

INTEGRATING SYSTEM ARCHITECTURE OPTIMIZATION INTO MODEL-BASED SYSTEMS ENGINEERING METHODS

Santiago Valencia-Ibáñez[1] & Susan Liscouët-Hanke[1]

[1]Department of Mechanical, Industrial and Aerospace Engineering, Concordia University, Montreal, Canada

Abstract

Complex systems like those found in modern aircraft can include thousands of interacting parts that produce emergent behaviors. Model-Based Systems Engineering (MBSE) manages this difficulty with centrally maintained models, but these models tend to be *descriptive*: they capture a system's functions, components, and interfaces without quantifying how well a given architecture performs. Meanwhile, System Architecture Optimization (SAO) searches large design spaces for high-performing architectures but has been developed largely outside any formal MBSE method. Prior attempts to connect the two have either fixed the architecture upstream of the optimization loop or extended modeling languages without a clear methodological framing of the architecture search. In this work, a methodology is introduced that places SAO inside the Arcadia MBSE method through an Architecture Design Space (ADS) between the *solution-agnostic* and *solution-specific* model layers, encoded as a Design Space Graph (DSG) whose nodes carry model metadata so that solutions can write back directly to the source model. The method is demonstrated on the primary flight control system architecture of a large business jet, a 133-variable problem with 38 constraints.

Keywords: Model-Based Systems Engineering (MBSE), System Architecture Optimization (SAO), Flight Control Systems

1. Introduction

Contemporary aircraft system architectures are organized so that the failure of a single component does not lead to a total system failure, which entails redundancy, spatial separation and segregation, dissimilar components, and reconfigurable modes of operation. This is an example of *complexity* in engineering systems, which is conventionally defined by the large number of interacting entities a system contains, whose couplings can produce emergent behaviors that no individual element exhibits in isolation [1]. In modern aircraft, this form of complexity has grown sharply. For instance, an aircraft-level avionics architecture design can be expressed as an optimization problem with between 50 000 and 500 000 design variables [2].

A *system architecture* is a description of a system's entities (its components and functions) together with the relationships and interfaces between them [3, 4]. The activity that produces such a description is the *system architecting* process, which can be decomposed into three parts [5]: 1) *definition* synthesizes the architecture by identifying its elements and their organization; 2) *representation* captures the resulting structure through models or diagrams; and 3) *evaluation* assesses each candidate architecture against quantitative metrics. The three parts inform one another, but in practice they correspond to distinct modeling tools and often to distinct engineering teams or disciplines.

The same integration trend that drives the rise in complexity also reshapes the way evaluation must be carried out. Traditional federated avionics, in which each function is hosted by dedicated hardware, is being progressively replaced by Integrated Modular Avionics (IMA), where different functions now share the same standardized physical resources and hardware modules [6–8]. Additionally, new design concepts, including electrified systems [9, 10], unconventional configurations [11, 12], and

novel Concepts of Operations (ConOps) [13, 14], impose new spatial and integration constraints which require more accurate evaluation methods that are not based on statistical data [15].

Model-Based Systems Engineering (MBSE) has emerged as a response to this complexity mainly at the descriptive level. It is defined as the formalized application of modeling principles, methods, and tools across the entire system lifecycle [16]. MBSE provides an effective way to manage and represent interconnected systems by replacing document-based artifacts with centrally maintained models. One of the advantages of using such models is an increased *traceability*, or the ability to establish continuous digital links between stakeholder needs, system requirements, and the architectural design components that satisfy them. One example of traceability in the Arcadia MBSE method is *realization*, which establishes a formal, semantic link between elements across consecutive engineering perspectives. For instance, a Logical Function defined in the Logical Architecture (LA) layer is explicitly modeled to *realize* a corresponding System Function identified in the more abstract System Analysis (SA) layer.

Despite their advantages in architecture definition and representation, most current MBSE implementations lack a systematic connection to evaluation. Following the taxonomy laid out in the ISO/IEC/IEEE 24641 standard [17], the models used in MBSE are typically *descriptive*: they record structural and behavioral views of the system but do not represent quantitative performance metrics beyond static and often hard-coded values coming from external analyses [18, 19]. System evaluation and optimization, on the other hand, depend on *analytical* models, such as those built within Multidisciplinary Design Analysis and Optimization (MDAO) frameworks. Several efforts have attempted to interface descriptive MBSE with analytical models, but the resulting couplings are mostly ad-hoc and lack methodological integration on the MBSE side.

This paper presents a framework that places systematic architecture design space exploration inside an established MBSE methodology – specifically, Arcadia [20]. Earlier work combining MBSE with analysis and optimization has focused primarily on sizing parameters rather than the architectural design space itself [21–24], leaving functional allocation, component selection, and interconnection decisions to manual judgment or disjointed optimization routines. The methodology presented in this paper includes these architecture decisions as design variables, explores the resulting design space with a multi-objective optimizer, and writes selected solutions back to the source model. It is illustrated with a case study concerning the Primary Flight Control System (PFCS) of a large business jet.

2. Literature Review

MBSE is conventionally described with three pillars: a *language*, a *method*, and a *tool* [25, 26]. SysML has become the de-facto MBSE language, and a number of modeling environments have been developed and deployed successfully. However, according to reviews of the field, the *method* pillar is the weakest of the three [25]. Method-agnostic languages like SysML force each adopting organization to develop its own modeling strategy. As a consequence, architecture trade studies and quantitative evaluation are rarely covered by a method and tend to be executed outside of the formal MBSE process. The Arcadia method, with its companion Capella tool, developed by Thales Group, is one of the few well-documented exceptions that ties together a language, a tool, and a method [25–27].

Even when a method is present, current MBSE practice rests on *descriptive* models that capture structure, behavior, and requirements with limited quantitative analysis capabilities. Meanwhile, *analytical* models, which compute quantitative design metrics, live outside the formal Systems Engineering (SE) workflow [18, 19]. Timperley et al.'s [28] survey of common MBSE methods, languages, and tools finds that no widely used MBSE environment supports more than parameter-level variability (i.e., variation in *numbers* within a design, like geometric or physical quantities) out-of-the-box.

The recently published ISO/IEC/IEEE 24641 standard [17] on MBSE methods and tools explicitly lists tasks such as *Simulate Systems Using Models* and *Make Decisions Using Models* among the activities that a successful MBSE methodology shall support. Under that reading, simulation and design space decision-making should not be seen as optional extensions to a descriptive foundation but as core components of any MBSE method.

2.1 System Architecture Optimization

Two largely separate lines of research have aimed to address the descriptive-analytical gap in system architecture development. The first, *System Architecture Optimization (SAO)*, develops the algorithm and problem formulations needed to search hierarchical, mixed-discrete architecture design spaces. The second, *coupling MBSE and MDAO*, develops the workflows with which architectures captured in MBSE can be sized or refined by MDAO systems. These two lines have evolved in parallel and have only recently begun to interact.

SAO uses numerical optimization algorithms to aid in the architecting process, which is seen as a black-box optimization problem. For example, Frank et al. [29] apply an evolutionary multi-architecture multi-objective algorithm to suborbital vehicle design space exploration; Apaza and Selva [30] automate the composition of encoding schemes and search operators for SAO problems; and Sonneveld et al. [31] demonstrate dynamic workflow generation for aileron structural optimization.

On the other hand, the work of Bussemaker et al. [32–34] moves SAO closer to MBSE by adopting graph-based problem representations whose vocabulary matches architectural reasoning and presenting notions that are largely compatible with MBSE concepts such as *functions*, *components*, and *allocations*. Bussemaker et al. introduce the Architecture Design Space Graph (ADSG), a directed graph in which nodes represent functions, components, ports, and quantities, and edges represent function-component mapping, connection, and incompatibility [32–34]. A configured ADSG is translated automatically into a formalized set of design variables, objectives, and constraints. Furthermore, the same authors introduce `ADORE`, a Python tool with a web-based interface that implements the ADSG and connects it to performance evaluation through Python and file-based interfaces [34, 35]. Even though the ADSG vocabulary is deliberately compatible with MBSE concepts, until very recently ADSG models existed independently of any MBSE language, tool, or method. The recent SysML v2 extension presented by Bussemaker et al. [36] is a step toward integrating SAO and MBSE, although it still lacks a formal MBSE method capable of bridging different levels of abstraction and preserving traceability between model elements.

2.2 Coupling MBSE and MDAO

There have been efforts to enrich MBSE models with quantitative data by coupling these models with MDAO analyses or workflows. Several works have successfully embedded MDAO within an MBSE process by sizing or optimizing the system described by the model. This type of connection addresses the parameter-level variability identified by Timperley et al. [28]. For example, Jeyaraj et al. [37] couple a Capella/Arcadia architecture description with an MDAO workflow for aircraft system architecting; Aïello et al. [22] couple SysML with MDAO for drone battery sizing; Habermehl et al. [23] model the MDAO problem itself as a SysML activity diagram inside the system model; and Fouda et al. [24] drive MDAO setup from an MBSE product model. More recently, Gauthier et al. [38] use SysML v2 as a pivot language between MBSE and MDAO and apply the approach to a thermal management system. A distinct feature of these works is that the design variables are essentially sizing parameters (e.g., geometry, flow rates, or controller gains). The system's architecture itself (i.e., which components are present, how they connect, and how functions are allocated) is fixed upstream before the MDAO loop begins. Consequently, the combinatorial, hierarchical decision-making that defines SAO problems is largely absent from these attempts to couple MBSE and MDAO.

2.3 Toward Architecture Optimization within an MBSE Method

Recent work has begun to close the distance between SAO and MBSE-MDAO coupling. As mentioned earlier, Bussemaker et al. [36] propose a SysML v2 extension that represents ADSG concepts in this modeling language. The approach is demonstrated on a guidance, navigation and control architecture with 79 million candidates optimized for mass and reliability. Gauthier et al. [38] also adopt SysML v2 as a pivot, but their formulation remains sizing-oriented. Even in these works that bring SAO and MBSE closer, SAO interacts with MBSE as a *language* and *tool* extension rather than as part of a prescriptive method that tells the systems engineer where and how to perform architecture optimization in relation to the various activities and perspectives of MBSE such as operational analysis, functional analysis, logical architecting, and physical architecting. Placing SAO *within* a prescriptive

method is more aligned with what ISO/IEC/IEEE 24641 asks for. This work takes up this position by clearly framing architecture optimization within the Arcadia method.

2.4 Application context: FCS and IMA Architecting

The Flight Control System (FCS) and IMA architecting literature offers test problems that combine large combinatorial design spaces with safety, redundancy, and certification requirements, and is therefore well-suited to demonstrate SAO inside an MBSE method. Bauer et al. [39] formulate Fly-by-Wire (FBW) FCS architecture design as a discrete optimization problem over combinations of actuators, power circuits, and computers under safety constraints, and find the global optimum among 10^{14} architecture candidates for the Airbus A340 roll control system. Other works in FCS and IMA design [2, 8, 40–42] extend this line of research across progressively larger and more multidisciplinary and multi-objective problems, reaching design spaces with hundreds of thousands of design variables [2]. Although many of these studies use MBSE-adjacent concepts such as functional analysis [2, 8, 40–42] and allocation of functions to components [2, 8, 40, 41], they are not conducted in a formal MBSE environment or following an MBSE method. Thus, design variables, constraints, objectives, and disciplinary analyses are assembled independently of a broader systems engineering process, and solutions are not stored in a system model that keeps traceability with respect to the decisions made or with the multiple perspectives and levels of abstraction present in the design process. This work addresses this limitation by placing SAO inside an MBSE method and representing both the Architecture Design Space (ADS) and any architecture instances resulting from the optimization in a format compatible with the overarching MBSE method.

On the MBSE side, George Mathew et al. [43] use Capella/Arcadia to implement a networked aircraft control system on IMA for an environmental control system case study. This is one of the closest prior MBSE method instances for this class of system, but it is carried out without optimization. The FCS and IMA literature is rich in optimization formulations and MBSE descriptions, but the two are rarely combined in the same study, which makes this class of system a natural place to exercise the methodological contribution introduced in this paper.

3. Methodology

This section presents a four-stage methodology for including SAO within an Arcadia-based MBSE process. The stages are solution-agnostic model development in Capella, formulation of an Architecture Design Space (ADS), multi-objective optimization, and write-back of selected solutions to the Capella model. The rationale for positioning the ADS between the solution-agnostic and solution-specific layers of the model is discussed before detailing each stage.

The Arcadia method separates solution-agnostic layers (Operational Analysis (OA) and SA), where the system is treated as a black box defined only by its high-level functions and its interactions with stakeholders, from solution-specific layers (LA and Physical Architecture (PA)), where the system's inner workings are developed at increasing specificity [27]. The LA layer refines the SA functions and allocates them to generic components, and the PA layer develops the architecture into a specification detailed enough to build. A system's architecture decisions are therefore captured in the solution-specific layers. The bulk of the architecture decision-making process takes place *between* the solution-agnostic and solution-specific layers.

The methodology proposed in this work makes use of this observation to place an ADS between the solution-agnostic and solution-specific layers of the models. Placing the ADS here allows a common understanding of the system's behavior and expectations (as represented in the OA and especially the SA layer) while simultaneously capturing the variability and breadth of potential options that can be brought about in the LA and PA layers. This way, instead of manually developing and evaluating a few architectures, it becomes possible to formulate and run an optimization problem that quantitatively compares thousands of architectures without leaving the MBSE environment. Figure 1 illustrates this methodology.

Although this work focuses on the Arcadia method, other well-known MBSE methods like the Object-Oriented Systems Engineering Method (OOSEM) [44] or the Requirements-Functional-Logical-Physical (RFLP) [45] framework use similar notions around system perspectives at different levels of

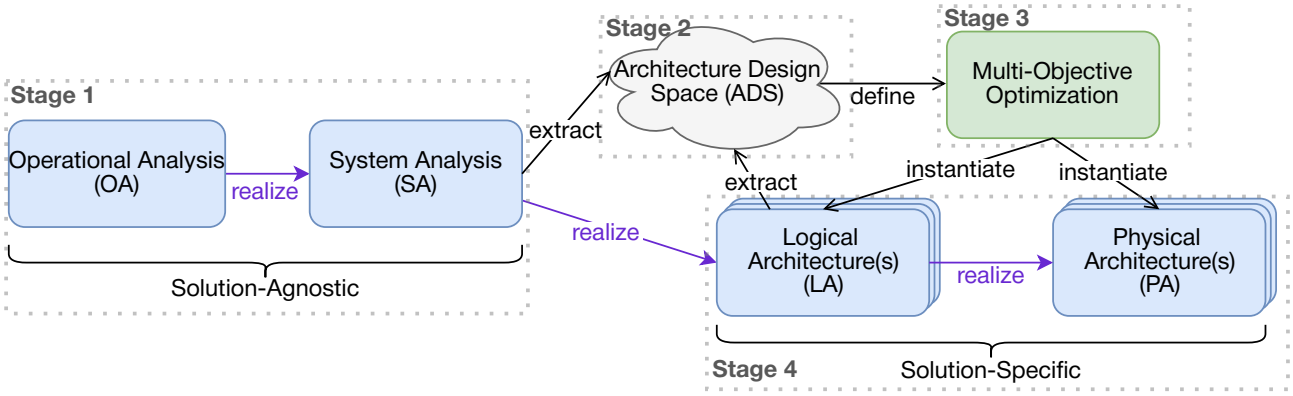


Figure 1 – General MBSE architecture optimization framework in Arcadia: an *Architecture Design Space (ADS)* is introduced between the solution-agnostic and solution-specific layers to enable systematic exploration of the architecture choices that can be made at the LA and PA layers.

abstraction that can be exploited to introduce the ADS at the appropriate level. The overall process is illustrated in Figure 1.

3.1 Stage 1: Solution-Agnostic Model Development

The goal of this stage is to understand what the system under development must accomplish to fulfill the needs and expectations of its stakeholders. In Arcadia, the OA layer focuses exclusively on the operational users and their activities, without yet depicting the system [27]. This provides flexibility in defining the system's scope or boundary later on and allows the modeler to fully capture how the system stakeholders would like to operate. The next layer (from a top-down perspective) is the SA. This layer defines the system boundary and determines the system's functions as well as how the system interacts with external actors [27]. At this level, although an understanding of what the system must do is developed, its internal workings are not yet defined. The Arcadia method is flexible enough that no layer is mandatory, so the solution-agnostic model development stage can develop only an SA layer without going through an OA layer first. This is the approach taken in Section 4 to illustrate the process introduced in this paper. However, at a minimum, a set of system functions (and their interactions, represented by *functional exchanges*) must be established in the solution-agnostic stage of the process. As an example, Figure 2 shows a simplified version of this stage for the PFCS design problem shown in Section 4. In this case, the system has three functions: *Calculate Commands*, *Transmit Commands*, and *Actuate Surfaces*. These functions, and the exchanges that link them

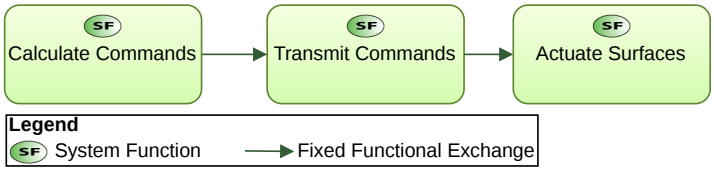


Figure 2 – Solution-Agnostic Model Development in Capella. In this simplified case, only system functions and their exchanges are defined in the model. More involved models at this point may include external actors, functional chains, and property values, as well as an upstream Operational Analysis (OA) layer.

together, form the basis for the subsequent ADS formulation stage.

3.2 Stage 2: Architecture Design Space Formulation

The model populated only with solution-agnostic elements developed in the previous stage is the starting point for the definition of the ADS. These elements are realized and refined in the LA and PA layers with a higher level of specificity. This stage is the most involved one in the entire process because it encodes the choices that the systems architect wants to evaluate systematically. The process supports three types of architectural decisions:

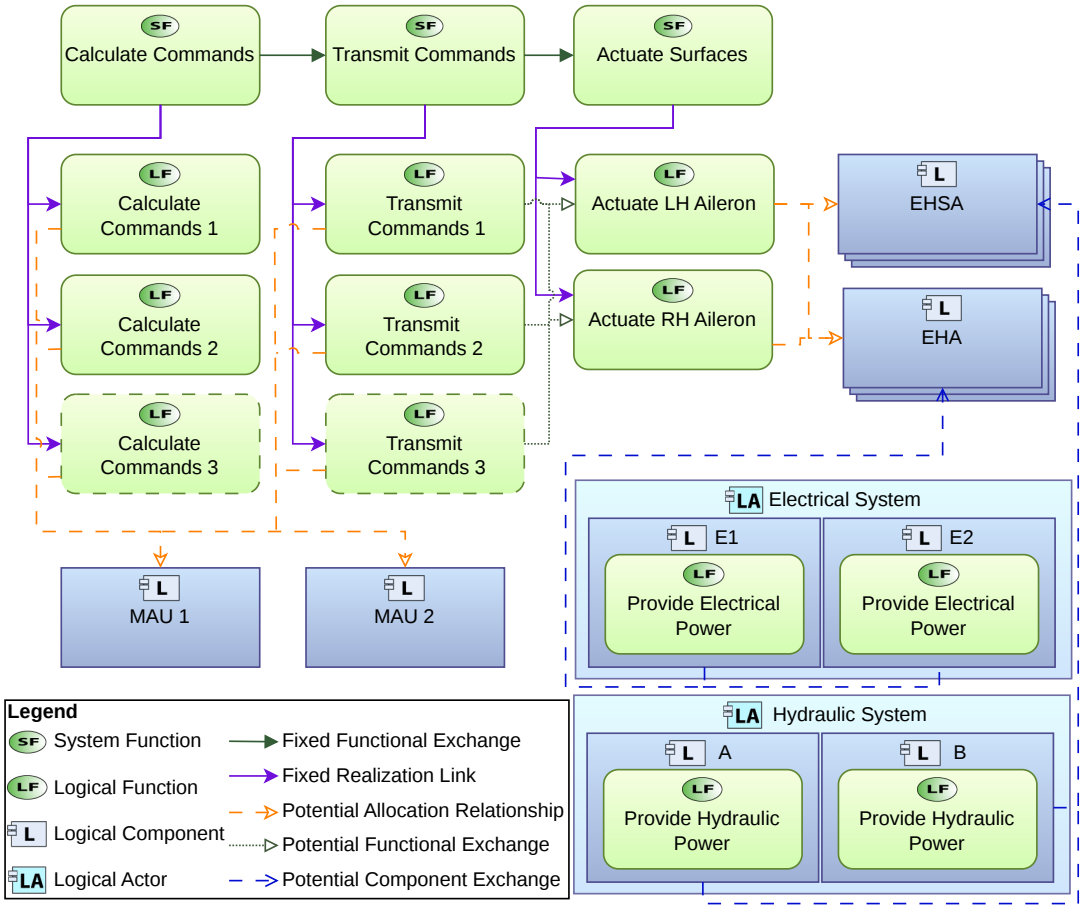


Figure 3 – Simplified Arcadia-oriented Architecture Design Space (ADS) view. *MAU* = Modular Avionics Unit; *EHSA* = Electro-Hydraulic Servo Actuator; *EHA* = Electro-Hydrostatic Actuator.

- **Function and component instantiation:** decide whether to instantiate a component or function. Useful for studies including redundancy considerations.
- **Function allocation:** choose which component a particular function will be allocated to. Useful for architectures with multiple redundant components or for technology selection choices.
- **Functional and component exchange creation:** given a pair of functions or components, decide whether to connect them with an exchange or not.

These choices can be represented with the *decision-option* and *connecting* architectural decision patterns identified by Selva et al. [46]. This framing is deliberate, as it makes it possible to represent the ADS as either an `ADORE` model or, more generally, as a Design Space Graph (DSG) using the open-source package `adsg-core`[1] developed by Bussemaker [34]. A DSG is a directed graph in which *selection choice* nodes represent discrete architecture decisions and edges encode structural dependencies between them. For the implementation of this work, a general DSG is adapted drawing inspiration from `ADORE` to represent the ADS in an Arcadia and Capella-compatible manner.

A potential ADS formed from the solution-agnostic model shown in Figure 2 and showcasing the decision types mentioned above is shown in Figure 3. This ADS[2] represents the design space for a partial PFCS architecture, considering only the aileron actuation. The *Calculate Commands* SA-level function is realized by a set of redundant functions in the LA layer. The dashed border of the *Calculate Commands 3* logical function indicates that it is optional – some architectures will only include two *Calculate Commands* functions at the LA level, while others will include three. A similar situation happens for the *Transmit Commands* SA function. This illustrates the function instantiation

[1] `https://github.com/jbussemaker/adsg-core`. Retrieved 2026-05-21.

[2] In this view, unlike most Capella diagrams, model elements from multiple layers are represented in the same diagram. This is inspired by Naval Group's Cross-Level Analysis add-on for Capella [47].

choice type mentioned above. Figure 3 also shows two Modular Avionics Units (MAUs) which can host any of the *Calculate* or *Transmit* functions. The orange dashed arrows represent potential allocation relationships of these functions to either of the two MAUs. This case represents the function allocation choice discussed above. The ADS representation in Figure 3 also shows potential functional exchanges or component exchanges. The dashed green arrows between the *Transmit* functions and the *Actuate* functions represent transmitter-to-actuator connections, while the dashed blue arrows between the electrical/hydraulic systems and the different actuators represent potential power system-to-actuator connections. These are examples of the functional and component exchange creation choices mentioned above.

For this work, the general DSG `NamedNode`s are specialized with Capella model element metadata. Component nodes (further specialized as layer-specific components) represent system elements or enclosing subsystem clusters (e.g., distribution networks, processing units). Similarly, Function nodes (further specialized as layer-specific functions) represent Capella functions. Both the Function and Component nodes in the specialized DSG contain enough information to be correctly instantiated in the Capella model from Stage 1 and to preserve traceability via cross-layer realization links.

The DSG is compiled into a design vector by `adsg-core`'s graph encoder. Each element of the vector corresponds to one selection choice or one derived connection decision; the encoder maps any valid vector back to a DSG instance for evaluation. Decisions that are structurally inactive for a given instance are *imputed* automatically, so the optimizer operates on a fixed-length vector regardless of which decisions are made [48].

3.3 Stage 3: Multi-Objective Optimization

Once the ADS has been completed, the design problem can be converted to an optimization problem. The `adsg-core` package provides a `DSGEvaluator` class that can be subclassed to incorporate the evaluation logic, including interfacing to external analysis tools, such that each combination of design decisions yields a set of numeric objectives and constraints.

The specialized evaluator can be incorporated into a custom problem definition inheriting from `adsg-core`'s `DSGArchOptProblem` class. This problem can then be solved using the `SBArchOpt` package [49], which wraps popular Python-based optimization libraries for use with `adsg-core` or `ADORE`, or directly with one of these libraries, including, for example, `pymoo` [50] for population-based multi-objective optimization algorithms.

3.4 Stage 4: Write-Back of Selected Solutions to the Capella Model

After the optimization run, each point on the non-dominated front (and any other point evaluated during the run) corresponds to a pruned DSG, which maps to a Capella LA/PA layer. The `py-capellambse` Python library [51] is used to write selected solutions as their own Capella models with updated components, function allocations, and functional and component exchanges. Additionally, quantitative information either at the component or system level is saved to the model using the Property Values Management Tool (PVMT) add-on, which is also included in `py-capellambse`. Because the DSG nodes carry Capella metadata from the outset, no intermediate translation step is required between the optimization result and the source model.

This step closes the loop between the MBSE model development and the quantitative architecture search: the SA layer provides the system needs and functional context, the Architecture Design Space represented as a DSG encodes the decision space and preserves a traceable link to the Capella model, the optimizer searches the space, and the result is written back as a model initialized from the upstream solution-agnostic model, but now populated with the solution-specific model elements in the LA and PA layers. The resulting model can inform more decisions in the upper layers, and the process can be repeated and refined as needed as more information on the system becomes available. The goal of placing the ADS and the optimization procedure between the solution-agnostic and solution-specific sections of the model is not to give a "final word" on what the system's architecture should be. Rather, it is to aid system architects in making more informed and thorough decisions, supported by a systematic search of the available design alternatives instead of relying purely on expert judgment and experience from past programs.

4. Case Study: Primary Flight Control System Architecture

The methodology described in Section 3 is applied to the Primary Flight Control System (PFCS) of a large business jet whose baseline architecture and system data are drawn from documentation for the Dassault Falcon 7X [52–54]. The specific geometry and system sizing parameters used in the analysis are estimates, so the baseline configuration used in the test case is an approximation of the as-built aircraft. Heuristics such as forcing symmetrical actuation technologies on opposite control surfaces also affect the overall architecture. While a preliminary safety filter could restrict the analysis to certifiable architectures [5, 55], this step is omitted here for simplicity.

The purpose of this test case is to demonstrate the methodology developed rather than to find an optimal PFCS architecture.

4.1 System Description

The Dassault Falcon 7X PFCS controls aircraft attitude on the three axes of motion through eight primary control surfaces: two elevators, two ailerons, one rudder, two multifunctional spoilers[3], and one Trimmable Horizontal Stabilizer (THS)[4]. The ailerons, elevators, and rudder are each driven by two actuators. Meanwhile, each spoiler is driven by a single actuator. The technology type, electrical system, and hydraulic system connections are the subject of this study. Additionally, the computing architecture of the FBW system, where mechanical links from the pilot inputs to the control surface actuation are completely replaced by electronic signals, is considered in the architecture tradespace. For this study, the FBW avionics are considered as part of an IMA platform which comprises six MAUs similar to the ones offered in the Honeywell Primus Epic avionics suite [56]. These MAUs are compact cabinets with double backplanes capable of hosting several modules, such as processors, power supplies, and I/O [56]. However, for this problem, the choice of module is not modeled; rather, only the allocation of a given function to a given MAU is considered.

Although the original Falcon 7X's FBW components use dedicated hardware separate from the main avionics suite, other modern FBW aircraft like the Embraer E190 host some flight control functions within MAUs [57]. Therefore, future aircraft may likely host the entire FCS avionics on an IMA platform.

A notional layout of the aircraft MAUs and actuators considered in this study is shown in Figure 4. The MAUs are located in the front, central, and aft equipment bays. As shown in Figure 4, the aircraft has six MAUs and twelve actuators. All six MAUs communicate with each other via an Avionics Standard Communications Bus version D (ASCB-D) network. The communication between the MAUs and actuators is carried out via dedicated ARINC 429 buses. The PFCS power comes from four independent electrical systems (designated E1, E2, E3, and E4) and three independent hydraulic systems (designated A, B, and C) [53, 54].

This system description is the starting point for the method explained in Section 3, as it limits the scope of the system and already indicates the functions that it will have to fulfill.

4.1.1 Modeling Assumptions

The model uses the following simplifying assumptions:

- **Fixed surface and actuator inventory:** Excluding the THS, which is not considered in this test case, the aircraft has seven primary control surfaces driven by twelve actuators (two per elevator, aileron, and rudder; one per spoiler). Left and right sides of each paired surface share the same actuator technology type, but per-slot electrical system and hydraulic system assignments may differ between sides. The number of surfaces and actuators is not a design variable.
- **Fixed power system topology:** The three hydraulic systems and four electrical systems described above are given. Adding or removing power systems is outside the problem scope.
- **Fixed six-MAU IMA platform:** Six MAUs at fixed airframe locations are assumed. Only the redundancy and allocation of the system's avionics-related functions (*Gather Data*, *Calculate Normal Law*, *Calculate Direct Law*, *Select Commands*, *Transmit Commands*) to those MAUs, and

[3]The Falcon 7X has a total of 6 spoilers, but only two of them, marked on Figure 4, assist in roll control [52].

[4]Dassault includes the THS as part of the PFCS [52], but this component is neglected in this study.

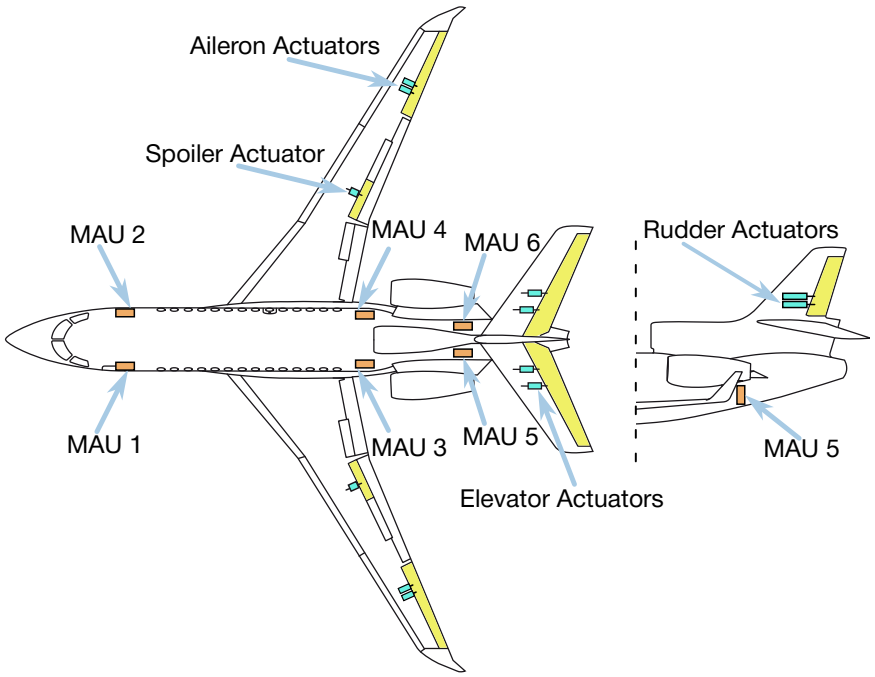


Figure 4 – Notional layout of Modular Avionics Units (MAUs) and actuators of the Dassault Falcon 7X-based PFCS considered in this study. The relevant control surfaces are shaded yellow; the others are for secondary flight control.

the ARINC 429 wiring from *Transmit Commands*-hosting MAUs to each actuator, are considered in the Architecture Design Space.

- **Four actuator technology options:** Each actuator slot can be assigned one of the following technologies [58]:
  - *Electro-Hydraulic Servo Actuator (EHSA)*: hydraulic servo actuator driven by a hydraulic system.
  - *Electrical Backup Hydraulic Actuator (EBHA)*: primarily hydraulic, with a local electric motor-pump providing backup pressure.
  - *Electro-Hydrostatic Actuator (EHA)*: self-contained actuator with a dedicated internal motor-pump; draws only from the electrical system, not from the shared hydraulic system.
  - *Electro-Mechanical Actuator (EMA)*: purely electric, using a ball-screw or roller-screw drive.

  Actuator mass and power are sized by an OpenMDAO [59] Multidisciplinary Design Analysis (MDA) using estimated Falcon 7X geometry data and based on an adapted version of Aircraft Systems Sizing Estimation Tool (ASSET), presented in [15].

- **Quasi-static eight-phase power model:** Power demand is sampled at eight discrete flight phases (ground, taxi, take-off, climb, cruise, descent, approach, landing) under quasi-static assumptions.

- **Per-flight-hour reliability with engine common cause:** All component failure rates are constant rates per flight hour. Two classes of structural common-cause dependency are captured. On the power side, the eight engine-up/down states (2^3 combinations) are enumerated exactly, so a single engine outage that simultaneously disables multiple hydraulic pumps or electrical generators is reflected in the joint failure probability. On the avionics side, failure of the upstream avionics functions disables every command path at once. This contribution is included as an additive term in each failure rate metric. All remaining components are treated as failing independently within their respective domains. Common-cause events that cross the power/avionics boundary (e.g., an electrical ssystem failure simultaneously de-energizing an actuator and

cutting power to an MAU) are not modeled. The electrical power supply to the MAUs is also not considered.

- **Manhattan distance-based ARINC 429 wire weight estimation:** The weight of the ARINC 429 wiring needed to connect the different MAUs to the actuators is estimated from the Manhattan distance[5] between the relevant MAU and its corresponding actuators. A more advanced wire routing tool like the one presented in [61] could be incorporated in the future.

The following subsections describe the process explained in Section 3 applied to this test case.

4.2 Stage 1: Solution-Agnostic Model Development

The OA layer is omitted from this test case for simplicity. At the SA layer, five basic functions of the PFCS are identified, all coming from the top-level function *Control Aircraft in Flight*. These functions come from Dassault's description of the PFCS, but could have also been developed after a functional analysis process like the one shown in [62]. The five PFCS functions specified at the SA layer in the Capella model are:

- **Gather Data**: Acquire pilot inputs and inertial, air-data, and other sensor measurements.
- **Calculate Commands**: Apply control laws to compute surface deflection commands.
- **Select Commands**: Select the active command source.
- **Transmit Commands**: Send the selected actuator commands over the ARINC 429 avionics bus to the actuators.
- **Actuate Surfaces**: Move the aircraft's control surfaces at the required deflections.

Since the power supply for the PFCS is also modeled, the *Provide Power* function is included too at this stage. However, it is delegated to an abstract external actor representing the power systems, which is later realized at the LA level as the hydraulic and electrical power systems, each with their own *Provide Electrical Power* and *Provide Hydraulic Power* functions following the baseline architecture.

The resulting Capella model is similar to the one shown in Figure 2, but with the additional *Gather Data*, *Select Commands*, and *Provide Power* functions that were not considered in that example.

4.3 Stage 2: Architecture Design Space Formulation

The ADS for this problem is formulated starting from the SA layer developed in the previous stage. The system functions from the SA layer are refined at the LA layer. Here, every SA function is made redundant, and it becomes possible to choose the number of redundant instances a given function will have. Additionally, the *Calculate Commands* function at the SA layer is decomposed into *Calculate Normal Law* and *Calculate Direct Law*. This aligns with FBW architectures where dissimilar Flight Control Computers (FCCs) with different control law capabilities are implemented in the aircraft, and with the roles of the main and secondary FCCs present in the Falcon 7X [52].

Similar to the smaller example shown in Figure 3, the ADS for this problem comprises function instantiation decisions that determine the redundancy for each of the basic functions established in the previous stage. Additionally, the architecture design space includes decisions regarding which MAU to allocate these functions to, how to connect the *Transmit Commands* function instances to the control surface actuators, the actuator technology choice, and the connection between the available power systems and the actuators. The full ADS for this problem is shown in Figure A1.

4.4 Stage 3: Multi-Objective Optimization

Once the ADS has been defined, an optimization problem can be formulated to aid in the selection of one architecture to develop further. The optimization problem described here has three objectives, 38 constraints, and 133 discrete design variables.

[5]The Manhattan distance (also known as *rectilinear distance* or *taxicab distance*) between two points is defined as the sum of the absolute differences of their respective Cartesian coordinates [60].

4.4.1 Objectives

The optimization problem to be addressed with the ADS described above has the following design objectives to be minimized:

1. **Total system mass** m [kg]: the sum of actuator mass, hydraulic and electrical system masses sized for the PFCS consumers' power demand, and ARINC 429 bus harness mass.
2. **Peak electrical or hydraulic power demand** P [kW]: the maximum FCS power draw across all three hydraulic systems, all four electrical buses, and eight flight phases.
3. **System failure rate** λ [1/FH]: the average probability per FH, λ, of the PFCS failure condition is computed via an exact enumeration of the joint power and avionics basic event spaces. Because failure rates span several orders of magnitude, the optimization minimizes $\log_{10}(\lambda)$. Prevention of the failure condition related to this objective requires at least one operative actuator on each control axis. Upstream common cause failures are accounted for implicitly such that total upstream failure combinations contribute directly to the overall failure condition probability. The component failure rates are compiled from the values published in [55, 58, 63].

4.4.2 Design Variables

When the design vector is flattened, it has 133 entries, decomposed as follows:

- **Actuator technology** (7 variables): one technology type selection per actuator slot across the four surface groups; two slots per elevator, aileron, and rudder; one slot per spoiler. The left and right sides share the same technology type. The four technology options are EHSA, EBHA, EHA, and EMA as described in Section 4.1.1.
- **Electrical system assignment** (12 variables): one bus selection per actuator slot from the four FCS electrical systems; active only for EMA, EHA, and EBHA actuators.
- **Hydraulic system assignment** (12 variables): one selection per actuator slot from the three aircraft hydraulic systems (A, B, C); active only for EHSA and EBHA actuators.
- **IMA function allocation** (30 variables): assignment of each of the refined LA-layer functions to each of the six MAUs.
- **Avionics wiring** (72 variables, conditionally active): Connection choices indicating whether MAU i is wired to actuator j via a functional exchange.

4.4.3 Constraints

The optimization problem includes 38 inequality constraints, summarized in Table 1. The failure conditions for the λ constraints and their maximum allowable average probabilities are taken from the Functional Hazard Assessment (FHA) performed in [40].

The function-count bounds are $[3, 5]$ for *Gather Data*, $[3, 5]$ for *Select Commands*, and $[3, 6]$ for *Transmit Commands*. The *Calculate Direct Law* and *Calculate Normal Law* functions shall occur exactly three times each, in line with common practice for FBW architecture development [55].

4.4.4 Baseline Architecture

The baseline described in Section 4.1 is used as a reference to compare the generated architectures. It corresponds to an architecture with EHSA actuators on all control surfaces except the spoilers, which use EBHAs, and matches the description of the Falcon 7X primary flight control architecture [52]. The *Transmit Commands* SA-level function is duplicated four times at the LA level and hosted on four of the six MAUs, with assumed connections that satisfy the two-MAU redundancy requirement per actuator. The baseline evaluates to a total mass of $m_0 = 1098$ kg, a peak power demand of $P_0 = 34.0$ kW, and a system failure rate of $\lambda_0 = 3.3 \times 10^{-8}$/FH. All constraints in Table 1 are satisfied. The location of the baseline architecture in the objective space of evaluated architectures after the optimization run described in Section 4.4.5 is shown in Figure 5.

Table 1 – Inequality constraints in the PFCS architecture problem.

Constraint family	Condition	Count
Source-clash avoidance	Two actuators on the same surface must not share a power source	5
Loss of yaw control with One Engine Inoperative (OEI) [40]	$\lambda_{\text{OEI-yaw}} \leq 10^{-9}$ /FH	1
Loss of both elevator surfaces [40]	$\lambda_{\text{both-elev}} \leq 10^{-9}$ /FH	1
Loss of one elevator [40]	$\lambda_{\text{one-elev}} \leq 10^{-5}$ /FH	1
Loss of both ailerons [40]	$\lambda_{\text{both-ail}} \leq 10^{-3}$ /FH	1
Complete loss of roll control [40]	$\lambda_{\text{roll}} \leq 10^{-9}$ /FH	1
Function-count lower bound	Each redundant function hosted by at least $n_{\min}$ MAUs	5
Function-count upper bound	Each redundant function hosted by at most $n_{\max}$ MAUs	5
Normal/Direct law separation	*Calculate Normal Law* and *Calculate Direct Law* not co-hosted on any single MAU	6
Actuator MAU connectivity	Each actuator commanded by at least 2 MAUs	12
Total		**38**

4.4.5 Optimization Results

A mixed-discrete Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [64] based on `pymoo`'s [50] implementation ran for 20 000 function evaluations with a population of 100 using the DSG problem formulation specified above. The initial population was randomly sampled, and one of its individuals was replaced with the baseline configuration's design vector. Of the 20 000 evaluated designs, 6 389 (32%) were feasible. The non-dominated front contains 85 architectures. Figure 5 shows an objective space visualization of all the designs evaluated during the optimization run.

As shown in Figure 5, the non-dominated front spans a mass range of 945–1438 kg, a peak power range of 17.4–45.5 kW, and a failure-rate range from 5.4×10^{-11}/FH to 4.0×10^{-8}/FH. The baseline Falcon 7X configuration lies outside the non-dominated front.

A trade-off between the three objectives can be seen in the non-dominated front. For example, lighter architectures in the front tend to have lower reliability. This occurs because factors that improve reliability, such as additional connections between MAUs and actuators, also penalize the total system weight. Furthermore, the actuator technology choice also produces a trade-off of objectives. For instance, EBHA-heavy architectures produce low failure rates due to their redundant design (essentially combining an EHA and an EHSA in a single package [58]) at the expense of higher weight. Similarly, although several architectures in the non-dominated front favor EMAs due to their lower mass and peak power consumption, these architectures suffer from higher failure rates.

4.5 Stage 4: Write-Back of a Selected Architecture to the Capella Model

Out of the 85 architectures in the non-dominated front, one is selected to illustrate the fourth step of the methodology outlined in Section 3. Several multi-criteria decision making techniques are available to pick such an architecture [65]. However, for this test case, a more pragmatic approach is taken based on the choice of actuation technology. Several designs in the non-dominated front feature EMAs in most of the control surfaces. Nevertheless, the EMA failure mode of actuator jamming causes load blocking and is typically unacceptable in critical applications such as primary flight controls [66]. The jamming failure modes are not captured by the reliability model, and no amount of avionics redundancy can override the jam. Furthermore, there is no precedent for EMA certification on primary roll or yaw surfaces at the authority levels required by a large business jet. The architecture selection therefore

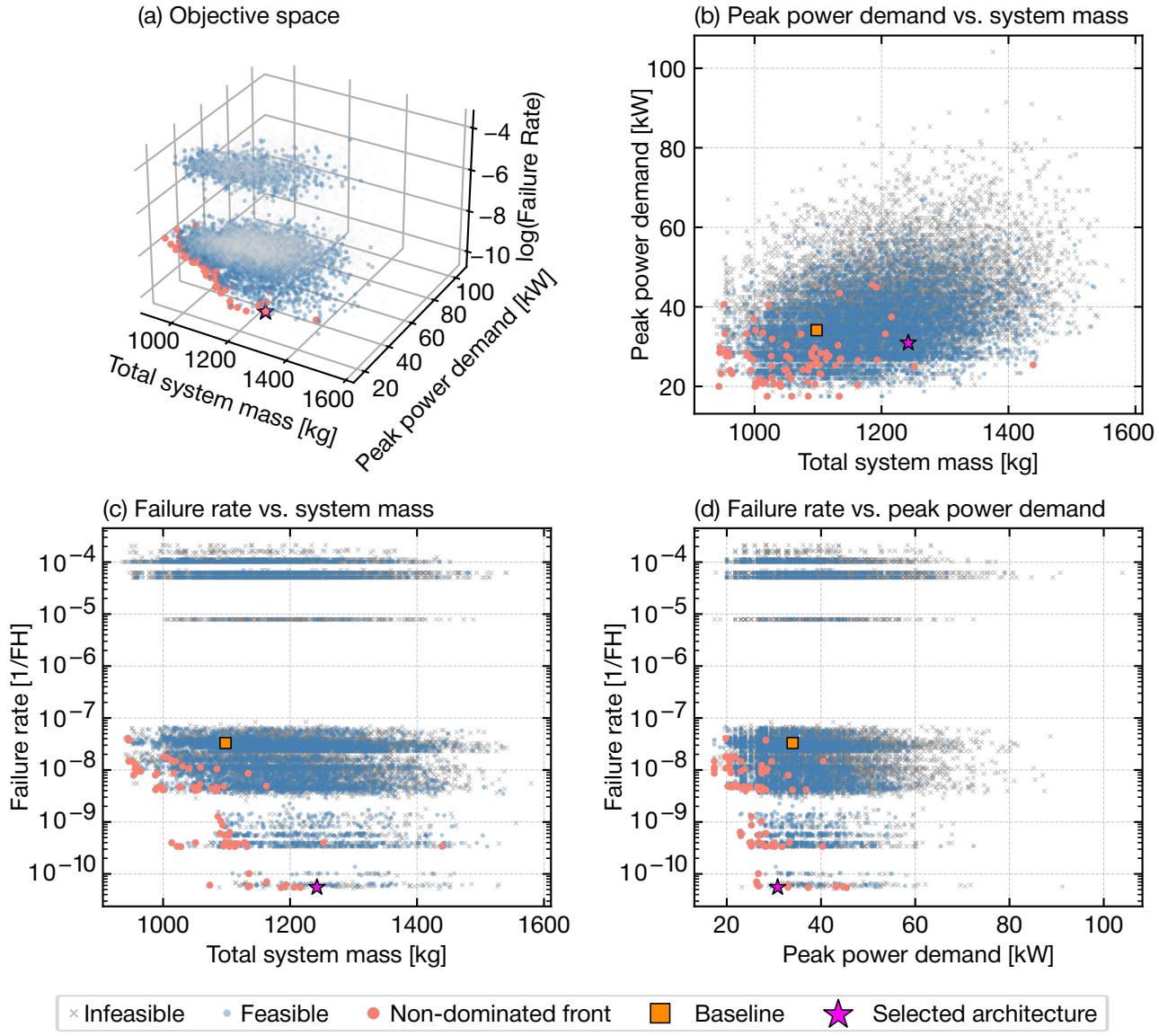


Figure 5 – Objective space visualization of the optimization run. (a) Three-dimensional view of the objective space. (b) Peak power demand vs. mass. (c) PFCS failure rate vs. mass. (d) PFCS failure rate vs. peak power demand.

restricts the candidate pool to designs with no EMA in any actuator slot.

In the obtained non-dominated front, only one architecture is entirely EMA-free. Thus, this architecture is chosen for write-back to the Capella model and for further study and comparison with the baseline. Its position on the non-dominated front and on the general objective space is shown in Figure 5.

Table 2 compares the main features of the baseline architecture and the selected no-EMA architecture from the optimization's non-dominated front. The EMA-free design trades a modest mass penalty for a large safety margin and a somewhat reduced peak power consumption. At the same time, all five safety constraints hold well inside their limits.

Compared to the baseline architecture, the selected architecture replaces the second elevator EHSA actuator with a self-contained EHA unit and changes the rudder actuators from EHSA to EBHA. Meanwhile, ailerons and spoilers retain their baseline types (EHSA and EBHA, respectively). The actuator technology selections and MAU functional allocations for the baseline design and the selected architecture are also listed in Table 2.

As shown in Table 2, the low failure rate of the selected architecture can be explained by two

Table 2 – Comparison of the baseline and selected EMA-free architectures. Power system masses include only components directly related to the PFCS. Function rows list the MAUs (1-6) that host each function.

Quantity	**Baseline**	**Selected**	**Change**
Objectives			
Total mass m [kg]	1098.4	1242.2	+13.1%
Actuators [kg]	745.6	808.3	+8.4%
Hydraulic system [kg]	255.8	257.2	+0.5%
Electrical system [kg]	78.9	146.2	+85.3%
ARINC 429 mass [kg]	18.1	30.5	+68.5%
Peak power P [kW]	34.0	30.8	-9.4%
Failure rate λ [/FH]	3.3×10^{-8}	5.5×10^{-11}	590$\times$ lower
Safety constraints [/FH]			
Loss of yaw control with OEI	7.7×10^{-13}	4.5×10^{-15}	Still $< 10^{-9}$
Loss of both elevator surfaces	1.1×10^{-12}	4.7×10^{-14}	Still $< 10^{-9}$
Loss of one elevator	6.6×10^{-8}	1.4×10^{-8}	Still $< 10^{-5}$
Loss of both ailerons	1.1×10^{-12}	1.1×10^{-12}	Still $< 10^{-3}$
Complete loss of roll control	7.2×10^{-16}	1.1×10^{-15}	Still $< 10^{-9}$
Actuator technology by surface			
Elevator	EHSA / EHSA	EHSA / EHA	
Aileron	EHSA / EHSA	EHSA / EHSA	
Rudder	EHSA / EHSA	EBHA / EBHA	
Spoiler	EBHA	EBHA	
MAU function assignment (hosting MAUs)			
Gather Data	1, 3, 4, 5, 6	1, 2, 3, 4, 5	
Calculate Normal Law	1, 2, 3	1, 3, 4	
Calculate Direct Law	4, 5, 6	2, 5, 6	
Select Commands	3, 4, 5, 6	1, 3, 4, 5	
Transmit Commands	3, 4, 5, 6	1, 2, 3, 4, 5	

technology changes. The baseline fits EHSAs for every surface except the spoilers, which use EBHAs. Meanwhile, the selected design keeps those choices except for one elevator slot, now EHA, and both rudder slots, now EBHA. The EBHA rudder adds an independent electric motor pump to each of its actuators, which, together with the EHA in the elevators, explains most of the failure rate drop and the heavier electrical system. In contrast, function hosting moves far less: the count of MAUs per function is nearly fixed between the baseline and the selected design, with only *Transmit Commands* widening from four duplications to five.

Figure 6 compares the connections between MAUs and actuators by displaying these connections in design structure matrices, where • entries mean there is a connection from the given MAU row to the corresponding actuator column. These matrices show the differences between the two designs regarding the spread of actuator authority. While the baseline uses only four of the six available MAUs and assigns each actuator exactly two *Transmit Commands* functions, the selected architecture uses five of the MAUs and connects either two or three MAUs to each actuator. In this architecture, MAU 3 is highly loaded, with connections to 11 of the 12 actuators.

The information used to construct the various representations of the selected architecture shown in Table 2 and Figure 6 is also stored in the Capella model generated automatically with the methodology developed in this work. This means that all function allocations, connections to power sources, and functional exchange definitions are saved to the model for further inspection or development. As an example view, the LA-level architecture is represented as a Capella Logical Architecture Blank (LAB) diagram in Figure 7. Here, it is shown that the architecture is completely retrieved and correctly

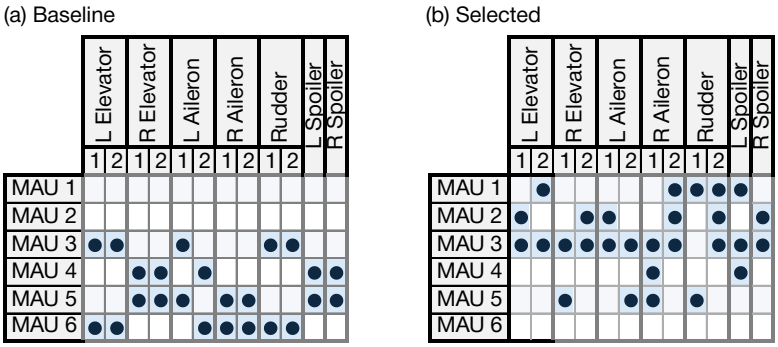


Figure 6 – MAU-to-actuator command routing as a design structure matrix. A dot marks that the MAU (row) issues commands to the actuator (column) via a functional exchange from a *Transmit Commands* function to an *Actuate* function. (a) Baseline architecture. (b) Selected non-dominated architecture.

represented in Capella due to the write-back process from the DSG representation, which includes the necessary metadata to populate the model. Quantitative information, like the design objective values, is also stored in the model.

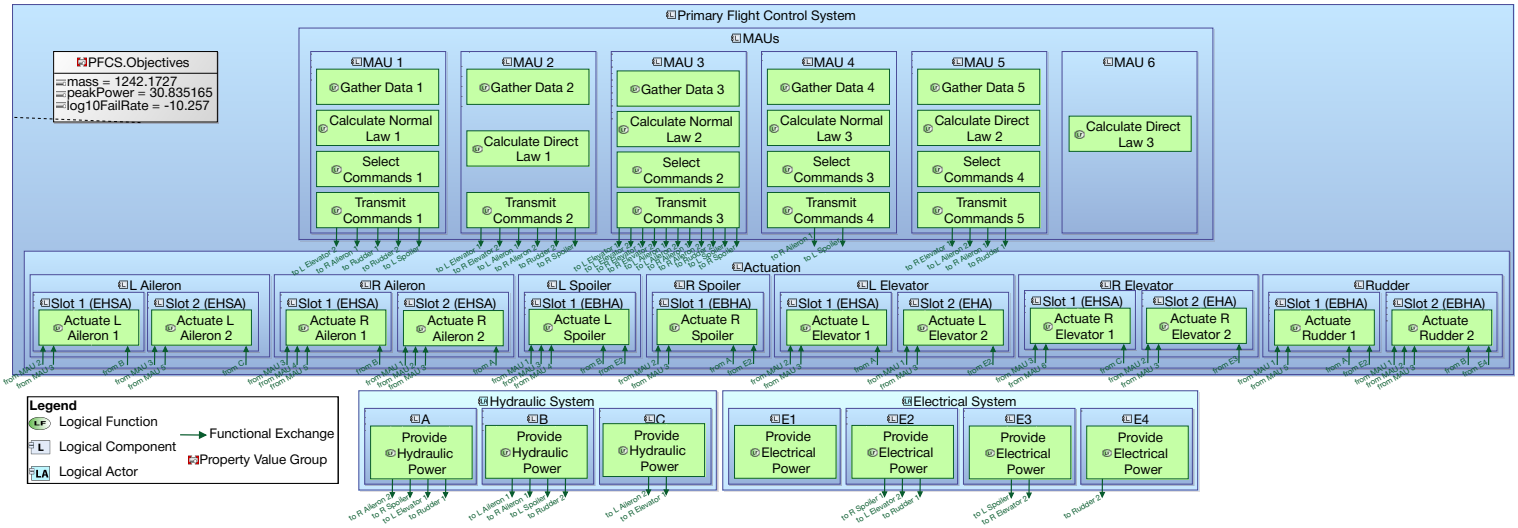


Figure 7 – Logical Architecture Blank (LAB) diagram of the selected no-EMA architecture in Capella. Only the functional exchanges that were part of the architecture design space are shown. Component exchanges, function and component ports, and functional exchange names are hidden for clarity.

This test case illustrates how the developed process ties together MBSE activities with SAO by placing SAO naturally within the MBSE development stages and perspectives.

5. Conclusions and Future Work

This paper presents a methodology that places System Architecture Optimization (SAO) within the Arcadia MBSE method, and its associated Capella tool, by introducing an Architecture Design Space (ADS) between the solution-agnostic and solution-specific model layers. A multi-objective optimizer is used to search the ADS, and selected designs are automatically written back to Capella as fully populated LA or PA layers. This process is applied to a Primary Flight Control System (PFCS) case study.

The main contribution of this work is methodological: it clearly places system architecture optimization activities within the Arcadia MBSE method (and other similar multi-layer approaches). The framework proposed in this paper enables systematic architecture decision making inside an MBSE environment. Additionally, with the specialized DSG nodes storing Capella metamodel metadata, traceability between upstream model elements and automatically instantiated LA or PA realizations is preserved across the optimization step, maintaining consistent and complete models even after exploring thousands of potential alternatives.

Certain limitations in this work are avenues for future research. For instance, in the test case, the optimizer searched a constrained space that yielded many infeasible designs, especially regarding the safety-related constraints. Thus, implementing generative filtering approaches based on previous work could greatly improve the proportion of feasible alternatives evaluated within a certain budget. Furthermore, the presented method could also be extended. For example, the same ADS pattern can be applied between the LA and PA layers for a more detailed second optimization step or across both of these solution-specific layers. Further work could also use the solution-agnostic OA and SA Arcadia layers to perform trade studies for novel ConOps related to, for example, Reduced-Crew Operations (RCO). Finally, although the procedure shown in this paper uses the Arcadia method, it could be generalized to other MBSE methods like OOSEM and RFLP.

6. Acknowledgments

This research is part of the Canadian collaborative research project Model-Based Aircraft Control System Architecture Optimization for Reduced-Crew Operations (MACrO), which receives funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and industry partner Bombardier Inc., under grant number 597530. The authors acknowledge the contributions of various Concordia University Aircraft Systems Lab members to the aircraft systems multidisciplinary analysis code used for the development of the test case presented in this paper.

7. Copyright Statement

The authors confirm that they, and/or their company or organization, hold copyright on all of the original material included in this paper. The authors also confirm that they have obtained permission, from the copyright holder of any third party material included in this paper, to publish it as part of their paper. The authors confirm that they give permission, or have obtained permission from the copyright holder of this paper, for the publication and distribution of this paper as part of the ICAS proceedings or as individual off-prints from the proceedings.

References

[1] Crawley E, Weck O, Eppinger S, Magee C, Moses J, Seering W, Schindall J, Wallace D and Whitney D. The Influence of Architecture in Engineering Systems. Engineering Systems Monograph, MIT Engineering Systems Division (esd), Presented March 29–31, 2004, Mar. 2004.

[2] Annighöfer B and Kleemann E. Large-Scale Model-Based Avionics Architecture Optimization Methods and Case Study. IEEE Transactions on Aerospace and Electronic Systems, Vol. 55, No. 6, p. 3424–3441, Dec. 2019.

[3] Haberfellner R, Weck O, Fricke E and Vössner S. Systems Engineering. Springer International Publishing, 2019.

[4] Crawley E, Cameron B and Selva D. System Architecture. Pearson Education, 2016.

[5] Jeyaraj A. K and Liscouët-Hanke S. A Safety-Focused System Architecting Framework for the Conceptual Design of Aircraft Systems. Aerospace, Vol. 9, No. 12, p. 791, Dec. 2022.

[6] Gaska T, Watkin C and Chen Y. Integrated Modular Avionics - Past, present, and future. IEEE Aerospace and Electronic Systems Magazine, Vol. 30, No. 9, p. 12–23, Sept. 2015.

[7] Lukić B, Ahlbrecht A, Friedrich S and Durak U. State-of-the-Art Technologies for Integrated Modular Avionics and the Way Ahead. Proc 2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC), p. 1–10, Oct. 2023.

[8] Radaei M. Mathematical Programming for Optimization of Integrated Modular Avionics. Proc SAE Technical Paper Series, March 2021.

[9] Brelje B. J and Martins J. R. Electric, hybrid, and turboelectric fixed-wing aircraft: A review of concepts, models, and design approaches. Progress in Aerospace Sciences, Vol. 104, p. 1–19, Jan. 2019.

[10] Fioriti M, Della Vecchia P, Donelli G and Hansmann P. Assessing the Integration of Electrified On-Board Systems in an MDAO framework for a small transport aircraft. Proc AIAA AVIATION 2021 FORUM, July 2021.

[11] Okonkwo P and Smith H. Review of evolving trends in blended wing body aircraft design. Progress in Aerospace Sciences, Vol. 82, p. 1–23, Apr. 2016.

[12] Oosterom W and Vos R. Conceptual Design of a Flying-V Aircraft Family. Proc AIAA AVIATION 2022 Forum, June 2022.

[13] Lim Y, Bassien-Capsa V, Ramasamy S, Liu J and Sabatini R. Commercial airline single-pilot operations: System design and pathways to certification. IEEE Aerospace and Electronic Systems Magazine, Vol. 32, No. 7, p. 4–21, July 2017.

[14] Dong L, Liu J, Sun Z, Chen X and Wang P. Resource Allocation Approach of Avionics System in SPO Mode Based on Proximal Policy Optimization. Aerospace, Vol. 11, No. 10, p. 812, Oct. 2024.

[15] Mohan V, Jeyaraj A. K and Liscouët-Hanke S. Systems Integration Framework for Hybrid-Electric Commuter and Regional Aircraft. Aerospace, Vol. 10, No. 6, p. 533, June 2023.

[16] Ramos A. L, Ferreira J. V and Barcelo J. Model-Based Systems Engineering: An Emerging Approach for Modern Systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), Vol. 42, No. 1, p. 101–111, Jan. 2012.

[17] ISO, IEC and IEEE. 24641:2023 Systems and Software engineering — Methods and tools for model-based systems and software engineering. International Standard, ISO/IEC/IEEE, May 2023.

[18] Zhang Y, Höpfner G, Berroth J, Pasch G and Jacobs G. Towards Holistic System Models Including Domain-Specific Simulation Models Based on SysML. Systems, Vol. 9, No. 4, p. 76, Oct. 2021.

[19] Nowodzienski P and Navas J. From Model-based to Model and Simulation-based Systems Architectures – achieving quality engineering through descriptive and analytical models. INCOSE International Symposium, Vol. 32, No. 1, p. 1247–1266, July 2022.

[20] Thales Group and Capella MBSE Team. Arcadia: A Tooled Method to Define, Analyse, Design & Validate System, Software, and Hardware Architectures. Technical Datasheet, Capella MBSE, n.d.

[21] Albarello N and Kim H. Application of an MBSE Approach for the Integration of Multidisciplinary Design Processes. Proc Complex Systems Design & Management, p. 85–96, 2014.

[22] Aïello O, Poitou O, Chaudemar J.-C and Saqui-Sannes P. Sizing a Drone Battery by coupling MBSE and MDAO. Proc Proceedings of the 11th European Congress Embedded Real Time Systems (ERTS 2022), pp 1–12, June 2022.

[23] Habermehl C, Höpfner G, Berroth J, Neumann S and Jacobs G. Optimization Workflows for Linking Model-Based Systems Engineering (MBSE) and Multidisciplinary Analysis and Optimization (MDAO). Applied Sciences, Vol. 12, No. 11, p. 5316, May 2022.

[24] Fouda M, Willrodt L, Almeida H, Cortez J, Hussein O, Castaneda F, Dhouib H, Dahik C, Drouet V and Brook R. An MBSE-Enabled MDAO Approach for the Conceptual Development of Complex Systems. Proc 34th Congress of the International Council of the Aeronautical Sciences - ICAS 2024, pp 1–25, Sept. 2024.

[25] De Saqui-Sannes P, Vingerhoeds R. A, Garion C and Thirioux X. A Taxonomy of MBSE Approaches by Languages, Tools and Methods. IEEE Access, Vol. 10, p. 120936–120950, 2022.

[26] Roques P. MBSE with the ARCADIA Method and the Capella Tool. Proc Proceedings of the 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Toulouse, France, Jan. 2016.

[27] Voirin J.-L. Model-Based System and Architecture Engineering with the Arcadia Method. 1 edition, ISTE Press, 2018.

[28] Timperley L, Berthoud L, Snider C and Tryfonas T. Mapping the MBSE Environment and Complementary Design Space Exploration Techniques. Proc 2024 IEEE Aerospace Conference, p. 1–20, March 2024.

[29] Frank C, Marlier R, Pinon-Fischer O. J and Mavris D. N. An Evolutionary Multi-Architecture Multi-Objective Optimization Algorithm for Design Space Exploration. Proc 57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, pp 1–19, Jan. 2016.

[30] Apaza G and Selva D. Automatic Composition of Encoding Scheme and Search Operators in System Architecture Optimization. Proc Volume 2: 41st Computers and Information in Engineering Conference (CIE), pp 1–11, Aug. 2021.

[31] Sonneveld J, Den Berg T, La Rocca G, Valencia-Ibáñez S, Manen B and Bruggeman A. Dynamic workflow generation applied to aircraft moveable architecture optimization. Proc Proceedings of the Aerospace Europe Conference - EUCASS - CEAS - 2023, pp 1–16, 2023.

[32] Bussemaker J. H, Ciampa P. D and Nagel B. System Architecture Design Space Exploration: An Approach to Modeling and Optimization. Proc AIAA AVIATION 2020 FORUM, June 2020.

[33] Bussemaker J. H and Ciampa P. D. MBSE in Architecture Design Space Exploration. Proc Handbook of Model-Based Systems Engineering, p. 1–41, 2022.

[34] Bussemaker J. System Architecture Optimization: Function-Based Modeling, Optimization Algorithms, and Multidisciplinary Optimization. PhD thesis, Delft University of Technology, Delft, the Netherlands, 2025.

[35] Bussemaker J. H, Boggero L and Nagel B. System Architecture Design Space Exploration: Integration With Computational Environments and Efficient Optimization. Proc AIAA Aviation Forum and Ascend 2024, pp 1–25, July 2024.

[36] Bussemaker J. H, Torrigiani F, Kupriyanov O, Allegaert E, Boggero L and Nagel B. System Architecture

Optimization Using SysML v2: Language Extension and Implementation. Proc 2026 IEEE International Systems Conference (SysCon), p. 1–8, Apr. 2026.

[37] Jeyaraj A. K, Tabesh N and Liscouët-Hanke S. Connecting Model-based Systems Engineering and Multidisciplinary Design Analysis and Optimization for Aircraft Systems Architecting. Proc AIAA AVIATION 2021 FORUM, pp 1–17, July 2021.

[38] Gauthier J.-M, Giret J.-C, Béchet S, Torrigiani F, Garg S and Bussemaker J. Integrating MBSE and MDAO through SysML v2. Proc 2026 International Workshop on Systems Engineering Research, p. 1–8, Jan. 2026.

[39] Bauer C, Lagadec K, Bès C and Mongeau M. Flight Control System Architecture Optimization for Fly-By-Wire Airliners. Journal of Guidance, Control, and Dynamics, Vol. 30, No. 4, p. 1023–1029, July 2007.

[40] Kreitz T, Bornholdt R, Krings M, Henning K and Thielecke F. Simulation-Driven Methodology for the Requirements Verification and Safety Assessment of Innovative Flight Control Systems. Proc SAE Technical Paper Series, Vol. 1 of *AEROTECH*, Sept. 2015.

[41] Lampl T, Wolf T and Hornung M. Preliminary design of advanced flight control system architectures for commercial transport aircraft. CEAS Aeronautical Journal, Vol. 10, No. 2, p. 613–622, Oct. 2018.

[42] Jiao Z, Yu B, Wu S, Shang Y, Huang H, Tang Z, Wei R and Li C. An intelligent design method for actuation system architecture optimization for more electrical aircraft. Aerospace Science and Technology, Vol. 93, p. 105079, Oct. 2019.

[43] George Mathew P, Liscouët-Hanke S and Le Masson Y. Model-Based Systems Engineering Methodology for Implementing Networked Aircraft Control System on Integrated Modular Avionics – Environmental Control System Case Study. Proc SAE Technical Paper Series, p. 1943, October 2018.

[44] Friedenthal S, Izumi L and Meilich A. 9.2.2 Object-Oriented Systems Engineering Method (OOSEM) applied to Joint Force Projection (JFP), a Lockheed Martin Integrating Concept (LMIC). INCOSE International Symposium, Vol. 17, No. 1, p. 1471–1491, June 2007.

[45] Baughey K. Functional and Logical Structures: A Systems Engineering Approach. Proc SAE 2011 World Congress & Exhibition, pp 1–9, Apr. 2011.

[46] Selva D, Cameron B and Crawley E. Patterns in System Architecture Decisions. Systems Engineering, Vol. 19, No. 6, p. 477–497, Nov. 2016.

[47] Naval Group SA and labs4capella. Cross-level analysis add-on for Capella (MBSE). 2022.

[48] Bussemaker J. H, Bartoli N, Lefebvre T, Ciampa P. D and Nagel B. Effectiveness of Surrogate-Based Optimization Algorithms for System Architecture Optimization. Proc AIAA AVIATION 2021 FORUM, July 2021.

[49] Bussemaker J. H. SBArchOpt: Surrogate-Based Architecture Optimization. Journal of Open Source Software, Vol. 8, No. 89, p. 5564, Sept. 2023.

[50] Blank J and Deb K. Pymoo: Multi-Objective Optimization in Python. IEEE Access, Vol. 8, p. 89497–89509, 2020.

[51] DB InfraGO AG. py-capellambse: A Python 3 headless implementation of the Capella modeling tool. `https://github.com/dbinfrago/py-capellambse`, 2026.

[52] Dassault Aviation. Falcon 7X: ATA 27 - Flight Controls. Online. `https://www.smartcockpit.com/download/falcon-7x-flight-controls/?wpdmdl=4764&refresh=6a0e7e75efd9e1779334773`, 2005.

[53] Dassault Aviation. Falcon 7X: ATA 24 - Electrical Power. Online. `https://www.smartcockpit.com/download/falcon-7x-electrical-power/?wpdmdl=4768&refresh=6a0e7e765a8011779334774`, 2005.

[54] Dassault Aviation. Falcon 7X: ATA 29 - Hydraulic System. Online. `https://www.smartcockpit.com/download/falcon-7x-hydraulic-system/?wpdmdl=4762&refresh=6a0e7e75c07df1779334773`, 2005.

[55] Jeyaraj A. K. A Safety-Focused Systems Architecting Framework for Aircraft Conceptual Design. PhD thesis, Concordia University, Montréal, Québec, Canada, May 2025.

[56] Cornilleau T. Dassault Aviation feedbacks on its military and civil IMA applications. ARTIST2 meeting on Integrated Modular Avionics. `http://www.artist-embedded.org/docs/Events/2007/IMA/Slides/ARTIST2_IMA_Cornilleau.pdf`, Nov. 2007.

[57] Regli C. Embraer EMB190 Summary. Private compilation of information and hints for Helvetic Airways Embraer E190/E195/E290/E295 models, 9 2025.

[58] Maré J.-C. Aerospace Actuators 1: Needs, Reliability and Hydraulic Power Solutions. ISTE Ltd and John Wiley & Sons, Inc., 2016.

[59] Gray J. S, Hwang J. T, Martins J. R. R. A, Moore K. T and Naylor B. A. OpenMDAO: an open-source framework for multidisciplinary design, analysis, and optimization. Structural and Multidisciplinary Optimization, Vol. 59, No. 4, p. 1075–1104, March 2019.

[60] Black P. E. Manhattan distance. In *Dictionary of Algorithms and Data Structures*, Black P. E ed., National Institute of Standards and Technology, Feb. 2019.

[61] Setty B. L and Liscouët-Hanke S. Automation of Wire Routing for Aircraft Conceptual Design. Proc AIAA SciTech 2026 Forum, Jan. 2026.

[62] Esdras G and Liscouët-Hanke S. Development of Core Functions for Aircraft Conceptual Design: Methodology and Results. Proc Proceedings of the Canadian Aeronautics and Space Institute AERO 2015 Conference, Montreal, Canada, pp 1–10, May 2015.

[63] Cabaleiro de la Hoz C. Definition, Design and Optimization of Aircraft On-Board System Architectures Considering Multiple Life Cycle Stages. PhD thesis, Politecnico di Torino, 2025.

[64] Deb K, Pratap A, Agarwal S and Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, Vol. 6, No. 2, p. 182–197, Apr. 2002.

[65] Hwang C.-L and Yoon K. Multiple Attribute Decision Making. Springer Berlin Heidelberg, 1981.

[66] Maré J.-C. Aerospace Actuators 2: Signal-by-Wire and Power-by-Wire. ISTE Ltd and John Wiley & Sons, Inc., 2017.

A. Test Case Architecture Design Space

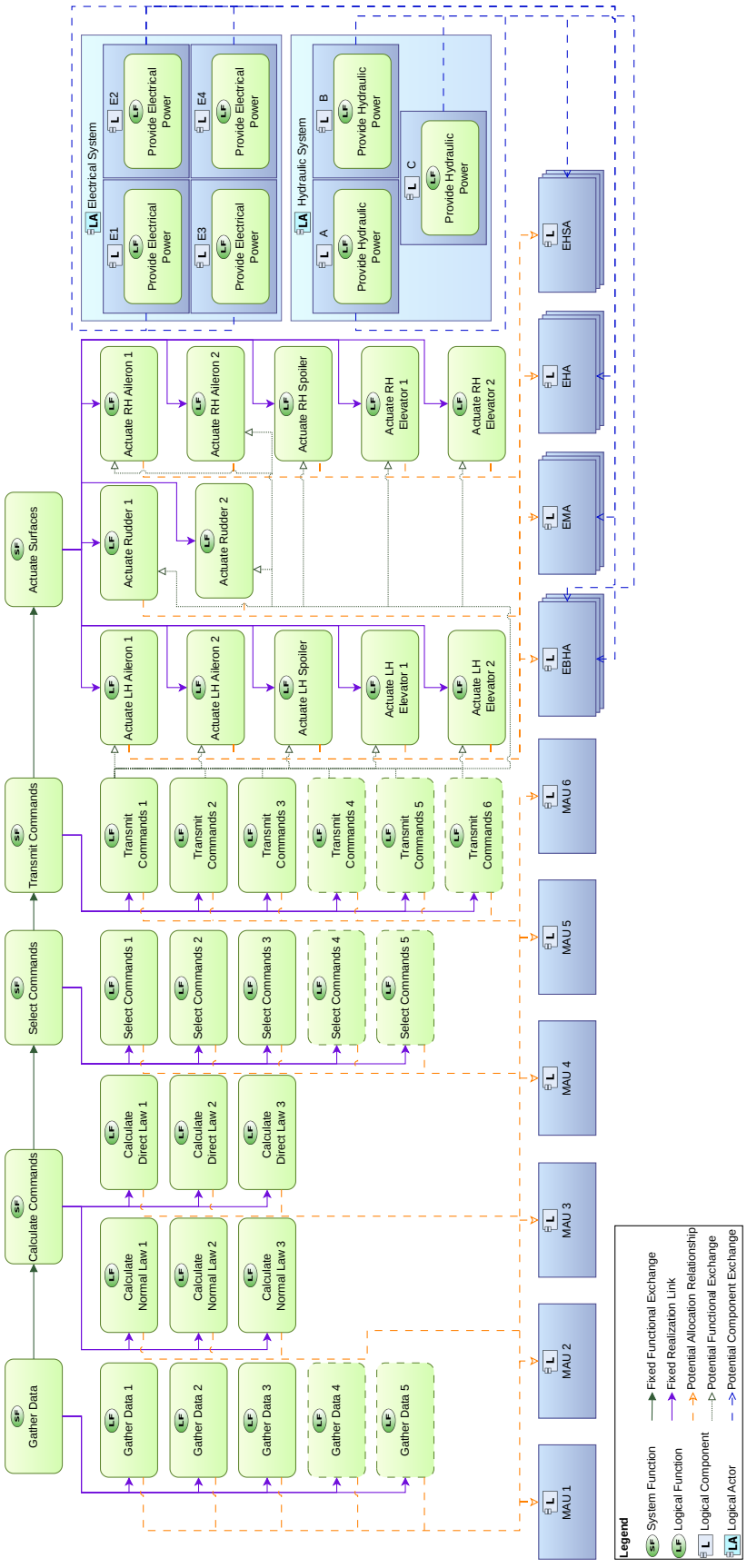


Figure A1 – Representation of the Architecture Design Space (ADS) for the test case problem. *MAU* = Modular Avionics Unit; *EHSA* = Electro-Hydraulic Servo Actuator; *EMA* = Electro-Mechanical Actuator; *EBHA* = Electrical Backup Hydraulic Actuator; *EHA* = Electro-Hydrostatic Actuator.