MDAx: Agile Generation of Collaborative MDAO Workflows for Complex Systems

A. Page Risueño*, J.H. Bussemaker†, P.D. Ciampa‡, B. Nagel§
DLR (German Aerospace Center), Institute of System Architectures in Aeronautics, Hamburg, Germany

In this paper, a new application for collaborative Multidisciplinary Design Analysis and Optimization (MDAO) workflow modeling is presented. The MDAO Workflow Design Accelerator, short MDAx, enables workflow integrators and disciplinary experts to model, inspect, and explore workflow components and their relationships, and export workflow configurations for execution on integration platforms. The necessity for such an MDAO design environment stems from the inherent complexity in aircraft design, in which the segregation of disciplines on technical and managerial scale result in time intensive workflow integration efforts. In practice, it can be observed that the integration of simulation tools to solve real-life MDAO problems produces large, interconnected workflow systems that lead to a loss in oversight of the application network, lack in transparency due to the many participants, and consistency issues with the resulting models. To facilitate a more effective collaboration among disciplinary experts, MDAx provides an intuitive workflow modeling environment using an expansion of the XDSM (eXtended Design Structure Matrix) format with additional design rules. Various functionalities to automate repetitive design tasks to resolve ambiguities and inconsistencies in complex workflows are provided. Importance is given to fearless workflow design through continuous feedback and user guidance without requiring expert knowledge in MDAO architecting, which shows considerable effects on the removal of barriers in the adoption of existing MDAO paradigms in collaborative teams. This paper introduces MDAx, its founding methodology and implementation, its user interface and effects on usability, and a case study demonstrating its impact on the coordination and communication among collaborators in a realistic design problem.

Nomenclature

ADO	=	Architecture and Design Optimization
CDS	=	Central Data Schema
I/O	=	Input/Output
$MDAO$	=	Multidisciplinary Design Analysis and Optimization
$MDAx$	=	MDAO Workflow Design Accelerator
OOP	=	Object-Oriented Programming
SoS	=	System of Systems
$SSOT$	=	Single Source of Truth
UI	=	User Interface
$XDSM$	=	eXtended Design Structure Matrix

I. Introduction

In spite of a spread of Multidisciplinary Design Analysis and Optimization (MDAO) and its principles across the aviation industry, considerable barriers still remain that prevent a deeper adoption of MDAO among engineering teams. While the application of MDAO in an aircraft design campaign is an intricate endeavor due to its technical complexity,

*Researcher, MDO group, Aircraft Design & System Integration, Hamburg, andreas.pagerisueno@dlr.de
†Researcher, MDO group, Aircraft Design & System Integration, Hamburg, jasper.bussemaker@dlr.de
‡Team lead MDO group, Aircraft Design & System Integration, Hamburg, pier-davide.ciampa@dlr.de, AIAA MDO TC member
§Head of institute, Institute of System Architectures in Aeronautics, Hamburg, bjoern.nagel@dlr.de

the barriers to a more widespread application of MDAO are not exclusively of technical, but also of organizational nature [1]; it is even suggested that the existing technical barriers are far easier to solve that organizational and cultural barriers [2]. Non-technical barriers such as the resistance to change, "black box" perception or absence of insight into the inner workings, and a lack of confidence in the approach prevent a genuine application of MDAO methodologies in industry [3].

Coupling disciplines in an MDAO problem formulation poses a difficult task not only due to the fact that those disciplines generally have become more numerous and complex with the increase in computational power in the past years and decades, but especially due to the lack of organizational integration of disciplinary teams within and across organizations. The general trend of aircraft manufacturers towards outsourcing design tasks [4, 5] increases the need for a more integrated basis on which cooperation takes place. Research into the application of collaborative MDAO have recently been a point of focus in projects such as AGILE [6], an EU-funded program that has contributed to new design paradigms and methodologies which provide the means to better integrate heterogeneous teams and therefore enable a more effective use of MDAO.

Collaborative MDAO Collaborative MDAO implies a close cooperation of domain experts on complex design problems through the integration of design competences and knowledge on an overall system scale. This is seen as an evolution of past MDAO systems in which disciplinary capabilities are integrated in a monolithic manner or through isolated, distributed analyses [7]. In collaborative MDAO, the disciplinary expert, his knowledge and his simulation tools are an integral part of the system. Due to their inherent limit in scope, however, domain experts tend to lack the overview of the complete system and their interaction with other domains, which emphasizes non-technical issues in the adoption of MDAO as previously mentioned. A key factor in overcoming these barriers is to employ the appropriate MDAO workflow modeling tools that facilitate a better understanding of the interactions in a complex system.

Current Approach and Limitations Although MDAO workflow modeling tools already exist and have actively been developed for the implementation of the AGILE paradigm [8], additional steps are required for an extensive adoption in industry. This can generally be attributed to three major drawbacks, some of which are mentioned in [9], including considerable inflexibilities in the setup processes, difficult customization of workflows, and lack of user-friendliness. Process inflexibility refers to the fact that prescribed steps must be adhered to in the MDAO workflow setup without the user understanding the reasons behind it. Further, tedious or lacking customization of the MDAO workflow forces existing workflows to adapt to a standard architecture as defined in literature [10], which is difficult to justify to engineers that have been using custom workflow structures successfully elsewhere. Lastly, the lack of user-friendliness, which manifests itself through a complicated user interface, extensive use of jargon, and strong reliance on implementation details, such as the use of a graph theoretic approach proposed in [11], significantly steepens the learning curve for users that are not familiar with the topic. In combination, these drawbacks pose a critical obstruction to introducing engineers to the AGILE paradigm.

Proposed Methodology With the MDAO Workflow Design Accelerator (MDAx) presented here, a different approach to simplify collaboration among engineering teams in large projects is attempted. Instead of defining an MDAO problem a priori and model their workflow according to predefined steps, workflow integrators using MDAx can directly interact with an eXtended Design Structure Matrix (XDSM)[12] interface using drag and drop operations with instant feedback and reversible actions without following forced procedures. This enables users to create and refine their workflow models that represent the tool configuration and process logic of an executable workflow in an efficient and flexible manner.

No prerequisites such as an existing schema or a tool repository must exist to start modeling, as they can be defined during the process as the project matures, allowing for the modeling of workflows according to custom preferences and enabling the exploration of possible workflow configurations that users may not have been aware of. Standard architectures can be generated on demand if the need arises, but are not strictly prescribed in order to generate executable workflows.

Being XDSM-centric, the user interface (UI) provides for standardized workflow model and enables an unambiguous inspection of tool interactions, which includes live assessment with respect to variable collisions and unconverged feedback couplings. A clean and straightforward graphical interface gets users started with minimal introduction and explanation, removing the steep learning curve in the process. Export features to various Process Integration and Design Optimization (PIDO) tools enable the generation and structuring of MDAO workflows before executing them in the framework of choice, minimizing manual labor in the setup of such workflows.

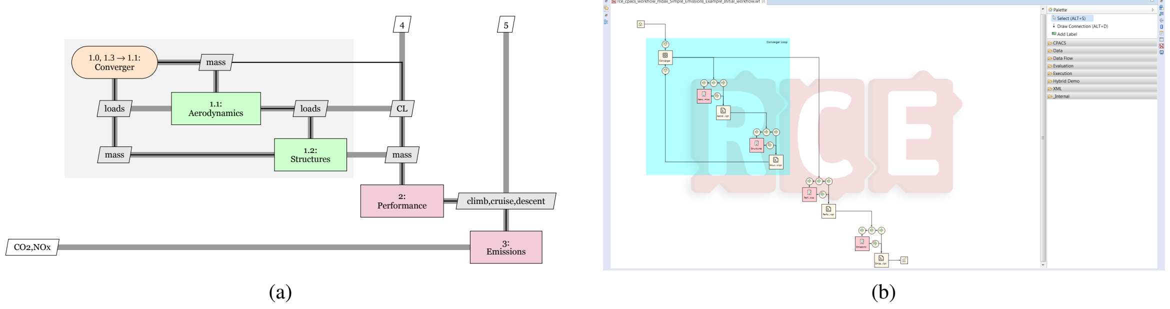


Figure 1 XDSM-representation of a workflow model (a) for the executable simulation workflow (b).

Rather than following a step-by-step process, MDAx is used like an MDAO workflow model editor to remove barriers to usability. It provides an environment for domain experts and integrators to explore and understand the overall system in a manner that requires little knowledge of specific methodologies and jargon. The aim is to take away the fear of looking across domain boundaries while making engineers aware of the complexity of the overall system, thereby making collaborative MDAO processes more efficient.

To emphasize important terminology, this paper makes a clear semantic distinction between simulation workflow and workflow model. While a simulation workflow represents the entity that orchestrates a sequence of operations among its components and regulates their exchange of data, a workflow model is merely its blueprint that can not be executed. In the context of MDAO, this blueprint is formalized in the form of an XDSM, as shown in Figure 1.

Paper Outline The structure of this paper is as follows. First, an overview of the state of the art is given in Section II, in which current tools and approaches are presented. The use cases for the development of MDAx are discussed in Section III, followed by a description of corresponding system requirements in Section IV. The presentation of the methodology used to create MDAx is done in Section V. In Section VI, more detailed insights into the user interface are provided, and the application of MDAx in a case study is presented in Section VII. A conclusion in Section VIII summarizes the design decisions and feature development of MDAx, and provides and outlook for future work.

II. State of the Art

A. Workflow Modeling

Workflow modeling is not a new endeavor and is being applied in many industries to set up and run simulations using computational tools. Tool integration is usually done using PIDO tools, also referred to as integration environments, that are specifically designed to provide a integrators with a User Interface (UI) for better usability. Most current PIDO tools, however, lack the capabilities and methodologies to support collaborative MDAO in the AGILE paradigm [13]. In particular, the absence of an ability to define tool interfaces, explore tool repositories, inspect tool couplings and formalize workflow models before their execution, poses a barrier to the full capacities of MDAO. It can be said that integration environments generally lay focus on the execution of tools and evaluation of results rather than the preceding modeling process, which in turn makes the current creation and application of MDAO systems inflexible and time intensive.

Recent attempts to establish a modeling environment for MDAO workflows propose the application of a graph-theoretic approach to model the exchange of data among simulation tools [11]. KADMOS [14], a software application developed as part of the AGILE project, expands this approach and implements a methodology associating specific graph-configurations with certain steps in the MDAO development process. It uses graph networks to store and manipulate workflow data and provides visualization capabilities through the client-side application VISTOMS [15]. An overview of these graphs is shown in Figure 2. Each step in the modeling process applies strict conditions on the associated graph structure, and only if those conditions are fulfilled is the graph structure "valid" and the modeling process can be continued. In the following, the most important open challenges in the approach and implementation of KADMOS are highlighted:

User Interface Although KADMOS provides a user interface through VISTOMS, it was not originally designed to be a UI workflow modeling tool. Its emphasis on graph networks results in an application that is build around mathematical structures and their application for automating specific modeling processes. User experience and ease of use come secondary to the strict application of its proposed methodology, manifesting itself in the lack of usability. UI design practices such as continuous user feedback and gratification, safe exploration, and habituation, are not accounted for.

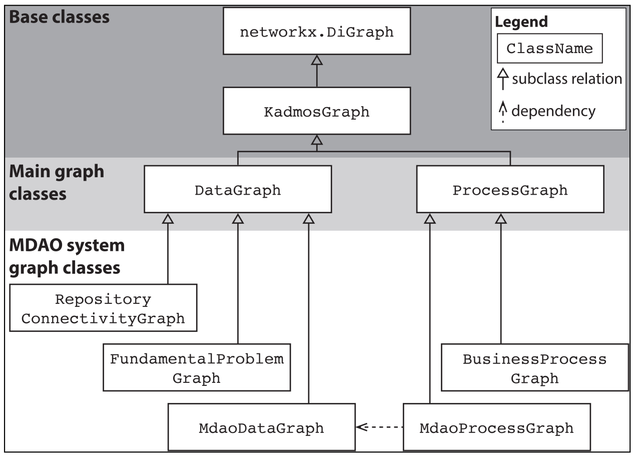


Figure 2 KADMOS graph class diagram. [16]

Language The focus on methodologies and mathematical representations of workflow models entails a heavy exposure to technical terminology in the application. Although the methodology may be sound given its assumptions, it is not approachable by the common user. This poses large entry barriers for those unfamiliar with associated academic publications, even for those experienced in modeling MDAO workflows. Heavy use of jargon with little accompanying documentation or guidance is regarded as detrimental in a collaborative environment.

Modeling Inflexibilities KADMOS' strict enforcement of its methodologies make the workflow modeling process stiff and inflexible. Its design choices emphasize the imposition of MDAO architectures defined in academia onto the workflow instead of focusing on customization and keeping workflow modeling as generic as possible, which is regarded as an important factor in bridging usability barriers.

Assumptions KADMOS uses assumptions in its methodology that are not well-reasoned and do not bear scrutiny in real-life scenarios. Self-loops, for instance, are not allowed in its workflows, as they may be seen as "Read and write simultaneously", whereas these scenarios occur frequently in MDAO workflows as "Read value, compute, and update" cases. Similarly, simulation tools without I/O are not allowed, whereas reality oftentimes requires execution of such tools that do not modify the parameters in the data schema, such as visualization programs, serializers, database tools, and others.

Software Engineering Lack of unit testing and a weak application of Object-Oriented Programming (OOP) means that KADMOS is volatile when it comes to extensions and modifications. SOLID principles [17] are not adhered to in the design of the application, making an adaption of the code-base time intensive. The lack of unit testing would make such an extension a difficult and error-prone task.

As a consequence, the authors decided to leverage the lessons learned from KADMOS and approach MDAO workflow modeling from a perspective of flexibility. Instead of focusing on a methodological use of a graph-theoretic representation of a workflow and strictly following predefined modeling stages, MDAx shifts the attention away from graphs and purely fixates on the use of the XDSM format as a modeling interface, where graphs and other mathematical structures are merely seen as implementation details. As an example, Figure 3 shows the comparison of a simple workflow represented in both the familiar graph and a possible matrix form, each one being valid representations of the data exchange among simulation tools when taking into account their position on the XDSM diagonal. The following section describes the methodology which was applied in MDAx to improve modeling capabilities in MDAO.

B. Development Towards Modeling of Complex Systems

The development of modern complex systems needs to account for an ever increasing number of capabilities to be delivered, as well as for organizational boundaries, integration and communication challenges, and constraints stemming from all stages of the product's life-cycle. Ideally every decision taken at each stage of the development should be evaluated along the entire life cycle. The management of such development complexity requires a shift to a novel system development paradigm.

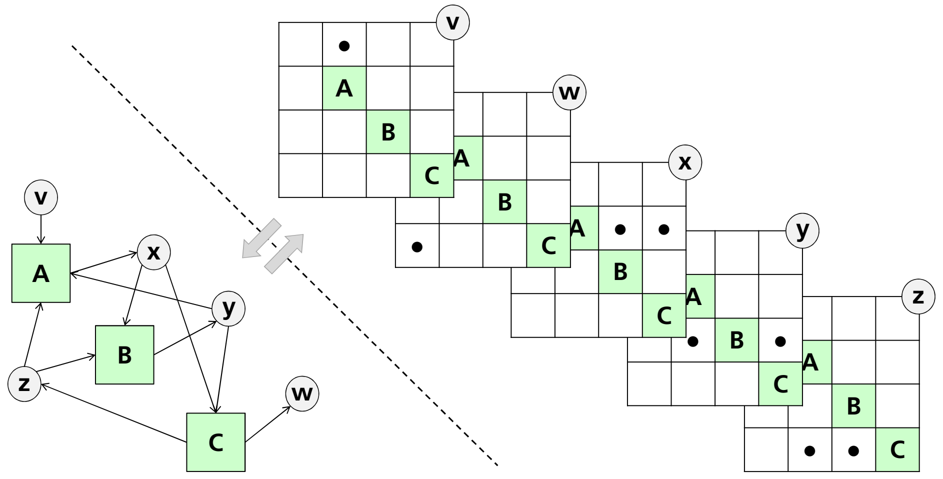


Figure 3 The same design problem represented in graph (left) and matrix (right) form. MDAO workflow entities and their relationships can be represented using various mathematical structures and are regarded as an implementation detail instead of being central to the methodology used in MDAx. In this example, the I/O couplings of each parameter in the graph are expressed as a stack of N^2 Diagrams.

In this context, the DLR Institute of Systems Architecture in Aeronautics is developing a novel "model-based conceptual framework" for architecting, designing and optimizing complex aeronautical systems. The expected impact is a drastic reduction in time and costs associated with the development, via an increased transparency, efficiency, and traceability of the design and decision making processes. The conceptual framework, introduced in [18], extends the scope of design and optimization methods to all the phases of the development life cycle of complex systems.

On one hand, the framework focuses on the acceleration of upstream architecting phases such as the trade-off of goals, the definition of scenarios and requirements accounting for all the involved stakeholders, the design and optimization of the architecture of the system of interest (e.g. an aircraft), or a system of systems (SoS). On the other hand, it is concerned with accelerating the downstream product design phases, including the selection of the capabilities needed at every design stage (e.g. conceptual, preliminary, detailed), the integration into a design process, and the deployment of design systems (e.g. computational environments) for the exploration of the design space and the selection of the optimum solution. The architecture of the conceptual framework has a layered structure, as shown in Figure 4. The identified five layers and major activities within each layer are:

Enterprise Layer Modeling and optimization of goals and capabilities via value-driven decision making approaches, enabling trade-off between policies by the enterprise.

System of Systems Layer Architecting and designing complex SoS scenarios for a given set of capabilities to be delivered, enabling trade-off between concept of operations.

Complex System Layer Architecture Design and Optimization (ADO) of a complex system of interest for a given set of requirements and concept of operations, enabling trade-off between architectures.

Development System Layer Deployment and operation of a development system and processes (e.g. MDAO process) for the design and optimization of the system of interest for a given architecture, selected design strategy (e.g. minimum costs), and dimensionality of the design space.

Competence Layer Providing heterogeneous capabilities (e.g. disciplinary analysis) and services available, or to be developed, enabling the system design and optimization.

The implementation of the concept is supported by the development of novel design methods and approaches, leveraging digital design engineering and modeling technologies. The work presented in this paper focuses on the Development System Layer, and is part of the European Commission funded project AGILE 4.0 (2019-2022) [*].

[*] https://www.agile4.eu/

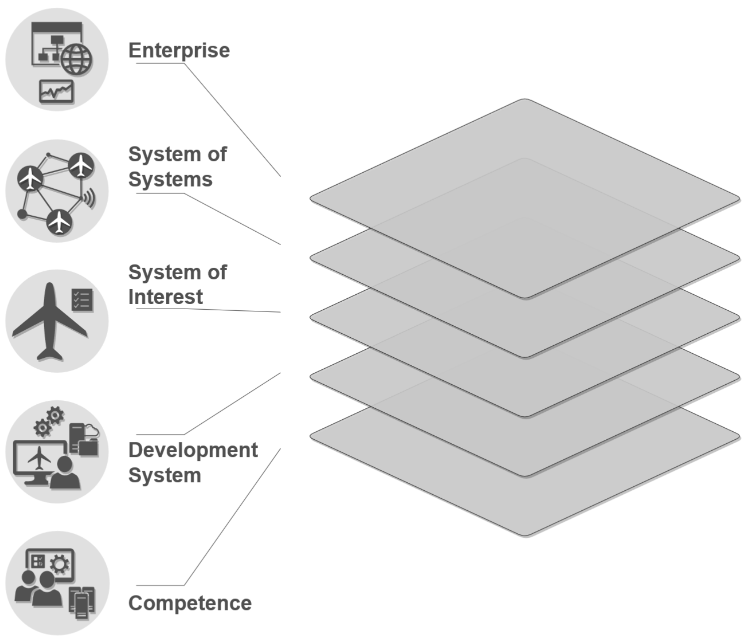


Figure 4 System architecting environment DLR-SL concept: multi-layered approach [18].

III. Identified Needs

This section presents the objectives that MDAx is designed for, which are derived from experience in working with both workflow integrators and disciplinary specialists, as defined per [19], and attempts to cover the most relevant scenarios in collaborative engineering.

The objectives are expressed as use cases and classified into three categories, as shown in Figure 5: Exploration, Verification, Documentation. The exploratory category deals with processes such as the set up and inspection of available workflow components, the organization and use of a common data schema, and the creation of a workflow model. The verification category is concerned with the inspection of the selected workflow configuration, the set up for the right conditions for the execution of the workflow, and the inspection of connections among the workflow components. Finally, the documentation category describes the aspect of formalizing and communicating the MDAO workflow models.

A. Exploration

1. Establish Data Schema

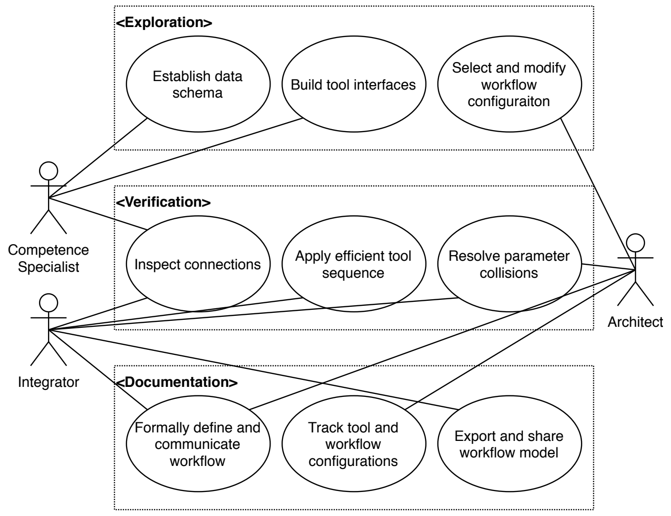
In order to simplify the communication between simulation tools, aircraft design workflows frequently use a Central Data Schema (CDS) such as CPACS [20] to establish a common language which the tools use to exchange data. These schemas are not static and often require modifications or expansions, especially when new disciplines join the tool landscape that have not been considered before. Due to the fact that tool interfaces are defined by those schemas, changes in a schema always propagate towards the tools that use it, entailing a lot of repetitive work.

On the other hand, projects that do not have an established data schema have an emerging need for the continuous definition and expansion of such a schema as the competences and contributions of each project member become more explicit. In these scenarios, tool definitions and creation of the data schema occur concurrently requiring frequent, iterative adjustments.

2. Build Tool Interfaces

A common scenario in projects involving multiple partners from heterogeneous disciplines is the derivation of workflow entities and their interfaces in the presence of an existing data schema. Project planning often requires a clear definition of work package contributions, and since workflow components such as software tools may not exist at these early stages, collaborators need to establish the required interfaces before the components are realized and integrated.

Parallels can be drawn between this practice and the software process of test-driven development (TDD) [21], where unit tests define the interface of a software element, such as arguments and return values, before the element is implemented by the developer to pass those tests. Engineering teams, such as those working together on complex simulation workflows, can benefit from the design of interfaces in the same way. The perks of abstracting away the implementation of entities, whether they are simulation tools or software components making up an application, can help teams across domains where the collaboration is paramount.



<Exploration>
Establish data schema
Build tool interfaces
Select and modify workflow configuraiton
Competence Specialist
<Verification>
Inspect connections
Apply efficient tool sequence
Resolve parameter collisions
Architect
Integrator
<Documentation>
Formally define and communicate workflow
Track tool and workflow configurations
Export and share workflow model

Figure 5 Overview of main use cases and actors in MDAx.

Tools that are already part of an existing tool landscape benefit from modeling their I/O by making their interfaces explicit. The lack of a formal definition of the interface of entities that exhibit black box behavior poses great challenges in the execution of workflows of any size, since bugs and inconsistencies are difficult to track and resolve. Any resulting design inconsistencies make it challenging to efficiently perform trade studies.

3. Select and modify workflow configuration

One of the main use cases for the development of MDAx is the selection and modification of workflow configurations that can subsequently be executed in the appropriate PIDO environment. Based on the available tool landscape, integrators and architects are often interested in the quick setup of workflow models to determine which tools to use for a specific MDAO problem at hand. Since the demands on the workflow configuration may change over time, such as when product models go from initial to preliminary design phases and the required fidelity increases, agile modifications and refinements of workflow models are needed. This necessitates capabilities to import, reuse and modify existing workflow models or their components according to those changed requirements. One of the main criteria for this to occur in a seamless manner is the ability to add or remove tools at any point in time, without a need to restart modeling from scratch.

More often than not, workflow integrators have a good idea about the tool landscape that is used for a particular design problem. In such cases, workflows are modeled using common sense and domain experience, and do not follow specific MDAO architectures. Specifically, the use and application of architectures defined in academia [10] are often seen as impractical or require expertise, and therefore not given much attention. There is a preference towards freedom and flexibility in workflow modeling without having to follow predefined methods or processes. Imposing a solution strategy takes control away from the user that has a lot of experience in the setup of his workflows. That said, an automated application is viewed as beneficial when the design problem and tool configuration is fully defined and the most efficient workflow, based on metrics such as execution time, is desired.

B. Verification

1. Inspect Connections

In cases where both the schema and simulation tool I/O are defined, workflow integrators are challenged with keeping an overview of the connections that exist between tools. The inability to inspect connections among tools in a clear manner constitutes a major reason for time intensive debugging activities. Not having the oversight to determine which parameters are exchanged between which entities and at which point in time means that large parts of the workflow must be examined to assert the origin of any appearing issues. Enabling an efficient inspection of tool connections allows tool integrators to keep and overview of the complete data exchange in a simulation workflow.

2. Apply efficient tool sequence

Instances in which workflow integrators are not familiar with the tool repository, or where complex and nested workflows are subject to change as new simulation tools and technologies are developed, finding an efficient tool sequence may prove to be difficult. When tool sequencing support is lacking, the efficiency of the workflow execution exclusively depends on the integrators experience and insight into the parameter couplings among tools.

Equally important is the support for removing redundant tools that do not impact the parameters of interest, adding avoidable inefficiencies to the workflow execution. Complex workflows, however, can make it difficult to manually determine which simulation tools affect those parameters, resulting in a need to automate such activities.

3. Resolve parameter collisions

Workflow integrators are frequently faced with the issue that parameter collisions are not emphasized during the modeling process, leading to an ambiguous exchange of parameters that inevitably results in design inconsistencies. Although collisions in workflows are common, they are often recognized late in the design process and only after investing considerable effort. Approaches to quickly find and resolve such collisions in a straightforward way do not exist, resulting in the application of trial-and-error debugging methods through manual search and workflow modification.

C. Documentation

1. Formally define and communicate workflow

Although workflows can be exchanged among collaborators on existing platforms using implementation-specific files, they often use internal standards and are not easy to interpret by users unfamiliar with the workflow, tool landscape, or integration platform. There exists a need to formulate the most relevant details of a workflow model in a structured, implementation-neutral form such as XDSM that can effortlessly be interpreted by humans and easily serialized to machine code.

2. Track tool and workflow configurations

Since workflow models are rarely static and have to evolve with time due to design maturity, changes in design requirements, or changes in the available tool landscape, the ability to track workflow modifications may become important. Similar to well-known version-control systems, needs exist for the state of a workflow to be tracked through the implementation of a workflow tree structure, where each workflow builds on top of another workflows, making it easy to track modifications and inspect differences among them.

3. Export and share workflow model

Workflow models are not only useful for the production of executable workflows, but can also be a useful tool in communicating workflow integration, discussing the tool landscape, and defining or refining tool interfaces with a project consortium. Aside from the requirement for exporting an executable workflow model, there exists as strong need in export functionalities that represent workflow information in commonly used data formats such as PDF, SVG, and HTML.

IV. Modeling Environment Requirements

Following the use cases from Section III, the main system requirements that the authors identified as being essential to an application that enables collaborative MDAO are presented and discussed. In particular, the *MoSCoW* method [22] is used for this analysis to identify and rank these requirements according to their importance. Many of the listed requirements are posed in the spirit of the Agile Software Development manifesto [23], as it pertains to many best practices in software development.

A. Must Have

Intuitive UI In order to remove barriers to MDAO workflow modeling, disciplinary engineers and integrators should only be exposed to an intuitive user interface and not know about implementation details.

Continuous Feedback To take full advantage of modeling capabilities, users should receive visual feedback on each action. Each action should yield an immediate effect that reveal program behavior to the user. This not only helps to keep track of introduced changes and issues, but also encourages a learning-by-doing approach, minimizing the need for documentation.

Workflow Inspection Inspecting relevant workflow details such as parameter exchange and collisions, feedback couplings, tool sequence and tool metadata, should be straightforward and obvious.

XDSM-Based To represent workflows in an implementation-neutral format, the XDSM scheme should be used as it conveys both data exchange and process flow of workflows in a concise and transparent way.

Customization Workflow models must be customizable and not strictly follow MDAO architectures described in academia. Practical scenarios often require a high degree of customization, especially in nested and complex sub workflows.

No Jargon Integrators and disciplinary experts should not have to have detailed knowledge on MDAO architecting or associated methodologies to be able to model workflows.

Modularization To follow best practices in software development, the business logic of the application must be an independent unit. It must be completely independent of the Input and Output (I/O) schema and format selected for the simulation workflow, any persistence choices, and the user interface.

Export In order to share and execute workflow models, capabilities must exist that provide a means to export the workflows to PDF and CMDOWS [24].

B. Should Have

Undo To allow for fearless modeling [25], redo/undo functionalities must be available to the user. This way, design decisions can easily be reversed and configurations changed without repetition.

Workflow Collapse Large workflows can quickly become convoluted, leading to difficulties reading and interpreting relevant information. To allow for a better overview and encapsulation of sub modules, workflow nesting and collapse capabilities should be available.

Schema Flexibility As projects rarely have a static CDS, and changes frequently occur, modifications and extensions of the used schema should be easily done. Existing workflow models must be flexible to changes in the used data schema.

Workflow Version Management Over the course of a collaborative design campaign, multiple versions of an MDAO workflow may exist. A way to manage these versions and relate them to each other should be provided.

Tool Redundancy Redundant tools for a specific design problem should be able to be found and removed automatically.

Tool Sequencing The sequence of tools in an executable workflow has a high impact on workflow performance. A way to apply sequencing algorithms onto the selected tool configuration should be available.

C. Could Have

Automatic Architectures Automated MDAO architecture generation is a very convenient feature that can save time in the implementation of executable workflows.

Workflow Import Domain-specific workflow models should be able to be created separately and subsequently integrated in the interdisciplinary workflow. Just like disciplinary tools are treated as black boxes, entire workflows should be as well. This enables an independent modeling process in which a sub-workflows can be isolated by defining their interface with the parent workflow.

D. Won't Have

Workflow Execution Close integration with a specific integration platform, or providing the capability to execute workflows, adds convenience to the use of a seamless integration between modeling and execution.

V. Methodology

As previously mentioned, the main idea of MDAx is to extend the XDSM format with a set of rules to make the XDSM modeling environment easier to use and understand. The application of graphs in the construction of an XDSM is regarded as an implementation detail and not of central importance to the modeling process. MDAx does not strictly follow the graph-based methodologies developed in [16] since the algorithms pertaining to the business logic of the application can be implemented in many different ways; using a non-graph approach, for instance. In essence, the focus on a clear user interface and straightforward use of modeling capabilities is more important than strict adherence to methodologies in the design of complex systems, which is expressed well by the Agile Software Development philosophy of *Individuals and interactions over processes and tools.*

A. XDSM Design Rules

In order to simplify modeling simulation workflows in the XDSM format, MDAx extends the rule set that is placed on that format. On one hand, an expansion of the existing rule set limits the design space of all possible workflow configurations. On the other hand, MDAx leverages the clarity that the format provides while aligning its usability in a modeling environment with the intentions of a workflow integrator. Challenges present themselves in finding the fine balance between flexibility, clarity, and simplicity in the use of this modeling environment.

Block Sequence While the XDSM format does not derive any execution information from the position of the blocks on the diagonal, the rule extension in MDAx does so. With the sequence of blocks counted from top left to bottom right, downstream blocks are generally executed at a later point in time than upstream blocks. This holds true for all sequential couplings and along concurrent threads as seen in Figure 6, where the blocks upstream from block *E* are run later in the workflow due to existing concurrency. Encoding the block execution precedence from left to right on the diagonal has been found to be intuitive as it aligns with the left-to-right writing system.

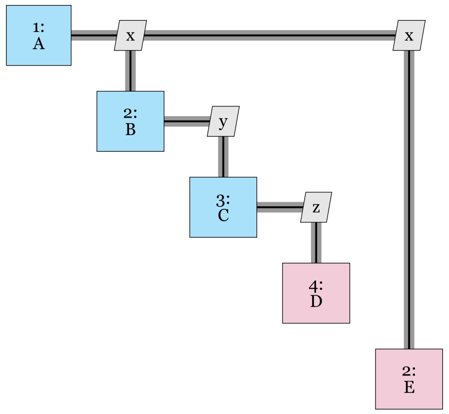


Figure 6 Block sequence in concurrent workflow where independent workflow threads are executed simultaneously, as indicated by block numeration.

Driver Encapsulation In XDSM, driver blocks can generally be placed anywhere on the diagram diagonal and may drive non-adjacent blocks. The process lines and numbers provide the information on when the blocks execute and which driver runs them. MDAx confines the driver block position to the top of the blocks that they "wrap", and wrapped blocks must also be adjacent to one another. This means that each driver has an envelope with at least one non-driver block, and any number of nested driver envelopes. Two envelopes may not overlap, meaning a tool block can only be in one envelope at a time.

This rule has several important effects on the modeling environment. First, each driver envelope becomes equivalent to an independent block and can the represented as such by collapsing the envelope, as seen in Figure 7. Complex sub-workflows can therefore be modeled as independent components and visualized as such, hiding the inner workings and only exposing their interfaces. The encapsulation of driver envelopes prohibits their overlap, where one block may be part of two separate envelopes at the same time. This reduces ambiguity of block execution sequences since each wrapped block can be distinguished as part of the same, and only that, envelope. However, a repeated execution of the same simulation tool within different envelopes requires a separate block in the diagram.

Schema-Based Data Exchange MDAx assumes a data exchange according to a CDS in its system composition and therefore applies further restrictions to the XDSM. The adherence to a common data schema in combination with an inherent sequential order based on block position results in limitations on how parameters can be exchanged, or routed, within the workflow. As a direct result, configurations as shown in Figure 8, where one block output "overtakes" the same parameter written later in the sequence, are prohibited in executable workflows, since the parameter would be overwritten by a more recent value implicitly. Although this rule constraints the possible workflow configurations, it ensures that data exchange is always performed consistently, even when dealing with a high degree of concurrency, and adds to more clarity in the temporal dimension of data exchange. The limitations can generally be remedied by repositioning the blocks on the diagonal or by modularizing tools to separate affected parameters into new blocks.

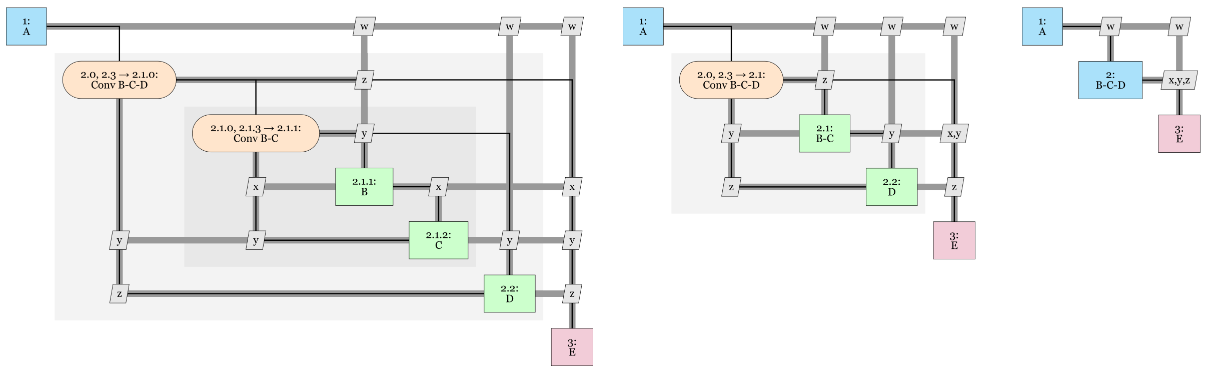

1: A
2.0, 2.3 → 2.1.0: Conv B-C-D
2.1.0, 2.1.3 → 2.1.1: Conv B-C
2.1.1: B
2.1.2: C
2.2: D
3: E
w
z
y
x
2.0, 2.3 → 2.1: Conv B-C-D
2.1: B-C
x,y
2: B-C-D
x,y,z

Figure 7 Collapsing nested driver envelopes to equivalent block representations. Each envelope, depicted as a gray rectangle, can be reduced to a block with the same I/O interface.

Block Couplings Two rules worth mentioning pertain to the existence of self-loops and lack of block I/O, that deviate from the approach presented in [11] and [14]. MDAx regards self-loops of blocks as "value updates" rather than "simultaneous reading and writing of the same parameter address". Value updates are a common scenario in practice where initial values are assumed or parameter fidelity changes, and should therefore exist in a workflow model. Similarly, blocks that have no input and/or no output and thus no connection to the used data schema are frequently used to visualize results, as in the case of TiGL [26], or to write to a database or file. Although not modifying the design, their presence in the workflow may be required, and therefore also represented in the workflow model.

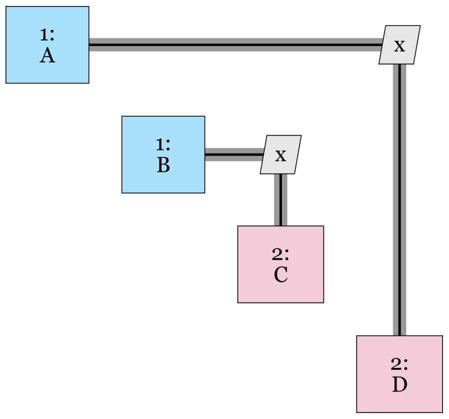


Figure 8 Example of a prohibited data exchange due to sequencing rules, where a parameter is overwritten by a preceding block.

Parameter Ambiguity In order to obtain executable workflows, ambiguity on the source of parameter values must be removed. These ambiguities can manifest themselves in the shape of parameter collisions and feedback couplings. Parameter collisions exist when more than one workflow component provides the same parameter at the same time, whereas parameter feedback occurs when parameters are required before being made available by the system. Although workflow models are considered valid despite the existence of collisions and feedback, MDAx will indicate them to be non-executable since practical applications of these workflows will inherently suffer from design inconsistencies.

B. Single Source of Truth

In order to have a fully consistent workflow diagram that displays identical behavior when receiving the same inputs, MDAx follows the *Single Source of Truth* (SSOT) principle. This principle dictates that data can be modified in one place only, and all other links to that data are done by reference. MDAx views the following items as its sources of truth in order to construct executable workflow diagrams:

1) Blocks containing I/O information, block meta data, and envelopes
2) Index of each block on the diagonal
3) Collision resolution decisions

The first item refers to the presence of block input and output information, as well as meta data such as block type, name and version. These are required to establish connections between present blocks based on their needed and provided parameters as described by a central data schema. Driver blocks, in addition, contain envelopes that outline all their encapsulated blocks for the application of feedback algorithms. To assemble the diagram and apply the modeling rules stated in Section V.A, the index of each block must be stored. This way, both data and process algorithms that determine block coupling and execution sequence can be applied. Finally, collision resolution decisions are stored to determine how coupling ambiguities in the form of parameter collisions are eliminated.

The application of the SSOT concept means that the same workflow can be created and recreated based only on those few pieces of information. This helps keeping the dependency system organized while asserting that information conflicts do not occur. All generated object structures, graph networks, and adjacency matrices are a mere result of these source objects. Changes in any of those items will propagate through the application and ultimately result in a different workflow. This also means that modifications to the workflow configuration are internally performed on these objects, triggering the diagram to adopt these adjustments using a lazy *call-by-need* evaluation strategy [27].

C. Diagram Generation

Due to the employment of XDSM design rules and the adherence to the SSOT, many of the state-of-the-art methodologies outlined in Section II are not strictly followed in the generation of a XDSM workflow model. Instead, MDAx relies on OOP paradigms to construct the diagram and to remove all workflow ambiguities. Blocks display a strongly polymorphic behavior such that the diagram does not know about the type of block it deals with; interactions with each block occur over common interfaces. This helps in keeping the algorithmic logic separate and independent of other code that concerns the construction of the XDSM diagram. The creation of the diagram itself can be divided into three phases, as seen in Figure 9, and can be described as follows:

Phase ① The initial phase performs a forward iteration on a container that stores all blocks sorted by index, loading each block object into the diagram diagonal and extracting their I/O definitions. Based on those definitions, the connections between all blocks are determined. At the end of this phase, a diagram that may contain parameter collisions and feedback couplings is established. Driver blocks do not have any connections at this point, as those are established in Phase 3 during the backward iteration.

Phase ② This phase is solely concerned with the resolution of parameter collisions. For this purpose, collision resolution objects that contain decisions on the method of removing collision, are applied. These objects typically contain information on which connections to keep, cut, and create. Figure 9 shows an example case where the collision affecting parameter a is resolved by eliminating the route B-a-C. Both automated and manual collision resolution, along with related parameter routing principles, are explained in more detail in Sections V.D.4 and VI.B, respectively. Important to keep in mind is that even when collision resolutions are applied, the original couplings are kept track of in order to uphold SSOT principles.

Phase ③ The last phase iterates the diagram blocks backwards and polymorphically calls their feedback resolution methods. These methods apply each block's feedback resolution algorithm onto its own envelope. Since only driver blocks contain envelopes, calling these methods on other block types has no effect. The backward progression takes advantage of the fact that each driver immediately precedes its envelope and that those envelopes are self-contained, making related algorithms straightforward and free of side-effects. Deeply nested workflow configurations are thus possible and follow the same mechanisms as flat workflows. At the end of this phase, a check is performed on any existing ambiguities in the diagram. If all collisions are removed and feedback couplings are eliminated, the process sequence for the workflow execution is determined. An exported workflow model can subsequently be executed in a PIDO tool such as RCE[†] without causing inconsistencies due to parameter exchange.

The generation and update of workflow diagram occurs dynamically based on user inputs, and generally undergo the same three phases. Many component states and configurations, however, are cached and do not have to be re-computed when re-loading the diagram so as to enhance performance. Changing a block position on the diagonal, for instance, may result in different forward and feedback parameter couplings since coupling placement depends on block placement. However, the coupling itself is not affected. This means that Phase 1, where all blocks are loaded into the diagram, does not have to be triggered. This application of lazy evaluation and caching of results preserves component states that are not affected by diagram modifications, making MDAx responsive even for large workflows.

D. Efficient Architecting

In order to simplify workflow modeling and reduce repetitive tasks, MDAx features various functionalities and automatic procedures allowing workflow integrators to focus on creative work and maximize value generation. The most important ones are described in this section.

[†]https://rcenvironment.de/

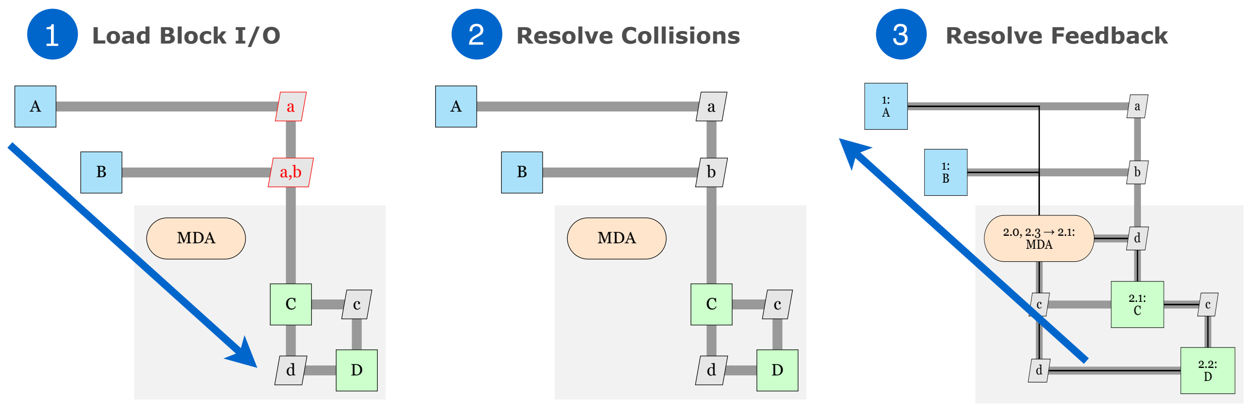


Figure 9 The three phases in MDAx workflow diagram generation. Phase 1 calls each block to construct the diagonal, load the I/O, and establish all variable connections. The diagram now contains all blocks and variable couplings. In Phase 2, any existing collision resolution decisions are applied to the diagram to eliminate ambiguities. If selected, automated resolution algorithms are applied to resolve all collision according to a single convention. Phase 3 iterates the block diagonal backwards and forces each block's feedback resolution algorithms to be applied on its own envelope. If no workflow ambiguities exist at the end of this phase, the execution sequence is derived.

1. Automated Sorting

Although MDAx emphasizes the flexibility for users to model customized workflows, it offers the application of automated sorting algorithms for more convenience since the execution of an efficient tool sequence is desired in most use cases. One such algorithm, as proposed in [28], minimizes the amount of parameter feedback couplings in the diagram. Figure 10 shows how the block sequence of a generic workflow can be reordered for minimum feedback connections through a single button click. These algorithms are recursive and work equally well on both flat and nested workflows at any complexity.

2. Collaborative Interface Modeling

To allow for workflows to be modeled from scratch even when no commonly agreed upon data schema exists, MDAx enables the concurrent definition of tool interfaces and data schema. The methodology in [14] expects a fully defined schema and tool repository, which in reality is often not available. The process of setting up a data schema among heterogeneous partners is not only very time intensive, but also highly iterative in the early stages of a project. Clashing interests have to be resolved and technical descriptions agreed on before the parameters can be formalized in a data format. MDAx views this phase as part of the modeling process and provides an environment to support engineers in a collaborative development of schema and tool interfaces while constructing the workflow.

3. Redundant Blocks and Parameters

Projects that reuse an existing tool configuration but only require the computation of a subset of parameters may not need to execute all simulation tools. To easily filter out blocks that are not required, those redundant for the computation, MDAx allows to select target blocks or parameters and remove blocks that are not needed. Figure 11 shows the Sellar problem [29] workflow where parameter f is selected as a single target parameter and all blocks that do not affect it, directly or indirectly, are marked red for removal. This way, tool configurations can instantly be minimized to only those required for a specific computation.

4. Automatic Collision Resolution

In order to avoid repetition in the collision resolution process, such as when a high number of parameters exhibit collisions, the availability of automated resolution capabilities becomes important to MDAx usability. Modeling workflows where the same simulation tool has to be executed multiple times at different stages, for instance, always results in parameter clashes that have to be resolved. MDAx offers three algorithms that help integrators to perform

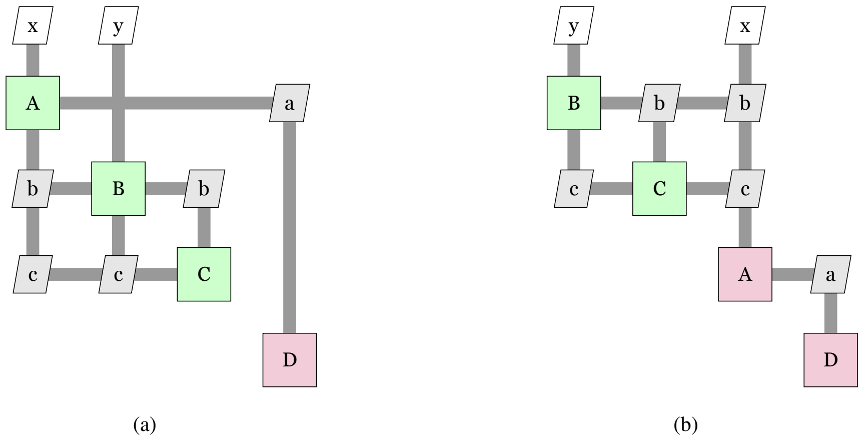


Figure 10 Automated block sorting algorithm applied to a generic workflow configuration (a) to minimize feedback couplings by adjusting block sequence (b).

automated parameter routing in case of collisions.

The algorithm in Figure 12a resolves collisions in such a way that each block receives the latest *available* parameter value as well as the last *global* value. Based on the perspective of a specific block, *available* refers to the fact that a parameter has been computed before the point of execution of that block, whereas *global* parameter values may be computed at some point after the blocks execution and therefore must be fed back for consistency. Feeding back the latest computed parameter in the block sequence ensures that the latest update of a parameter value will be used in all blocks that require it.

A second algorithm shown in Figure 12b eliminates collisions by taking the *closest* available and feedback values, allowing workflow sections to be encapsulated and made independent of downstream processes. Closely related is the algorithm seen in Figure 12c that follows the same logic, but takes into account consecutive blocks that provide that parameter value. Here, the latest value of the closest consecutive set of blocks is used to resolve parameter collisions.

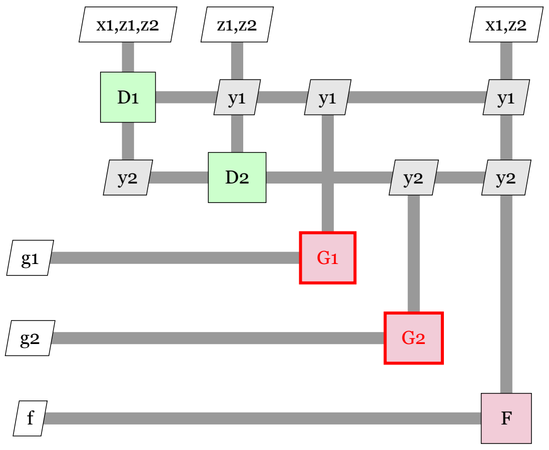


Figure 11 *G1* and *G2* marked red for removal when selecting parameter *f* as a target. Multiple target parameters and blocks may be chosen.

If selected, the application of the mentioned automatic collision resolution algorithms occurs automatically during Phase 2 of the workflow construction process described in Section V.C.

VI. User Interface

Formulating MDAO workflows is a complicated and iterative task. To ease this task, the user should be updated about the progress and given hints on how to proceed next. To do this, MDAx employs a graphical user interface that enables users to create and manipulate executable MDAO workflows. The following design guidelines are adhered to:

Intuitive The UI should be easy to understand, and it should be possible to operate it without in-depth knowledge of MDAO.

Explain When Needed For more complicated concepts, as will be inevitably needed, clearly explain the concept at the same spot as where it is used.

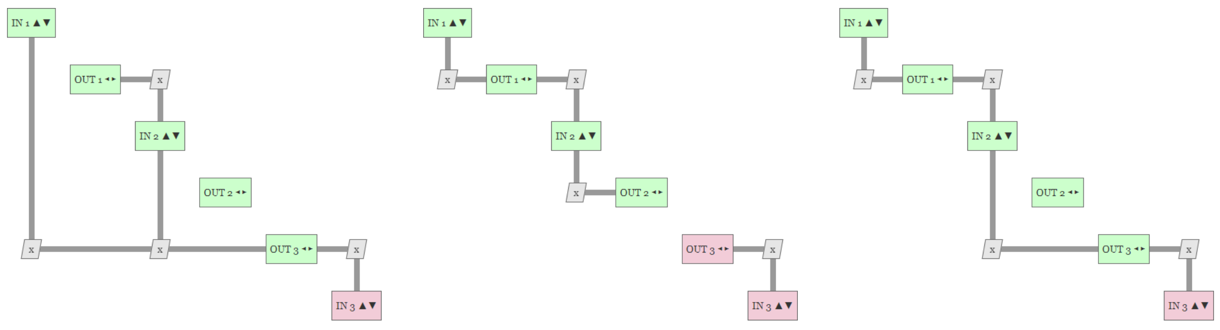


Figure 12 Automated collision resolution options. In algorithm (a), the *latest* available and *latest* feedback values are routed to the blocks requiring the parameter. In (b), the values from the *closest* blocks in the sequence are taken. Similarly, (c) resolves the collision by routing the parameter through the latest of the *closest consecutive* set of blocks that provides the value.

Customizable The workflow should be fully customizable: it should be possible to change anything at any time.

Users have some expectations when using software. Some common user patterns [30] describing these expectations are taken into account into the design of the UI:

Safe Exploration Users should get the feeling that it is "safe" to explore the user interface. The main implication is that some mechanism of reversing any action is needed. In MDAx, an undo/redo mechanism (the Multi-Level Undo pattern) is implemented to provide safety. From a software perspective there are two ways to implement this: Command (implement all actions as commands with an undo action) or Memento (store the program state after every action, and simply replace the program state with a previous state when undoing an action). MDAx uses the Command pattern.

Instant Gratification Users want to get immediate feedback from the actions they perform. This means that the UI should be quick to respond, always represent the most recent underlying data, and let the user know when some action has been completed. The backend and frontend code has been optimized to be as fast as possible. For actions that take longer than half a second, the user will know that the program is working through a spinner. Finally, for actions that do not lead to visible changes in the workflow, a popup message with a success message upon completion is shown.

Deferred Choices Related to Instant Gratification, this means that users expect short processes for doing some actions, where only immediately important choices (like giving some input) have to be taken. It should be possible to take other choices, that might be important, but not needed for the thing the user wants currently, at a later time. When adding tools, it is not needed to also provided the tool input and output files, but the user can add a tool right away and start editing its input and output from there. Furthermore, there are never forms where more than approximately three items are required input: the rest is optional and can be added later on.

Incremental Construction Users do not create things in a precise order, but rather starts with a small piece, and then incrementally and iteratively build the thing. Here, getting feedback is important to allow users to fix mistakes and stay on the right path. It should be possible to easily start over if the current progress is not what the user wants. Similarly, it is important to immediately show the user the new state of the workflow after some action is performed on it. The central design philosophy is that editing the workflows should be flexible, which enables incremental construction.

Habituation Users already have much experience using other user interfaces, and therefore for common operations the user interface of MDAx should be as similar as possible to other user interfaces. Examples of this include commonly used keyboard shortcuts for saving a file or creating a new file, or using the gears icon for showing the workflow settings.

Streamlined Repetition Sometimes the user might need to repeat some chain of actions many times to achieve something. In these cases, a way of performing them all at once using only a few clicks should be provided. For MDAx, adding many tools to a workflow is an example of this. A batch import function was implemented to add multiple tools at once to ease this.

The main structure of the program is built around the **Canvas Plus Palette** pattern: the main window contains a canvas whereupon the workflow is built, with small buttons above it that enable actions to be performed on the workflow. Editing and inspecting elements in the workflow is directly done by clicking on the elements, or dragging them to rearrange the order of the tool blocks. This is a design pattern used often for editors [30]. The workflow itself is displayed using the eXtended Design Structure Matrix (XDSM) notation [12]. This is a well-known format in the MDAO field, and directly conveys the most important information needed for understanding a workflow: which tools are there, what data do they exchange, in what order are they executed, and how do convergers and optimizers interact with them.

In addition, the following patterns [30] are used to make the program as easy to use as possible:

Tree Table The variables being communicated between tools are represented as a hierarchy, and displayed in the user interface as a tree table: data is displayed as a list in a table, indentation is used to denote hierarachy, and nested levels are collapsible. This allows us to display long lists with many nested lists in them.

Action Panel Buttons for manipulating the centrally displayed workflow are displayed in a panel (toolbar) that is always visible. This is done so that it is always clear what actions can currently be performed on the workflow. In addition, buttons related to similar actions are placed near each other according to the **Button Groups** pattern.

Modal Panel Popups that take attention away from the main screen (modals) are used for requesting input from the user for more complicated actions, and for showing data such as the variable tree. The modals feature **Escape Hatches**, ways of getting out of the current action, and **Prominent Done Buttons**, easily recognizable buttons that need to be clicked to apply the action. A **Preview** is shown when the action to be applied potentially changes the layout of the workflow.

A. UI Implementation

The core of MDAx is implemented as a Python library, where all workflow logic is handled. The UI of MDAx is implemented as a layer on top of the code. A web-based UI architecture is used: on the backend side, a Python script starts a web server and initializes the core; on the frontend side a Vue.js[‡] application renders the frontend and handles user input. Communication between the backend and frontend is handled using a websocket. Such an architecture allows us to separate the execution of the backend and frontend, such that in the future the application can be offered as a service accessible through any common browser, without the user having to install anything on their local computer. However, it is also possible to deploy it as a standalone application if that is needed.

Figure 13 shows the main screen of the UI: the workflow is displayed on the central canvas, and above it are buttons for manipulating the workflow and managing the current project. One important item is the status indicator: it tells the user whether the workflow currently would be executable, and can either show a red exclamation mark, indicating the workflow has issues (as shown in the figure), or a green check mark, indicating the workflow is runnable. In addition to there being no executable blocks, the two main reasons a workflow can have issues are:

Unconverged Feedback If a workflow has feedback connections that are not handled by an MDAO block (e.g. a converger or an optimizer), these feedback connections are unconverged.

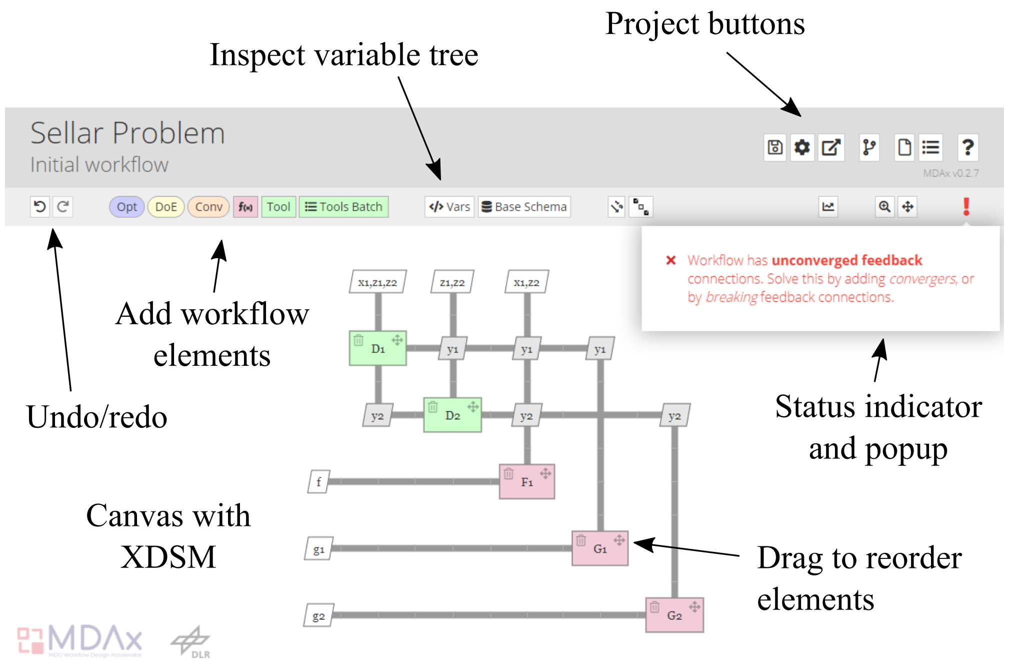
Collisions When multiple tools output the same variable, this is called a collision. Collisions need a collision resolution, which can either be done using an automated strategy (as discussed in Section V.D.4), or manually using the UI (see Section VI.B).

Inspecting the workflow variable tree and individual variables is shown in Figure 14. The connections from and to the individual variables can be edited from the variable information screen. The interface for adding converger elements, which is very similar to the interface for adding optimizers and Design of Experiment (DoE) elements, is shown in Figure 15.

1. Workflow Branching

Over the course of a design campaign, multiple workflows might be needed that are based on the same base workflow but are used for different things. For example, one might just run an analysis to debug and validate the workflow, one might perform a Design of Experiments (DoE) to inspect the sensitivities of the base design, whereas another one might actually run an optimization to find the best possible design, all using the same analysis tools. To facilitate this, MDAx uses the concept of workflow branches to easily derive new workflows from existing workflows. This results in a tree

[‡]https://vuejs.org/



✕ Workflow has **unconverged feedback** connections. Solve this by adding *convergers*, or by *breaking* feedback connections.

Figure 13 Main screen of MDAx with loaded Sellar problem. The workflow is displayed as an XDSM on the central canvas, above which is the toolbar with buttons for manipulating the workflow: undo/redo, adding new elements, inspecting the variable tree, sorting, etc. On the right, the status indicator shows whether the workflow has any issues (i.e. is not executable yet), and a textual explanation is given when the user clicks the status indicator.

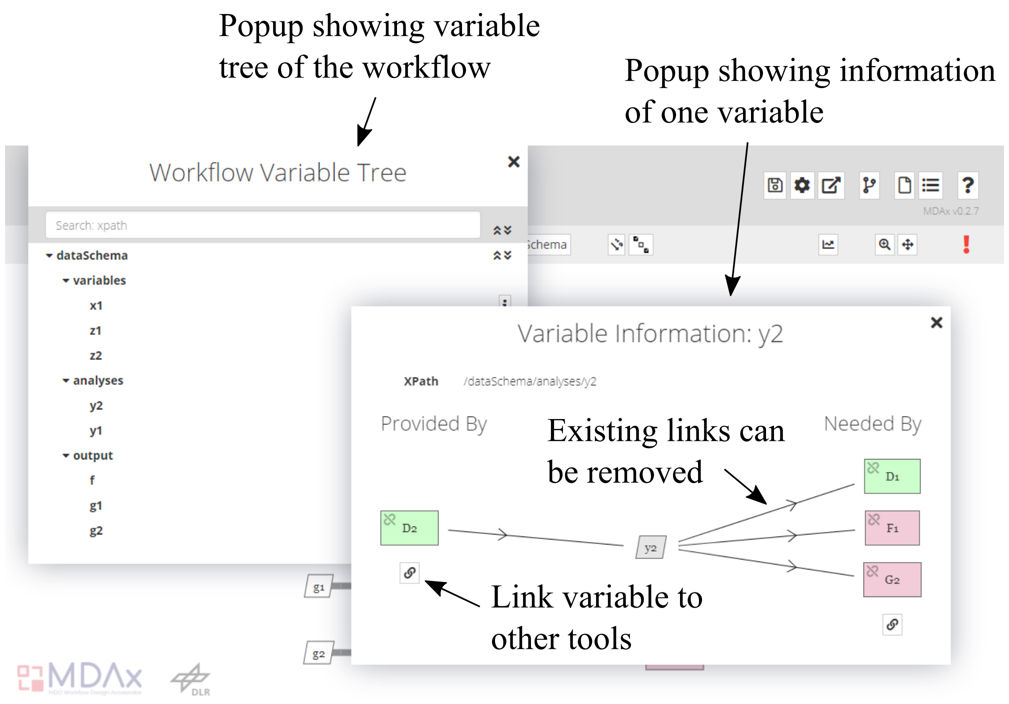
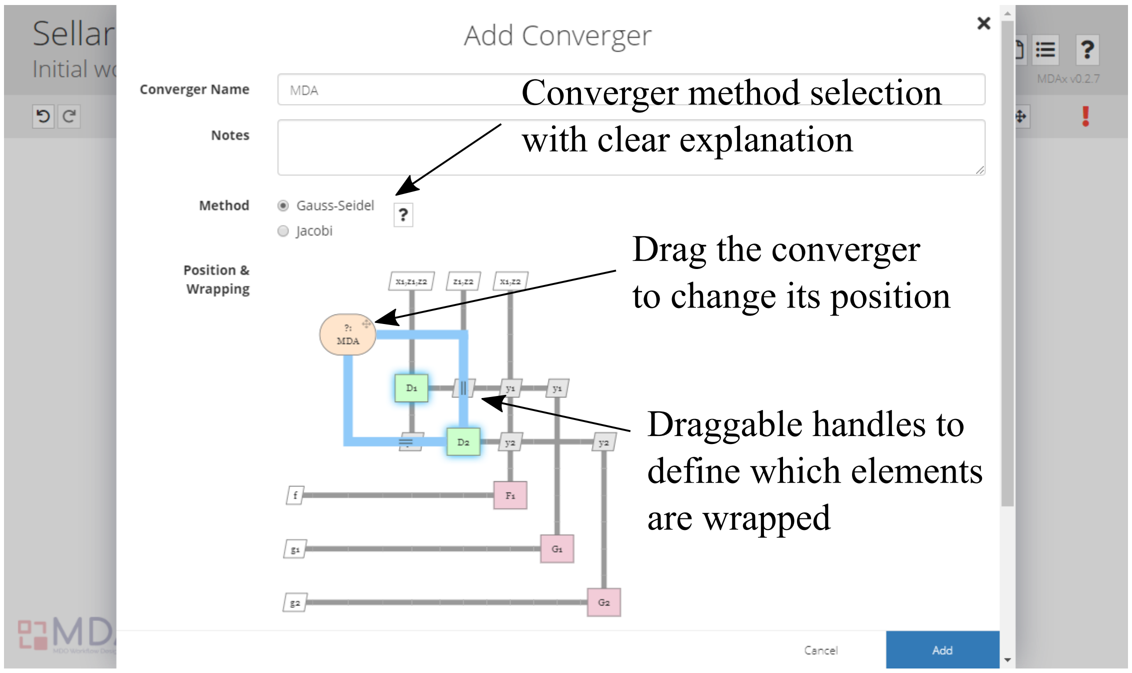


Figure 14 Popups showing the workflow variable tree and variable information. The variable tree lines can be collapsed to help display large trees. The variable information popup allows editing the connections to and from this variable: tools connecting to the variable are outputting the variable, tools connecting from the variable need the variable as input.



Sellar
Initial w
Add Converger
Converger Name
MDA
Notes
Method
Gauss-Seidel
Jacobi
?
Position & Wrapping
MDA
D₁
D₂
y₁
y₂
F₁
G₁
G₂
f
g₁
g₂
Cancel
Add
MDAx v0.2.7
Converger method selection with clear explanation
Drag the converger to change its position
Draggable handles to define which elements are wrapped

Figure 15 Modal panel showing the interface for adding a converger element. The position and wrapping behavior, taking into account nested loops, can be changed using the interactive XDSM. Similar interfaces are used for adding optimizer and Design of Experiments (DoE) elements.

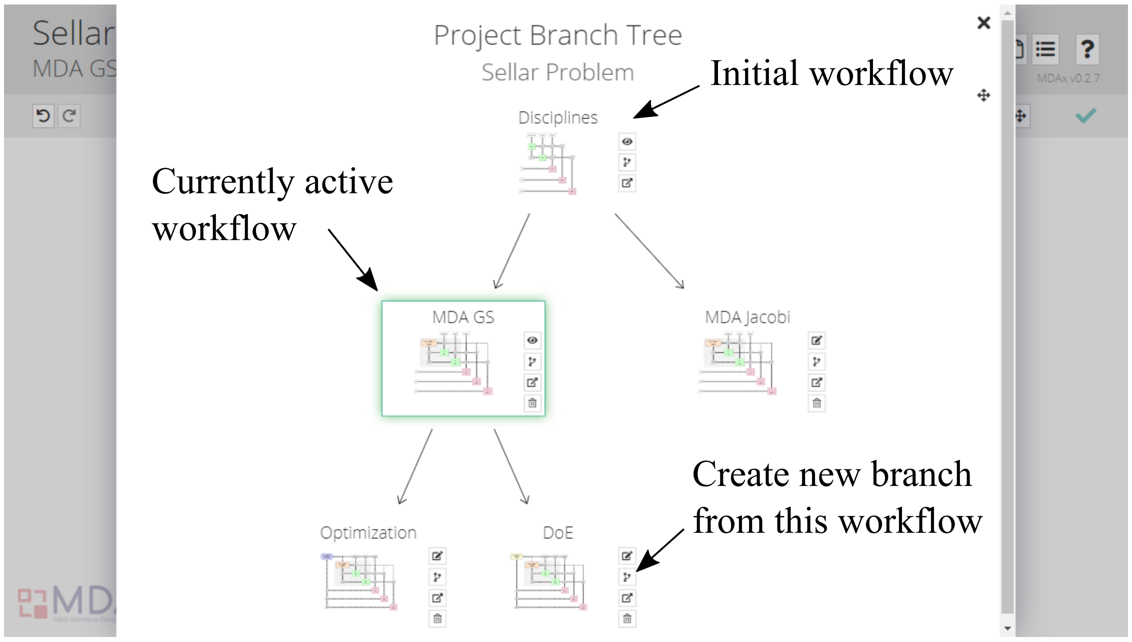


Figure 16 The workflow branch tree showing workflows derived from each other. The Sellar problem is shown for which two workflows using different converger strategies (Gauss-Seidel and Jacobi) are derived from the initial workflow. Thereafter, two design strategies (Optimization and DoE) are derived from the Gauss-Seidel workflow. The workflow branch tree helps keep track of different versions of similar workflows.

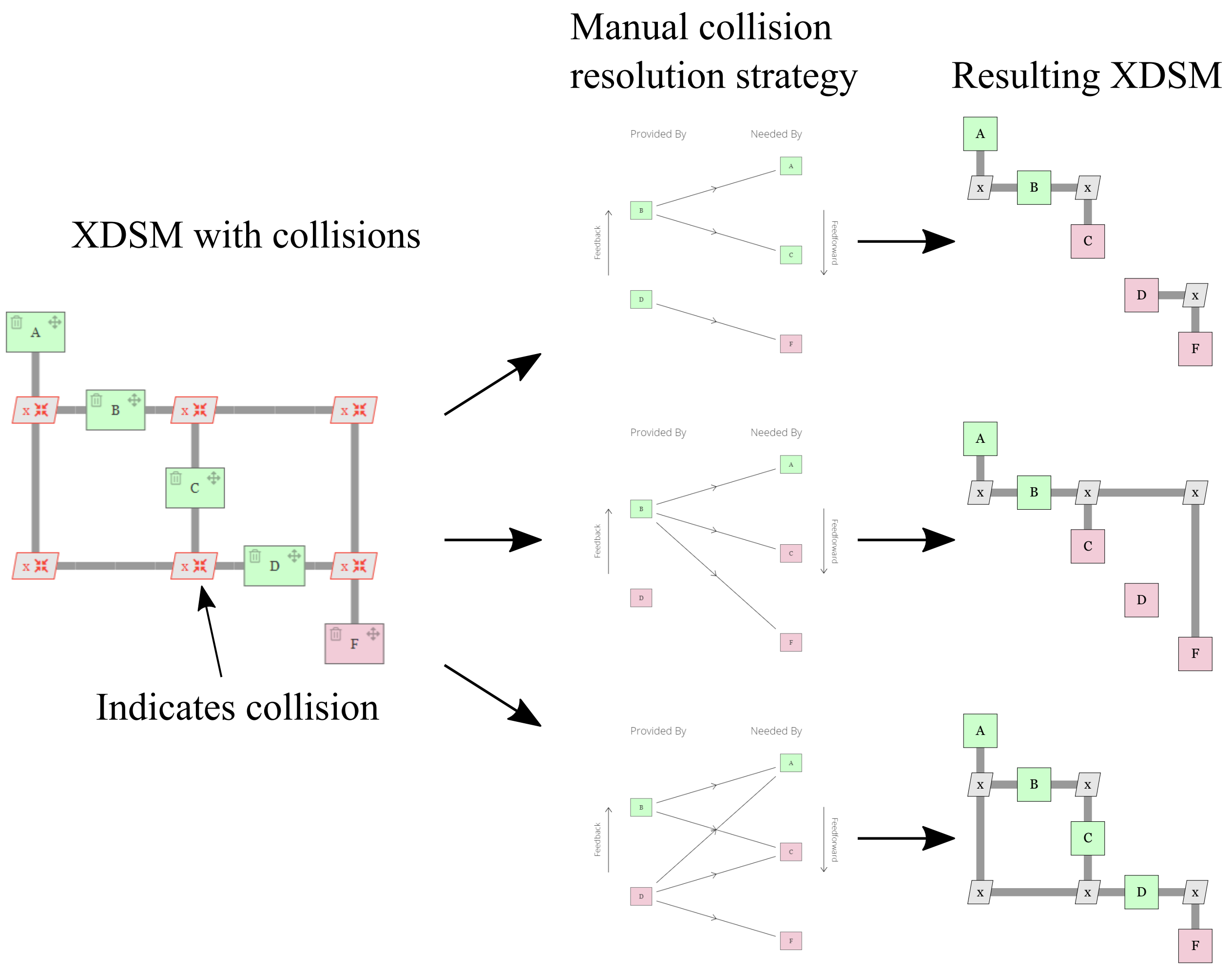


Figure 17 Selection of manual collision resolution strategies. Collisions are indicated in the XDSM by a red color and an icon with converging (colliding) arrows. By opening the variable info popup, the "edit routing" environment can be accessed. Here, the collision resolution strategy can be created (shown in the center column). Resulting XDSMs are shown on the right.

structure, where the user can easily trace which workflows derive from which. Its workings are analogous to a version control system, however no retroactive updating is supported: only leaf workflows (i.e. workflows from which no other workflows derive) can be edited. Figure 16 shows the interface implementing the branch tree.

B. Manual Collision Resolution

Collisions occur when multiple tools output the same variable. In that case, there exists and ambiguity in choosing which output is used for supplying other tools with input. MDAx offers two main strategies for resolving collisions: applying an automated collision resolution strategy (see Section V.D.4) or defining a collision resolution manually. In most cases, the automated strategy prepares the workflow well enough to give meaningful results when executed, but a manual collision resolution might be needed in special cases. Manual collision resolutions are defined per variable, and define which output of which tool is connected to the input of which tools. Only one tool output can be connected at a time, otherwise there would be a collision again.

Manual collision resolutions are defined using a table-like display of the tools of the workflow, with providing (i.e. writing output) tools on the left and needing (i.e. needing as input) tools on the right, as shown in Figure 17. The user can then connect blocks on the left with blocks on the right to construct the preferred collision resolution strategy. Not all combinations are possible:

- Routes may not overtake each other (see Figure 8).
- Routes may not cross into another driver envelope, or loop: no feedback may be provided to a tool surrounded by a feed forward connection, and vice-versa.

These restrictions result from the use of a CDS, as discussed in Section V.A. The UI makes sure that the restrictions are enforced, by not allowing prohibited connections to be made, and by highlighting possible connection targets when selecting an output block on the left.

C. Export Formats

MDAx can export the created workflows in the following formats:

PDF Saves the rendered XDSM in a PDF file.
SVG Same as PDF export, but saved as an SVG (Scalable Vector Graphics) file.
HTML Interactive visualization of the XDSM that allows users to inspect tool couplings, variable tree, and block meta data. The workflow is static and can not be modified.
RCE Exports the workflow for execution in RCE (Remote Component Environment), a Process Integration and Design Optimization (PIDO) environment developed by the DLR. Export is provided in several different modes, for increased compatibility with different execution methods of the tools.
CMDOWS The Common MDO Workflow Schema [24], a proposed standard for storing MDAO workflows that can be interpreted by various PIDO environments for execution.
Tool I/O Exports a zip file with the tool input and output files that can be used to reconstruct the created workflow. Can be helpful in the exploration phases, or to keep the tool repository up-to-date with the actual workflow in MDAx. Additionally, a file containing the complete workflow tree is exported, which can be seen as the base central schema for that workflow.
Workflow Input An empty XML file containing all variables that are workflow input: variables that are purely input (i.e. not outputted by any tool), and variables that need to have an initial value for convergence. Can be helpful for creating a starting point for workflow execution.

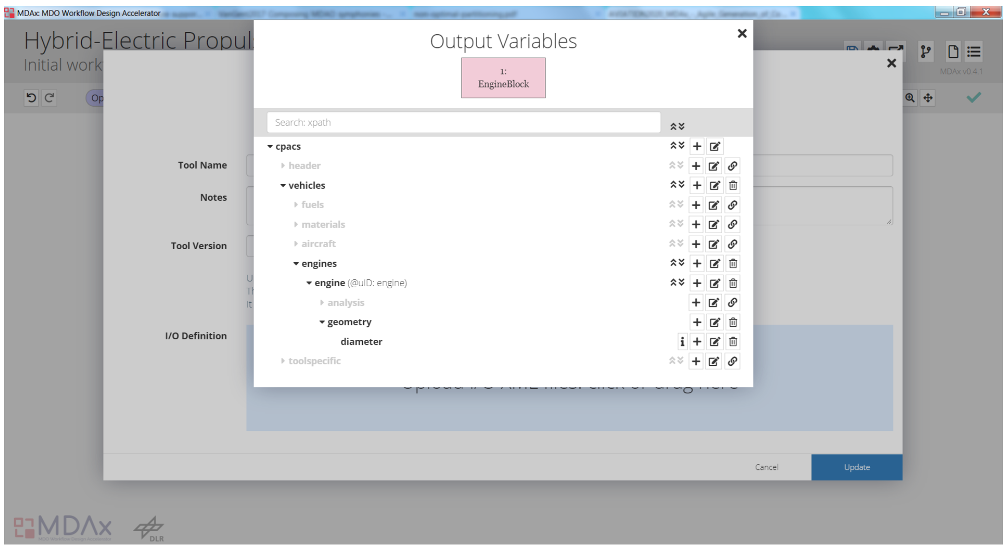
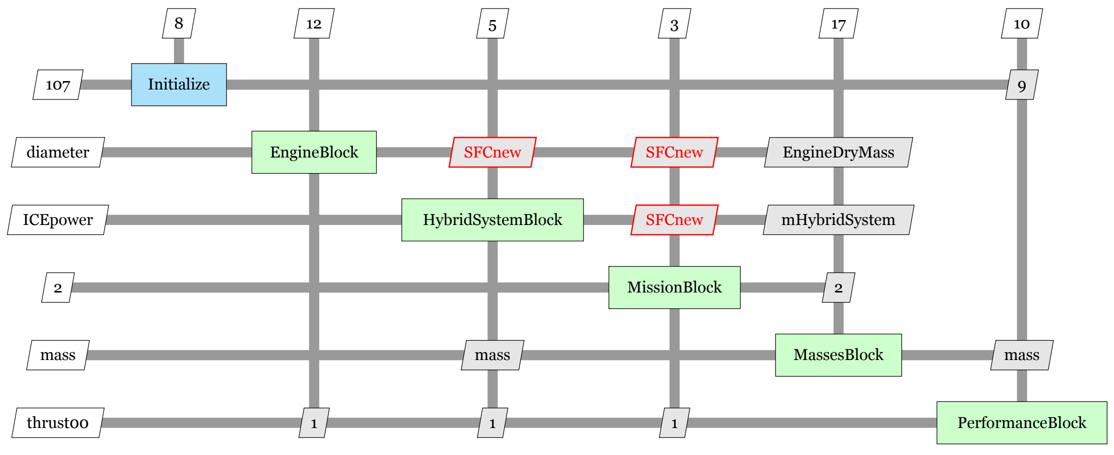


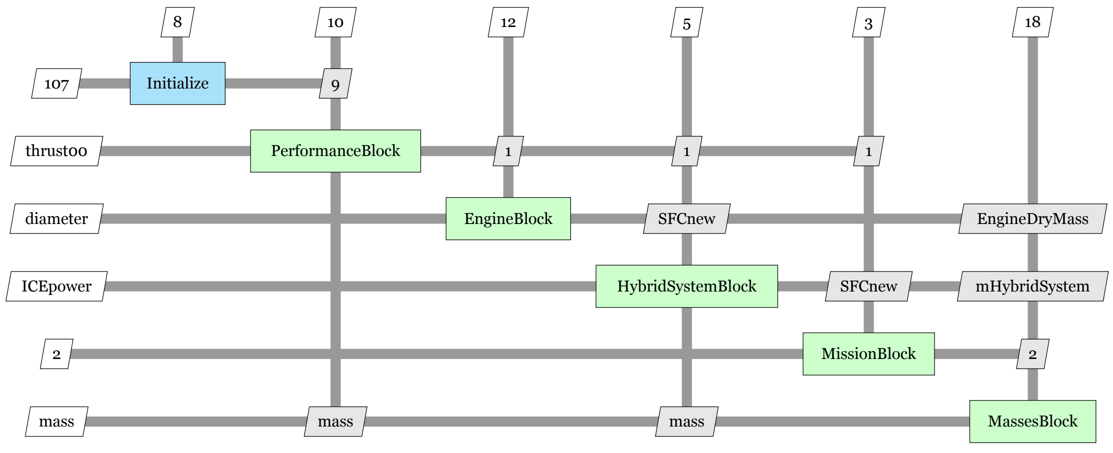
Figure 18 Variable tree menu for the definition of input and output parameters according to the used data schema. By clicking on the link icon of a tree node, the parameter is added to the outputs of the simulation tool *EngineBlock*.

Table 1 Description of simulation tools in the simplified workflow for a design of a hybrid-eletric propulsion aircraft used in Figure 19.

Simulation Tool	Description
Initializer	Generates a parametric baseline design using textbook methods.
EngineBlock	Calculates engine performance such as specific fuel consumption (*SFCnew*), geometry (*diameter*), and mass (*EngineDryMass*)
HybridSystemBlock	Determines performance and mass of the combined hybrid-electric propulsion and battery system
MissionBlock	Computes mission fuel consumption for all phases of the mission.
MassesBlock	Determines total aircraft masses (*mTOM*, *mOEM*).
PerformanceBlock	Provides engine thrust model for design conditions.



(a)



(b)

Figure 19 Example workflow for a hybrid-electric propulsion aircraft. The unconfigured workflow (a) contains parameter collisions, marked red, as well as numerous feedback connections. Using automated collision resolution algorithms and function sorting results in a collision-free workflow with minimal feedback couplings (b).

VII. Application

To demonstrate the capabilities that have been described in the preceding sections, a simplified, but realistic workflow for the preliminary design of an aircraft with hybrid-electric propulsion is used. Table 1 lists the used simulation tools and describes their role in the workflow .

Before workflow modeling can begin, the workflow components for the given MDAO problem must be built up in the form of a tool repository while following a specified CDS for consistent data exchange. To do this, MDAx implements features that allow competence specialists to upload a schema instance and easily define tool inputs and outputs in a generic sense. Figure 18 shows the selection modal for the output parameters of tool *EngineBlock*.

Once the inputs and outputs have been defined for the simulation tools required to solve the MDAO problem, and the tools have been added to the workflow diagram, the integrator can inspect the connections among them, as shown in Figure 19a, and rectify all ambiguities that may exist in the workflow. MDAx distinguishes between two types of workflow ambiguities as mentioned in Section V.A: parameter collisions and feedback couplings. Both these ambiguities are present in the workflow model of the imported tool and must be resolved before exporting it for a successful execution.

To resolve the collision parameter *SFCnew*, the user can either use manual or automatic features in MDAx, as described in Sections VI.B and V.D.4, respectively. In this case, the application of automatic resolution algorithm *Latest Variable* results in the desired configuration, wherein *HybridSystemBlock* updates the parameter that it receives from upstream.

Before applying a converger to remove feedback couplings between the tools, the workflow can be sequenced to minimize the amount of feedback couplings, which reduces the amount of parameters that must be converged when exporting and executing the workflow. This can easily be done automatically using the algorithm described in Section V.D.1 without the use of manual drag-and-drop operations, resulting in the workflow seen in Figure 19b.

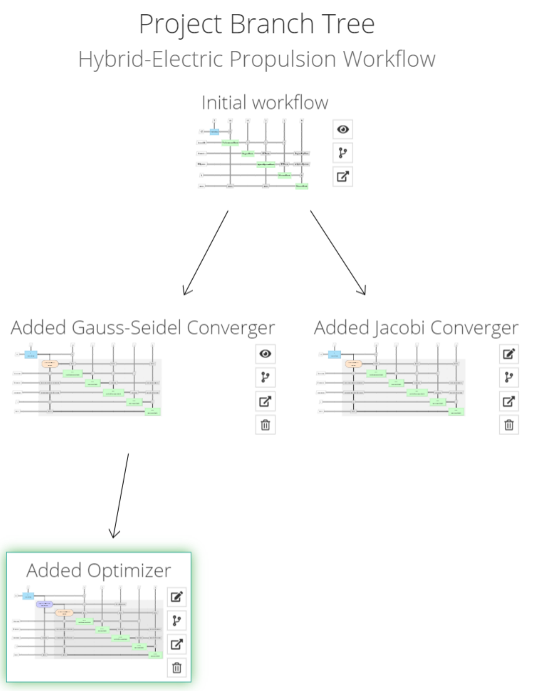


Figure 20 Workflows can be frozen and used as a basis to derive more sophisticated workflows of different configurations.

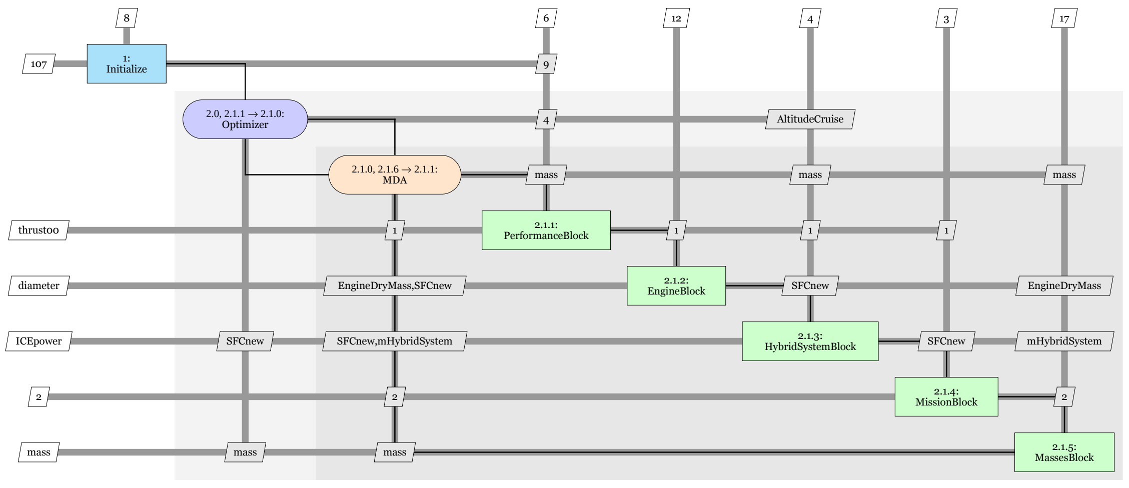


Figure 21 Optimization workflow for a hybrid-electric propulsion aircraft without parameter collisions or feedback couplings, which was achieved through an application of collision resolution algorithms and convergence mechanisms.

Having the initial configuration defined, the workflow can be branched off of to store this workflow for future references. Multiple scenarios can be implemented going off the same configuration, as seen in Figure 20, which can be useful in tracking the development of workflow models across the project life cycle. An element of safety is provided to the user by freezing workflow models that have been proven to deliver the desired results in the execution environment, and using these models for further development.

Without having to define a specific MDO architecture, the user can make ad-hoc modifications by adding convergers and optimizers to the workflow deemed appropriate for the specific use case. In this example, a Gauss-Seidel converger and Optimizer are added to the workflow to drive the optimization, shown in Figure 21.

With all ambiguities resolved and driver elements added, the workflow can be exported to run in a PIDO environment. Given that the used competence tools correctly implement their input and output interfaces, MDAx guarantees a successful execution of the complete workflow, meaning that tools are guaranteed to run if the parameters they receive have the appropriate range. Since each competence is treated as a black box, MDAx cannot guarantee specific values for the parameter that are exchanged in the workflow, but instead builds on the promise that the specified parameters will be available at execution time.

Currently, MDAx supports a direct export to RCE, as well as to the MDAO exchange schema CMDOWS which enables the execution of the workflow model in OpenMDAO [31] through the translation framework OpenLEGO [32]. An example of the exported optimization workflow shown in Figure 21 to RCE 9 is seen in Figure 22.

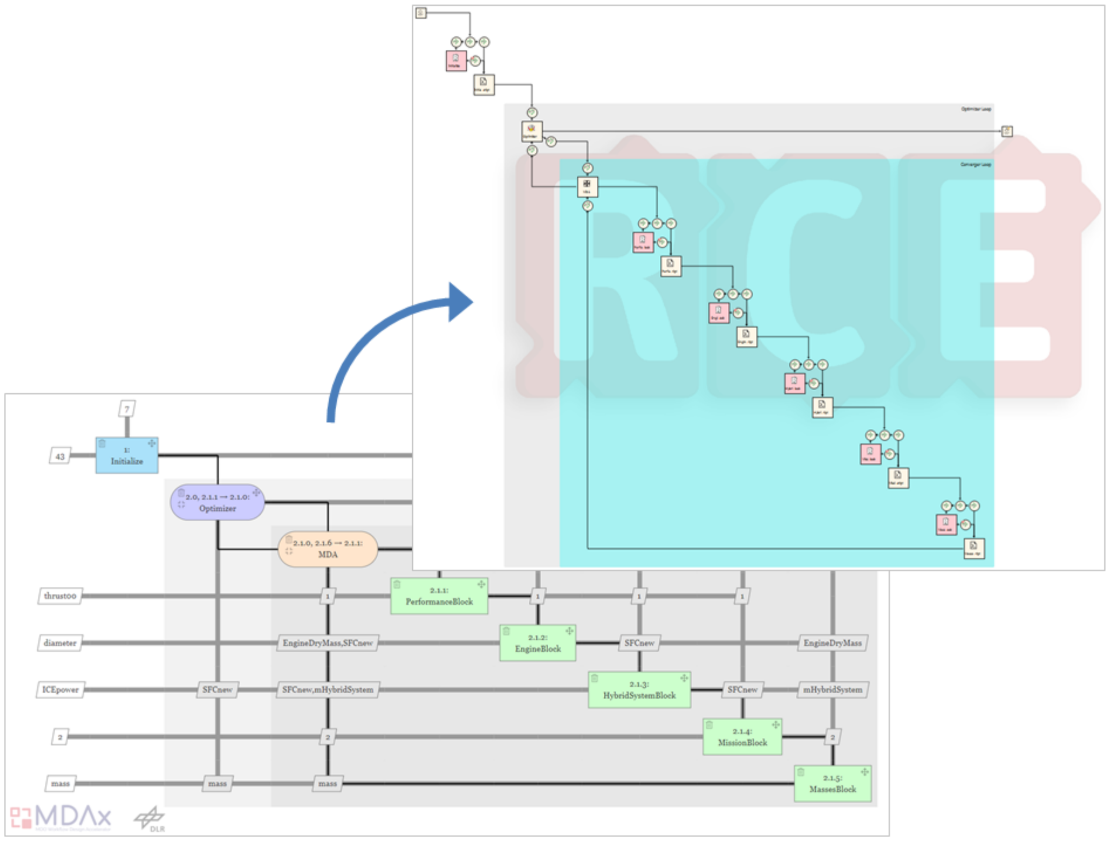


Figure 22 Exported optimization workflow model for the hybrid-electric propulsion design to RCE 9. It can be seen that the MDAx workflow model (left) directly maps to an executable RCE workflow (right). In order to launch the execution of this workflow, the driver elements must be configured and a system input file must be provided.

Because RCE uses files as the primary means to exchange data among workflow components, additional elements have to be added to the RCE workflow in order to keep the data exchange consistent with the workflow model in MDAx. For instance, only those input and output parameters that have been defined in MDAx are allowed to be provided to or received from the tool in RCE, in order to maintain consistency. Therefore, any modifications in the tool interface,

Table 2 List of use cases as described in Section III and their implementation in MDAx.

Use Case	Implementation in MDAx
Establish data schema	Variable tree modal; Basefile import
Build simulation tool interfaces	I/O file import; Variable tree links
Import and modify workflow configuration	I/O file import; Tool/Driver creation modal
Inspect connections	Canvas click operations; Variable tree modal
Apply efficient tool sequence	Canvas drag-and-drop operations; Sequencing algorithms; Redundancy elimination
Resolve parameter collision	Manual resolution modal; Automatic resolution algorithms
Formally define and communicate MDAO workflow	XDSM-centric UI
Track tool and workflow configurations	Workflow branching
Export and share workflow model	Export to RCE, CMDOWS, PDF, SVG, HTML (interactive)

like adding input parameters to make a more precise analysis, must first be implemented in the MDAx model before exporting the updated workflow. In addition to ensuring that the workflow model in MDAx stays up to date, changes to the workflow model are guaranteed to not cause workflow failure due to the presence of ambiguities such as parameter collisions.

In order to execute this workflow and run the optimization, the user is left with configuring the driver elements, such as selecting an optimization algorithm or setting converge characteristics, and providing a (possibly empty) system input file. The complete execution is controlled by RCE and does not affect its MDAx model.

VIII. Conclusion and Outlook

In the current state of affairs, simulation workflows are assembled manually through time-intensive and repetitive tasks, and modifications to a workflow (e.g. through the addition of removal of simulation tools, or modification of tools to increase analysis fidelity) entail significant effort, especially in complex systems with many interacting disciplines.

As has been shown, MDAx offers a flexible modeling environment for simulation workflows that attempts to remedy these bottlenecks. Its XDSM-centric design reduces the learning curve in workflow modeling, enabling engineers without a background in MDAO methods to participate in and drive collaborative design studies. Since all workflow components are assumed black boxes, the scope of MDAx is targeted at generic applications to cover a wide array of engineering disciplines.

The entire process from defining simulation tool interfaces, over resolving workflow ambiguities, to exporting a well-defined workflow for execution in a PIDO environment can be handled in MDAx, covering all use cases as defined in Section III. Table 2 lists these use cases and summarizes how MDAx fulfills them. The table shows that MDAx offers features for each of the posed use cases to eliminate bottlenecks in the application of MDAO methodologies, ultimately facilitating a closer collaboration between disciplinary experts in the realm of MDAO and workflow design.

The further developments of MDAx include a web integration to improve collaboration on workflow models by providing a workspace for shared tool repositories and workflow databases. Moreover, the representation of nodes that guide the execution of parts of the workflow based on predefined conditions, as well as the definition of sub-workflows are part of the ongoing research concerning workflow modeling.

A more elaborate dissemination of complex MDAO case studies will be presented in future publications, as MDAx will be used in AGILE 4.0, where multiple case studies involve large-scale modeling of MDAO processes that are assembled by cross-organizational teams.

Acknowledgments

The research presented in this paper has been performed in the framework of the AGILE 4.0 project (Towards Cyber-physical Collaborative Aircraft Development) and has received funding from the European Union Horizon 2020 Programme under grant agreement n∘ 815122.

References

[1] Sobieszczanski-Sobieski, J., and Haftka, R. T., "Multidisciplinary Aerospace Design Optimization: Survey of Recent Developments," *34th Aerospace Sciences Meeting and Exhibit*, 1996.

[2] Agte, J., de Weck, O., Sobieszczanski-Sobieski, J., Arendsen, P., Morris, A., and Spieck, M., "MDO: assessment and direction for advancement—an opinion of one international group," *Structural and Multidisciplinary Optimization*, Vol. 40, No. 1-6, 2009, pp. 17–33. doi:10.1007/s00158-009-0381-5.

[3] Belie, G., "Non-Technical Barriers to Multidisciplinary Optimization in the Aerospace Industry," *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, American Institute of Aeronautics and Astronautics, 2002. doi:10.2514/6.2002-5439.

[4] Tang, C. S., Zimmerman, J. D., and Nelson, J. I., "Managing new product development and supply chain risks: The Boeing 787 case," *Supply Chain Forum: An International Journal*, Vol. 10, Taylor & Francis, 2009, pp. 74–86.

[5] Ehret, O., and Cooke, P., "Conceptualising aerospace outsourcing: Airbus UK and the lean supply approach," *International Journal of Technology Management*, Vol. 50, No. 3/4, 2010, pp. 300–317.

[6] Ciampa, P. D., Prakasha, P. S., Torrigiani, F., Walther, J.-N., Lefebvre, T., Bartoli, N., Timmermans, H., Della Vecchia, P., Stingo, L., Rajpal, D., et al., "Streamlining Cross-Organizational Aircraft Development: Results from the AGILE Project," *AIAA Aviation 2019 Forum*, 2019, p. 3454.

[7] Ciampa, P. D., and Nagel, B., "Towards the 3rd Generation MDO Collaborative Environment," *30th Congress of the International Council of the Aeronautical Sciences*, 2016.

[8] Ciampa, P. D., and Nagel, B., "The AGILE Paradigm: the next generation of collaborative MDO," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, American Institute of Aeronautics and Astronautics, 2017. doi:10.2514/6.2017-4137.

[9] van Gent, I., Aigner, B., Beijer, B., and Rocca, G. L., "A Critical Look at Design Automation Solutions for Collaborative MDO in the AGILE Paradigm," *2018 Multidisciplinary Analysis and Optimization Conference*, American Institute of Aeronautics and Astronautics, 2018. doi:10.2514/6.2018-3251.

[10] Martins, J. R., and Lambe, A. B., "Multidisciplinary design optimization: a survey of architectures," *AIAA journal*, Vol. 51, No. 9, 2013, pp. 2049–2075.

[11] Pate, D. J., Gray, J., and German, B. J., "A graph theoretic approach to problem formulation for multidisciplinary design analysis and optimization," *Structural and Multidisciplinary Optimization*, Vol. 49, No. 5, 2014, pp. 743–760.

[12] Lambe, A. B., and Martins, J. R. R. A., "Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes," *Structural and Multidisciplinary Optimization*, Vol. 46, No. 2, 2012, pp. 273–284. doi:10.1007/s00158-012-0763-y.

[13] van Gent, I., Ciampa, P. D., Aigner, B., Jepsen, J., La Rocca, G., and Schut, J., "Knowledge architecture supporting collaborative MDO in the AGILE paradigm," *18th AIAA/ISSMO multidisciplinary analysis and optimization conference*, 2017, p. 4139.

[14] van Gent, I., La Rocca, G., and Veldhuis, L. L., "Composing MDAO symphonies: graph-based generation and manipulation of large multidisciplinary systems," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017, p. 3663.

[15] Aigner, B., van Gent, I., La Rocca, G., Stumpf, E., and Veldhuis, L. L., "Graph-based algorithms and data-driven documents for formulation and visualization of large MDO systems," *CEAS Aeronautical Journal*, Vol. 9, No. 4, 2018, pp. 695–709.

[16] van Gent, I., and La Rocca, G., "Formulation and integration of MDAO systems for collaborative design: A graph-based methodological approach," *Aerospace Science and Technology*, Vol. 90, 2019, pp. 410–433.

[17] Martin, R. C., *Clean Architecture*, Prentice Hall, 2017.

[18] Ciampa, P. D., La Rocca, G., and Nagel, B., "A MBSE Approach to MDAO Systems for the Development of Complex Products," *AIAA Aviation 2020 Forum*, 2020.

[19] Ciampa, P. D., Moerland, E., Seider, D., Baalbergen, E., Lombardi, R., and D'Ippolito, R., "A collaborative architecture supporting AGILE design of complex aeronautics products," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017, p. 4138.

[20] Nagel, B., Böhnke, D., Gollnick, V., Schmollgruber, P., Rizzi, A., La Rocca, G., and Alonso, J. J., "Communication in aircraft design: Can we establish a common language," *28th International Congress of the Aeronautical Sciences*, 2012, pp. 1–13.

[21] Beck, K., *Test-driven development: by example*, Addison-Wesley Professional, 2003.

[22] IIBA, K. B., *A Guide to the Business Analysis Body of Knowledge*, International Institute of Business Analysis, 2009.

[23] Fowler, M., Highsmith, J., et al., "The agile manifesto," *Software Development*, Vol. 9, No. 8, 2001, pp. 28–35.

[24] van Gent, I., La Rocca, G., and Hoogreef, M. F. M., "CMDOWS: a proposed new standard to store and exchange MDO systems," *CEAS Aeronautical Journal*, Vol. 9, No. 4, 2018, pp. 607–627. doi:10.1007/s13272-018-0307-2, URL https://doi.org/10.1007/s13272-018-0307-2.

[25] McKay, E. N., *UI is Communication*, Elsevier LTD, Oxford, 2013. URL https://www.ebook.de/de/product/20428661/everett_n_mckay_ui_is_communication.html.

[26] Siggel, M., Kleinert, J., Stollenwerk, T., and Maierl, R., "TiGL: an open source computational geometry library for parametric aircraft design," *Mathematics in Computer Science*, Vol. 13, No. 3, 2019, pp. 367–389.

[27] Hudak, P., "Conception, evolution, and application of functional programming languages," *ACM Computing Surveys (CSUR)*, Vol. 21, No. 3, 1989, pp. 359–411.

[28] Gebala, D. A., and Eppinger, S. D., "Methods for analyzing design procedures," *Massachusetts Institute of Technology*, 1991.

[29] Sellar, R., Batill, S., and Renaud, J., "Response surface based, concurrent subspace optimization for multidisciplinary system design," *34th aerospace sciences meeting and exhibit*, 1996, p. 714.

[30] Tidwell, J., *Designing Interfaces*, O'Reilly UK Ltd., 2015. URL https://www.ebook.de/de/product/11247302/jenifer_tidwell_designing_interfaces.html.

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

[32] de Vries, D., van Gent, I., la Rocca, G., and Binder, S., "OpenLEGO demonstration: A link between AGILE and OpenMDAO," *First European OpenMDAO workshop*, 2017.