Aerospace Science and Technology 175 (2026) 111748

Contents lists available at ScienceDirect

Aerospace Science and Technology

journal homepage: www.elsevier.com/locate/aescte

Dynamic multidisciplinary design analysis and optimization workflows: Concept and implementation

Anne-Liza M.R.M. Bruggeman [1,*], Gianfranco La Rocca, Maurice F.M. Hoogreef

Delft University of Technology, Faculty of Aerospace Engineering, Kluyverweg 1, Delft, 2629HS, The Netherlands

ARTICLE INFO

Communicated by Kontis Konstantinos

Keywords:
MDAO
(System) architecture optimization
Dynamic MDAO workflows
XDSM
Common MDO workflow schema (CMDOWS)

ABSTRACT

Many architectural design trade-offs must be performed during the conceptual design of complex systems, such as aircraft, to identify promising design concepts. A few architectures are usually selected and traded to keep the process manageable, but this can suffer from biases. Ideally, an optimizer, supported by a multidisciplinary system evaluator, should enable architecture design space exploration. However, incorporating architectural design choices into a multidisciplinary design optimization is challenging due to changing design variables, constraints, and disciplinary tools associated with different architectural designs. This paper proposes a new formal methodology for dynamic Multidisciplinary Design Analysis and Optimization (MDAO) workflows. These workflows allow the design variables, tools, and constraints to change during execution, enabling evaluation and optimization of different architectures within a single MDAO system definition. Switches, branches, and subworkflows are introduced to enable dynamic behavior within the workflow. Together, they allow for a complete mathematical problem definition and consistent formalization. Accordingly, extensions to the eXtended Design Structure Matrix (XDSM) and the Common MDO Workflow Schema (CMDOWS) are presented to enable the visualization, storage, and exchange of dynamic workflows. An automated MDAO formulation, integration, and execution process is extended to ease the setup of these workflows. The methodology has been verified and validated using a mathematical optimization problem from literature. The dynamic workflow successfully discovered a Pareto front comprising multiple architectural design options. Furthermore, the results demonstrate that a single dynamic workflow can identify optimal architectures more efficiently than multiple static workflows, requiring significantly less execution time and fewer function evaluations.

1. Introduction

The primary objective of the conceptual design phase of an aircraft, or other complex engineering systems, is to identify the most promising design concept that satisfies the requirements of all stakeholders involved. Various design trade-offs need to be performed at different levels of the design. For instance, at aircraft architecture level, decisions need to be made on the aircraft configuration [1], type of propulsion system [2], or fuel type [3,4]. At subsystem level, trade-offs focus more on the type of material or the production method to be used [5]. Over the past few decades, aircraft design has become so complex that it is challenging to fully understand all the implications of a design choice [6]. The various disciplines involved, such as aerodynamics, structures, propulsion, and manufacturing, are all tightly interlinked while also being complex systems themselves. As stated by the AIAA MDO Technical Committee, the main challenge in aircraft design is "*how to decide what to change, and to what extent to change it, when everything influences everything else*" [7].

Due to this complexity, digitalization has become one of the key elements in driving innovation in aircraft design [6,8] and has been identified by the European Commission as a key enabler to meet the ambitious sustainability goals for future aviation [9]. Multidisciplinary Design Analysis and Optimization (MDAO) is a methodology that exploits digital solutions to support the decision-making process by searching for optimal designs while accounting for and exploiting multidisciplinary implications [10]. MDAO allows for more design iterations compared to a manual design process [11], and numerous design parameters can be traded simultaneously [12]. Therefore, it can explore the design space more thoroughly and efficiently than would be possible in conventional design, finding optimal solutions which at times include non-intuitive designs, outside the conventional design space [12].

* Corresponding author.
E-mail addresses: A.M.R.M.Bruggeman@tudelft.nl (A.M.R.M. Bruggeman), G.LaRocca@tudelft.nl (G. La Rocca), M.F.M.Hoogreef@tudelft.nl (M.F.M. Hoogreef).
[1] Present address: GKN Fokker Aerospace B.V., Anthony Fokkerweg 4, 3351 NL Papendrecht, email address: Anne-Liza.Bruggeman@fokker.com

https://doi.org/10.1016/j.ast.2026.111748
Received 11 November 2025; Received in revised form 12 January 2026; Accepted 19 January 2026
Available online 22 January 2026

Abbreviations

CMDOWS	Common MDO Workflow Schema
CO	Collaborative Optimization
DoE	Design of Experiments
FPG	Fundamental Problem Graph
IDF	Individual Discipline Feasible
KADMOS	Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System
MDA(O)	Multidisciplinary Design Analysis (and Optimization)
MDF	Multidisciplinary Feasible
MDG	MDAO Data Graph
MPG	MDAO Process Graph
NSEA +	Non-dominated Sorting Evolutionary Algorithm
PIDO	Process Integration & Design Optimization
RCG	Repository Connectivity Graph
UCA	User Customized Action
UCI	User Customized Interface
VSDS	Variable-Size Design Space
XDSM	eXtended Design Structure Matrix
XML	eXtensible Markup Language
XSD	XML Schema Definition

While MDAO is a powerful and proven technique to explore a given system architecture [13], its capability to trade off multiple architectural design options is beyond the state of the art. A design architecture consists, among others, of the system elements and their relationships [14]. Therefore, different architectures may require different analyses and architecture-specific design variables and constraints. For instance, the design variables for a composite part may include the number of plies and fiber directions, while for a metal part, it may include the thickness. In the case the material is also set as a design variable in a Design of Experiments (DoE) or optimization, dependent (or *hierarchical) design variables* will be present that change per iteration. Similarly, the required analysis tools may vary depending on the architecture being evaluated, as some tools are specific to certain architectures. For example, different manufacturability analysis tools may be required to assess the performance and constraints for a composite or a metal component. In general, current approaches do not readily support, in a transparent way, the formulation and execution of MDAO workflows in which the design variables, analysis tools, and constraints can change at each iteration. Therefore, system architectural design choices are generally made outside of the MDAO workflow, based on experience and low-fidelity analyses [15,16]. Only a handful of architectures are selected for further exploration using MDAO [16,17], which may lead to a bias towards more conventional and familiar designs, leaving potential sweet spots in the design space unaddressed [6].

Despite the current limitations in formulating and executing MDAO workflows with changing variables and tools, several methods can be found in literature that still use MDAO to address system architecture optimization problems. One strategy is to ensure that the same MDAO workflow can be used for each architecture [18–21]. In this case, the design variables, analyses, and constraints remain the same for all architectures evaluated. This methodology puts challenging requirements on the flexibility of analysis tools as they must be able to handle a wide variety of input, which is not always feasible. Design vector imputation [21–23] can be used to handle changing design variables in the case of hierarchical design variables. Another approach to system architecture optimization problems is to select a few promising architectures and optimize each in a separate workflow [17]. In principle, this method can be extended to optimize all possible architectures [19,24]. In practice, the number of architectures can be extremely large due to the combinatorial nature of the problem, making exhaustive evaluation of the architecture design space computationally unsustainable [19,25]. Filtering techniques such as compatibility rules or matrices [25–27], boundary conditions [19], or constraints [24,27] can be used to discard infeasible architectures before analyzing them. Still, many computational resources are allocated to evaluate suboptimal architectures [24].

Alternatively, nested optimizations (consisting of inner and outer optimization loops) can be performed to allocate more resources to promising architectures [25,28–30]. The outer optimization loop takes the architectural design decisions and is therefore responsible for finding the best architecture, while the inner optimization loop sizes and optimizes a given architecture. The main challenge here is that, according to the decisions taken by the outer optimization loop, an inner optimization loop must be formulated and completed for the architecture that needs to be optimized (per iteration of the outer loop).

Methods have been developed to automatically create a workflow on the fly (for the inner loop) based on a specific architecture (selected by the outer loop), for example, by connecting mathematical equations to each potential component in a system [15,30–32]. Based on the architectural design decisions made by the outer optimizer, components and their corresponding mathematical equations are connected, resulting in a mathematical model that can be used to evaluate the system architecture. Using the nested optimization approach, a large variety of architectural designs can be evaluated while computational resources are directed to the more favorable architectures.

While creating optimization loops on the fly has proven to be effective (it allows for changing disciplines, design variables, and constraints), it suffers from transparency issues and is error-prone. The inner optimization loops are generated on the fly, and therefore never explicitly formulated or documented beforehand, thereby hampering traceability of design decisions, which is one of the limitations hindering MDO adoption as identified by Belie [33]. Inspection and debugging become challenging, making it difficult to ascertain whether the optimization process is aligned with the intended problem formulation. Furthermore, the methodology may require a significant amount of problem-specific programming [29,34], which can result in error-prone and time-consuming setup of the workflow and hamper reusability. The nested optimization approach does not allow to simultaneously make architectural design decisions whilst also optimizing the individual architectures. Evaluations of nested optimizations can be more time-consuming than having a single optimization and may become intractable for large design problems with many variables and architectural choices.

To fully integrate architectural design choices in the optimization problem and to overcome the limitations discussed above, we propose *dynamic MDAO workflows*. We define dynamic MDAO workflows as workflows in which the active set of design variables, constraints, and disciplinary tools may change as a function of the current design point [35]. While prior work has begun to explore similar directions [36,37], the methodology proposed in this paper strives for inspectability and traceability with the possibility of having only a single optimization loop, whilst also providing a consistent mathematical problem definition. Industrial adoption of dynamic workflows requires the MDAO system definition to be fully explicit, with all interconnections defined and traceable. Such transparency is essential for the certifiability of the design process, ensuring that all steps and dependencies can be verified and validated.

This paper proposes a new, formal methodology that supports the formulation of dynamic workflows that are fully inspectable before execution, while allowing changes in the process flow during execution. By introducing new elements, namely switches and workflow branches, a static visualization of the dynamic MDAO workflow is created that can be inspected and debugged by the user while allowing the workflow to dynamically adjust the design variables, tools, and constraints during execution depending on the architectural design choices made by the

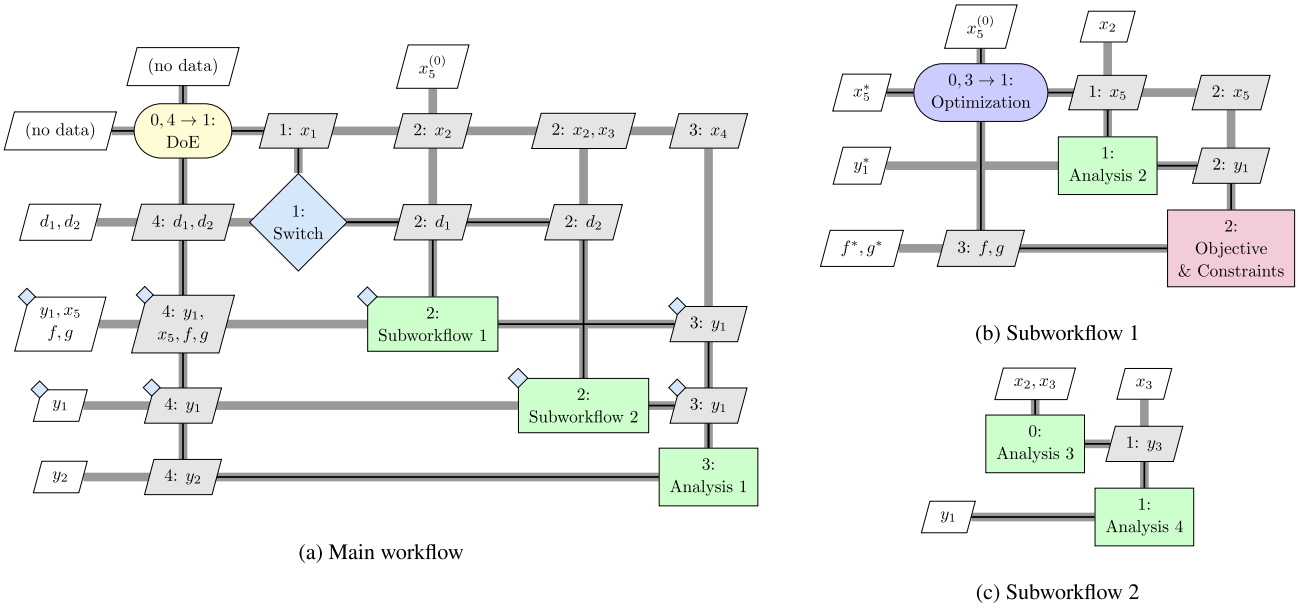


Fig. 1. Example of a dynamic MDAO workflow including one switch and two subworkflows.

optimizer. All of this can be achieved within a single optimization loop, therefore not requiring a nested optimization approach. While the application of the proposed dynamic workflows has been presented in previous work related to the design and manufacturing of an airframe component [38], this paper elaborates on the methodology's foundations, the formalism, and the technology developments that enable their deployment in a collaborative engineering environment. Finally, the proposed methodology and its implementation are verified and validated using a benchmark mathematical problem.

The structure of the paper is as follows. Section 2 introduces an illustrative example of a dynamic MDAO workflow to provide the conceptual context. Section 3 then presents the formal definition of the methodology. This includes an augmented version of the eXtended Design Structure Matrix (XDSM) [39] to visualize dynamic MDAO workflows, and the corresponding mathematical formalism. Next, the implementation of the methodology in an automated MDAO workflow formulation and execution chain is presented in Section 4, including the extension of the Common MDO Workflow Schema (CMDOWS) [40], proposed in previous work to store and exchange MDAO system formulations. Section 5 describes the mathematical case used to verify and validate the proposed methodology and evaluate the computational efficiency of the proposed dynamic workflows with respect to the traditional approach, where each architecture is addressed in a separate static optimization. The paper ends with the conclusion and recommendations in Section 6.

2. Dynamic MDAO workflows

Dynamic MDAO workflows are characterized by their ability to adapt their execution flow during runtime. At each iteration, different disciplinary blocks and/or constraint functions may be activated or deactivated depending on the current architectural design. To illustrate this concept, Fig. 1 presents an example dynamic workflow using an XDSM representation that contains newly introduced symbols and variables, defined in the next sections. Fig. 1(a) shows the main workflow, which contains a switch that can activate different parts of the workflow, namely subworkflow 1 or subworkflow 2, shown in Fig. 1(b) and (c), respectively. Depending on the architecture evaluated in a given iteration, either subworkflow 1 or subworkflow 2 is executed, and the active subworkflow may change from one iteration to the next. Subworkflows may themselves contain an optimization, as illustrated by subworkflow 1 in this example. With this concept, different design variables, analysis tools, and/or constraints can be activated and deactivated, as discussed in more detail in the next sections.

3. Formalization of dynamic MDAO workflows

To enable the workflow flexibility to activate or deactivate different parts of the workflow, we introduce three new concepts (Fig. 2):

1. **Switch**
   A switch is a new conditional element in the MDAO workflow/problem definition that (de-)activates different branches of a workflow. It receives input from the optimizer and/or other disciplinary tools and, based on the execution conditions of each branch, determines which branch to execute. An execution condition is a conditional expression, consisting of a parameter, constraint operator (equal to, greater than, etc), and reference value. One branch can have multiple execution conditions. At each iteration, a different branch may be executed.

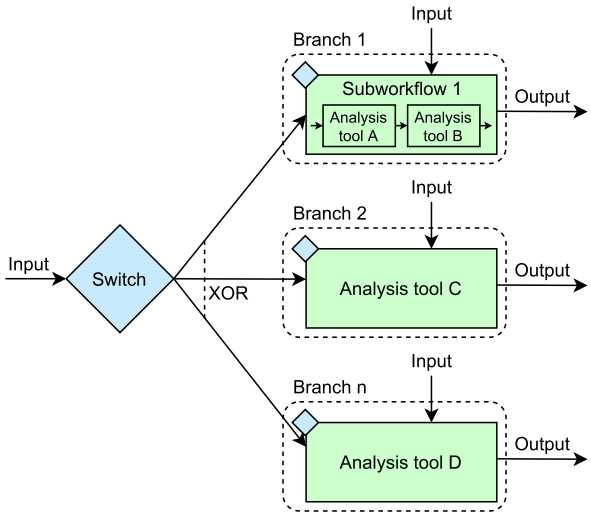


Fig. 2. Overview of three concepts to enable dynamic MDAO workflows, namely the switch, branch, and subworkflow.

2. **Branch**
   A branch is a part of an MDAO workflow, such as a disciplinary tool, chain of disciplinary tools, or subworkflow, that is activated or deactivated by a switch. A branch has associated conditional expressions that are evaluated by the switch to determine whether it should be executed. Note that the branches connected to a given switch are mutually exclusive, meaning that a switch can activate only one branch per iteration.
3. **Subworkflow**
   A subworkflow is a workflow in a workflow and hence introduces different "levels" in the MDAO problem formulation. On the main level, the subworkflow appears as a single executable block, while it is rather a pointer to a lower-level subworkflow. As a subworkflow can be an MDAO workflow itself, it may also contain switches, branches, and subworkflows. A subworkflow does not always have to be a branch depending on a switch, but can also exist as an independent component in a workflow.

These three concepts form the foundation of the methodology proposed in this paper. By combining these concepts, a dynamic workflow can be obtained that allows for changing analysis tools, design variables, and constraints during execution. Based on the values of the architectural design variables, the switches adapt the workflow in such a way that only the relevant analysis tools are executed. The introduction of subworkflows is one way of dealing with dependent design variables and activating constraints when required. Alternatively, default values can be assigned to inactive constraints, as will be explained in more detail in Section 3.2. Another method for addressing dependent design variables within a single optimizer is design vector imputation, as implemented by Bussemaker et al. [29]. The latter approach is not implemented in this paper.

This chapter presents the formalization of the dynamic workflows. First, an extension to the XDSM is proposed in Section 3.1 to be able to visualize the workflows. The XDSM for a general Multidisciplinary Feasible (MDF) Gauss-Seidel strategy, including the dynamic elements, is presented in Section 3.2. The corresponding mathematical equations are given in Section 3.3.

3.1. Proposed extensions to the extended design structure matrix

The current standard used in MDAO to visualize a workflow is the XDSM as developed by Lambe and Martins [39]. To be able to visualize the dynamic workflows, including the newly introduced concepts, an extension to the XDSM is proposed. Four new graphical elements are introduced as detailed in Table 1. No new icon for a subworkflow is proposed, as the subworkflow can already be represented by a discipline block.

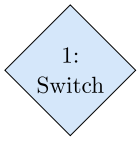
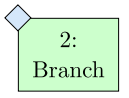
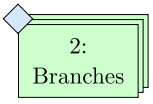
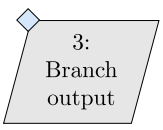
3.2. XDSM formalization of dynamic workflows

With the proposed extensions to the XDSM, it is possible to fully formalize and visualize dynamic workflows, as shown in Fig. 3. For this example, the MDF Gauss-Seidel MDAO strategy has been selected [41]. One switch with three branches is present. Two of these branches are subworkflows. The XDSM of subworkflow 1 is shown in Fig. 4(a). This subworkflow consists of a Gauss-Seidel convergence loop. The XDSM of subworkflow 2 (Fig. 4(b)) consists of an optimization loop without any converger.

The used nomenclature, shown in Table 2, is similar to the one used by Martins and Lambe [41]. New subscripts are introduced to identify the functions and variables that apply to a branch $()_b$, a switch $()_s$, or a specific subworkflow $()_{sub}$. Furthermore, an additional class of variables has been introduced, referred to as *decision variables*, denoted by $\mathbf{d}$. Decision variables are Boolean indicators that determine whether a branch should be executed. Each decision variable is generated as an output of the switch and serves as an input to a corresponding branch. Every branch receives one decision variable as input. The switch evaluates the predefined conditions for all branches and sets the decision variable of a branch to 1 (or True) when its conditions are satisfied. As the branches are mutually exclusive, only one decision variable can be True in a given iteration. All other decision variables are set to 0 (or False), effectively disabling their associated branches during that iteration.

The output of a branch can be divided into two variable groups: $\mathbf{y}_{b_0}$ and $\mathbf{y}_{b_j}$. Variables in $\mathbf{y}_{b_0}$ are computed by all branches, ensuring their values are always available for use by other disciplines - for example, the cost or weight of a given component. Although each branch may cal-

Table 1
Overview of the new XDSM graphical elements.

Graphical Element	Explanation
	The switch is represented by a blue diamond. The diamond shape is used because decisions are typically represented by a diamond in flowcharts. The color blue has been chosen as this one has not yet been used by any other element in the XDSM.
	A branch is represented as a standard discipline block, with a small blue diamond in the upper left corner. This diamond indicates that the branch is activated by the adjacent switch. A branch icon can represent either a single analysis tool or a subworkflow.
	To more compactly visualize all branches of a switch, a stacked discipline representation is used. Each stacked block includes a small blue diamond in the upper left corner. Although the branches are displayed in parallel (similar to parallel disciplines), they are mutually exclusive - only one branch is selected for execution in each iteration.
	The output of a branch is visualized using a standard output block, also marked with a small blue diamond in the upper left corner. This indicates that the output is a conditional variable - it is only computed when the corresponding branch is executed. Additionally, multiple branches from a switch may produce the same output variable, but only the output from the selected branch is calculated during a given iteration.

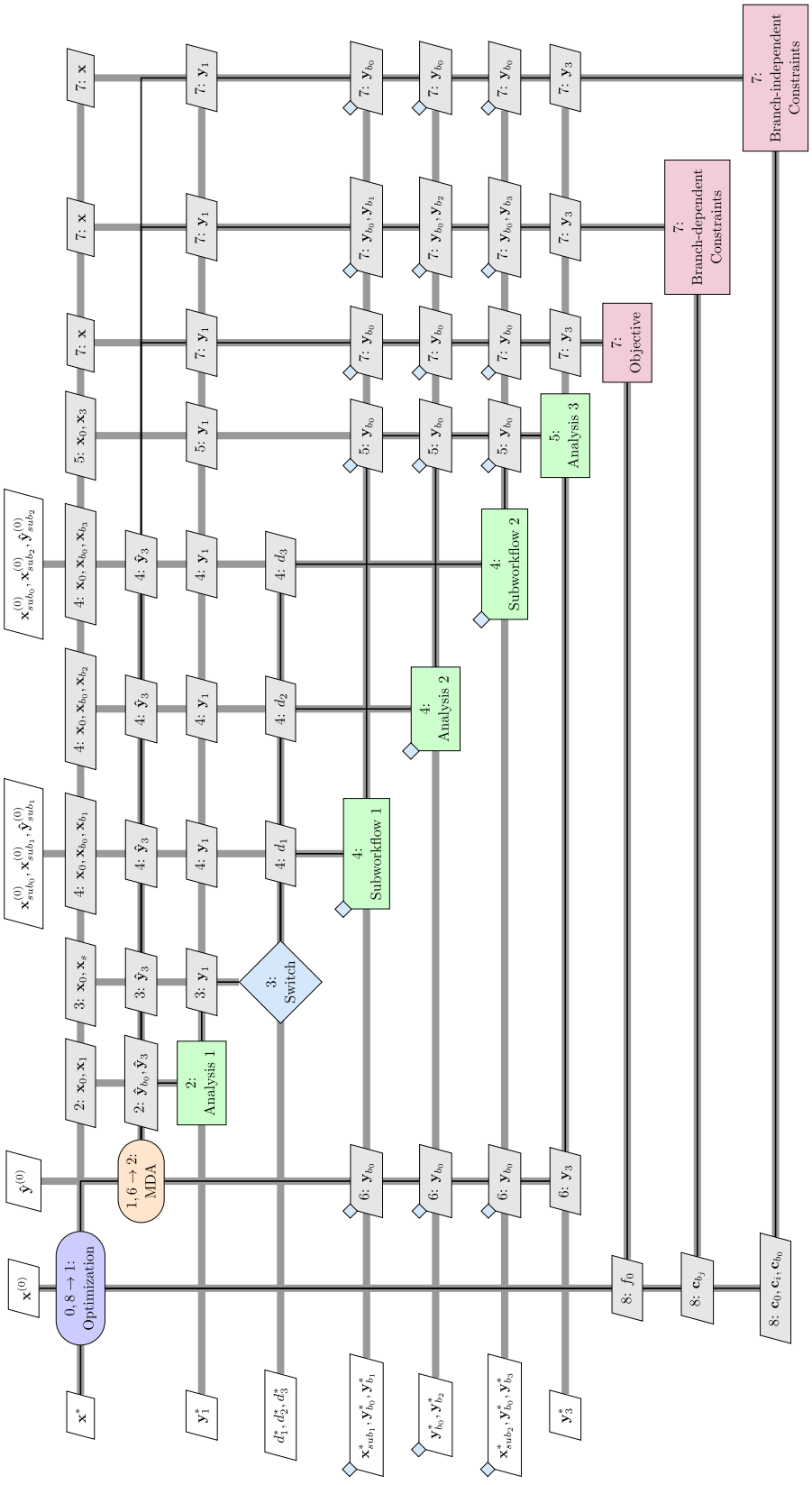
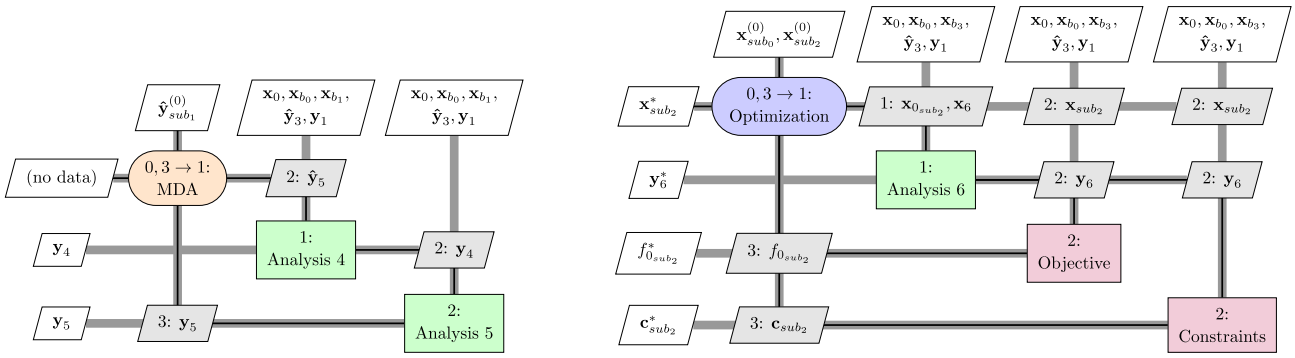


Fig. 3. XDSM representation of an MDAO system based on the MDF Gauss-Seidel strategy, including a switch, branches, and subworkflows.



(a) XDSM representation of subworkflow 1 consisting of a multidisciplinary analysis block converged using the Gauss-Seidel scheme

(b) XDSM representation of subworkflow 2 consisting of an optimization without converger

Fig. 4. XDSM representations of the subworkflows as used in Fig. 3.

culate these variables differently, they are consistently computed across all branches. In contrast, $\mathbf{y}_{b_j}$ variables are computed exclusively by a specific branch and thus are only available when that particular branch is executed. This means that any discipline depending on $\mathbf{y}_{b_j}$ variables should be included in the corresponding j branch, with the exception of constraint blocks, as elaborated later in this section. For consistency between the input and output, the design variables are also divided into two groups: $\mathbf{x}_{b_0}$ and $\mathbf{x}_{b_j}$. The design variables in $\mathbf{x}_s$ are used only as input for switches. Together with other design or coupling variables, they are used by the given switch to determine the branch to execute.

The constraint block contains two types of constraints: branch-dependent and branch-independent. Branch-independent constraints are those that must be evaluated regardless of the selected branch. A branch-dependent constraint is only required for a specific product architecture and depends on $\mathbf{y}_{b_j}$. This constraint is evaluated only when a specific branch is executed. Branch-dependent constraints of not executed branches are automatically assigned a default value to ensure the constraint is satisfied and not active. Note that branch-dependent constraints can be evaluated at different levels. In the example in Figs. 3 and 4, the constraints of subworkflow 1 are evaluated by the optimizer at the main level. The constraints in subworkflow 2 can either be evaluated in the subworkflow itself, at the main level, or a combination of both.

Special attention must be given to the input and the output of subworkflows. All input variables required by a subworkflow must be supplied by the main workflow. This includes initial guesses necessary to start convergence loops or optimizations within the subworkflow, which must also be present in the main workflow, for example, $x^{(0)}_{sub_2}$ in Fig. 3. These variables can be either an input to the main workflow or variables calculated by other disciplines. Similarly, the output of a subworkflow (including the final values of design variables) needs to be fed back to the main workflow. Therefore, the following set of equations holds for the output of the subworkflows for the MDF example:

$$\begin{bmatrix}\mathbf{y}_{b_0}\\ \mathbf{y}_{b_1}\end{bmatrix} = \begin{bmatrix}\mathbf{y}_4\\ \mathbf{y}_5\end{bmatrix}, \quad \begin{bmatrix}\mathbf{y}_{b_0}\\ \mathbf{y}_{b_3}\end{bmatrix} = \begin{bmatrix}\mathbf{x}^*_{sub_2}\\ \mathbf{y}^*_6\\ f^*_{0_{sub_2}}\\ \mathbf{c}^*_{sub_2}\end{bmatrix} \tag{1}$$

Note that an input variable can be defined as a design variable at only one level, either at the main level or within a subworkflow. Defining the same design variable at both levels leads to inconsistency, as the resulting design would not align with the value assigned at the main workflow level.

A more compact visualization of the dynamic workflow presented in Fig. 3 is shown in Fig. 5. This visualization uses the stacked branches graphical element from Table 1.

3.3. Mathematical formulation

The mathematical formulation of the main level optimization, as presented in Fig. 3, is given in set of Eqs. 2.

$$\begin{aligned}
\text{minimize}\quad & f_0(\mathbf{x},\mathbf{y})\\
\text{with respect to}\quad & \mathbf{x}\\
\text{subject to}\quad & \mathbf{c}_0(\mathbf{x},\mathbf{y}) \geq 0\\
& \mathbf{c}_i(\mathbf{x}_0,\mathbf{x}_i,\mathbf{y}_i) \geq 0\\
& \mathbf{c}_{b_0}(\mathbf{x}_0,\mathbf{x}_{b_0},\mathbf{x}_{b_{j,d_j=1}},\mathbf{y}_{b_0}) \geq 0 \text{ where } \begin{cases} d_j = 1\\ d_{m\neq j} = 0\end{cases}\\
& \begin{cases}\mathbf{c}_{b_j}(\mathbf{x}_0,\mathbf{x}_{b_0},\mathbf{x}_{b_j},\mathbf{y}_{b_0},\mathbf{y}_{b_j}) \geq 0 & \text{if } d_j = 1\\ \mathbf{c}_{b_{j_{default}}} \geq 0 & \text{if } d_j = 0\end{cases}\\
\text{where}\quad & \mathbf{y}(\mathbf{x},\mathbf{y})\\
& \mathbf{y}_i(\mathbf{x}_0,\mathbf{x}_i,\mathbf{y}_{b_0},\mathbf{y}_{k\neq i})\\
& \mathbf{y}_{b_0}(\mathbf{x}_0,\mathbf{x}_{b_0},\mathbf{x}_{b_{j,d_j=1}},\mathbf{y}_i) \text{ where } \begin{cases} d_j = 1\\ d_{m\neq j} = 0\end{cases}\\
& \mathbf{y}_{b_j}(\mathbf{x}_0,\mathbf{x}_{b_0},\mathbf{x}_{b_j},\mathbf{y}_i)\\
& \mathbf{d}(\mathbf{x}_0,\mathbf{x}_s,\mathbf{y}_i)\\
\text{for}\quad & i = 1,k,\ldots,N_d \text{ and } j = 1,m,\ldots,N_b
\end{aligned} \tag{2}$$

Table 2
Overview of the mathematical notation used within the dynamic MDAO problem formulations.

Symbol	Definition
$\mathbf{c}$	Vector of design constraints
$\mathbf{d}$	Vector of decision variables
f / $\mathbf{f}$	Objective function(s)
N_b	Number of branches
N_d	Number of disciplines
$\mathbf{x}$	Vector of design variables
$\mathbf{y}$	Vector of coupling variables
Subscripts	
$()_b$	Functions or variables specific to branches of a switch
$()_i$ / $()_j$	Functions or variables specific to discipline i or branch j
$()_s$	Functions or variables specific to a switch
$()_{sub}$	Functions or variables specific to a specific subworkflow
$()_0$	Functions or variables shared by multiple disciplines or branches
Superscripts	
$()^{(0)}$	Initial guesses for functions or variables
$()^*$	Functions or variables at their optimal value
$\hat{()}$	Independent copies of variables distributed to other disciplines

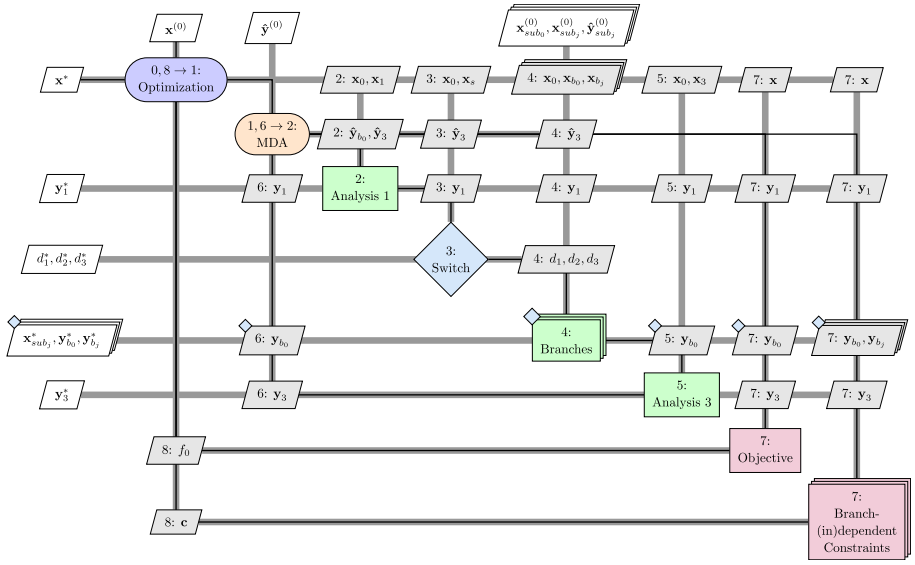


Fig. 5. XDSM representation for an MDAO system based on the MDF Gauss-Seidel strategy, including the compact representation of the branches.

The definition of the design variables, coupling variables, constraints, and decision variables used in this optimization is given in set of Eqs. (3).

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_i \\ \mathbf{x}_s \\ \mathbf{x}_{b_0} \\ \mathbf{x}_{b_j} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_i \\ \mathbf{y}_{b_0} \\ \mathbf{y}_{b_j} \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{c}_i \\ \mathbf{c}_{b_0} \\ \mathbf{c}_{b_j} \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} d_1 \\ \dots \\ d_{N_b} \end{bmatrix} \tag{3}$$

As shown in set of Eqs. (2), four different types of constraints can be formulated: constraints that apply to the global problem, $\mathbf{c}_0$; constraints that apply to a specific discipline $\mathbf{c}_i$; constraints that apply to all branches, $\mathbf{c}_{b_0}$; constraints that apply only to a specific branch, $\mathbf{c}_{b_j}$. $\mathbf{c}_{b_0}$ is always calculated and its input depends on which branch has been executed. $\mathbf{c}_{b_j}$ is only calculated when branch j is executed, hence when decision variable (d_j) equals one. If not, a default value ($\mathbf{c}_{b_{j_{default}}}$) is assigned to ensure the constraint is satisfied but remains inactive.

Similar to the constraints, the optimization problem includes three types of coupling variables. Branch-dependent coupling variables ($\mathbf{y}_{b_j}$) are computed only when their corresponding branch is executed. Branch-independent coupling variables ($\mathbf{y}_{b_0}$) are always evaluated, but their computed values depend on the selected branch. Finally, $\mathbf{y}_i$ is also always evaluated, independent of the selected branch.

The mathematical formulation of subworkflow 2 from Fig. 4(b) is presented in set of Eqs. (4).

$$\begin{aligned} \text{minimize} \quad & f_{0_{sub_2}}(\mathbf{x}_{sub}, \mathbf{y}_{sub}(\mathbf{x}_{sub}, \mathbf{y}_{sub})) \\ \text{with respect to} \quad & \mathbf{x}_{sub} \\ \text{subject to} \quad & \mathbf{c}_{0_{sub_2}}(\mathbf{x}_{sub}, \mathbf{y}_{sub}(\mathbf{x}_{sub}, \mathbf{y}_{sub})) \geq 0 \\ & \mathbf{c}_{i_{sub_2}}(\mathbf{x}_{0_{sub}}, \mathbf{x}_{i_{sub}}, \mathbf{y}_{i_{sub}}(\mathbf{x}_{0_{sub}}, \mathbf{x}_{i_{sub}}, \mathbf{y}_{k \neq i_{sub_2}})) \geq 0 \\ & \text{for } i = 1, k, \dots, N_{d_{sub_2}} \end{aligned} \tag{4}$$

The definition of the design variables, coupling variables and constraints within subworkflow 2 is given in set of Eqs. (5).

$$\mathbf{x}_{sub} = \begin{bmatrix} \mathbf{x}_{0_{sub_0}} \\ \mathbf{x}_{i_{sub_0}} \\ \mathbf{x}_{0_{sub_2}} \\ \mathbf{x}_{i_{sub_2}} \end{bmatrix}, \quad \mathbf{y}_{sub} = \begin{bmatrix} \mathbf{y}_{i_{sub_2}} \end{bmatrix}, \quad \mathbf{c}_{sub} = \begin{bmatrix} \mathbf{c}_{0_{sub_2}} \\ \mathbf{c}_{i_{sub_2}} \end{bmatrix} \tag{5}$$

Although a subworkflow may include dynamic elements, none are present in this example. Therefore, only two types of constraints and one type of coupling variables are present. This is equivalent to a standard MDF formulation as described by Martins and Lambe [41].

As explained in Section 3.2, the output of a subworkflow needs to be fed back into the main workflow. Therefore, set of Eqs. (6) relates the output of the subworkflows to the variables in the main workflow.

$$\begin{bmatrix} \mathbf{y}_{b_0} \\ \mathbf{y}_{b_1} \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{sub_1} \end{bmatrix}, \quad \begin{bmatrix} \mathbf{y}_{b_0} \\ \mathbf{y}_{b_3} \end{bmatrix} = \begin{bmatrix} \mathbf{x}^*_{sub_2} \\ \mathbf{y}^*_{sub_2} \\ f^*_{0_{sub_2}} \\ \mathbf{c}^*_{sub_2} \end{bmatrix} \tag{6}$$

4. Integration of dynamic workflows into automated MDAO problem formulation and execution

One of the main challenges in applying MDAO is the time-consuming and error-prone process of setting up an executable workflow [11]. To address this, various methods and tools have been developed to support the user in formulating MDAO problems and automating executable workflow setup [42–45]. These approaches resulted in a significant reduction of setup time, from 40 to 97% for different use cases [46,47]. However, introducing dynamic aspects into MDAO workflows increases their complexity and, consequently, their setup time. Therefore, the dynamic aspects have been integrated into the automated MDAO workflow formulation and execution process proposed by Van Gent and La Rocca [44].

4.1. The MDAO workflow formulation and execution process

An overview of the MDAO process as presented by Van Gent and La Rocca [44] is shown in Fig. 6. The process is based on the idea that an MDAO problem must be fully formulated before it can be translated into an executable workflow. The formulation phase starts with a repository containing all available disciplinary tools, including those not necessarily relevant to the current problem. The user then defines the design variables, objective(s), and constraints and selects the appropriate tools needed to solve the problem. The final step in the formulation phase is the selection of an MDAO strategy [41]. Available strategies include optimization strategies (e.g., MDF, IDF, CO), multidisciplinary analysis (MDA), or DoE approaches. Once the problem is fully formulated, it is translated into an executable workflow for a selected Process Integration & Design Optimization (PIDO) tool, such as Optimus [62], RCE [59], or OpenMDAO [48]. Executing this workflow results in an (optimal) design and often leads to iterative refinements of the problem formulation, including adjustments to analysis tools or solution strategy.

Different tools are available to support the different phases of the MDAO process. In this study, KADMOS (Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System) [44] is used to automate workflow formulation. Optimus by Noesis Solutions [62] is used for workflow execution. The formulated workflow is exported from KADMOS using CMDOWS [40]. To enable import into Optimus, an in-house developed Optimus-CMDOWS importer, based on the work of Hoogreef et al. [50], translates the CMDOWS file into an executable Optimus workflow. To support dynamic workflows, new features have been developed and integrated across all four tools, KADMOS, CMDOWS, the Optimus-CMDOWS importer, and Optimus, as detailed in the following section.

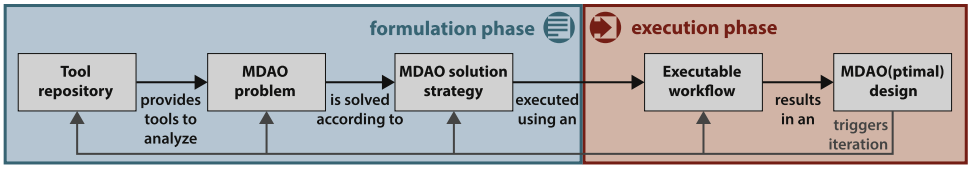


Fig. 6. Overview of an MDAO process illustrating the formulation of the MDAO problem followed by its translation into an executable workflow, based on Van Gent [49].

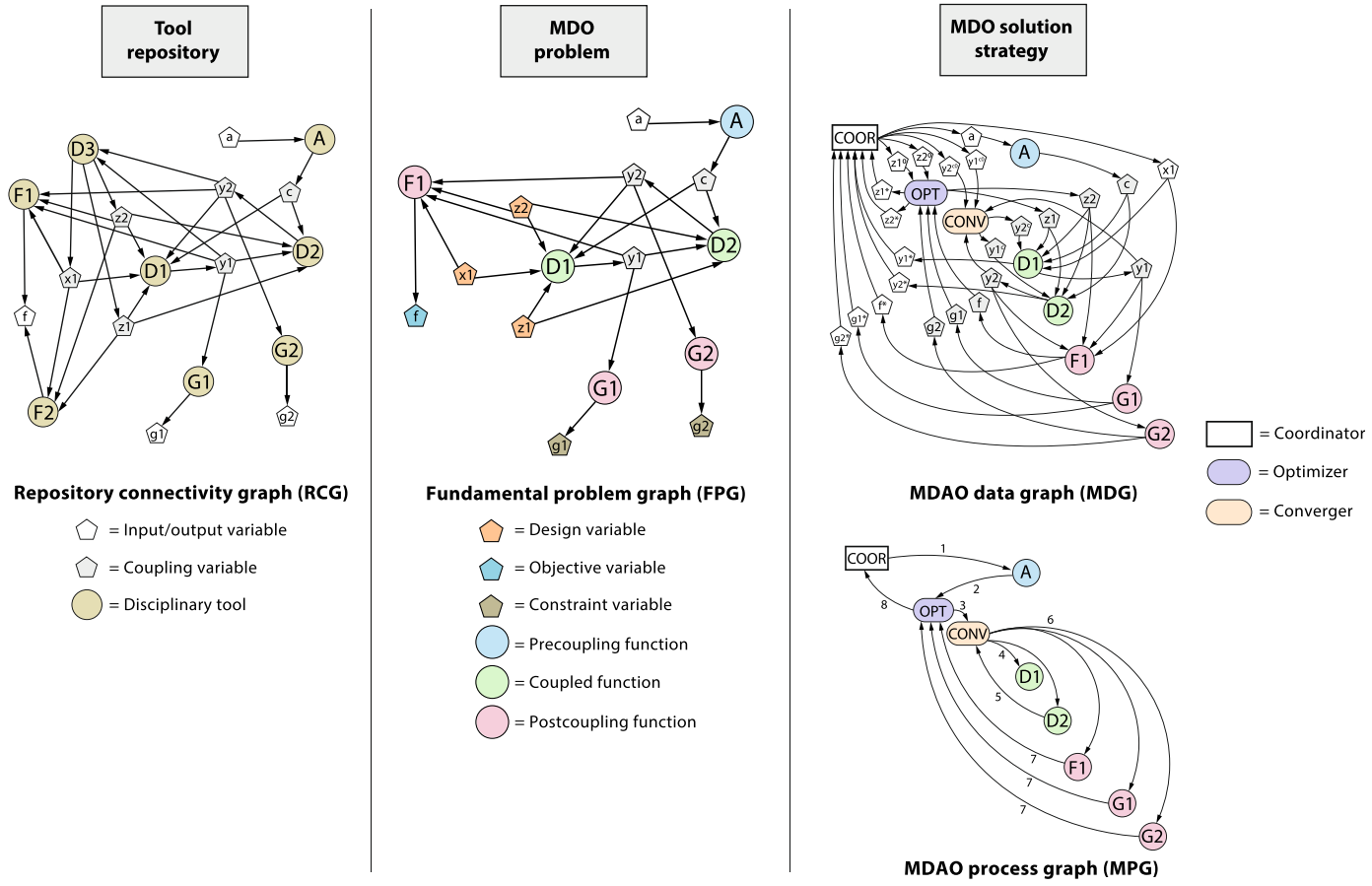


Fig. 7. Overview of the three steps performed in KADMOS, along with the corresponding graphs generated at each step. Figure based on the work of Van Gent et al. [40].

4.2. KADMOS extension for dynamic workflow formulation

KADMOS is a graph-based MDAO workflow formulation tool developed in Python [44], which extends the work by Pate et al. [51]. It is open-source and available on the TU Delft Gitlab [60]. Starting from a repository of disciplinary tools, KADMOS automatically formulates MDAO workflows based on user input. To generate consistent workflow graphs, the input and output of all disciplinary tools must adhere to a common schema such that KADMOS can automatically establish data connections between the tools. The graphs used by KADMOS are directed graphs consisting of two node types: function nodes and variable nodes. The direction of the edges indicates the data flow, i.e., which variables serve as input or output to specific functions. Different graphs are created for each of the three steps in the formulation phase of the MDAO process, as presented in Fig. 6. An overview of these steps and corresponding graphs is given in Fig. 7. For a detailed explanation of the graph structure and internal processes within KADMOS, the reader is referred to the work of Van Gent and La Rocca [44].

To enable dynamic workflow formulations, two new KADMOS extensions have been developed: one for handling subworkflows and one for managing switches and branches. These extensions are explained using the example from Fig. 1.

4.2.1. Subworkflows

Fig. 8 illustrates the KADMOS implementation of subworkflow 1 from the example problem of Fig. 1. On the main level (left side of the figure), the subworkflow is represented as a function node. The variable nodes and the directed edges indicate the subworkflow's input and output. The function node includes an additional attribute, `graph`, which stores a KADMOS graph object representing the internal structure of the subworkflow (right side of the figure). The input variables used in the subworkflow are the same as those used in the main workflow. This is also true for the output variables, where the optimal result (indicated with a superscript star) is fed back into the main workflow.

A new function `add_subworkflow(name, repository, tools)` has been created to add a subworkflow to the Repository Connectivity Graph (RCG) formulated by KADMOS in the first step of the MDAO system formulation process. With this function, the user can assign a name to the subworkflow and specify what tools to include. KADMOS then automatically creates the subworkflow and takes care of adding the input and output variables from the subworkflow to the main workflow.

For each (sub)workflow, the user needs to define the MDAO strategy and variable problem roles, such as design variable or constraint. These are then automatically applied in the second and third steps of the KADMOS process. If a subworkflow generates additional output due to the selected strategy, such as objective(s), constraints, or final design variable values - as is the case in Fig. 1(b) - KADMOS automatically adds these extra variables as new output variables in the upper-level workflow. Note that there is no limitation to the number of workflow levels that can be introduced in the MDAO problem; a subworkflow can also contain a subworkflow.

4.2.2. Switches & branches

A new switch element has been introduced in KADMOS to integrate the switch logic and its associated branches. As shown in Fig. 9, the switch element is implemented as a specialized function node that includes an additional attribute `conditions`, which stores the activation conditions for each branch. The switch can be added using the function `add_switch(name, branches, conditions)`, which allows the user to specify the name of the switch, define the tools or subworkflows that constitute each branch, and assign the corresponding activation conditions. The switch is treated by KADMOS as a standard analysis tool with respect to its input and output. Therefore, the conditions must adhere to the same data schema used by other analysis tools. KADMOS automatically generates edges between the variables used in the conditions and the switch. Furthermore, it creates new decision variables for each branch and connects them appropriately within the workflow graph. As shown in Fig. 9, a decision variable is represented by a variable node. Once a switch is added to the RCG, the MDAO problem can be formulated as usual. As branches must always follow their associated switch in the executable workflow, the switch and its branches are treated as a single composite node by the KADMOS sequencing algorithms. These algorithms are used by KADMOS to determine the execution order of the disciplinary tools. For more information on the sequencing algorithms implemented in KADMOS, the reader is referred to Bruggeman [52].

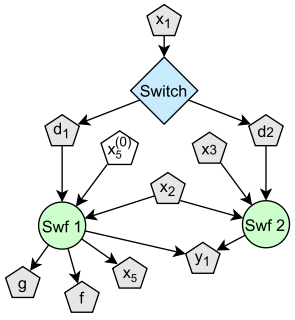


Fig. 9. KADMOS extension to integrate the switch. The switch is represented as a function node, the decision variables as variable nodes.

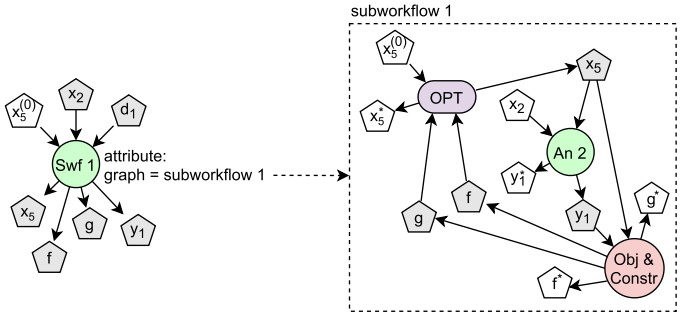


Fig. 8. KADMOS extension for subworkflows. On the main level, the subworkflow is represented as a function node containing an extra attribute, `graph`. The value of this attribute is a KADMOS graph object. In this example, both graphs are of the type MDAO Data Graph (MDG), shown in Fig. 7.

4.3. Storage and exchange of dynamic MDAO workflows using CMDOWS

To store MDAO workflow formulations, CMDOWS is used. CMDOWS is an open-source eXtensible Markup Language (XML) schema definition (XSD) that facilitates not only the storage but also the exchange of MDAO workflow formulations across various process integration tools. A comprehensive description of CMDOWS is provided in Van Gent et al. [40], and the schema is publicly available via the TU Delft Gitlab repository [61]. A high-level overview of the CMDOWS schema is shown in Fig. 11 to highlight the CMDOWS extensions developed in this study to support dynamic workflows. In this figure, black XML elements represent the original CMDOWS schema, while red XML elements indicate the schema extensions. CMDOWS organizes all information related to MDAO workflows into three categories, namely `Information`, `Nodes`, and `Connections`. In the `Information` category, general information describing the CMDOWS file is stored, such as file version, organization data, and problem definition. The `Nodes` category stores all nodes present in the MDAO workflow. These include executable blocks, such as mathematical functions and disciplinary tools (design competences), variable nodes, and architecture[1] elements (e.g., optimizers, convergers, DoEs). The third category `connections` stores all data and process relationships between elements in the workflow.

To support the formalization, storage, and exchange of dynamic MDAO workflows, three new concepts have been introduced in the CMDOWS schema: subworkflows, switches, and branches. These extensions are discussed in the following sections.

4.3.1. Subworkflows

The top part of Fig. 11 shows in red the subworkflow extension. As discussed in Section 4.2.1, KADMOS treats a subworkflow as an executable block in the main workflow. Therefore, a `subWorkflow` element has been added to the `executableBlocks` element in the `Nodes` category. Just as any disciplinary tool, a subworkflow can be described using `uID`, `label`, `inputs`, and `outputs`. Upper and lower bounds, or a list containing allowed value options for an input, can be given here as well.

Subworkflows are stored using either the `cmdows` or `cmdowsFile` element. The `cmdows` element is of the same type as the root of the main workflow (as indicated by the horizontal red arrow in Fig. 11), reflecting the concept of a workflow within a workflow. This allows both the main workflow and its subworkflows to be defined within a single XML file. Alternatively, subworkflows can be defined in separate CMDOWS files, in which case, a reference path is provided using the `cmdowsFile` element. This enables the reuse of previously formulated workflows as subworkflows within new workflows.

Fig. 12 presents an example of how the subworkflows from Fig. 1 are stored in the CMDOWS schema. Each subworkflow is an executable block in the main workflow. This executable block represents a workflow, and therefore, the subworkflow element contains a new instantiation of the cmdows element. This cmdows element contains again the three categories `Information`, `Connections`, and `Nodes`, storing the information specific to the subworkflow.

4.3.2. Switches & branches

The bottom part of Fig. 11 shows in red the added elements to formalize switches and their branches. Switches are stored under the `architectureElements/executableBlocks` element, as they contribute to the architecture of an MDAO system, similar to an optimizer or converger, but are not part of the underlying physical problem that is solved.

Each `switch` element has a `uID`, `label`, and at least two `branches`. Each branch has three child elements, namely `conditions`, `relatedExecutableBlockUIDs`, and `decisionVariable`. The `conditions` element specifies when a branch will be executed. One branch can have multiple conditions. A condition consists of a `parameterUID`, `constraintOperator` (equal to, greater than, smaller or equal then, etc.) and a `referenceValue`. In case of equality conditions, a `requiredEqualityPrecision` can be provided.

The `relatedExecutableBlockUIDs` element determines which mathematical functions, design competences, or subworkflows are included in the branch. The value of each related UID element, therefore, refers to one of the uIDs present in the `executableBlocks` as indicated by the vertical red arrow in Fig. 11. The `decisionVariable` indicates which decision variable corresponds to the given branch.

An example of how the switch is integrated into the CMDOWS schema is shown in Fig. 13 using the example from Fig. 1.

4.4. Materialization and execution of dynamic MDAO workflows using optimus

Once an MDAO workflow has been formulated, it needs to be translated into an executable workflow. The automatic translation of a formulated workflow into an executable one specific for a PIDO tool of choice is called *materialization* [50]. In previous work [50,53–55], importers have been developed to interpret CMDOWS files of static workflows and materialize executable (static) workflows into Optimus [62], OpenMDAO [48] and RCE [59]. In this work, a new CMDOWS-Optimus importer has been developed in Python to enable also the materialization of dynamic workflows. Optimus was selected as PIDO tool due to its native support for key functionalities required to implement switches and subworkflows, as will be detailed in the following sections.

4.4.1. Subworkflows

Subworkflows are natively supported in Optimus through the OptInOpt interface, which allows the main workflow to reference and trigger the execution of a separate Optimus workflow. An OptInOpt consists of three elements as shown in Fig. 10: an input UCI (blue frame), a UCA (yellow frame), and an output UCI (red frame). UCI stands for *User Customized Interface* and UCA for *User Customized Action*. The input UCI handles data exchange between the main and subworkflow. The UCA triggers the execution of the referenced subworkflow. The output UCI retrieves the output values from the subworkflow and supplies them back to the main workflow.

Fig. 10. Example of an OptInOpt interface, enabling subworkflows in Optimus.

4.4.2. Switches & branches

While a switch element was already available in Optimus, it was not initially compatible with the OptInOpt interface. To enable their integration, a custom switch was developed specifically to support the combination of switches with subworkflows. An example of this combination is shown in Fig. 14. The switch element evaluates the conditions for each branch and activates the correct one. Next, the input UCI (blue frame) provides the required input to the selected subworkflow which is executed by the UCA (yellow frame). Finally, the output UCI (red frame) collects the output from the subworkflow and transfers it to the main workflow. Regardless of which branch is executed, the structure of the output from the output UCI remains consistent. As discussed in Section 3, branches may have different input and output variables; however, this variability is seamlessly handled within Optimus. All input (design variables and calculated input) required for the different branches is provided and, depending on which branch is executed,

[1] Note that so far the term *architecture* has been used to indicate a product architecture. However, traditionally the MDAO strategy is also referred to as MDAO architecture; hence, the architectural elements here are referring to the MDAO architecture.

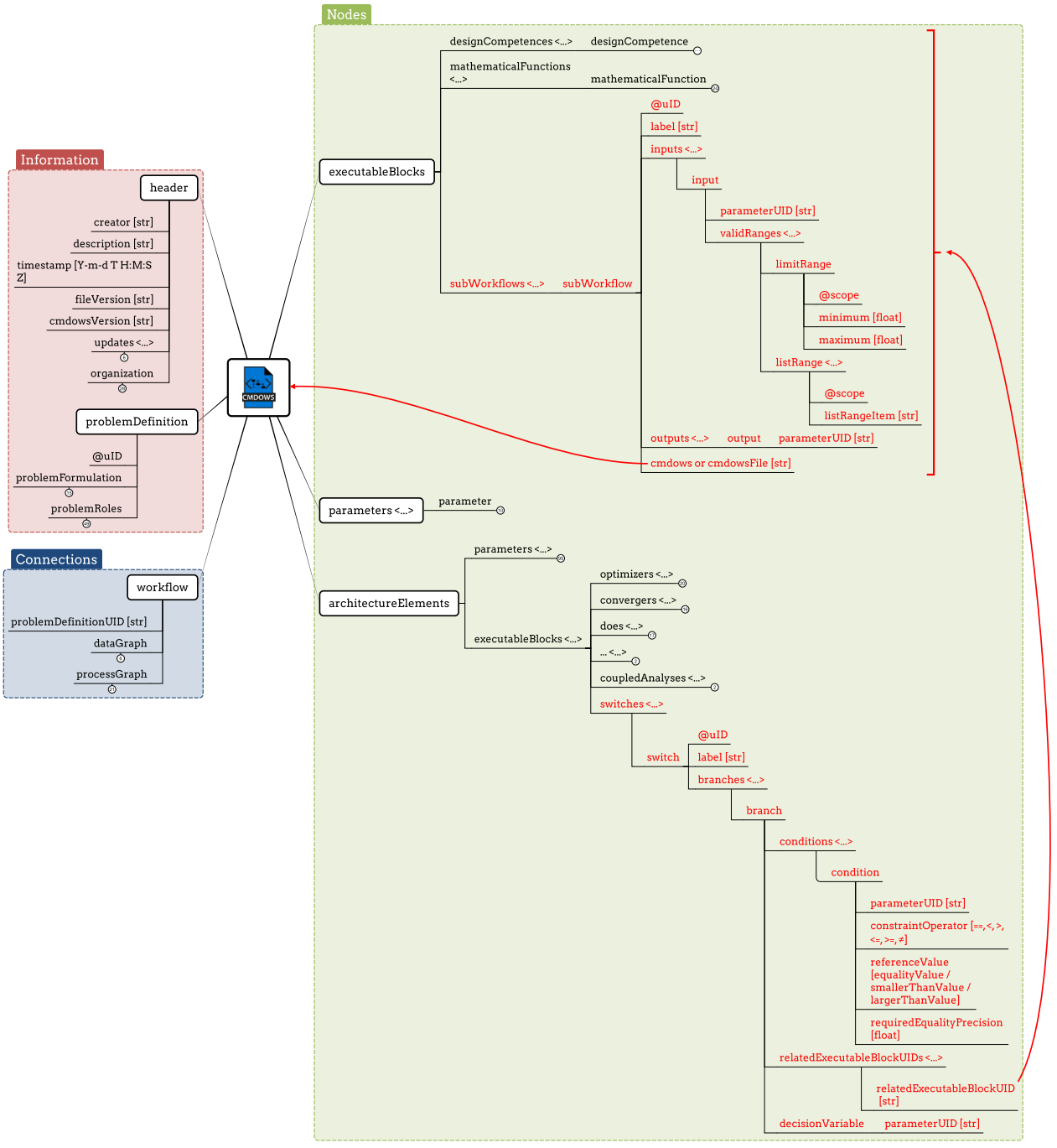


Fig. 11. Overview of the extended CMDOWS format. Workflow data are stored in three main categories: information, nodes, and connections. The extensions to enable the dynamic workflows are indicated in red. The `executableBlocks` element is extended with the new `subWorkflows` element. This includes a reference to a subworkflow, specified as a `cmdows` element (indicated with the horizontal red arrow) or using a reference path to a different CMDOWS file, using `cmdowsFile`. `architectureElements` is extended with the new `switches` element. The switch has at least two branches, which consist of one of the `executableBloks`, indicated with the vertical red arrow. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

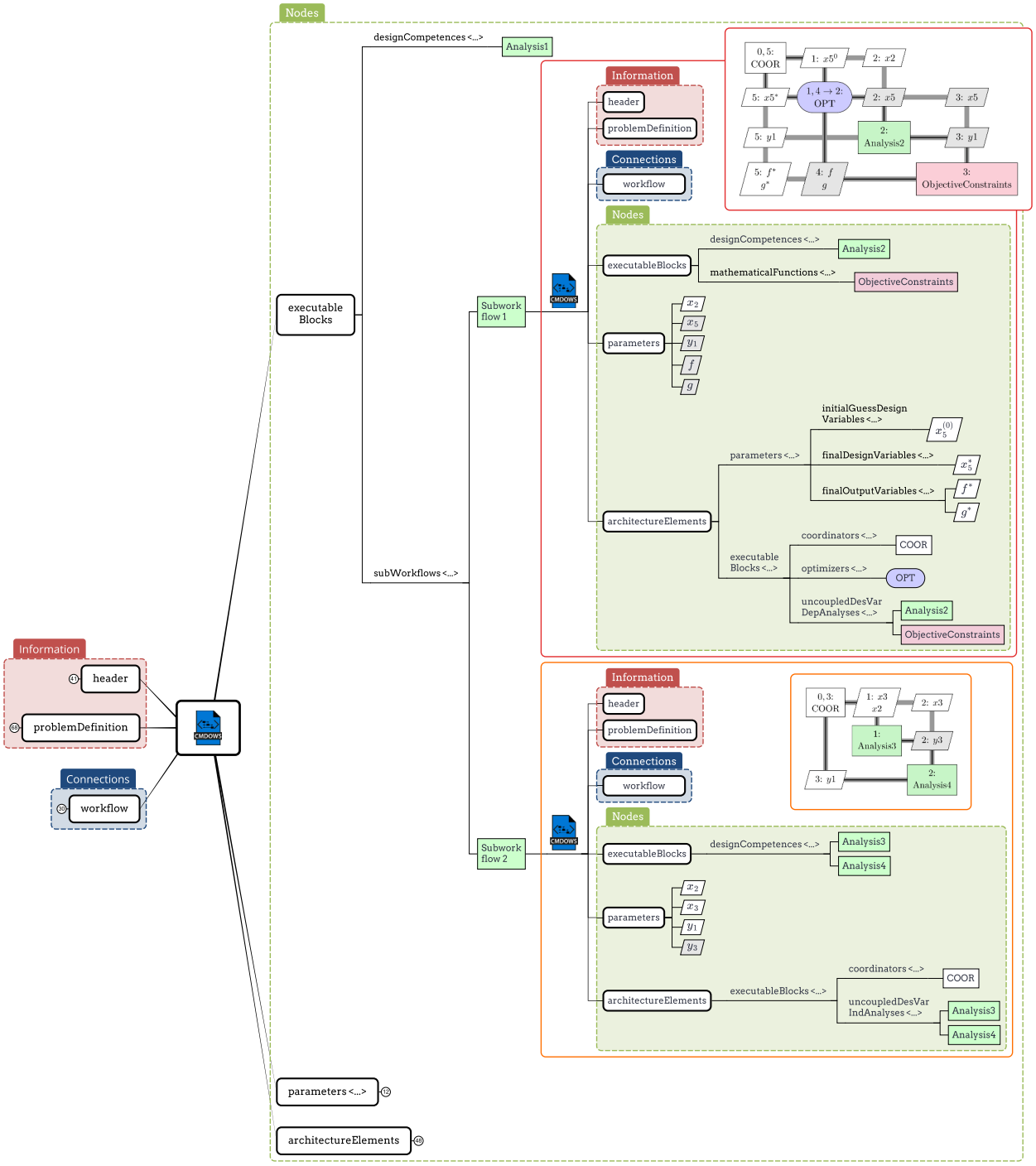


Fig. 12. Visualization on how the subworkflows from Fig. 1 are stored in the CMDOWS schema. In this example, the main workflow and the subworkflows are defined within the same CMDOWS file.

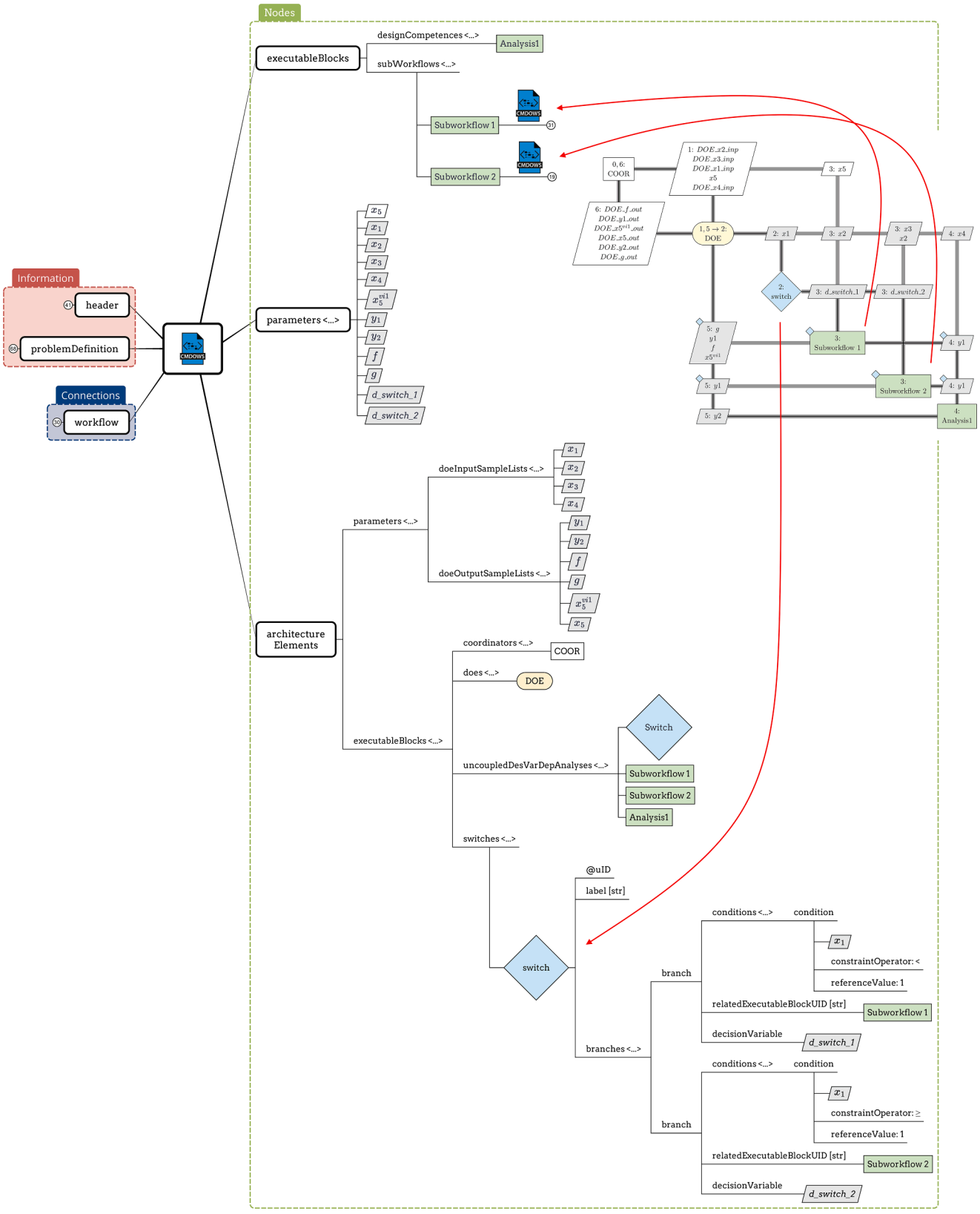


Fig. 13. Visualization on how the switch from Fig. 1 is represented in the CMDOWS schema.

some is not used. This may sound computationally expensive, however if the output of a certain analysis tool is required for only a single branch, such tool should be included in that specific branch. A default value (e.g. NaN) is assigned to the output variables of the untriggered branches.

An example Optimus workflow, including a switch, branches, and subworkflows, is shown in Fig. 15. This workflow is based on the example from Fig. 1 and has been materialized using the CMDOWS-Optimus plugin. The main workflow, shown in the top left part of the figure, consists of one switch with two branches, each one containing a subworkflow. The first subworkflow is visualized on the top right of the figure. It includes an OptInOpt and a standard toolchain used to generate the XML file required as output by the switch. The inclusion of a nested OptInOpt is necessary, because in Optimus the optimization method must be applied to an entire Optimus graph. In this case, the optimization should only be applied to a specific part of the workflow (namely, the graph shown on the bottom right), as the XML output should only be generated once the optimization is complete. The second subworkflow (shown in the bottom left) implements an MDA rather than an optimization. Therefore, it does not contain an OptInOpt but it does contain the same toolchain as subworkflow 1 to generate the required XML output for the switch. These toolchains are added automatically by the CMDOWS-Optimus importer.

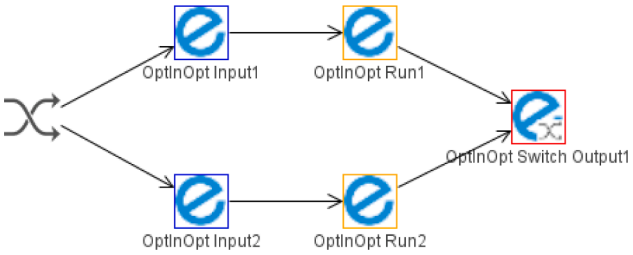


Fig. 14. Example of switch and OptInOpt implementation in Optimus.

5. Verification & validation

To verify and validate the proposed dynamic workflow methodology, it has been applied to a mathematical test case for which results have been previously published. This is the multi-objective Variable-Size Design Space (VSDS) Goldstein function, as formulated by Valencia-Ibáñez [21] and presented in set of Eqs. (7)–(9). This formulation is a multi-objective extension of the original VSDS Goldstein function formulated by Pelamatti et al. [56]. The multi-objective version was used as verification case instead of the original one because the results obtained in this research differed from those of Pelamatti et al. [56], but aligned with those presented by Valencia-Ibáñez [21]. Manual verification of some points confirmed the correctness of the results reported by Valencia-Ibáñez [21]. An example calculation is given in Appendix B.

The multi-objective VSDS Goldstein function serves as a representative test case for architectural optimization problems involving discrete architectural decisions. It includes two architectural design variables, w_1 and w_2, with four and two options respectively, resulting in eight distinct architectures. Each architecture corresponds to a unique subproblem, characterized by slightly different sets of input variables, equations, and constraints. The variable-size version of the Goldstein function was chosen as this problem has a different number and type of design variables for each architecture, which is a typical characteristic in a system architecture optimization problem. Furthermore, even though the objectives are the same for each architecture, the mathematical equations used to calculate these objectives differ. This is representative for an architecture optimization problem, where, for example, the cost of a product can be computed using different methods, according to the specific architecture being evaluated.

The overall optimization problem has eleven design variables: five continuous ($\mathbf{x}$), four discrete ($\mathbf{z}$), and two architectural ($\mathbf{w}$). It has two objective functions (f and f_b) and one constraint (g), as shown in set

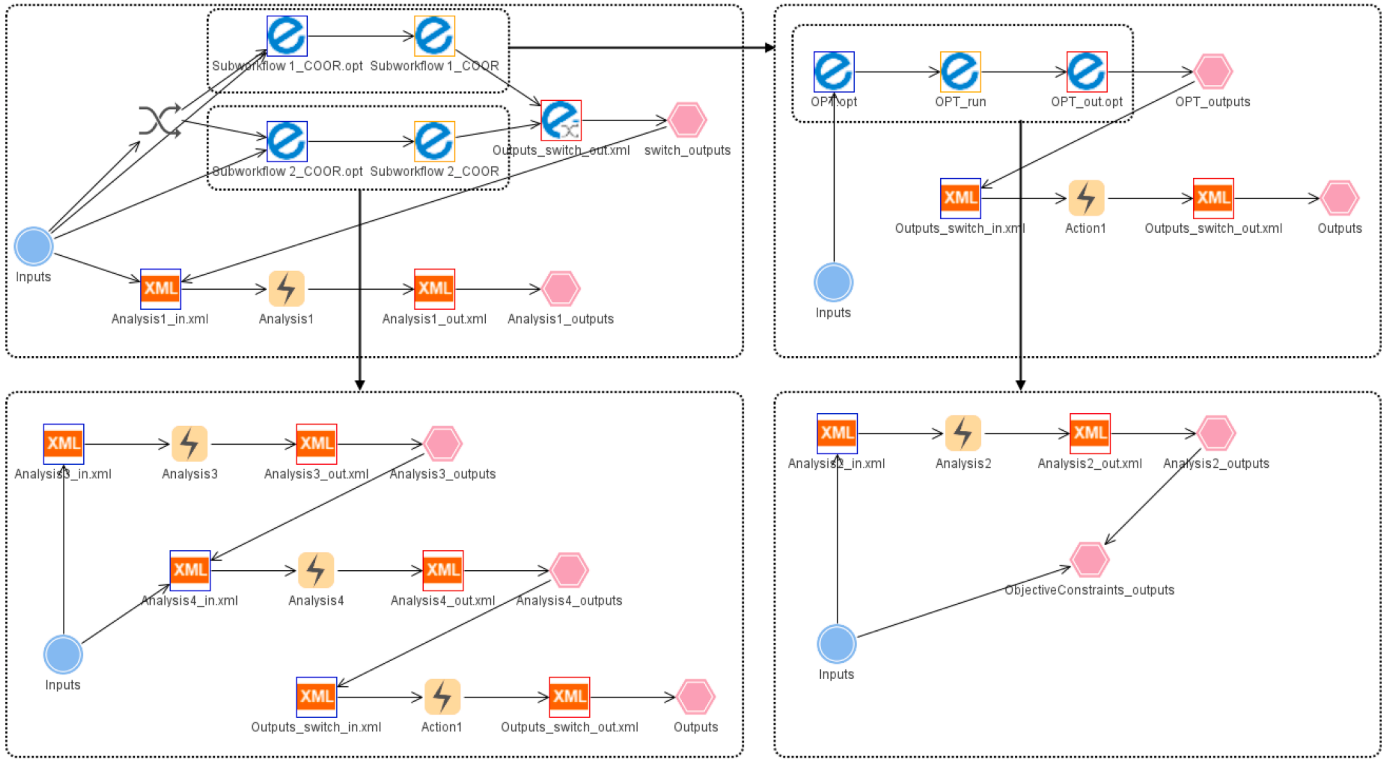


Fig. 15. Optimus workflow representing the dynamic workflow from Fig. 1. The workflows have been automatically generated (*materialized*) using the CMDOWS-Optimus importer.

of Eqs. (7)–(9). The architecture number used in set of Eqs. (8) is defined in Table 3. The difference between the discrete variables **w** and **z** is that **w** represents architectural design choices determining which architecture is evaluated, while **z** represents a discrete design variable used to optimize a given architecture. The full formulation of the eight objective and four constraint functions is provided in Appendix A, based on the formulations by Pelamatti et al. [56] and Valencia-Ibáñez [21].

$$
\begin{aligned}
&\text{minimize} && f(\mathbf{x},\mathbf{z},\mathbf{w}),\, f_b(\mathbf{x},\mathbf{z},\mathbf{w}) \\
&\text{with respect to} && \mathbf{x} = \{x_1,\dots,x_5\} \text{ with } x_i \in [0,100] \text{ for } i = 1,5 \\
&&& \mathbf{z} = \{z_1,\dots,z_4\} \text{ with } z_i \in \{0,1,2\} \text{ for } i = 1,4 \\
&&& \mathbf{w} = \{w_1, w_2\} \text{ with } w_1 \in \{0,1,2,3\} \text{ and } w_2 \in \{0,1\} \\
&\text{subject to} && g(\mathbf{x},\mathbf{z},\mathbf{w}) \le 0,
\end{aligned} \tag{7}
$$

where:

$$
f(\mathbf{x},\mathbf{z},\mathbf{w}) = \begin{cases}
f_1(x_1,x_2,z_1,z_2,z_3,z_4) & \text{if } w_1 = 0 \text{ and } w_2 = 0 \\
f_2(x_1,x_2,x_3,z_2,z_3,z_4) & \text{if } w_1 = 1 \text{ and } w_2 = 0 \\
f_3(x_1,x_2,x_4,z_1,z_3,z_4) & \text{if } w_1 = 2 \text{ and } w_2 = 0 \\
f_4(x_1,x_2,x_3,x_4,z_3,z_4) & \text{if } w_1 = 3 \text{ and } w_2 = 0 \\
f_5(x_1,x_2,x_5,z_1,z_2,z_3,z_4) & \text{if } w_1 = 0 \text{ and } w_2 = 1 \\
f_6(x_1,x_2,x_3,x_5,z_2,z_3,z_4) & \text{if } w_1 = 1 \text{ and } w_2 = 1 \\
f_7(x_1,x_2,x_4,x_5,z_1,z_3,z_4) & \text{if } w_1 = 2 \text{ and } w_2 = 1 \\
f_8(x_1,x_2,x_3,x_4,x_5,z_3,z_4) & \text{if } w_1 = 3 \text{ and } w_2 = 1,
\end{cases}
$$

$$
f_b(\mathbf{x},\mathbf{z},\mathbf{w}) = f(100 - \mathbf{x},\mathbf{z},\mathbf{w}) + \frac{5}{7} \cdot \text{architecture number} \tag{8}
$$

and:

$$
g(\mathbf{x},\mathbf{z},\mathbf{w}) = \begin{cases}
g_1(x_1,x_2,z_1,z_2) & \text{if } w_1 = 0 \\
g_2(x_1,x_2,z_2) & \text{if } w_1 = 1 \\
g_3(x_1,x_2,z_1) & \text{if } w_1 = 2 \\
g_4(x_1,x_2,z_3,z_4) & \text{if } w_1 = 3
\end{cases} \tag{9}
$$

Table 3
Overview of the eight architecture/subworkflow definitions.

Subworkflow/Architecture number	1	2	3	4	5	6	7	8
Objective function	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
Constraint function	g_1	g_2	g_3	g_4	g_1	g_2	g_3	g_4

Due to the presence of architectural design variables, which are categorical or discrete by nature, gradient-based optimization methods are typically not applicable at the system level. Moreover, as the set of active design variables and model outputs may change between iterations, complete gradient information cannot be guaranteed. Consequently, system-level optimization is generally limited to non-gradient-based methods, and therefore, evolutionary algorithms are used in this use case. However, individual subworkflows may encapsulate continuous variables optimization problems (for example subworkflow 2 in Fig. 4(b)), for which gradient-based methods can still be effectively employed to improve computational efficiency.

The XDSM of the main workflow is shown in Fig. 16. An example of the XDSM for architecture 1 is shown in Fig. 17. The discipline C1 evaluates Tables A.1 through A.4 as shown in Appendix A, to get the values of x_3, x_4, c_1 and c_2. Discipline F1 evaluates the two objective equations (f_1 and f_b), while G1 evaluates the constraint equation g_1.

5.1. Performance of multi-objective dynamic workflow

The dynamic workflow methodology described in Sections 3 and 4 has been applied to the multi-objective VSDS Goldstein function to assess the correct working (Section 5.1.1) and performance (Section 5.1.2) of the workflow.

5.1.1. Comparison of Pareto optimal solutions between static and dynamic workflows

To verify the dynamic workflows, a baseline was established by solving the multi-objective optimization for each of the eight architectures of

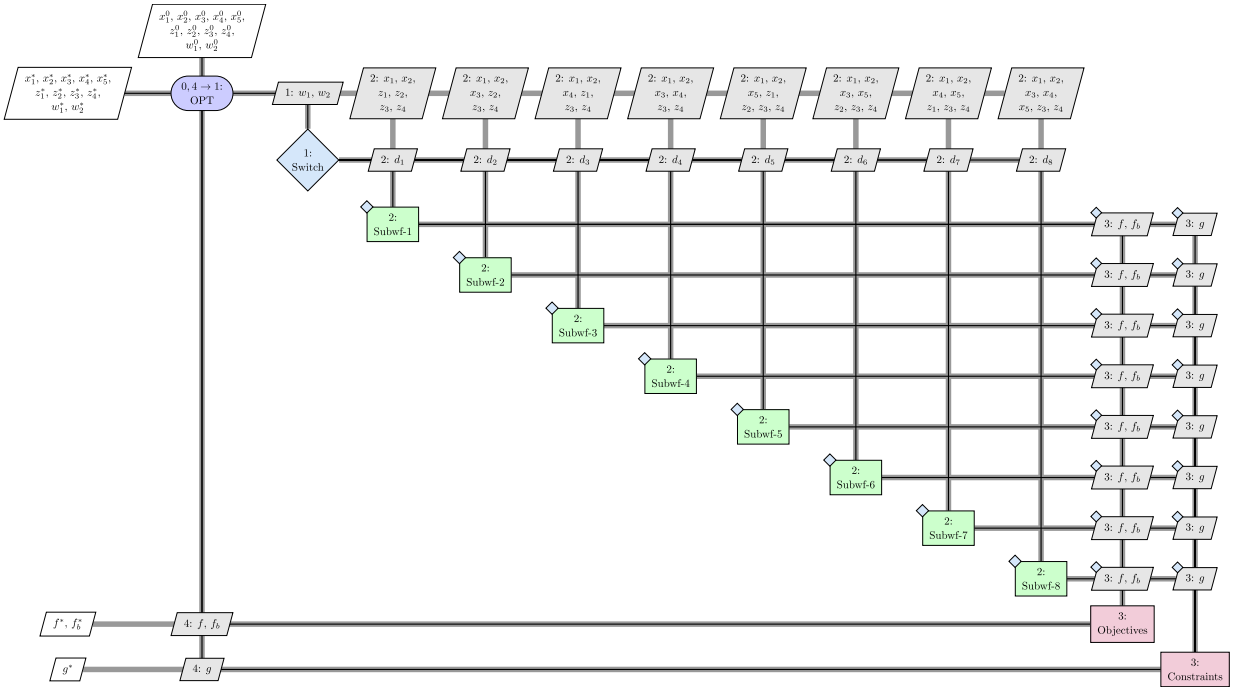


Fig. 16. XDSM formalization of the top-level workflow of the multi-objective Variable Size Design Space Goldstein function.

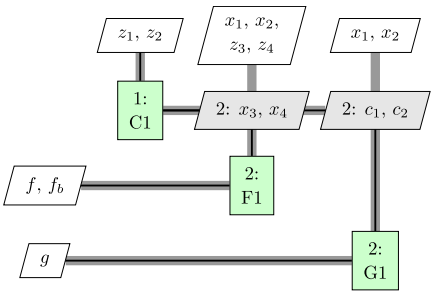


Fig. 17. XDSM formalization of the subworkflow representing system architecture 1 of the Variable Size Design Space Goldstein function. Discipline C1 calculates the value of x_3, x_4, c_1, and c_2 according to Tables A.1 to A.4. F1 calculates the two objective values and G1 calculates the constraint value.

the VSDS Goldstein function separately, using static MDAO workflows (meaning one MDAO workflow for each architecture). The results are shown in Fig. 18. Each optimization was performed using the NSEA+ (Non-dominated Sorting Evolutionary Algorithm) until a Pareto front with at least 1000 points was found. The used optimization settings are shown in Table 4. The values for the weighting factor and inverse crossover probability were determined through a series of tests using different parameter values. All optimizations used a population size of 100 individuals, consistent with the setup used in the work of Valencia-Ibáñez [21]. The initial populations were randomly generated using different random seeds. The Pareto fronts shown in Fig. 18(a) are similar to the Pareto fronts found by Valencia-Ibáñez [21] as shown in Fig. 18(b). From these figures, it can be concluded that the overall Pareto front consists of a combination of three architectures, namely 5 (purple), 7 (pink), and 8 (grey).

Fig. 19 shows the Pareto front obtained using the dynamic MDAO workflow implementation presented in Fig. 16. This optimization was performed using the same NSEA+ algorithm and settings. The optimization process terminated once 1000 non-dominated solutions were found. For comparison, the results from the static workflows (black dots) and the dynamic workflow (in color) are overlapped in Fig. 19. The comparison clearly indicates that the dynamic workflow successfully identified all architectures on the Pareto front with similar accuracy.

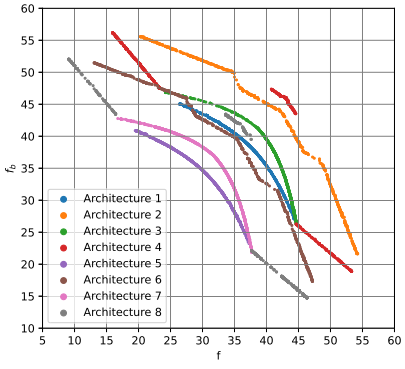
The main advantage of the dynamic workflow over the static workflows lies in its execution efficiency. The dynamic workflow achieved an impressive 70% reduction in computational time compared to solving each static Pareto front individually, as shown in Table 5. This substantial time saving highlights the key benefit of enabling the optimizer to make architectural design decisions within a single, dynamic workflow: the optimizer can steer the search towards more promising architectures, reducing the number of function evaluations required to construct the Pareto front. This is shown in Fig. 20. 95% of the function evaluations of the dynamic workflow were performed on a Pareto-optimal architecture, while only 41% of the function evaluations for the static workflows were performed on a Pareto-optimal architecture. This targeted exploration of the dynamic workflows contributes directly to the reduced execution time. Note that the dynamic workflow in this work did not use any imputation of inactive design variables. Evaluation of the results showed that in 16% of the evaluated design points, inactive design variables were varied. The use of imputation could therefore reduce the execution time even further.

Table 4
Optimization settings used to solve the VSDS Goldstein function.

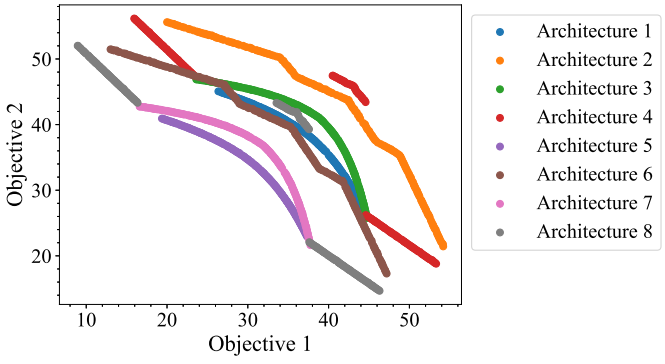
	Multi-objective optimization	Single-objective optimization
PIDO tool	Optimus 2021.1SP2	Optimus 2021.1SP2
Optimization algorithm	NSEA+	Differential evolution
Population size	100	10 · nr. design variables
Weighting factor	0	0.7
Inverse crossover probability	0.3	0.85
Average stopping stepwidth	N/A	0.15

Table 5
Execution times for the static and dynamic multi-objective optimizations.

Workflow	Execution time [s]	Number of function evaluations [-]
Static - Architecture 1	36,549	4,300
Static - Architecture 2	101,264	9,100
Static - Architecture 3	71,498	7,500
Static - Architecture 4	168,545	17,500
Static - Architecture 5	81,385	8,200
Static - Architecture 6	129,656	12,100
Static - Architecture 7	111,511	10,400
Static - Architecture 8	175,080	16,400
Static - Total	875,488	85,500
Dynamic	262,949	10,000



(a) Pareto front for each of the eight subproblems of the VSDS Goldstein function, evaluated using 8 static workflows



(b) Pareto front from Valencia-Ibáñez [21] of the VSDS Goldstein function. Objective 1 represents f and Objective 2 f_b

Fig. 18. Verification of the results obtained for the VSDS Goldstein function.

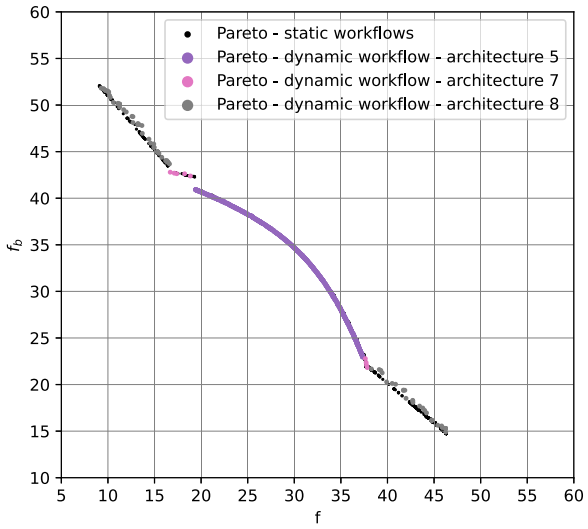


Fig. 19. Pareto front for the static and dynamic workflow solving the VSDS Goldstein function. Note that the plotting of Architecture 5 completely overlaps with a part of the Pareto front of the static workflows.

5.1.2. Performance verification of dynamic MDAO workflow

To objectively assess the quality of the Pareto front obtained using the dynamic workflow, the normalized hypervolume metric was used. A hypervolume indicator [57] is calculated by computing the volume of the Pareto front with respect to a reference point in the n dimensions of the respective front. In this case, our reference point is $f = 60, f_b = 60$, identical to the point used by Valencia-Ibáñez [21]. Normalization is achieved by dividing the computed hypervolume by that of a reference Pareto front. In this case, we divided the hypervolume of the dynamic workflow by that of the static workflows. This provides a dimensionless measure of performance relative to the best-known result. The execution of the VSDS Goldstein problem using the dynamic workflow was repeated ten times for different random seeds. This time, the number of optimization iterations was limited to 100 (equivalent to 10,000 function evaluations). In all ten cases, the dynamic workflow was able to find all Pareto optimal architectures. The results of the normalized hypervolume evaluation versus the number of optimization iterations are shown in Fig. 21. Important to note here is that the quality of the Pareto fronts depends mainly on the optimization algorithm and the settings used within this algorithm; however, the dynamic workflows play an important role as they facilitate the evaluation of different architectures within a single optimization workflow. For reference, the 95% confidence intervals evaluated at 5-10-50-100 iterations are compared in Fig. 21 with those obtained by the global exploration approach used by Valencia-Ibáñez [21], where a single optimizer was used with design vector imputation and a single disciplinary tool capable of evaluating all eight architectures.

The results obtained using the dynamic workflows show a similar trend as those from Valencia-Ibáñez [21], in the way the normalized hypervolume converges with the number of iterations, thereby verifying the correct implementation of the dynamic workflow approach.

5.2. Single-objective dynamic workflow results

To further assess the performance of dynamic workflows, also on other types of optimization, the following section details the capabilities and performance quantification of dynamic workflows for a single-objective optimization problem. A comparative study between the static and dynamic workflows is performed on the single-objective VSDS Goldstein function to highlight the performance improvement that the dynamic workflow approach offers, also for single-objective problems. In fact, for the dynamic workflow formulation, it does not matter whether a single or multi-objective problem is being solved.

The optimization settings used for the single objective optimization are shown in Table 4. Similar to the multi-objective optimization, the settings for the weighting factor and inverse crossover probability were determined through a series of tests. The population size was set to ten times the number of design variables. This resulted in a population size of 60 for subproblems 1 to 4, 70 for subproblems 5 to 8, and 110 for the dynamic workflow. The dynamic workflow was executed ten times with different random seeds, resulting in different initial populations. To ensure a fair comparison between static and dynamic workflows, the static workflows were also executed ten times, with the same initial populations as the dynamic workflows.

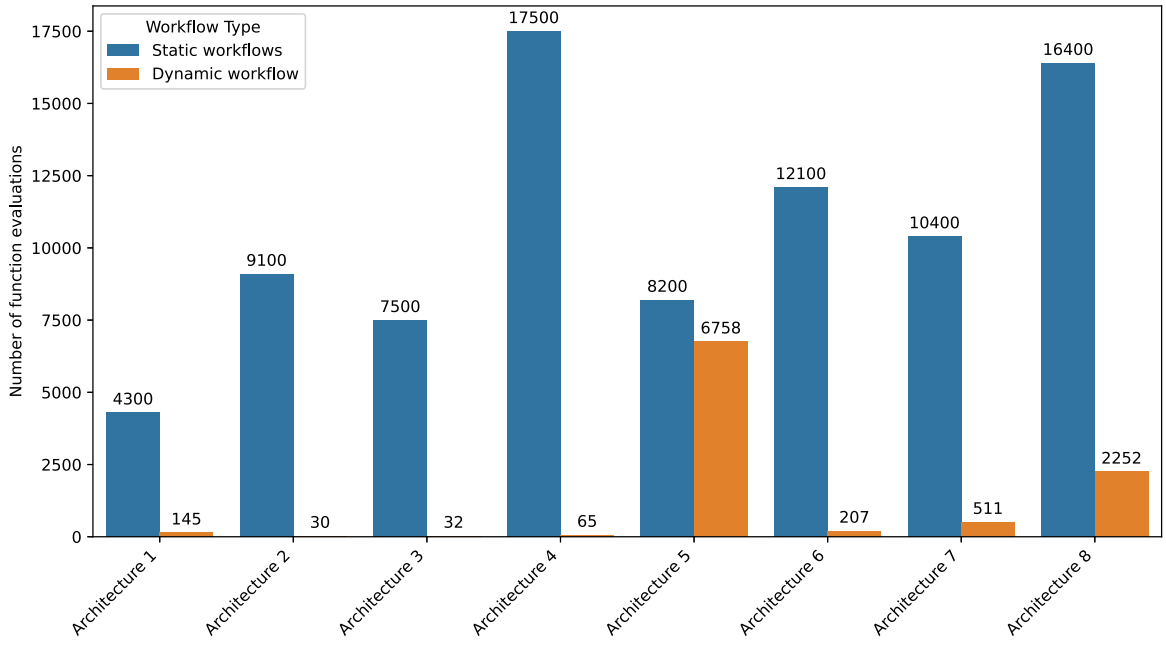


Fig. 20. Comparison of the number of function evaluations per architecture for the static and dynamic workflows.

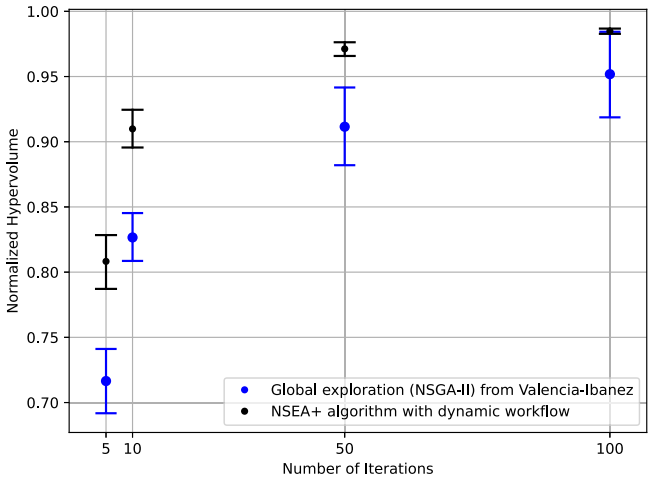


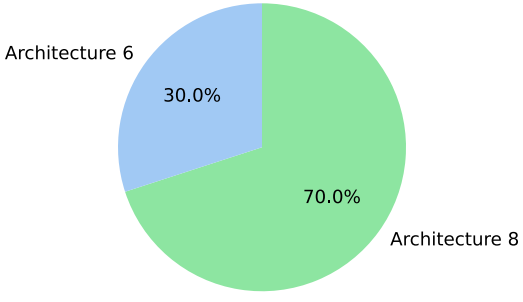
Fig. 21. 95% confidence interval for the normalized hypervolume for the Pareto fronts of the VSDS Goldstein function obtained for different numbers of iterations, compared to the results of Valencia-Ibáñez [21]. The execution of the optimization problem has been repeated ten times with different random seeds, hence the confidence interval.

The results are shown in Fig. 22. The pie chart in Fig. 22(a) indicates how often each architecture was found to be optimal by the dynamic workflow. Fig. 22(b) shows the 95% confidence interval of the optimal objective value for each of the static workflows and the dynamic workflow. Based on these results, it can be concluded that the dynamic workflow identified the global optimum in 7 out of 10 runs. It was verified that the other three times, the dynamic workflow converged to a local optimum associated with the second-best architecture (i.e. architecture 6), rather than reaching the global optimum associated with architecture 8. This is confirmed by the larger confidence interval achieved by the dynamic workflow compared to the one from the static optimization of architecture 8, as shown in Fig. 22(b).

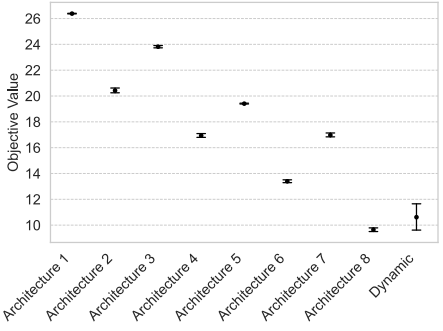
Fig. 23 provides a comparative analysis between the dynamic workflow and the combined static workflows in terms of three key performance metrics: number of iterations, number of function evaluations, and overall execution time. By combined static workflows we refer to the cumulative execution of individual static workflows for each architecture. In this approach, the optimization is performed separately for each possible architecture configuration, and the resulting performance metrics, such as the number of iterations, function evaluations, and total execution time, are aggregated. These summed values are then compared against the corresponding metrics from the dynamic workflow, which explores all architectures within a single optimization process. This comparison shows that the dynamic workflow approach required on average 81% fewer iterations and 68% fewer function evaluations compared to the combined static workflows. Note that the population size for the dynamic workflows needed to be higher than for the static workflows due to the number of design variables, resulting in different percentages for the number of iterations and function evaluations.

In addition to the reduced number of iterations and function evaluations, the dynamic workflows also achieved a shorter execution time compared to the static workflows. On average, the dynamic workflows completed the optimization 32% faster than the combined static workflows. This smaller time reduction, when compared with the large decrease in number of iterations and function evaluations, is attributed to the extra overhead introduced by the OptInOpt interface in Optimus. While the static workflows had an average execution time of 9.75s per function evaluation, the dynamic workflow took 20.5s per function evaluation. This extra overhead is due to the specific implementation in Optimus and may not be representative of other PIDO tools. Moving from Windows to Linux would for example already lower this overhead [42]. Although the time per function evaluation is notably higher for the dynamic workflows, the significantly lower number of function evaluations results in an overall time saving of 32%. These results are in line with previous research that applied dynamic workflows to design a wing rib accounting for multiple production methods [38]. The results from that study showed a computational time reduction of almost 14% using a dynamic workflow compared to its equivalent static workflows.

Although the methodology has been applied here on a mathematical use case, it is scalable and generic. Hence, it can be applied to any complex system or MDAO problem that requires changing design variables, analysis tools, or constraints during execution of the workflow. For reference, the application of the methodology to an aerospace-relevant use case focusing on the design and production of an aircraft wing rib is shown in Appendix C.

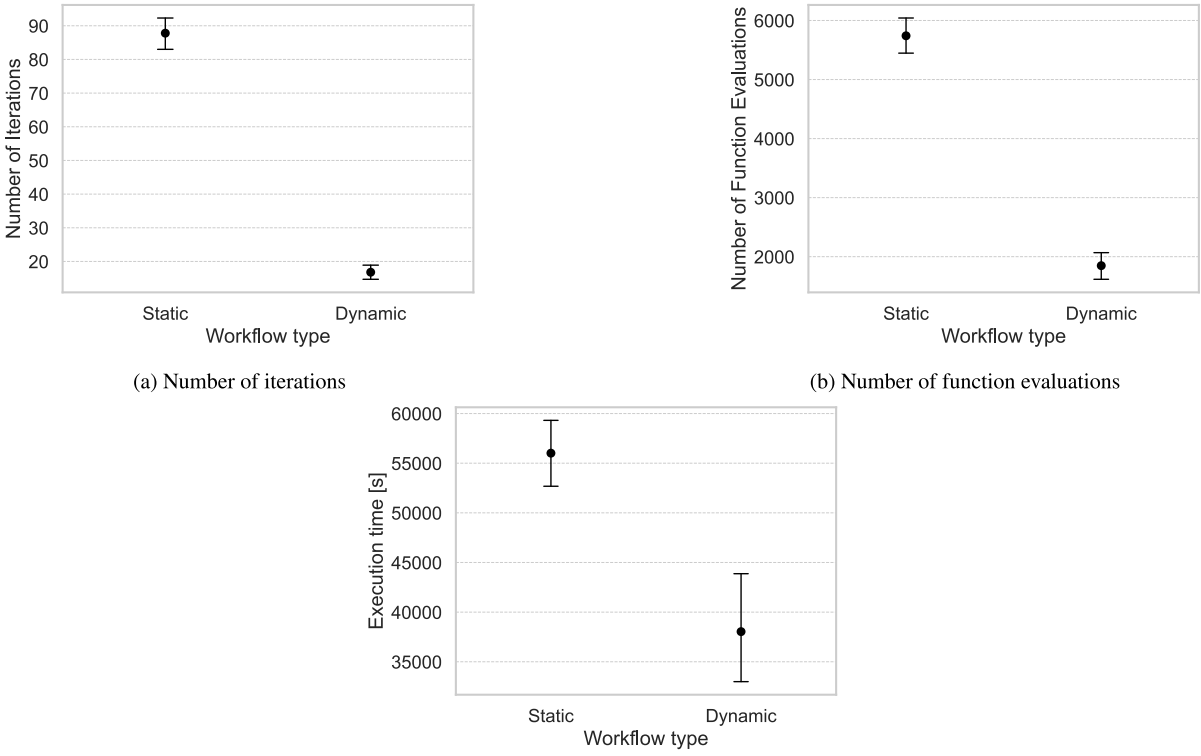


(a) Frequency at which each architecture was identified as the optimum by the dynamic workflow



(b) Comparison of the 95% confidence intervals (n=10) for the optimum values found using 8 individual static workflows and one dynamic workflow

Fig. 22. Comparison of the optimal points found by the static workflows and dynamic workflows. Each (sub)workflow has been executed ten times for different random seeds. In 3 out of 10 cases, the dynamic workflow got stuck in a local optimum (architecture 6), hence the larger confidence interval in figure b.



(a) Number of iterations

(b) Number of function evaluations

(c) Execution time

Fig. 23. Comparison of the static and dynamic workflows with a 95% confidence interval (n = 10).

6. Conclusions

Within the aircraft conceptual design phase, many design options need to be quickly evaluated to find the best fit for the given requirements. MDAO can be used to evaluate and optimize a design accounting for many different disciplinary perspectives. However, current MDAO methods provide limited support for evaluating different architectural designs within a single workflow, restricting the scope of trade-offs. Since different architectural options involve varying design variables, disciplinary tools, and constraints, designers are forced to set up and solve a multitude of MDAO workflows, with obvious consequences in setup and computation time. To overcome this limitation, a new type of MDAO workflow is proposed, called dynamic workflows, where the optimizer is enabled to switch between different system architectures during an optimization run, adapting design variables, constraints, and tools at each iteration based on the current architecture being evaluated.

This paper proposed a new methodology and tools to formalize, formulate, and execute dynamic workflows. To enable their formalization, the concepts of switches, branches, and subworkflows are introduced. Together, they allow for a complete and sound mathematical problem definition, consistent formalization, and automated integration into an executable simulation workflow. Additionally, an extension of the XDSM is proposed that includes new graphical elements to visualize switches and branches with a consistent mathematical definition. This enables the user to visually inspect the impact of architectural design choices on the different execution modes of the same workflow. Furthermore, with this new methodology, a nested approach is not necessarily required to solve system architecture optimization problems as the methodology enables the optimizer to make both architectural design decisions as well as optimize the architecture itself. Contrary to on-the-fly-generated workflows, the methodology in this paper provides a clear and complete problem formulation, providing the traceability and transparency that is required for the certifiability of the design process in industry applications.

A key limitation of the proposed methodology is that system-level gradient information is generally unavailable due to the presence of discrete architectural decisions and dynamically changing model structures. However, at a sub-system level (e.g. subworkflows), gradient-based methods can be applied to speed up the optimization process. A second limitation is the increased complexity of the workflows which can increase the setup time. Therefore, the integration of the dynamic workflows into an automated formulation and execution process has been shown, which can reduce the required setup time. Finally, usage of dynamic workflows relies on the PIDO tool of choice to include a switch or switch-like workflow component, which might not be available in all software packages.

Concerning the formulation and storage of dynamic workflows, the advances described in this paper are based on the extension of tools and data schema developed in previous work, namely KADMOS and CMDOWS. In their new release, they can fully automate the formulation process of dynamic workflows and store the results in a standard format to enable their translation into executable workflows, using a PIDO tool of choice. In this study, the target execution environment is Optimus for which a dedicated CMDOWS interpreter was developed.

The methodology was verified and validated using a multi-objective mathematical optimization problem that has previously been used in the literature as a benchmark case. The results demonstrated that the dynamic workflow is capable of identifying a Pareto front that consists of multiple architectural design options. Furthermore, it achieved this with a 70% reduction in execution time compared to the traditional approach based on solving multiple static workflows. This efficiency gain

is attributed to the optimizer's ability to steer the search toward more promising architectures, thereby reducing the time spent evaluating sub-optimal design alternatives.

The dynamic workflow was also applied to a single objective version of the same benchmark problem. The optimizer converged to the global optimum 70% of the time and delivered significant performance improvements. On average, it achieved an 81% reduction in the number of iterations, a 68% reduction in number of function evaluations, and a 32% reduction in total execution time compared to the equivalent static workflows, showing its ability to find the best architecture and optimize it in significantly less time and fewer function evaluations than optimizing each architecture separately. Although in this paper we discussed the application to a mathematical use case, the proposed methodology is fully generic and scalable, so that it can be applied to any complex system or MDAO problem that requires changing design variables, analysis tools, or constraints during the execution of the workflow. An aerospace-relevant application of the proposed methodology, focusing on the design and production of an aircraft wing rib, is provided in C.

Future work will focus on the application of the dynamic workflows to a wider variety of case studies, including increased numbers of architectural design choices. While the number of required static workflows grows combinatorially as the number of architectural options increases, the proposed dynamic workflow approach is expected to deliver even more significant time savings and efficiency benefits, enabling architecture design space studies not possible before.

CRediT authorship contribution statement

Anne-Liza M.R.M. Bruggeman: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Gianfranco La Rocca:** Writing – review & editing, Writing – original draft, Visualization, Supervision, Resources, Project administration, Methodology, Funding acquisition, Conceptualization; **Maurice F.M. Hoogreef:** Writing – review & editing, Writing – original draft, Visualization, Methodology, Conceptualization.

Data availability

The Optimus results of the multi- and single-objective VSDS Goldstein problem using dynamic and static workflows are available at https://doi.org/10.4121/e5ac7f75-50ee-439d-a2ad-2da341560969. The scripts to formulate the dynamic and static workflows can be found on the KADMOS repository in the feature/dynamic_workflow_paper branch. The results have been generated using KADMOS v0.10, (https://gitlab.tudelft.nl/lr-fpp-mdo/kadmos), CMDOWS v0.10, (https://gitlab.tudelft.nl/lr-fpp-mdo/cmdows), and Optimus 2021.1SP2 (https://www.noesissolutions.com/our-products/optimus). The CMDOWS-Optimus importer is a proprietary tool from TU Delft and Noesis Solutions and is therefore not publicly accessible.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The research presented in this paper has partially been performed in the framework of the DEFAINE (Design Exploration Framework based on AI for froNt-loaded Engineering) project and has received funding from the European Union program ITEA 3 Call 6 project 19009.

Appendix A. Variable-size design space Goldstein function problem formulation

A full formulation of the VSDS Goldstein function problem was given in set of Eqs. (7) through (9) in Section 5. This appendix describes the mathematical equations of the eight objective equations ($f_1, \dots, f_8$) and four constraint equations ($g_1, \dots, g_4$). The input variables used are the five continuous design variables ($x_1, \dots, x_5$) and four discrete design variables ($z_1, \dots, z_4$). c_1 and c_2 are two constants whose values depend on the design variables.

A.1. General functions

To prevent the repetition of long equations, some general functions A, B and C are defined. These functions will be referred to when describing each of the objective and constraint equations.

$$\begin{aligned} A(x_1, x_2, x_3, x_4, z_3, z_4) =& 53.3108 + 0.184901x_1 - 5.02914x_1^3 \cdot 10^{-6} \\ &+ 7.72522x_1^{z_3} \cdot 10^{-8} - 0.0870775x_2 \\ &- 0.106959x_3 + 7.98772x_3^{z_4} \cdot 10^{-6} \\ &+ 0.00242482x_4 + 1.32851x_4^3 \cdot 10^{-6} \\ &- 0.00146393x_1x_2 - 0.00301588x_1x_3 \\ &- 0.00272291x_1x_4 + 0.0017004x_2x_3 \\ &+ 0.0038428x_2x_4 - 0.000198969x_3x_4 \\ &+ 1.86025x_1x_2x_3 \cdot 10^{-5} - 1.88719x_1x_2x_4 \cdot 10^{-6} \\ &+ 2.50923x_1x_3x_4 \cdot 10^{-5} - 5.62199x_2x_3x_4 \cdot 10^{-5} \end{aligned} \tag{A.1}$$

$$B(x_1, x_2, x_3, x_4, x_5, z_3, z_4) = A(x_1, x_2, x_3, x_4, z_3, z_4) + 5\cos\left(2\pi \frac{x_5}{100}\right) - 2 \tag{A.2}$$

$$C(x_1, x_2, c_1, c_2) = -\left(x_1 - 50\right)^2 - \left(x_2 - 50\right)^2 + \left(20 + c_1 * c_2\right)^2 \tag{A.3}$$

A.2. Objective equations

The eight objectives functions are defined as follows:

$$f_1(x_1, x_2, z_1, z_2, z_3, z_4) = A(x_1, x_2, x_3(z_1), x_4(z_2), z_3, z_4) \tag{A.4}$$

where x_3 and x_4 are defined as a function of z_1 and z_2 according to the relations defined in Tables A.1 and A.2, respectively.

Table A.1
Value of x_3 as a function of z_1.

z_1	x_3
0	20
1	50
2	80

Table A.2
Value of x_4 as a function of z_2.

z_2	x_4
0	20
1	50
2	80

Table A.3
Value of c_1 as a function of z_1.

z_1	c_1
0	3
1	2
2	1

Table A.4
Value of c_2 as a function of z_2.

z_2	c_2
0	0.5
1	-1
2	-2

Table A.5
Value of c_1 as a function of z_3.

z_3	c_1
0	3
1	2
2	1

Table A.6
Value of c_2 as a function of z_4.

z_4	c_2
0	0.5
1	-1
2	-2

$$f_2(x_1, x_2, x_3, z_2, z_3, z_4) = A(x_1, x_2, x_3, x_4(z_2), z_3, z_4) \tag{A.5}$$

where x_4 is defined as a function of z_2 according to the relations defined in Table A.2.

$$f_3(x_1, x_2, x_4, z_1, z_3, z_4) = A(x_1, x_2, x_3(z_1), x_4, z_3, z_4) \tag{A.6}$$

where x_3 is defined as a function of z_1 according to the relations defined in Table A.1.

$$f_4(x_1, x_2, x_3, x_4, z_3, z_4) = A(x_1, x_2, x_3, x_4, z_3, z_4) \tag{A.7}$$

$$f_5(x_1, x_2, x_5, z_1, z_2, z_3, z_4) = B(x_1, x_2, x_3(z_1), x_4(z_2), x_5, z_3, z_4) \tag{A.8}$$

where x_3 and x_4 are defined as a function of z_1 and z_2 according to the relations defined in Tables A.1 and A.2, respectively.

$$f_6(x_1, x_2, x_3, x_5, z_2, z_3, z_4) = B(x_1, x_2, x_3, x_4(z_2), x_5, z_3, z_4) \tag{A.9}$$

where x_4 is defined as a function of z_2 according to the relations defined in Table A.2.

$$f_7(x_1, x_2, x_4, x_5, z_1, z_3, z_4) = B(x_1, x_2, x_3(z_1), x_4, x_5, z_3, z_4) \tag{A.10}$$

where x_3 is defined as a function of z_1 according to the relations defined in Table A.1.

$$f_8(x_1, x_2, x_3, x_4, x_5, z_3, z_4) = B(x_1, x_2, x_3, x_4, x_5, z_3, z_4) \tag{A.11}$$

A.3. Constraint equations

The four constraint equations are defined as follows:

$$g_1(x_1, x_2, z_1, z_2) = C(x_1, x_2, c_1(z_1), c_2(z_2)) \tag{A.12}$$

where c_1 and c_2 are defined as a function of z_1 and z_2 according to the relations defined in Tables A.3 and A.4, respectively.

$$g_2(x_1, x_2, z_2) = C(x_1, x_2, c_1, c_2(z_2)) \tag{A.13}$$

where $c_1 = 0.5$ and c_2 is defined as a function of z_2 according to the relations defined in Table A.4.

$$g_3(x_1, x_2, z_1) = C(x_1, x_2, c_1(z_1), c_2) \tag{A.14}$$

where $c_2 = 0.7$ and c_1 is defined as a function of z_1 according to the relations defined in Table A.3.

$$g_4(x_1, x_2, z_3, z_4) = C(x_1, x_2, c_1(z_3), c_2(z_4) \tag{A.15}$$

where c_1 and c_2 are defined as a function of z_3 and z_4 according to the relations defined in Tables A.5 and A.6, respectively.

Appendix B. Example calculation of the VSDS Goldstein function

This appendix shows an example calculation for objective equation f_8. The design point used is defined in Table B.1.

$$\begin{aligned} f_8(x_1, x_2, x_3, x_4, x_5, z_3, z_4) =& 53.3108 + 0.184901x_1 - 5.02914x_1^3 \cdot 10^{-6} \\ &+ 7.72522x_1^{z_3} \cdot 10^{-8} - 0.0870775x_2 \\ &- 0.106959x_3 + 7.98772x_3^{z_4} \cdot 10^{-6} \\ &+ 0.00242482x_4 + 1.32851x_4^3 \cdot 10^{-6} \\ &- 0.00146393x_1x_2 - 0.00301588x_1x_3 \\ &- 0.00272291x_1x_4 + 0.0017004x_2x_3 \\ &+ 0.0038428x_2x_4 - 0.000198969x_3x_4 \\ &+ 1.86025x_1x_2x_3 \cdot 10^{-5} - 1.88719x_1x_2x_4 \\ &\cdot 10^{-6} \\ &+ 2.50923x_1x_3x_4 \cdot 10^{-5} - 5.62199x_2x_3x_4 \\ &\cdot 10^{-5} \\ &+ 5\cos\left(2\pi\frac{x_5}{100}\right) - 2 \end{aligned} \tag{B.1}$$

$$\begin{aligned} f_8(100, 100, 100, 100, 50, 0, 0) =& 53.3108 + 0.184901 \cdot 100 - 5.02914 \cdot 100^3 \\ &\cdot 10^{-6} \\ &+ 7.72522 \cdot 100^0 \cdot 10^{-8} - 0.0870775 \cdot 100 \\ &- 0.106959 \cdot 100 + 7.98772 \cdot 100^0 \cdot 10^{-6} \\ &+ 0.00242482 \cdot 100 + 1.32851 \cdot 100^3 \cdot 10^{-6} \\ &- 0.00146393 \cdot 100 \cdot 100 - 0.00301588 \cdot 100 \\ &\cdot 100 \\ &- 0.00272291 \cdot 100 \cdot 100 + 0.0017004 \cdot 100 \\ &\cdot 100 \\ &+ 0.0038428 \cdot 100 \cdot 100 - 0.000198969 \cdot 100 \\ &\cdot 100 \\ &+ 1.86025 \cdot 100 \cdot 100 \cdot 100 \cdot 10^{-5} \\ &- 1.88719 \cdot 100 \cdot 100 \cdot 100 \cdot 10^{-6} \\ &+ 2.50923 \cdot 100 \cdot 100 \cdot 100 \cdot 10^{-5} \\ &- 5.62199 \cdot 100 \cdot 100 \cdot 100 \cdot 10^{-5} \end{aligned}$$

Table B.1
Values used for the different design variables.

Design variable	x_1	x_2	x_3	x_4	x_5	z_3	z_4	w_1	w_2
Value	100	100	100	100	50	0	0	3	1

$$
\begin{aligned}
&+ 5\cos\left(2\pi\frac{50}{100}\right) - 2 \qquad \text{(B.2)}\\
=&53.3108 + 18.4901 - 5.02914 + 7.72522\\
&\cdot 10^{-8}\\
&- 8.70775 - 10.6959 + 7.98772 \cdot 10^{-6}\\
&+ 0.242482 + 1.32851 - 14.6393 - 30.1588\\
&- 27.2291 + 17.004 + 38.428 - 1.98969\\
&+ 18.6025 - 1.88719 + 25.0923 - 56.2199\\
&- 5 - 2\\
=&8.94192
\end{aligned}
$$

Appendix C. Application of dynamic MDAO workflows to an aeronautical use case

To demonstrate the applicability of the methodology presented in this paper to an aeronautical use case, the design and manufacturing of an aircraft wing rib is presented [38]. When trading different production methods, different analyses need to be performed and method-specific constraints must be applied to ensure manufacturability of the rib. For example, in metal machining, elements such as the reachability and accessibility of the machining head to the product determine the product's manufacturability [58]. In contrast, for composite stamp-forming, such requirements are not relevant. Instead, the resulting weave angles after stamp-forming play a critical role in determining manufacturability.

Fig. C.1 shows how dynamic MDAO workflows can be applied to the integrated design and manufacturing optimization of the wing rib. In this workflow, the rib design is optimized while treating the production method as an architectural design variable. The considered architectural options are a metal machined rib and a composite stamp-formed rib. Depending on the production method selected by the optimizer, either the machining subworkflow or the stamp-forming subworkflow, shown in Fig. C.2(a) and (b), respectively, is activated. Both subworkflows contain a single analysis tool that evaluates the manufacturability of the rib for the selected production method. For machining, this analysis evaluates the accessibility rate, whereas for stamp-forming, the resulting weave angles are assessed. The outputs of these analyses are used to formulate constraints within the optimization loop and are therefore treated as branch-dependent constraints. If a branch is not executed, the corresponding constraint is automatically assigned a default value to ensure it is not violated. In this example, the only architecture variable is the production method, whose selection automatically drives the rib material. The example could be extended by adding the material as an architectural variable and adding stamp-forming workflows specific to metal structures. This would result in multiple switches in the main workflow. Using the methodology described in this paper, a single MDAO workflow can thus be formulated to evaluate multiple production methods and materials within one optimization problem, while consistently enforcing the manufacturability constraints specific to each method. For more details on the wing rib use case and the underlying analysis models, the reader is referred to Bruggeman et al. [38].

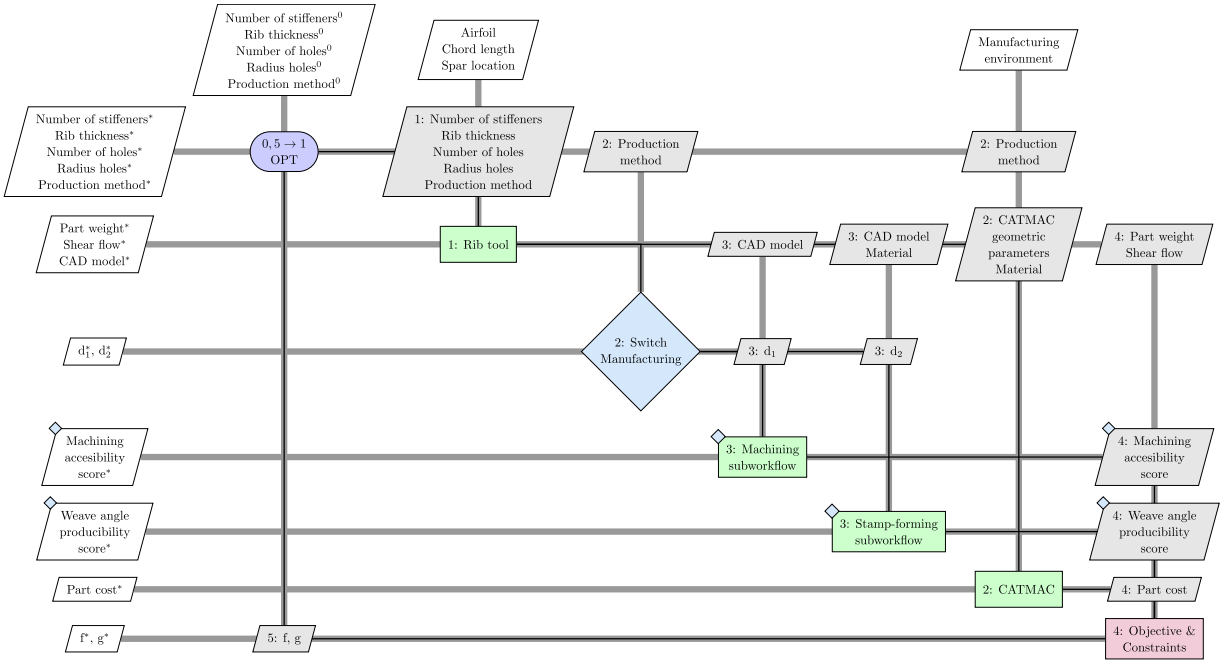


Fig. C.1. XDSM representation of the design and manufacturing of a wing rib (based on the work of Bruggeman et al. [38]).

(a) XDSM representation of the machining subworkflow

(b) XDSM representation of the stamp-forming subworkflow

Fig. C.2. XDSM representations of the production subworkflows in Fig. C.1 (based on the work of Bruggeman et al. [38]).

References

[1] E. Schwartz, I.M. Kroo, Aircraft design: trading cost and climate impact, in: 47th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, 2009, pp. 1–16. https://doi.org/10.2514/6.2009-1261

[2] A. Kuśmierek, C. Galiński, W. Stalewski, Review of the hybrid gas - electric aircraft propulsion systems versus alternative systems, Prog. Aerosp. Sci. 141 (2023) 1–30. https://doi.org/10.1016/j.paerosci.2023.100925

[3] K. Dahal, S. Brynolf, C. Xisto, J. Hansson, M. Grahn, T. Grönstedt, M. Lethveer, Techno-economic review of alternative fuels and propulsion systems for the aviation sector, Renew. Sustain. Energy Rev. 151 (2021) 1–15. https://doi.org/10.1016/j.rser.2021.111564

[4] P. Proesmans, R. Vos, Comparison of future aviation fuels to minimize the climate impact of commercial aircraft, in: AIAA Aviation 2022 Forum, 2022, pp. 1–25. https://doi.org/10.2514/6.2022-3288

[5] U.P. Breuer, Commercial Aircraft Composite Technology, Springer International Publishing Switzerland, 1 edition, 2016. https://doi.org/10.1007/978-3-319-31918-6

[6] T.M. Young, Aircraft design innovation: creating an environment for creativity, Proc. Inst. Mechan. Eng. Part G J. Aerosp. Eng. 221 (2) (2007) 165–174. https://doi.org/10.1243/09544100JAERO129

[7] AIAA, Technical Committee on Multidisciplinary Design Optimization (MDO), White Paper on Current State of the Art, AIAA, 1991. ISBN: 978-15-634-7021-9.

[8] I.M. Kroo, Innovations in aeronautics - 2004 AIAA dryden lecture, in: 42nd AIAA Aerospace Sciences Meeting and Exhibit, 2004, pp. 1–11. https://doi.org/10.2514/6.2004-1

[9] European Commission, Fly the Green Deal, Europe's Vision for Sustainable Aviation, Report of the Advisory Council for Aviation Research and Innovation in Europe (ACARE), Publications Office of the European Union, Luxembourg, 2022. https://doi.org/10.2777/732726

[10] J. Sobieszczanski-Sobieski, A. Morris, M. van Tooren, Multidisciplinary Design Optimization supported by Knowledge Based Engineering, Wiley, 1 edition, 2015. https://doi.org/10.1002/9781118897072

[11] F. Flager, J. Haymaker, A comparison of multidisciplinary design, analysis and optimization processes in the building construction and aerospace industries, in: 24th International Conference on Information Technology in Construction, 2007, pp. 625–630. http://itc.scix.net/paper/w78_2007_31.

[12] K.G. Bowcutt, A perspective on the future of aerospace vehicle design, in: 12th AIAA International Space Planes and Hypersonic Systems and Technologies, 2003, pp. 1–10. https://doi.org/10.2514/6.2003-6957

[13] A. Gazaix, F. Gallard, V. Ambert, D. Guénot, M. Hamadi, P. Sarouille, S. Grihon, T. Druot, J. Brézillon, T. Lefebvre, N. Bartoli, R. Lafage, V. Gachelin, J. Plakoo, N. Desfachelles, S. Gurol, B. Pauwels, C. Vanaret, Industrial application of an advanced bi-level MDO formulation to aircraft engine pylon optimization, in: AIAA Aviation 2019 Forum, 2019, pp. 1–28. https://doi.org/10.2514/6.2019-3109

[14] ISO/IEC/IEEE, ISO/IEC/IEEE 24765 - Systems and software engineering - Vocabulary, Standard, International Organization for Standardization & Institue of Electrical and Electronics Engineers, Geneva, CH & New York, USA, 2017. https://doi.org/10.1109/IEEESTD.2017.8016712

[15] C. De Tenorio, M. Armstrong, D. Mavris, Architecture subsystem sizing and coordinated optimization methods, in: 47th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, 2009, pp. 1–11. https://doi.org/10.2514/6.2009-37

[16] C.P. Frank, A design space exploration methodology to support decisions under evolving uncertainty in requirements and its application to advanced vehicles, Ph.D. thesis, Georgia Institute of Technology, 2016. https://doi.org/http://hdl.handle.net/1853/55628

[17] F. Villeneuve, D.N. Mavris, A new method of architecture selection for launch vehicles, in: AIAA/CIRA 13th International Space Plane and Hypersonics Systems and Technologies, 2005, pp. 1–17. https://doi.org/10.2514/6.2005-3361

[18] M.A. Walton, D.E. Hastings, Applications of uncertainty analysis to architecture selection of satellite systems, J. Spacecr. Rockets 41 (1) (2004) 75–84. https://doi.org/10.2514/1.9210

[19] R. Bornholdt, T. Kreitz, F. Thielecke, Function-driven design and evaluation of innovative flight controls and power system architectures, in: SAE Technical Paper 2015-01-2482, 2015, pp. 1–17. https://doi.org/10.4271/2015-01-2482

[20] M. LaSorda, J.M. Borky, R.M. Sega, Model-based architecture and programmatic optimization for satellite system-of-systems architectures, Syst. Eng. 21 (4) (2018) 372–387. https://doi.org/10.1002/sys.21444

[21] S. Valencia-Ibáñez, Optimization Strategies for System Architecting Problems, Master's thesis, Delft University of Technology, 2023. https://resolver.tudelft.nl/uuid:8ffb95af-3394-4bb2-9636-8c52a59bdded.

[22] J.H. Bussemaker, P. Saves, N. Bartoli, T. Lefebvre, R. Lafage, System architecture optimization strategies: dealing with expensive hierarchical problems, J. Global Optim. 91 (2025) 851–895. https://doi.org/10.1007/s10898-024-01443-8

[23] S. Valencia-Ibáñez, Optimization strategies for system architecting problems, in: AIAA SciTech 2025 Forum, 2025, pp. 1–11. https://doi.org/10.2514/6.2025-2847

[24] M. Guerster, E.F. Crawley, Architectural options and optimization of suborbital space tourism vehicles, in: 2018 IEEE Aerospace Conference, 2018, pp. 1–18. https://doi.org/10.1109/AERO.2018.8396775

[25] C.P. Frank, R.A. Marlier, O.J. Pinon-Fischer, D.N. Mavris, An evolutionary multi-architecture multi-objective optimization algorithm for design space exploration, in: 57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 2016, pp. 1–19. https://doi.org/10.2514/6.2016-0414

[26] M. Armstrong, C. De Tenorio, E. Garcia, D. Mavris, Function based architecture design space definition and exploration, in: The 26th Congress of International Council of the Aeronautical Sciences, 2008, pp. 1–13. https://doi.org/10.2514/6.2008-8928

[27] D. Raudberget, P. Edholm, M. Andersson, Implementing the principles of set-based concurrent engineering in configurable component platforms, in: NordDesign 2012, 2012, pp. 1–8. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A548337.

[28] N. Albarello, J.B. Welcomme, C. Reyterou, A formal design synthesis and optimization method for systems architectures, in: 9th International Conference of Modeling, Optimization and Simulation, 2012, pp. 1–8. https://hal.science/hal-00728577v1.

[29] J. Bussemaker, R. Garcia Sánchez, M. Fouda, L. Boggero, B. Nagel, Function-based architecture optimization: an application to hybrid-electric propulsion systems, 33rd Annual INCOSE International Symposium 33 (1) (2023) 251–272. https://doi.org/10.1002/iis2.13020

[30] D.M. Judt, C. Lawson, Development of an automated aircraft subsystem architecture generation and analysis tool, Eng. Comput. 33 (5) (2015) 1327–1352. https://doi.org/10.1108/EC-02-2014-0033

[31] H. Broodney, D. Dotan, L. Greenberg, M. Masin, Generic approach for systems design optimization in MBSE, INCOSE International Symposium 22 (1) (2012) 184–200. https://doi.org/10.1002/j.2334-5837.2012.tb01330.x

[32] Y. Bile, A. Riaz, M.D. Guenov, A. Molina-Cristóbal, Towards automating the sizing process in conceptual (Airframe) systems architecting, in: 2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 2018, pp. 1–24. https://doi.org/10.2514/6.2018-1067

[33] G. Belie, Non-technical barriers to multidisciplinary optimization in the aerospace industry, in: 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, 2002, pp. 1–6. https://doi.org/10.2514/6.2002-5439

[34] J.H. Bussemaker, T. De Smedt, G. La Rocca, P.D. Ciampa, B. Nagel, System architecture optimization: an open source multidisciplinary aircraft jet engine architecting problem, in: AIAA Aviation 2021 Forum, 2021, pp. 1–20. https://doi.org/10.2514/6.2021-3078

[35] A.M.R.M. Bruggeman, G. La Rocca, From requirements to product: an MBSE approach for the digitalization of the aircraft design process, INCOSE International Symposium 33 (1) (2023) 1688–1706. https://doi.org/10.1002/iis2.13107

[36] J.S. Sonneveld, T. van den Berg, G. la Rocca, S. Valencia-Ibáñez, B. van Manen, A.M.R.M. Bruggeman, B. Beijer, Dynamic workflow generation applied to aircraft moveable architecture optimization, in: Aerospace Europe Conference 2023 - 10th EUCASS - 9th CEAS, 2023, pp. 1–16. https://doi.org/10.13009/EUCASS2023-544

[37] S. Garg, R.G. Sánchez, J.H. Bussemaker, L. Boggero, B. Nagel, Dynamic formulation and execution of MDAO workflows for architecture optimization, in: AIAA Aviation Forum and Ascend 2024, 2024, pp. 1–18. https://doi.org/10.2514/6.2024-4402

[38] A.M.R.M. Bruggeman, M. Nikitin, G. La Rocca, O.K. Bergsma, Model-Based approach for the simultaneous design of airframe components and their production process using dynamic MDAO workflows, in: AIAA Scitech 2024 Forum, 2024, pp. 1–17. https://doi.org/10.2514/6.2024-1530

[39] A.B. Lambe, J.R.R.A. Martins, Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes, Struct. Multidiscip. Optim. 46 (2012) 273–284. https://doi.org/10.1007/s00158-012-0763-y

[40] I. Van Gent, G. La Rocca, M.F.M. Hoogreef, CMDOWS: a proposed new standard to store and exchange MDO systems, CEAS Aeronaut. J. 9 (4) (2018) 607–627. https://doi.org/10.1007/s13272-018-0307-2

[41] J.R.R.A. Martins, A.B. Lambe, Multidisciplinary design optimization: a survey of architectures, AIAA J. 51 (9) (2013) 2049–2075. https://doi.org/10.2514/1.J051895

[42] M.F.M. Hoogreef, Advise, Formalize and Integrate MDO Architectures - A Methodology and Implementation, Ph.D. thesis, Delft University of Technology, 2017. https://doi.org/10.4233/uuid:cc2af611-6d78-4439-9b10-7e62ae579029

[43] F. Gallard, C. Vanaret, D. GuÃ©not, V. Gachelin, R. Lafage, B. Pauwels, P.-J. Barjhoux, A. Gazaix, GEMS: a python library for automation of multidisciplinary design optimization process generation, in: 2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 2018, pp. 1–26. https://doi.org/10.2514/6.2018-0657

[44] I. Van Gent, G. La Rocca, Formulation and integration of MDAO systems for collaborative design: a graph-based methodological approach, Aerosp. Sci. Technol. 90 (2019) 410–433. https://doi.org/10.1016/j.ast.2019.04.039

[45] A. Page Risueño, J.H. Bussemaker, P.D. Ciampa, B. Nagel, MDAx: agile generation of collaborative MDAO workflows for complex systems, in: AIAA Aviation 2020 Forum, 2020, pp. 1–26. https://doi.org/10.2514/6.2020-3133

[46] A.R. Kulkarni, M.F.M. Hoogreef, G. La Rocca, Combining semantic web technologies and KBE to solve industrial MDO problems, in: 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2017, pp. 1–18. https://doi.org/10.2514/6.2017-3823

[47] P.D. Ciampa, P.S. Prakasha, F. Torrigiani, J. Walther, T. Lefebvre, N. Bartoli, H. Timmermans, P. Della Vecchia, L. Stingo, D. Rajpal, I. Van Gent, G. La Rocca, M. Fioriti, G. Cerino, R. Maierl, D. Charbonnier, A. Jungo, B. Aigner, K. Anisimov, A. Mirzoyan, M. Voskuijl, Streamlining cross-organizational aircraft development: results from the AGILE project, in: AIAA Aviation 2019 Forum, 2019, pp. 1–45. https://doi.org/10.2514/6.2019-3454

[48] J.S. Gray, J.T. Hwang, J.R.R.A. Martins, K.T. Moore, B.A. Naylor, OpenMDAO: an open-source framework for multidisciplinary design, analysis, and optimization, Struct. Multidiscip. Optim. 59 (4) (2019) 1075–1104. https://doi.org/10.1007/s00158-019-02211-z

[49] I. Van Gent, AGILE MDAO Systems: A Graph-based Methodology to Enhance Collaborative Multidisciplinary Design, Ph.D. thesis, Delft University of Technology, 2019. https://doi.org/10.4233/uuid:c42b30ba-2ba7-4fff-bf1c-f81f85e890af

[50] M.F.M. Hoogreef, R. d'Ippolito, R. Augustinus, G. La Rocca, A multidisciplinary design optimization advisory system for aircraft design, in: 5th CEAS Air and Space Conference, 2015, pp. 1–15. https://resolver.tudelft.nl/uuid:8beb86d6-40a7-4c03-bc18-8512bb5bb548.
[51] D.J. Pate, J. Gray, B.J. German, A graph theoretic approach to problem formulation for multidisciplinary design analysis and optimization, Struct. Multidiscip. Optim. 49 (5) (2014) 743–760. https://doi.org/10.1007/s00158-013-1006-6
[52] A.M.R.M. Bruggeman, Automated Execution Process Formulation using Sequencing and Decomposition Algorithms for Collaborative MDAO, Master's thesis, Delft University of Technology, 2019. https://resolver.tudelft.nl/uuid:7d402ca8-2f9e-41e4-abaa-ff0e600fbc14.
[53] I. van Gent, R. Lombardi, G. La Rocca, R. d'Ippolito, A fully automated chain from MDAO problem formulation to workflow execution, in: EUROGEN 2017, 2017, pp. 1–15. https://resolver.tudelft.nl/uuid:1b0841b5-f6a6-4750-8490-0e83e8347c10.
[54] D. de Vries, Towards the Industrialization of MDAO, Master's thesis, Delft University of Technology, 2017. https://resolver.tudelft.nl/uuid:ccc91e95-a790-4fbc-bcaf-effedc9dbd4d.
[55] I. van Gent, B. Aigner, B. Beijer, G. La Rocca, A critical look at design automation solutions for collaborative MDO in the AGILE paradigm, in: 2018 Multidisciplinary Analysis and Optimization Conference, 2018, pp. 1–23. https://doi.org/10.2514/6.2018-3251
[56] J. Pelamatti, L. Brevault, M. Balesdent, E. Talbi, Y. Guerin, Bayesian optimization of variable-size design space problems, Optimiz. Eng. 22 (2021) 387–447. https://doi.org/10.1007/s11081-020-09520-z
[57] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms–a comparative case study, in: International Conference on Parallel Problem Solving from Nature, 1998, pp. 292–301. https://doi.org/10.1007/BFb0056872
[58] M. Hoefer, N. Chen, M. Frank, Automated manufacturability analysis for conceptual design in new product development, in: Industrial and Systems Engineering Research Conference, 2017, pp. 1–6. https://dr.lib.iastate.edu/handle/20.500.12876/44271.

Software references

[59] RCE, 2025. https://rcenvironment.de, accessed on: 20-10-2025.
[60] KADMOS, 2025. https://gitlab.tudelft.nl/lr-fpp-mdo/kadmos, accessed on: 20-10-2025.
[61] CMDOWS, 2025. https://gitlab.tudelft.nl/lr-fpp-mdo/cmdows, accessed on: 20-10-2025.

Commercial references

[62] Optimus, 2025. https://www.noesissolutions.com/our-products/optimus, accessed on: 20-10-2025