Formulation and integration of MDAO systems for collaborative design
A graph-based methodological approach

van Gent, Imco; La Rocca, Gianfranco

DOI
10.1016/j.ast.2019.04.039

Publication date
2019

Document Version
Final published version

Published in
Aerospace Science and Technology

Aerospace Science and Technology 90 (2019) 410–433

Contents lists available at ScienceDirect

Aerospace Science and Technology

www.elsevier.com/locate/aescte

Formulation and integration of MDAO systems for collaborative design: A graph-based methodological approach

Imco van Gent *, Gianfranco La Rocca

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

ARTICLE INFO

Article history:
Received 2 December 2018
Received in revised form 27 February 2019
Accepted 21 April 2019
Available online 3 May 2019

Keywords:
MDO
MDAO
Graph theory
Collaborative design
Design tool

ABSTRACT

This paper proposes a novel methodology and its software implementation, called KADMOS (Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System), to increase the agility of design teams in collaborative Multidisciplinary Design Analysis and Optimization (MDAO). Agility here refers to the ease and flexibility to assemble, adjust and reconfigure MDAO computational systems. This is a necessary feature to comply with the complex and iterative nature of the (aircraft) design process. KADMOS has been developed on the notion that a formal specification of an MDAO system is required before proceeding with integration of the executable workflow. A thorough formulation of the system becomes essential when such system is built on the many contributions of large, heterogeneous design teams. KADMOS can automate the generation of such formulations through a graph-based methodological approach. The graph syntax and manipulation algorithms form the core content of this paper. First, a simple MDAO benchmark problem is used to illustrate KADMOS's working principles. Second, a wing aerostructural design case is discussed to demonstrate KADMOS's capabilities to enable collaborative MDAO on large problems of industry-representative complexity. Next to its graph-theoretic foundation, KADMOS makes use of two data schemas: one containing the parametric representation of the product being designed and a second to store the achieved formulation of the MDAO system. The latter enables the interchangeable use of different process integration and design optimization platforms to automatically integrate the generated MDAO system formulation as an executable workflow. The proposed approach has been estimated to be capable of halving the time typically required to set up and iteratively reconfigure a complex MDAO system, while allowing discipline experts and system architects to maintain constant oversight and control of the overall system and its components by means of human-readable dynamic visualizations.

1. Introduction

Past research indicates that Multidisciplinary Design Analysis and Optimization (MDAO) can offer huge benefits in design with estimated performance gains of 8-10% for innovative aircraft design and even 40-50% gains for designing radically new concepts [8,55]. Despite the high potential gains, MDAO is not as widely used in industry as one would expect. Both technical and non-technical barriers [1,6,46,49,50] are hampering its full exploitation.

Fig. 1 illustrates the five main stages of a generic MDAO-based development process, arranged in two main phases. On the left side one has the *formulation phase* where the MDAO problem is defined based on a set of design competences (i.e. a repository of disciplinary analysis tools). The outcome of this phase is the so-called MDAO solution strategy: a complete and formal specification, although inexecutable, of the formulated MDAO system. Different strategies, referred in literature as *MDAO architectures* [38], can be used to solve a given MDAO problem, where each strategy is based on a different approach to coordinate the involved design competences. Classic examples are the monolithic (thus including one optimizer) architectures MultiDisciplinary Feasible (MDF) and Individual Discipline Feasible (IDF) and the distributed (thus including multiple optimizers) ones, such as Collaborative Optimization (CO) [9] and Bi-Level Integrated System Synthesis (BLISS)-2000 [52]. In this work, by the term MDAO strategy, we also address other forms of multidisciplinary coordination systems without optimization, such as single point design convergence schemes (hence the MDA part of MDAO) and Design Of Experiments (DOE). The formulated MDAO solution strategy, based on whichever of the aforementioned architectures, forms the blueprint of the workflow that will

* Corresponding author.
E-mail addresses: i.vangent@tudelft.nl (I. van Gent), g.larocca@tudelft.nl (G. La Rocca).

https://doi.org/10.1016/j.ast.2019.04.039

Abbreviations

AMR	Architecture-specific Mathematical Relation
BLISS	Bi-Level Integrated System Synthesis
CDS	Central Data Schema
CO	Collaborative Optimization
CPACS	Common Parameteric Aircraft Configuration Schema
CMDOWS	Common MDO Workflow Schema
DOE	Design Of Experiments
DVD	Design Variable Dependent
DSM	Design Structure Matrix
DVI	Design Variable Independent
FDT	Functional Dependency Table
FPG	Fundamental Problem Graph
FT	Fuel Tank
IDF	Individual Discipline Feasible
I/O	Input/Output
KADMOS	Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System
MDA	Multidisciplinary Design Analysis
MDAO	Multidisciplinary Design Analysis and Optimization
MDF	MultiDisciplinary Feasible
MDG	MDAO Data Graph
MDO	Multidisciplinary Design Optimization
MPG	MDAO Process Graph
MTO	Maximum Take-Off
PIDO	Process Integration and Design Optimization
PSN	Process Step Number
QOI	Quantity Of Interest
RCG	Repository Connectivity Graph
XDSM	eXtended Design Structure Matrix
XML	eXtensible Mark-up Language
ZF	Zero Fuel

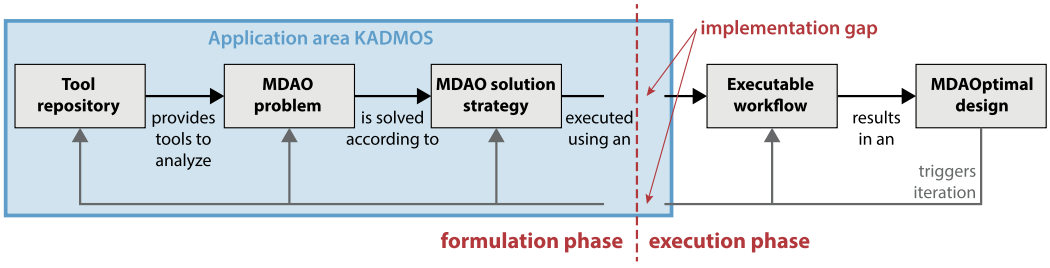


Fig. 1. Overview of MDAO terminology for the MDAO-based development process used in this paper. The process is divided in two phases: formulation (left) and execution (right). The application area of KADMOS within the overall MDAO system development process is indicated by the blue box. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

be integrated and operated in the *execution phase* of the MDAO-based development process, as illustrated in the right part of Fig. 1.

In practice, an *implementation gap* impairs a streamlined transition from the formulation to the execution phase and back. This is due to the fact that the non-formal specification of the formulated MDAO solution strategy needs to be translated into an executable system by means of a process that is largely based on manual operations. This translation of the MDAO blueprint into a software-specific executable workflow is usually done through the graphical user interface of a Process Integration and Design Optimization (PIDO) platform such as ModelCenter Integrate [69], RCE [65], Optimus [70], or a script-based interface, as in the case of OpenMDAO [26,27,63]. As a consequence, the implementation gap grows with the size of the MDAO system and the heterogeneity of the design team. The latter is defined as the diversity of the team measured by the amount of disciplinary domains, work locations, software implementation methods and roles in the development and utilization of the MDAO system.

Moreover, in a realistic design project multiple iterations of the whole MDAO-based development process are generally performed. Insight gained during or after an execution run will typically trigger changes in the tool repository (e.g. addition, removal, replacement), in the formulation of the MDAO problem (e.g. change of objective, addition or removal of constraints and design variables), or in the MDAO solution strategy (e.g. from a DOE architecture for design space exploration to another for actual optimization). This reconfiguration process of the MDAO system can be iterated until a satisfactory design is found or the project deadline has been reached. The time required for each iteration strongly depends on the ease to adjust and reconfigure the MDAO system definition and the ability to bridge the implementation gap.

In the industrial aircraft design context, both the size of typical MDAO problems and the need to leverage on distributed and heterogeneous design teams severely frustrate any collaborative MDAO ambition. In the words of Pate et al. [44] *the formulation of these problems has become increasingly complex as the number of analysis tools and design variables included in typical studies has grown. In this context the problem of determining a feasible data flow between tools to produce a specified set of system-level outputs is combinatorially challenging. Especially when complex and high-fidelity tools need to be included, the cost and time requirements to integrate the MDAO system can easily approach the cost and time requirements of creating any of the discipline analyses themselves.* A recent survey by Ciampa and Nagel [10] on collaborative MDAO-oriented research projects performed at the German Aerospace Center confirmed that 60-80% of the overall project time was used just to achieve a first executable data flow.

Next to the impact on process lead time, the current lack of agility in the formulation and integration of MDAO systems hinders both an efficient reuse of existing tools and the exploitation of new ones, as for every MDAO problem (re-)formulation, the tools have to be (re)integrated into a new workflow. Even when the same set of tools needs to be coordinated according to a different MDAO architecture, reshuffling the previously generated executable workflow is neither a trivial operation, nor fast (a notable exception being an implementation in OpenMDAO V0 [25]).

This lack of (re)configuration agility versus the potential benefits of the MDAO method with respect to existing design methods is effectively summarized in Fig. 2 by Flager and Haymaker [18], which is the result of a comparison study between the Boeing legacy design method and an MDAO-based process for the development of a hypersonic vehicle [8,55]. The main hurdle iden-

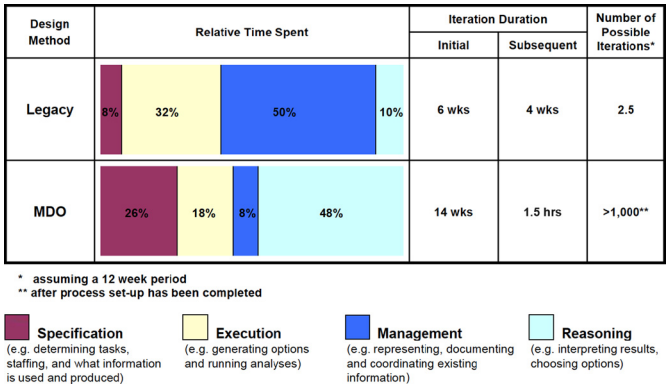


Fig. 2. Comparison of legacy and MDAO design process metrics for the design of a hypersonic vehicle [18].

tified for the MDAO approach was the long set-up time of the MDAO workflow (14 weeks), which was more than double the time necessary to deliver the first design using the legacy method (6 weeks), with the consequent risk associated to the late availability of the first results. The potential of the MDAO-based process, however, is also clear: once the set-up of the MDAO workflow is complete, an enormous amount of iterations can be performed (1000 vs. 2.5 for the legacy method), leaving much more time to the interpretation of the results rather than the coordination of the generated information.

The longer specification time required for the MDAO system specification is due to the fact that the automation of the multi-disciplinary analysis workflow (which is a necessary condition for MDAO) is only possible if all the connections between the different disciplinary analyses are created correctly, down to the smallest detail. Without the right formalization for system specification in the formulation phase, the creation of this automated workflow is a complex and time-consuming trial-and-error process. In industrial design situations, where deadlines are strict and risks need to be minimized, the resulting postponement of the initial design, in combination with the time-consuming detailed specification, poses a major hurdle. As a consequence, implementations of large MDAO systems in industry are scarce, generally tailor-made to a specific application and too difficult to reconfigure.

The formalization and methodology presented in this paper enable an agile approach for the specification and manipulation of MDAO systems of whatever size, to facilitate their transformation into executable workflows by means of any PIDO platform of choice and thereby drastically narrowing the implementation gap. Its software implementation, called KADMOS (Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System), offers MDAO system architects the ability to formulate large and complex MDAO systems, based on a repository of tools provided by heterogeneous design teams. The specific KADMOS application area within the overall MDAO-based development process is indicated by the blue box in Fig. 1. This paper illustrates and demonstrates KADMOS's ability to support all stages of the MDAO system formulation phase, from the definition of a design competence repository, up to the generation of a complete system formulation, ready to be translated into executable workflows.

This paper caters to a broad audience, reflecting the variety of people's roles involved in collaborative MDAO projects. Readers unfamiliar with the MDAO field (e.g. disciplinary specialists), or those who would like to have a deeper understanding of the motivation for KADMOS's development, should read the state of the art in Sect. 2. A top-level description of KADMOS is provided in Sections 3 and 4 and is of interest to any reader involved in collaborative MDAO. Readers specifically interested in the developed graph syntax and conditions (e.g. integrators, collaborative framework developers) should read Sections 5 and 6, while other readers can safely skip those sections. A case study concerning the collaborative design of a passenger aircraft wing is presented in Sect. 7 and is relevant to readers (e.g. executives and aircraft manufacturers) interested in evaluating the impact of the KADMOS package on a collaborative MDAO project of realistic complexity.

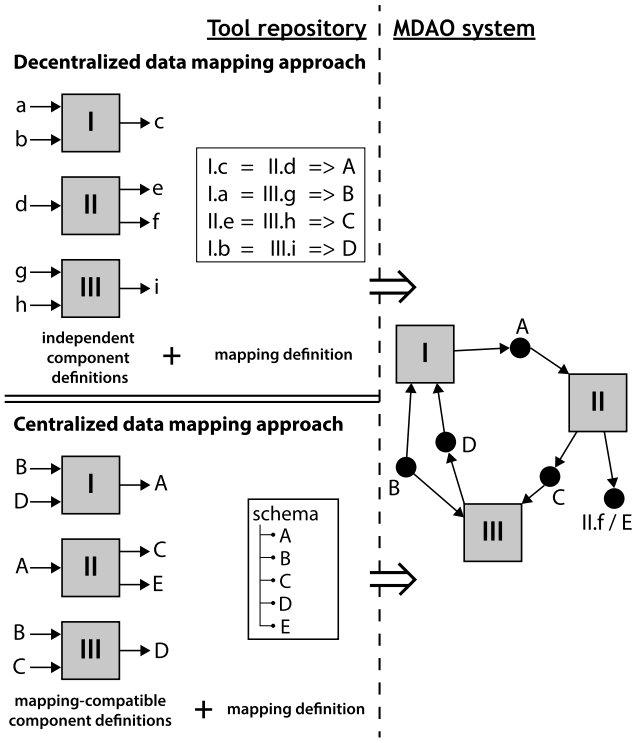


Fig. 3. Schematic summary of the two main system composition approaches found in literature.

2. State of the art

The composition, representation and manipulation of large, complex engineering design systems has been under investigation for several decades. This section describes the state of the art, from which the functional requirements for KADMOS are derived in Sect. 3.

2.1. System composition

The composition of the MDAO system, i.e. the way its different elements are interconnected into a computational system, is an unreported topic in most work. Though it might seem trivial, depending on the approach, the assembly of all components can be a cumbersome task for large MDAO systems, with a big impact on the time spent in the formulation phase. This is especially true when a large, heterogeneous design team is involved. A key assumption affecting the system composition is whether variables are related to a single component or shared and/or coupled to multiple components. Two basic approaches for system composition can be found in literature: decentralized data and centralized data mapping, both summarized in Fig. 3. While both approaches would eventually result in the same MDAO system, the role and responsibilities of system integrators and tools providers differ for each composition method.

Tosserams et al. [54] have a decentralized data mapping approach in their Ψ language for problem partitioning. In their system each component can be defined independently and it is the responsibility of the component provider to define local and global

input variables. When the components are assembled, the system integrator has to manually map the global inputs/outputs (I/O) of all components, that is to identify which globally defined variables are actually referring to the same value. A similar approach can be found in object-oriented frameworks such as πMDO [39] and OpenMDAO. In OpenMDAO the user can provide different naming conventions for the same variable in each component and then explicitly state that the components are actually referring to the same variable by connecting them.

The decentralized data mapping approach leads to a high workload for the system integrator, proportional to the size of the MDAO system. A clear advantage of this approach is the freedom left to the various tool providers, as they can decide their own variable names as they see fit. Hence, the MDAO system integrator is assigned extra responsibilities, in exchange of a non-intrusive approach towards the tool providers. Furthermore, as connections are handcrafted by the integrator, a valid MDAO system can be instantiated directly, without the need to fix problematic nodes or connections post hoc.

In the centralized data mapping approach, a system-wide naming convention is used to establish component connectivity. Alexandrov and Lewis [3,4] use this method in their linguistic approach called REMS; component providers define their I/O variables according to a shared dictionary of variable names, which needs to be expanded with each new system component that introduces new variables. The integrator is thus relieved of the mapping burden, at the expense of an extra management task for the dictionary and a limited freedom in variable naming for the tool providers. This approach can also be used in OpenMDAO by promoting variable names to a common shared variable name within the model, thereby automatically connecting these variables. Also Hoogreef [31] uses the centralized data mapping approach in his ontology-based MDAO-support framework InFoRMA, in which the user builds and assembles the system using an interactive N^2 chart to add components and define couplings.

Another variant of the centralized approach would be to adopt a Central Data Schema (CDS). The adoption of a CDS avoids the issue of updating the dictionary (REMS) or list of promoted names (OpenMDAO) for each MDAO system definition. Nagel et al. [43] have proposed a standard XML-based schema for aircraft design, called Common Parametric Aircraft Configuration Schema (CPACS) [60]. In practice, CPACS provides an extensive predefined standard dictionary, which is meant to contain all the typical I/O data of the analysis tools used in conceptual and preliminary aircraft design.[1] When using CPACS, each disciplinary tool in the MDAO system takes a CPACS file instance as input and has to write its results to a new CPACS file. All data is then gathered by merging the different complementary CPACS files. Thus, once all component providers have "made the investment" of rendering their tool CPACS-compatible, assembling even large executable MDAO systems becomes a relatively easy task for the integrator.

The CPACS definition has so far been used in multiple collaborative projects on aircraft MDAO [24,35,41,42] and has proven its value facilitating the assembly of large and distributed MDAO systems. Whereas CPACS is aircraft specific, a CDS would prove its value in any other application domain, given a schema is made available a priori. The approach is gaining momentum in the MDAO community, with a similar type of schema under development in the wind energy domain by Dykes et al. [14]. This schema also uses file I/O to exchange data (YAML-based instead of XML), however, conceptually a CDS approach does not necessarily depend on file-based data exchange (e.g. a string-based naming convention or Python dictionary).

Although schema-centered rerouting of the tool connections facilitates workflow assembly, it comes with two limitations. First, one can easily loose oversight on the system's connectivity. For example, if a tool providing the values for certain CPACS data nodes is removed, it is difficult to determine whether some other tool is affected by the lack of those input data. Second, contrary to the handcrafted connections in the decentralized approach, the indirect component couplings of the schema-based approach can easily lead to a system with problematic variables and connections. Hence, a dedicated formulation system is required to provide system oversight, check for problematic variables and connections, and fix them. This formulation system is proposed in this paper and provides the missing link between the handcrafted decentralized approach and the automatic centralized manner of composing MDAO systems.

2.2. System representation

System representations are usually motivated by their intended use, such as performing specific machine operations (e.g. partitioning) or providing a human-readable overview of the MDAO system. Well-known human-readable representations are the N^2 chart [34], the Design Structure Matrix (DSM) [53] and the Functional Dependency Table (FDT) [57]. The DSM approach, which only specifies the data coupling between different components, was extended by Lambe and Martins [33] into the eXtended Design Structure Matrix (XDSM) to include also information concerning the process to be executed. Today, the XDSM notation can be considered a de-facto standard within the MDAO community for providing human-readable overviews of MDAO workflows, independent from any proprietary PIDO tool formalization. Since the readability of an XDSM as a static document degrades with the size of the represented computational system, web-based dynamic XDSM visualization tools have been developed by Gazaix et al. [19] and Aigner et al. [2], with their respective developments of the XDSMjs [67] package and VISTOMS [66].

All commercial PIDO platforms provide a graphical user interface to assemble and display an executable MDAO workflow. In the background these workflows are represented in a machine-interpretable format, often based on XML or proprietary standards. Object-oriented frameworks for MDAO have been described in scientific literature that create machine-interpretable representations of given computational systems by constructing programming objects [25,39]. This approach has the major advantage that a software package can be created that contains an integrated collection of methods to execute the MDAO system the best way possible (e.g. using different optimization strategies or taking benefit from parallel computing methods). The downside of this representation is that it forces the user to employ the same platform for both the formulation and the execution of the computational workflow. However, most of these platforms are rather geared towards the execution of the computational system and offer limited support in the formulation phase. For example, they are not able to provide adequate human-readable overviews of large and complex systems, which is a deal breaker when formulating MDAO systems in a collaborative environment.

In other work, machine-interpretable representations of MDAO systems are created by defining a language. Examples of these linguistic representations are the χ [17], Ψ [54] and REMS languages. These languages allow an integrator to compose the MDAO system in a relatively straightforward way (once familiar with the syntax), so that algorithms built upon the language can perform system manipulations, like problem decomposition and coordination. A radically different representation of MDAO systems was investi-

[1] Any missing data in the schema can be temporarily stored under a "free" element. If such data appear more often than a single project instance, the structure under the free element can be suggested for adoption in the official schema.

gated by Hoogreef [31], who makes use of ontologies to model and store MDAO systems in his InFoRMA platform. By representing the MDAO system as a 'meaningful' (semantic) web of data, semantic reasoning engines [7] are used in InFoRMA to assess and manipulate the system.

Pate et al. [44] realized that many of the MDAO system representations can be brought back to the basic mathematical construct of (directed) graphs. This is true for some languages such as REMS and InFoRMA's ontologies, but also for purely graphical representations like the N^2 chart and the DSM. Based on this realization, Pate et al. defined a graph-based syntax to store and manipulate MDAO systems. Three types of directed graphs are defined in Pate et al.'s graph syntax that map one-on-one to the three stages of the MDAO system development process illustrated in Fig. 1. These are the maximal connectivity graph to represent the tool repository, the Fundamental Problem Graph (FPG) to represent the statement of the MDAO problem, and the problem solution graph to represent the MDAO solution strategy adopted to solve the fundamental problem. These graph definitions and their manipulation algorithms are mainly focused on the challenge of finding possible combinations of design and analysis tools to solve a given MDAO problem. Thus their main objective is the automatic determination of the FPG, starting from the larger maximal connectivity graph.

2.3. System manipulation

Apart from the (X)DSM visualizations, all of the representations discussed in the previous section are machine-interpretable and are set up to enable various forms of computerized manipulation of the MDAO system. These manipulations are generally related to the link between subsequent stages in the formulation phase (Fig. 1) and aim at automating (part of) the work required to advance from one stage to the other.

The first link is between the tool repository and the MDAO problem formulation. In general, automation in the first link involves the creation of methods to 1) identify potential design variables, objective values and constraints, 2) mark them as such, 3) find valid combinations of tools to analyze the problem and finally 4) to remove unnecessary functions (tools) and variables. The work of Pate et al. [44] addressed in Sect. 2.2 focuses specifically on this area. They offer a graph syntax and suggest algorithms to determine different possible MDAO problems based on a tool repository definition. Also in the REMS language, the necessary manipulations to achieve the MDAO system formulation are based on the full graph of available functions and coupled variables.

The second link, between the MDAO problem and the MDAO solution strategy, is more complex and handled in many different ways in earlier work. In this link the MDAO problem has to be decomposed to make it computationally tractable. This is generally achieved by means of partitioning and clustering methods. The Ψ language is an example of a manipulation language specifically developed for system decomposition. Other system decomposition methods are available in literature, based on DSM [28,47,58] and FDT [5,40].

After decomposition, a solution strategy has to be imposed on the MDAO problem to coordinate its execution. As anticipated in the introduction, different MDAO architectures exist in literature, each one providing a different recipe to coordinate the computational system. Martins and Lambe [38] have provided an extensive review of the most commonly used, including both monolithic (e.g. MDF, IDF), and distributed types (e.g. CO, BLISS-2000). The fact that the same MDAO problem can be solved using different solution strategies was one of the reasons to develop the MDAO frameworks πMDO, OpenMDAO, and InFoRMA, so that different architectures could be tested quickly on the same problem by automatically reconfiguring the executable workflow.

The third link between the MDAO solution strategy and the executable workflow concerns with the implementation gap discussed above. Past research frameworks have either tightly integrated the executable workflow in the same formulation framework, as is the case for πMDO and OpenMDAO, or worked on methods to establish an automated link to external PIDO platforms. The latter is the case of InFoRMA, where a built-in mechanism parsing the system knowledge base creates the matching executable workflow in Optimus. Ψ provides an export module to enable the creation of executable workflows in MATLAB [68].

Despite these frameworks that can provide executable workflows based on different solution strategies, an implementation gap is still experienced when performing collaborative MDAO within a large, heterogeneous team, where different PIDO systems or computational environments and optimization packages are at hand. A framework like OpenMDAO offers excellent possibilities to set up single-user executable MDAO workflows, but is limited in its capability to support collaborative MDAO system formulation and to execute workflows on a distributed server environment. Some PIDO platforms available on the market allow the integration of multilevel and distributed MDAO workflows through manual "drag&drop" manipulations via their interface, but, at date, none provides actual formulation capabilities [12,30].

3. Requirements for the KADMOS system

From the review of the state of the art it can be concluded that while multiple solutions exist for the execution of even large computational systems, there is lack of adequate support in the formulation phase. Methods have been developed to specifically address some of the stages in this phase, but no comprehensive solutions addressing the whole MDAO system formulation in a collaborative environment have been found. While some methodologies have been proven able to close the implementation gap by means of dedicated interfaces with a PIDO tool, none provide the flexibility to choose between different integration platforms. Several graph-based representation and manipulation approaches have been proposed by various authors, which have proven very effective within the limited scope of some of the formulation stages. The KADMOS system presented in this paper aims at filling this gap in the state of the art, by leveraging on the high potential of graph-based representation and manipulation methods, to deliver a neutral, open-source platform, specifically targeted to the development of large, distributed and collaborative MDAO systems. To this purpose, the following top-level requirements were set:

1. **System composition:** The system should be based on the CDS approach for I/O data exchange.
2. **System representation:** The syntax should be based on the graph-theoretic foundation for MDAO systems initiated by Pate et al., but should also provide the human-readability of XDSMs.
3. **System manipulation:** KADMOS should support automated formulation of MDAO solution strategies for MDA, DOE and MDO architectures. MDO-type architectures should include the previously mentioned monolithic and distributed approaches, at least including the MDF, IDF, CO and BLISS-2000 architectures.
4. **PIDO platform independence:** The MDAO solution strategy formulated with KADMOS should be portable to a range of PIDO platforms, while maintaining complete independence from them.
5. **Controlled automation:** The system should automate all the repetitive, non-creative tasks necessary to advance from one

formulation stage to the other, while keeping the system architect in control of all settings and strategic decisions that require engineering judgment (e.g. to define the problem, to pick the architecture).

6. **Tool heterogeneity:** The system should support a broad range of design tool types ranging from simple mathematical relations, to more complex surrogate model relations, up to complex disciplinary tools to be executed as black boxes on separate server domains because of intellectual property constraints.
7. **Scalability:** The system should be able to handle systems of any size and complexity.

4. KADMOS graph types

The following graphs represent the fundamental means used in KADMOS to store and manipulate the MDAO system throughout its entire formulation process:

Data graphs

Repository Connectivity Graph (RCG) Based on a repository of CDS-compatible design and analysis tools, a graph can be created that links all the I/O variables and represents the tool repository.

Fundamental Problem Graph (FPG) The FPG is an enriched (i.e. containing additional attributes) subset (i.e. due to removal of unnecessary functions and variables) of the RCG. It is created by performing graph manipulations on an RCG to define a graph that represents a valid (in terms of the strict KADMOS graph conditions discussed in the next sections) MDAO problem. This graph can then be used to impose an MDAO architecture on it.

MDAO Data Graph (MDG) The MDAO solution strategy stage is represented by two graphs: a data and a process graph. The MDG is the data graph that stores the data exchanged by the executable blocks and the CDS nodes that are required to solve the MDAO problem, according to the selected architecture. The executable blocks in the MDG include both the repository tools from the FPG and the *architecture elements*, necessary to implement the MDAO architecture at hand, such as convergers, optimizers, and Architecture-specific Mathematical Relations (AMRs) (i.e. consistency constraint function).

Process graphs

MDAO Process Graph (MPG) This graph does not contain any data node, but only the executable blocks from the MDG and the specification of their execution sequence.

The general KADMOS graph syntax will be described in the next section. The detailed description of the individual graphs is provided in Sect. 6, supported by a small illustrative example. KADMOS [61] is implemented as a Python package. All graphs are subclasses of the `DiGraph` class from the NetworkX [29] package; see the class diagram in Fig. 4.

Although there is a conceptual resemblance between KADMOS graphs and those proposed by Pate et al, there are notable differences in the graph syntax, as well as in the scope and implementation of the entire MDAO support system. The most significant difference with respect to Pate's graph-based methodology concerns the scope. Whereas Pate's graph formulation is focused on the transition between the first and second stage of the MDAO development process, KADMOS has a broader scope including the transitions to stages three and four. The inclusion of these additional stages required a more sophisticated definition of the syntax and the graph-theoretic conditions the graphs have to satisfy. In addition, KADMOS is based on the CDS approach for system composition, which further differentiates the syntax and conditions. Finally, the KADMOS syntax covers both monolithic and distributed MDO architectures, whereas the latter were not covered by Pate's approach.

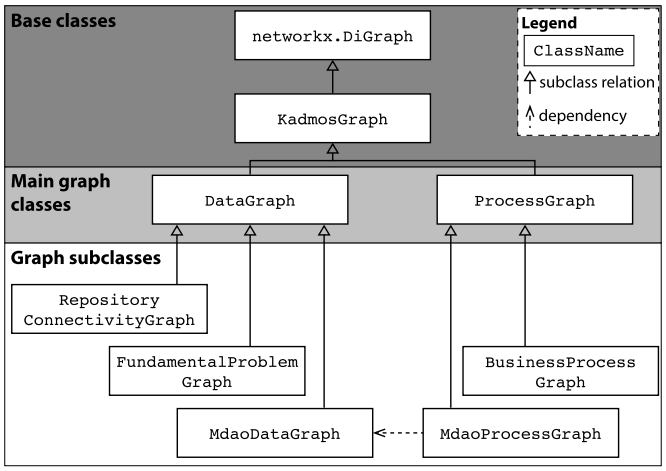


Fig. 4. Class diagram of the KADMOS package. N.B. `BusinessProcessGraph` class is not discussed in this paper.

5. KADMOS graph syntax and main graph classes

This section provides a formal definition of the graph syntax adopted in KADMOS. The two main graph classes (Fig. 4) are defined as well. The syntax follows the notation of Diestel [13] and Pate et al. [44]. Key concepts of graph theory are briefly revisited here for convenience, followed by the definition of nodes (Sect. 5.1), edges (Sect. 5.2), and main graph classes (Sect. 5.3) used in KADMOS.

A graph G is built using a set of vertices V (or nodes) and a set of edges E (also called connections):

$$G = (V, E)$$

in which $E \subseteq [V]^2$, meaning that the elements of E are two-element subsets of V. All KADMOS graphs are a special type of graphs, called directed graphs (or *digraphs*),[2] where E contains a set of ordered pairs to indicate the edge direction. The node v would be connected to the node w with the edge $e = (v, w)$. Every edge in a digraph has an initial vertex $\text{init}(e)$ and a terminal vertex $\text{ter}(e)$. The set of edges going out of v are denoted with $E^+(v)$ and the total amount of edges going out of v is called the *outdegree* of the node and is denoted with $\delta^+(v)$. Similarly, the set of incoming edges are denoted as $E^-(v)$ and the *indegree* is denoted with $\delta^-(v)$.

An example digraph is shown in Fig. 5. This graph would be defined with the two sets:

$$V = \{a, b, c, d, e, f\}$$

$$E = \{(a, a), (a, d), (b, a), (b, e), (d, b), (d, c), (e, b)\}$$

A *loop* in a digraph would be defined by an edge with $\text{init}(e) = \text{ter}(e)$. In KADMOS graphs these kind of loops are not allowed, meaning that only *simple digraphs* are used. A *looped pair* is allowed in simple digraphs. Looped pairs are defined as pairs of nodes $\{v, w\}$ for which there is an edge in both directions, hence $(E^+(v) \supset e \mid \text{ter}(e) = w) \wedge (E^-(v) \supset e \mid \text{init}(e) = w)$. See the pair

[2] More specifically, most graphs are directed *cyclic* graphs, though acyclic graphs are also possible and supported.

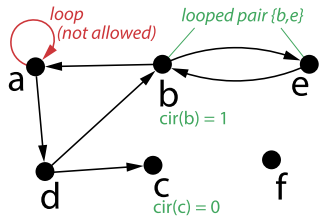


Fig. 5. Example of a directed graph illustrating some key concepts.

$\{b, e\}$ in Fig. 5. The amount of such looped pairs of one node v with respect to its neighbors is referred to as the *circularity index*: $\text{cir}(v)$. This is one of the fundamental KADMOS extensions to Pate's syntax.

A *path* $Q = (V, E)$ from v_0 to v_k in graph G is a subgraph of G ($Q \subseteq G$) with $V = \{v_0, v_1, ..., v_k\}$ and $E = \{(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)\}$. See as example the path with $V = \{e, b, a, d, c\}$ in Fig. 5. A *cycle* C is a path for which $v_0 = v_k$, e.g. the cycle with $V = \{b, a, d, b\}$ in Fig. 5.

In an acyclic directed graph the nodes can always be ordered in such a way that one moves forward when following the edges. This is called a *topological ordering* of the graph. Hence, if a directed graph has the topological ordering of the vertices $\langle v, w, \ldots \rangle$, it means that for every edge in the graph the initial vertex also comes before the terminal vertex in the ordering (e.g. if $\exists(v, w) : v$ comes before w in the ordering).

To combine graphs, a notation for the union of a set of sets is required. If we define I to be a non-empty set such that for each $i \in I$ there is a corresponding set A_i, then the set of sets $\mathcal{A} = \{A_i \mid i \in I\}$ is called an indexed family of sets with index i and indexing set I [51]. The union of this family of sets can be denoted in different ways:

$$\bigcup_{i \in I} A_i = \bigcup_{A \in \mathcal{A}} A = \{x \mid x \in A \text{ for some } A \in \mathcal{A}\}$$

Finally, the size of a set B is called the *cardinality* and is denoted by $|B|$. A set difference is denoted as $A \setminus B = \{x \in A \mid x \notin B\}$.

5.1. Node definitions

The definition of nodes in a KADMOS graph is done using attributes. The different node attributes are listed in Table 1. The attribute values are used throughout the whole package to inspect and manipulate the graphs. The main attribute of a node is its *category*, denoted as $\text{cat}(v)$, which can have the two following values:

function (v_f): operators, also called executable blocks. Every possible operator is defined as a function node in KADMOS, e.g. a mathematical expression, an analysis tool, an optimizer, a converger. The subset of function nodes in a graph G is denoted by $V_f = \{v \in V \mid \text{cat}(v) = \text{function}\}$.

variable (v_v): elements from the CDS. These variable nodes can represent different variable types, such as scalars, vectors and matrices. The subset of variable nodes in a graph G is denoted by $V_v = \{v \in V \mid \text{cat}(v) = \text{variable}\}$.

All nodes can get an *instance* specified. A node instance attribute is used to allow multiple instances of a node that refer to the same tool in the tool repository, or to the same element in the CDS. A node instance is an integer value denoted by $\text{ins}(v)$, where the default instance value is zero. A practical example of the use of node instances for a variable is the situation where two functions (e.g. A and B) have the same variable b as output, but the values written by those tools are required at different moments in the MDAO system execution. In such a case, it could be required that A first determines the value of b ($\text{ins}(b) = 0$) which is used by some other tools, and B later overwrites this value ($\text{ins}(b) = 1$) to be used by other tools in the MDAO system. Hence, the node instances are required in the graph to be able to indicate which functions are using and producing which node instance.

Another key attribute of both function and variable nodes is the role they play in a certain graph. Two types of roles are defined in KADMOS graphs: the *problem role* (denoted $\text{pr}(v)$) and the *architecture role* (denoted $\text{ar}(v)$). The problem role is used in the FPG to indicate special variables, such as design variables, constraints, objective, and quantities of interest (QOIs). The problem role of a function in the FPG is related to its position within the MDAO system and will be further explained in the next section. The architecture roles are used in the MDG and MPG. These roles indicate special variable and function nodes required by MDAO architectures, such as initial guesses, copy variables, optimizers and consistency constraint functions. A full list of architecture roles is provided in Table 3 and is further discussed in Sect. 6.

An attribute specific for function nodes is the *mode* attribute, denoted by $\text{mode}(v)$. This attribute is used to indicate that the same tool from the repository can be executed in different operational modes (e.g. an aerodynamic analysis tool can operate both in viscous or inviscid mode), using different I/O branches in the CDS. By using the keyword 'mode' the I/Os of the different function modes can be filtered automatically, as shown in Fig. 6. The advantage of using the mode attribute is that a tool with multiple execution modes can still be stored in the repository as one tool, with only one mapping definition to the CDS.

Another function-node-specific attribute is the ordered set containing the *process step numbers* (PSNs), denoted by $\text{psn}(v)$. These step numbers are integers used to specify the execution order of the function nodes in a process graph (Fig. 7b). The process numbering adheres to the XDSM convention. A function node can have multiple PSNs to allow for cycles. The maximum and minimum PSNs present in a graph G are denoted by respectively $\text{maxpsn}(G)$ and $\text{minpsn}(G)$.

5.2. Edge definitions

Two edge types are defined, called data and process edge and belonging, respectively, to the two main graph classes `DataGraph` and `ProcessGraph`.

5.2.1. Data edge (e_d)

This edge represents the relation between a variable node and a function node. All data edges in a graph are denoted by $E_d = \{e \in E \mid \text{cat}(e) = \text{data}\}$. A data edge is always defined using one variable node and one function node. If $\text{ter}(e_d) \in V_f$ then the edge is an *input edge* and, vice versa, if $\text{ter}(e_d) \in V_v$ then the edge is an *output edge*.

5.2.2. Process edge (e_p)

This edge represents the relation between function nodes in a process graph. The subset of process edges in graph G is $E_p = \{e \in E \mid \text{cat}(e) = \text{process}\}$. Just like function nodes, the process edge can have a PSN attribute, denoted by $\text{psn}(e)$. For process edges the PSN is not an ordered set, but a single integer value, since each process edge can only represent a single step (Fig. 7b).

5.3. Main graph classes definitions

The main KADMOS graph classes `DataGraph` and `ProcessGraph` (see class diagram in Fig. 4), are defined using the node and edge definitions from the previous sections.

Any instance of the `DataGraph` class is a digraph $D = (V_D, E_D)$ complying to the following conditions:

Table 1
List of attributes for graph nodes and edges.

Attribute name	Notation	Valid for	Typical values
category	cat()	$V \wedge E$	function, variable, data, process
subcategory	sct()	V	input, coupling, collision, hole, sink (see Table 2)
instance	ins()	V	0, 1, 2, 3
problem role	pr()	V	design variable, objective, coupled, post-coupling
architecture role	ar()	V	initial guesses, copies, optimizer (see Table 3)
mode	mode()	V_f	viscous, inviscid, 1, 2, 3, A, B, C
process step number	psn()	$V_f \wedge E_p$	{0, 7}, {1} for V_f or 1, 2, 3, 4, 5 for E_p

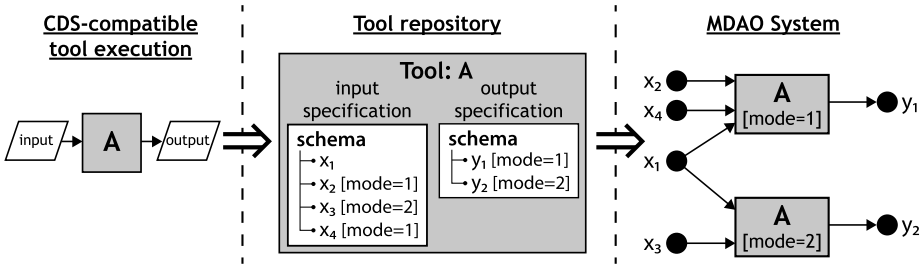
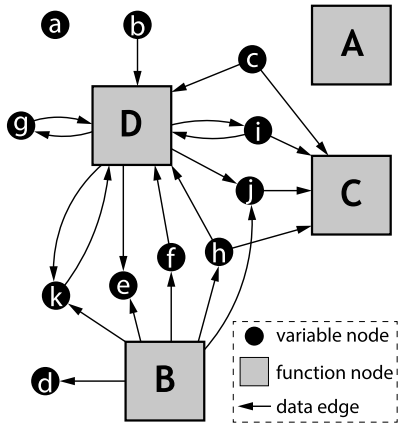
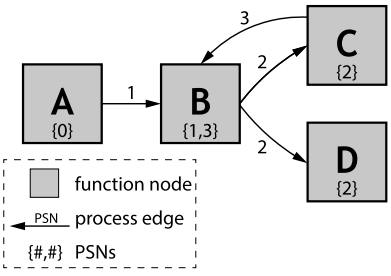


Fig. 6. Illustration of the use of the attribute *mode*. Tool A can operate in two modes, using/producing different I/O values (right). Tool A is stored in the repository as one tool (center), requiring only one mapping to the CDS accounting for both modes (center, left).



(a) Example `DataGraph` instance



(b) Example `ProcessGraph` instance

Fig. 7. Example instances for each of the two main graph classes in KADMOS.

(1) $|V_f| + |V_v| = |V_D|$
(2) $|E_d| = |E_D|$

Where condition (1) states that all nodes belong to either the *function* or *variable* category and condition (2) that all edges should be data edges, thus connecting function nodes with variable nodes, as shown in Fig. 7a.

Instances of the `ProcessGraph` class are digraphs $P = (V_P, E_P)$ meeting the following conditions:

(1) $|V_f| = |V_P|$
(2) $|E_p| = |E_P|$
(3) $\forall v \in V_P : \delta^-(v) + \delta^+(v) \geq 1$
(4) $\text{minpsn}(P) = 0$
(5) $\forall e \in E_P : \text{psn}(e) \in \mathbb{Z}^>$
(6) $\forall v \in V_P : |\text{psn}(v)| > 0 \wedge \forall i \in \text{psn}(v) : i \in \mathbb{Z}^\geq$
(7) $\forall e \in E_P : \text{psn}(e) \in \text{psn}(\text{ter}(e))$
(8) $\forall e \in E_P : \text{psn}(\text{init}(e)) \ni x \mid x < \text{psn}(e)$

Where conditions (1) and (2) mean that all nodes should be function nodes and all edges should be process edges, respectively. Condition (3) states that each node should be connected to at least one edge and condition (4) enforces the process definition to start at a PSN of zero. Conditions (5) and (6) ensure that all PSNs are positive integers, and only nodes can have a PSN of zero. Finally, conditions (7) and (8) enforce process continuity by demanding the nodes around a process edge to have corresponding PSNs (e.g. a process edge e with $\text{psn}(e) = 4$ should have an $\text{init}(e)$ that contains a PSN of 3 or lower and a $\text{ter}(e)$ containing a PSN of 4). An example of a process graph is shown in Fig. 7b.

5.4. Node subcategorization

A more refined *subcategorization* of all the nodes in a graph can be defined based on their indegree, outdegree, and circularity index. This categorization can be inferred automatically based on these three properties. All possible *subcategories*, denoted by sct(v), are listed in Table 2. The subcategories play an important role in the different graph classes, as some subcategories are not allowed in certain stages of the MDAO system, while others require a specific treatment in the graph manipulation algorithms. For example, the hole and collision node subcategories (also present in the syntax by Pate et al. [44]), are generally considered problematic and need to be removed or fixed. A crucial new subcategory in the KADMOS syntax is introduced with the circular variable nodes, which are related to the circularity index introduced in the previous subsection. These nodes require a specific treatment in the data graphs, as will be discussed in Sections 6.2 and 7.

Table 2
Subcategory definition of graph nodes and references to example nodes.

cat	sct	δ^-	δ^+	cir	Fig. 7a
variable	hole	0	0	0	a
	supplied input	0	1	0	b
	supplied shared input	0	>1	0	c
	output	1	0	0	d
	collision	>1	0	0	e
	coupling/ pure circular coupling	1	1	0 1	f g
	shared coupling/ shared circular coupling	1	>1	0 1	h i
	collided coupling/ collided circular coupling	>1	1	0 1	j k
	collided shared coupling/ collided shared circular coupling	>1	>1	0 ≥1	– –
function	hole	0	0	0	A
	source	0	>0	0	B
	sink	>0	0	0	C
	complete	>0	>0	≥0	D

6. KADMOS graphs

The four graph types (RCG, the FPG, the MDG and MPG) introduced in Sect. 4 (Fig. 4), are subclasses of the two main classes discussed in the previous section. In this section we make use of a simple analytical MDAO problem to clarify the determination and use of these four graph classes: the Sellar problem [48]. This problem can be described by the following tools, where the tools indicated with D represent the actual disciplines, F the objective function, and G the constraints:

$$\text{D [mode=1]} \quad \Rightarrow \quad y_1 = c \cdot (z_1^2 + x_1 + z_2 - 0.2 \cdot y_2)$$

$$\text{D [mode=2]} \quad \Rightarrow \quad y_2 = c \cdot (\sqrt{y_1} + z_1 + z_2)$$

$$\text{F} \quad \Rightarrow \quad f = x_1^2 + z_2 + y_1 + e^{-y_2}$$

$$\text{G [mode=1]} \quad \Rightarrow \quad g_1 = \frac{y_1}{3.16} - 1$$

$$\text{G [mode=2]} \quad \Rightarrow \quad g_2 = 1 - \frac{y_2}{24.0}$$

To better illustrate the different KADMOS graphs we assume here to start with a broader tool repository, where, next to D, F and G, the following five fictitious tools are added:

$$\text{A} \quad \Rightarrow \quad b = \text{f}(a)$$

$$\text{B} \quad \Rightarrow \quad b = \text{f}(b); \qquad z_1 = \text{f}(b); \qquad z_2 = \text{f}(b)$$

$$\text{C} \quad \Rightarrow \quad c = \text{f}(b)$$

$$\text{E} \quad \Rightarrow \quad y_1 = \text{f}(b, z_1, z_2); \qquad y_2 = \text{f}(b, z_1, z_2)$$

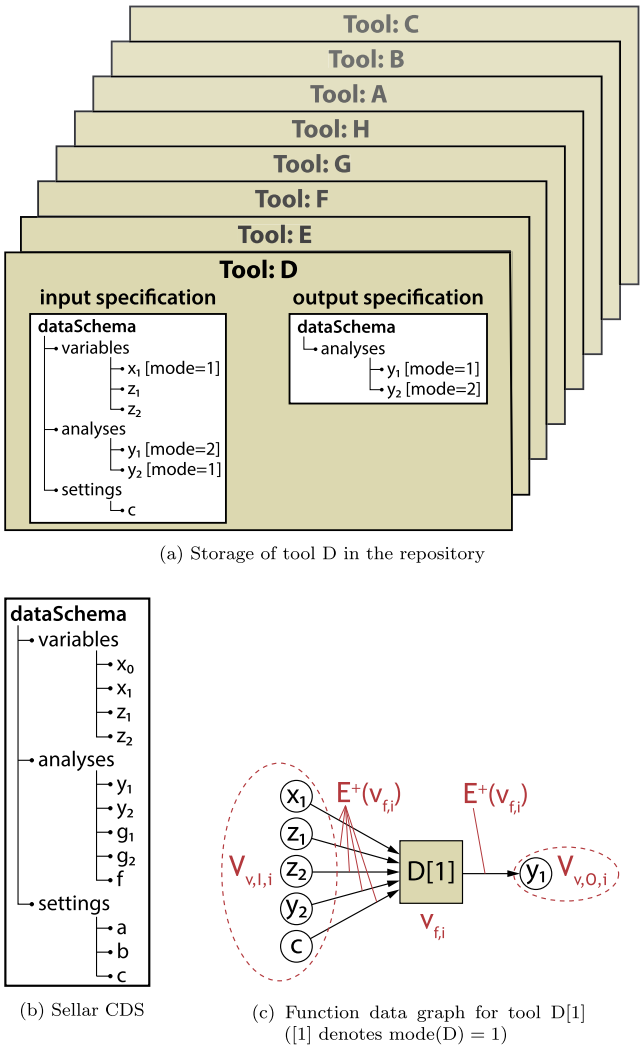
$$\text{H} \quad \Rightarrow \quad x_1 = \text{f}(x_0)$$

Before the first graph in the KADMOS approach, the RCG, can be created, the tools have to be made compatible to a single CDS and stored in the tool repository, as illustrated in Figs. 8a and 8b.

6.1. Repository connectivity graph

The RCG is a specific type of data graph (see Sect. 5.3). The RCG $R = (V_R, E_R)$ can be built by combining the data graphs that represent individual functions. For each unique function/mode combination $i \in I_r$ stored in the tool repository $I_r = \{1, 2, \ldots, m\}$ a data graph can be constructed:

$$D_i = (V_{D,i}, E_{D,i})$$



(a) Storage of tool D in the repository

(b) Sellar CDS

(c) Function data graph for tool D[1] ([1] denotes mode(D) = 1)

Fig. 8. Extended Sellar problem tool repository.

Where the nodes are:

$$V_{D,i} = v_{f,i} \bigcup V_{v,I,i} \bigcup V_{v,O,i}$$

in which $v_{f,i}$ represents the function node, $V_{v,I,i}$ are all the variables from the CDS based on the input file, and $V_{v,O,i}$ contains all the output file variables. The edges of D_i are then:

$$E_{D,i} = E^-(v_{f,i}) \bigcup E^+(v_{f,i})$$

where:

$$E^-(v_{f,i}) = \{(v_v, v_{f,i}) \mid v_v \in V_{v,I,i}\}$$

$$E^+(v_{f,i}) = \{(v_{f,i}, v_v) \mid v_v \in V_{v,O,i}\}$$

The data graph of the tool D[1] ([1] denotes mode(D) = 1) is shown in Fig. 8c. The nodes and edges of the RCG can be written as:

$$V_R = \bigcup_{i \in I_r} V_{D,i}$$

$$E_R = \bigcup_{i \in I_r} E_{D,i}$$

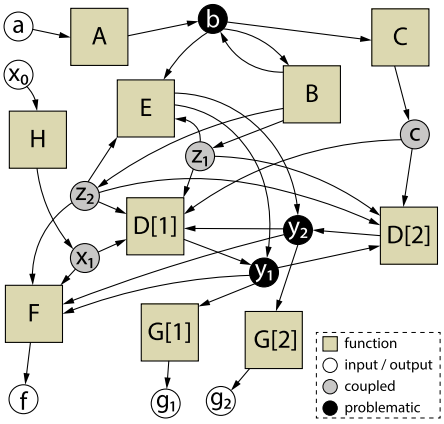


Fig. 9. RCG of the extended Sellar tool repository.

Algorithm 1. FPG composition process.

1.	M: Anticipate whether the MDAO architecture to be imposed is of type MDA, DOE or MDO.
2.	M: Mark problem roles of variables based on the type of MDAO architecture:
a:	If architecture type = MDO or DOE, then mark design variables.
b:	If architecture type = MDO, then mark objective.
c:	If architecture type = MDO, then mark constraints.
d:	Mark QOIs for all architecture types.
3.	M: Solve problematic nodes based on conditions (3) and (4).
4.	A: Check graph validity based on all conditions except condition (5).
5.	M: Merge functions to compress graphs.
6.	A: Assign problem roles of functions (condition (5)).
7.	A: Specify execution sequence.
8.	A: For distributed MDO architectures, specify the distribution of the coupled functions.

Legend
A: automated step
M: manual step using scripting commands

The RCG shown in Fig. 9 is automatically instantiated by KADMOS based on the Sellar tool repository defined above. After instantiation, the subcategory of the nodes (Table 2) can be determined to inspect the graph, leading to the identification of the four main node subcategories for each variable node v, as illustrated in Fig. 9 legend:

- input when $\delta^-(v) = 0 \wedge \delta^+(v) > 0$
- output when $\delta^-(v) = 1 \wedge \delta^+(v) = 0$
- coupled when $\delta^-(v) = 1 \wedge \delta^+(v) \geq 1 \wedge \text{cir}(v) = 0$
- problematic for all other nodes

These main node subcategories will be further discussed in the next section. Note that for all nodes in an RCG $\text{ins}(v) = 0$.

6.2. Fundamental problem graph

The FPG contains the definition of the fundamental MDAO problem to be solved, thus it is a subset of the RCG, containing only the tools that are strictly necessary to solve the optimization problem at hand, plus extra information on the specific role of the tools (which one is a coupled discipline, which the objective function, etc.) and involved variables (which ones are design variables, which fixed parameters, etc.). The FPG is the starting point to impose one of the MDAO architectures in the third stage of the formulation phase.

The FPG has to meet stricter requirements than the RCG. In addition to the standard data graph conditions, an FPG $F = (V_F, E_F)$ has to meet the following extra conditions:

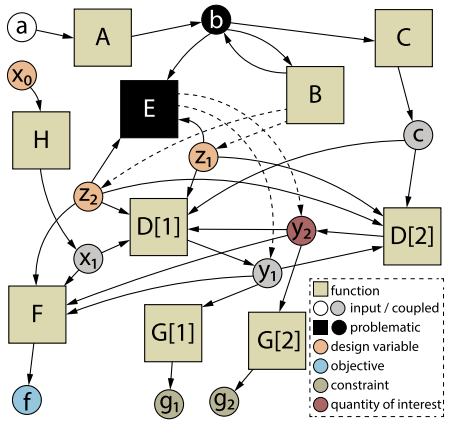
(1) $\forall v \in V_{F,f} : v \in V_{R,f} \vee v$ is merged based on $\subseteq V_{R,f}$
(2) $\forall v \in V_{F,v} : v \in V_{R,v}$ if $\text{ins}(v) = 0$, else $\exists w \in V_{R,v} : w = v$ except $\text{ins}(w) \neq \text{ins}(v) \wedge \text{ins}(w) = 0$
(3) $\forall v \in V_{F,v} : \text{cir}(v) = 0 \wedge \delta^-(v) + \delta^+(v) \geq 1 \wedge \delta^-(v) \leq 1$
(4) $\forall v \in V_{F,f} : \delta^-(v) \geq 0 \wedge \delta^+(v) > 0$
(5) $\forall v \in V_{F,f} : \text{pr}(v) \in$ {uncoupled-DVI, uncoupled-DVD, coupled, post-coupling}
(6) $\forall v \in V_{F,v}$: if $\text{pr}(v) =$ design variable $\Rightarrow \delta^-(v) = 0$
(7) $\forall v \in V_{F,v}$: if $\text{pr}(v) =$ QOI $\Rightarrow \delta^-(v) = 1$
(8) $\forall v \in V_{F,v}$: if $\text{pr}(v) =$ objective $\vee$ constraint $\Rightarrow \delta^+(v) = 0$
(9) $\forall v \in V_F : \exists$ path $Q = (V_Q, E_Q)$ where $v \in V_Q \wedge V_Q \ni w$ where $\text{pr}(w) \in$ {QOI, objective, constraint}

Where condition (1) states that function nodes have to be present in the RCG already or can only be merged nodes based on function nodes from the RCG. Condition (2) states that all variables should also be present in the RCG or can only be instances of RCG variables. Condition (3) implies that the variables cannot be holes, collided, or circular. Condition (4) demands that all function nodes are of subcategory 'source' or 'complete'. Both conditions (3) and (4) are related to the subcategories listed in Table 2. Condition (5) states that all function nodes in the graph should have a problem role assigned (these roles will be further discussed later in this section). Condition (6) stipulates that design variables should be inputs, whereas QOIs can be couplings or outputs (condition (7)), and objective and constraint nodes must be outputs (condition (8)). Finally, condition (9) demands that, for all nodes, at least one path can be created that leads to a QOI, objective, or constraint. Hence, nodes not included in any of those paths can be removed.

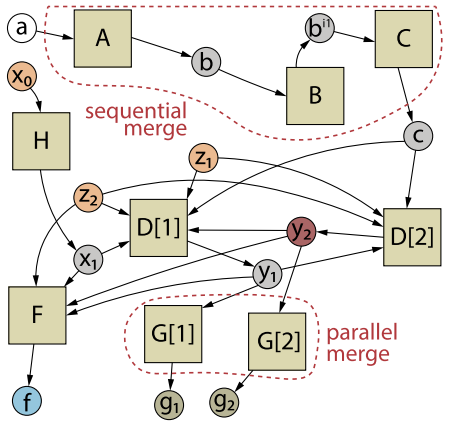
While the RCG can be defined by KADMOS in full automation, on the sole basis of the tool repository, the FPG graph is defined on the basis of the design team specification of the MDAO problem to be solved (e.g. what is the objective? what are the constraints?). Several support functions are provided by KADMOS to assist the design team in the FPG composition process. A suggested semi-automatic composition process for crafting the FPG is given in Algorithm 1.

The FPG composition process for the Sellar problem is illustrated in Fig. 10. This process starts from the RCG in Fig. 9. The anticipated architecture type (step 1) is MDO. In step 2 (refer to Fig. 10a) x_0, z_1, and z_2 are marked as design variables. In order to make z_1 and z_2 valid design variables the incoming edges (B, z_1) and (B, z_2) must be removed. The objective f and constraints g_1 and g_2 can directly be marked as such, since these are already of the valid subcategory *output*. Finally, if the design team is interested in keeping track of the coupling variable y_2, this has to be defined as a QOI. Therefore, the preferred function to determine y_2 has to be selected (between tool D and E). In this case, tool D is selected, also for the variable y_1, thus the edges (E, y_1) and (E, y_2) are removed.

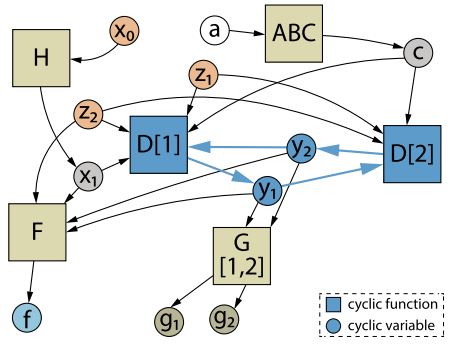
The only problematic nodes left in the graph (step 3) are tool E and variable b. After the previous edge removal, tool E is now a sink and can safely be removed from the graph. Node b expresses a special case, in which a tool takes an initial value and then updates that same value. Hence, the input and output point to the same location of the schema. This is common practice for systems using a CDS approach, when tools take initial guesses or update the values (e.g. a wing geometry) stored in a schema file. This type of node is made valid by splitting it in two instances using a KADMOS method resulting in b $(\text{ins}(b) = 0)$ and b^{i1} $(\text{ins}(b) = 1)$. Hence, this is the moment when node instances are created. The splitting of nodes is a crucial operation, since not every collided or circular node in the data graph can be solved by just removing edges. In realistic design cases, collisions and looped pairs are commonly present, as will be further discussed in Sect. 7 case study. Fig. 10b



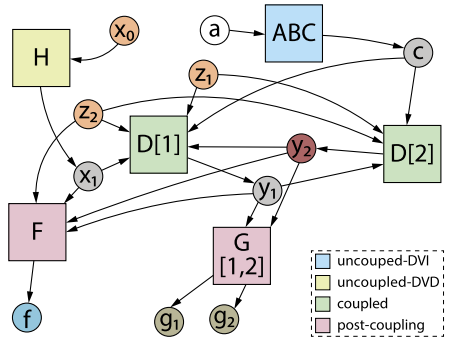
(a) steps 1-2



(b) steps 3-5



(c) step 6 - cycles



(d) step 6 - function problem roles

Fig. 10. Sellar problem FPG composition.

depicts the FPG of the Sellar problem after the execution of the previous steps.

In step 4 a KADMOS check method is executed which returns a positive result. Then the team can merge functions that belong together and that can be executed in sequence (i.e. without any feedback coupling) or in parallel (i.e. without any coupling). Hence, in step 5 the function sequence $\langle A, B, C\rangle$ is merged as one function ABC, and the set of parallel functions $\{G[1], G[2]\}$ is merged as G[1,2]. See Fig. 10c for the resulting graph. These function mergers are useful to declutter the system (and maintain oversight) by clustering more disciplines into fewer "macro-discipline" blocks.

Step 6 of the FPG composition process consists in the determination of the problem role of the various functions. The function problem roles in an FPG belong to one of four main groups: uncoupled-DVI, uncoupled-DVD, coupled, and post-coupling, where the suffix of the two uncoupled types indicates whether the functions are Design Variable Dependent (DVD) or Design Variable Independent (DVI). The coupled functions are the set of functions that are involved in cycles in the FPG. These cycles indicate that a method is required to converge the system, since any order of executing the tools will always require some feedback loop. The single cycle present in the Sellar FPG is depicted in Fig. 10c. Hence, tools D[1] and D[2] have the problem role 'coupled'. The uncoupled functions are all the functions on an incoming path with respect to the cycles, for example the function ABC on the path $\{a, \text{ABC}, c, \text{D[1]}\}$. The default problem role for uncoupled functions is uncoupled-DVI. However, if the FPG contains design variables, then some of the uncoupled functions could also be DVD and should be marked as such, see tool H in this example. This distinction is required later to correctly position the uncoupled functions either outside (if DVI) or inside (if DVD) the main cycle handling design variables (i.e. DOE or optimizer block). The post-coupling functions are simply all the remaining functions. The problem roles are shown in Fig. 10d.

In step 7 the description of the fundamental MDAO problem is completed by specifying the sequence of the functions. The sequences for each problem role are determined automatically in KADMOS. The uncoupled and post-coupling sequences are straightforward, since no cycles are present in those function sets. Any sequence of functions that constitutes a topological order of the nodes is valid, where for the uncoupled functions the most optimal sequence is a sequence where the DVD functions are required as late as possible, hence $\langle \text{ABC}, \text{H}\rangle$. The sequencing of the coupled functions is more challenging as the amount of feedback variables can depend on the sequence given. For this purpose, sequencing algorithms have been implemented that search for the optimal sequence by minimizing the amount of feedback couplings. However, for the Sellar problem both possible sequences have the same amount of feedback variables, thus the sequence $\langle \text{D[1]}, \text{D[2]}\rangle$ was arbitrarily selected. The post-coupling sequence is set to $\langle \text{F}, \text{G[1,2]}\rangle$.

If a distributed MDO architecture needs to be implemented, then the coupled functions in the FPG need to be grouped to indicate how the system needs to be distributed. This grouping can be determined automatically using partitioning algorithms, such as the Metis partitioning algorithm [32] implemented in KADMOS. In the small Sellar example, the two groups are simply the two coupled functions. The other functions in the FPG will be grouped automatically when the distributed MDO architecture is imposed, as further discussed in Sect. 6.6. The FPG in Fig. 10d has been defined with the MDO architecture types in mind, but it is possible to use nearly the same FPG also for MDA and DOE by simply reassigning variable problem roles based on step 2 of Algorithm 1.

Table 3
Lists of possible architecture roles for function and variable nodes in an MDG.

Existing functions and AMRs	New functions in MDG
uncoupled-DVI	coordinator
uncoupled-DVD	optimizer
coupled	converger
post- coupling	DOE
	Surrogate Model (SM)

New variable nodes in MDG	
final design variable	DOE input samples
final output	coupling copy
final coupling	design variable copy
DOE output samples	coupling weight
initial guess design variable	SM approximate
initial guess coupling variable	

6.3. MDAO data graph

The MDG, together with the MPG in the next section, belongs to the last set of graphs produced by KADMOS to enable the third stage of the formulation phase. The MDG is constructed automatically by a KADMOS graph manipulation algorithm that embeds the previously generated FPG into an MDAO solution strategy of choice. The MDG $M_D = (V_{M_D}, E_{M_D})$ has to meet the following conditions:

(1) $V_F \subset V_{M_D}$
(2) $\exists! v \in V_{M_D} : \text{ar}(v) = \text{coordinator} \Rightarrow v = \text{COOR}$
(3) $\forall v \in V_{M_D} : \exists$ cycle $C = (V_C, E_C)$ where $v \wedge \text{COOR} \in V_C$
(4) $\forall v \in V_{M_D,f} \cap V_{F,f} : \text{ar}(v) \in \{\text{1st col. Table 3}\}$
(5) $\forall v \in V_{M_D,f} \setminus V_{F,f}$: if $(\exists w \in V_{F,f}$ where $w = v$ except $\text{ins}(w) \neq \text{ins}(v)) \vee (v$ is an AMR), then $\text{ar}(v) \in \{\text{1st col. Table 3}\}$, else $\text{ar}(v) \in \{\text{2nd col. Table 3}\}$
(6) $\forall v \in V_{M_D,v} \setminus V_{F,v} : (\text{ar}(v) \in \{\text{3rd and 4th col. Table 3}\}) \vee (\exists w \in V_{F,f}$ where $w = v$ except $\text{ins}(w) \neq \text{ins}(v)) \vee (v \in \{\text{ter}(e) \mid e \in E^+(\text{AMR})\})$

Condition (1) states that all nodes from the FPG are also present in the MDG. Condition (2) implies that a single coordinator node should be added. This node provides the system-level inputs and collects system-level outputs. Condition (3) demands that all nodes in the graph are on a cycle that includes the coordinator node. Conditions (4)-(6) concern the architecture roles of the graph nodes. All nodes from the FPG, their instances, and AMRs get an architecture role from the same set as the problem roles in condition (4). All new function nodes also have an architecture role (condition (5)). New variables generally get architecture roles assigned, except when they are added as instances or to serve as outputs of AMRs by the algorithm that imposes the MDAO architecture (condition (6)).

Based on the Sellar problem FPG given in Fig. 10d, KADMOS can impose any MDO architecture. Here, as an example, we illustrate the implementation of the MDF architecture, with a Gauss-Seidel convergence scheme, which requires the addition of a converger block to drive the MDA convergence cycle, see Algorithm 2 and Fig. 11a.

Multiple algorithms are available in KADMOS to automatically determine the reconfigured data connections necessary to solve the optimization problem based on various MDO architectures. Hence, the same FPG in Fig. 10d can be used to impose IDF, to reconfigure the MDF with a Jacobi convergence schema, or even to apply distributed architectures such as BLISS-2000 or CO. The reconfiguration with CO for the Sellar case is discussed in Sect. 6.6. Other architecture types (i.e. without optimization) would require small adjustments of the FPG, for example, a DOE strategy would be based on design variables, but instead of objective and constraint variables, it can only have QOIs. How these different architectures can be used will be shown in the case study in Sect. 7.

Algorithm 2. MDG algorithm for MDF with Gauss-Seidel convergence scheme (Refer to Fig. 11a for the final graph and Fig. 11c for the XDSM).

1. Check FPG based on graph conditions.
2. Copy the FPG as a starting point for the MDG.
3. Add coordinator block (COOR).
4. Add and connect the converger block (CONV) to the coupled functions.
   a: Remove feedback coupling between coupled functions ⇒ edge (y_2, D[1]) in the FPG.
   b: Connect feedback coupling to the converger ⇒ edge ($y2$, CONV) in the MDG.
   c: Add and connect a copy variable for each feedback coupling ⇒ node y_2^c and its edges.
   d: Add initial guess of the copy variable and connect it to the COOR and CONV blocks ⇒ node y_2^{c0} and its edges.
   e: Add final coupling value variables between the coupled functions and the coordinator ⇒ node y_2^* and its edges.
5. Add and connect optimizer (OPT):
   a: Connect design variables as output of the Optimizer ⇒ edges (OPT, x_0), (OPT, z_1), and (OPT, z_2).
   b: Add initial guess for the design variables and connect them to the coordinator (COOR) and optimizer (OPT) blocks ⇒ e.g. node x_0^0 and its edges.
   c: Connect the objective and constraint variables from the post-coupling functions as input to the optimizer ⇒ edges (f, OPT), (g_1, OPT), (g_2, OPT).
   d: Add final objective and constraint value variables between the post-coupling functions and the coordinator ⇒ e.g. node $f*$ and its edges.
6. Connect any remaining input nodes to the coordinator ⇒ edge (COOR, a).
7. Check MDG based on all conditions.

6.4. MDAO process graph

The MPG is the only process type graph discussed in this paper. This graph defines the process steps that are required to solve the MDAO problem based on the selected architecture, hence it contains the execution sequence of all the blocks in the architecture, including the necessary iteration cycles, their nesting, etc. An MPG is always based on an MDG. The MPG $M_P = (V_{M_P}, E_{M_P})$ should meet the following additional conditions with respect to the `ProcessGraph` class defined in Sect. 5.3:

(1) $V_{M_P} = V_{M_D,f}$
(2) $\exists! v \in V_{M_P} : \text{psn}(v) = \text{minpsn}(M_P) \Rightarrow v = \text{COOR}$
(3) $\exists! v \in V_{M_P} : \text{psn}(v) = \text{maxpsn}(M_P) \Rightarrow v = \text{COOR}$
(4) $\forall v \in V_{M_P} : \exists$ cycle $C = (V_C, E_C)$ where $v \wedge \text{COOR} \in V_C$
(5) $\exists$ cycle $C = (V_C, E_C) \in M_P$ for which $\{\text{psn}(e) \mid e \in E_C\} = [1, \text{maxpsn}(V_C)]$

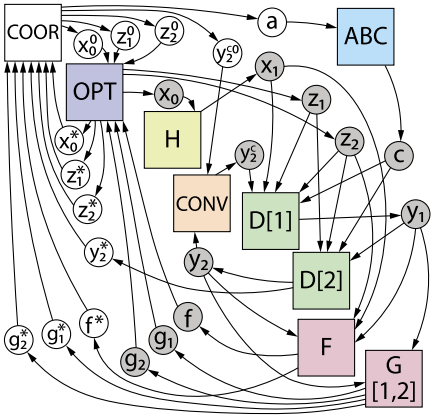
Where condition (1) states that each function node from the MDG should also be present in the MPG. Conditions (2) and (3) ensure that the process only starts and ends at the coordinator node. Condition (4) demands that all nodes are part of a cycle that contains the coordinator and condition (5) ensures that there is at least one cycle with continuously increasing step numbers. The MPG corresponding to the earlier discussed MDG is automatically determined by the KADMOS algorithm provided in Algorithm 3.

6.5. XDSM visualization of KADMOS graphs

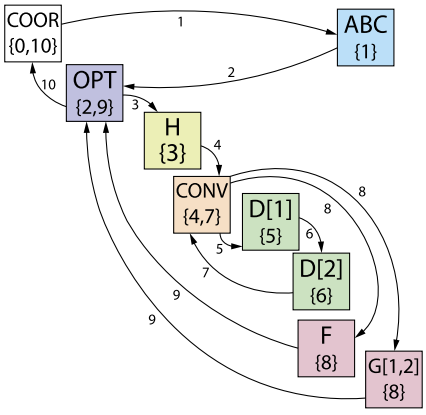
In order to offer a convenient visualization of the assembled MDAO system, KADMOS provides a method to combine the MDG and MPG into a single representation based on the XDSM. To do that, KADMOS actually defines the XDSM graph as the union of the data and process graphs:

$$\text{XDSM} = M_D \cup M_P$$

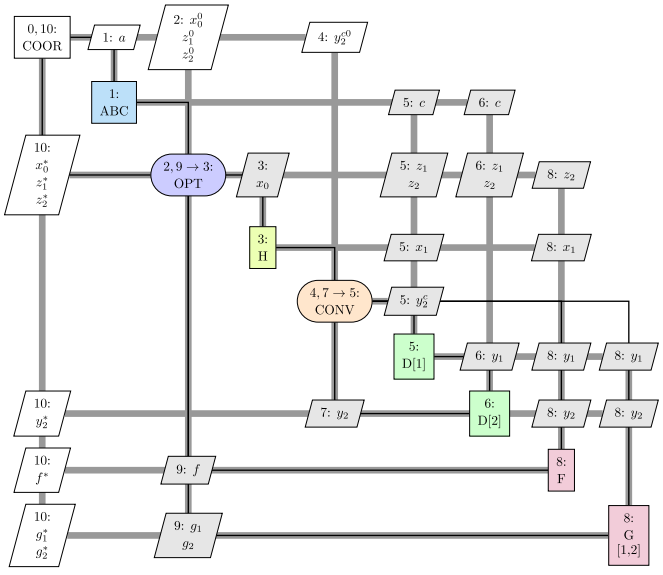
The result is shown in the example in Fig. 11c. This XDSM is automatically created using the Python graph objects MDG and MPG



(a) MDG



(b) MPG



(c) Automatically generated XDSM of the combined MDG+MPG

Fig. 11. MDAO graphs and XDSM view of the Sellar problem using the MDF architecture with a Gauss-Seidel convergence scheme.

and a graph-based extension of the LaTeX-based XDSM writer [64] by Lambe and Martins [33]. The same method can also be used to solely visualize data graphs, which would result in 'XDSM data flow' visualizations. Without process information, such an XDSM would be equivalent to an N^2 chart.

Algorithm 3. MPG algorithm for MDF with Gauss-Seidel convergence scheme (see Fig. 11b for the final graph and Fig. 11c for the XDSM).

1. Check MDG based on graph conditions.
2. Start with an empty directed graph object of the `MdaoProcessGraph` class.
3. Add the function nodes from the MDG.
4. Set PSN to 0 and assign PSN = 0 to COOR block.
5. Add a process from the COOR block to the OPT block via the uncoupled-DVI functions ⇒ edges (COOR, ABC){1} and (ABC, OPT){2} and the corresponding PSNs on the nodes.
6. Add a process from the OPT block to the CONV block via the uncoupled-DVD functions ⇒ edges (OPT, H){3}, (H, CONV){4} and node PSNs.
7. Add an iterative process from the CONV block through the coupled functions back to the CONV block ⇒ edges (CONV, D[1]){5}, (D[1], D[2]){6}, and (D[2], CONV){7}.
8. Add a process from the CONV block to the OPT block via the post-coupling functions ⇒ e.g. edges (CONV, F){8} and (F, OPT){9}.
9. Add a process from the OPT block back to the COOR block ⇒ edge (OPT, COOR){10}.

Of the two graphs that describe an XDSM, the MDG will grow in size more quickly as the MDAO system becomes larger and more complex. Since the growing amount of off-diagonal information to be displayed would affect the readability of the XDSM visualization, KADMOS can be set to visualize just the number of connections, rather than the full list of exchanged I/O data. Hence instead of '4: x_1, z_1, z_2' as shown in Fig. 11c, KADMOS would only state '4: 3 connections'. This concise notation will be used in the case study (Sect. 7).

As discussed in the introduction, the formulation of an MDAO system in large, collaborative design projects can be severely impaired by a lack of proper visualizations. Debugging, documenting, exchanging information and, even more, maintaining oversight of the whole systems would easily become impossible, thereby compromising the success of the very MDAO initiative. The graph-based formalization provided by KADMOS appears to be a key enabler to such visualization, even beyond the KADMOS native XDSM generation ability. As discussed later in this paper (Sect. 6.7), its compact, structured and rigorous syntax provided the opportunity to develop an advanced dynamic visualization tool, called VISTOMS, able to interactively display in the browser XDSMs of any size (as well as other type of useful visualizations) and inspect any single exchanged data, by toggling and expanding every XDSM block. Details on VISTOMS are provided by Aigner et al. [2].

6.6. Reconfiguration of the MDAO system: collaborative optimization

One of the motivations for describing the different stages of MDAO systems using a graph syntax, is that the system can be reconfigured very easily. In Fig. 11 the monolithic MDF architecture has been implemented on the FPG shown in Fig. 10d, but a radically different strategy for solving the same MDAO problem could also be used, such as the distributed CO architecture. MDF and CO are just two of the standard MDAO architectures currently available in KADMOS, but new ones can be defined, possibly requiring the addition of new FPG graph manipulation methods. When CO was added to KADMOS, for example, its algorithms were built using a combination of the methods previously developed for monolithic architectures and newly developed methods, specific for distributed architectures (e.g. determination of global and local design variables and constraints, addition of new AMRs, etc.). With a growing library of MDAO architectures, the amount of basic graph manipulation methods included in KADMOS also grows, making it easier to add new architectures (or variations of architectures) to the package.

The basic steps of the MDG creation for CO are summarized in Algorithm 4. The combination of MDG and MPG for the CO strategy

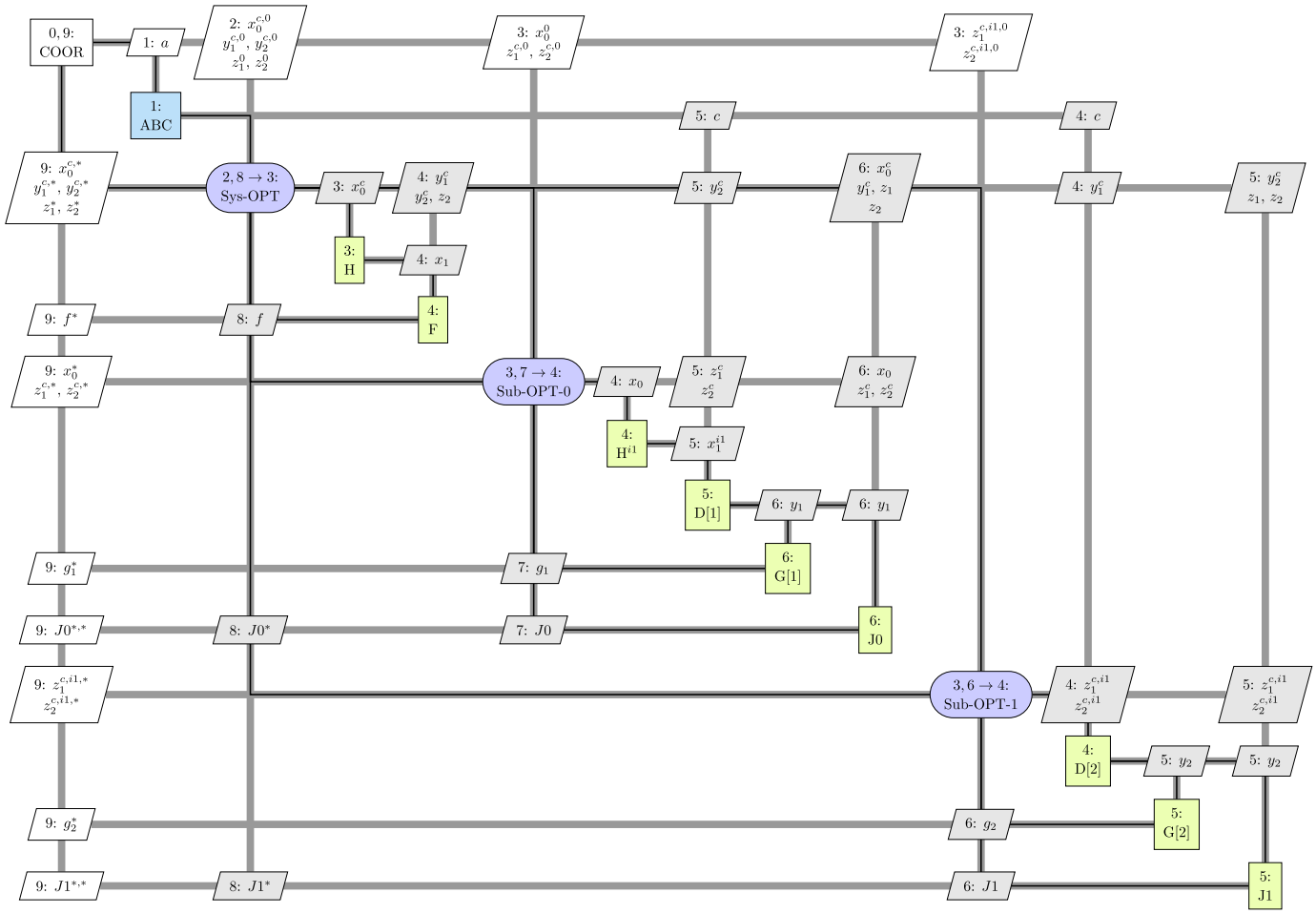


Fig. 12. Automatically generated XDSM of the CO architecture imposed on the FPG shown in Fig. 10d.

is depicted in Fig. 12. Multiple concepts from the KADMOS graph syntax come forward in this small example. For example, many instances and copies of variables are created, especially for the global design variables z_1 and z_2, as these are used by functions in all three optimization cycles. In addition, also a function instance is created for the function H (see: H^{i1} in the sub-OPT-0 cycle), which is done in this case to keep the suboptimization cycle independent of the functions in the main Sys-OPT cycle. The algorithm also adds the AMRs J0 and J1, which are the consistency constraint/objective functions that are part of the CO formulation. Also note that all functions within the optimization cycles now have the architecture role 'uncoupled-DVD' functions, since the distribution of D[1] and D[2] through separate optimization cycles means that no convergence cycle is required within any optimizations, making the categories 'coupled' and 'post-coupling' irrelevant in this situation.

The implementation of other monolithic and distributed architectures will be further discussed in Sect. 7.

Algorithm 4. MDG algorithm for CO (Refer to Fig. 12 for the corresponding XDSM).

1-3. Same as steps 1-3 in Algorithm 2.
4. Analyze the distribution of the whole system based on the provided distribution (step 8 in Algorithm 1) of the coupled functions in the FPG:
   a: Identify global objective variable ⇒ f.
   b: Identify global and local constraint variables ⇒ global: -, local: g_1 with D[1] and g_2 with D[2] group.
   c: Identify global and local design variables ⇒ global: z_1, z_2, local: x_0 with D[1] group.
   d: Determine function grouping: for each function, assess whether it belongs to the system-level and/or to one of the disciplinary groups.
   ⇒ system-level: ABC, H, F; D[1] group: H, D[1], G[1]; D[2] group: D[2], G[2].
5. Split functions that occur multiple times in the function grouping ⇒ H^{i1}.
6. Start loop for each subsystem group:
   a: Localize the disciplinary group by introducing copies of design variables and couplings ⇒ e.g. z_1^c.
   b: Add the consistency objective function AMR ⇒ J0, J1.
   c: If disciplinary group contains cycles: add converger (similar to step 4 in Algorithm 2) ⇒ N/A.
   d: Add and connect optimizer (similar to step 5 in Algorithm 2) ⇒ Sub-OPT-0 and Sub-OPT-1.
7. Add and connect system-level optimizer (similar to step 5 in Algorithm 2) ⇒ Sys-OPT.
8-9. Same as steps 6-7 in Algorithm 2.

6.7. Storage and exchange of MDAO solution strategy formulations: CMDOWS

With the automatic determination of the MDG and MPG, the formulation phase of the MDAO solution strategy is completed. The entire process discussed so far is summarized in the top rows of Fig. 13. At this point, to bridge the implementation gap, a solution is required to translate KADMOS's neutral representation of the MDAO system into an executable workflow. Since all the graphs generated by KADMOS are stored as inexecutable Python objects, considered options were to have KADMOS translate them in the proprietary format of some PIDO tool of choice or to provide an Application Programming Interface (API). These options were however discarded in favor of full platform independence.

Tailoring KADMOS functionality to a single PIDO platform, or committing to the development and maintenance of multiple

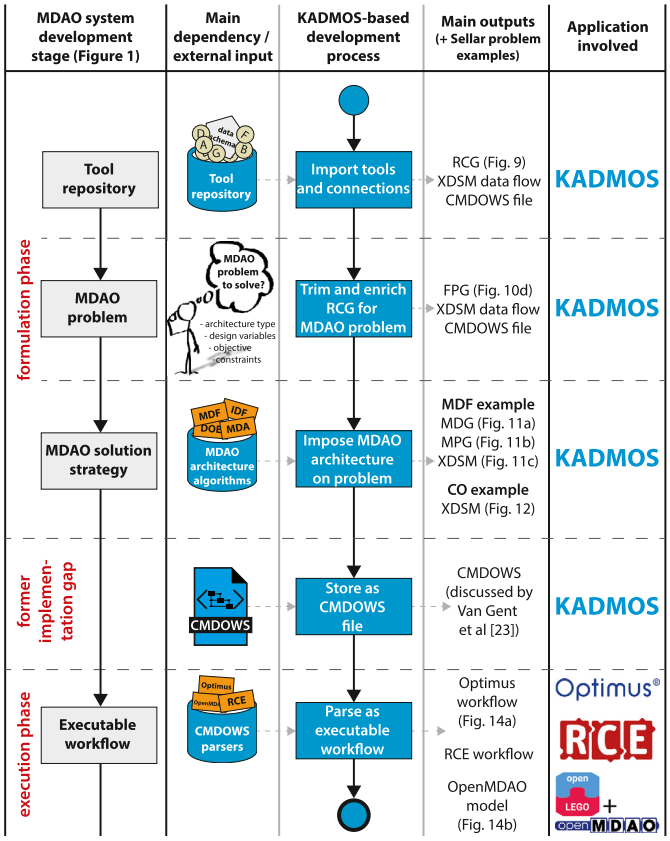


Fig. 13. Overview of the formulation phase supported by KADMOS and bridged implementation gap through the CMDOWS file.

translators (or an API) for a range of platforms seemed inconvenient.[3] Instead, a more lean solution was devised with the definition of a neutral data exchange format to export the generated MDAO systems formulations, leaving the translator development responsibility to the PIDO system users or to the PIDO developers themselves. This approach was actually actively supported by two prominent PIDO platform developers, namely Noesis and the German Aerospace Center, who saw in the neutral data exchange schema a potential solution to the flexibility demand of their customers, as well as a means to deliver more easily custom solutions to their advanced users.

This led to the development of the Common MDO Workflow Schema (CMDOWS). CMDOWS [59] is an XML-based, open-source format that is currently proposed as the "CPACS counterpart" for MDAO system specifications storage. CMDOWS is not only used to export the final output of KADMOS, but also to store the definition of the MDAO system at any stage of its development. Hence, the earlier stages of the MDAO system formulation represented by the RCG and FPG can also be stored as a CMDOWS file. Thus, although the primary objective of CMDOWS is to enable links with PIDO tools, it actually allows other MDAO support applications that can contribute to the MDAO system formulation phase to connect to KADMOS.

Indeed, in the collaborative MDAO project AGILE [11], multiple platforms are combined in a broader MDAO framework [21]: a business process integration platform to facilitate the design team in the specification of the optimization problem and MDAO architecture, various online design tool and surrogate model repositories to collect design competence definitions, but also visualization tools, such as the aforementioned VISTOMS. It is through the latter that KADMOS can provide the necessary human-readable representations of the MDAO system, so needed by the design team to stay in control of their complex formulations.

At the moment CMDOWS translators have been developed for the PIDO platforms Optimus [20], RCE [65], and OpenMDAO [63] using the OpenLEGO package [62] of De Vries [56], largely proving the KADMOS capability to close the implementation gap (Fig. 13), while remaining PIDO platform independent. Developments and case studies in the AGILE project [22] have shown that these translators can actually be created with a relatively small effort, thanks to the conceptual similarities between the CMDOWS data structure and the proprietary encoding used by the above-mentioned PIDO tools. Examples of automatically generated executable workflows for the Sellar solution strategy are shown in Fig. 14. The full description of the CMDOWS format is outside the scope of this paper, but detailed info is given by Van Gent et al. [23].

7. Case study: aerostructural wing design optimization

In the previous section the simple Sellar problem was used as a means to explain the KADMOS functionality, its graph syntax and definitions. However, the real KADMOS value stands in its ability to support the formulation and reconfiguration of large scale (in terms of amount of tools and the size of the design team involved) MDAO systems, of realistic complexity and relevance for the industry. This is the goal of the aerostructural wing design optimization case study presented in this section. This test case uses a multidisciplinary, distributed tool repository constituted by a mix of proprietary design tools, mostly working as black boxes (i.e. aerodynamic solver, weight estimation tool, etc.), which have been linked together using the CDS approach. The aircraft configuration considered in this case study is a conventional passenger jet used in one of the AGILE project design campaigns. In this case study it is assumed that an MDAO integrator is creating a script for the design team to formulate and reconfigure the MDAO solution strategy using KADMOS methods. These reconfigurations are representative of a typical MDAO process, where an initial design point is determined first through a multidisciplinary convergence study, then design space exploration is performed using a DOE to assess the sensitivity of the design to some parameters of interest, to finally set up the actual optimization problem.

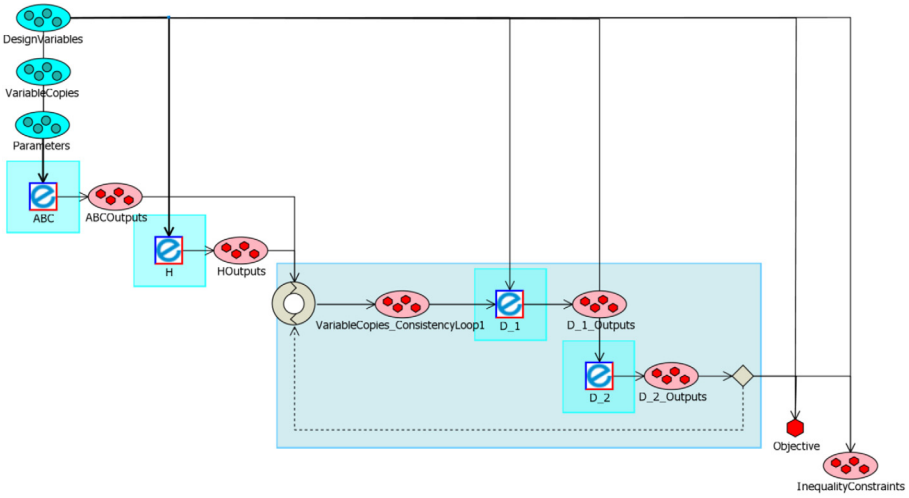
7.1. Tool repository

The collection of tools available in the database is summarized in Table 4. This includes twelve disciplinary tools developed by different experts, some of which featuring multiple execution modes, thus representing a truly heterogeneous repository. All tools have been made compatible with the CDS standard CPACS. As long as this compatibility is respected, any extra tool can be added to the repository in a straightforward manner. After the database has been imported, an RCG is created by KADMOS containing 2,909 nodes and 12,068 edges. By taking advantage of the mode attribute (see Fig. 6) the twelve tools in the database lead to 25 function nodes in KADMOS. Due to the sheer amount of nodes and edges, the obtained RCG cannot be visualized as a graph because of obvious readability issues. The XDSM data flow with summarized connections can be used instead, as depicted in Fig. 15, although limited to a subset of the database.

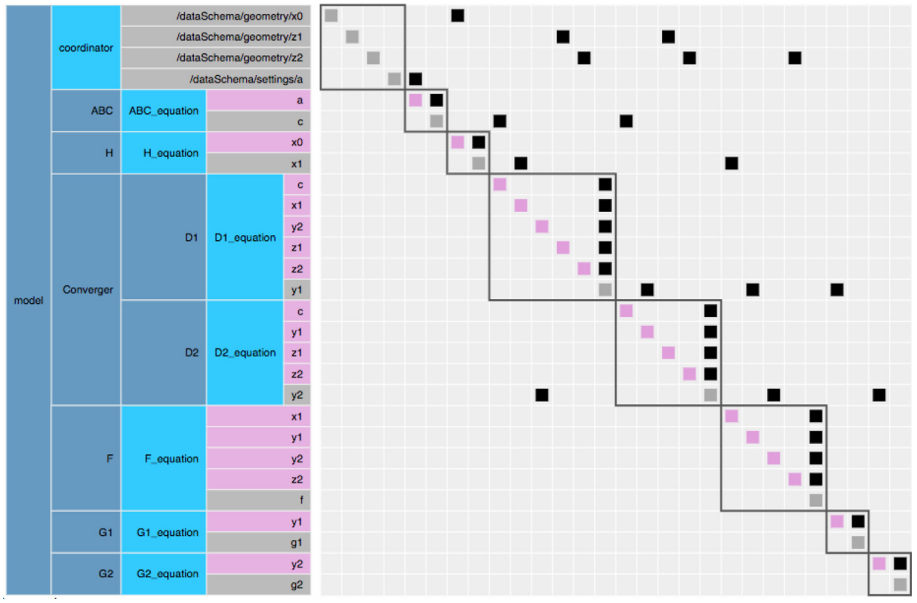
7.2. Initial design point (design convergence study)

In a realistic design project the team would not directly jump to setting up the aerostructural optimization that they have in mind.

[3] N.B. The API option was considered inconvenient as it is a too programming-intensive solution for a research-based software development with its application in a domain dominated by users with an engineering background rather than an IT background.



(a) Optimus user interface screenshot



(b) OpenMDAO model view

Fig. 14. Executable workflows created with two different workflow execution platforms, starting from the same CMDOWS file. A screenshot of the workflow generated with a third platform, RCE, is provided in Fig. 16c.

Rather, the tools need to be tested first and a converged initial design point is required to start any optimization study. As discussed in Sect. 3, a design convergence study is one of the possible MDAO architectures supported by KADMOS. Following the algorithm suggested in Sect. 6.2, the steps leading to the generation of the FPG are given here below, while the generated graph is depicted in Fig. 16a:

1. Select the MDAO architecture type MDA
2. The design team is interested in the mass balance of the aircraft and marks the QOIs:
   m_{MTO}: Maximum Take-Off (MTO) mass determined by function MaCal.
   m_{wing}: Wing mass determined by function EMWET, using loads from Q3D[FLC].
   m_{fuel}: Fuel mass determined by function SMFA using lift-to-drag ratio from Q3D[VDE].
   m_{ZF}: Zero-Fuel (ZF) mass by MaCal.

3a. All functions that do not provide any QOI and that are not coupled to tools providing them are automatically removed. In this case functions such as PHALANX, PROTEUS, OBJ, CNSTRNT, Q3D[APM] are removed from the RCG.
3b. For the initial geometry, the HANGAR function with mode AGILE_AC_wing is selected. The other HANGAR mode and the INITIATOR function are both removed.
3c. The HANGAR and SCAM functions still cause collisions, since both write to the same wing geometry elements. At this stage, the HANGAR tool is selected and SCAM is removed.
4. Graph is found to be valid by KADMOS on all necessary FPG conditions (see Sect. 6.2).
5a. Q3D[VDE] and SMFA are merged sequentially as function node.
5b. Q3D[FLC] and EMWET are merged sequentially as function node.
6-7. Function problem roles and order are set as shown in Fig. 16a.

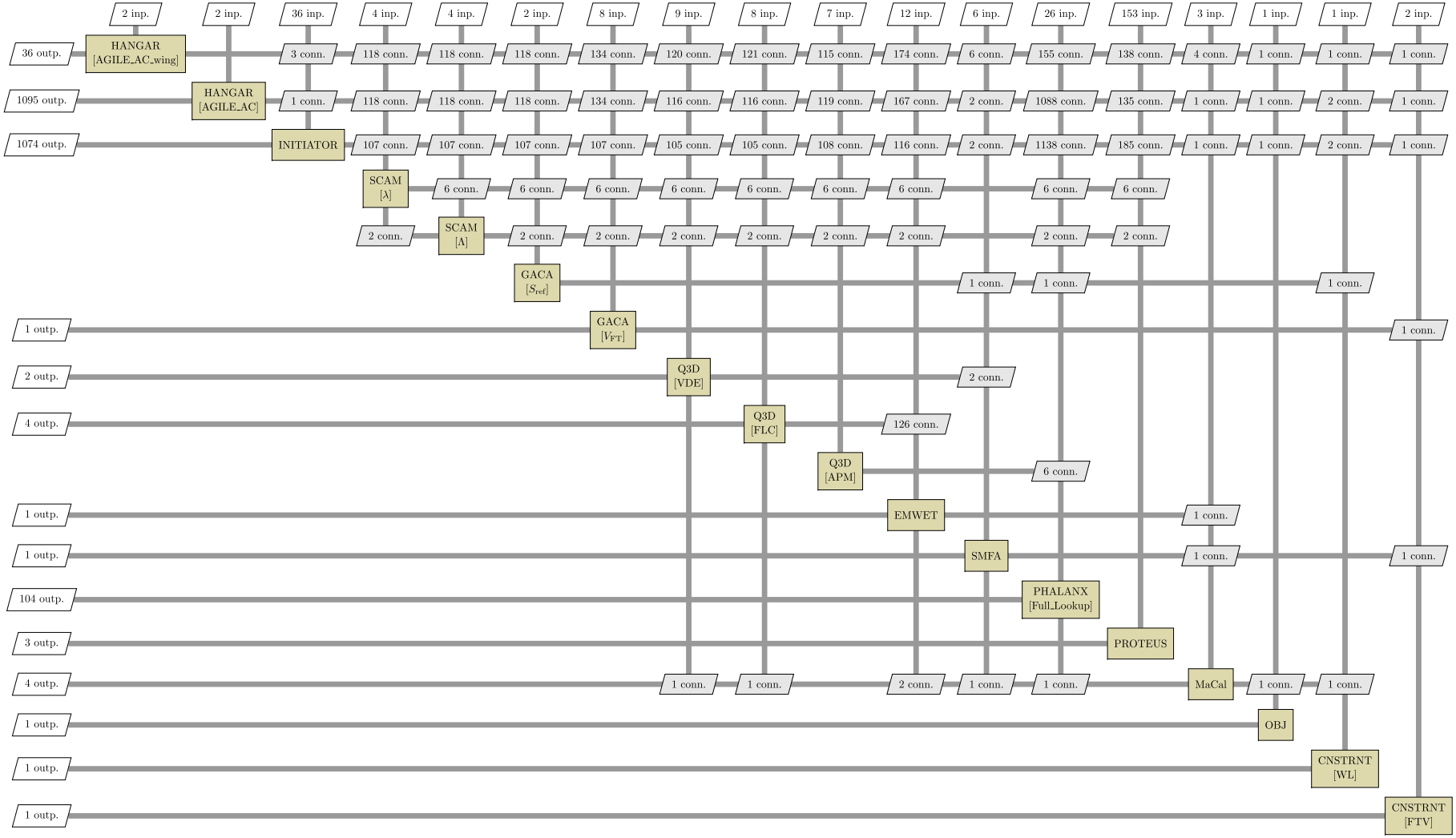


Fig. 15. Automatically generated XDSM data flow (N^2 chart) visualization of a subset of the RCG (not all tool modes are shown to maintain readability) [conn. = connection, inp. = input, outp. = output].

Table 4
Tool repository.

Tool name	Description	Execution method	Modes	Mode description
HANGAR	Tool loads CPACS file with aircraft geometry. Used to have distinction between tool settings and aircraft design in XDSM.	local	AGILE AC	Conventional aircraft design created in the AGILE project
			AGILE AC wing	Adjusted wing of the AGILE conventional aircraft design to meet case study set-up.
INITIATOR [16]	Tool initiates an aircraft design based on top-level aircraft requirements.	local	main	–
SCAM	Simplified CPACS Aircraft Morphing. Adjust. aircraft geometry in different ways	local	λ	Wing taper morph
			Λ	Wing sweep morph
			Γ	Wing dihedral morph
			c_r	Wing root chord morph
			b	Wing length morph
			ξ	Wing spar position change
GACA	Geometrical Analysis of CPACS Aircraft components.	local	S_{wing}	Determination wing reference area
			V_{FT}	Determination wing fuel tank volume
Q3D [37]	Quasi-3D Aerodynamic solver.	remote	VDE	Viscous Drag Estimation
			FLC	Flight Load Case (vortex lattice method)
			APM	Aeroperformance Map
EMWET [15]	Wing mass estimation tool.	remote	main	–
SMFA	Simplified Mission Fuel Analysis: Breguet's range equation.	remote	main	–
PHALANX [45]	Flight dynamics toolbox.	remote	Full Lookup	Full = full dynamic model
			Full Simple	Simple = empirical engine deck
			Symm. Lookup	Symm. = only longitudinal dynamics
			Symm. Simple	Lookup = external engine deck
PROTEUS [36]	Aeroelastic wing analysis tool.	remote	main	–
MaCal	Mass Calculation tool for maximum take-off mass and zero-fuel mass.	remote	main	–
OBJ	Normalized objective function.	script	main	$f_{\text{MTOM}} = m_{\text{MTO}}/m_{\text{MTO,ref}}$
CNSTRNT	Constraint value analysis.	scripts	WL (wing loading)	$c_{\text{WL}} = (m_{\text{MTO}}/S_{\text{wing}}) - \text{WL}_{\text{ref}}$
			FTV (fuel tank vol.)	$c_{\text{FT}} = m_{\text{fuel}}/(\rho_{\text{fuel}} \cdot \eta_{\text{FT}}) - V_{\text{FT}}$

Imposing the MDA with a Gauss-Seidel convergence scheme architecture on this FPG results in the solution strategy shown in Fig. 16b. Exporting these results to a CMDOWS file and parsing it in RCE provides the team with the executable workflow shown in Fig. 16c.

7.3. Design space exploration (DOE)

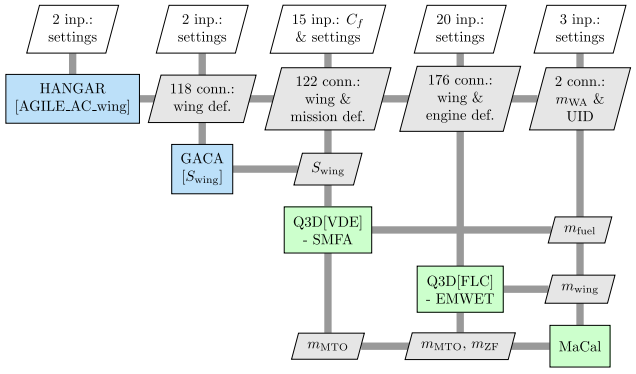
All the stages of the MDAO development process shown in Fig. 1 were covered by the MDA convergence study discussed in the previous section. At this point an iteration is triggered to reconfigure the MDAO system such to perform design space exploration through DOE. The following steps have to be taken to reconfigure the previously generated FPG:

1. Change to MDAO architecture type DOE.
2. QOIs remain and the following design variables are selected:
   b and c_r: wing span and root chord length
   λ_1 and λ_2: taper ratios of two wing segments
   Λ_1 and Λ_2: wing sweep of two wing segments
   ξ_{FS} and ξ_{RS}: wing front and rear spar loc.
   Γ: dihedral of the wing
   C_f: Friction coefficient
   The SCAM tool is added, because of its ability to adjust some design variables, which are not explicitly stated in CPACS.
3. Collisions are now caused by HANGAR and SCAM writing to the same wing elements. Furthermore, SCAM introduces circular variables as it has the wing geometry entries as input and output. The collided and circular variables are fixed by creating variable instances.
4. Graph validity is checked against all necessary FPG conditions (see Sect. 6.2).
5. The five modes of the SCAM function are merged into one function node.

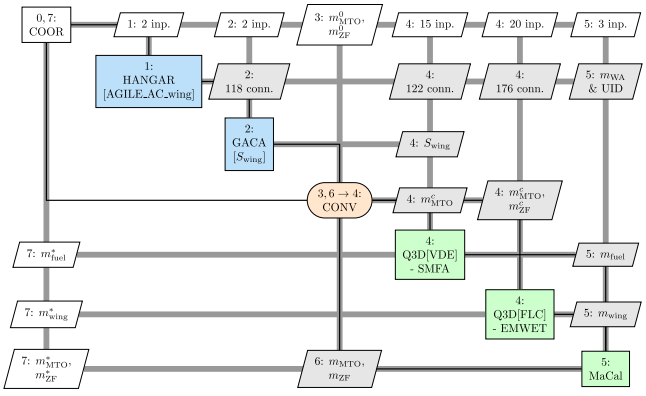
6-7. Function problem roles and order are set as shown in Fig. 17a.

The obtained FPG is shown in Fig. 17a. Two key concepts of the KADMOS graph syntax are used at step 3 of FPG creation process: the circularity index and instances. Initially, some of the wing definition nodes in the FPG are problematic (in a similar fashion as node b in Fig. 10a). This is because the HANGAR function provides a full initial wing definition that is then used and changed by the SCAM function. Fifteen values are changed by SCAM to adjust the wing based on top-level parameters and these fifteen values are initially of the subcategory 'collided shared coupling' with a circularity index of one. These collisions cannot be solved by simply removing connections, since the wing definition is supposed to be updated by SCAM and the initial geometry should come from HANGAR. Therefore, KADMOS automatically solves the collision by creating two instances of these variables, one instance before the SCAM function and one after. In this way, the collision is solved while the two instances still refer to the same position in the CDS, as is required.

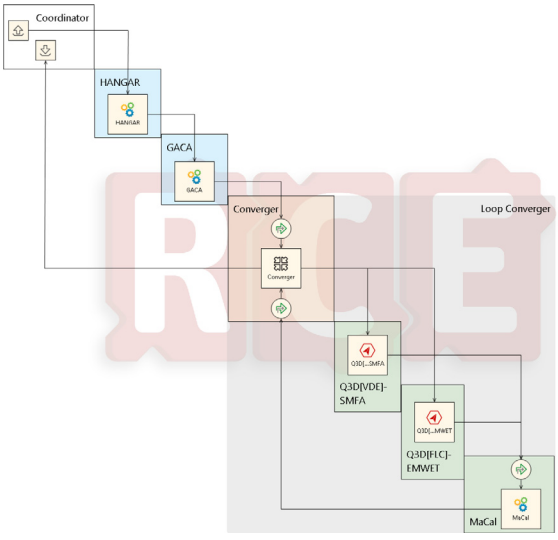
The generated XDSM for the DOE solution strategy is shown in Fig. 17b. The Jacobi scheme was selected to test the effect of parallelizing all three disciplinary groups. The graph manipulation



(a) XDSM data flow of the FPG



(b) XDSM of the combined MDG+MPG for architecture MDA with Gauss-Seidel convergence



(c) RCE workflow instance of the XDSM

Fig. 16. Automatically generated XDSM visualizations of the KADMOS graphs (a, b) and RCE workflow for the first MDA convergence study.

algorithm in KADMOS is able to position and connect the nested iterative elements (i.e. DOE and CONV) such that each design point to be analyzed is converged within the DOE block. Additional data elements that are required for the design variables and QOIs are also added and connected, for example, the new vector with samples for the design variables (e.g. $\boldsymbol{b}^s$) at PSN 2 and the vectors with final values of the QOIs (e.g. $\boldsymbol{m}^*_{MTO}$) at PSN 9. Again, the implementation gap is bridged by storing the solution strategy in a CMDOWS file and parsing it as an executable workflow, with any of the aforementioned PIDO platforms.

7.4. MDO study

Based on the interpretation of the design space exploration, the design team can now formulate an MDO problem. Let's assume the previous DOE study showed that the QOIs were not sensitive to changes of dihedral angle (Γ) and taper ratio (λ_1, λ_2); then these variables can be excluded from the MDO problem formulation. The FPG that needs to be defined is based on the following problem definition:

$$\text{minimize: } f_{MTOM} = \frac{m_{MTO}}{m_{MTO,ref}}$$

$$\text{with respect to: } b, c_r, \Lambda_1, \Lambda_2, \xi_{FS}, \xi_{RS}, C_f$$

$$\text{subject to: } c_{WL} = \frac{m_{MTO}}{S_{wing}} - WL_{ref} \leq 0$$

$$c_{FT} = \frac{m_{fuel}}{\rho_{fuel} \cdot \eta_{FT}} - V_{FT} \leq 0$$

Hence, the Maximum Take-Off Mass (MTOM) needs to be minimized for a given mission by changing the geometry of the wing, while satisfying a constraint on the wing loading (c_{WL}) and making sure that the fuel tank can carry the required fuel for the mission (c_{FT}). The FPG is again based on the previous FPG and is created by performing the following operations in a KADMOS script:

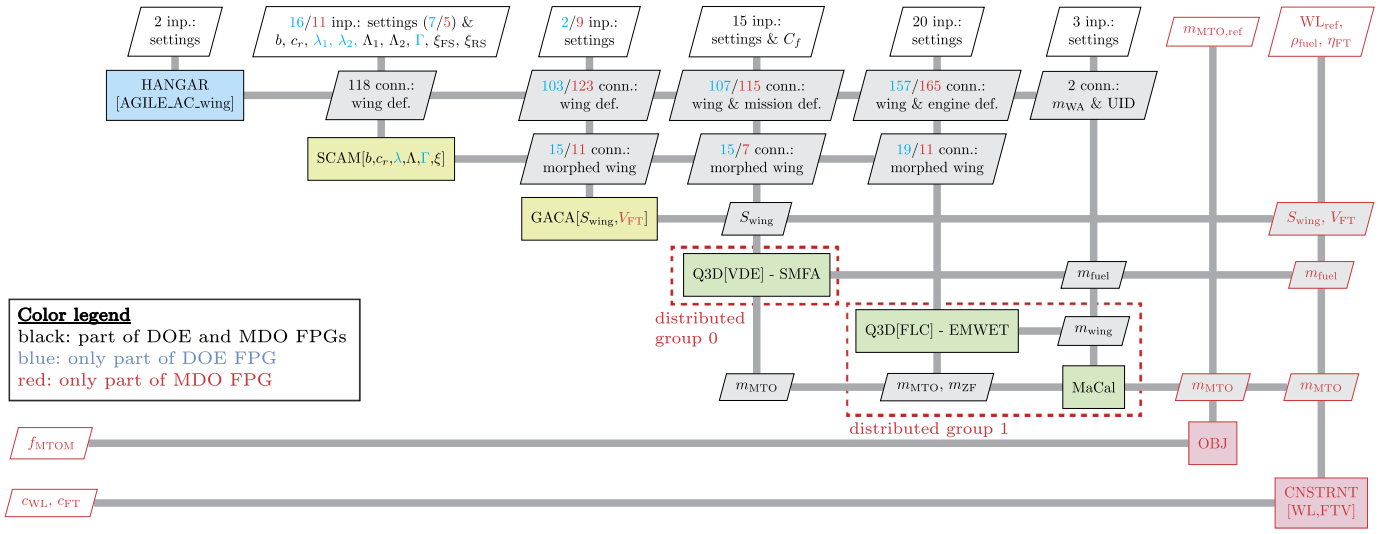
1. Select the MDAO architecture type MDO.
2. Design and QOIs remain from the previous FPG, except Γ, λ_1 and λ_2, and the following objective and constraints are selected:
   f_{MTOM}: output of function OBJ, hence this function from the RCG is added to the FPG
   c_{WL}: output of function CNSTRNT
   c_{FT}: output of function CNSTRNT
   In addition, the mode V_{FT} of the GACA tool is added, since the fuel tank volume is required for the constraint calculation.
5. The GACA modes are merged into one block as well as the two CNSTRNT modes.

6-7. Function problem roles and order are set as shown in Fig. 17a.

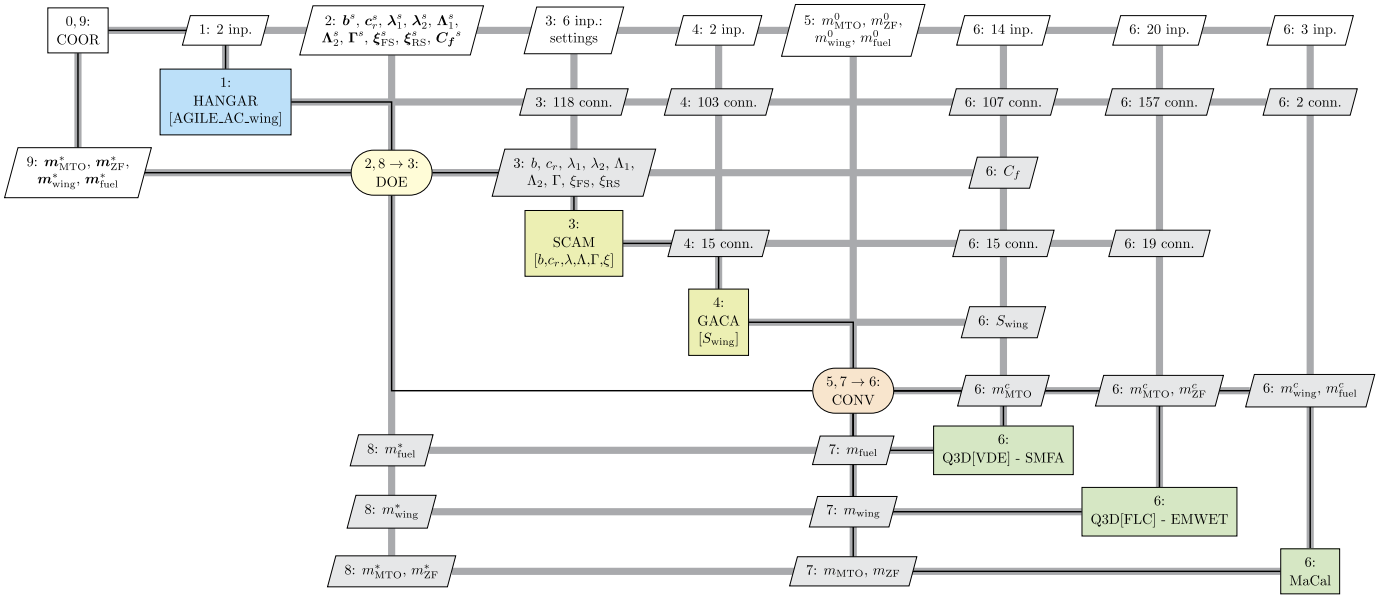
8. The coupled functions are distributed in two groups, as indicated in Fig. 17a.

On this FPG different MDO architectures can be imposed, with the resulting XDSMs for IDF and BLISS-2000 shown in Fig. 18 and Fig. 19. Of these two architectures, the distributed BLISS-2000 (Fig. 19) renders the idea of the extensive analysis and manipulation of the FPG necessary for the automatic formulation of a complex solution strategy. As for the previous cases, the formulated MDAO solution strategies can be stored as CMDOWS files and instantiated as executable workflows in a PIDO platform of choice. Other possible MDAO system reconfigurations, which could become interesting after the first MDO study, might include the replacement of some tool in the repository, a modification of the objective function or the addition of extra constraints, or a change in the MDO strategy. All of them could be easily accommodated and implemented in very short time (given a CPACS compatible tool repository): minutes instead of hours or even days, as would be required with the conventional manual approach.

This case study demonstrated how the KADMOS syntax and algorithms can enable a design team to quickly formulate and



(a) XDSM data flow of the FPG for the DOE and MDO type architectures used in this case study



(b) XDSM of the combined MDG+MPG for the architecture DOE with a Jacobi convergence scheme

Fig. 17. Automatically generated XDSM visualizations of the KADMOS graphs for the development of the design space exploration system. Note that the FPG is also used for the MDO strategies, be it with small changes as indicated.

reconfigure an MDAO system, starting from a CDS-based repository of design tools. KADMOS has also been extensively tested as core component in the broader collaborative MDAO framework [21] developed in the AGILE project, as discussed by Van Gent et al. [22]. Based on the AGILE collaborative design campaigns, expert MDAO users estimated an overall time reduction of 50% achievable by setting up the tool repository and deploying KADMOS.

8. Summary and conclusions

8.1. Graph-based methodological approach

In this paper, a novel graph-based methodological approach and its software implementation, called KADMOS, are proposed to enable the formulation and integration of large collaborative MDAO systems. Starting from a distributed repository of disciplinary tools, whose I/O have been previously mapped on a common data schema, KADMOS can automatically generate a directed graph called Repository Connectivity Graph (RCG) and run preliminary checks to identify possible issues in the repository connectivity. Then, based on the user specification of some quantity of interests (i.e. quantities to be evaluated as objectives and constraints, or simply to be monitored), KADMOS automatically transforms the RCG into the Fundamental Problem Graph (FPG). The FPG is a subset of the RCG, including only the tools and inputs strictly necessary to produce the selected quantities of interest. At this point, the user can intervene again and select a solution strategy, among those currently supported by KADMOS (i.e. multidisciplinary convergence study, DOE or various monolithic and distributed MDO architectures), to apply on the previously defined fundamental problem. Thus KADMOS automatically transforms the FPG into a new set of two plots, the MDAO Data Graph (MDG) and

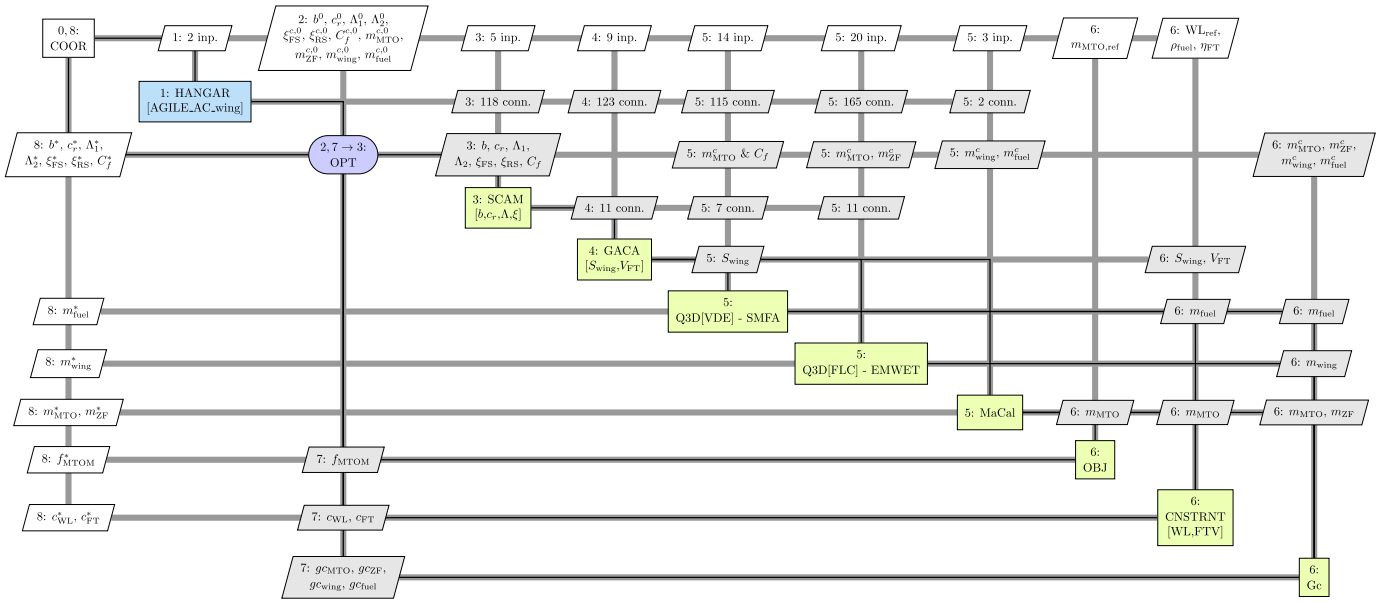


Fig. 18. XDSM of the combined MDG+MPG for the IDF architecture.

the MDAO Process Graph (MPG), which, together, realize the complete formulation of the MDAO system.

8.2. Formulation time reduction

As result of all these automatic graph manipulations and transformations, KADMOS can reduce the whole formalization process from weeks to minutes, for MDAO systems of any size, including tools of different level of fidelity, ranging from simple equations to high fidelity tools or their surrogate models. Even considering the time investment required to make the disciplinary tools compatible with the adopted common data schema, the overall time saving in the formulation process has been estimated in the order of 50%.

8.3. Extended design agility

Furthermore, the step-by-step formulation approach implemented in KADMOS, dramatically increases the agility of the design team in the application of collaborative MDAO. After the execution of a multidisciplinary convergence study, for example, designers can easily set up a DOE study and evaluate the sensitivity of the results to certain parameters. This information can then be used to formulate an optimization problem that includes as design variables only the parameters resulting as most effective in the DOE. After that, also the burden to embed the problem into one of the various and diverse MDO architectures is totally eliminated, because KADMOS can manipulate the same FPG into any MDO architecture in just minutes. If preliminary results of the optimization suggest the use of a more convenient architecture, changes will be effortlessly. Also if constraints and/or design variables need to be added or removed, or different objectives selected, or different disciplinary tools be involved (as far as compliant to the common data schema), KADMOS provides the necessary agility to easily adjust any of the aforementioned, thus supporting the typical iterative nature of the design process.

8.4. System oversight through visualizations

The benefits of KADMOS are not limited to the reduction in formulation time and design agility augmentation. Since KADMOS stores all generated graphs by means of the standardized storage format CMDOWS, the VISTOMS package (co-developed with KADMOS and CMDOWS-compatible) can be used at any stage of the formalization process to generate the necessary visualizations (e.g. toggle-able XDSMs) to report the status of the MDAO system in its development, to ease debugging and, most of all, to guarantee discipline experts and MDAO architects the required oversight to manage distributed computational systems of any size.

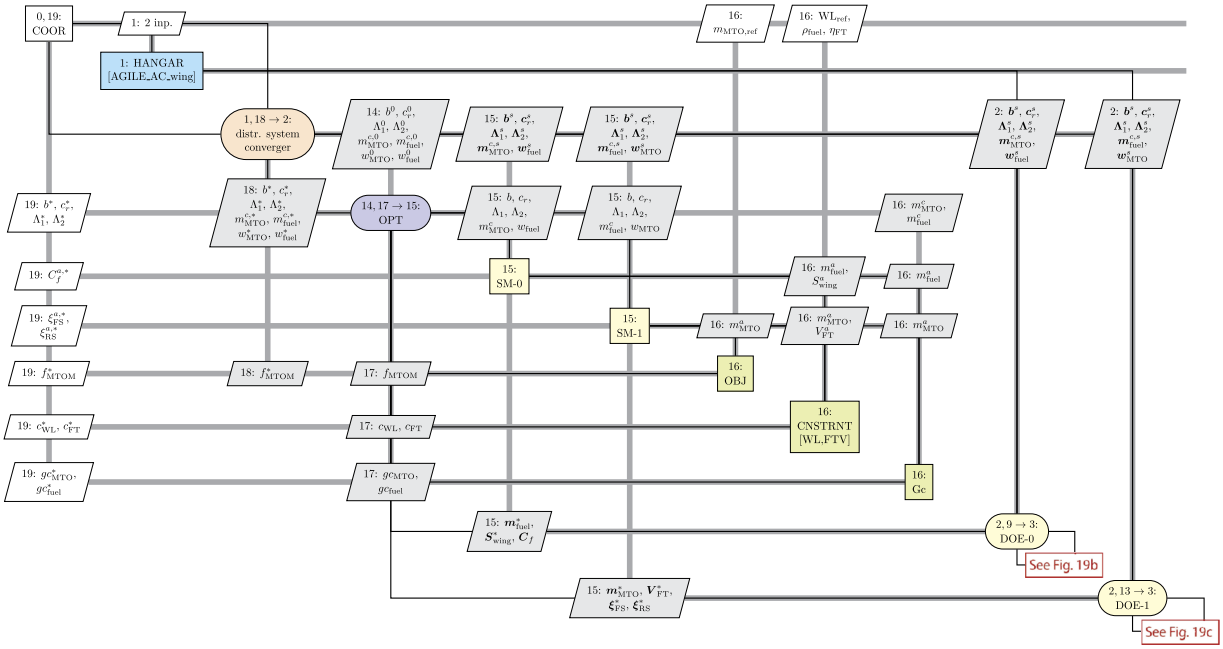
8.5. Closure of the implementation gap

Last but not least, the formulations produced by KADMOS (stored as CMDOWS files), while totally PIDO platform neutral in nature, lend themselves to a direct translation into executable workflows. Three translators have been developed so far, to enable the automatic integration of the solution strategies formulated by KADMOS in the open-source platforms OpenMDAO and RCE and the commercial platform Optimus, thus demonstrating the ability of the proposed approach to close the implementation gap that usually impairs the transition from the formulation to the execution phase of MDAO systems.

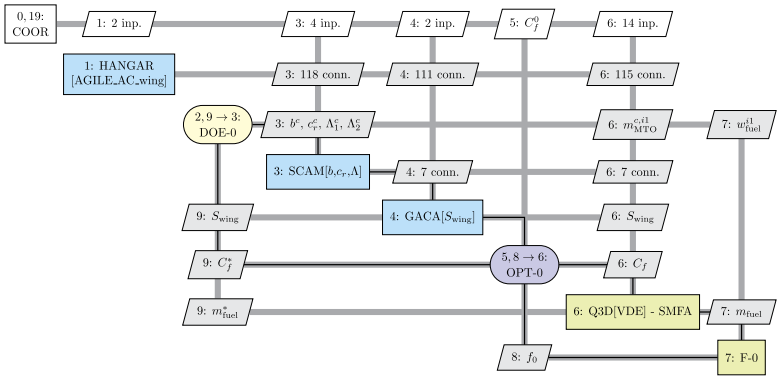
8.6. Originality

Although the graph-based methodological approach proposed in this paper takes inspiration from the work of Pate et al., it drastically extends its scope, to address all the stages of the MDAO system formulation process. To this purpose, the graph syntax had to be largely extended and refined with respect to Pate's, by including new concepts such as the circularity index, variable instances, modes, problem roles and architecture roles. These concepts were also required to support the assembly of distributed optimization architectures (e.g. CO and BLISS-2000 variants), which is another original aspect of the presented work.

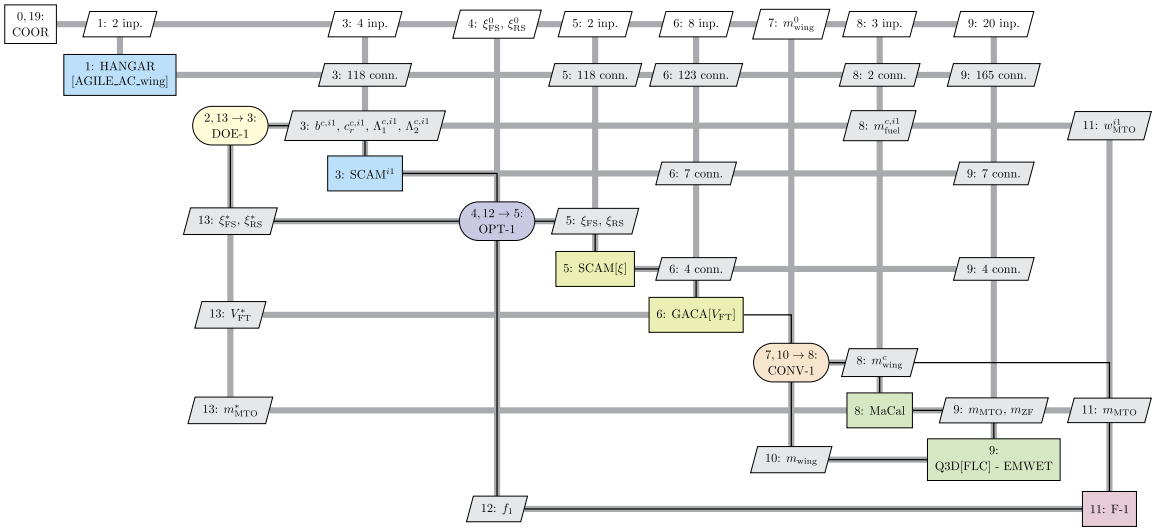
While several KADMOS capabilities match those offered by Hoogreef's InFoRMA system, the theoretical basis drastically differs from Hoogreef's use of semantic webs to represent MDAO systems. Furthermore, the use of the CMDOWS to store the various graphs, allows KADMOS to benefit from the VISTOMS visualizations and offers the flexibility to select among different PIDO platforms for workflow integration.



(a) System-level BLISS-2000 approach



(b) Sub-level 0 BLISS-2000



(c) Sub-level 1 BLISS-2000

Fig. 19. Automatically generated XDSM visualization in three parts of the KADMOS graphs for the BLISS-2000 solution strategy imposed on the FPG shown in Fig. 17a.

8.7. Outlook

Having achieved the necessary targets in terms of formulation process time reduction, increased agility, continuous system oversight and implementation gap reduction, the next planned KADMOS advancement concerns the use of machine learning techniques to advise the user on the most suitable MDAO architecture to solve the optimization problem at hand, on the basis of the main features of such problem. In a first step the advice will be provided before starting the optimization, and later, dynamically, based on the info gathered during optimization.

Declaration of Competing Interest

The authors declare that there are no known conflicts of interest associated with this publication and that the only source of financial support has come from the European Union, which has not influenced the outcome of the work.

Acknowledgements

The research presented in this paper has been performed in the framework of the AGILE (Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts) project—awarded with the ICAS (International Council of the Aeronautical Sciences) Award for Innovation in Aeronautics—and has received funding from the European Union Horizon 2020 Programme (H2020-MG-2014-2015) under grant agreement n° 636202. The authors are grateful to the partners of the AGILE consortium for their contributions and feedback.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.ast.2019.04.039.

References

[1] J. Agte, O. De Weck, J. Sobieski, P. Arendsen, A. Morris, M. Spieck, MDO: assessment and direction for advancement - an opinion of one international group, Struct. Multidiscip. Optim. 40 (1–6) (2010) 17–33.
[2] B. Aigner, I. van Gent, G. La Rocca, E. Stumpf, L.L.M. Veldhuis, Graph-based algorithms and data-driven documents for formulation and visualization of large MDO systems, CEAS Aeronaut. J. (2018).
[3] N.M. Alexandrov, R.M. Lewis, Reconfigurability in MDO problem synthesis, part 1, in: Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, vol. 4307, AIAA Paper, 2004.
[4] N.M. Alexandrov, R.M. Lewis, Reconfigurability in MDO problem synthesis, part 2, in: Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, vol. 4307, AIAA Paper, 2004.
[5] J. Allison, M. Kokkolaras, P. Papalambros, Optimal partitioning and coordination decisions in decomposition-based design optimization, J. Mech. Des. 131 (8) (2009) 081008.
[6] R. Belie, Non-technical barriers to multidisciplinary optimisation in the aerospace industry, in: 9th AIAA/ISSMO Symposium of Multidisciplinary Analysis and Optimisation, 2002, pp. 4–6.
[7] T. Berners-Lee, J. Hendler, O. Lassila, et al., The semantic web, Sci. Am. 284 (5) (2001) 28–37.
[8] K.G. Bowcutt, A perspective on the future of aerospace vehicle design, in: 12th AIAA International Space Planes and Hypersonic Systems and Technologies, AIAA Paper, Norfolk, VA, 2003, pp. 2003–6957.
[9] R.D. Braun, Collaborative Optimization: an Architecture for Large-Scale Distributed Design, PhD thesis, Stanford University, 1996.
[10] P.D. Ciampa, B. Nagel, Towards the 3rd generation MDO collaboration environment, in: 30th Congress of the International Council of the Aeronautical Sciences, 2016.
[11] P.D. Ciampa, B. Nagel, The AGILE paradigm: the next generation of collaborative MDO, in: 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2017.
[12] P.D. Ciampa, E.H. Baalbergen, R. Lombardi, A collaborative architecture supporting AGILE design of complex aeronautics products, in: 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2017.
[13] R. Diestel, Graph Theory, Graduate Texts in Mathematics, vol. 173, 2010.
[14] K. Dykes, P. Rethore, F. Zahle, K. Merz, IEA Wind Task 37 Final Proposal Wind Energy Systems Engineering: Integrated RD&D, Tech. rep., International Energy Agency, 2015.
[15] A. Elham, Adjoint quasi-three-dimensional aerodynamic solver for multi-fidelity wing aerodynamic shape optimization, Aerosp. Sci. Technol. 41 (2015) 241–249.
[16] R. Elmendorp, R. Vos, G. La Rocca, A conceptual design and analysis method for conventional and unconventional airplanes, in: ICAS 2014: Proceedings of the 29th Congress of the International Council of the Aeronautical Sciences, St. Petersburg, Russia, 7–12 September 2014, International Council of Aeronautical Sciences, 2014.
[17] L. Etman, M. Kokkolaras, A. Hofkamp, P.Y. Papalambros, J. Rooda, Coordination specification in distributed optimal design of multilevel systems using the χ language, Struct. Multidiscip. Optim. 29 (3) (2005) 198–212.
[18] 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.
[19] A. Gazaix, F. Gallard, V. Gachelin, T. Druot, S. Grihon, V. Ambert, D. Guénot, R. Lafage, C. Vanaret, B. Pauwels, et al., Towards the industrialization of new MDO methodologies and tools for aircraft design, in: 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2017, p. 3149.
[20] 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.
[21] I. van Gent, B. Aigner, B. Beijer, J. Jepsen, G. La Rocca, Knowledge architecture supporting the next generation of MDO in the AGILE paradigm, Prog. Aerosp. Sci. (2019), accepted for publication.
[22] 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: 19th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2018.
[23] I. van Gent, G. La Rocca, M.F.M. Hoogreef, CMDOWS: a proposed new standard to store and exchange MDO systems, CEAS Aeronaut. J. (2018).
[24] S. Görtz, C. Ilic, A. Schuster, J. Jepsen, M. Leitner, M. Schulze, J. Scherer, M. Petsch, R. Becker, S. Zur, Multi-level MDO of a long-range transport aircraft using a distributed analysis framework, in: 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2017, p. 4326.
[25] J. Gray, K.T. Moore, T.A. Hearn, B.A. Naylor, A standard platform for testing and comparison of MDAO architectures, in: 8th AIAA Multidisciplinary Design Optimization Specialist Conference (MDO), Honolulu, 2012, pp. 1586–1611.
[26] J. Gray, T.A. Hearn, K.T. Moore, J.T. Hwang, J.R.R.A. Martins, A. Ning, Automatic evaluation of multidisciplinary derivatives using a graph-based problem formulation in OpenMDAO, in: 15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Atlanta, GA, 2014.
[27] J. 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.
[28] M.D. Guenov, S.G. Barker, Application of axiomatic design and design structure matrix to the decomposition of engineering systems, Syst. Eng. 8 (1) (2005) 29–40.
[29] A.A. Hagberg, D.A. Schult, P.J. Swart, Exploring network structure, dynamics, and function using NetworkX, in: 7th Python in Science Conference, SciPy2008, Pasadena, CA, USA, 2008, pp. 11–15.
[30] M.F.M. Hoogreef, G. La Rocca, An MDO advisory system supported by knowledge-based technologies, in: 16th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Dallas, Texas, 22–26 June 2015, AIAA 2015-2945, American Institute of Aeronautics and Astronautics (AIAA), 2015.
[31] M.F.M. Hoogreef, Advise, Formalize and Integrate MDO Architectures - a Methodology and Implementation, PhD thesis, Delft University of Technology, 2017.
[32] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1999) 359–392.
[33] 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 (2) (2012) 273–284.
[34] R. Lano, The N^2 Chart, Tech. rep., TRW, 1977.
[35] T. Lefebvre, N. Bartoli, S. Dubreuil, M. Panzeri, R. Lombardi, P. Della Vecchia, F. Nicolosi, P.D. Ciampa, K. Anisimov, A. Savelyev, Methodological enhancements in MDO process investigated in the AGILE European project, in: 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2017.
[36] T. Macquart, N. Werter, R. De Breuker, Aeroelastic tailoring of blended composite structures using lamination parameters, in: 57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 2016, p. 1966.
[37] J. Mariens, A. Elham, M.J.L. van Tooren, Quasi-three-dimensional aerodynamic solver for multidisciplinary design optimization of lifting surfaces, J. Aircr. 51 (2) (2014) 547–558.
[38] 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.

[39] J.R.R.A. Martins, C. Marriage, An object-oriented framework for multidisciplinary design optimization, in: 3rd AIAA Multidisciplinary Design Optimization Specialist Conference, Waikiki, Hawaii, USA, 2007.
[40] N.F. Michelena, P.Y. Papalambros, A hypergraph framework for optimal model-based decomposition of design problems, Comput. Optim. Appl. 8 (2) (1997) 173–196.
[41] E. Moerland, R.G. Becker, B. Nagel, Collaborative understanding of disciplinary correlations using a low-fidelity physics-based aerospace toolkit, CEAS Aeronaut. J. 6 (3) (2015) 441–454.
[42] E. Moerland, T. Pfeiffer, D. Böhnke, J. Jepsen, S. Freund, C. Liersch, G.P. Chiozzotto, C. Klein, J. Scherer, Y.J. Hasan, On the design of a strut-braced wing configuration in a collaborative design environment, in: 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2017.
[43] B. Nagel, D. Böhnke, V. Gollnick, P. Schmollgruber, A. Rizzi, G. La Rocca, J.J. Alonso, Communication in aircraft design: can we establish a common language? in: 28th International Congress of the Aeronautical Sciences, Brisbane, 2012.
[44] 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.
[45] T. Pfeiffer, B. Nagel, D. Böhnke, A. Rizzi, M. Voskuijl, Implementation of a heterogeneous, variable-fidelity framework for flight mechanics analysis in preliminary aircraft design, in: 60. Deutscher Luft- und Raumfahrtkongress, 2011.
[46] P. Piperni, A. DeBlois, R. Henderson, Development of a multilevel multidisciplinary-optimization capability for an industrial environment, AIAA J. 51 (10) (2013) 2335–2352.
[47] J. Rogers, DEMAID/GAA - enhanced design manager's aid for intelligent decomposition (genetic algorithms), in: 6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Sept 1996, pp. 4–6.
[48] R.S. Sellar, S.M. Batill, J.E. Renaud, Response Surface Based, Concurrent Subspace Optimization for Multidisciplinary System Design, AIAA paper 714:1996, 1996.
[49] S. Shahpar, Challenges to overcome for routine usage of automatic optimisation in the propulsion industry, Aeronaut. J. 115 (1172) (2011) 615.
[50] T.W. Simpson, J.R.R.A. Martins, Multidisciplinary design optimization for complex engineered systems: report from a national science foundation workshop, J. Mech. Des. 133 (10) (2011) 101002.
[51] D. Smith, M. Eggen, R. St Andre, A Transition to Advanced Mathematics, Nelson Education, 2014.
[52] J. Sobieski, J.S. Agte, R.R. Sandusky, Bilevel integrated system synthesis, AIAA J. 38 (1) (2000) 164–172.
[53] D.V. Steward, The design structure system: a method for managing the design of complex systems, IEEE Trans. Eng. Manag. (3) (1981) 71–74.
[54] S. Tosserams, A. Hofkamp, L. Etman, J. Rooda, A specification language for problem partitioning in decomposition-based design optimization, Struct. Multidiscip. Optim. 42 (5) (2010) 707–723.
[55] J.H. Vandenbrande, T.A. Grandine, T. Hogan, The search for the perfect body: shape control for multidisciplinary design optimization, in: 44th AIAA Aerospace Science Meeting and Exhibit, Reno, NV, 2006-928, 2006.
[56] D. de Vries, Towards the Industrialization of MDAO, Master's thesis, Delft University of Technology, 2017.
[57] T.C. Wagner, P.Y. Papalambros, General framework for decomposition analysis in optimal design, in: ASME Design Engineering, vol. 65-2, 1993, pp. 315–325.
[58] T. Wilschut, P.L. Etman, J.J. Rooda, I. Adan, Multi-level flow-based Markov clustering for design structure matrices, J. Mech. Des. (2017).

Software references (All web links were accessed on 7-4-2019)

Open-source packages

[59] CMDOWS: cmdows-repo.agile-project.eu.
[60] CPACS: github.com/DLR-LY/CPACS.
[61] KADMOS: kadmos-repo.agile-project.eu.
[62] OpenLEGO: github.com/daniel-de-vries/OpenLEGO.
[63] OpenMDAO: github.com/OpenMDAO/OpenMDAO.
[64] pyXDSM: github.com/mdolab/pyXDSM.
[65] RCE: rcenvironment.de.
[66] VISTOMS: mdo-system-interface.agile-project.eu.
[67] XDSMjs: github.com/OneraHub/XDSMjs.

Commercial packages

[68] MATLAB: mathworks.com/products/matlab.html.
[69] ModelCentre: phoenix-int.com/product/modelcenter-integrate.
[70] Optimus: noesissolutions.com/our-products/optimus.