Dynamic Formulation and Execution of MDAO Workflows for Architecture Optimization

Sparsh Garg*, Raúl García Sánchez†, Jasper H. Bussemaker‡, Luca Boggero§ and Björn Nagel¶
DLR (German Aerospace Center), Institute of System Architectures in Aeronautics, Hamburg, Germany

One of the critical aspects to consider in early design phases of complex systems is the system architecture, as it has major influence on final performance of the product. System Architecture Optimization (SAO) is a technique where the architecting process is formulated as a numerical optimization problem, which allows more extensive and less biased design space exploration. To evaluate the performance of architecture alternatives at the system-level, Multidisciplinary Design Analysis and Optimization (MDAO) is seen as a key enabling technology. Application of MDAO for architecture optimization is currently challenged by the observation that different architectures require different MDAO problem formulations, for example in discipline selection, data connections, and execution order. This work proposes a set of capabilities that MDAO formulation platforms should provide in order to support such dynamic MDAO workflow formulations. Special focus is placed to enable such capabilities in the context of collaborative MDAO, a specialization of MDAO where emphasis is on collaborative and cross-organizational definition, integration, and execution of MDAO workflows using central data formats. The proposed capabilities are implemented in MDAx, an existing MDAO workflow modeling tool developed for formulating collaborative MDAO workflows. The new developments are demonstrated by an open-source launch vehicle architecture design problem, showing that a dynamic MDAO workflow can be generated that modifies its execution behavior automatically for each architecture to be evaluated.

Nomenclature

A_e/A_g	=	Nozzle expansion ratio,	N_s	=	Number of stages
ϵ	=	Cone angle,	R_e	=	Elliptical ratio
l	=	Stage length,	S_{tF}	=	Fuel tank surface
$L_t\_D_t$	=	Length to diameter ratio,	S_{tO}	=	Oxidizer tank surface
$\dot{m}$	=	Mass flow,	T	=	Thrust
M_F	=	Fuel mass,	V_s	=	Stage volume
M_O	=	Oxidizer mass,	V_{tF}	=	Fuel tank volume
M_p	=	Propellant mass,	V_{tO}	=	Oxidizer tank volume
m_{pay}	=	Payload mass			

I. Introduction

When designing complex systems, an important aspect to consider is the system architecture: a description of the different components that the system consists of and how they collaborate to fulfill the system's functions [1]. Choosing the right architecture for the problem at hand is important and significantly affects the final system performance [2]. However, even for a low number of architectural decisions, a combinatorial explosion of alternatives leads to a large design space of architecture candidates [3]. Traditionally due to time and effort constraints, only a small number of architectures is considered and analyzed in detail before settling for a system architecture to be used for

*Researcher, DDP Group, Department of Digital Methods for System Architecting, sparsh.garg@dlr.com
†Researcher, DDP Group, Department of Digital Methods for System Architecting, raul.garciasanchez@dlr.de
‡Researcher, DDP Group, Department of Digital Methods for System Architecting, jasper.bussemaker@dlr.de
§Head of DDP Group, Department of Digital Methods for System Architecting, luca.boggero@dlr.de
¶Institute Director, Institute of System Architectures in Aeronautics, Hamburg, bjoern.nagel@dlr.de

downstream detailed design phases. This may lead to bias and conservatism in selecting the system architecture [4], especially for novel systems where no prior design experience exists [5].

A possible solution to address these problems is System Architecture Optimization (SAO): the application of numerical optimization algorithms to automatically explore the architecture design space [6]. Combined with simulation-based performance evaluation, this can lead to a reduction in design bias and a consideration of more novel architectures. SAO requires the implementation of two interacting components: the architecture generator and the architecture evaluator. The architecture generator uses optimization algorithms to generate architecture candidates from the architecture design space, by mapping architectural decisions to design variables. The architecture evaluator provides quantitative performance feedback to the architecture generator for a given architecture candidate. This feedback should be provided without user interaction, so that the generator can automatically search the design space. System architecting is an integrative discipline that holistically considers many different engineering disciplines and comes up with a compromise between the different interests [7]. The architecture evaluator should reflect this, in that it considers the multidisciplinary nature of the involved analyses from the start. Multidisciplinary Design Analysis and Optimization (MDAO) is a promising technique for implementing such automatic computations, as it enables coupling all relevant engineering disciplines for designing a system, and provides a system-level compromise and overview of system performance [8].

To formulate and execute MDAO problems for application in SAO, MDAO platforms have to deal with multiple challenges, such as mixed-discrete coupling variables and support for optimization algorithms that can solve mixed-discrete, multi-objective, hierarchical, black-box optimization problems [2]. Another less researched challenge is that of supporting evaluation of all possible architecture alternatives in a given SAO problem. For example, selecting different components in an architecture can change the design variables, disciplines, and data connection involved in an MDAO formulation. If the number of architecture alternatives is not excessively high, separate MDAO problems can be manually formulated for each architecture alternative. For problems with larger numbers of possible architectures, however, this approach is not feasible. Instead, the MDAO problem needs to be readjusted automatically for each analyzed architecture alternative, as also observed by Bruggeman et al. [9].

In large-scale MDAO campaigns, disciplines and experts might be organizationally and/or geographically distributed. The resulting data interfacing and exchange issues are partially addressed by collaborative MDAO techniques [10]. Using a Central Data Schema (CDS) ensures that all disciplines "speak the same language" and enables implementation and reuse of disciplines in collaborative MDAO workflows. An example of a CDS is CPACS [11], an open-source XML-based format for exchanging aircraft design data which has been applied successfully in many projects across disciplines and organizations. Setting up workflows based on a CDS is supported by workflow formulation platforms that automatically detect data connections based on input-output definitions of disciplinary tools. One such tool is MDAx, the MDAO Workflow Design Accelerator, developed within the DLR [12, 13].

This paper presents recent efforts to enable the use of collaborative MDAO techniques for SAO. First, the influences that changing architecture alternatives have on the formulation of an MDAO workflow are investigated in Section II. Section III presents mechanisms for supporting the influences in a collaborative MDAO platform, and the implementation of these mechanisms in the MDAx tool. The implementation is demonstrated in Section IV with an open-source launch vehicle architecture design problem. Section V concludes the paper.

II. Methodology

Addressing different system architectures requires modifications to the MDAO problem formulation and execution which can be accommodated into four mainstream categories, termed *architectural influences*. A sample of MDAO problems available in literature were chosen to identify the architectural influences. Each of these problems analyses a given fixed system architecture which was later modified to evaluate and understand how the MDAO problem would be affected by these architecture modifications.

This section will proceed with the introduction of the architectural influences followed by the use of the supersonic business jet (SSBJ) problem [14] from literature, as an example to show the methodology involved in identifying these architectural influences. Finally, the discovered architectural influences will be verified using more SAO problems available in literature.

A. Architectural Influences

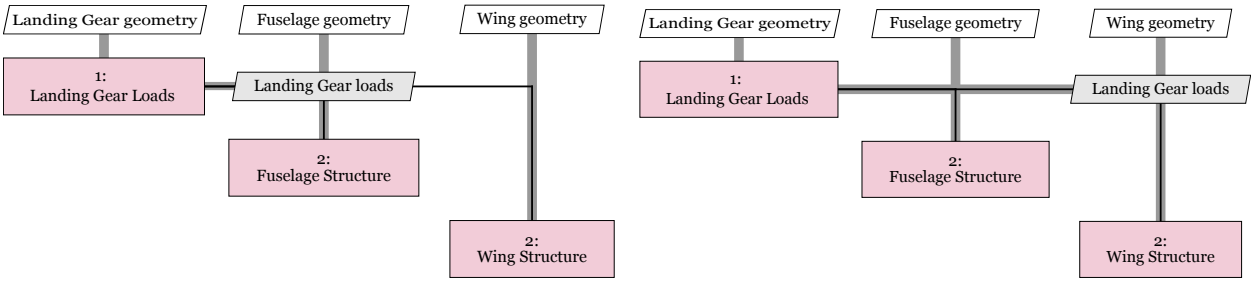
Bussemaker et. el. [15] laid down the ground work for identifying the different architectural influences that could exist in a system architecture optimization problem and presented some mitigation strategies to address these influences during the formulation process of the corresponding MDAO problem. Using it as a starting point, the aforementioned

methodology was adopted to converge on a total of four different types of architectural influences:

1) **Conditional variables:** Each system is made of different components, and each component is defined by different variables. As a consequence, the variables existing in the MDAO problem will change depending on the architecture being analysed. The variables whose existence depends on an architectural decision are called conditional variables.
2) **Data connection:** In some occasions, the connections between design disciplines might change because of an architectural decision. This happens when there is a modification in the coupling of a certain variable leading to a change in terms of which disciplines the variable is connected to. Therefore, in MDAO problems used for system architecture optimization, rerouting of variables can happen, leading to dynamic connections between disciplines.
3) **Discipline repetition:** It is common in MDAO that a discipline has to be repeated a fixed number of times. However, in system architecture optimization, sometimes the discipline multiplicity depends on an architectural decision, and therefore has to be readjusted automatically.
4) **Discipline activation:** As a consequence of an architectural decision, there are multiple cases where some of the disciplines included in the MDAO problem are no longer required for that specific architecture. This is usually the case when two or more technologies are available to perform a certain function.

The existence of **conditional variables** in the MDAO problem leads to the modification of inputs/outputs of the different design disciplines, and therefore have to be readjusted for each system architecture automatically. In the SSBJ problem, a possible architectural decision leading to this modification could be the inclusion of a winglet. If a winglet is included in the architecture, new variables will appear in the MDAO problem, such as the winglet length or the cant angle. To manage complex problems, it is also necessary to exchange more complex data structures, such as arrays, between the different disciplines. MDAO platforms should also be capable of dealing with undefined parts of these structures, for example variable length arrays.

A change in **data connections** is seen when an architectural decision leads to a change in how the disciplines among which a variable is coupled. An example in the SSBJ problem would be the aircraft landing gear placement. A change in the attachment location from wing to fuselage might influence the load cases used for sizing the wing and fuselage structures. In this case, the connections existing between the different design disciplines could be modified as a consequence of the architectural decision, as shown in Figure 1.



(a) Landing gear attached to the fuselage **(b) Landing gear attached to the wing**

Fig. 1 Example of a data connection influence: if the landing gear is attached to the fuselage, the "Landing Gear Loads" discipline will be connected to the "Fuselage Structure" discipline; if the landing gear is attached to the wing, this connection is removed and replaced by a connection between the loads and the "Wing Structure" discipline.

Discipline repetition usually happens when the number of instances of a component, in the system architecture, is an architectural decision itself. For example, consider a modified propulsion discipline which takes as input the properties of an engine. If each engine had different properties and the number of engines was an architectural decision, the number of times the discipline would have to be **repeated** would vary depending on the selected number of engines. This repetition will involve a variation of inputs/outputs for each discipline iteration. It also leads to the inclusion of new connections in the MDAO problem.

Finally, the relevance of a discipline to a given architecture determines the **discipline activation** or deactivation status. An example of discipline activation would be when an architectural decision exists on the choice of the fuselage

material. Depending on the decision taken (e.g. metallic or composites) the discipline used for the structural analyses might have to be exchanged (Figure 2). When multiple disciplines can be added or excluded, connections always change, and even the coupling variables could end up being different.

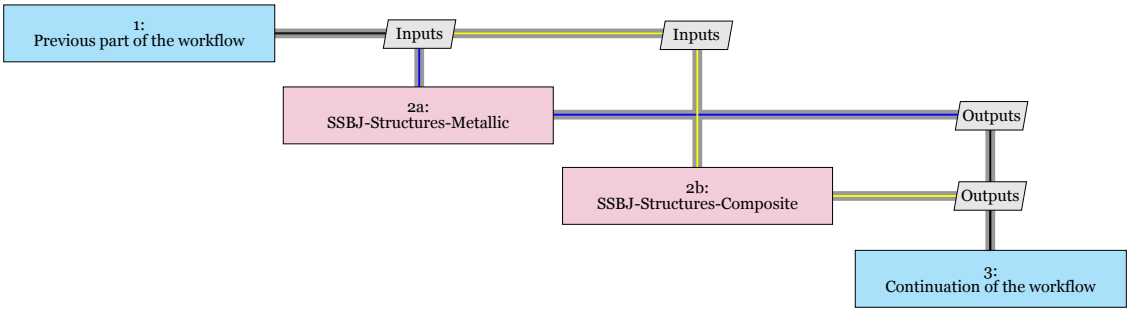


Fig. 2 Example of discipline activation: if a metallic material is chosen, the metallic structures discipline is activated and the necessary connections are made (dark blue path). In the case of composites, another discipline is used to perform structural calculations and different connections are needed (yellow path).

B. Verification of Architectural Influences

In this section, two pre-existing system architecture problems from literature are analyzed to verify that the architectural influences that have been identified, indeed affect the implementation of MDAO in the system architecting process. First, [16] presents a benchmark problem based on the design of an aircraft jet engine. Different components were included in the architecture (fan, gearbox,...). Each of these components is represented by a different discipline, and are therefore an example of **discipline activation**. The number of instances of some of these components (such as compressor or turbine stages) is also an architectural decision, leading to the **discipline repetition** influence. At each compressor stage execution iteration there was an option to bleed cooling air to the turbine. This would be an example of both, **conditional variables** and **data connection**, with the respective architectural decisions being whether to bleed air or not and if yes, then to which turbine is this air being passed to.

Architectural influences can also be found in the example shown in [17], which aims to design an hybrid-electric aircraft propulsion system using system architecture optimization. There is a part within the MDAO problem where the propulsion system thrust is calculated using the mechanical power generated by the motor or the turboshaft, as an input (among others). A simplified version of the same is shown in Figure 3. In this problem, all the components of the propulsion system that might contribute to the generation of mechanical power have their own design discipline. The existence of these disciplines depends on the choice of components made in each separate system architecture, and therefore are an example of **discipline activation**. The number of times these disciplines will be executed depends on the number of instances of their correspondent architectural component, which in-turn is also an example of **discipline repetition**. Finally, there are two instances where the motor and the turboshaft co-exist in the system architecture. In the first case, the turboshaft and the motor contribute directly to the generation of mechanical power to be used for thrust production (parallel configuration). In the second case, the power generated by the turboshaft is used to feed the motor (series configuration). Depending on the case, the power generated by the turboshaft will be an input for the Motor or the Performance discipline, and is therefore and example of **data connection**.

III. Implementation of Architectural Influences in a Collaborative MDAO Platform

In the previous section four architectural influences have been introduced. This section will first introduce high-level strategies for dealing with architectural influences in MDAO workflows. This is followed by the description of the implementation of the chosen strategy in MDAx, the MDAO platform modified in this work [12, 13].

A. Strategies for Applying MDAO for SAO

MDAO problems are formulated in several steps [18]: a tool repository is defined, the MDAO problem is defined by selecting tools from the repository and establishing data connections, and the solution strategy is defined by adding non-linear solvers and/or applying an MDAO architecture [19]. To benefit from MDAO for SAO problems, another

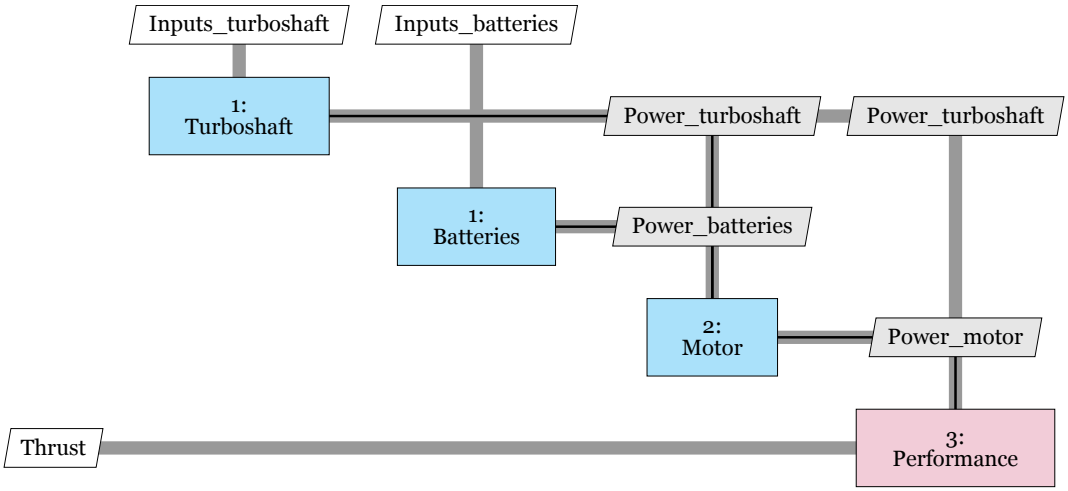


Fig. 3 Simplified process for the calculation of the aircraft propulsion thrust considering different sources of mechanical power. Architectural choices will govern whether the mechanical power to generate thrust will be provided by the turboshaft, the motor or both.

step needs to added to include the architectural influences. These architectural influences are included by specifying the "influence logic": the logic (if statements, loops, etc.) which defines how the MDAO workflow should modify its behavior for evaluating different architecture instances. After formulation, the MDAO problem is deployed in some execution environment, where it can be invoked as part of the SAO loop for a given architecture instance generated by an architecture generator [2]. The results of the MDAO problem are communicated back to the architecture generator in the form of performance metrics, after which the next iteration of the architecture generator starts. It should be noted that the formulation phase and the optimization loop form two coupled yet separate steps of the whole process, as also depicted in Figure 4. The formulation and deployment tasks are implemented by a "formulator" which, depending on where it is applied in the process, can require user interaction or not. The following high-level strategies for using MDAO for architecture evaluation can be distinguished:

1) **Single static** (Figure 4a): a single static MDAO problem is defined, influence logic is implemented at the discipline-level, and the formulator is applied in the MDAO formulation phase. No change in MDAO tooling is required to support this strategy. This is a more restricted way of dealing with architectures which is only possible if all architecture influences can be handled at the discipline-level. Examples of this strategy include the design of a business jet family [20], and the comparison of various electrification levels of aircraft on-board systems [21].
2) **Multi static** (Figure 4b): multiple static MDAO problems are defined, each of which solves a predefined set of architecture instances. These sets are either defined by clustering architecture instances, or manually based on architecture choice values. This is a more conventional way of dealing with multiple architectures. This strategy is either executed using a router that determines within the SAO loop which static MDAO problem should be used for the given architecture instance, or by manually running parallel SAO problems for each of the architecture sets. Examples of this strategy include an architecture clustering approach based on active design variables [22], and the execution of two independent static MDAO problems for different material selections [23].
3) **Single dynamic** (Figure 4c): a single dynamic MDAO problem is defined, which implements influence logic directly in the workflow such that its behavior is automatically modified for a given architecture instance. The effectively established data connections and tool execution sequence is represented by the effective MDAO problem, a hypothetical problem representing the fact that the dynamic MDAO problem changes its behavior for different architecture instances. This is a more novel approach to SAO which holds promise for improving efficiency of the process. Examples of this strategy include an approach for dynamically switching between sub-workflows based on an architectural choices by Bruggeman et al. [23], and a dynamically changing sub-workflow based on the number of instances of some architecture element by Sonneveld et al. [24].
4) **On-demand** (Figure 4d): automatically formulate a static MDAO problem within the SAO loop based on the architecture instance. Examples of this strategy are a hybrid-electric propulsion system problem [17] and a jet

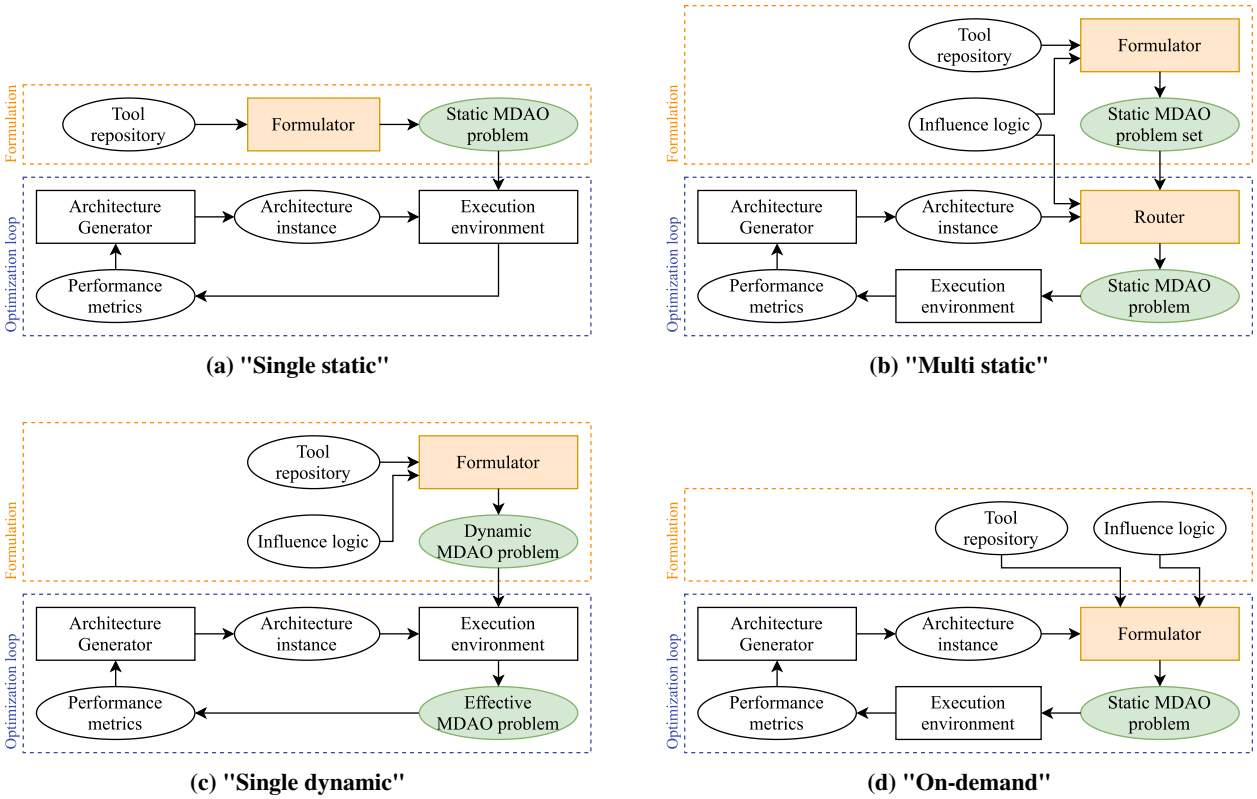


Fig. 4 High-level strategies for integrating architecture influences in MDAO workflows. The formulation phase (orange dashes) is executed before running the optimization loop (blue dashes). The optimization loop is executed without user interaction.

engine architecture problem [16].

The single static strategy requires a level of control over disciplinary tool development which might not be possible for collaborative MDAO [25, 26], and might simply not be flexible enough for complex SAO problems. The multi static strategy requires managing multiple workflows simultaneously, which will not be feasible above a relatively low number of workflows. The on-demand strategy requires an automated formulator (or "MDAO bot" [18]) that also automatically deploys the workflow. In collaborative MDAO, however, the formulation phase is usually separate from the execution phase [26], because both phases involve specialized software requiring one or more manual steps. This currently prevents the application of on-demand MDAO problem generation for collaborative MDAO, which is why the single dynamic strategy is chosen and implemented for supporting architectural influences in this work.

B. Supporting Architectural Influences in MDAx

MDAx is an MDAO formulation platform developed by the DLR that allows to model, inspect and explore workflow components and their relationships [12, 13]. To execute the MDAO problem, an external integration platform called Remote Component Environment (RCE) [27], which is developed by the DLR, can be used. Using these two platforms the MDAO problem can be formulated and executed according to collaborative MDAO principles.

This section presents how MDAx was extended to support the previously presented architectural influences based on the "single dynamic" strategy (see Figure 4c): the MDAO workflow model contains all logic for dynamically modifying its execution behavior, which is materialized in the exported RCE workflow where the SAO problem is executed.

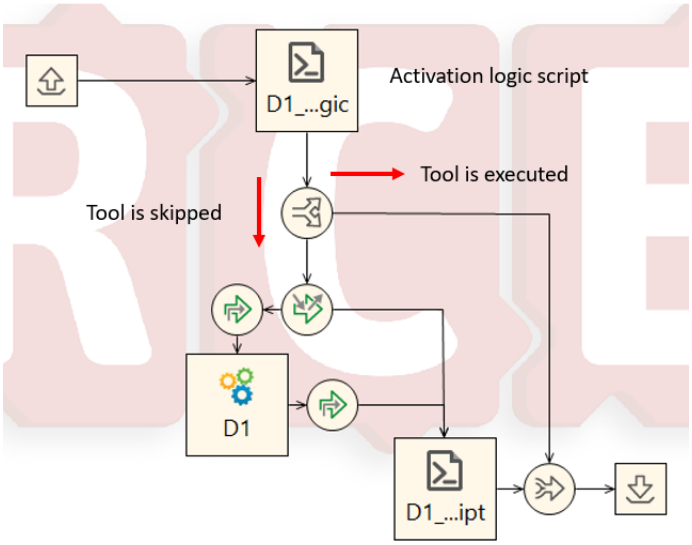


Fig. 5 Example of a tool with activation logic in RCE. First, the workflow XML enters into the activation logic script, which checks if the activation logic assertion attached to the tool is true or false. If is true, the switch passes the workflow XML to the tool and the tool is executed. If not, the workflow XML is passed directly to the next discipline.

1. Discipline Activation

Disciplines whose inclusion in the MDAO problem is contingent upon an architectural decision have conditions known as *activation assertions*, which determine their execution within the MDAO problem. If a discipline is associated with an activation assertion in MDAx, its RCE export incorporates an additional script executed prior to the discipline. This script evaluates the condition, and if it is satisfied, the discipline is executed; otherwise, the discipline is bypassed in the workflow execution (Figure 5).

In RCE, all necessary information for executing the MDAO problem, including the relevant variables, is stored in an XML file known as the workflow XML file. Architectural decisions impacting discipline inclusion can be linked to specific nodes or information within these nodes in the workflow XML file. The condition determining the inclusion of a tool may involve the existence of a node (indicating the presence of a component in the system architecture), the repetition of a node (indicating the number of component instances), or the information contained within a node (usually related to the properties of a component in the system architecture). Assertions for these conditions have been implemented as composable Python classes. The activation logic script, illustrated in Figure 5, utilizes these classes to evaluate the assertion, subsequently routing the input file either through the tool (if the discipline is active) or bypassing the tool (if the discipline is skipped)

2. Discipline Repetition

The implementation of discipline repetition in an MDAO platform can be approached in two ways. The first method involves creating a distinct instance for each repetition of the discipline, known as a parallel configuration. The second approach is to utilize a single discipline instance, configuring the workflow to repeatedly "enter" the same discipline, known as a series configuration. Figure 6 illustrates these two approaches using XDSM notation.

The series configuration has been implemented in MDAx and RCE to simplify workflow management in line with the principles of collaborative MDAO. To understand this implementation, it is important to recognize that multiple instances of a component (determining the number of repetitions) are represented in the workflow XML file as repeated nodes corresponding to the component and its properties.

Each time the discipline is executed, a new block called "Global to Local" selects the appropriate inputs for that iteration, converting data from a global (workflow XML file) to a local (tool) representation. The tool then executes normally, and its output is converted back by the "Local to Global" block, ensuring that the outputs are correctly written into the workflow XML file. The local versions of the input and output XML files are specific not only to the tool but

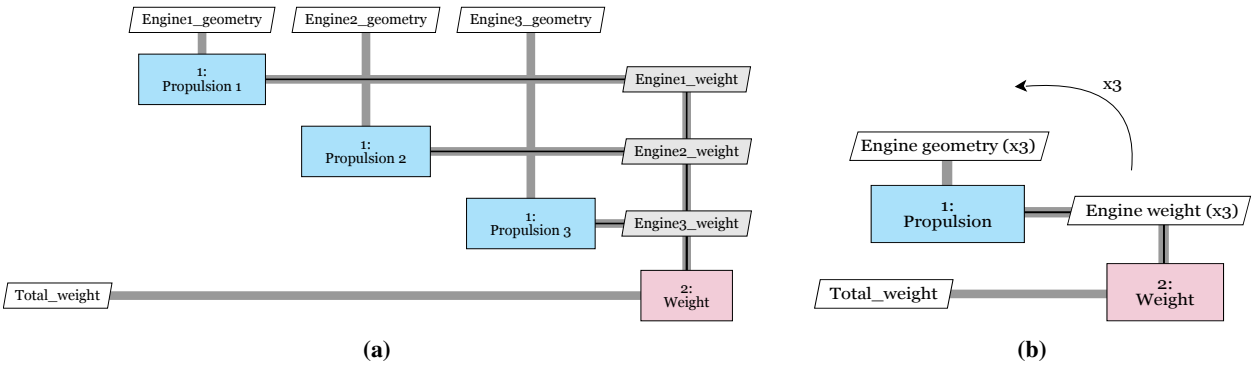


Fig. 6 Example of the two possible approaches to implement discipline repetition: (a). A different instance is used every time the discipline has to be executed (parallel configuration). (b). An unique instance is created an re-executed multiple times (series configuration).

also to each iteration, allowing variation in inputs and outputs across different executions of the same tool.

Finally, an "Iterator" block tracks the number of iterations and detects the loop's end once the tool has been executed the specified number of times as defined in the workflow (Figure 7)

3. Data Connection

The implementation of data connection in an MDAO platform depend on how connections between disciplines are generated. If a decentralized approach is used [18], then it is the connection itself which has to be created or suppressed. A central data schema (CDS) allows these connections to be modified by changing input-output specifications of the involved disciplines.

In MDAx, the MDAO problem formulation takes into account the inputs and outputs of disciplines and their corresponding couplings. Consequently, all the possible connections between disciplines exist in the formulation itself. Now, for a connection to be deactivated due to an architectural decision, the coupling variable itself can be deactivated in the input definition of the involved disciplines. The variable deactivation conditions are implemented in a similar manner as discipline activation. Thus, for each input variable that may be deactivated, an assertion condition can be specified to determine when deactivation should occur. These assertions are implemented in the previously introduced "Global to Local" block. If the assertion for a given input variable is true, the corresponding XML node is filtered out of the input file passed to the tool.

4. Conditional Variables

In MDAx, nodes corresponding to potential input variables are searched in the workflow XML file before executing the discipline. If a node is found, it is added to the tool input file; if not, it is ignored. This procedure enables MDAx to handle conditional variables effectively when they are present as inputs. In the case of conditional variables found as outputs, an analogous methodology to the one used for data connection is implemented. The condition determining its deactivation from the MDAO problem is stated in the configuration file. Following this, a Local to Global block checks the assertion after the tool is executed. If the assertion is satisfied, then the variable is deleted form the outputs and is not merged into the workflow XML file, deleting it from the MDAO problem.

IV. Demonstration Case: Multistage Rocket Problem

This section demonstrates the previously presented methodology for supporting architectural influences in a collaborative MDAO framework for the purpose of architecture optimization. A multistage launch vehicle architecture optimization problem is introduced and solved. It is a realistic engineering design problem, allowing to show the potential of system architecture optimization in combination with dynamic collaborative MDAO.

The section begins by introducing the design problem, followed by the description of the different involved disciplines. Finally, some example architectures are executed to verify the implementation of the methodology and MDAO problem in MDAx, and the results of the optimization problem are analysed.

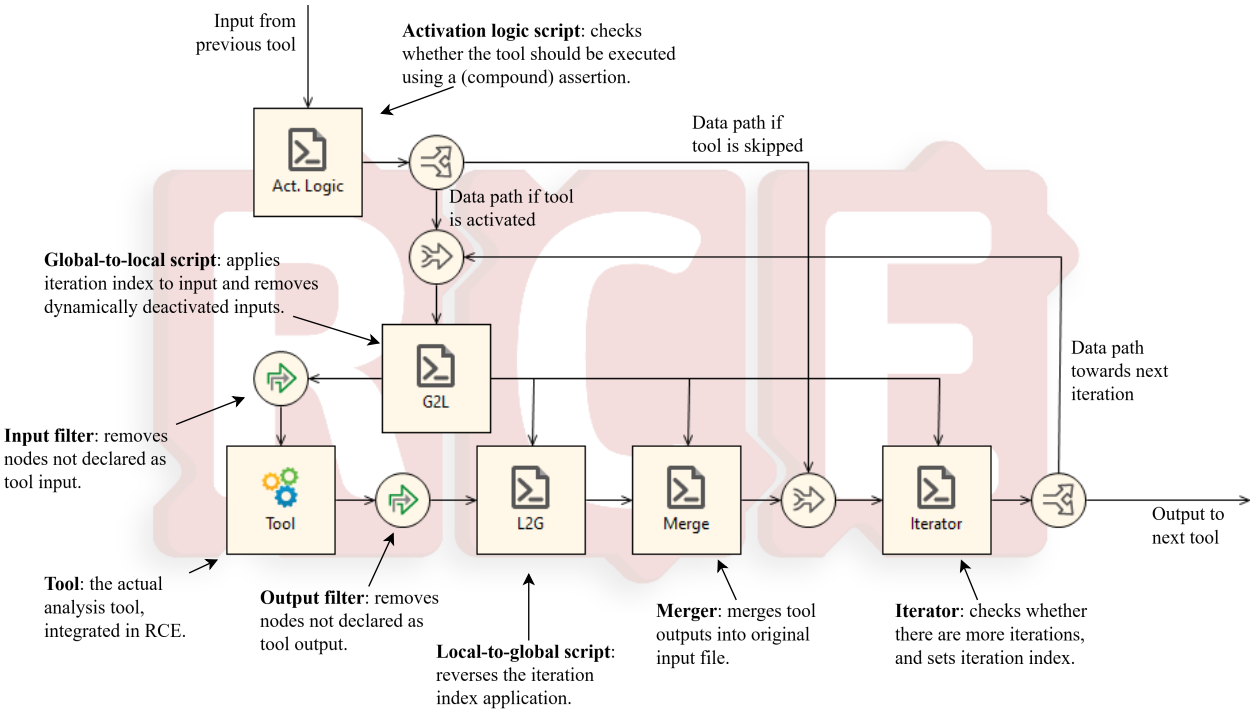


Fig. 7 An overview of dynamic MDAO implementation in RCE, as exported by MDAx. The activation logic script determines whether the tool is skipped or not, see also Figure 5. The Global to Local (G2L) component is in charge of multiple processes, such as determining the number of iterations, keeping track on the iteration the execution is at each moment or selecting the inputs/outputs for each iteration. The Local to Global (L2G) prepares the tool outputs to be merged in the correct place in the workflow XML. Finally, the iterator block determines if the repetition of the discipline has ended or not.

A. Benchmark Problem Description

A multistage rocket is a launch vehicle designed to carry a manned or unmanned payload from Earth's surface to outer space. Therefore, synonymous to an aerospace vehicle design optimization problem, the primary objective is to maximize the mass of payload that can be lifted to a circular orbit at 400 km of altitude, while minimizing the cost. Since payload mass and cost are the main considerations influencing the design rocket, these quantities will be the key drivers of the optimization problem. To achieve the best possible design, the optimizer will have control over various architectural decisions: the number of stages (N_s), the head shape (H_s), the engine type used for each stage, and the number of engines for each stage.

Seven different design disciplines are considered in the optimization process, including trajectory analysis and propulsion sizing. Some constraints are also added to ensure the feasibility of the design. The remainder of this section briefly introduces the included disciplines and analysis blocks. For a more detailed description, the reader is referred to [28]. The problem implementation is available as open-source Python code*.

The first discipline in the workflow execution is **propulsion**. The propulsion discipline varies with the type of propellant. It allows a choice between three possible engines, taken from real-life designs. They receive an array containing the engine type and the number of engines of a rocket stage as input. The outputs include the thrust generated by a rocket stage and some additions quantities of interest in case of liquid propulsion engines, like mass flow or the nozzle expansion ratio.

The next discipline is the **geometry calculator**. This discipline provides some geometric parameters as output, such as the surface and volume for fuel and oxidizer tanks and the rocket head. It receives various inputs such as the engines used for the stage and the parameters associated to each possible head shape (semispherical, conical or elliptical).

Afterward, the **propellant mass** discipline is executed. This discipline calculates the propellant mass (if it solid) or the fuel/oxidizer mass (if it is liquid) for each rocket stage. The contribution of other rocket components to the total

*`https://github.com/raul7gs/Space_launcher_benchmark_problem`

rocket mass is calculated in the **structural mass** discipline. In the case of solid propulsion, the component mass is calculated for the engine casing. While, for liquid propulsion, the mass of tanks, pumps and the insulation are calculated. In both cases the rocket head mass is considered as well.

To calculate the maximum payload mass that the rocket can lift to the desired orbit, the **trajectory** discipline is used. This discipline first calculates the vehicle drag coefficient. Assuming that there is no payload mass, the equations of motion (Equation 1) are solved numerically to determine if the rocket has enough power to arrive at the orbit (Figure 8). If the power is more than required, the payload mass is increased to the limit while ensuring that the orbit altitude can be reached, thereby determining the maximum payload mass that can be carried to that orbit.

$$\frac{\partial^2 x}{\partial t^2} = \frac{Tcos(\alpha) - 0.5\rho S V^2 C_D}{(m_0 + m_{pay}) - \dot{m} * t} - gsin(\gamma) \tag{1}$$

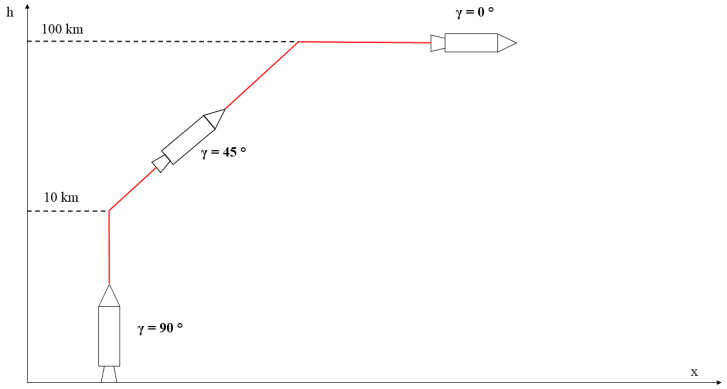


Fig. 8 The trajectory is assumed to consist of three phases: First a vertical take off, then a constant 45 degrees climb, finally the rocket increases its speed to achieve the desired orbit (in this case shown as 100km, however this is a problem input and can be changed).

The second objective is calculated in a discipline called **cost**. Three main components are assumed to contribute to the total rocket cost: the engines, the propellant and the rocket head structure. The engines costs are calculated using the TRANSCOST model [29]. Propellant and structural costs are estimated using current prices [28].

Finally, two constraints are considered to determine the design feasibility. First contraint ensures that the rocket can withstand the loads during the mission. This is done by determining the maximum dynamic pressure in the trajectory (maxQ) and ensuring that the value is lower than the rocket structural limit. The second constraint ensures that the payload mass calculated can fit inside the rocket head shape.

The problem XDSM can be found in Appendix VI. Conditional variables are indicated with an asterisk next to them. Variables involved in data connection are expressed in bold and italic, including between parenthesis the architectural decision causing them. Discipline repetition is indicated in the top right of the discipline block, again with the associated architectural decision. Lastly, for the cases with activation logic, the assertion condition is expressed in a third additional row in the discipline block.

B. Verification

To verify the successful implementation of the process, two distinct system architectures were executed within the RCE workflow. The first architecture features a singular-stage liquid-fueled rocket with a spherical head, with additional details such as stage dimensions, vehicle dynamic pressure thresholds, and payload densities, integrated into the input workflow XML file (see Figure 9). The second architecture comprises of a three-stage rocket, with all stages employing liquid propulsion except the first, accompanied by a modification in nose shape to an elliptical configuration. While the first architecture demonstrates the implementation of discipline activation and repetition, the second one features all four influences.

```xml
<Rocket>
  <Stage UID="stage_1">
    <Engines>
      <Liquid>
        <VULCAIN/>
      </Liquid>
    </Engines>
    <Geometry>
      <Length>26.47</Length>
    </Geometry>
  </Stage>
  <Geometry>
    <Head_shape>Sphere</Head_shape>
    <L_D>10.83</L_D>
  </Geometry>
  <Structure>
    <Max_q>50000.0</Max_q>
  </Structure>
  <Payload>
    <Density>2810.0</Density>
  </Payload>
</Rocket>
```

Fig. 9 Example of a workflow XML input file. In this case, a single stage rocket.

A notable distinction in the execution processes of these architectures pertains to the frequency of tool invocation, as depicted in Table 1. For instance, **discipline activation** is evident in the single-stage rocket scenario, where the absence of solid propulsion obviates its inclusion during workflow execution.

Table 1 Number of executions of each tool for the different system architectures

Architecture	Propulsion (L)	Propulsion (S)	Geometry	Structure	Mp	Trajectory	Cost	Str. Constr.	Payload
1 stage	1	0	1	1	1	1	1	1	1
3 stages	2	1	1	3	1	1	1	1	1

Moreover, Table 1 also shows **discipline repetition** across system architectures. This is particularly evident in disciplines such as structural mass, where the number of executions aligns with the number of stages. Notably, the iteration-specific data handling automatically adjusts input and output data locations, exemplified in Figure 10.

```xml
<Rocket>
  <Stage UID="stage_2">
    <Geometry>
      <Length>30.53</Length>
      <Stage_volume>427.57</Stage_volume>
      <Oxidizer_tank_volume>141.44</
          Oxidizer_tank_volume>
      <Fuel_tank_volume>286.13</
          Fuel_tank_volume>
      <Oxidizer_tank_surface>161.99</
          Oxidizer_tank_surface>
      <Fuel_tank_surface>299.04</
          Fuel_tank_surface>
    </Geometry>
  </Stage>
</Rocket>
```

(a) Geometric data second stage

```xml
<Rocket>
  <Stage UID="stage_3">
    <Geometry>
      <Length>10.82</Length>
      <Stage_volume>151.53</Stage_volume>
      <Oxidizer_tank_volume>50.12</
          Oxidizer_tank_volume>
      <Fuel_tank_volume>101.40</
          Fuel_tank_volume>
      <Oxidizer_tank_surface>75.49</
          Oxidizer_tank_surface>
      <Fuel_tank_surface>124.06</
          Fuel_tank_surface>
    </Geometry>
  </Stage>
</Rocket>
```

(b) Geometric data third stage

Fig. 10 Three stage system architecture geometric data. The output differs for each iteration (stage), as inputs were taken from different parts of the workflow XML file.

Regarding **data connection** influence, consider the mass structure discipline as an illustrative example. Here, the necessity of propellant mass input arises solely in the context of solid engine deployment. This adjustment is reflected in Figure 11, wherein propellant mass data is selectively omitted from mass structure inputs when liquid engine configurations are employed.

```xml
<Rocket>
  <Stage UID="stage_{INDEX}">
    <Engines>
      <Solid></Solid>
    </Engines>
    <Geometry></Geometry>
    <Propellant_Mass></Propellant_Mass>
  </Stage>
  <Geometry></Geometry>
</Rocket>
```

(a) Solid fuelled stage

```xml
<Rocket>
  <Stage UID="stage_{INDEX}">
    <Engines>
      <Liquid></Liquid>
    </Engines>
    <Geometry></Geometry>
  </Stage>
  <Geometry></Geometry>
</Rocket>
```

(b) Liquid fuelled stage

Fig. 11 Simplified intermediate input files for the mass structure discipline. When a liquid engine is used, the propellant mass information is deleted.

Finally, the type of fuel utilized in each stage influences the inputs for various disciplines. For instance, within the geometry calculator discipline, the provided geometric data varies based on the head shape of the system architecture, such as sphere radius or ellipse length-to-diameter ratio. This capability underscores the platform's proficiency in handling **conditional variables**, adapting inputs dynamically in response to specific architectural configurations. The resultant XDSM for the demonstration case can be seem in Figure 16. The figures also shows some prospective additions to the XDSM representation in order to incorporate the architecture influences.

C. Results and Discussion

To solve the problem, the genetic algorithm NSGA-II has been used [30]. By a general rule of thumb, a population size of 10 per design variable was taken, leading to a total population of 150. Also, 30 generations were decided to be

analysed to which ensures good performance of a genetic algorithm. This leads to a total of 4500 evaluated design point, out of which 763 were found to be feasible. The results obtained can be observed in Figure 12.

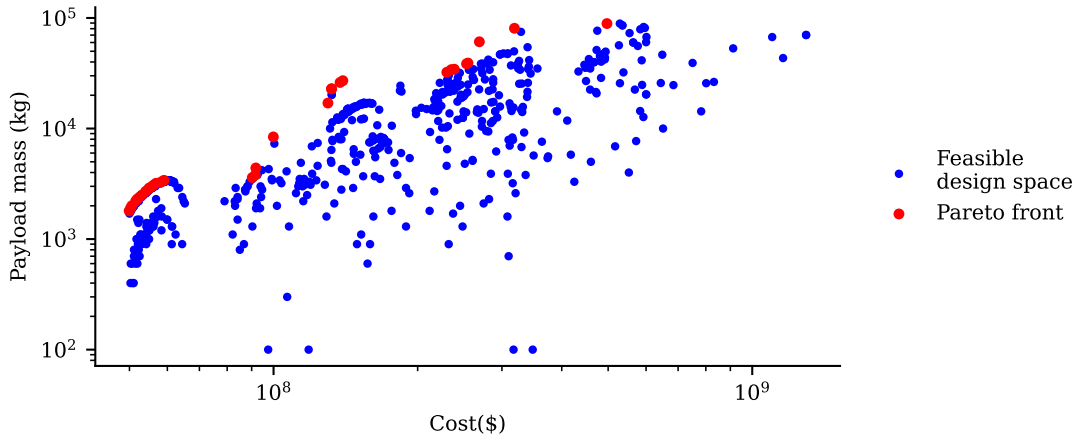


Fig. 12 Feasible design space (blue) and Pareto front (red). The objectives are the cost (x-axis; to be minimized) and the payload mass (y-axis; to be maximized).

The Pareto front consists of 32 of the feasible design points. Different architectures can be found in the Pareto front. For example, taking a look at the number of stages (Figure 13), it can be observed that a lower number of stages is chosen when lower payload masses are lifted, as it is cheaper. However, when the payload mass increases, it becomes necessary to also increase the number of stages required to achieve that orbit, at the compromise of a higher cost.

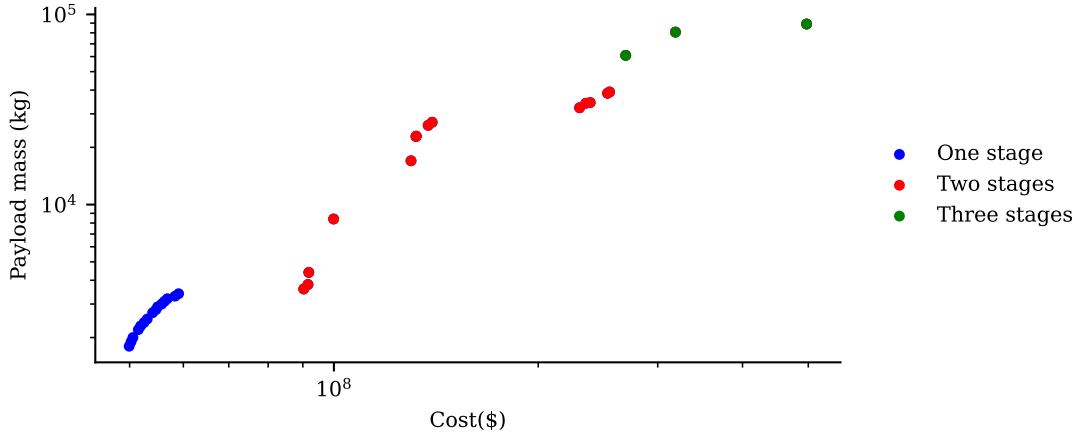


Fig. 13 Pareto front points based on different number of stages. When the number of stages increases, the rocket payload mass capability increases, but the cost increases too.

Another important architectural choice is the rocket head shape. It should be noted, that for the current research, the decision for the head shape is primarily governed by the drag coefficient values of the different shapes. The coefficients for the three shapes under question are governed by the equations available in open literature [31, 32]. For further details into the chosen method for the geometry calculation discipline and the performance of the three shapes under consideration, the reader is referred to [28]. It can be observed that semispherical heads are chosen for the lower payload (Figure 13). This is because, among the possible options for head shape, a semisphere maximizes the amount of volume available for payload given a certain surface (and therefore cost). However, it is the worst aerodynamically, explaining why for bigger payload masses and bigger rockets, where aerodynamics is more important, it starts to be substituted by

conical heads and then at the end by elliptical, which are the one with the lowest drag coefficient.

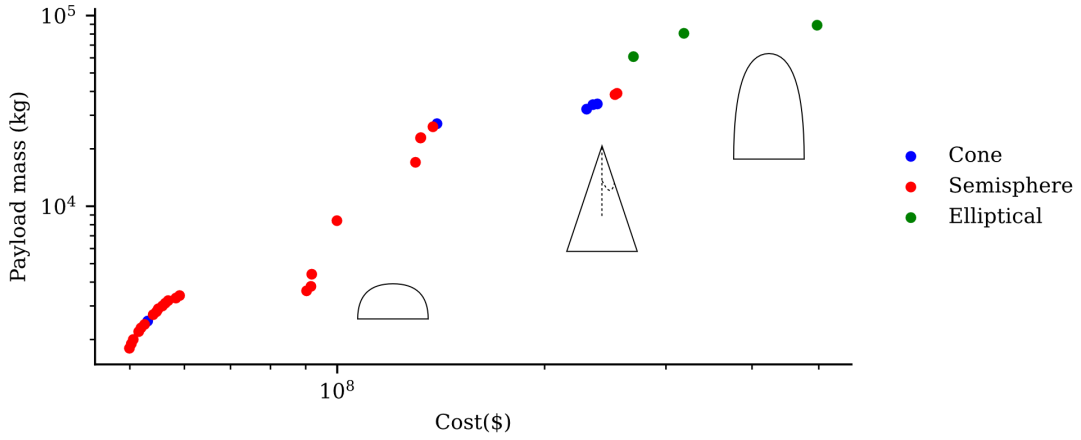


Fig. 14 Different architectures found in the Pareto front according to the head shape. Progression of head cone from semishpere to conical as payload increases

Finally, for the choice of the type of propellant, two clusters are observed which are based on the chosen propellant for the first stage of the rocket. For smaller rockets with lower payload, liquid propellant is used due to its lower density and thus, a lower mass. However, since more power is needed with the increase in payload mass, solid propellant has to be chosen which consequently, results in increased total mass and cost of the rocket (Figure 15). Finally, for the second and third stages of the rocket, a liquid propellant is chosen everytime since less power is needed. The general trends of the results achieved are consistent with real life examples of rockets used in Ariane 5, Space Shuttle and Saturn V. For further details and drawing more parallels with real life examples, the reader is again referred to [28].

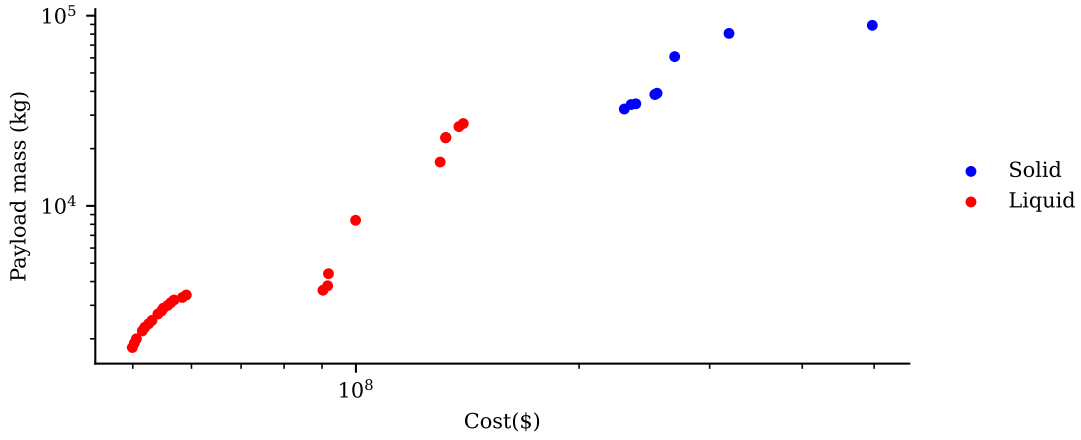


Fig. 15 Variation in the choice of propellant type (solid or liquid) for the first stage of the rocket in the Pareto front. Two distinct Pareto fronts can be observed with liquid propellant being favoured for rockets with smaller payload capacity and solid propellant for higher payload carrying larger rockets.

V. Conclusion and Outlook

System Architecture Optimization (SAO) can be used to objectively look for the best possible architectures in a large combinatorial design space. MDAO can be used to evaluate the different architectures, so that the different coupled

design disciplines are considered. To implement SAO in industry, it is necessary that the MDAO platform used is adapted to collaborative MDAO and can automatically readjust the MDAO problem for each system architecture.

Various strategies to implement MDAO in SAO problems are laid down. The strategy which best suits the task of dynamically modifying the MDAO formulation, for the different architectures involved in a SAO problem, is selected and further implemented in a chosen MDAO platform. To do so, a set of architectural influences are identified which address the various modifications that are required in the dynamic modification MDAO problem formulation from one architecture to the other. These are as follows:

1) **Conditional variables:** Variables in the MDAO problem change based on the architecture being analyzed. The existence of conditional variables in the MDAO problem leads to the modification of inputs/outputs of the different design disciplines, and therefore have to be readjusted for each system architecture automatically
2) **Data connection:** The connections between design disciplines may change due to architectural decisions. Thus, to address dynamic coupling and rerouting of variables between disciplines in MDAO problems, MDAO platform should be capable of of handling dynamic data connection.
3) **Discipline repetition:** In system architecture optimization, the number of times a discipline is repeated can depend on architectural decisions, requiring automatic readjustment of the same at the MDAO problem formulation level.
4) **Discipline activation:** Some disciplines may become unnecessary due to architectural decisions, especially when multiple technologies can perform the same function, leading to selective discipline activation. Thus, a MDAO platform should be able allow for dynamic and selective activation of disciplines.

These architectural influences are the ground work for any expert wishing to achieve a similar automation in MDAO problem formulation phase of a design study which mandates dynamically changing the problem formulation. Certain implementation strategies are also proposed, implemented and validated for each architectural influence which can be extended to any MDAO problem formulation platform.

The in-house MDAO problem formulation software MDAx is extended to implement the dynamic MDAO problem formulation strategy. This was achieved by implementing the distinct architectural influences that guide the modifications in the MDAO problem formulation, encompassing various system architectures. A rocket design problem was developed and solved to verify that the implementation was done correctly, and to show the potential of the methodology when applied to realistic design cases.

This research is already a step forward in the implementation of MDAO inside the SAO process, allowing to reduce the gap impeding the implementation of SAO in the industry. However, there is still some development to be done to further reduce this gap, for example, using the feedback from the MDAO analysis to govern the architectural decisions and thus completing the system architecting-MDAO loop. A major next task is to implement the user-interface for the new features in the MDAO platform, enhancing its applicability and making it easier to use for engineers without requiring knowledge of the backend code. Another step would be to improve the pre- and post-processing of disciplinary blocks, which is required for execution of the dynamic MDAO features, resulting in reduced complexity of the executable workflow. Also, it can be foreseen that complex MDAO workflows and nested workflows might require the application of the features to groups of tools rather than individual tools.

Influence logic governs the dynamic behavior of the MDAO workflow and is currently defined by the user directly. Logically, however, it depends on the behavior of the SAO design space, for example in terms of which components/technologies can be selected to fulfill a given function. Future work will investigate how to better integrate the definition SAO design problems and associated dynamic MDAO workflows, for example by automatically deriving influence logic from architectural choices, and by developing a process for side-by-side development of the SAO design problem and dynamic MDAO workflow.

The results of this research have potential to be applied in other novel areas of research like multi-fidelity MDAO, efficient global MDAO using surrogate models, and a combination of the aforementioned in combination with SAO. Furthermore, in the interest for maintaining collaboration and widening the application of this research, one possible extension could be to include exports of the dynamically executed MDAO workflows for other MDAO platforms and process integration and design optimization platforms used in research and industry. Finally, work is being done to develop a standard for the visual representation of dynamic MDAO problems by a new variant of the XDSM notation, for which first steps were taken in the current research but future work is still needed.

VI. Appendix

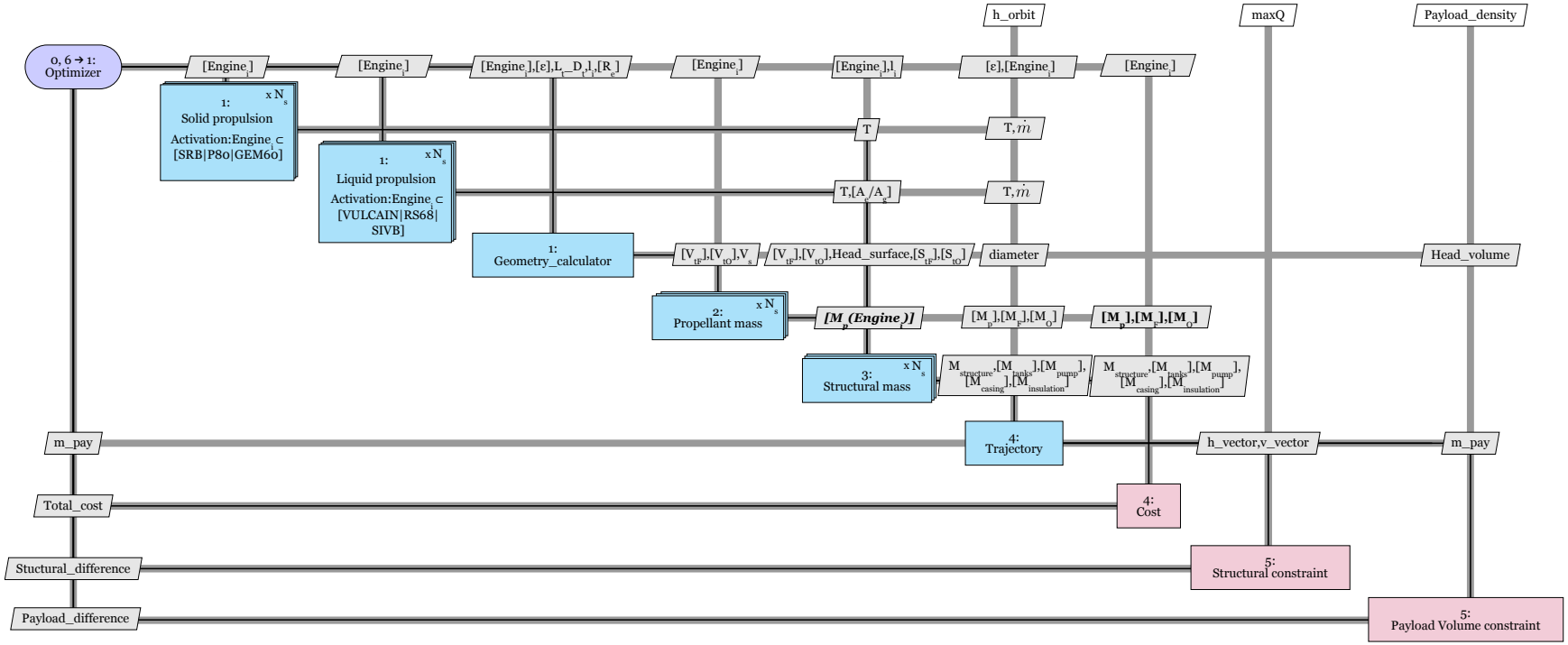


Fig. 16 XDSM of the space rocket benchmark problem depicting the analysis disciplines, constraints and the variable couplings between them. Dynamic XDSM is a provisional extension of XDSM to include the four architectural influences: *Conditional variables* are indicated within square brackets. Variables involved in *data connection* are expressed in bold and italic, including between parenthesis the architectural decision causing them. *Discipline repetition* is indicated in the top right of the discipline block where the multiplicity is a result of an architectural decision. Lastly, the activation condition behind *discipline activation* is expressed in an additional row in the discipline block.

References

[1] Crawley, E., Cameron, B., and Selva, D., *System architecture: strategy and product development for complex systems*, Prentice Hall Press, 2015.

[2] Bussemaker, J. H., Boggero, L., and Nagel, B., "System Architecture Design Space Exploration: Integration with Computational Environments and Efficient Optimization," *AIAA AVIATION 2024 FORUM*, Las Vegas, NV, USA, 2024.

[3] Iacobucci, J. V., *Rapid Architecture Alternative Modeling (RAAM): a framework for capability-based analysis of system of systems architectures*, Georgia Institute of Technology, 2012.

[4] Roelofs, M., and Vos, R., "Uncertainty-Based Design Optimization and Technology Evaluation: A Review," *2018 AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, 2018. https://doi.org/10.2514/6.2018-2029.

[5] DeTurris, D., and Palmer, A., "Perspectives on Managing Emergent Risk due to Rising Complexity in Aerospace Systems," *INCOSE International Symposium*, Vol. 28, No. 1, 2018, pp. 40–54. https://doi.org/10.1002/j.2334-5837.2018.00466.x.

[6] Bussemaker, J. H., and Ciampa, P. D., *MBSE in Architecture Design Space Exploration*, Springer International Publishing, 2023, pp. 589–629. https://doi.org/10.1007/978-3-030-93582-5_36.

[7] NASA, "NASA Systems Engineering Handbook," Tech. Rep. Rev 2, NASA, 2016.

[8] Sobieszczanski-Sobieski, J., Morris, A., van Tooren, M. J., La Rocca, G., and Yao, W., *Multidisciplinary Design Optimization Supported by Knowledge Based Engineering*, Wiley, 2015. https://doi.org/10.1002/9781118897072.

[9] Bruggeman, A. M., and La Rocca, G., "From Requirements to Product: an MBSE Approach for the Digitalization of the Aircraft Design Process," *INCOSE International Symposium*, Vol. 33, No. 1, 2023, pp. 1688–1706. https://doi.org/10.1002/iis2.13107.

[10] Ciampa, P. D., and Nagel, B., "AGILE Paradigm: The next generation collaborative MDO for the development of aeronautical systems," *Progress in Aerospace Sciences*, Vol. 119, 2020, p. 100643. https://doi.org/10.1016/j.paerosci.2020.100643.

[11] Alder, M., Moerland, E., Jepsen, J., and Nagel, B., "Recent advances in establishing a common language for aircraft design with CPACS," *Aerospace Europe Conference 2020*, 2020.

[12] Page Risueño, A., Bussemaker, J., Ciampa, P. D., and Nagel, B., "MDAx: Agile Generation of Collaborative MDAO Workflows for Complex Systems," *AIAA AVIATION 2020 FORUM*, American Institute of Aeronautics and Astronautics, 2020. https://doi.org/10.2514/6.2020-3133.

[13] Garg, S., Bussemaker, J. H., Boggero, L., and Nagel., B., "MDAx: ENHANCEMENTS IN A COLLABORATIVE MDAO WORKFLOW FORMULATION TOOL," 34th Congress of the International Council of the Aeronautical Sciences, ICAS 2024, Florence, Italy, 2024.

[14] Sobieszczanski-Sobieski, J., Agte, J., and Sandusky, R., Jr, "Bi-level integrated system synthesis (BLISS)," *7th AIAA/USAF/-NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 1998, p. 4916.

[15] Bussemaker, J. H., Garg, S., and Boggero, L., "The Influence of Architectural Design Decisions on the Formulation of MDAO Problems," *3rd European Workshop on MDO*, 2022.

[16] Bussemaker, J. H., De Smedt, T., La Rocca, G., Ciampa, P. D., and Nagel, B., "System Architecture Optimization: An Open Source Multidisciplinary Aircraft Jet Engine Architecting Problem," *AIAA AVIATION 2021 FORUM*, American Institute of Aeronautics and Astronautics, 2021. https://doi.org/10.2514/6.2021-3078.

[17] Bussemaker, J. H., García Sánchez, R., Fouda, M., Boggero, L., and Nagel, B., "Function-Based Architecture Optimization: An Application to Hybrid-Electric Propulsion Systems," *INCOSE International Symposium*, Vol. 33, No. 1, 2023, pp. 251–272. https://doi.org/10.1002/iis2.13020.

[18] van Gent, I., "Agile MDAO Systems: A Graph-based Methodology to Enhance Collaborative Multidisciplinary Design," 2019. https://doi.org/10.4233/UUID:C42B30BA-2BA7-4FFF-BF1C-F81F85E890AF.

[19] Martins, J., and Lambe, A., "Multidisciplinary Design Optimization: A Survey of Architectures," *AIAA Journal*, Vol. 51, No. 9, 2013, pp. 2049–2075. https://doi.org/10.2514/1.J051895.

[20] Bussemaker, J. H., Ciampa, P. D., Singh, J., Fioriti, M., Cabaleiro, C., Wang, Z., Peeters, D., Hansmann, P., Vecchia, P. D., and Mandorino, M., "Collaborative Design of a Business Jet Family Using the AGILE 4.0 MBSE Environment," *AIAA AVIATION 2022 FORUM*, Chicago, USA, 2022. https://doi.org/10.2514/6.2022-3934.

[21] Fioriti, M., Cabaleiro De La Hoz, C., Lefebvre, T., Della Vecchia, P., Mandorino, M., Liscouet-Hanke, S., Jeyaraj, A. K., Donelli, G., and Jungo, A., "Multidisciplinary design of a more electric regional aircraft including certification constraints," *AIAA AVIATION 2022 Forum*, American Institute of Aeronautics and Astronautics, 2022. https://doi.org/10.2514/6.2022-3932.

[22] Frank, C., Marlier, R., Pinon-Fischer, O., and Mavris, D., "An Evolutionary Multi-Architecture Multi-Objective Optimization Algorithm for Design Space Exploration," *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Reston, Virginia, 2016, pp. 1–19. https://doi.org/10.2514/6.2016-0414.

[23] Bruggeman, A.-L., Nikitin, M., La Rocca, G., and Bergsma, O., "Model-Based Approach for the Simultaneous Design of Airframe Components and their Production Process Using Dynamic MDAO Workflows," *AIAA SCITECH 2024 Forum*, American Institute of Aeronautics and Astronautics, 2024. https://doi.org/10.2514/6.2024-1530.

[24] Sonneveld, J., van den Berg, T., La Rocca, G., Valencia-Ibáñez, S., van Manen, B., and Bruggeman, A., "Dynamic workflow generation applied to aircraft moveable architecture optimization," 2023. https://doi.org/10.13009/EUCASS2023-544.

[25] Ciampa, P. D., and Nagel, B., "Towards the 3rd generation MDO collaborative environment," *30th ICAS*, 2016, pp. 1–12.

[26] van Gent, I., Ciampa, P., Aigner, B., Jepsen, J., La Rocca, G., and Schut, S., "Knowledge architecture supporting collaborative MDO in the AGILE paradigm," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, , No. June, 2017, pp. 5–9. https://doi.org/10.2514/6.2017-4139.

[27] Boden, B., Flink, J., Mischke, R., Schaffert, K., Weinert, A., Wohlan, A., Ilic, C., Wunderlich, T., Liersch, C. M., Goertz, S., Ciampa, P. D., and Moerland, E., "Distributed Multidisciplinary Optimization and Collaborative Process Development Using RCE," *AIAA Aviation 2019 Forum*, American Institute of Aeronautics and Astronautics, 2019. https://doi.org/10.2514/6.2019-2989.

[28] García Sánchez, R., "Adaptation of an MDO platform for system architecture optimization," *Delft University of Technology*, 2023.

[29] Koelle, D. E., "Handbook of cost engineering for space transportation systems with Transcost 7.2: statistical-analytical model for cost estimation and economical optimization of launch vehicles," 2007.

[30] 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, 2002, pp. 182–197. https://doi.org/10.1109/4235.996017.

[31] Hoerner, S. F., "Fluid-Dynamic Drag. Theoretical, experimental and statistical information," *Copyright by: SF Hoerner Fluid Dynamics, Vancouver, Printed in the USA, Card Number 64-19666*, 1965.

[32] Fedaravičius, A., Kilikevičius, S., and Survila, A., "Optimization of the rocket's nose and nozzle design parameters in respect to its aerodynamic characteristics," *Journal of vibroengineering*, Vol. 14, No. 4, 2012, pp. 1885–1891.