel cells (PEMFC) and solid oxide fuel cells (SOFC) (Spencer & Martin 2013). Comparatively, PEMFCs operate at low temperatures, have short start-up times (in the order of seconds) and higher specific power (O'Hayre et al. 2016). SOFCs, on the other hand, operate at temperatures up to 1000° C and because of that have start-up times in the order of tens of minutes. Furthermore, SOFCs do not require humidification and can potentially achieve higher efficiencies if the waste heat is used constructively (Dicks & Rand 2018).

PEM fuel cells can further be divided into low- and high-temperature categories. The low temperature variant is associated with operating temperatures up to 100° C, whereas high-temperature PEMFCs can go up to 200° C (Tiwari et al. 2024). Out of all fuel cell types mentioned, the low temperature PEMFC has the highest state of technology readiness, being

relatively well established in the automotive industry (Tiwari et al. 2024). It is also expected to be the most employed technology in the coming decades (Bhatti et al. 2022).

As stated previously, apart from the stack, the fuel cell system comprises the *balance of plant* (BOP). This essentially constitutes all auxiliary systems and components required for the operation of the fuel cells, namely related to thermal, exhaust and power management, as well as pressurization and humidification. It thus contributes significantly to the volume and mass of the whole FC system, in some cases outweighing the fuel cell stack itself (Adler & Martins 2023).

Figure 3.6 schematically represents the composition of a fuel cell system. The hydrogen storage unit and electric motor are not part of the system, but have a direct connection to it. The electrical block may also require a power management and distribution system.

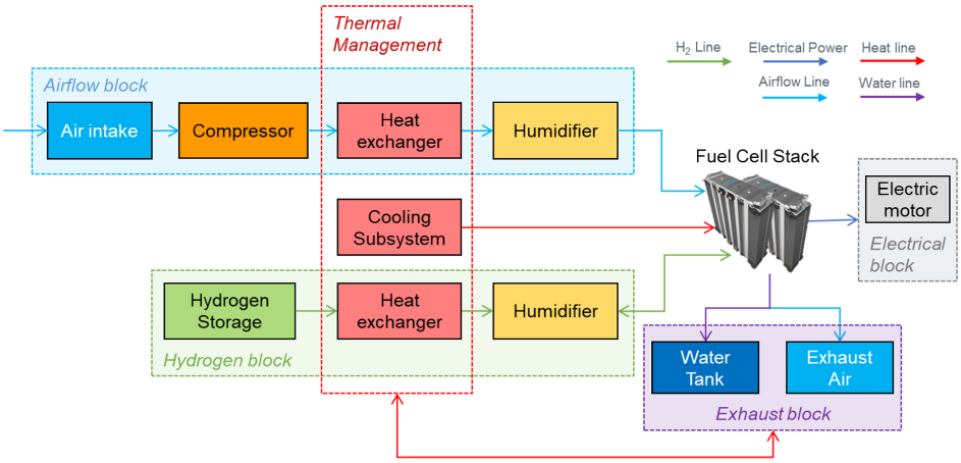


Figure 3.6: Fuel cell system architecture. Reproduced from Massaro et al. (2023).

In the case of PEMFCs, the maintenance of adequate humidification is crucial to its performance (Palladino et al. 2023), thus warranting the use of humidifiers for both intake air and fuel. This humidification can be performed efficiently through a closed-loop water system using the condensed water expelled from the fuel cells (Millis et al. 2009). A turbo-compressor regulates the pressure and mass flow of the air fed to the fuel cells, assuring a correct stoichiometric fuel mixture independently of altitude (Hales et al. 2023).

An example thermal management system can be analyzed in more detail in Figure 3.7. A cooling subsystem, using a fluid such as an ethylene glycol water mixture (Vietze & Weiland 2022), removes heat from the fuel cell stacks and uses it to control the temperature of both inlet air and hydrogen in separate dedicated heat exchangers. In the case of the fuel, the heat

exchanger may also serve as an evaporator. Any additional excess heat is transferred to other systems through another heat exchanger or dissipated in a radiator (Palladino et al. 2021).

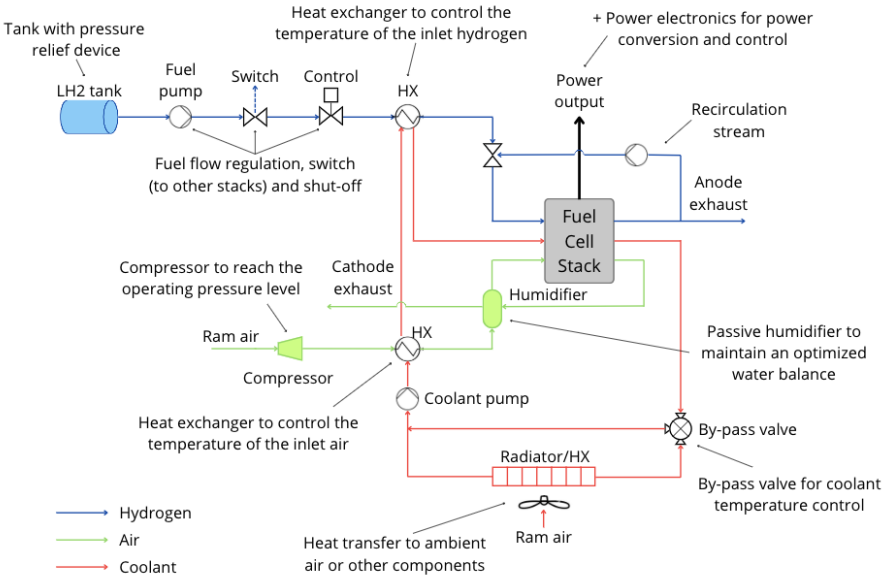


Figure 3.7: Schematic layout of the fuel cell BOP including hydrogen tanks. Adapted from Palladino et al. (2023).

3.3 Fuel System

In previous sections, the subjects of tank design and integration have been briefly analyzed. Thus, in the present section, emphasis will be placed on the set of components and subsystems (downstream of the tank) required to safely transport the fuel and provide it to the propulsive units in the required thermodynamic state. This includes fuel lines, safety venting lines, pumps, sensors, valves, resistive heaters and heat exchangers, among other components.

3.3.1 Types of Fuel System

Ultimately, a fuel cell consumes hydrogen in the gaseous state. Assuming liquid storage, the type of fuel system is largely defined by the point at which evaporation occurs. This dictates whether the fuel is transported as a gas or a liquid, having consequences, for example, on the sizing and design of the fuel lines.

The distinction naturally arises between a gaseous and a liquid fuel system. In the former, the liquid hydrogen is evaporated by means of an electric heater or heat exchanger (Vietze & Weiland 2022), either in the tank or immediately after its exit. In the case of hydrogen

storage as a compressed gas, the system would in most part be the same, yet dispensing the vaporization step. In the latter case, the fuel is transported as a cryogenic liquid and vaporized just before being fed to the fuel cell. Here, the physical proximity to the fuel cell allows for the use of part of its expelled heat to provoke a phase change of the hydrogen using a heat exchanger (Adler & Martins 2023).

There are two possible concepts for the movement of the fuel itself. It can be either pressure-driven or pump-driven. A pressure-driven system relies on the passive transport of hydrogen fuel via the pressure differential that exists along the distribution lines. In this case, the fuel must be stored at a pressure higher than the required inlet pressure of the propulsion system, as to account for pressure losses throughout the system. A consequence of this is that the tank must be designed to withstand these higher pressures, resulting in a heavier structure (Burschyk et al. 2024). Alternatively to higher tank pressures, compressors can be used downstream to raise the pressure to its required level (Millis et al. 2009). In a pump-driven system, the system pressure can be actively controlled by a fuel pump, thus also allowing for a lower tank storage pressure.

The most common approach found in the literature is the liquid fuel system. In part, this is due to the much closer similarity to current kerosene fuel systems. Furthermore, an active pump-driven system allows for better control over fuel flow. A gaseous system induces much more dynamic ullage pressure variations in the tank, making it more challenging to control (Adler & Martins 2023).

3.3.2 Fuel Pumps

Liquid fuel systems require pumps to pressurize and transport the fuel. In the case of gas turbine aircraft, the pressure rise effort is usually divided between a boost pump and a high pressure pump (Brewer et al. 1978). For fuel cell propulsion, a single boost pump is sufficient, as fuel cell intake pressures are much lower, generally in the realm of 3 bar for PEMFC (Vietze & Weiland 2022). Typically, more pumps are installed than practically necessary for reasons of redundancy and therefore related to safety (Airbus Deutschland GmbH 2003).

The two most common types of fuel pumps are centrifugal pumps, usually considered more adequate for higher flow rates, and reciprocating pumps (piston pumps), for lower flow rates at high pressure. For all pumps, avoidance of cavitation is crucial for durability and performance. This is achieved by ensuring sufficiently high pressure at the pump inlet, which can be done

with the use of superchargers in the case of reciprocating pumps and inducers in the case of centrifugal pumps (Peschka 1992).

The first generation of hydrogen aircraft is expected to be equipped with electrically driven multistage centrifugal pumps. Possible alternatives are piston and diaphragm pumps, yet these designs are currently at lower levels of technology readiness (Turner et al. 2022). High-pressure pumps are usually mounted on the engine, making them suitable for mechanical drive by the engine shaft. Specifically, both in the Lockheed investigations (Brewer 1991) and in the CRYOPLANE project (Airbus Deutschland GmbH 2003), the chosen configuration was an engine-driven centrifugal pump.

Finally, transfer pumps may also be part of the architecture, whose function is to move fuel between tanks. For this application, centrifugal pumps as well as jet pumps are seen as potentially suitable (Turner et al. 2022). Jet pumps in particular have the advantage of having no moving parts, increasing reliability, and can be driven by hydrogen recirculation (Han et al. 2022).

3.3.3 Insulation

In liquid fuel systems, the fuel supply lines need to be insulated in order to minimize the amount of vapor in the system (Brewer et al. 1978). The two most commonly assessed types of insulation are vacuum-jacketed and foam insulation. For both concepts, Peschka (1992) presents a double-walled design, the inner line made of stainless steel and the outer line of either aluminium or also stainless steel. In the foam design, the space between the walls is filled with sealed polyurethane foam.

When comparing the two, vacuum-jacketed lines have the best performance, achieving significantly lower values of heat loss for the same diameter. They do, however, come with significant downsides when it comes to structural reliability, as well as higher manufacturing, installation and maintenance costs (Brewer 1991).

For a foam-insulated line to provide insulation similar to that of a vacuum jacket, it will inevitably require a larger diameter (large foam thickness) and be overall heavier than its counterpart. Nevertheless, foam insulation was chosen in the Lockheed investigations for reasons of reliability, operational simplicity and costs (Brewer 1991). The final design is shown in Figure 3.8.

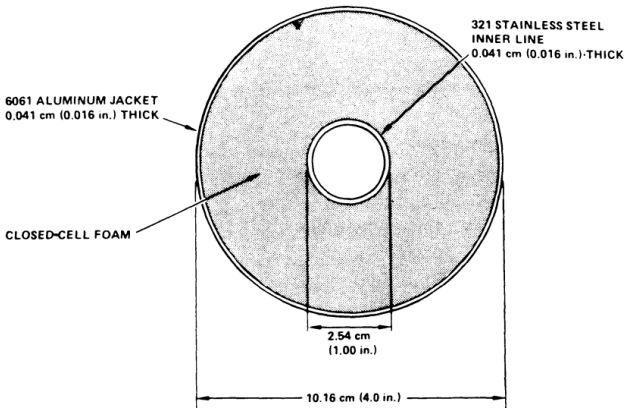


Figure 3.8: Cross section of the selected fuel line design of the Lockheed investigations. Consists of an inner line of stainless steel and an outer line of aluminium. In between, closed-cell polyurethane foam is used. Reproduced from Brewer (1991).

In the CRYOPLANE project, cryogenic composite lines are selected, listing low mass, high strength, and resistance to damage and vibrations as some of the advantages. However, the feasibility of such lines is not guaranteed (Airbus Deutschland GmbH 2003).

3.3.4 Tank Pressurization

As hydrogen is consumed throughout a flight, the amount of fuel in the tank gradually decreases, while the tank volume naturally remains the same. This causes a pressure drop in the tank that should be monitored and combated to ensure that it remains within acceptable bounds. There are two main methods usually used to maintain pressure in the tank: autogenous pressurization and injection of another gas, commonly helium (Millis et al. 2009).

Autogenous pressurization simply refers to the use of the propellant itself as a means of pressurization, typically by vaporizing part of the liquid fuel on demand (Turner et al. 2022). This can be done with resistive heaters or heat exchangers, both internal or external to the tank, with the latter case requiring re-injection of the evaporated fuel.

A helium pressurization system works on the basis of using helium to replace the consumed volume of fuel, thus maintaining adequate pressure. In this case, an additional tank is needed to store the helium, along with pipes, valves and control systems for its use (Millis et al. 2009).

It is notable that even in cases where heat exchangers or helium pressurization are used, resistive heaters internal to the tank are commonly still added for redundancy (Vietze & Weiland 2022, Millis et al. 2009).

3.3.5 Fuel Conditioning

Fuel conditioning refers to the steps required to bring the fuel to the correct state before feeding it to the engine or fuel cell. Pressure and mass flow are regulated by flow control valves. As mentioned in Section 3.2, humidification is necessary for PEM fuel cells. A humidifier can be integrated in an efficient manner by using the condensed water from the fuel cell exhaust in a closed-loop water system (Millis et al. 2009).

The temperature of the hydrogen must also be regulated. It is typically heated and, in the case of liquid fuel systems, previously vaporized. These two functions of vaporization and heating can be performed: separately, using a component for vaporization followed by a different component for heating; or simultaneously, by vaporizing and heating the fuel concurrently in the same component. To do this, electric heaters and/or heat exchangers are used. Heat exchangers tend to be favored, since they allow for a symbiotic use of heat removed from the engine or fuel cell stack. Their application eliminates the need for large radiators (Palladino et al. 2021), reducing the drag of the thermal management system (Adler & Martins 2023) and thus increasing overall efficiency.

4 Methodology

The automatic exploration of the design space for hydrogen fuel system architectures constitutes a system architecture optimization problem. Such SAO studies are made possible by the use of Dynamic MDAO. The following chapter presents the methodologies used to conceive and implement the SAO problem at hand.

First, the employed Dynamic MDAO framework is introduced through its application to a benchmark problem. The framework is then applied to the fuel system optimization problem. The main steps required for its implementation are sequentially presented. These can be categorized into four main activities: the definition of the architecture design space, the creation of the design and analysis disciplines, the formulation of the dynamic MDAO problem, and its implementation in a PIDO environment.

4.1 Framework Introduction and Training

Apart from specific training in the form of workshops and tutorials on ADORE, MDAx and RCE, a benchmark problem was solved in order to familiarize the author with the implementation of a Dynamic MDAO problem from start to finish. The problem is of mathematical nature and is based on a modified Fourier series.

First, the mathematical problem is succinctly described, followed by a discussion of its implementation with the used SAO framework. The implementation is also improved by including the Node Factory Evaluator (NFE) (Bussemaker 2025), explained further in Section 4.1.2.

4.1.1 Problem Formulation

The mathematical benchmark problem, presented in detail and solved by García Sánchez (2024), consists of a system architecture optimization problem based on a modified Fourier series. The problem contains all four types of architectural influences and was designed to check the implementation of architectural influences in MDO platforms. The objective is to determine the best approximation function F(x) (eq. 4.1) to approximate a periodic objective function f(x).

$$F(x) = A_0 + \sum_{n=1}^{N_A} \sum_{j=1}^{Nwa_n} A_n cos(w_{nj} * w_0 * x) \; + \sum_{m=1}^{N_B} \sum_{k=1}^{Nwb_m} B_m sin(w_{mk} * w_0 * x) \tag{4.1}$$

To reach the optimal approximation, the optimizer has control over architectural decisions such

as the number of terms in the function, and design variables such as the coefficients A_0, A_n, B_m or the frequencies w.

Four disciplines are needed for the assessment of the problem. The Y discipline provides as output the cosine terms of F(x). In turn, the Z discipline provides the sine terms. If there is more than one sine or cosine term, the respective discipline is repeated. The C discipline defines the constant term A_0, using select outputs of Y and Z as inputs. Finally, an objective function block calculates the error between the periodic function f(x) and F(x). The periodic function chosen for this problem is the *sawtooth wave* function.

It is not within the scope of this thesis to describe the problem or the disciplines in detail. The interested reader is directed to the work of García Sánchez (2024). The emphasis is on the implementation of the SAO framework and problem, which is the topic of the next section.

4.1.2 Problem Implementation

The applied SAO framework is illustrated in Figure 4.1, with reference to the platforms and data interfaces used. To employ such a framework, the first step is usually the creation of the architecture design space in ADORE. The optimization problem is also fully defined in ADORE, where the user can select the optimization algorithm, objectives and constraints, and where architectural decisions are encoded as design variables. When the SAO loop is executed, ADORE acts both as the optimizer and architecture generator.

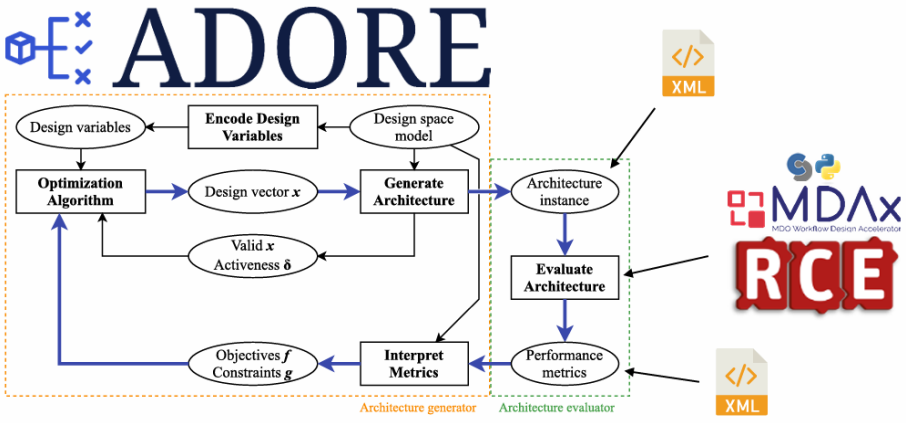


Figure 4.1: The SAO framework used at the DLR Institute of System Architectures in Aeronautics. ADORE retains the functions of optimizer and architecture generator. MDAx is used to create the MDO problem, which is then exported and executed in RCE. The evaluation disciplines themselves are generally Python tools and CPACS is used as an XML-based central data schema. Adapted from Bussemaker (2024).

After creating the design space, the dynamic MDAO workflow is defined in MDAx. Since MDAx does not yet support the formulation of such problems in its GUI, this has to be done in the back-end. The majority of the process is just like any other MDO problem, with the definition of the disciplines to be included and their inputs and outputs. MDAx then establishes discipline connections automatically, made possible by the use of a central data schema. The main difference with dynamic MDAO lies in the architectural influences, which are represented at this stage through the definition of the *assertions* introduced in Chapter 2.

These assertions are defined in the configuration file of each discipline. The configuration file is a JSON file where all the information regarding architectural influences is stored. Figure 4.2 shows the configuration file of the Y discipline of the mathematical benchmark problem. The `activationLogic` key holds the assertion that dictates whether the tool is included or not in the MDO problem execution. In this case, the Python class used to check the assertion is `XPathExists()`. So, if the xpath given as argument is present in the workflow XML file, the tool is activated. Another architectural influence affecting the Y tool is repetition. The repetition assertion can, in this case, be interpreted as the number of instances of the "y" node in the workflow XML. For example, if there are two "y" nodes in the file, the discipline will be executed two times. Table A.1 lists all currently implemented Python classes used to check assertions.


```
{"activationLogic": "XPathExists('/disciplines/y')",
 "executionName": "Y",
 "name": "Y",
 "notes": "",
 "repetition": "NumberofInstances('/disciplines', 'y')",
 "version": "1.0"
}
```


Figure 4.2: Configuration file of the Y discipline. The xpath "`disciplines/y`" is associated to cosine terms in the approximation function. Thus, the tool will only be executed if there are cosine terms in the function, and will be executed the same number of times as the number of cosine terms. Reproduced from García Sánchez (2024).

After the problem is fully formulated in MDAx, it is exported to run in a PIDO environment, in this case RCE. The mathematical benchmark problem workflow is presented in Figure 4.3. The "AdoreOpt" block represents the interface with ADORE, which works as an optimizer and an architecture generator. This block is not part of the MDAx export and must be integrated into the workflow in RCE. The Y and Z disciplines may be executed simultaneously, the structures surrounding them being related to the discipline activation and repetition architectural

influences. Then, the C and Objective Function disciplines are run in that order, both being affected by the "data connection" architectural influence, before the loop restarts.

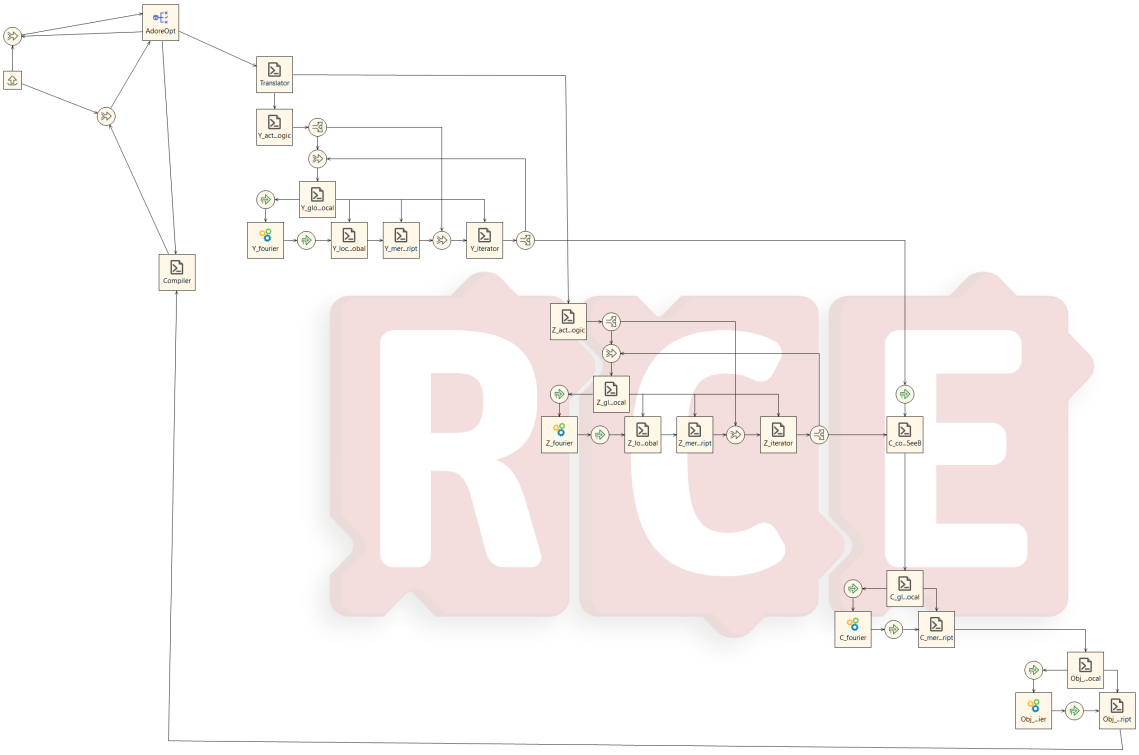


Figure 4.3: RCE workflow of the mathematical benchmark SAO problem, as exported from MDAx.

What is yet to be mentioned are the interfaces between the architecture generator and the architecture evaluator. In the workflow illustrated in Figure 4.3, The AdoreOpt block is exporting architectures using ADORE's underlying data model, which does not correspond to the data model used in the evaluation tool chain. In aircraft design, this data model is typically the CPACS data schema. For this reason, the proposed architecture files need to be translated into the appropriate data model, and the outputs of the evaluation tool chain need to be translated back before being returned to the AdoreOpt block. These tasks are performed by the Translator and Compiler scripts, respectively, shown in Figure 4.4.

ADORE also supports this translation being done internally, through what is called the XML Node Factory Evaluator (NFE). The NFE is nothing more than a rule-based translation to a predefined data format, where the user establishes relationships between architecture elements and XML nodes that should be created (Bussemaker 2025). These relations are established through so-called node factories, whereas metric factories allow the nodes for performance metrics to be read and translated back into the ADORE data model. The Node

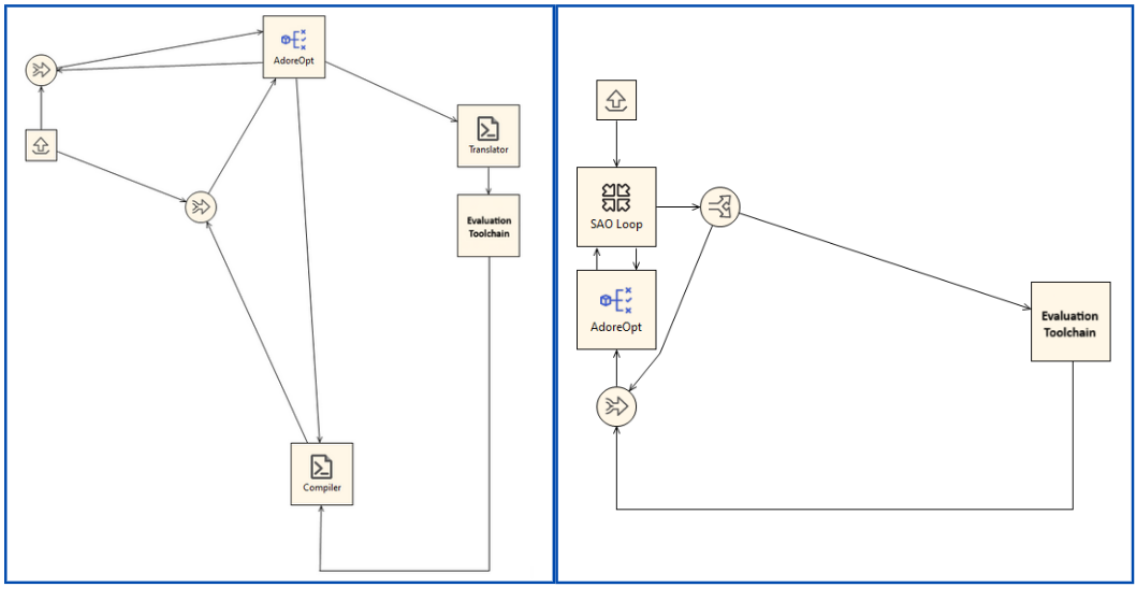


Figure 4.4: Comparison between the original architecture export (left), requiring translation scripts before and after the evaluation tool chain, and the NFE-based export (right), where the resulting XML file can be directly used as input to the evaluation tool chain. The "SAO Loop" block, in this case, does not influence the workflow.

Factory Evaluator will be described in more detail in Section 4.5.2.

The author's contribution to the mathematical benchmark problem was the implementation of an NFE-based data export. This is realized through a static base XML file (denominated NFE base file), containing all the node and metric factories, which is given as an additional input to the AdoreOpt block. The result of this is that translation scripts are no longer needed before and after the evaluation tool chain, as the exported XML file can be used directly (see Figure 4.4).

With the workflow fully operational, what remains is the execution of the optimization problem, the results of which are analogous to those presented by García Sánchez (2024). Having gone through the implementation of the mathematical benchmark SAO problem, the attention now turns to the actual application case: optimization of hydrogen fuel system architectures in the context of early stage aircraft design.

4.2 Definition of the Architecture Design Space

The typical first step in the implementation of a system architecture optimization problem is the definition of the architecture design space. The current section goes through this process, which includes deciding on the system boundaries and what is and is not included in the

analysis. This is naturally influenced by the chosen reference aircraft, which is also presented herein.

4.2.1 Propulsion System Architectures

Initial investigations for the present thesis focused on the types of propulsion system and corresponding possible architectures for use in a hydrogen-fueled aircraft. This is necessary because the propulsion system and the fuel system naturally have many interfaces, making it difficult to clearly separate the two. For example, in the case of a fuel cell there is significant interaction between its balance of plant and the fuel system (Millis et al. 2009). Because of this, in order to design an optimal fuel system, the propulsion system architecture should be known in some detail.

One of the products of the literature review was the outlining of the architectural design space for the propulsion system of a propeller-driven hydrogen aircraft, represented in Figure 4.5 in the form of an ADSG. To this design space, the additional option of a turbofan engine could be added, among others. However, and as discussed in Chapter 1, the scope was established from the beginning to revolve around the integration of fuel cells in the architecture as a main propulsive element.

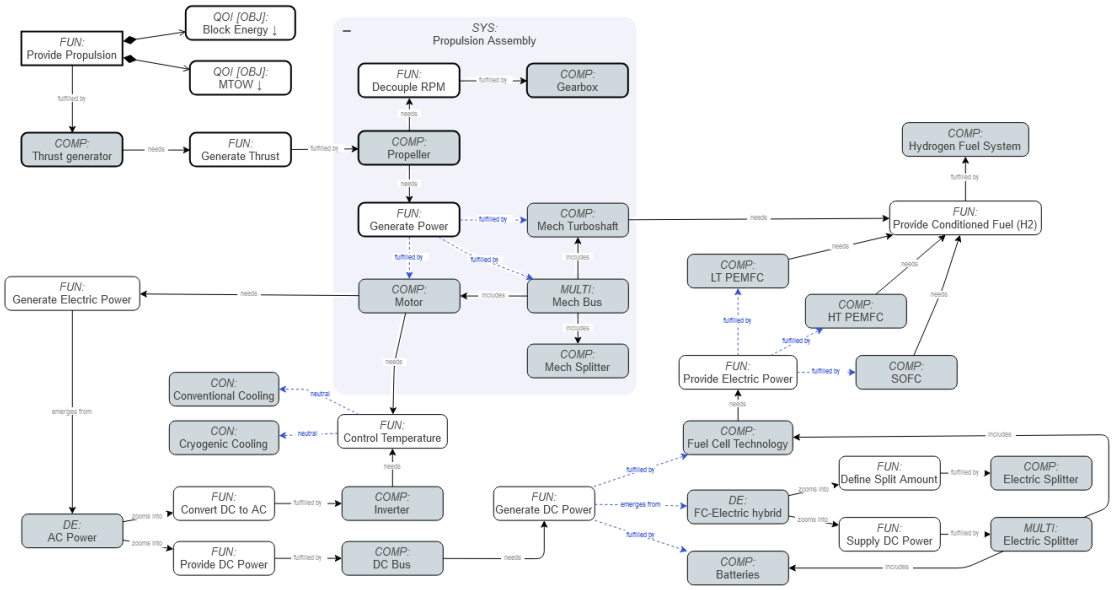


Figure 4.5: Architecture Design Space Graph (ADSG) for the propulsion system of a propeller-driven hydrogen aircraft modeled in ADORE. Adapted from Bussemaker et al. (2023).

In the ADSG, the boundary function "Provide Propulsion" is associated with two performance metrics: block energy and maximum take-off weight (MTOW), both of which ought to be minimized. Four architectural decisions are discernible, represented by the blue arrows. The first decision relates to the fulfillment of the "Generate Power" function and calls for the selection between an electric motor, a mechanical turboshaft and a hybrid solution integrating both components.

Whenever an electric motor is involved, additional decisions regarding the power source (fuel cell, battery or hybrid), cooling system and type of fuel cell (if applicable) come up. The implications of all three of these architectural choices are discussed in Chapter 3. For example, selecting between cryogenic and conventional cooling has implications not only on the composition and functioning of the cooling system, but also on the type of electric motor and wiring (Hales et al. 2023). Consequently, the total mass and efficiency of the propulsion system are influenced.

As will be discussed later on, a preexisting liquid fuel system sizing tool was used as a starting point for the creation of the various fuel system component disciplines. This tool was made for a specific architecture that did not include batteries in the propulsion system. For this reason, and to limit the complexity of the chosen system, a fully fuel-cell-based system is opted for. Furthermore, LT-PEM fuel cells are chosen because, as mentioned in Chapter 3, they are the most promising technology for the coming decades (Bhatti et al. 2022). Lastly, conventional cooling is opted for.

With these decisions made, the requirements are set for the selection of an adequate reference aircraft, which is the subject of the next section.

4.2.2 Reference Aircraft

The chosen reference aircraft is the ESBEF-CP1 (Figure 4.6), a hydrogen-powered regional concept aircraft. The name is a german acronym for "Development of Systems and Components for Electrified Flight", with CP1 standing for "Concept Plane 1". The ATR-72 inspired concept was developed by the DLR Institute of System Architectures in Aeronautics for the internal DLR project EXACT (DLR 2025) and used in the ESBEF project as a reference for system design (Bielsky et al. 2024).

As mentioned previously, the focus of the work is on aircraft that use fuel cells as the main propulsion units. Thus, within the realm of aircraft developed in the EXACT project and gen-

Figure 4.6: ESBEF-CP1: a hydrogen-powered concept aircraft developed by the German Aerospace Center (DLR). Reproduced from Bielsky et al. (2023).

erally being worked on at the Institute, the ESBEF-CP1 stood out as the most suitable option. Table 4.1 compiles the aircraft's most relevant Top-Level Aircraft Requirements (TLARs).

Table 4.1: TLARs of the ESBEF-CP1 reference aircraft. Adapted from Bielsky et al. (2024).

Characteristic	Value
Design Range [NM]	1000
Cruise Speed [Ma]	0.55
Cruise Altitude [ft]	27000
Max. PAX number [-]	70

As can be seen in Figure 4.6, the aircraft has a distributed propulsion arrangement, a high-wing and a T-tail configuration. Its ten propulsion units (referred to as pods) each contain a hybrid fuel cell system, a liquid cooling system, an air supply system and an electric powertrain consisting of an electric motor, a controller and a gearbox. The estimated mass of each pod is 350 kg (Bielsky et al. 2024). Table 4.2 presents the mass breakdown of the aircraft.

Table 4.2: Mass breakdown of the ESBEF-CP1 reference aircraft. Adapted from Bielsky et al. (2024).

Breakdown	Mass [kg]
Manuf. empty mass	17387
Operating empty mass	18787
Max. zero-fuel mass	26287
Max. landing mass	26512
Max. take-off mass	27038

Developed with an entry into service in 2040 in mind, technologies assumed to possess an appropriate readiness level in the mid 2030's are considered (Bielsky et al. 2023). Relevant assumptions are, for example, an energy density of 500 Wh/kg for Lithium-based batteries and a specific power of 6 kW/kg for fuel cell stacks (Bielsky et al. 2024).

A schematic of the hydrogen fuel storage and supply system can be seen in Figure 4.7. Fuel is stored in liquid form in two CFRP tanks located in the aft fuselage, behind the cabin deck. The hydrogen is then transported, also in liquid form, in triple-walled pipes routed through the pressurized area of the aircraft. Redundancy is assured by the assumption that all pods can be supplied by both tanks. After initial preliminary iterations, the fuel supply system (tanks not included) is estimated to have a mass of around 850 kg (Bielsky et al. 2024).

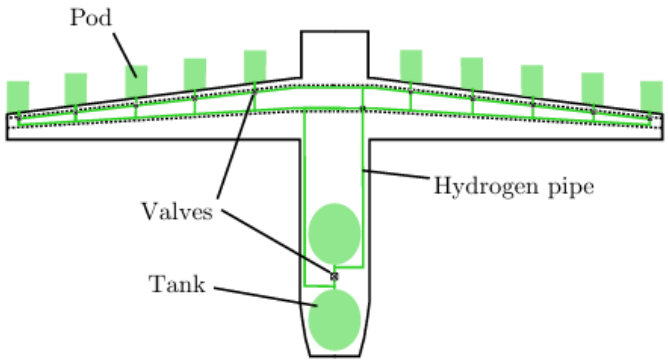


Figure 4.7: Architecture of the ESBEF-CP1's hydrogen storage and distribution system. Two hydrogen tanks in the aft fuselage store and supply liquid hydrogen, routed within the pressurized area of the aircraft. Reproduced from Bielsky et al. (2024).

4.2.3 Fuel System Architecture Design Space

Having selected a reference aircraft, the framing of the architecture optimization problem for the aircraft's fuel system can be taken to the next step. This consists of defining the design space for said system.

The definition of this design space was influenced by various factors, such as available information, modeling capabilities and time constraints for implementation. Along with the ADSG of Figure 4.5, it is the culmination of the literature review and constitutes the final architecture design space of the SAO problem. It is presented in Figure 4.8 in the form of an ADSG. To facilitate understanding, a more transparent element signifies that it is not part of the optimization problem, but is kept for the sake of completeness in the description of the fuel system and its possible architectures.

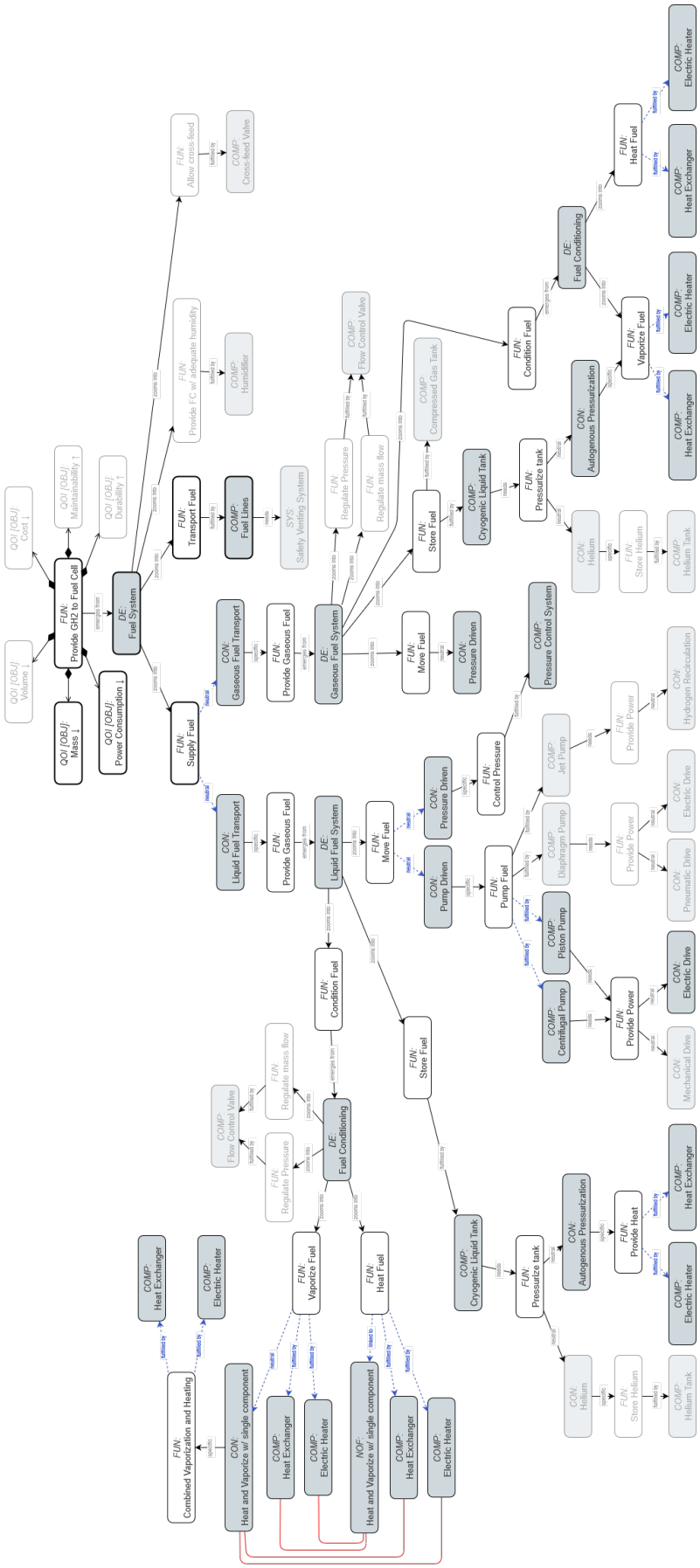


Figure 4.8: Architecture Design Space Graph (ADSG) for a hydrogen fuel system, modeled in ADORE. Elements with reduced visibility are not part of the optimization problem but are kept for completeness.

Although it is clear which aircraft system is being analyzed, the process of outlining the system boundary is not trivial. Inside this boundary are the elements considered to interact directly with the fuel and that are necessary for its transport from the tank to the fuel cell. Additionally, the elements needed for the provision of the fuel at the required thermodynamic state, namely pressure and temperature, are also included. Thus, some systems and components that influence the fuel system design are left out of the scope of the analysis. This includes the tank and most of the fuel cell balance of plant (e.g., the cooling system). Although it is indispensable to take these elements into account in the design of a real system, the decision to leave them outside the boundary was based on a number of factors, including the increase in problem complexity and time constraints for the development of suitable models.

The boundary function "Provide GH2 to Fuel Cell" is associated with two active performance metrics: mass and electric power consumption, both of which should be minimized. An additional four relevant quantities of interest are identified, but left out of the problem due to the lack of available means for their quantification. The aforementioned metrics are volume, cost, durability and maintainability.

Two main types of fuel system emerge at the first architectural decision: liquid and gaseous. The type of fuel storage is not subject to an architectural decision, as liquid storage is seen as the sole viable option. The same is the case for the type of drive in a gaseous system, with a pressure driven system being the single possibility. For the liquid system, the type of drive constitutes another architectural choice. It emerges in the fulfillment of the "Move Fuel" function, calling for selection between the concepts of a pressure driven and pump driven system.

The fuel lines component is subject to an architectural decision at the attribute level, as can be seen in Figure 4.9. Line insulation can be foam-based or vacuum-jacketed.

For a pump driven system, four possible types of pump are represented, as well as the respective drive types. However, only the centrifugal and piston pump variants are modeled and incorporated in the design study, since they are considered to be the most likely to be integrated in the first generation of passenger hydrogen aircraft (Turner et al. 2022). These low-pressure boost pumps are assumed to be located in the vicinity of the fuel tanks, which, as was indicated in the previous section, are in the aft fuselage. Due to the lack of a clear option for the extraction of mechanical power in this region of the aircraft, only an electrical drive is considered in this case. Potential future work could consist of the inclusion of models

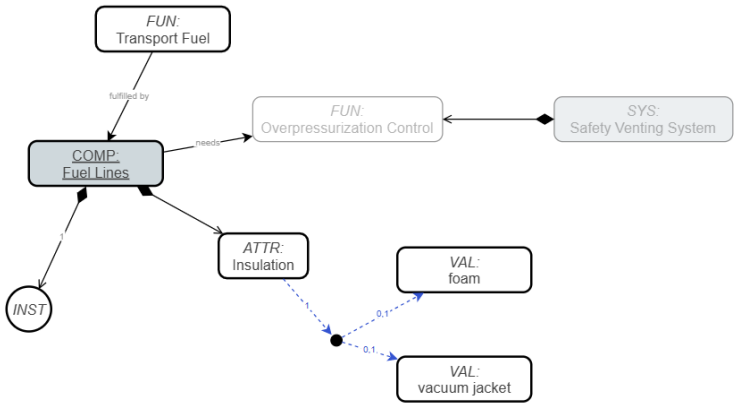


Figure 4.9: The attribute "Insulation" of the "Fuel Lines" component can take two values: foam and vacuum jacket. This is subject to an architectural decision.

for the diaphragm and jet pump.

Both liquid and gaseous systems induce the tank pressurization function. This function can be fulfilled by two distinct concepts: autogenous pressurization and helium injection (Millis et al. 2009). In this study, autogenous pressurization is exclusively considered, simply due to the lack of available means and time constraints for the development of a helium pressurization model. The vaporization of the fuel can then be carried out using either an electric heater or a heat exchanger. Like in the case of the pumps, the inclusion of helium injection in the design problem could be a relevant topic for future developments.

Lastly, the conditioning of the fuel shall be addressed. In this context, fuel conditioning refers to the necessary steps to bring the fuel to the right thermodynamic state for feeding to the fuel cell. This can essentially be broken down into two functions: the vaporization of the fuel and the further heating necessary to reach the required fuel cell inlet temperatures.

In the case of the liquid fuel system, these two functions can either be performed separately, using different components, or together in a single component. In each case, the choice is always between an electric heater and a heat exchanger. The red lines in the graph connect incompatible components and are thus denominated incompatibility constraints. For example, if for the fulfillment of the "Vaporize Fuel" function, the concept of vaporization and heating with a single component is chosen, then the only feasible choice for the "Heat Fuel" function is the analogous non-fulfillment (NOF) block (Bussemaker et al. 2020). Therefore, no additional component is used to fulfill this function.

In the gaseous fuel system, vaporization is carried out directly at the tank outlet. For this reason, assuming that further heating is only done inside the propulsion pod, both functions cannot be performed by the same component. In this case, however, the vaporization of fuel for tank pressurization can be done with the same component as for the fuel being routed to the fuel cells. This is assuming that the necessary mass flow is routed back into the tank in a separate line.

A number of components and subsystems such as valves, humidifier and venting system are included in the graph with the intent of representing, with as much completeness as possible, all functions and components required in a fuel system. They are, however, not included in the optimization problem. This is left for subsequent work, mostly due to time constraints.

4.3 Design & Analysis Disciplines

In order to design the fuel system, various disciplinary sizing models are needed. As a starting point, a preexisting liquid fuel system sizing tool, developed for a specific architecture, was taken. From this tool, models for the sizing of a heat exchanger, electric heater, fuel lines and centrifugal boost pump were extracted and transformed into individual component sizing tools.

Additionally, a piston pump sizing tool was developed in its entirety and the functionality of using vacuum jacketed insulation was added to the fuel lines tool, which previously only included foam insulation. Several smaller alterations were also made to the previously existing models, for example to accommodate for the fact that both liquid or gaseous fuel systems are a possibility.

The previously mentioned disciplines are introduced next, with reference to inputs and outputs, as well as the employed models.

4.3.1 Centrifugal Boost Pump

The centrifugal boost pump discipline sizes a three-stage, variable-speed centrifugal pump driven by a 270-V DC electric motor, based on a reference pump (Brewer 1991, p. 105).

The discipline takes as input the fuel mass flow, the tank pressure, the required fuel cell inlet pressure and, if available, the pressure loss along the fuel lines. Pump parameters such as efficiency and rotational speed (rpm) are obtained from pump characteristic curves like the one of Figure 4.10 and affinity laws as presented by Moran (2016). Based on these, the effective pressure rise provided by the pump is calculated.

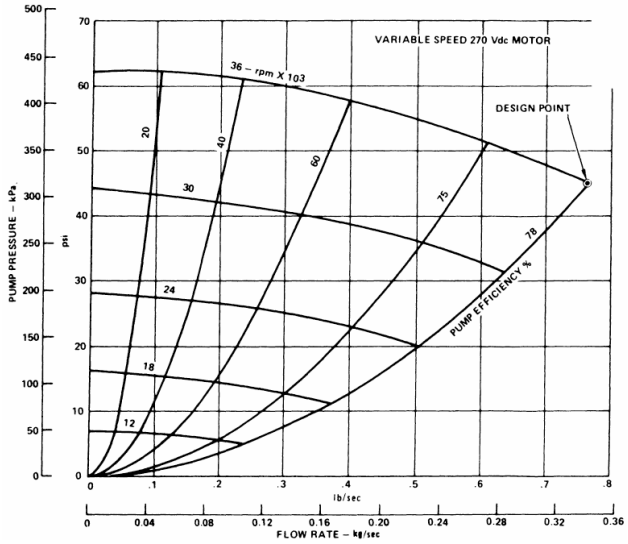


Figure 4.10: Boost pump characteristics. Reproduced from Brewer (1991).

The rise in enthalpy provoked by the pump (Δh_p) is quantified using the following equation, expressed in imperial units (Brewer 1991, p. 129):

$$\Delta h_p = \Delta P \cdot \left[\frac{1}{\rho_f} \left(\frac{1}{\eta_p} - 1 \right) + 0.0175 \right] \tag{4.2}$$

where ΔP corresponds to the pump pressure rise, ρ_f is fuel density, and η_p is pump efficiency.

The efficiency of the motor is obtained by use of data on the transient performance of the reference boost pump (Brewer 1991, p. 115), whereas the mass of the motor assembly is obtained as a function of its power and speed (Brewer & Morris 1978, p 124).

As output, the discipline provides the state of the fuel at pump exit (pressure, temperature and enthalpy), as well as the total mass, including electric motor and frequency converter, and electric power consumption. The pump's mass is calculated on the basis of its dimensions and the employed material, whereas the power consumption is a function of volumetric fuel flow, pressure rise and pump efficiency.

4.3.2 Piston Boost Pump

The piston pump discipline uses as reference a low-speed, 5 cylinder "bucket-type" piston pump designed to handle boiling hydrogen, first presented by Biermann & Kohl (1959). The performance of the pump is further detailed by Biermann & Shinko (1960). Its design specifications are shown in Figure 4.11.

Bore and stroke, in.	3 × 1.5
Total displacement, cu. in.	53.03
Design speed, rpm	240
Displacement at design speed, gal/min	55.1
Design inlet conditions	Saturated liquid with zero head
Discharge pressure, lb/sq in. gage	135
Average piston speed, ft/sec	1.0
Lubrication	Entire pump submerged in pumped fluid
Total piston clearance, diametrical, in.	0.002
Operating temperature, °F	-430

Figure 4.11: Reference piston pump design specifications. Reproduced from Biermann & Kohl (1959).

Based on the performance of the reference pump, a conservative overall efficiency of 0.8 is assumed for the sized pump. The design speed is fixed to that of the reference, at 240 rpm.

The inputs to the discipline are the same as those of the centrifugal pump discipline. With these and with the assumptions listed above, the brake power (BP) and torque (T) are calculated according to the following expressions:

$$BP = \Delta P \cdot \frac{\dot{m}_f}{\rho_f} \tag{4.3}$$

$$T = \frac{BP}{\omega} \tag{4.4}$$

where ΔP is the gauge pressure, $\dot{m}_f$ is the fuel mass flow, ρ_f is the fuel density and ω is the angular rotational frequency.

The required cylinder radius, r, is then calculated by:

$$r = \sqrt[3]{\frac{T}{N_c \cdot \pi \cdot \Delta P}} \tag{4.5}$$

with N_c being the number of cylinders, which in this case is 5.

The complete dimensioning of the pump, necessary for mass estimation, is then done by maintaining proportionality, using the ratio between the obtained cylinder radius and that of the reference pump. Its mass is then determined by the total volume and material density. The electric power consumption is computed by dividing the brake power by the efficiencies of the pump and electric motor.

Apart from mass, power consumption and pump design specifications, the discipline also outputs the fuel state at pump exit, which is then used as an input for the next executed tool.

4.3.3 Fuel Lines

This discipline sizes the main fuel line and what is referred to as the recirculation line. This line exists such that more fuel is always provided than is needed, to prevent situations of possible fuel starvation. To ensure this, it circulates the excess fuel back to the tank. Additionally, the mass flow required for the pressurization line is also calculated. This is the line that extracts liquid fuel from the tank to vaporize for tank pressurization. This line is not sized, however, as it is assumed to be very short and thus the mass is assumed to be negligible when compared to that of the other piping.

The inputs to this disciplinary tool include the type of insulation used for the given architecture, the fuel mass flow, an array of flight altitudes taken from the aircraft mission trajectory, the thermodynamic state of the fuel at the entry to the line, the type of fuel system (liquid or gaseous), and the geometric positions of engines and tank from the aircraft model.

The length of the sized lines is calculated based on the positions of the engines and the tanks. Then, with the given mass flow, the pipe diameter is sized in accordance with maximum allowable flow speeds, the values of which depend on whether the flow is liquid or gaseous. Knowing the length and diameter of the line, the pressure losses are obtained using the *Darcy-Weisbach* equation.

The change in specific enthalpy (Δh) of the fuel between both ends of the line is calculated by:

$$\Delta h = \frac{\dot{q} \cdot L}{\dot{m}_f} \tag{4.6}$$

where $\dot{q}$ is the heat flux per unit length, L is the length of the line and $\dot{m}_f$ is the fuel mass flow.

The method used to determine the heat flux depends on the type of insulation chosen. For foam insulation, it is done through a heat transfer analysis for multilayered cylinders (Çengel 2003, p. 148-153). In the case of vacuum jacket insulation, an estimated value of around 1.8 W/m is taken from Brewer (1991), as seen in Figure 4.12, which is evaluated for a line with an inner diameter of 1 in. This estimate is therefore conservative for smaller diameters, which is generally the case in the problem at hand.

The configuration of the fuel line is important both for the heat flux calculations as well as for mass estimation. With foam insulation, it corresponds to the selected design from the

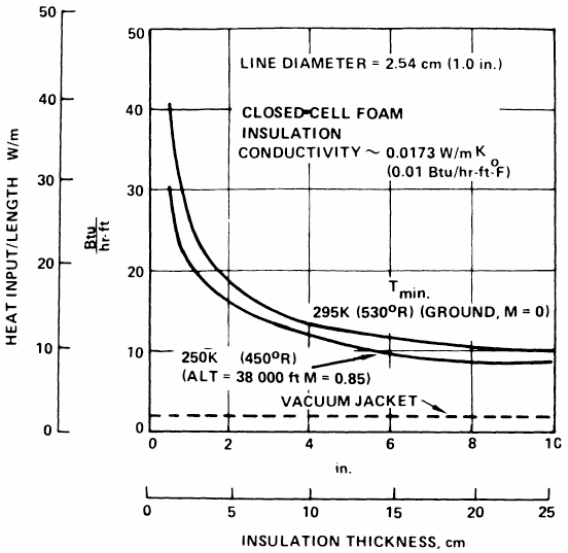


Figure 4.12: Fuel line heat leak for foam and vacuum insulation. Reproduced from Brewer (1991).

Lockheed investigations (Brewer 1991), depicted in Figure 3.8, consisting of an inner line of stainless steel and an outer line of aluminium, both with a thickness of 0.406 mm. Between the two, 38.1 mm of polyurethane foam is used. The vacuum-jacketed line retains the same inner and outer layers, the annular space between them being composed of alternate layers of fiberglass fabric and aluminized mylar, ten of each, both with a thickness in the vicinity of 10^{-1} mm.

Among the many outputs of the discipline, it is worth mentioning that the calculated pressure loss of the fuel line is used as an input to the pump sizing disciplines, creating a feedback that then requires a convergence loop to resolve.

4.3.4 Electric Heater

The electric heater discipline takes as input the thermodynamic state of the fuel at entry, the fuel mass flow, and the intended function of the heater. The intended function depicts whether the heater is meant to vaporize the fuel, heat it (further heating after vaporization) or both, referred to as conditioning.

Depending on the function, the required increase in enthalpy is obtained, with which the electric power consumption (P) of the heater is calculated:

$$P = \frac{\dot{m}_f \cdot \Delta h}{\eta_h} \tag{4.7}$$

with $\dot{m}_f$ being the fuel mass flow, Δh being the intended enthalpy rise and η_h the efficiency of the heater.

The sizing and subsequent mass calculations are performed through the scaling of a reference heater. Furthermore, the discipline's outputs are the fuel exit state, as well as the mass, volume and electric power consumption of the electric heater.

4.3.5 Heat Exchanger

The present discipline sizes a "shell and tube" type heat exchanger using the mean temperature difference (MTD) method (Shah & Sekulić 2003, p. 186-190, 291-298). The hydrogen fuel flows in the tubes, whereas the shell is occupied by a 50% volume ethylene glycol solution. This coolant fluid is assumed to carry excess heat extracted either from a fuel cell or a different aircraft system.

Several assumptions are made in the implementation of this relatively complex model, the most relevant of which are stated below.

- Steady-state flow.
- Constant fluid and material properties.
- No heat is lost to the surroundings.
- Thin-walled tubes (no thermal resistance).

Inputs to the discipline are the function of the heater, the design driving fuel mass flow, as in the maximum mass flow expected in the aircraft mission, and the thermodynamic state of the fuel at the inlet.

Depending on the function of the heat exchanger (vaporization, heating or conditioning), the required exit state of the fuel is determined. The heat exchanger is sized and its mass is estimated in accordance with the dimensions of the shell, tubes and baffles, and with the material used, which in this case is aluminium.

The main outputs of the discipline are the pressure losses within the heat exchanger, its mass and volume, as well as the fuel exit state, namely temperature, pressure, and enthalpy.

4.3.6 Objective Functions

Two disciplines are included in the problem, which quantify the performance metrics of the system. These are the *System Mass* and *Power Consumption* disciplines. They constitute the objective functions of the optimization problem.

The system mass discipline receives as input the masses of each individual component in the fuel system and outputs the total system mass, whereas the power consumption discipline evaluates, in an analogous manner, the total electric power consumption of the fuel system.

4.4 Dynamic MDAO Problem

In order to carry out the system architecture optimization, the problem must be put together in the context of dynamic MDAO, being represented in the form of an XDSM (Section 4.4.1), and implemented in MDAx (Section 4.4.2).

4.4.1 Optimization Problem Formulation

Formally described, the system architecture optimization problem consists of the minimization of two objective functions: mass and electric power consumption. This is done with respect to ten architectural decisions encoded as discrete design variables and an additional design variable, the initial tank pressure. The value of the additional design variable is dependent on the "pump vs pressure driven" architectural decision. Given to the problem are the aircraft geometric configuration, mission and performance information.

The design space, illustrated in Figure 4.8, is composed of 2304 possible architectures, of which 80 are valid. This is only considering discrete design variables, the number of possible designs being much higher if one were to take into account the continuous component sizing design variables (Bussemaker et al. 2023).

The MDO problem formulation is shown in Figure 4.13 in the form of an XDSM. The employed notation, introduced by Garg et al. (2024b), is slightly different from that of the typical XDSM. It is adapted to dynamic MDO problems in order to represent architectural influences.

The variables exchanged between disciplines are expressed by the name of their respective XML nodes in the workflow XML file, which follows the CPACS format. Special emphasis is given to "stateInfo". Unlike all other variables, it represents a shared section of the file where the thermodynamic state of the fuel is stored after the execution of each discipline. Thus, this element includes three child nodes named `pressure`, `temperature`, and `enthalpy`, as seen in Figure 4.14. After the execution of each discipline, these variables are updated and retrieved by the next discipline. This method is used because in an MDO problem of this kind, there is no predefined and constant order in which disciplines are executed. Because of this, the disciplines themselves cannot be coded to get the required fuel state from the outputs of any specific discipline, needing instead a common location used by all.

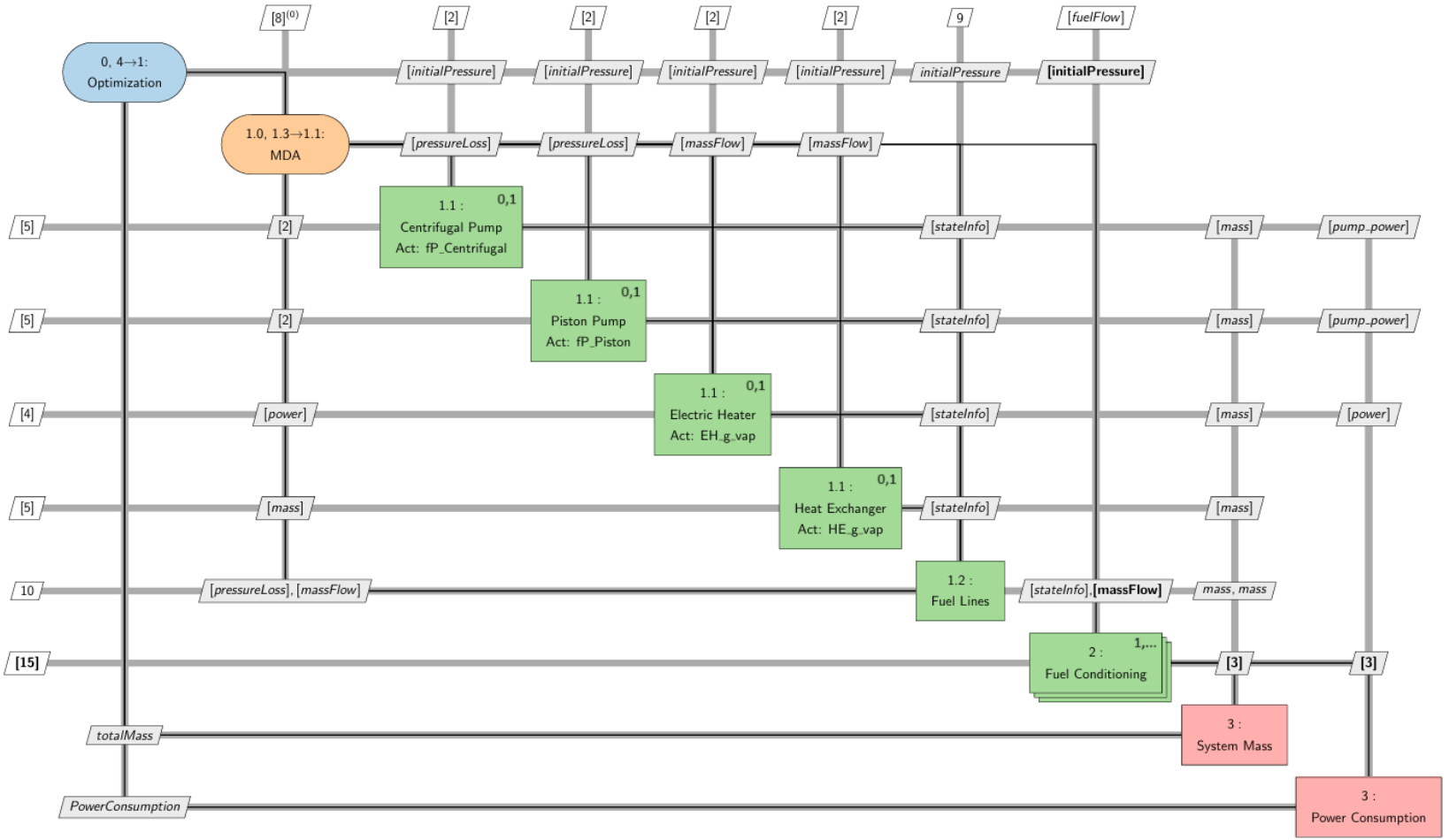


Figure 4.13: XDSM of the hydrogen fuel system architecture optimization problem. The extended notation (Garg et al. 2024b) allows for the representation of the four architectural influences. If the number of executions of a discipline is dependent on an architectural decision, this is indicated in the top right corner of the block. *Conditional variables* are indicated in square brackets, whereas variables involved in *data connection* are expressed in bold and italic. Disciplines subject to *discipline activation* have the activation condition expressed in an additional row in the block. Finally, a stack of disciplinary blocks signifies *discipline repetition*.

```
<stateInfo> <!-- dynamic information on the fuel state as per last ran tool -->
    <pressure></pressure>
    <temperature></temperature>
    <enthalpy></enthalpy>
</stateInfo>
```

Figure 4.14: The "stateInfo" section of the CPACS XML file. It serves as a common location for the storage of the fuel thermodynamic state.

The execution of the first four disciplines in the workflow is mutually exclusive, which explains why they all have the same block numbering in the XDSM (Figure 4.13). The first two are associated with a liquid, pump-driven fuel system, whereas the last two are associated with a gaseous fuel system. Additionally, if the proposed system architecture is a liquid, pressure-driven system, then none of the disciplines numbered 1.1 are executed. The activation conditions, mentioned under the discipline name, refer to the existence of the XML node identifying the component in the workflow XML file.

The fuel lines discipline (1.2) is not associated with any architectural influences that affect its execution, being necessary for all architectures. Some of its outputs are, however, required as inputs to upstream tools. Specifically, the pressure loss in the lines is needed by the pump disciplines, and the mass flow of the pressurization line is needed by the electric heater and heat exchanger tools. Because of this, a feedback coupling ensues, calling for a multidisciplinary analysis to ensure consistency. For this, an MDA driver coordinates iterative analysis loops between these disciplines until convergence of the coupling variables is achieved. The driver requires initial values for these coupling variables, which are provided as external inputs.

After the MDA, the "Fuel Conditioning" block (2.0) is executed. This block does not represent a discipline, but rather what is referred to as a sub-workflow or nested workflow. In other words, it consists of a smaller workflow contained within the main workflow that is the complete SAO problem.

This workflow is subject to the architectural influence of discipline repetition. The number of repetitions is associated with the number of heating components in the system architecture, with exception to those in the 1.1 blocks. These are associated with the functions of vaporization and pressurization in the case of a gaseous fuel system. The number of repetitions can thus vary between 1 and 3.

An example of an architecture where only one repetition is needed is any gaseous fuel system.

Since the heater that jointly fulfills the functions of vaporization and pressurization is sized in 1.1, only one additional heater is needed to fulfill the "heating" function, defined as the further heating required for adequate fuel cell entry temperature. Contrarily, an example of an architecture where three repetitions are required is a liquid fuel system where the functions of vaporization and heating are performed by separate components. Then, a third component for pressurization is always necessary.

Illustrated in figure 4.15, the *Fuel Conditioning* sub-workflow is composed of two blocks, the electric heater and heat exchanger disciplines. These are never both executed within the same repetition. As discussed at the end of Chapter 2, whenever discipline repetition is used, different inputs are used for each iteration. Thus, for each heater in the architecture, the necessary information for its sizing is given as input to the nested workflow. Then, depending on the type of heater, the corresponding discipline is activated.

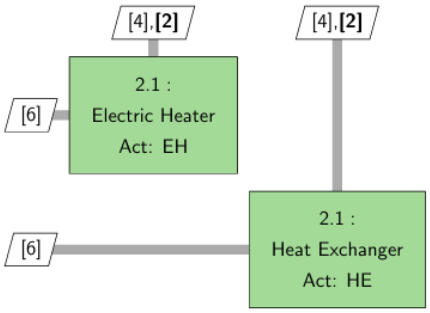


Figure 4.15: The "Fuel Conditioning" sub-workflow. Only one discipline is executed in each repetition. The activation condition is the existence of the respective component node in the XML file given as input to the nested workflow.

The utility of this nested workflow is clear when one contemplates what the main workflow would have to look like without it. Essentially, in the maximally distributed case, a different disciplinary block would have to exist for each of the three possibly present functions: vaporization, heating and pressurization[1]. Also, it would be required that the blocks for heating come after those for vaporization, as some of the outputs of the latter are inputs to the former. In addition, since each function can be performed by two possible components, two disciplines per function would have to be used: one for the electric heater and one for the heat exchanger. Thus, instead of one block in the main workflow, six blocks would need to be used

[1] an additional function exists, denominated "conditioning", which represents the fulfillment of both vaporization and heating functions in a single component. When it comes to workflow execution, however, this function can be bundled with vaporization, as the remaining function (pressurization) is independent of it, meaning there are no data dependencies between them.

in total. Furthermore, these blocks would not be unique, but rather three repetitions of the same two blocks. Figure 4.16 illustrates this alternative option. The use of a nested workflow in conjunction with discipline repetition allows for a much more elegant solution.

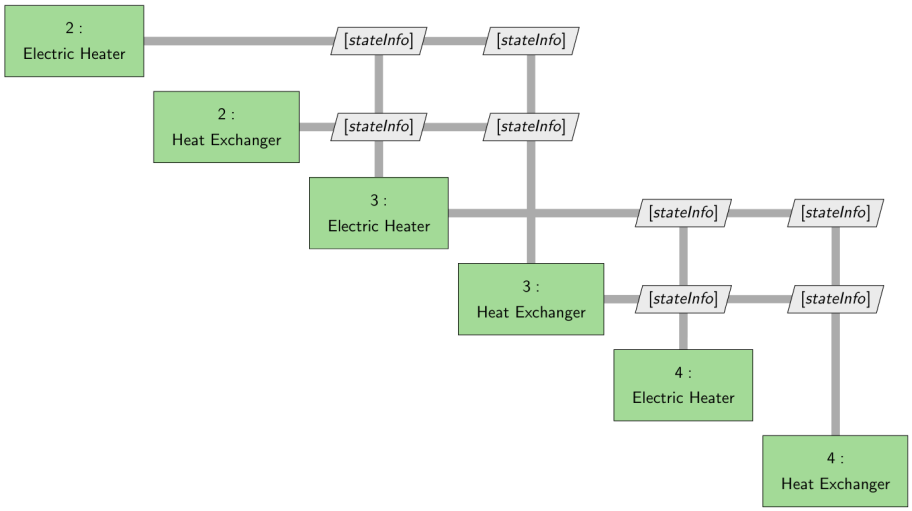


Figure 4.16: One of the alternatives to the sub-workflow would be the usage of two disciplinary blocks per function, one for each type of heater. Instead of one block in the main workflow, six blocks would be needed in total.

As an intermediate solution, one might think to use only two disciplinary blocks, one for each component, with discipline repetition. In any case, it would constitute a less elegant solution, as two disciplinary blocks would need to be used. But it could be seen as an alternative if for some reason the use of a nested workflow was not desired. The reason why this is not feasible is that all of the executions of one discipline would have to be carried out before the second discipline. Consider the example architecture of a liquid fuel system with a heat exchanger for vaporization and an electric heater for heating. Consider also that the electric heater disciplinary block with discipline repetition is upstream of the heat exchanger block in the workflow (as depicted in Figure 4.17). The vaporization step must be physically performed prior to heating. Thus, the heat exchanger for vaporization would be sized first, some of the outputs of which would then be passed on for the sizing of the electric heater used for the "heating" function. For this, the electric heater discipline would have to be skipped in order for the heat exchanger to be first sized. Then, there would be no way to return back upstream in the workflow for the sizing of the subsequent electric heater.

To conclude the description of the MDO problem formulation, attention is now directed to the objective function disciplines: System Mass and Power Consumption. These are not affected

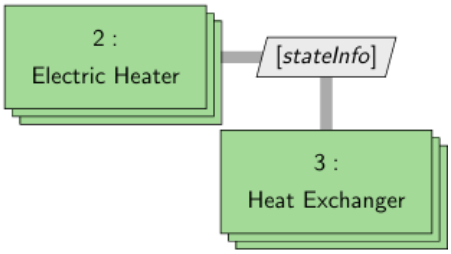


Figure 4.17: Representative XDSM of another potential alternative to the nested workflow. This concept would not be feasible, as some architectures would require going back upstream in the workflow.

by any architectural influence. They are also completely independent of each other and thus can be executed simultaneously, returning back to the optimizer the value of the performance metric variables that constitute the objectives of the optimization problem: the total mass and electric power consumption of the proposed hydrogen fuel system architecture.

4.4.2 Generation of the MDAx Workflow Export

As mentioned previously, MDAx does not yet support the formulation of dynamic MDAO problems in its GUI. For this reason, it must be carried out in the back-end.

The formulation consists first of creating the disciplines, which are each defined by an input file, an output file and a configuration file. After that, the workflow is defined by indicating the order in which the disciplines are to appear. Finally, the MDA driver is added by indicating the type of converger (in this case, a Gauss-Seidel converger) and the disciplines it wraps. Based on the inputs and outputs of each discipline, MDAx then automatically establishes data connections between them, applying collision resolution methods when ambiguities exist. An example is when a discipline requires as input a variable that is an output of more than one other discipline. Different collision resolution methods exist to resolve this ambiguity, such as taking the most recently available version of the said variable or establishing the connections in such a way that minimizes feedback in the workflow. Once all of this is carried out, a workflow file is exported which can then be run in a PIDO environment such as RCE.

Input & Output Files

The input and output files are simple XML files that follow the CPACS data format, just like the workflow XML file. However, instead of being thousands of lines long and containing all the information about the aircraft, these files contain only the elements that are inputs or outputs of the respective discipline. They can be generated manually or using MDAx's GUI,

which simplifies the definition of disciplinary inputs and outputs by taking advantage of its base schema. Then, these files can be exported and used in the back-end problem definition. As an example, the input and output files of the piston pump discipline are shown in Figure 4.18.

```
<cpacs>
    <vehicles>
        <aircraft>
            <model uID="AircraftModel">
                <systems>
                    <genericSystems>
                        <genericSystem uID="ata28_fuelSystem">
                            <components>
                                <component uID="engineDeliveryLine">
                                    <pressureLoss/>
                                </component>
                                <!-- ... -->
                            </components>
                        </genericSystem>
                    </genericSystems>
                </systems>
                <analyses>
                    <trajectories>
                        <trajectory uID="block_1000nm_7500kg">
                            <FlightPoints>
                                <FuelFlow/>
                            </FlightPoints>
                        </trajectory>
                    </trajectories>
                </analyses>
            </model>
        </aircraft>
    </vehicles>
    <!-- ... -->
</cpacs>
```

```
<cpacs>
    <vehicles>
        <aircraft>
            <model uID="AircraftModel">
                <systems>
                    <genericSystems>
                        <genericSystem uID="ata28_fuelSystem">
                            <components>
                                <component uID="fuelPump_Piston">
                                    <pump_power/>
                                    <pump_rpm/>
                                    <mass>
                                        <total/>
                                        <boostPump/>
                                        <boostMotor/>
                                        <freqConv/>
                                    </mass>
                                    <out_pressure/>
                                    <out_enthalpy/>
                                    <out_temp/>
                                </component>
                            </components>
                        </genericSystem>
                    </genericSystems>
                </systems>
            </model>
        </aircraft>
    </vehicles>
    <!-- ... -->
</cpacs>
```

Figure 4.18: The input (left) and output (right) files of the piston pump discipline, truncated for better understanding and visibility.

Introduced by Burschyk et al. (2025), the CPACS system definition schema is used, where aircraft subsystems are defined within a "`genericSystem`" node and uniquely identified by their "`uID`" attribute. In this case, the fuel system is identified by the `uID`: "`ata28_fuelSystem`". Within this section, all the system components are defined as a "`component`" element, identified by their `uID` attribute. The child nodes of this component are then the component parameters. As can be conferred in Figure 4.18, the piston pump tool takes as input (among others not shown) the pressure loss of the main fuel line and the fuel mass flow, taken from the aircraft "trajectories" section. As output, the discipline provides the piston pump's performance metrics: mass and power consumption; a design specification: rpm; and the fuel state at it's outlet, which, as mentioned previously, is written to the "stateInfo" section.

Configuration Files

The remaining file required for the definition of a discipline is the configuration file. An example of a configuration file has already been shown in Figure 4.2. It contains relevant information on the discipline name and version, but most of all on the architectural influences that affect it. Specifically, the assertion conditions that dictate exactly how the influence occurs are defined.

The activation of all disciplines numbered 1.1 in the problem XDSM (Figure 4.13) is depen-

dent on an activation assertion. Because of this, they contain the "activationLogic" key in their respective configuration files, as can be seen in the example of the Electric Heater discipline of Figure 4.19. The value of this key then consists of a Python class that checks for this assertion with the necessary inputs to the class. For all the mentioned disciplines, the employed class is "XPathExists()", which returns a true statement if the xpath given as argument exists in the workflow XML file. In this case, the xpath corresponds to that of the component node relevant to the respective discipline. Furthermore, since no other architectural influences are present in these tools, no other keys related to architectural influences are present in the configuration file.

```
1  {"activationLogic": "XPathExists('/cpacs/vehicles/aircraft/model[@uID=\"AircraftModel
   \"]/systems/genericSystems/genericSystem[@uID=\"ata28_fuelSystem\"
   ]/components/component[@uID=\"electricHeater_g_vap\"]')",
2   "executionName": "Electric Heater",
3   "name": "Electric Heater",
4   "notes": "",
5   "version": "1.0"
6  }
```

Figure 4.19: Configuration file of the Electric Heater tool in the main workflow. It is subject to discipline activation and the activation assertion is true if the electric heater for vaporization in a gaseous fuel system is part of the architecture.

The Fuel Lines tool and the function evaluation tools are not associated with any architectural influences and thus their configuration files consist solely of the naming and version information. Therefore, for the main workflow, this leaves the Fuel Conditioning sub-workflow, which is subject to discipline repetition. To describe its repetition assertion, a thorough analysis of how discipline repetition works is warranted.

Discipline repetition was conceived to be able to iterate through different instances of the same component type, differentiating them through their uID attribute. It is implemented such that the components to be iterated through must share the same element name. Also, the uID must contain the component instance number, this number being the only allowable difference in the uID string. The following example illustrates a group of components capable of being iterated through.

- <engine uID="engine_1">
- <engine uID="engine_2">
- <engine uID="engine_3">

By having such a structure, the "Global to Local" script, in RCE, can replace the instance number by the keyword "INDEX" for a different instance in each repetition. This signifies to the input filter that that specific instance is the one from which the inputs should be taken in that repetition. The "Local to Global" script changes the `uID` name back before the next iteration. So, to reiterate, in the first repetition, the `uID` of "engine_1" would be renamed to "engine_{INDEX}". In the second repetition, the same would happen for "engine_2", and so on.

Because of these strict naming requirements, the components intended to be iterated through in the Fuel Conditioning sub-workflow would have to suffer some alterations in the way they are presented in the workflow XML file. Whereas the other components are identified by their `uID`, such as "`fuelPump_Piston`" or "`engineDeliveryLine`", for example, the electric heaters and heat exchangers intended to be sized in the nested workflow would be attributed a generic `uID` of "`heater_{INDEX}`" (here "INDEX" simply represents the instance number of the component).

Since in these cases the `uID` does not identify the type of heater, an additional child node called "`type`" is added, where the value "EH" signifies that it is an electric heater, whereas "HE" constitutes a heat exchanger. Figure 4.20 shows an exemplary representation of one such component in the workflow XML file.

```
<component uID="heater_1" block="sub_wf">
  <name>heater_1</name>
  <description>functionalComponent</description>
  <systemElementUID></systemElementUID>
  <parentUID>engine</parentUID>
  <type>EH</type>
  <function>l_cond</function>
  <mass></mass>
  <volume></volume>
  <power></power>
  <pressureLoss></pressureLoss>
  <out_pressure></out_pressure>
  <out_enthalpy></out_enthalpy>
  <out_temp></out_temp>
</component>
```

Figure 4.20: Section of the workflow XML file corresponding to a fuel system component meant to be sized in the Fuel Conditioning sub-workflow. the "`type`" element indicates that it is an electric heater, whereas the "`function`" element informs that it is part of a liquid fuel system architecture, with the function of conditioning (vaporization and heating in the same component). The "`block`" attribute is used as a common attribute in order to count the number of instances for repetition.

Furthermore, an additional attribute, called "block", is included in the component element of these heater instances, the value of which is set to "sub-wf", indicating that they are meant to be processed in the sub-workflow. This attribute is added because the number of repetitions of a discipline is determined by the number of instances of a certain component in the workflow XML file. Thus, since all components are instantiated in an element called "component" and the uIDs of the heater elements must necessarily be different in order to iterate through their numbering, another attribute had to be included that uniquely identified the correct component instances. This is also conferrable in Figure 4.20, and becomes clearer when analyzing the repetition assertion in the configuration file of the Fuel Conditioning nested workflow.

Figure 4.21 shows the configuration file of the Fuel Conditioning sub-workflow. Discipline repetition is represented by the "repetition" key. The value of this key is the number of repetitions that ought to be carried out. This number can be either directly given as a hard-coded integer, or determined by the "NumberofInstances()" Python class. This class returns the number of component instances that exist in the XML file, which correspond to the xpath given as argument. So, in this case, the class will return the number of components inside the "ata28_fuelSystem" that have the "block" attribute.

```
1  {"repetition": "NumberofInstances('/cpacs/vehicles/aircraft/model[@uID=\"AircraftModel
   \"]/systems/genericSystems/genericSystem[@uID=\"ata28_fuelSystem\"]/components',
   'component[@block=\"sub_wf\"]')",
2   "executionName": "Fuel Conditioning",
3   "name": "Fuel Conditioning",
4   "notes": "",
5   "version": "1.0"
6  }
```

Figure 4.21: Configuration file of the Fuel Conditioning nested workflow. The number of repetitions is determined by the number of component instances that contain the "block" attribute.

During the creation of the main workflow in MDAx, the nested workflow is defined as any other discipline. The constitution of the sub-workflow itself is done through its creation as a separate workflow. Later, in RCE, the sub-workflow is integrated into the main workflow as a tool.

Within the sub-workflow, the Electric Heater and Heat Exchanger disciplines are also subject to discipline activation. Recall that a different heater element is provided as input for each repetition. To distinguish between the type of component, the strategy used for the disciplines in the main workflow (by uID) does not work, as the uID does not uniquely identify the component. Thus, the activation assertion has to be based on the contents of the element's type node, as illustrated in Figure 4.22.

```
{"activationLogic": "ElStrEq('/cpacs/vehicles/aircraft/model[@uID=\"AircraftModel\"
]/systems/genericSystems/genericSystem[@uID=\"ata28_fuelSystem\"
]/components/component[@block=\"sub_wf\"]/type','HE')",
 "executionName": "Heat Exchanger",
 "name": "Heat Exchanger",
 "notes": "",
 "version": "1.0"
}
```

Figure 4.22: Configuration file of the Heat Exchanger tool of the nested workflow. The activation assertion is true if the value of heater element's `type` node is "HE".

In the configuration file, the activation assertion is prescribed using the "`ElStrEq()`" Python class. This class takes two arguments, the first of which is the xpath of the node whose contents want to be checked. The second argument is the exact string that the evaluated node should contain in order for the assertion to be true. Figure 4.22 shows the configuration file of the Heat Exchanger tool in the nested workflow. Here, the discipline is activated if the value of the heater element's `type` node is the string "HE". In the case of the electric heater, the condition is analogous, but the checked value is "EH".

Once both workflow exports are generated (main and sub-workflow), they can be opened in RCE and integrated to form the total problem analysis workflow. The following section details the further work required to obtain the final optimization workflow.

4.5 Implementation in a PIDO Environment

In MDAx, one defines an analysis framework. The export of this framework is intended to be virtually ready for execution in a PIDO environment, provided that the required tools are already integrated. Essentially, the only thing that still needs to be done is to define the input files to the workflow and the location in which output files should eventually be stored.

However, the formulation of dynamic MDAO problems is a capability that is still in development within MDAx. Because of this, the workflow exports for these problems are generally not ready for immediate use, requiring alterations to best fit the intended problem-specific purpose. Moreover, one of the purposes of the current thesis work was in fact to identify potential improvement areas in MDAx's capabilities by applying them to a more realistic use case. The necessary alterations to the workflow are addressed in Section 4.5.1.

Then, if one seeks to solve an optimization problem, the optimization loop must still be integrated inside the PIDO environment. This is the subject of Section 4.5.2.

4.5.1 Analysis Workflow

This section is divided between the presentation of the main workflow and the nested workflow, starting with the higher-level one first.

Main Workflow

Figure 4.23 shows the part of the workflow related to the MDA convergence loop, as exported directly from MDAx to RCE. The MDA converger block is in the top left corner, followed by the four disciplinary tools whose activation is dependent on an architectural decision. Finally, in the bottom right corner, the Fuel Lines tool is present. This tool has a simpler structure around it, as it lacks the activation logic script that is needed for the other tools.

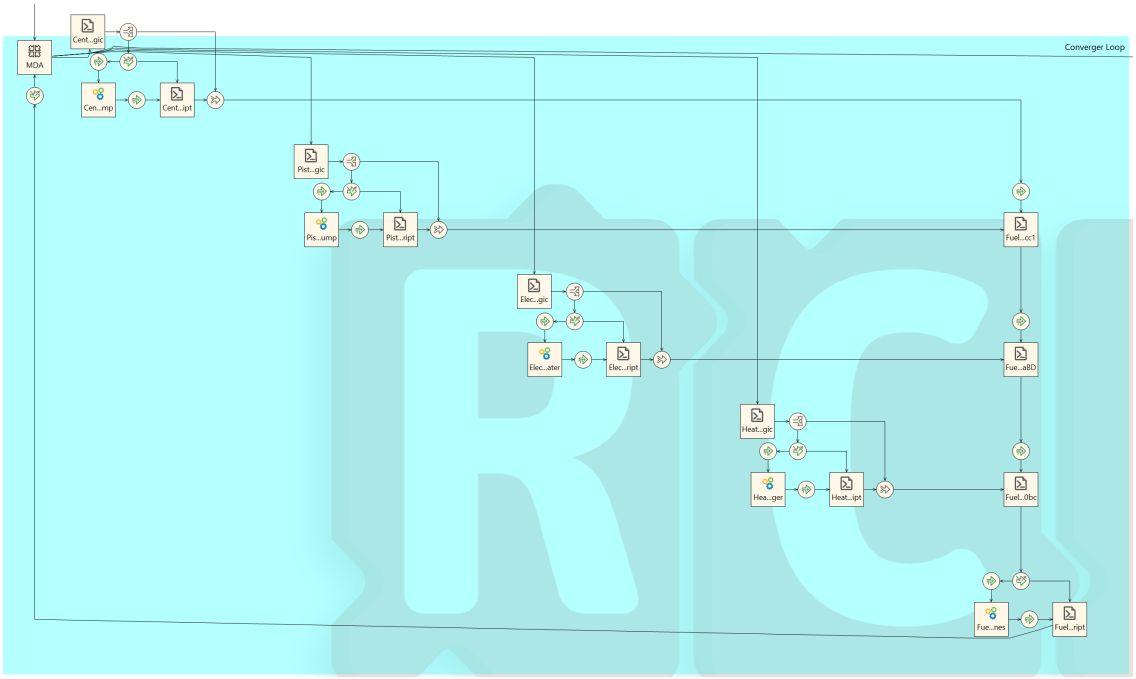


Figure 4.23: The MDA convergence loop part of the workflow, opened in RCE as exported from MDAx.

The MDA loop is one of the areas where collisions occur. Recall from Chapter 2 that collisions arise, for example, when multiple tools output the same variable, which in turn is an input to another tool. This is the case here, as the thermodynamic state of the fuel is required as input to the Fuel Lines tool. The tool from which this information comes from depends on the architecture being analyzed.

Because of this, the Fuel Lines tool has three *Merge Scripts* above it. These constitute the default collision resolution method employed in MDAx. As can be seen in Figure 4.23, these

scripts have two inputs, one coming from the left and one from the top. The inputs are two distinct XML workflow files, coming from two different tools. The Merge Script then merges the two files, with one of them having priority over the other. This means that when the value of a variable differs between the two files, the value taken is the one from the prioritized file.

The selection of which file to prioritize is fixed and cannot change from iteration to iteration. This constituted an issue, since sometimes the contents in the "stateInfo" section of the XML file would be deleted. As mentioned before, this section stores the thermodynamic state of the fuel. So, there would be cases where one input file, coming from the tool that was actually activated, would have this section populated, and the other file, coming from a tool that was skipped, would have it empty. If the latter file was predefined as the prioritized file, this information would be lost, causing the analysis to fail.

In addition, since the execution of all the tools upstream of the Fuel Lines tool is mutually exclusive, only an apparent collision exists. Because only one of the tools is used for the evaluation of each architecture, there is actually no ambiguity present. The non-activated tools are simply skipped and thus have no influence on the workflow. Because of this, the elements used for collision resolution can be removed. The result is then the MDA loop of Figure 4.24.

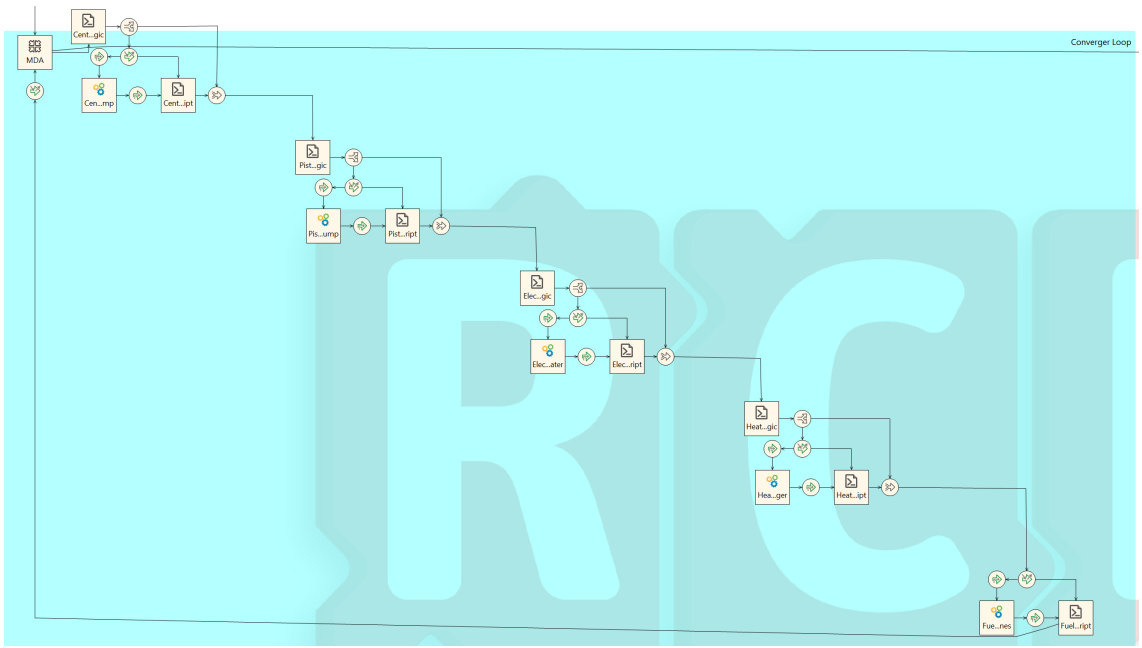


Figure 4.24: The MDA convergence loop part of the workflow, after manual modifications to remove elements related to collision resolution.

Directly after the MDA loop and before the objective function tools comes the Fuel Conditioning tool, which in fact is not a tool, but a nested workflow. Depicted in Figure 4.25, it is surrounded by additional scripts due to the fact that discipline repetition is applied. Thus, the Global to Local, Local to Global, and Iterator scripts are needed beyond the usual merge script present for all tools. The purpose of each of these scripts is described in Chapter 2 and summarized in Figure 2.21.

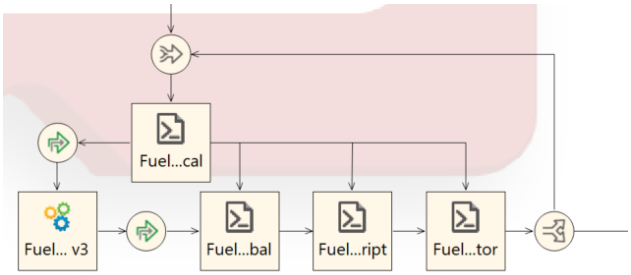


Figure 4.25: The fuel conditioning sub-workflow as part of the main workflow, represented in the bottom left corner. The required scripts are, from top to bottom and left to right, the Global to Local, Local to Global, Merger, and Iterator scripts.

Another complication encountered was that whenever the outputs of a tool were not necessary to any other tool in the workflow, this tool would be directly connected to an output writer. This was the case with the two objective function tools, namely System Mass and Power Consumption. Since the outputs of these tools are not required anywhere else, the workflow has two output writers. Thus, the analysis outputs two files instead of one, each missing information on one of the performance metrics.

The ideal behavior from MDAx would be, in these cases, to merge the outputs of such tools using a merge script, connecting then to a single output writer. Then, a single output file containing all relevant information would be the product of the analysis, as is intended. Since this is not the case, the mentioned alterations had to be made manually in RCE. The results are shown in Figure 4.26.

The integration of the Electric Heater and Heat Exchanger tools in the sub-workflow is depicted next.

Nested Workflow

As portrayed in Figure 4.15, the fuel conditioning sub-workflow consists of two disciplines that do not have any data dependencies. Essentially, they are not connected in any way, which stems from the fact that their activation conditions are mutually exclusive.

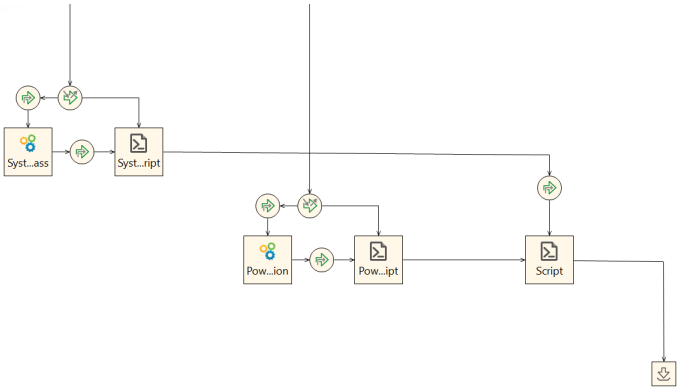


Figure 4.26: The final section of the analysis workflow. It consists of the System Mass tool, followed by the Power Consumption tool. The outputs of both are then merged into a single XML file and sent to an output writer (bottom right). In the original MDAx export, both tools connect directly to different output writers, with no merge script present.

MDAx does not support the creation of dynamic MDAO workflows of this kind. Thus, each discipline has to be created as a separate workflow consisting of only one tool. Then, the two workflows are combined in RCE to form the total Fuel Conditioning workflow, shown in Figure 4.27. In this case, the tools are directly connected. This is not an issue since one of the tools will always be skipped, and therefore will not modify the final output.

The workflow is then integrated into the main workflow as a tool, which is why the input provider and the output writer are deactivated. This is simply a particularity in the integration of sub-workflows in RCE.

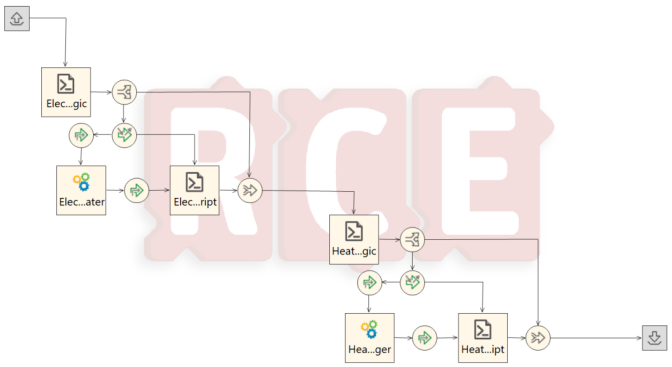


Figure 4.27: The Fuel Conditioning sub-workflow consists of the Electric Heater tool, followed by the Heat Exchanger tool. It is integrated into the main workflow, therefore the input provider and output writer are deactivated.

4.5.2 Integration of the Optimization Loop

With the analysis workflow fully formulated, the last step required in order to be able to conduct system architecture optimization (SAO) studies is the integration of the optimization loop. This consists of adding a set of RCE components to the workflow, which are predefined as the standard optimization template for use in SAO problems. As can be seen in Figure 4.28, this template is composed of an input provider, a converger denominated "SAO Loop", the AdoreOpt tool, a joiner and a switch component.

In addition, to integrate the optimization loop, the creation of the NFE base file is necessary. In this case, an additional script is required to compensate for specific shortcomings of the NFE functionality. Visible in Figure 4.28, this script is at the interface between the set of components required for optimization and the analysis workflow. A further description of this script is given on page 82.

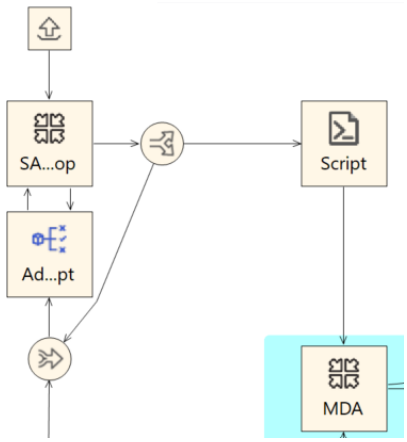


Figure 4.28: The section of the complete SAO workflow which is related to the optimization, with exception of the MDA driver. This driver is fed architecture instances for analysis and the AdoreOpt tool receives and compares the performance of said architectures.

The NFE base file is what will eventually be the workflow XML file. It contains the complete description of the reference aircraft in CPACS format, yet the section of the file pertaining to the fuel system only contains the components that will always be part of the architecture. This corresponds only to the fuel lines. In each iteration, AdoreOpt then adds the rest of the components in accordance with the architecture proposed by the optimizer.

This is possible through the use of the Node Factory Evaluator (NFE), as introduced previously in this chapter. For this, the base file contains a section where node and metric facto-

ries are defined. Node factories determine how architecture elements, like components, are mapped onto XML nodes. These are then added to the workflow file. Metric factories define where values for performance metrics can be read from the file, after the analysis is complete (Bussemaker 2025).

Figure 4.29 shows an excerpt of the developed NFE node and metric factory definitions. The xpath to the factories is `cpacs/nfeSettings`, above which the entire definition of the aircraft is stored. In the image, a node factory and a metric factory are outlined. The node factory refers to the Piston Pump component, and consists firstly of an element definition, which connects the architecture element with the factory. Next, the xpath where the node shall be created is given, followed by the tag of the created node. Attributes can then be defined, such as the `uID` in this case. Finally, the contents of the node are defined, which consist of child nodes and node values.

Essentially, what this node factory expresses is that if the Piston Pump component is present in the architecture, a node called `component` shall be created in the given xpath, with the given `uID` and contents.

The metric factory associates one of the quantities of interest of the architecture with the location in the file from which its value should be extracted. In this manner, the AdoreOpt tool can associate the architectures it provides with the performance indicators coming from the analysis of said architectures.

However, the NFE methodology does not suffice for the specific problem at hand because of the discipline repetition that occurs in the nested workflow. This is due to the conjugation of the two following facts:

- NFE translation associates specific architecture elements with specific XML nodes and respective attributes.
- The capabilities installed in discipline repetition do not allow for the skipping of element indices nor starting the iteration with an element index different to 1.

This is better understood with an example. In the context of the components meant to be sized in the Fuel Conditioning sub-workflow, the case may arise where a proposed architecture contains only the components associated with the `uID`s `heater_1` and `heater_3`. Another similar situation is when only `heater_2` and `heater_3` are present. Both of these cases would cause the discipline repetition to fail, as it is neither equipped to skip indices nor to start

```
<cpacs>
    <!-- ... -->
    <nfeSettings>
        <factories>
            <factory><!-- piston pump component -->
                <element><name>Piston Pump</name><type>COMPONENT</type></element>
                <xpath>
                /cpacs/vehicles/aircraft/model/systems/genericSystems/genericSystem[@u
                ID="ata28_fuelSystem"]/components</xpath><tag>component</tag>
                <attributes>
                    <uID>fuelPump_Piston</uID>
                </attributes>
                <contents>
                    <name>fuelPump_Piston</name>
                    <description>functionalComponent</description>
                    <systemElementUID>fuelPump_2</systemElementUID>
                    <parentUID>fuselage</parentUID>
                    <transformation>
                        <!-- ... -->
                    </transformation>
                    <mass>
                        <!-- ... -->
                    </mass>
                    <pump_power/>
                    <pump_rpm/>
                    <out_pressure/>
                    <out_enthalpy/>
                    <out_temp/>
                </contents>
            </factory>
        </factories>

        <metrics>
            <metric><!-- System Mass -->
                <qoi><name>Mass</name></qoi>
                <xpath>
                /cpacs/vehicles/aircraft/model/systems/genericSystems/genericSystem[@u
                ID="ata28_fuelSystem"]/mass</xpath>
            </metric>
        </metrics>
    </nfeSettings>
</cpacs>
```

Figure 4.29: A section of the NFE base file, portraying the definition of a node factory and a metric factory. Both are essentially used to create a mapping between the ADORE data model and the used CDS (in this case, CPACS).

at an index different to 1.

Because of this, the heater component `uID`s need to be reordered before the analysis of each architecture. Due to what is mentioned in the first bullet point above, this required reordering cannot be done directly through an NFE specification. The solution is then to add an additional script to the workflow for this purpose. This additional script simply changes the component `uID` numbering such that both of the issues previously mentioned do not occur. For example, if the proposed architecture contains the components associated with the `heater_2` and `heater_3` `uID`s, these are simply renamed to `heater_1` and `heater_2`. This additional script is simply denominated as "Script" in the RCE workflow, as can be conferred in Figure 4.28.

The topics addressed in this section constituted the integration of the optimization loop in what was previously a workflow capable only of analysis. Having concluded this, the result is the final RCE workflow for the multidisciplinary optimization of hydrogen fuel system architectures for a regional short-range aircraft.

Shown in Figure 4.30, the workflow wraps an optimization loop around the previously presented analysis workflow consisting of the MDA convergence loop, the Fuel Conditioning subworkflow and the two Objective Function tools.

In its presented state, the workflow is fully capable of performing optimization studies with the intention of finding promising architecture candidates to be integrated in the final design of the aircraft. This is the main topic of the next chapter.

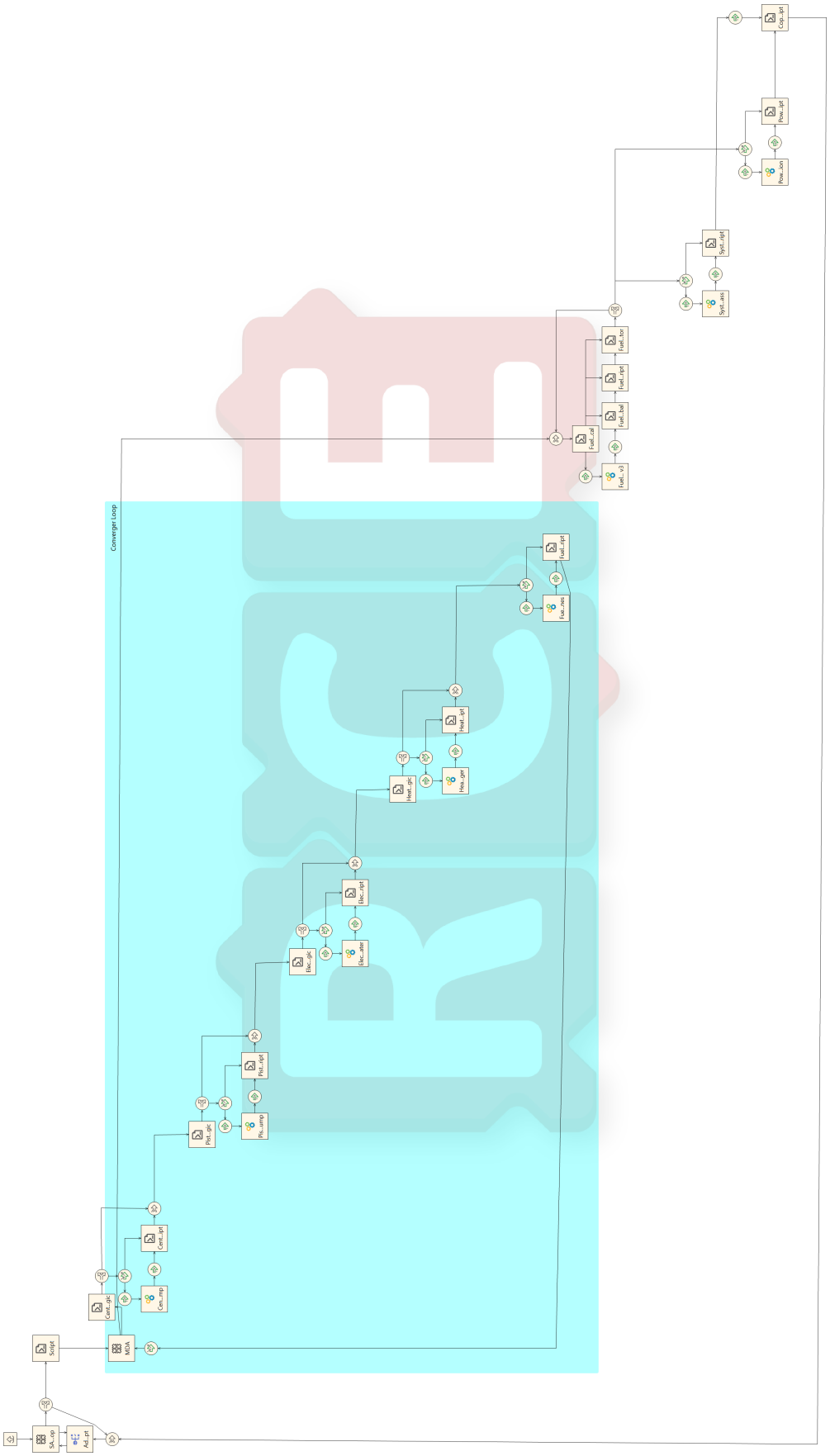


Figure 4.30: The complete RCE workflow for the multidisciplinary optimization of hydrogen fuel system architectures for a regional short-range aircraft.

5 Results & Discussion

In this chapter, the previously presented SAO framework is applied to the design of a hydrogen fuel system for a regional short-range aircraft with distributed propulsion. In Section 5.1, the results are presented and discussed, including the analysis of the optimal architecture and a comparison of two different optimization algorithms. In Section 5.2, the efficiency of the employed methodology is evaluated by comparing it with two different approaches with lower degrees of automation. Its utility is quantified by the required time to implement the method and generate results, highlighting the potential for time saving.

5.1 System Architecture Optimization Results

This section showcases the results of the SAO study. As the design space is small and evaluation times are short, it is possible to run a design of experiments (DOE) that covers all feasible architectures. This is presented in Section 5.1.1, accompanied by a discussion of the conclusions drawn. Section 5.1.2 then compares the performance of two different optimization strategies, one based on an evolutionary algorithm and another based on the use of surrogate models.

5.1.1 Design of Experiments

As mentioned in Section 4.4.1, the architecture design space is composed of 2304 possible architectures, of which 80 are valid. In the used SAO framework, invalid architectures do not reach the architecture evaluation step, as can be recalled from Figure 2.11. This means that the complete valid design space of 80 architectures can be evaluated in 80 iterations. The combination of this relatively small design space and the relatively short evaluation time for each architecture allows for the conduction of a design of experiments (DOE). The DOE consists of evaluating all valid architectures, one after the other, in an automated manner which is conducted by the AdoreOpt tool. The results are shown in Figure 5.1. It should be noted that the results obtained are for one side of the aircraft. The total values for mass and electric power consumption would then be double what is shown.

What is immediately noticeable is the lack of a Pareto front. Instead, an optimal architecture is found which is superior to all others regarding both objectives. This signifies that the two objectives are not in conflict with each other. In other words, configurations with low power consumption are also associated with a lower mass. The reasons for this are discussed throughout this section, but it shall be noted that this occurs specifically within the boundaries

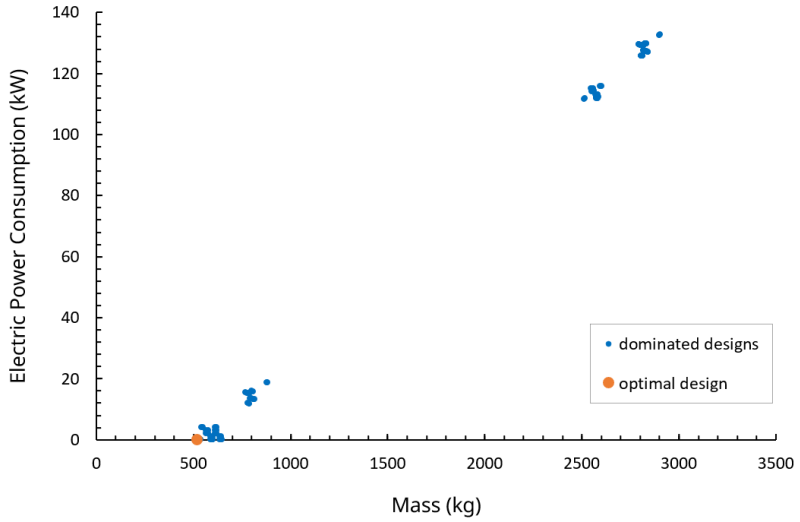


Figure 5.1: Results of the DOE. Each point represents a valid and evaluated architecture, which is associated with a certain mass (x-axis) and electric power consumption (y-axis). The optimal design is shown in orange.

of the problem and the assumptions made. It is expected that an expansion of the system boundaries and the use of different assumptions (such as the specific power of heaters) could result in a Pareto front of architectures.

What is also visible is that architectures with little to no electric power consumption seem to be possible. In reality, this is again related to the system boundaries that were defined for the problem. Within the scope of the problem, there are two main components that consume electric power: fuel pumps and electric heaters. This means that, for example, an architecture with a pressure driven system that uses only heat exchangers as a heat transfer component does not consume electric power. However, the heat exchangers interact with other systems, such as the fuel cell cooling system, by using the hot coolant to heat the fuel. This system requires power to pump the cooling fluid, for example. So, although still more synergistic and probably more efficient than using an electric heater, there would still be significant power consumption at the whole aircraft level.

In Figure 5.1, there are two main point groups, each divided into two secondary groups. As shown in Figure 5.2, the main clusters relate directly to the choice between a heat exchanger or an electric heater for the functions of heating or conditioning. Recall that three main functions exist: vaporization, associated with latent heat transfer to vaporize the fuel; heating, associated with sensible heat transfer to increase the fuel's temperature to the required fuel cell entry temperature; and conditioning, consisting of the fulfilling of both vaporization and heating

functions in a single component. So, an architecture is associated either with the vaporization and heating functions, with a separate heater for each, or with the conditioning function with one heater fulfilling both simultaneously. Furthermore, the heating function elevates the fuel temperature from its boiling point of around 20 K to the fuel cell entry temperature, generally above 0 ° C (273 K). This requires significantly more energy than the vaporization. Additionally, there is another function which is decoupled from the three main ones. It is denominated *pressurization* and is defined as the vaporization of part of the fuel in the tank in order to maintain adequate tank pressures. In liquid systems, it requires an additional heater, whereas in gaseous systems it is performed in the same heater used for the vaporization function.

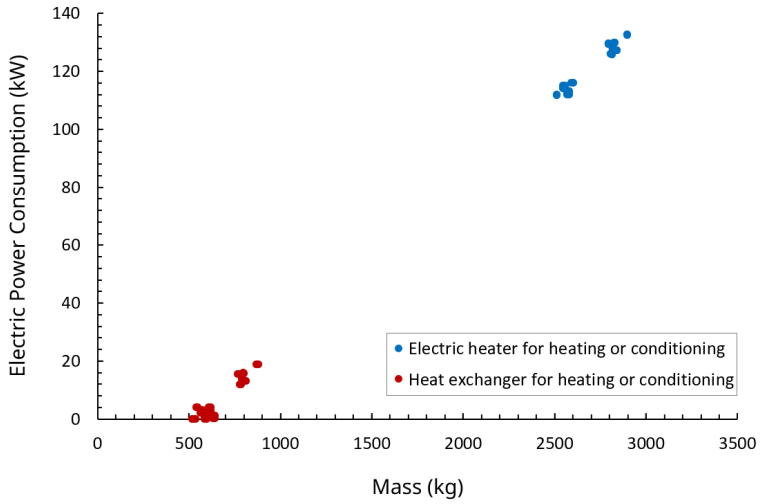


Figure 5.2: The choice between an electric heater and a heat exchanger for the heating or conditioning of the fuel has the single biggest influence on the performance of the architecture.

The discrepancy in electric power consumption between the two types of heaters is explained by what was previously alluded to: the fact that heat exchangers do not directly consume electric power. The higher mass associated with electric heaters is related to the pre-existing sizing models used. Essentially, based on these models, the electric heater has a significantly lower specific power (heating power per unit mass), thus being heavier for the same amount of required energy output. Therefore, regarding the component mass, the results appear to be quite sensitive to the assumptions made for the specific power of both component types. This warrants a deeper exploration and sensitivity analysis of the aforementioned assumptions, which is left for future work.

Figure 5.3 shows only architectures where vaporization and heating are carried out in different components, thus excluding conditioning. The type of heater used for vaporization is the cause for the secondary clusters within each main grouping. Whenever a heat exchanger is used, lighter architectures with lower power consumption are achieved.

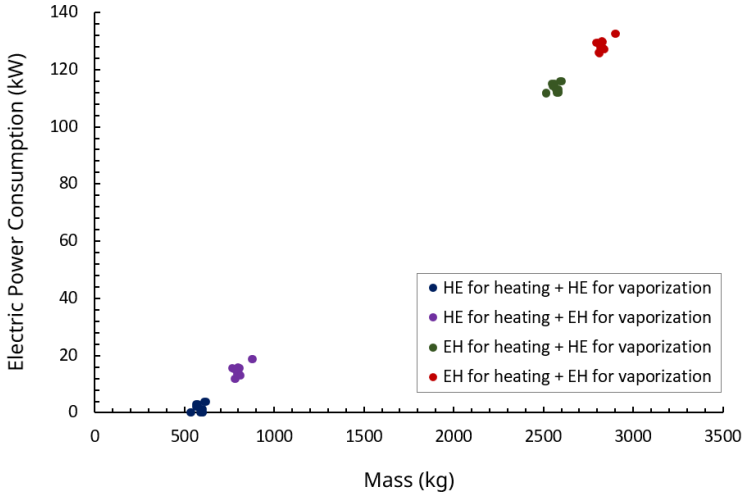


Figure 5.3: The choice between an electric heater and a heat exchanger for the vaporization of the fuel is what creates the two sub-groups within the two main clusters. To shorten the legend, abbreviations are used: HE stands for Heat Exchanger and EH stands for Electric Heater.

The reasoning is the same as for the previous case, with heat exchangers having no direct power consumption and the assumption of a higher specific power. As mentioned previously, the further heating of the fuel after vaporization requires significantly more energy than the vaporization itself. This is why it is a more dominant function, influencing the two main clusters, whereas vaporization has a secondary influence.

Essentially, in the context of this problem, and for the reasons given previously, a heat exchanger heavily outperforms an electric heater with respect to both metrics. This is the case for all functions and explains the lack of a Pareto front.

With the choice on the type of heater for the three main functions being so decisive on the performance of the architecture, the influence of other architectural decisions becomes less prominent. To be able to draw any conclusions from them, one of the four clusters has to be analyzed independently. By zooming into one cluster, the differences within it are exacerbated. Figure 5.4 shows the group selected for this task.

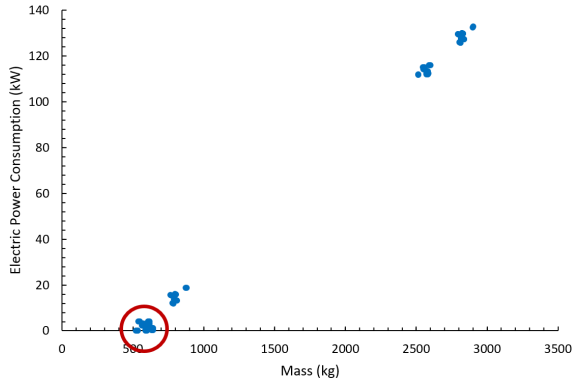
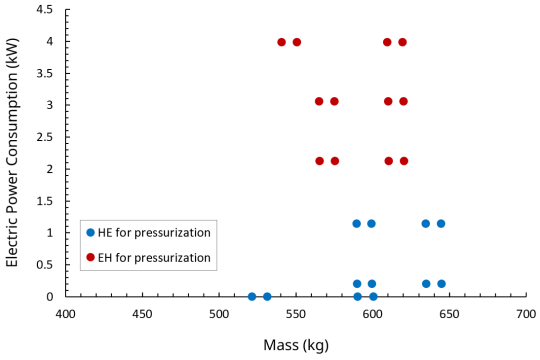
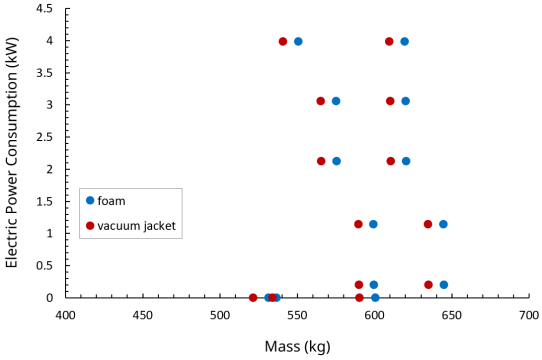


Figure 5.4: The cluster to be further analyzed is circled in red. It is the best performing group and consists of the set of architectures that use a heat exchanger for conditioning, or for both heating and vaporization separately.

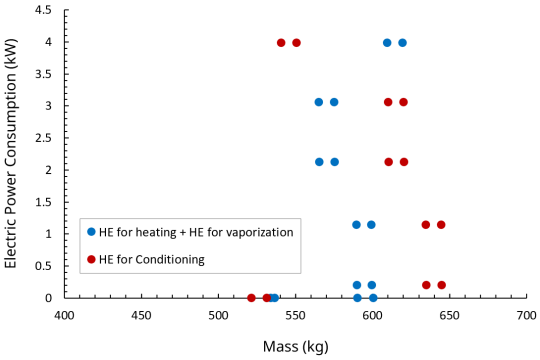
In Figure 5.5, the architectures in this cluster are differentiated on the basis of the type of heater used for pressurization, the type of insulation, the number of heaters used for heating and vaporization, the type of drive, the type of pump and the type of fuel system.



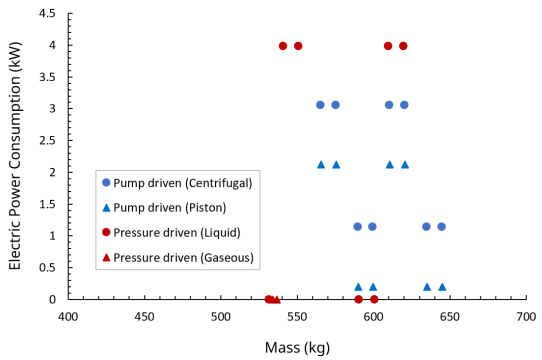
(a) Type of heater used for pressurization



(b) Type of insulation



(c) Comparison between the use of 2 separate heaters or a single one



(d) Type of drive, pump and fuel state

Figure 5.5: The best performing cluster is magnified, so that less dominant architectural decisions can be analyzed.

As the function of pressurization only exists explicitly in liquid fuel systems, Figure 5.5a only exhibits liquid fuel system architectures. Among these, once again, the type of heater used has a significant influence on the architecture. The 50% of the designs with the highest power consumption use electric heaters. However, in these smaller scales, the mass of the heater does not appear to be as noteworthy. Some electric heater architectures outperform those with heat exchangers, suggesting that other decisions also have a significant effect.

Regarding the type of insulation, depicted in Figure 5.5b, in otherwise identical architectures, using vacuum jacket over foam generates a lighter design. This is due to the difference in the density and thickness of the materials used between the inner and outer metal layers, as explained in Section 4.3.3. However, the difference in weight is not large, especially when compared to other architectural decisions. Additionally, vacuum jacket insulation has disadvantages that are not contemplated in the study, such as higher costs and structural reliability issues (Brewer 1991). Thus, given the not so significant mass advantage, foam insulation could constitute a simpler option.

As expected, using two separate heaters or a single one does not affect the electric power consumption of the architecture (Figure 5.5c). However, considering Figures 5.5c and 5.5d together, it can be concluded that the use of a single heat exchanger for the vaporization and heating of the fuel produces lighter architectures in pressure driven systems. In contrast, for pump driven architectures, the use of two separate heaters is beneficial. An explanation for this could be that for pump driven systems, the energy transferred to the fuel by the pump allows for a reduction in the scale of the heater used for vaporization, which does not transfer in the same way when a single component is used.

Comparing the type of drive (Figure 5.5d), pressure driven systems are the only ones capable of zero power consumption. Within the boundaries of this study, a pump adds weight and power consumption with no contrary drawbacks for the pressure driven system. However, a number of factors are not taken into account which would indeed be drawbacks to the pressure drive. One of which is the fact that, in a pressure driven system, the tank pressure would have to be higher. This has to be taken into account in the tank sizing, as the tank must be able to withstand said pressures, potentially increasing its mass. As the tank is outside the scope of the analysis, this is not reflected in the results. The lack of a pump would also require a more sophisticated pressure and flow control system, adding additional weight which is not accounted for in the present study.

The type of fuel pump is another architectural decision. As can be seen in Figure 5.5d, no discernible differences in mass appear between architectures with centrifugal or piston pumps. When it comes to power consumption, piston pump configurations consume significantly less. This is mostly related to the models used for the sizing of both pumps. The piston pump is exactly sized for the required pressure rise. In the case of the centrifugal pump model, predefined design points exist and the point closest to the required application is taken. Therefore, in reality, if a centrifugal pump were to be designed specifically for this application, such a large discrepancy in power consumption would not exist.

Finally, the optimal architecture is presented in Figure 5.6. It consists of a liquid fuel, pressure driven system. The functions of vaporization and heating are fulfilled by a single component, a heat exchanger. Tank pressurization is also ensured by a heat exchanger and the fuel lines have a vacuum jacket insulation. As explained previously, the exclusive use of heat exchangers in a pressure driven system allow for the architecture to have no electric power consumption. The mass of the architecture is 521.55 kg for one side of the aircraft, which corresponds to 1043.10 kg for the total system.

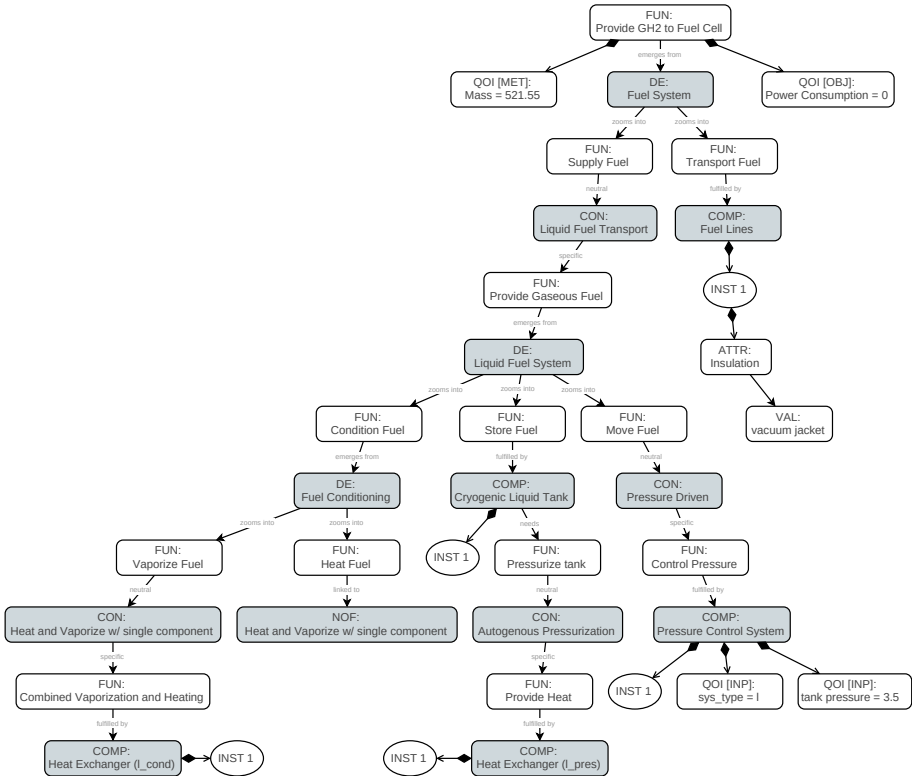


Figure 5.6: The optimal architecture consists of a liquid fuel, pressure driven system. It exclusively uses heat exchangers and the fuel lines are insulated by a vacuum jacket.

It should be emphasized once again that these results, naturally including the optimal architecture, are significantly influenced by the definition of the system boundary and thus by what is and is not taken into account. For the design of a fuel system at the level of detail required in industry, it would be recommended to expand the design space to include at least the tank design and the fuel cell BOP, namely the thermal management system. Furthermore, additional quantities of interest should be analyzed, such as cost, reliability and maintenance, among others. These topics are reiterated in Chapter 6 as part of the recommendations for future work.

Another important point is that there are several different architectures in the vicinity of the optimal architecture, as can be seen in Figure 5.5. That is, with similar values of mass and electric power consumption. The performance of these architectures is so similar that, given the uncertainties stemming from the limited system boundaries and the assumptions mentioned, the optimal architecture might not be the most desirable. Like stated above, by taking additional performance metrics into account, it is possible that other architectures could prove to have stronger arguments for their implementation.

5.1.2 Optimization Studies

Having analyzed the design of experiments and the best performing architecture in the design space, the effectiveness of various optimization strategies to achieve similar optima was assessed.

Two approaches were studied: evolutionary optimization and surrogate-based optimization (SBO). In the first case, the algorithm used was the NSGA-II (Deb et al. 2002), an elitist multiobjective genetic algorithm. Several optimizations were performed with progressively decreasing number of generations and population size, and therefore decreasing number of evaluations. An additional optimization was conducted using Bayesian Optimization (Bussemaker et al. 2024b). This is a surrogate-based optimization algorithm that builds a surrogate model of the design space and uses additional infill points to further explore the design space and refine the model. The results are displayed in Table 5.1.

These results show, on a very small scale, the benefits of surrogate-based optimization. Compared to the NSGA-II algorithm, the surrogate-based approach is capable of evaluating more architectures faster, while simultaneously being more effective in finding the optimum. In this case, the global optimum was actually obtained, with only 37.5 % of the evaluations of the

total DOE and 28.8 % of the time.

Table 5.1: Optimization results of the different approaches to the SAO problem.

ID	1	2	3	4	5
Type	DOE	NSGA-II	NSGA-II	NSGA-II	SBO
generations	-	5	5	4	-
population size	-	8	6	6	-
initial DOE size	-	-	-	-	15
infill points	-	-	-	-	15
total evaluations	80	40 (-50 %)	30 (-62.5 %)	24 (-70 %)	30 (-62.5 %)
runtime (h:mm)	4:48	2:18 (-52 %)	1:41 (-65 %)	1:24 (-71 %)	1:23 (-71 %)
opt. mass (kg)	521.55	534.09	540.78	534.09	521.55
opt. power (W)	0	0	3058.5	0	0
mass dev. (%)	-	2.4	3.7	2.4	0
power dev. (%)	-	0	∞	0	0

SBO is generally worthwhile for more complex problems. When design optimization problems involve many design variables and computationally expensive models, evolutionary algorithms can quickly become infeasible because they require a large number of function evaluations (Bussemaker et al. 2021). Using a surrogate model, function evaluations are made faster and computationally cheaper, allowing for a more comprehensive exploration of the design space (Bussemaker et al. 2024b). However, even in an relatively small problem such as the one presented herein, its performance is superior in terms of both efficiency and effectiveness.

5.2 Comparison to Conventional Approach

In addition to the system architecture optimization study for the hydrogen fuel system, the utility of the employed SAO framework is analyzed. The purpose is to quantify the time savings potential of such an automated procedure.

For this, three approaches to the analysis of the complete design space (80 architectures) are compared. The difference between them lies in the level of automation, going from a completely manual method to the Dynamic MDAO methodology used in this thesis. Between the two, a semi-automatic approach is also considered.

The manual method consists of, for each architecture, the manual generation of the architecture XML file, which is used as input to the analysis workflow. This is followed by the creation of said workflow in RCE, which is unique to the specific architecture and therefore must be

created for each. Finally, the analysis workflow is executed independently in RCE for each architecture. Table 5.2 summarizes the analysis of the manual approach. The duration of each task was obtained experimentally and is multiplied by the number of architectures to be analyzed (80).

Table 5.2: Task breakdown of the manual approach with respective time calculation.

Task	Time (hh:mm)	Notes
manual generation of architecture XML file	4:00	3 mins/arch
manual creation of RCE workflow	53:20	40 mins/arch
execution of analysis workflow in RCE	4:00	3 mins/arch
Total time	61:20	

Next, a more automated approach is presented in Table 5.3. Here, the architecture design space is modeled in ADORE, where individual architectures are manually generated. The respective XML file is then exported, eliminating the need to actually write the architecture into it. The mapping between the ADORE data model and the CPACS format in which the architecture should be represented is ensured by the Node Factory Evaluator (NFE). For this, the NFE base file must be created. Then, like in the manual approach, an analysis workflow is developed for each architecture. However, in this case it is created in MDAx and simply exported to RCE, which significantly simplifies the process. In RCE, a few quick steps are needed to set up the workflow before it is finally executed.

Table 5.3: Task breakdown of the semi-automatic approach with respective time calculation.

Task	Time (hh:mm)	Notes
creation of architecture design space model in ADORE	3:00	
development of the NFE base file	2:00	
generation of architecture in ADORE and XML file export	1:20	1 min/arch
creation of analysis workflow in MDAx	20:00	15 mins/arch
setup & execution of analysis workflow in RCE	5:20	4 mins/arch
Total time	31:40	

Finally, the Dynamic MDAO approach is evaluated in Table 5.4. The advantage of this method is that only one workflow is developed, capable of analyzing all architectures. This additionally allows for the creation of an automated analysis loop, where architecture generation is also automated and ensured by ADORE.

Table 5.4: Task breakdown of the Dynamic MDAO approach with respective time calculation.

Task	Time (hh:mm)
creation of the architecture design space model in ADORE	3:00
creation of dynamic MDAO problem in MDAx	4:00
development of the NFE base file	2:00
execution of the DOE in RCE	4:48
Total time	13:48

Even in a relatively small design problem such as the one presented, the advantage of the use of an automated procedure like Dynamic MDAO is clear and quantified in Table 5.5. Repetitive manual tasks are eliminated, resulting in significant time savings. It is also straightforward that the larger the design problem, the more worthwhile this approach is. This is because the total time it takes to implement is much less sensitive to the size of the problem when compared to the other two approaches.

Table 5.5: Comparison of the different approaches based on total duration.

Approach	Duration (hh:mm)
Manual	61:20
Semi-automatic	31:40 (- 48 %)
Automatic	13:48 (- 77%)

Taking into account the necessary time for training on the used SAO framework, the approach is still considered valuable to any industrially relevant design case. Depicted in Table 5.6, the required training time is estimated at around 30 hours, including introductions to ADORE, MDAx, RCE and the full implementation of a benchmark problem such as the Fourier problem introduced in Chapter 4.

Table 5.6: Task breakdown of the necessary training on the SAO framework.

Task	Time (hh:mm)
MDAx workshop	3:00
ADORE workshop	2:30
RCE tutorials	2:00
implementation of Fourier benchmark problem	22:00
Total time	29:30

In practical application, design spaces are generally much larger than that of the current problem. Therefore, this training time is negligible when compared to the time required in a fully

manual approach. The semi-automatic approach also requires most of the training mentioned, excluding that of the creation of dynamic MDAO problems.

Naturally, the non-automated approaches presented would not be feasible. Traditionally, only a small number of architectures would be selected for analysis based on their potential, as perceived by expert opinion (Roelofs & Vos 2018). The use of system architecture optimization thus allows for a much more thorough exploration of the design space, while being also free of subjectivity and bias (Bussemaker et al. 2020).

6 Conclusion & Future Work

This research aimed to study the design space of a hydrogen fuel system for a fuel-cell-based regional airliner. This goal has been pursued through the use of system architecture optimization, where particularly promising architectures are discovered by formalizing and solving an optimization problem. To do so, an architecture generator is required to formalize the architectural design space. Then, an architecture evaluator is used to provide feedback to the optimizer on the performance of each architecture.

For this purpose, the SAO framework used at the DLR Institute of System Architectures in Aeronautics is employed. ADORE retains the functions of optimizer and architecture generator. MDAx is used to create the MDO problem, which is then exported and executed in RCE, a PIDO environment. Additionally, CPACS is used as an XML-based central data schema.

Recent developments in MDAx have equipped it with capabilities that allow the dynamic and automatic formulation of MDAO workflows to accommodate different architectures within an optimization study. These constitute the concept of dynamic MDAO and enable the execution of system architecture optimization. Therefore, this thesis also aimed to demonstrate that the employed framework can be used for realistic application cases in aeronautical engineering.

The results of the design space exploration show that the fuel system's performance, in terms of mass and electric power consumption, is strongly dependent on the type of heater used for the various heating functions of the system. These are vaporization, heating, conditioning and pressurization. Architectures that favor the use of heat exchangers in detriment of electric heaters significantly outperform their counterparts. This appears to suggest that heat exchangers are favorable, at least in part due to the presupposed reuse of expelled heat from other aircraft system. In contrast to the direct consumption of electric power from electric heaters, this constitutes a more efficient and synergistic solution. Furthermore, significant differences also arise between the mass of the two types of systems. This is related to the assumptions made about the specific power of both heater types.

For the reasons stated above, it is intuitively reasonable that heat exchangers could generally constitute a better solution. However, such a stark difference suggests that the system boundaries and assumptions made also have a significant influence. The fuel cell balance of plant consumes electric power. It would also have to interact directly with a heat exchanger used in the fuel system. Thus, the inclusion of the BOP in the analysis is expected to reduce the differences in power consumption between heat exchanger and electric heater architectures.

Regarding the system mass, a sensitivity analysis of the assumptions about the specific power of the components would be a relevant topic of future work.

Given the strong dominance of the type of heater in the results, other architectural decisions become less prominent. A vacuum jacket insulation provides a marginal weight saving compared to foam insulation. Since the difference is not large, other factors such as cost, reliability and manufacturing could be more decisive. These are not taken into account in the study. The type of pump is also not a driving architectural choice, the performance of both pump types not having a significant influence on the architecture. Finally, the optimal architecture is pressure driven. However, pressure driven systems require higher tank pressures. In turn, this should be taken into account in the tank sizing, potentially increasing its mass. As the tank is outside the scope of the analysis, this is not reflected in the results.

In summary, the results obtained allow for the identification of trends. However, they are also influenced by the definition of the system boundary and the assumptions made. Thus, the optimal architecture should not be seen as a recommendation. For the design of a fuel system at the level of detail required in industry, the design space should be expanded to include at least the fuel cell BOP and the tank design. Furthermore, additional quantities of interest should be accounted for, such as maintenance, cost, reliability, among others. It is foreseeable that just by adding these metrics, the apparent value of architectures could shift.

In the implementation of this SAO problem, existing design tools have been improved and new tools have been developed. A solid foundation now exists for the systematic exploration of hydrogen fuel system architectures. This is one of the main outcomes of the work. Further developments are now possible, in order to conduct more comprehensive studies. This mainly consists of the inclusion of additional disciplines, thus expanding the system boundary and/or providing additional architectural alternatives. As mentioned above, the system boundary may be expanded by including the tank design and fuel cell BOP. Additionally, humidifiers and the venting system could be added to the analysis. Regarding architecture alternatives, the option for helium pressurization should be studied, as well as additional pump types and respective drives. The inactive elements in the fuel system ADSG (Figure 4.8) can provide guidance on potential directions of design space expansion.

Finally, this research has demonstrated the use of system architecture optimization in a realistic engineering application. Particularly, it has demonstrated the capabilities of the specific employed framework. A comparison of different optimization approaches highlights the poten-

tial of surrogate-based optimization. The advantages of using an automated approach in the system architecting phase have been quantified in terms of time-saving potential, compared to a fully manual and a semi-automatic approach. Additionally, it is emphasized that such an approach allows for a much larger exploration of the design space, while also eliminating expert bias and subjectivity.

However, the employed framework is in continuous development. The problems encountered throughout the implementation of the optimization problem are documented and suggestions for improvement in the functionalities of MDAx are made. Presented in the next section, this is an additional outcome of this research.

Future Work in MDAx

In this section, recommendations are made for adaptations to the dynamic MDAO implementation in MDAx. The goal is to improve the framework's capacity to fulfill the needs of complex system architecture optimization problems in an engineering context. For this, the following future work is proposed:

- In the export of workflows to RCE, if the output of more than one tool is not required as input to any other tool, each of the tools has its own output writer. The workflow should only have one output writer, and thus the outputs of these tools should be merged and connected to a single output writer.
- If a workflow export contains a nested workflow, having this nested workflow been integrated in RCE, the sub-workflow is not recognized in RCE. This is because, in MDAx, the sub-workflow is still defined as a user-integrated tool. It should be defined as a user-integrated workflow, which corresponds to the folder where it is stored in RCE.
- Activation logic scripts are exported to RCE in Jython language. This uses an old version of Python and does not accept square brackets in XPaths. Thus, for XPaths with attributes, the language has to be manually changed in RCE to Python. The default language should be changed to Python.
- Merge scripts for concurrency resolution: in the case of a common use XPath like "`stateInfo`", the "xml to integrate" file will always override the "XML" file. In concurrency resolution, it is not possible to know a priori which file will contain information in that XPath, which sometimes leads to information being erased. The merge scripts should be further developed to deal with such common use XPaths.

Bibliography

Adler, E. J. & Martins, J. R. (2023), 'Hydrogen-powered aircraft: Fundamental concepts, key technologies, and environmental impacts', *Progress in Aerospace Sciences* **141**, 100922.

Aero Corner (2024), 'Lockheed p-80 shooting star - price, specs, photo gallery, history - aero corner'.
URL: *https://aerocorner.com/aircraft/lockheed-p-80-shooting-star/*

Aerospace Industries Association, ed. (2016), *Life Cycle Benefits of Collaborative MBSE Use for Early Requirements Development*.

Aerospace Vehicle Systems Institute, T. A. M. (2024), 'Exponential growth of system complexity'.
URL: *https://savi.avsi.aero/about-savi/savi-motivation/exponential-system-complexity/*

Airbus (2024), 'Airbus takes superconductivity research for hydrogen-powered aircraft a step further'.
URL: *https://www.airbus.com/en/newsroom/press-releases/2024-05-airbus-takes-superconductivity-research-for-hydrogen-powered*

Airbus Deutschland GmbH, ed. (2003), *Liquid Hydrogen Fuelled Aircraft - System Analysis: Final Technical Report: Cryoplane*.

Alder, M., Moerland, E., Jepsen, J. & Nagel, B. (2020), Recent advances in establishing a common language for aircraft design with cpacs.

Bhatti, W., Wu, W., Doyle, F., Llambrich, J., Webber, H. & Town, N. (2022), 'Fuel cells - roadmap report'.
URL: *https://www.ati.org.uk/wp-content/uploads/2022/03/FZO-PPN-MAP-0032-Fuel-Cells-Roadmap.pdf*

Bielsky, T., Kuelper, N. & Thielecke, F. (2023), Overall parametric design and integration of on-board systems for a hydrogen-powered concept aircraft, *in* 'Aerospace Europe Conference, Lausanne'.

Bielsky, T., Thielecke, F., Atanasov, G., Burschyk, T. & Nagel, B. (2024), Collaborative parametric overall aircraft and systems design to evaluate hydrogen-powered regional aircraft, Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V.

Biermann, A. E. & Kohl, R. C. (1959), 'Preliminary study of a piston pump for cryogenic fluids', *NASA MEMO 3-6-59E* .

Biermann, A. E. & Shinko, W. G. (1960), 'Performance study of a piston-type pump for liquid hydrogen', *NASA Technical Note D-276* .

Boden, B., Flink, J., Först, N., Mischke, R., Schaffert, K., Weinert, A., Wohlan, A. & Schreiber, A. (2021), 'Rce: An integration environment for engineering and science', *SoftwareX* **15**, 100759.

Brand, J., Sampath, S., Shum, F., Bayt, R. & Cohen, J. (2003), Potential use of hydrogen in air propulsion, *in* 'AIAA International Air and Space Symposium and Exposition: The Next 100 Years', American Institute of Aeronautics and Astronautics, Reston, Virigina.

Brewer, G. D. (1991), *Hydrogen aircraft technology*, CRC Press, Boca Raton.

Brewer, G. D., Morris, R. E., Davis, G. W., Versaw, E. F. & Cunnington, Jr, G. R. (1978), Study of fuel systems for lh2-fueled subsonic transport aircraft, volume 2, Technical report, Lockheed-California Co., Burbank (USA).
URL: *https://www.osti.gov/biblio/6404648*

Brewer, G. & Morris, R. (1978), 'Study of lh2 fueled subsonic passenger transport aircraft', *Contractor report NASA/CR–145369* .

Burschyk, T., Alder, M., Mancini, A., Bielsky, T., Kriewall, V., Thielecke, F. & Nagel, B. (2025), 'Introduction of a system definition in the common parametric aircraft configuration schema (cpacs)', *Aerospace* **12**, 373.

Burschyk, T., Fröhler, B., Alder, M. & Zill, T. (2024), Global sensitivity analysis of liquid hydrogen storage design parameters for overall aircraft design, *in* '34th Congress of the International Council of the Aeronautical Sciences, ICAS 2024'.

Bussemaker, J. (2024), 'Adore documentation'.
URL: *https://adore.mbse-env.com/docs/adore-v151-documentation*

Bussemaker, J. (2025), System architecture optimization: Function-based modeling, optimization algorithms, and integration with mdao. PhD thesis, submitted for publication.

Bussemaker, J. & Boggero, L. (2022), Technologies for enabling system architecture optimization.

Bussemaker, J., Boggero, L. & Ciampa, P. D. (2022), 'From system architecting to system design and optimization: A link between mbse and mdao', *INCOSE International Symposium* **32**(1), 343–359.

Bussemaker, J., Garg, S. & Boggero, L. (2022), The influence of architectural design decisions on the formulation of mdao problems.

Bussemaker, J. H., Bartoli, N., Lefebvre, T., Ciampa, P. D. & Nagel, B. (2021), Effectiveness of surrogate-based optimization algorithms for system architecture optimization, *in* 'AIAA AVIATION 2021 FORUM', American Institute of Aeronautics and Astronautics, Reston, Virginia.

Bussemaker, J. H., Boggero, L. & Nagel, B. (2024a), System architecture design space exploration: Integration with computational environments and efficient optimization, *in* 'AIAA AVIATION FORUM AND ASCEND 2024', American Institute of Aeronautics and Astronautics, Reston, Virginia.

Bussemaker, J. H. & Ciampa, P. D. (2022), Mbse in architecture design space exploration, *in* A. M. Madni, N. Augustine & M. Sievers, eds, 'Handbook of Model-Based Systems Engineering', Springer International Publishing, Cham, pp. 1–41.

Bussemaker, J. H., Ciampa, P. D. & Nagel, B. (2020), System architecture design space exploration: An approach to modeling and optimization, *in* 'AIAA AVIATION 2020 FORUM', American Institute of Aeronautics and Astronautics, Reston, Virginia.

Bussemaker, J. H., Saves, P., Bartoli, N., Lefebvre, T. & Nagel, B. (2024b), Surrogate-based

optimization of system architectures subject to hidden constraints, *in* 'AIAA AVIATION FORUM AND ASCEND 2024', American Institute of Aeronautics and Astronautics.

Bussemaker, J., Sánchez, R. G., Fouda, M., Boggero, L. & Nagel, B. (2023), 'Function-based architecture optimization: An application to hybrid-electric propulsion systems', *INCOSE International Symposium* **33**, 251–272.

Capaccio, A. (2017), 'F-35 program costs jump to $406.5 billion in latest estimate', *Bloomberg* .

URL: *https://www.bloomberg.com/politics/articles/2017-07-10/f-35-program-costs-jump-to-406-billion-in-new-pentagon-estimate*

Ciampa, P. D., La Rocca, G. & Nagel, B. (2020), A mbse approach to mdao systems for the development of complex products, *in* 'AIAA AVIATION 2020 FORUM', American Institute of Aeronautics and Astronautics, Reston, Virginia.

Ciampa, P. D. & Nagel, B. (2016), 'Towards the 3rd generation mdo collaborative environment', *ICAS* (30).

Ciampa, P. D. & Nagel, B. (2020), 'Agile paradigm: The next generation collaborative mdo for the development of aeronautical systems', *Progress in Aerospace Sciences* **119**, 100643.

Ciampa, P. D. & Nagel, B. (2021), Accelerating the development of complex systems in aeronautics via mbse and mdao: a roadmap to agility, *in* 'AIAA AVIATION 2021 FORUM', American Institute of Aeronautics and Astronautics, Reston, Virginia.

Crawley, E. F., Cameron, B. & Selva, D. (2016), *System Architecture: Strategy and Product Development for Complex Systems*, Pearson, Boston.

Deb, K., Pratap, A., Agarwal, S. & Meyarivan, T. (2002), 'A fast and elitist multiobjective genetic algorithm: Nsga-ii', *IEEE Transactions on Evolutionary Computation* **6**(2), 182–197.

Dicks, A. L. & Rand, D. A. J. (2018), *Fuel Cell Systems Explained*, Wiley.

DLR (2025), 'Dlr exact - architecting aviation'.

URL: *https://exact-dlr.de/*

European Union (2011), ‘Flightpath 2050 : Europe’s vision for aviation : maintaining global leadership and serving society’s needs.’.

Gallard, F., Vanaret, C., Guénot, D., Gachelin, V., Lafage, R., Pauwels, B., Barjhoux, P.-J. & Gazaix, A. (2018), Gems: A python library for automation of multidisciplinary design optimization process generation, *in* ‘2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference’.

García Sánchez, R. (2024), Adaptation of an mdo platform for system architecture optimization, Master’s thesis, Delft University of Technology.

Garg, S., Bussemaker, J., Boggero, L. & Nagel, B. (2024a), Mdax : Enhancements in a collaborative mdao workflow formulation tool, *in* ‘34th Congress of the International Council of the Aeronautical Sciences, ICAS 2024’.
URL: *https://elib.dlr.de/206786/*

Garg, S., Garcia Sanchez, R., Bussemaker, J. H., Boggero, L. & Nagel, B. (2024b), Dynamic formulation and execution of mdao workflows for architecture optimization, *in* ‘AIAA AVIATION FORUM AND ASCEND 2024’, American Institute of Aeronautics and Astronautics, Reston, Virginia.

Gray, J. S., Hwang, J. T., Martins, J. R. R. A., Moore, K. T. & Naylor, B. A. (2019), ‘Openmdao: an open-source framework for multidisciplinary design, analysis, and optimization’, *Structural and Multidisciplinary Optimization* **59**(4), 1075–1104.

Hales, M. O., Wood, N., Harrison, S., Taylor, S., Husband, M., Stonham, J., Zhao, C., Ettridge, D. & Westmore, D. (2023), H2gear hydrogen electric powertrain – system architecture, *in* ‘AIAA AVIATION 2023 Forum’, American Institute of Aeronautics and Astronautics, Reston, Virginia.

Han, J., Feng, J., Chen, P., Liu, Y. & Peng, X. (2022), ‘A review of key components of hydrogen recirculation subsystem for fuel cell vehicles’, *Energy Conversion and Management: X* **15**, 100265.

Haskins, C., ed. (2006), *INCOSE - Systems Engineering Handbook: A Guide For System Life*

Cycle Processes and Activities.

Hirshorn, S. R., Voss, L. D. & Bromley, L. K. (2017), *NASA Systems Engineering Handbook*.

Iacobucci, J. V. (2012), Rapid architecture alternative modeling (RAAM): A framework for capability-based analysis of system of systems architectures, PhD thesis, Georgia Institute of Technology.

IEA (2025), 'Tracking aviation', *International Energy Agency* .
URL: *https://www.iea.org/energy-system/transport/aviation*

INCOSE, ed. (2007), *Systems Engineering Vision 2020*, INCOSE.

INCOSE, ed. (2023), *INCOSE Systems Engineering Handbook*, 5 edn, John Wiley & Sons, Nashville, TN.

IPCC (2023), *Climate Change 2023: Synthesis Report. Contribution of Working Groups I, II and III to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change [Core Writing Team, H. Lee and J. Romero (eds.)]. IPCC, Geneva, Switzerland.*, Intergovernmental Panel on Climate Change.

Jiang, W. & Fahimi, B. (2010), 'Active current sharing and source management in fuel cell–battery hybrid power system', *IEEE Transactions on Industrial Electronics* **57**(2), 752–761.

Khandelwal, B., Karakurt, A., Sekaran, P. R., Sethi, V. & Singh, R. (2013), 'Hydrogen powered aircraft : The future of air transport', *Progress in Aerospace Sciences* **60**, 45–59.
URL: *https://www.sciencedirect.com/science/article/pii/S0376042112000887*

Kleiner, S. & Kramer, C. (2013), Model based design with systems engineering based on rflp using v6, *in* M. Abramovici & R. Stark, eds, 'Smart Product Engineering', Lecture Notes in Production Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 93–102.

Lafage, R., Defoort, S. & Lefebvre, T. (2019), Whatsopt: a web application for multidisciplinary design analysis and optimization, *in* 'AIAA Aviation 2019 Forum', American Institute of Aeronautics and Astronautics, Reston, Virginia.

Lambe, A. B. & Martins, J. R. R. A. (2012), 'Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes', *Structural and Multidisciplinary Optimization* **46**(2), 273–284.

Liu, Y., Zhang, S., Zhang, J., Yao, K., Luo, M. & Liu, Y. (2024), 'Towards the design of future aircraft: A critical review on the tools and methodologies', *Aerospace Research Communications* **2**.

Living Warbirds (2024), 'Lockheed p-80 shooting star airplane videos and airplane pictures'.
URL: *http://www.livingwarbirds.com/lockheed-p80.php*

López González, E., Sáenz Cuesta, J., Vivas Fernandez, F. J., Isorna Llerena, F., Ridao Carlini, M. A., Bordons, C., Hernandez, E. & Elfes, A. (2019), 'Experimental evaluation of a passive fuel cell/battery hybrid power system for an unmanned ground vehicle', *International Journal of Hydrogen Energy* **44**(25), 12772–12782.

Losey, S. (2024), 'F-35s to cost $2 trillion as pentagon plans longer use, says watchdog', *Defense News* .
URL: *https://www.defensenews.com/air/2024/04/15/f-35s-to-cost-2-trillion-as-pentagon-plans-longer-use-says-watchdog/*

Madni, A. M., Augustine, N. & Sievers, M. (2023), *Handbook of Model-Based Systems Engineering*, Springer International Publishing, Cham.

Maier, M. W. & Rechtin, E. (2009), *The Art of Systems Architecting*, 3rd ed. edn, CRC Press, Boca Raton.

Martins, J. R. R. A. & Lambe, A. B. (2013), 'Multidisciplinary design optimization: A survey of architectures', *AIAA Journal* **51**(9), 2049–2075.

Martins, J. R. R. A. & Ning, S. A. (2021), *Engineering design optimization*, Cambridge University Press, Cambridge and New York NY.

Massaro, M. C., Biga, R., Kolisnichenko, A., Marocco, P., Monteverde, A. H. A. & Santarelli, M.

(2023), 'Potential and technical challenges of on-board hydrogen storage technologies coupled with fuel cell systems for aircraft electrification', *Journal of Power Sources* **555**, 232397.

Mavris, D., de Tenorio, C. & Armstrong, M. (2008), Methodology for aircraft system architecture definition, *in* '46th AIAA Aerospace Sciences Meeting and Exhibit', American Institute of Aeronautics and Astronautics, Reston, Virigina.

Menshenin, Y., Mordecai, Y., Crawley, E. F. & Cameron, B. G. (2020), Model-based system architecting and decision-making, *in* A. M. Madni, N. Augustine & M. Sievers, eds, 'Handbook of Model-Based Systems Engineering', Springer International Publishing, Cham, pp. 1–42.

Millis, M. G., Tornabene, R. T., Jurns, j. M., Guynn, M. D., Tomsik, T. M. & Van Overbeke, T. J. (2009), 'Hydrogen fuel system design trades for high-altitude long-endurance remotely-operated aircraft', *NASA/TM—2009-215521* .

Moran, S. (2016), 'Pump sizing: Bridging the gap between theory and practice', *Chemical Engineering Progress* .

Müller, J. & Day, M. (2019), 'Surrogate optimization of computationally expensive black-box problems with hidden constraints', *INFORMS Journal on Computing* **31**(4), 689–702.

Nash, D., Aklil, D., Johnson, E., Gazey, R. & Ortisi, V. (2012), *Hydrogen Storage: Compressed Gas*, Vol. 4, pp. 119–143.

O'Hayre, R. P., Cha, S.-W., Colella, W. G. & Prinz, F. B. (2016), *Fuel cell fundamentals*, third edition edn, Wiley, Hoboken New Jersey.

Page Risueño, A., Bussemaker, J., Ciampa, P. D. & Nagel, B. (2020), Mdax: Agile generation of collaborative mdao workflows for complex systems, *in* 'AIAA AVIATION 2020 FORUM', American Institute of Aeronautics and Astronautics, Reston, Virginia.

Palladino, V., Bartoli, N., POMMIER-BUDINGER, V., Benard, E., Schmollgruber, P. & Jordan, A. (2023), 'Optimization of a hydrogen-based hybrid propulsion system under aircraft performance constraints', *Chinese Journal of Aeronautics* **36**(5), 41–56.

Palladino, V., Jordan, A., Bartoli, N., Schmollgruber, P., Pommier-Budinger, V. & Benard, E.

(2021), Preliminary studies of a regional aircraft with hydrogen-based hybrid propulsion, *in* 'AIAA AVIATION 2021 FORUM', American Institute of Aeronautics and Astronautics, Reston, Virginia.

Pate, D. J., Gray, J. & German, B. J. (2014), 'A graph theoretic approach to problem formulation for multidisciplinary design analysis and optimization', *Structural and Multidisciplinary Optimization* **49**(5), 743–760.

Peschka, W. (1992), *Liquid Hydrogen*, Springer Vienna, Vienna.

Roelofs, M. & Vos, R. (2018), Correction: Uncertainty-based design optimization and technology evaluation: A review, *in* '2018 AIAA Aerospace Sciences Meeting', American Institute of Aeronautics and Astronautics, Reston, Virginia.

Scholz, D. (2015), 'Aircraft design'. Lecture Notes. Hamburg University of Applied Sciences. **URL:** *http://lecturenotes.aircraftdesign.org/*

Selva, D., Cameron, B. & Crawley, E. (2016), 'Patterns in system architecture decisions', *Systems Engineering* **19**(6), 477–497.

Shah, R. K. & Sekulić, D. P. (2003), *Fundamentals of Heat Exchanger Design*, Wiley.

Sillitto, H., Martin, J., Mckinney, D., Griego, R., Dori, D., Krob, D., Godfrey, P., Arnold, E. & Jackson, S. (2019), 'Systems engineering and system definitions', *The International Council on Systems Engineering (INCOSE)* .

Soleymani, M., Mostafavi, V., Hebert, M., Kelouwani, S. & Boulon, L. (2024), 'Hydrogen propulsion systems for aircraft, a review on recent advances and ongoing challenges', *International Journal of Hydrogen Energy* **91**, 137–171.

Spencer, K. M. & Martin, C. A., eds (2013), *Investigation of Potential Fuel Cell Use in Aircraft: Technical Report IDA Document D-5043*, Institute for Defence Analyses.

Tiwari, S., Pekris, M. J. & Doherty, J. J. (2024), 'A review of liquid hydrogen aircraft and propulsion technologies', *International Journal of Hydrogen Energy* **57**, 1174–1196.

Torenbeek, E. (1982), *Synthesis of Subsonic Airplane Design*, Springer Netherlands.

Turner, J., Contaut, S., Tarantino, A. & Masson, A. (2022), 'Cryogenic hydrogen fuel system and storage - roadmap report'.

US EPA (2025), 'Global greenhouse gas overview'.
URL: *https://www.epa.gov/ghgemissions/global-greenhouse-gas-overview*

van Gent, I. (2019), Agile MDAO Systems: A Graph-based Methodology to Enhance Collaborative Multidisciplinary Design, PhD thesis, Delft University of Technology.

van Gent, I. & La Rocca, G. (2019), 'Formulation and integration of mdao systems for collaborative design: A graph-based methodological approach', *Aerospace Science and Technology* **90**, 410–433.

Verstraete, D. (2013), 'Long range transport aircraft using hydrogen fuel', *International Journal of Hydrogen Energy* **38**(34), 14824–14831.
URL: *https://www.sciencedirect.com/science/article/pii/S036031991302212X*

Verstraete, D., Hendrick, P., Pilidis, P. & Ramsden, K. (2010), 'Hydrogen fuel tanks for subsonic transport aircraft', *International Journal of Hydrogen Energy* **35**(20), 11085–11098.

Vietze, M. & Weiland, S. (2022), 'System analysis and requirements derivation of a hydrogen-electric aircraft powertrain', *International Journal of Hydrogen Energy* **47**(91), 38793–38810.

Wallington, T., Woody, M., Lewis, G., Keoleian, G., Adler, E., Martins, J. & Collette, M. (2025), 'Hydrogen as a sustainable transportation fuel', *Renewable and Sustainable Energy Reviews* **217**, 115725.

Wood, N., Hales, M. O. & Taylor, S. (2023), Notional aircraft platforms for hydrogen fuel cell propulsion, *in* 'AIAA AVIATION 2023 Forum', American Institute of Aeronautics and Astronautics, Reston, Virginia.

Xu, L., Huangfu, Y., Ma, R., Xie, R., Song, Z., Zhao, D., Yang, Y., Wang, Y. & Xu, L. (2022), 'A comprehensive review on fuel cell uav key technologies: Propulsion system, manage-

ment strategy, and design procedure', *IEEE Transactions on Transportation Electrification* **8**(4), 4118–4139.

Zaefferer, M. & Horn, D. (2018), A first analysis of kernels for kriging-based optimization in hierarchical search spaces, *in* A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete & D. Whitley, eds, 'Parallel Problem Solving from Nature – PPSN XV', Vol. 11102 of *Lecture Notes in Computer Science*, Springer International Publishing, Cham, pp. 399–410.

Çengel, Y. A. (2003), *Heat Transfer: A Practical Approach*, 2 edn, McGraw Hill.

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here.

This paper was not previously presented to another examination board and has not been published.

Garching, 10. June

_____

first and last name

A Appendix: Python Classes for Verification of Architectural Influence Assertions

This appendix contains all the Python classes implemented in MDAx to check the different assertions associated with architectural influences. This implementation is introduced in García Sánchez (2024), from which Table A.1 is extracted. Each row of the table includes the assertion description, the corresponding Python class name, the arguments it requires, and an example of how the class is applied in a discipline configuration file.

In addition, conditions consisting of multiple assertions can also be employed. This is done using logical operators such as OR (|), negation (~) and AND (&) (García Sánchez 2024).

Table A.1: Python classes implemented in MDAx to check assertions. Reproduced from García Sánchez (2024).

Assertion	Class	Arguments	Example
Existence of a node	XPathExists()	node xpath	XpathExists('rocket/Propulsion /Liquid_engine')
Number of nodes (lower)	XPathNLt()	node xpath, reference	XPathNLt('rocket/Propulsion /Liquid_engine/engine',2)
Number of nodes (equal)	XPathNEq()	node xpath, reference	XPathNEq('rocket/Propulsion /Liquid_engine/engine' ,2)
Number of nodes (higher)	XPathNGt()	node xpath, reference	XPathNGt('rocket/Propulsion /Liquid_engine/engine',2)
Contain value	ElHasValue()	node xpath	ElHasValue('rocket/Geometry /material')
Contain exact string	ElStrEq()	node xpath, string	ElStrEq('rocket/Geometry /material','aluminium6061')
Contain fragment of a string	ElStrContains()	node xpath, string	ElStrContains('rocket/Geometry /material','aluminium')
Contain value lower than reference	ElNumLt()	node xpath, reference	ElNumLt('rocket/Geometry /number_of_fins', 3)
Contain value equal than reference	ElNumEq()	node xpath, reference	ElNumEq('rocket/Geometry /number_of_fins', 3)
Contain value higher than reference	ElNumGt()	node xpath, reference	ElNumGt('rocket/Geometry /number_of_fins', 3)