Composing MDAO symphonies

Graph-based generation and manipulation of large multidisciplinary systems

van Gent, Imco; La Rocca, Gianfranco; Veldhuis, Leo

DOI
10.2514/6.2017-3663

Publication date
2017

Document Version
Accepted author manuscript

Published in
18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference

This is an Accepted Manuscript of an article published in: 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 5-9 June 2017, Denver, Colorado, . ISBN: 978-1-62410-504-3 Available online: https://arc.aiaa.org/doi/10.2514/6.2017-3663

Composing MDAO symphonies: graph-based generation and manipulation of large multidisciplinary systems

Imco van Gent*, Gianfranco La Rocca†, and Leo L. M. Veldhuis‡
Delft University of Technology, Kluyverweg 1, 2629 HS, Delft, The Netherlands

This paper proposes a novel methodology and its software implementation, called KAD-MOS (Knowledge- and graph-based Agile Design for Multidisciplinary Optimization Sys-tem), which aims at increasing the agility of aircraft design teams that perform collaborative multidisciplinary design analysis and optimization (MDAO) by means of graph manipula-tion techniques. By agility, the ease and flexibility to assemble, adapt and adjust MDAO computational systems is intended here, as necessary to better fit the 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 its actual implementation, especially to be able to compose large and complex systems in multidisciplinary design teams. This specification system is under development as part of the EU project AGILE where a new generation of aircraft MDAO systems is investigated to support collaboration of heteroge-neous teams of experts. KADMOS improves the agility of the design team in three ways: 1) reducing the set-up time required to compose large and complex MDAO models, 2) enabling the systematic inspection and debugging of this model, and 3) manipulating the model for automated creation and reconfiguration of optimization strategies, including the accompanying executable workflow. This is achieved by means of a graph-based analysis system that combines different existing advantageous techniques for performing MDAO, such as the use of a single shared data schema containing a parametric representation of the aircraft, knowledge-based technologies, and simulation workflow (SWF) software packages. Two MDAO case studies will be presented in the paper. The first case study is based on a simple analytical problem, generally used in literature for MDAO benchmarking studies. The second case study concerns a detailed wing aerostructure design using a collection of wing design tools. While the simple and compact analytical problem is used in this pa-per to demonstrate the functionalities of the tool, the wing design case demonstrates the capability of KADMOS to support quick formulation, (re)configuration, and execution of MDAO workflows using distributed and heterogeneous sets of analysis tools.

I. Introduction

Past research indicates that MDAO can offer huge benefits in creative design. Boeing Phantom Works scientists[1, 2] estimate that MDAO can offer 8-10% performance gains for innovative aircraft design and even 40-50% gains for designing radically new and undeveloped concepts in the traditional measures of performance, such as range, speed, payload capacity.[3] Despite the high potential gains, MDAO is not as widely used as one would expect. Both technical and non-technical barriers[4–7] are hampering its full exploitation, as discussed below in this section.

To get a better understanding of the current MDAO challenges it is convenient to refer to Figure 1, where the different parts of the MDAO-based development process and their relation are illustrated. The development process in this figure can be roughly cut in two parts. On the left side one has the formulation phase where the MDAO problem is defined (or provided) and a formal (not executable) model of the MDAO

*Ph.D. Student, Flight Performance and Propulsion Section, Faculty of Aerospace Engineering, AIAA student member.
†Assistant Professor, Flight Performance and Propulsion Section, Faculty of Aerospace Engineering, AIAA member.
‡Full Professor, Head of Flight Performance and Propulsion Section, Faculty of Aerospace Engineering, AIAA Member.

system is generated: the formal specification. This specification is the blueprint of the executable workflow, which is integrated and executed during the execution phase of the MDAO-based development process (right part of Figure 1) to find the actual optimal design. Hence, the formal specification of the MDAO model generated in the formulation phase constitutes the neutral representation of the MDAO system, which needs to be translated into a software-specific execution, as is indicated with the simulation workflow (SWF) definition block in the figure.

In a realistic design case the optimization is generally performed more than once, because the analysis of the design that is found after a certain run will provide new insights. These insights will be translated to an adjustment of the MDAO problem formulation (e.g. change of objective, addition of constraints, etc.) and a reconfiguration of the MDAO process (e.g. addition, removal, replacement of analysis tools, or change of optimization strategy). This process of problem adjustment and process reconfiguration is iterated until a satisfactory design is found (or the project deadline has been reached).

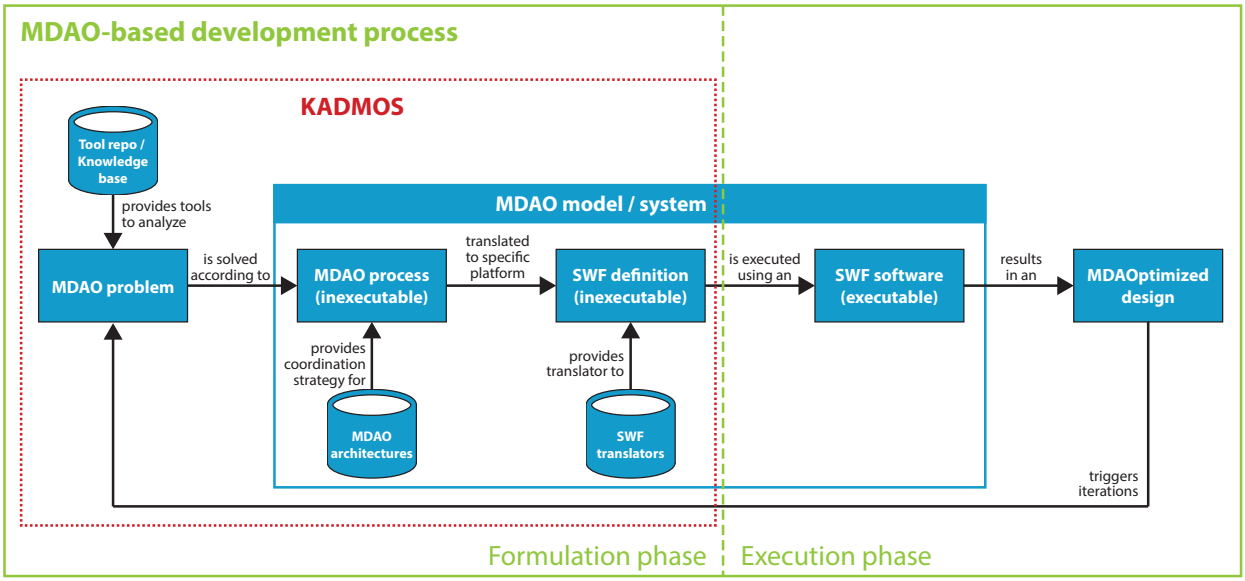


Figure 1: Overview of MDAO terminology for the MDAO-based development process used in this paper. The process can be divided in two phases: formulation (left) and execution (right). The domain of KADMOS within the development process is indicated by the red box.

One of the most critical technical barriers for MDAO comes from the large (and continuously increasing) size of typical MDAO problems. In the words of Pate et al.[8] 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 survey[9] of MDAO-oriented research projects, performed at the German Aerospace Center (DLR) and in a European context, has shown that 60-80% of the project time was used to integrate the numerous tools and set up a first executable data flow for a required set of system-level outputs.

This integration challenge hinders the exploitation of new tools and the reuse of existing tools, as for every MDAO problem the tools have to be refitted for a new MDAO workflow. On a higher level the different tools have to be implemented using one of the available MDAO architectures,[10] such as Multidisciplinary Feasible (MDF) or Individual Discipline Feasible (IDF). The architecture selection has to be done in an early stage and it is not easy to adjust the architecture once the SWF has been created (a notable exception being an implementation in OpenMDAO[11]), although a different architecture might provide benefits in computation time. Thus, both on the tool level and the architecture level the MDAO-based development process is not agile enough yet.

A less technical, rather organizational barrier stems from the fact that the current aircraft conceptual and preliminary design practice at major commercial aircraft developers is geared towards the use of empirical

models based on tube-and-wing (TAW) jetliners. In addition, the organizations are also set up based on the developments of past TAW configurations of the previous decades. A major strength of the TAW configuration is that it allows a relatively strong decoupling of different disciplines (aerodynamics, structures, etc.) and different systems (wing, fuselage, tail, engines). Naturally, collaboration between different disciplines is required to have a balanced and robust design with the right system couplings, however, this collaboration is not at the same level as the disciplinary collaboration that is required to perform MDAO. Hence, there is a big difference in the required level of formal specification between performing 'manual' design optimization and 'automatic' design optimization.

One can conclude from both the current design practice and the observations by Pate et al.[8] that implementation of a large MDAO model in a practical design situation is problematic. If used at all, most MDAO workflows are generally tailor-made to their application, difficult to reconfigure, and not created using an adequately standardized system and methodology.

This lack of agility versus the potential benefits of the MDAO method with respect to existing design methods is effectively summarized in Figure 2. Flager and Haymaker[12] have performed research into the design process metrics of both a legacy (current) design method and an MDAO-based process for the design of a hypersonic vehicle by Boeing.[1,2] The main hurdle indicated in Figure 2 is the long set-up time of the MDAO workflow (14 weeks) compared to the set-up time for the legacy method (6 weeks). The potential of the MDAO-based process is also clear: once the set-up of the MDAO workflow is complete an enormous amount of iterations can be performed compared to the legacy method. Furthermore, the MDAO-based process enforces the whole team to spend more time on Specification and Reasoning (see legend of Figure 2 for a definition of both terms) while less time is spent on Execution and Management, however, these benefits come with an additional risk as well: in the MDAO design process the Specification phase is longer and this postpones the moment that one obtains the first results.

This longer specification time is caused by the fact that the MDAO system requires a more formal specification; to automate the system of disciplinary analyses the connections between those analyses have to be defined down to the smallest detail before a functioning system can be implemented. In industrial design situations, where deadlines are strict and risks need to be minimized, the risky postponement of the first results in combination with the time-consuming formal specification forms a major hurdle. Therefore, the focus of the system development presented in this paper is on the formulation phase. The goal is to enable the formal specification of any MDAO problem and process and to include an automated connection to an executable workflow using SWF software packages[a].

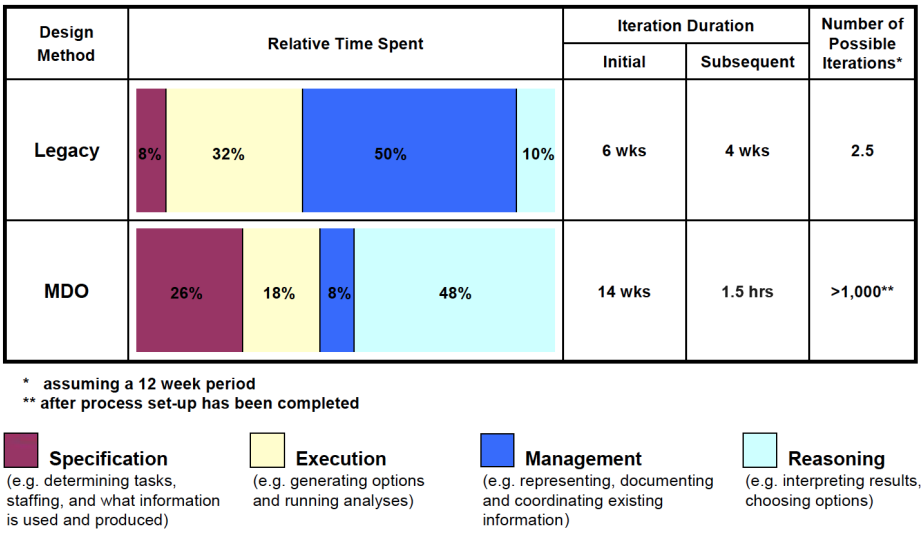


Figure 2: Comparison of legacy and MDAO design process metrics for the design of a hypersonic vehicle[12]

The formal specification can be further clarified by considering a well-known analogy to the performance of MDAO, which was introduced by Cramer:[7] symphonic orchestras. Just like with MDAO, a symphonic

[a]The SWF software packages used in the AGILE project are provided by two different partners: RCE by the German Aerospace Center (DLR) and Optimus by Noesis Solutions

orchestra attempts to provide a music performance (optimal design) by combining a lot of different musicians (disciplines) using vastly different instruments (disciplinary tools). Usually, the instruments are "integrated" by a composer (head designer) while the balancing of all musicians is left to the conductor (optimizer). In this analogy, a key component for music performance in large groups is still missing. Symphonic orchestras can play very long, complex pieces thanks to a key system in place: music notation. Musical score provides the formal specification of the symphonies played by orchestras. Clearly, without a proper music notation system, performing large, complex pieces would be near to impossible. The same holds for large, complex MDAO problems. If we try to formulate and solve these MDAO problems without a central notation system, then it is like we try to compose and conduct an orchestra without being allowed to use sheet music.

It is exactly this lack of formal specification in combination with the opportunities it offers to formalize large, complex MDAO systems that motivated the development of KADMOS, the novel graph-based system for agile configuration and execution of MDAO frameworks presented in this paper. KADMOS aims at providing a system composition tool for these MDAO systems, thereby facilitating the formal specification of the automated design process required for MDAO-based developments. Quantitatively, KADMOS has reached its goal when the set-up time for an MDAO workflow is reduced significantly, however (and maybe even more importantly), the goal is especially reached when an MDAO support system is created that provides enough confidence to perform MDAO-based design in large, heterogeneous teams of experts.

II. Methodology

KADMOS is part of the bigger framework developments in MDAO that is taking place within the EU project AGILE[b]. In this project a new generation of aircraft MDAO systems is investigated to support collaboration of heterogeneous teams of experts. This heterogeneity of the design team is a key assumption in the methodology. Indeed, based on the vast experience of the consortium members, it is acknowledged that no complex MDAO model can be fully integrated or understood by a single person or small team, but only by an heterogeneous team of experts, including discipline specialists, aircraft designers, and MDO system architects.

A. KADMOS functional requirements

The aim of KADMOS is to enable the formal specification of large, complex MDAO models. To this purpose, the following three basic functionalities have been identified:

- **Formalize:** The system should support the formal specification of large, complex MDAO models. Hence, its methods should be scalable while always storing the full MDAO model down to the smallest detail. Furthermore, the heterogeneity of the team demands that the system supports a modular specification approach, such that each individual expert can provide and control its own building block, without having to understand the full system.

- **Inspect:** The system should enable different experts in the team (i.e. disciplinary specialists, integrators) to inspect parts of the model that are relevant to them, in order to validate model completeness and correctness, thereby increasing the level of trust in the model.

- **Manipulate:** The system should also be able to automatically manipulate the content stored within. With a fully machine-interpretable implementation of the MDAO problem formulation, computerized manipulations will reduce the chances of errors or inconsistencies in the model by replacing repetitive error-prone human tasks. Simple manipulations would be the adjustments of input or output variables of a disciplinary tool. More advanced manipulations are the automated transformations required to go from a database of interconnected tools to an executable MDAO workflow according to a certain MDAO architecture.

The focus of the work presented in this paper is on the formalize and manipulate requirements of the system and how KADMOS uses graphs to enable this. The inspection requirement is closely related to the visualization of large MDAO systems, which is the topic of another paper.[13] A simple mathematical problem is used to explain the technical functionalities (Section IV.1), while an aerostructural wing design problem

[b]AGILE project website: `http://www.agile-project.eu/`

(Section IV.2) is used to showcase the capability of KADMOS to deal with a complex MDAO system of industrial relevance.

B. KADMOS key technologies

KADMOS exploits the benefits of four different elements that are currently used or under development in the MDAO research domain:

- Central data schema
- Knowledge-based technologies
- Graph-based approach
- SWF software packages

The role of these elements will be briefly described in the upcoming sections in order to provide a top-level overview of KADMOS. The main steps performed by KADMOS in the overall process of setting up an MDAO system are shown in Figure 3. The figure shows the various graphs generated and manipulated by KADMOS, based on the input by the various MDAO system stakeholders. Examples based on the Sellar problem[14] are shown in the most right column.

1. *Central data schema*

The idea of a central data schema is to have a standardized structure available, which can be used to store and retrieve all data that needs to be passed around in a workflow. The key advantage of having such a schema is depicted in Figure 4: with a central data schema the number of possible tool interfaces is reduced to a minimum. A data schema can be defined using different formats. At the moment KADMOS supports the use of XML-based formats. Main motivation for the support of XML is the use of XML-based Common Parametric Aircraft Configuration Schema (CPACS)[15] in the AGILE project.

CPACS is an XML schema data definition for a complete air transportation system that can be used to connect different analysis tools. CPACS was developed by the German Aerospace Institute (DLR)[17] to enable engineers to exchange information between their tools. It describes the characteristics of aircraft, rotorcraft, engines, climate impact, fleets and mission in a structured, hierarchical manner. KADMOS exploits the fact that every tool in a workflow has to read its input and writes its output to the same schema. Within AGILE it was decided to make all disciplinary tools CPACS compatible, i.e. to map the input and output of each tool to specific nodes in the CPACS structure. Then all the 'CPACS-ized' tools are collected in a tool repository, see top left cell in Figure 3. Thanks to this arrangement of the tools a complete interlinked overview can be created by KADMOS in the form of a graph structure, as shown in the first row of Figure 3. More details on this first KADMOS graph are provided in Section III.A.

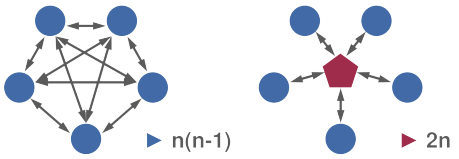


Figure 4: Connections between tools with and without a central data model[16]

2. *Knowledge-based technologies*

The second element used by KADMOS are so called knowledge-based technologies,[18] in particular techniques for knowledge capture and formalization. These knowledge-based technologies form the starting point of KADMOS, as the system operates on a knowledge base that is created and maintained by the design team. A knowledge base is essentially a database, but structured in such a way that one can store and retrieve information about a design that is normally left implicit. The algorithms in KADMOS can use this information to make the MDAO set-up process more agile.

As can be seen in the top left of Figure 3, the knowledge base is the first main input for KADMOS. Four different file types are stored in the KADMOS knowledge base at the moment.

- **schema definition:** an XML-based definition of the central data schema.
- **tool info file:** a file storing meta-data of a tool, such as tool provider, license information, fidelity level, execution time, accuracy, etc.

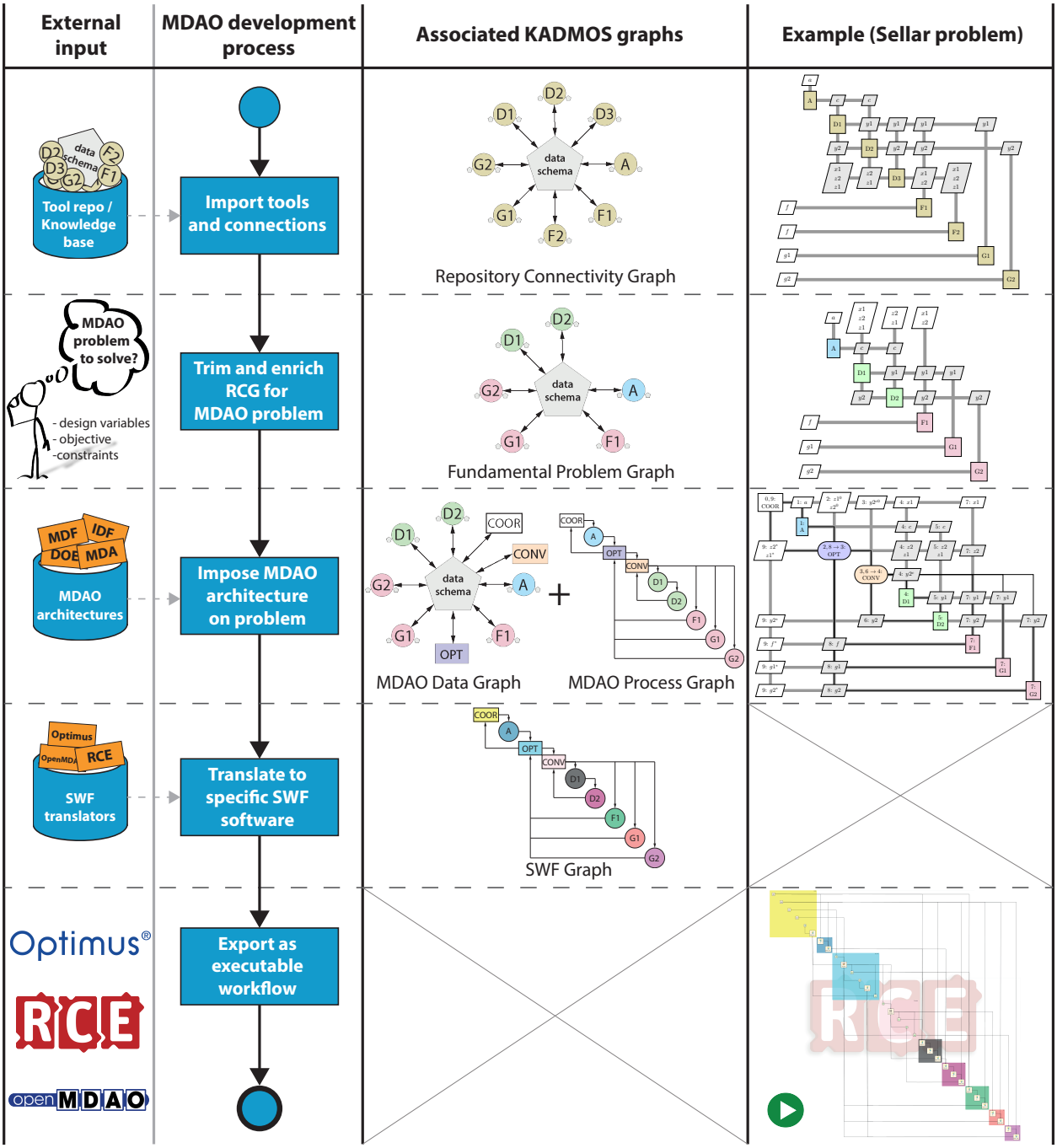


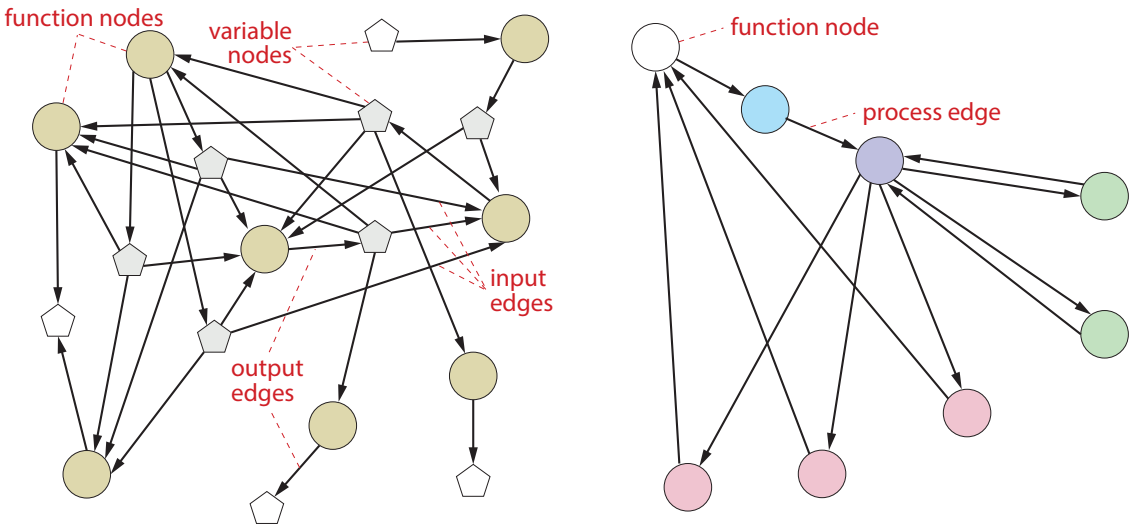
Figure 3: Top-level overview of KADMOS and its relation to the MDAO-based development process depicted in Figure 1 (all visualizations are based on the Sellar problem[14] described in Section IV.1)

- **tool input file:** an XML file containing only those data elements from the central data schema that are required to run the disciplinary analysis.
- **tool output file:** an XML file containing only those data elements from the central data schema that are written as output by the disciplinary analysis.

The four file types currently defined are used by KADMOS to import the knowledge base and start its graph-based analysis. Future developments might require the use of additional file types, for example to include information on tool sensitivities, however, the current file types form a minimal requirement for tool definition in the knowledge base. Details of the knowledge base, which is a separate class in KADMOS, are not provided in this paper. More information can be found in the work by Makus.[19]

3. Graph-based approach

The knowledge-based design process that is supported by KADMOS extends the graph-based approach proposed by Pate et al.[8] The work of Pate et al. was focused on the data connectivity between different tools, while KADMOS takes the graphs a step further to also include the graphs belonging to the solution strategy that have both a data and process connectivity aspect (this topic is mentioned, but no algorithms are shown for automatic determination of the solution strategy by Pate et al.). Furthermore, the graph-based approach is enriched with additional elements from the AGILE project. The complete approach is visualized in the third column of the top-level overview of Figure 3. A graph-based approach means that graphs[20] are used to store and manipulate data. In KADMOS, graphs are used to store data on the MDAO model because this format enables the creation, debugging, and manipulation of that same model. A specific form of graphs, called directed graphs, is used, of which two examples are shown in Figure 5. Directed graphs consist of nodes and directed edges. In KADMOS, the nodes represent either executable blocks that process inputs to outputs (i.e. disciplinary tools, optimizers, convergers), or one of the variables from the central data schema. The nodes representing executable blocks are called 'function nodes', while the nodes representing elements from the data schema are referred to as 'variables nodes'. A more elaborate description of the KADMOS graphs is provided in Section III.



(a) Data graph

(b) Process graph

Figure 5: Examples of the two KADMOS graph classes (data and process graph) and terminology used for different components of these graphs

Graph-based analysis has previously been used to analyze MDAO problems, for example to determine the required derivatives[21] or graph partitions for parallel processing.[22] To the authors' knowledge, KADMOS is the first graph-based package that supports the full chain of the MDAO-based development process going from disciplinary tools integrated in a knowledge base using a central data model to automatically creating

distributed executable workflows in SWF packages. Related work was performed by Alexandrov et al.,[23,24] Balachandran and Guenov,[25] and Hoogreef.[26]

4. SWF software packages

The open-source SWF software package RCE (Remote Component Environment), developed by the DLR,[27] is the only package currently implemented in KADMOS. Support for a second SWF software package will be included in the future: Optimus by Noesis Solutions. Both these packages are part of the AGILE project. The ability of KADMOS to script executable SWF for these specific packages is a key element of the system as recent developments within AGILE for these packages enable the execution of SWFs across distributed networks. This means that a heterogeneous collection of analysis tools can be executed on servers all over the world while respecting the intellectual property rights and security measures (i.e. firewalls) of all partners involved.[28] This collaborative software architecture implemented in the SWF packages is one of the AGILE project developments that KADMOS benefits from.

C. KADMOS' software architecture

KADMOS is programmed in Python using an object-oriented approach. The UML class diagram of the system is shown in Figure 6. KADMOS has three main classes:

- `KnowledgeBase`
- `KadmosGraph`
- `RceWorkflow`

The `KnowledgeBase` class has been developed to import the different files into the knowledge base in KADMOS. The final output of the `KnowledgeBase` class is the repository connectivity graph (RCG), which is the graph where the data connections between the different tools in the repository are stored, see first row of Figure 3. The RCG is an object belonging to the second main class of KADMOS: `KadmosGraph`.

Graph objects in KADMOS can belong to different classes, but all of them are a subclass of `KadmosGraph`. `KadmosGraph` is itself a subclass of the `nx.DiGraph` class, meaning that the open-source NetworkX[29] (`nx`) package is the main graph package used to create and manipulate graphs. `nx.DiGraph` is the directed graph object from NetworkX and forms the basis of the `KadmosGraph` class and its subclasses. The subclasses of `KadmosGraph` correspond to the different graph objects shown in Figure 3, which are further discussed in Section III. Each of these graph objects contain the specific graph manipulation algorithms to go from one graph type to the other (e.g. `RepositoryConnectivityGraph` to `FundamentalProblemGraph` as indicated in Figures 6 and 3). These algorithms are not explained in detail here, but will be shown in more detail in the Sellar case study in section IV.1.

The `RceGraph` in the bottom right of Figure 6 is still a directed graph object in KADMOS. This graph object is translated into an executable RCE workflow using the `RceWorkflow` class, as is visualized in the two last rows of Figure 3. This translation has been programmed such that any `RceGraph` object can be directly translated into the workflow file, provided that the graph object only contains nodes that are known to the system. Hence, any adjustments to the MDAO process and SWF can be done inside KADMOS using its graph manipulation algorithms, without having to adjust anything in the SWF software. This approach gives the user more flexibility in the adjustment of analysis tools or MDAO architectures, since the full graphs will be processed automatically after changes at any position in the development process.

A final package worth mentioning is used in the `KadmosGraph` class by the `create_xdsm()` method. This method uses an extended version of the `XDSMwriter`[30,c]. Combining this writer with KADMOS enables a fully automated creation of visualizations for the different KADMOS graphs. These visualizations are shown in the fourth column of Figure 3. The XDSMs created by these methods are static PDF files, however, a dynamic visualization using is also supported by KADMOS, though not presented in this paper.[13]

In conclusion, the software architecture of KADMOS depicted in the UML class diagram in Figure 6 provides a continuous flow that runs from the `KnowledgeBase` class to the `RceWorkflow` class. This flow matches the MDAO-based development process shown in Figures 1 and 3. The KADMOS package has been published open source under the Apache License 2.0.[d]

[c]The XDSM writer is available at: `http://mdolab.engin.umich.edu/content/xdsm-overview`
[d]KADMOS is available at: `https://bitbucket.org/imcovangent/kadmos`

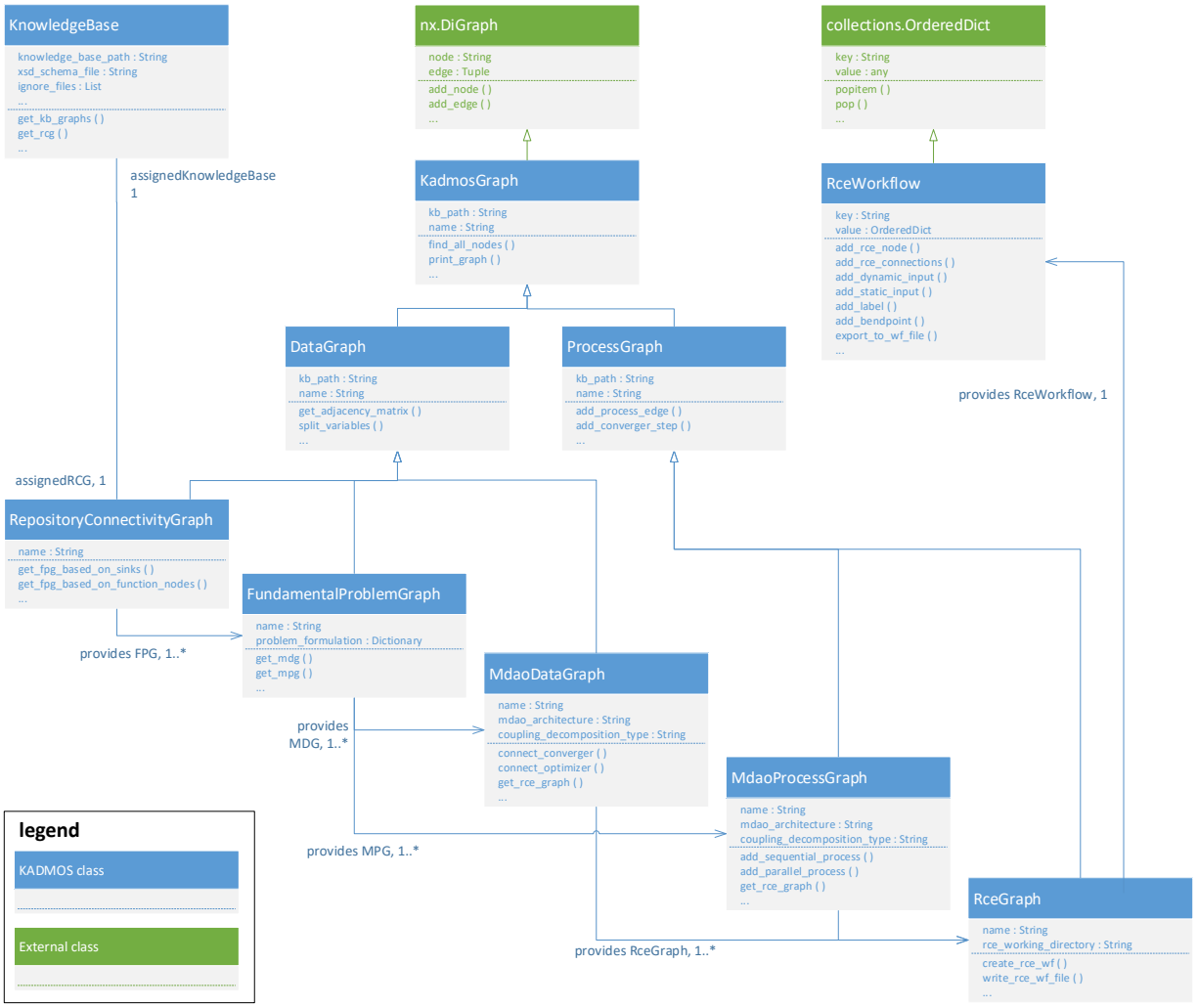


Figure 6: UML class diagram of KADMOS (blue) including its dependence on external Python classes (green). The flow of different graph classes in the bottom matches the MDAO-based development process shown in Figure 3

.

III. KADMOS graphs

In this section a more detailed description of the different KADMOS graphs is provided. This is done to offer insight into how the graphs should be defined in order to support the full process from a collection of tools in a knowledge base to an automatically created executable workflow. For the sake of brevity, only key elements of the different graphs are discussed here. The key graph to be defined in the system is the fundamental problem graph (FPG). This graph defines the MDAO problem that needs to be solved and should be crafted using a combination of the designer's knowledge of the problem and available tools, and the graph manipulation methods provided by KADMOS. Ultimately, the FPG should be defined such that the KADMOS graph manipulation algorithms that impose an MDAO architecture[10] on the graph (see third row in Figure 3) can operate on it, while at the same time provide enough operational flexibility to adjust the problem formulation. In KADMOS, different MDAO architectures are available, such as MDF and IDF. More on the MDAO architectures will be described in Section III.C.

As is shown in Figure 5, two different classes of directed graphs are created by KADMOS. A graph that connects function nodes with variable nodes to indicate I/O relations is called a 'data graph' (Figure 5a). The second graph class is the 'process graph' (Figure 5b): these type of graphs only contain function nodes and are used to store the process of node execution. The following graph types belong to the MDAO development process depicted in Figure 3:

- Data graphs:
  - **Repository Connectivity Graph (RCG):** Using the repository with CPACSized analysis tools in the knowledge base, a directed graph can be created that links all the I/Os of different analyses. Such a graph would represent all the information present in the knowledge base. It is worth noting that the RCG shown in Figure 3 can be visualized using the data flow aspect of the eXtended Design Structure Matrix (XDSM)[30] for relatively small problems (see last column of the figure), however, realistic aircraft design tools could use and produce data concerning hundreds of elements from the central data schema. This information would still be stored in the RCG, though visualizing it is much more challenging and a topic specifically addressed in another paper.[13] For convenience, the relatively small Sellar problem[14] is used in Section IV to illustrate the working principles of KADMOS.
  - **Fundamental Problem Graph (FPG):** Once all the tools in the repository have been connected, the next graph should contain only the variables and tools required to solve the MDAO problem at hand (actually, it can also be a monodisciplinary optimization or just a convergence that needs to be implemented). Indeed, the FPG is a subset of the nodes and edges in the RCG. The translation from RCG to FPG is more elaborately discussed in Section III.B.
  - **MDAO Data Graph (MDG):** The FPG is used by KADMOS to impose an MDAO architecture on the MDAO model. The adjusted nodes and edges between the function and variable nodes which are required to impose the architecture form the MDG. Thereby, the MDG contains the information on how data is interconnected between tools from the repository and newly introduced MDAO architectural elements like convergers and optimizers. The information stored in the MDG is exactly what one visualizes with the data flow in an XDSM.
- Process graphs:
  - **MDAO Process Graph (MPG):** As this is a process graph, it does not contain any data nodes, but merely the executable blocks (disciplinary analysis tools and architectural elements) and their execution order. The information stored in the MPG is exactly what one visualizes with the process flow in an XDSM. Therefore, the superposition of the MDG and MPG lead to an XDSM, as is shown in Figure 3.
  - **SWF Graph:** The combination of MDG and MPG can be translated into a single executable SWF graph matching the SWF software platform of choice. With the right export functions this SWF graph can be transformed into a workflow file that can be opened and executed in the SWF software (currently only RCE is supported by KADMOS).

In addition to the general description above a more detailed description of the key graph types is provided in the next sections.

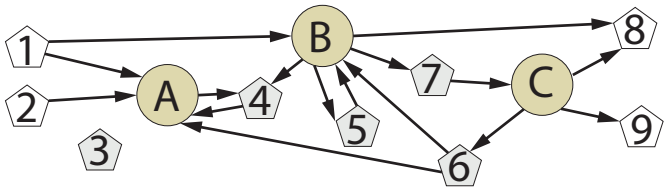
A. Repository connectivity graph

Key elements of the RCG are the unique naming of its nodes and the categorization of the different nodes based on their connectivity. The unique node naming for variables is performed by translating the leaf elements of the (XML) data schema into unique strings using a combination of the unique identifiers (UIDs) and XML XPaths called 'UXPaths'. This is a crucial step in the creation of the RCG, as the correct coupling between different tools depends on the consistent naming of the variable nodes.

After the RCG has been created, the different nodes have to be categorized. The two main categories for KADMOS graph nodes are *variable* and *function*. The subcategory of a node can be determined based on the incoming and outgoing edges of each node (referred to as 'indegree' and 'outdegree' in graph theory). Table 1 lists the different subcategories for variable and function nodes.

Table 1: Subcategories of the different graph nodes based on their indegree and outdegree and some illustrated examples in a graph (below table)

Node category	Node subcategory	Indegree	Outdegree	Example node	Allowed in FPG?
variable	hole	0	0	3	no
	supplied input	0	1	2	yes
	supplied shared input	0	>1	1	yes
	output	1	0	9	yes
	collision	>1	0	8	no
	coupling	1	1	7	yes
	or				
	pure circular coupling			5	no
	shared coupling	1	>1	6	yes
	or				
	shared circular coupling			-	no
	collided coupling	>1	1	-	no
	or				
	collided circular coupling			4	no
	collided shared coupling	>1	>1	-	no
	or				
	collided shared circular coupling			-	no
function	hole	0	0	-	no
	inputless	0	>0	-	yes
	outputless	>0	0	-	no
	complete	>0	>0	A,B,C	yes



Example nodes (see fifth column of the table)

Variable nodes with at least one incoming and outgoing edge can always fall within one of two subcat-

egories, see Table 1. These variables could be circular or non-circular. A variable is denoted circular if it has at least one pair of incoming and outgoing edges connected to the same function node. Non-circular variables never have a pair connected to the same function node. In the RCG circularity is allowed, but since it needs to be resolved in the FPG, circular variables have to be identified accordingly. Once the RCG is defined, the next step is to transform it into an FPG.

B. Fundamental problem graph

Depending on the size and complexity of the RCG several manipulations might be required to create the FPG. Before an FPG can be used to impose an MDAO architecture, it has to meet four requirements which need their respective actions (→):

1. Smallest size possible → node removal and contraction
2. No nodes in invalid subcategories → node/edge removal and node splitting
3. Contain information to impose architecture → add problem formulation and problem role attributes
4. Basic analysis order → add function nodes order with minimal feedback (or other ordering requirement)

1. FPG requirement 1: Smallest size possible

The size of the FPG can be decreased in several ways. The most straightforward way would be to remove nodes: either complete functions and their associated inputs and outputs can be deleted, or individual output nodes of certain functions can be removed if they are deemed irrelevant by the designer for the MDAO problem at hand.

A second way in which the graph size can be decreased is by function node contraction. If it is known that a group of function nodes will always have to be executed as a sequential or parallel process, then these functions can be contracted into one merged function node. This does mean that the algorithm that imposes the MDAO architecture will consider this contracted function node as a single function again.

2. FPG requirement 2: No nodes in invalid subcategories

The allowed node subcategories in the FPG are listed in the last column of Table 1. In short: collisions, circularity, and holes are not allowed in the FPG. If such nodes are present in the graph they have to be resolved. This can be done in three different ways: node removal, edge removal, and node splitting.

Node and edge removal are straightforward ways to resolve collisions. A more complex method is to use node splitting. Collided or circular nodes might occur naturally in the FPG, for example, multiple tools might write a value for the wing span as output (collision), or a tool might use (i.e. input) and adjust (i.e. output) the wing span if it changes the wing geometry (circular). However, the KADMOS graph algorithms which impose the MDAO architecture do not allow these type of nodes. Therefore these nodes can be split into instances, as is illustrated in Figure 7. A single node in an invalid subcategory will be split into separate variable node instances. KADMOS contains methods to automatically split invalid FPG variable nodes based on their connectivity and (for some cases) the analysis order.

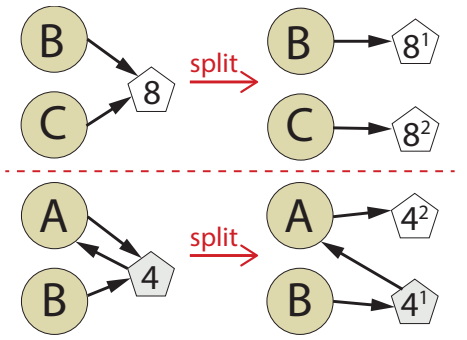


Figure 7: Illustration of two node splitting examples using nodes from Table 1.

3. FPG requirement 3: Contain information to impose architecture

In addition to removal and splitting of nodes, some FPG variable nodes also need to be enriched by specifying the 'problem role' attribute. Possible problem role attributes are: design variable, objective variable, constraint variable, and state variable. At least all the outputs of the graph have to get one of the problem roles assigned, otherwise they are considered irrelevant outputs for the problem at hand.

Other information that is required to manipulate the FPG are the MDAO architecture to be imposed, the preferred decomposition of circular dependencies between functions, and some more detailed settings per architecture. The possible MDAO architectures in KADMOS are discussed in Section III.C. Decomposition of the circular dependencies can be done based on Gauss-Seidel or Jacobi convergence (as defined in Martins and Lambe[10]).

4. FPG requirement 4: Basic analysis order

The final FPG requirement concerns the basic analysis order. This is not necessarily the same analysis order defined in the MPG, but it is an input for the MDAO architecture algorithm so that KADMOS can categorize the different functions in three groups, as depicted in Figure 8: pre-coupling, coupled, and post-coupling functions. Pre-coupling functions are the first functions in the analysis order that do not have any data fed back to them, but only forward data (i.e. typically initial geometry and geometry manipulations of the central data schema). The coupled functions are the group of functions that contain at least one circular dependency (i.e. an aerodynamic solver and structural analysis of a wing which both depend upon each others output of loads and wing weight). The post-coupling functions are functions that use the outputs of the pre-coupling and coupled functions to calculate MDAO model output (i.e. typically objective and constraint functions). At the moment the basic analysis order has to be specified manually, but it is foreseen that future extensions of KADMOS will include methods for the automatic determination of the basic analysis order. Though it is worth noting that in the case studies until now a logical order of the function nodes could always be defined intuitively.

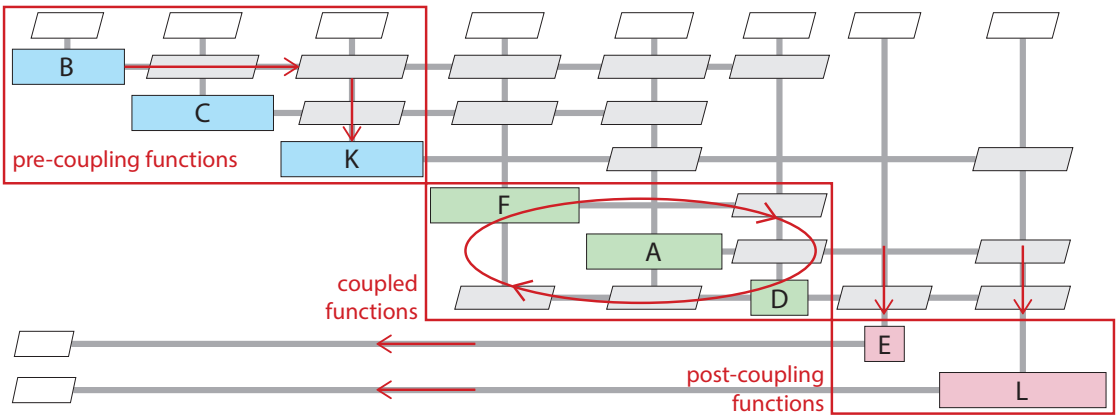


Figure 8: Automatic categorization of the FPG functions based on the basic analysis order: [B, C, K, F, A, D, E, L].

If a valid FPG has been crafted which meets the four basic requirements, then the subsequent graphs (see Figure 3) are created fully automatically by KADMOS.

C. MDAO data graph, MDAO problem graph, and simulation workflow graph

Currently, six different MDAO architectures can be imposed on the FPG, namely:

1. **Tool chain test:** where the tools are simply put all in parallel or sequence simply to test if they all run as expected. Hence, no convergence or optimization is performed.

2. **MDA:** where any circular dependencies between coupled tools will be converged for the given inputs. No optimization is performed with this strategy and tools can either be run in parallel or sequence. With parallel execution feedforward couplings between coupled functions will also be converged.

3. **DOE:** where a design of experiments is automatically imposed on the FPG.

4. **Optimizer:** where a straightforward optimization is performed on the FPG, without solving any circular dependencies (i.e. this is equal to MDF without the converger nested inside the optimizer).

5. **MDF:** where the MDF strategy[10] is imposed on the FPG. The converger inside the optimizer can be set to either Gauss-Seidel (sequential) or Jacobi (parallel) execution of the coupled tools.

6. **IDF:** where the IDF strategy[10] is wrapped around the FPG. All tools are parallelized and an additional consistency constraint function is added to converge coupling variables.

The algorithms for the MDG and MPG creation are further discussed in the Sellar problem use case (Section IV.1). The MDG contains the information on how data is interconnected between tools and newly introduced MDAO architectural elements like convergers and optimizers. Similar to the problem role attribute in the FPG, the MDG nodes can get 'architecture role' attributes. The possible attributes are listed in Table 2. The architecture roles for function nodes are also applicable for the MPG, since the MPG does not contain any data nodes, but merely the executable blocks (analysis tools and architectural elements) and in which order they should be executed. In essence, the information stored in the MDG and MPG represents a neutral specification of the execution process that should be implemented in a specific software platform.

Table 2: Lists of possible architecture roles for function (left) and variable nodes (right)

Function nodes	Variable nodes
coordinator	initial guess coupling variable
optimizer	final coupling variable
converger	coupling copy variable
DOE	initial guess design variable
pre-coupling analysis	final design variable
pre-iterator analysis	final output variable
post-iterator analysis	consistency constraint value
coupled analysis	DOE input sample list
post-coupling analysis	DOE output sample list
consistency constraint function	

Hence, referring back to the symphonic orchestra analogy from Section I: using graphs as music notation system, the MDG and MPG are the sheet music of our symphony which has been created by the designer/composer using KADMOS as his composition tool. Actually playing the symphony would require to translate the neutral specification into a workflow file that can be opened and executed in the SWF platform, as indicated in the bottom of Figure 3 and discussed in the introduction of this section.

The KADMOS graph definitions and algorithms have been tested on multiple use cases within the AGILE project. In this paper one illustrative use case is presented (the Sellar problem in Section IV.1) and one realistic aircraft design case (the wing design problem in Section IV.2).

IV. Results

Although KADMOS is still under development, the first research goal has been achieved: to test the ability of using a graph-based approach to go from a repository of disciplinary tools to an executable workflow using different MDAO architectures for the same MDAO problem. This capability has first been developed and tested using the Sellar problem and is presented in Section IV.1. The next case study has demonstrated the scalability of the KADMOS algorithms by going through the same chain for a larger repository. This wing design case study is briefly described in Section IV.2.

1. Proof-of-concept: Sellar problem

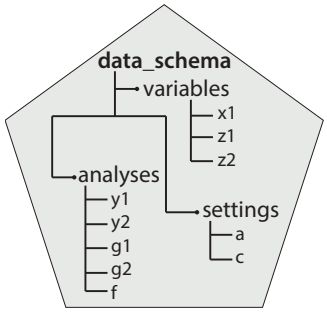
The Sellar problem[14] is a small, analytical MDAO problem that consists of five different analysis tools:

$$
\begin{aligned}
&\text{D1} \quad \Rightarrow \quad y_1 = c \cdot (z_1^2 + x_1 + z_2 - 0.2 \cdot y_2) \\
&\text{D2} \quad \Rightarrow \quad y_2 = c \cdot (\sqrt{y_1} + z_1 + z_2) \\
&\text{F1} \quad \Rightarrow \quad f = x_1^2 + z_2 + y_1 + e^{-y_2} \\
&\text{G1} \quad \Rightarrow \quad g_1 = \frac{y_1}{3.16} - 1 \\
&\text{G2} \quad \Rightarrow \quad g_2 = 1 - \frac{y_2}{24.0}
\end{aligned}
$$

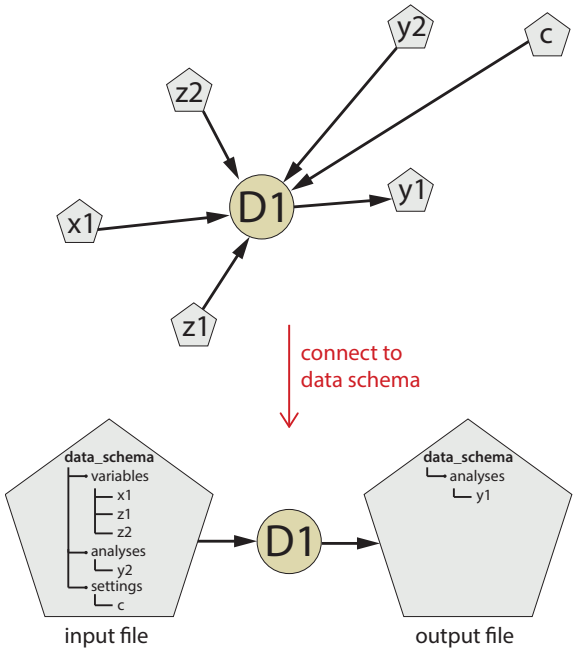
To illustrate the different KADMOS graphs three additional tools are added in the knowledge base:

$$\begin{aligned} \text{A} &\Rightarrow c = \text{f}(a) \\ \text{D3} &\Rightarrow x_1 = \text{f}(y_1) \\ &\Rightarrow z_1 = \text{f}(y_2) \\ &\Rightarrow z_2 = \text{f}(y_1, y_2) \\ \text{F2} &\Rightarrow f = \text{f}(x_1, z_1, z_2) \end{aligned}$$

The directed graph that connects the tools and variables is shown in Figure 10b. Following the central data schema approach described in Section II.B.1 a small XML schema is defined for the case study. The full schema is depicted in Figure 9a. In Figure 9b, one can see which I/O files need to be defined in the knowledge base to connect the D1 tool to the central schema. Similar files are created for the other tools.



(a) Data schema for the Sellar problem

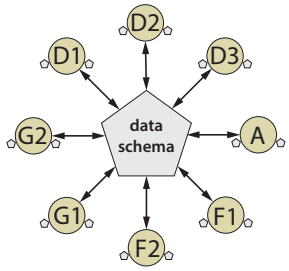


(b) Connecting tool D1 to the central data schema

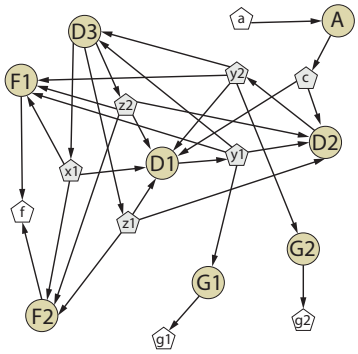
Figure 9: Central data schema for the Sellar problem and illustration of the required I/O files to connect the D1 tool to the central data schema in (a)

An RCG can now be created for the eight tools in the knowledge base. If all the variable nodes are contracted, then the resulting RCG is as depicted in Figure 10a. The full connection details can be inspected using a data flow XDSM, as is shown in Figure 10c. In the XDSM one can see the complete repository. Coupling variables are shown at off-diagonal positions between tools, and the top row and most-left column show respectively the MDAO model input and output variables (in white parallelograms).

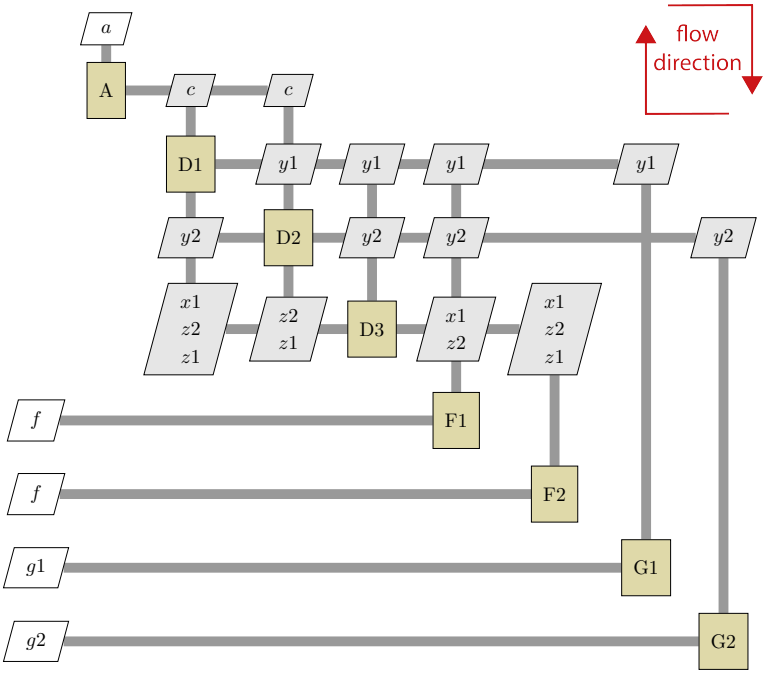
The FPG can now be determined by applying one of KADMOS' graph manipulations. In this case two tools need to be removed: D3 and F2. The algorithm automatically removes the tools including its connections to the data schema. If any variable nodes are left unconnected, these are also removed from the graph. Removing D3 and F2 leads to the FPG and associated data flow XDSM depicted in Figure 11. As can be seen in the data flow XDSM, multiple model inputs (x_1, z_1, z_2) have been created by removing the tools, three coupling variables are left (c, y_1, y_2), and the three model outputs remain (f, g_1, g_2). Before the solution strategy can be imposed on the FPG, variables with a specific role (e.g. design variables, objective value, constraint values) need to be indicated, as was described in Section III.B.3. At this point the MDAO problem description has to be formalized by the design team. In case of the Sellar problem the optimization



(a) Top-level RCG



(b) RCG



(c) Data flow XDSM

Figure 10: Three different views of the same RCG for the Sellar case: (a) top-level with contracted data schema, (b) full graph, and (c) data flow XDSM

problem to be solved can be defined as:

$$\begin{aligned} \text{minimize } & f \\ \text{with respect to: } & z_1, z_2 \\ \text{subject to: } & g_1 \leq 0 \\ & g_2 \leq 0 \end{aligned}$$

Note that, contrary to the original Sellar problem, x_1 has been kept as a constant to have the effect of including a disciplinary parameter in the solution strategy. Thus the following 'problem role' attributes have to be set in KADMOS for the MDAO problem stated above, see also Figure 11b:

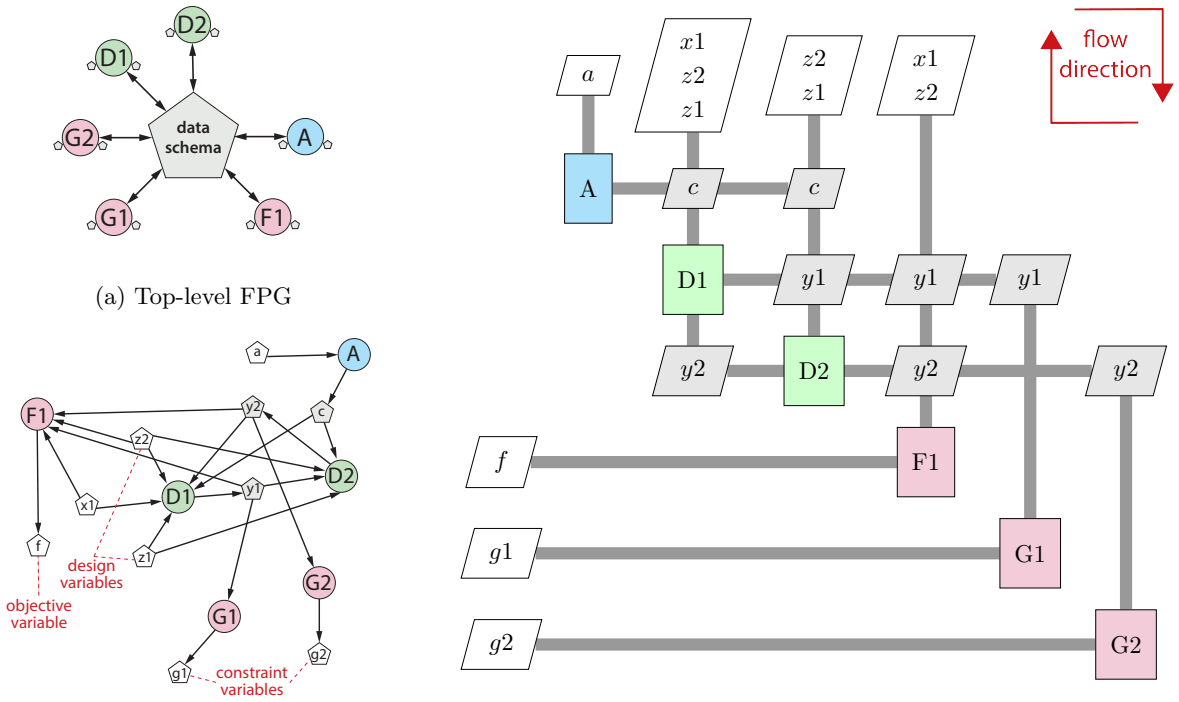
- design variables: z_1, z_2
- objective variable: f
- constraint variables: g_1, g_2

In addition to the role attribute, the variable's upper and lower bounds can also be defined for the design variables and constraint values in KADMOS. If these are not defined, they can also still be set in the SWF software package before running the workflow.

With the complete FPG in place a solution strategy of choice can be imposed on it. Of the strategies listed in Section C, the MDF strategy with a Gauss-Seidel coupling decomposition approach will be explained in more detail to give an idea on how the KADMOS graph algorithms operate.

The two graphs required to define an MDAO model are shown in Figures 12a and 12b: the MDG and the MPG. The superposition of these two graphs using the XDSM notation is depicted in Figure 12c. The data flow in the XDSM (indicated by thick grey lines) is equal to the information stored in the MDG, while the process flow in the XDSM (indicated by thin black lines and numbering in the blocks) is stored in the MPG. So how does KADMOS get from the FPG to these two graphs?

In case of an MDF with Gauss-Seidel decomposition, the algorithm for MDG determination operates as follows:



(a) Top-level FPG

(b) FPG

(c) Data flow XDSM

Figure 11: Three different views of the same FPG for the Sellar case: (a) top-level with contracted data schema, (b) full graph including problem role attributes, and (c) data flow XDSM

MDG algorithm for MDF with Gauss-Seidel decomposition (see Figure 12 for final XDSM)

1. Automatically categorize FPG function nodes as shown in Figure 8.
2. Check FPG on validity of nodes, edges, and problem formulation.
3. Copy the FPG as a starting point for the MDG.
4. Add MDAO architectural nodes: coordinator (COOR), optimizer (OPT), and converger (CONV).
5. Find the variable nodes with a problem role attribute.
6. Connect the converger block (CONV) to the coupled functions (green blocks in XDSM).
   a: Remove feedback coupling between coupled functions.
   b: Connect feedback coupling to the converger (see '6: $y2$' in the XDSM).
   c: Add and connect a copy variable for each feedback coupling (see '4: $y2^c$' in the XDSM).
   d: Add an initial guess of the copy variable and connect it to the COOR and CONV blocks (see '3: $y2^{c0}$' in the XDSM).
   e: Add final coupling value variables between the coupled functions and the coordinator (see '9: $y1^*$' and '9: $y2^*$' in the XDSM).
7. Connect optimizer (OPT) to the rest of the workflow:
   a: Connect design variables as output of the Optimizer (see '4: $z2$ $z1$' in the XDSM).
   b: Add initial guesses of the design variables and connect them to the coordinator (COOR) and optimizer (OPT) blocks (see '2: $z1^0$ $z2^0$' in the XDSM).
   c: Connect the objective and constraint variables from the post-coupling functions F1, G1, and G2 as input to the optimizer (see '8: f' in the XDSM).
   d: Add final objective and constraint value variables between the post-coupling functions and the coordinator (see '9: f^*' and '9: $g1^*$' in the XDSM).
8. Connect any remaining input and output nodes to the coordinator (see '4: $x1$' in the XDSM).

In case of the MDF architecture, the MPG is determined as results of the following manipulations:

MPG algorithm for MDF with Gauss-Seidel decomposition (see Figure 12 for final XDSM)

1. Automatically categorize FPG function nodes as shown in Figure 8.
2. Check FPG on validity of nodes, edges, and problem formulation.
3. Start with an empty directed graph object of the `MdaoProcessGraph` class.
3. Add the function nodes from the FPG and the architectural nodes (COOR, OPT, CONV).
4. Group pre-coupling functions in those that come before one of the design variables (pre-design-functions) and those that come after one of the design variables (post-design-functions).
5. Set process step to 0 and assign step number 0 to COOR block (see top-left corner).
6. Add a sequential process from the COOR block to the OPT block via the pre-design-functions (0: COOR → 1: A → 2: OPT).
7. Add a sequential process from the OPT block to the CONV block via the post-design-functions (2: OPT → 3: CONV).
8. Add an iterative sequential process from the CONV block through the coupled functions back to the CONV block (3: CONV → 4: D1 → 5: D2 → 6: CONV).
9. Add a parallel process from the CONV block to the OPT block via the post-coupling functions (6: CONV → (7: F1, 7: G1, 7: G2) → 8: OPT)
10. Add a process step from the OPT block back to the COOR block (8: OPT → 9: COOR).

Similar algorithms have been written for the other MDAO architectures, as listed in Section III.C. In KADMOS, the different algorithmic steps have been modularized into methods such as `add_optimizer()` and `add_sequential_process()` so that each MDAO architecture can reuse them. The resulting graphs using the MDF strategy are shown in Figure 12. As is already hinted by the fact that the superposition of the MDG and MPG graphs contain the information stored in the XDSM, KADMOS provides a full and detailed specification of the MDAO model to be executed with these two graphs.

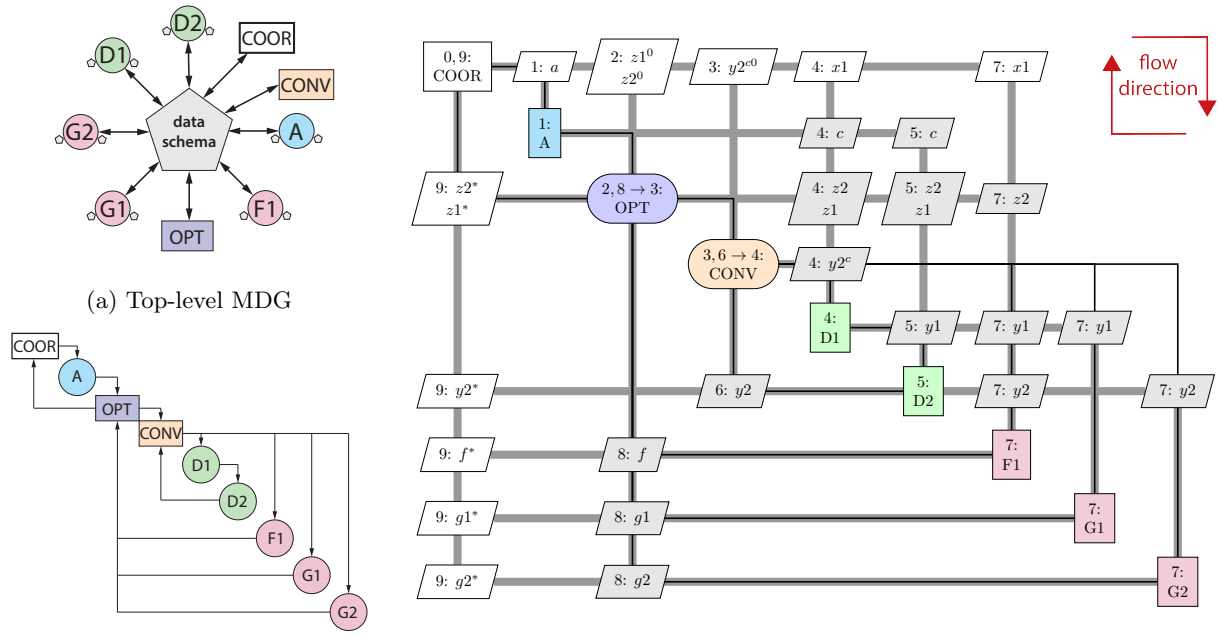
Note also that the graph-based methods increase agility on the formal specification of the MDAO problem and model. If the design team would need to change anything in the tool definitions (e.g. extra inputs) or the MDAO problem formulation (e.g. x_1 becomes a design variable instead of a parameter), then these things can be changed by adjusting just a couple of coded lines in the Python script using the KADMOS package or changing some of the files in the knowledge base. After such adjustments all changes will be automatically processed in all other steps of the MDAO-based development process by rerunning the Python script.

After the formal specification provided by the MDG and MPG the next step in the development process will have to become SWF specific. In this case the capability of KADMOS to write an executable workflow for RCE is shown. KADMOS' method `get_rce_graph()` can simply be used to translate the combination of MDG and MPG into an RCE-specific graph. Subsequently, this graph representation is exported to a JSON-based RCE workflow file. These two steps are shown in Figure 13, but are not explained in further detail here. Thanks to this export function of KADMOS, the MDAO symphony that has been composed can now also be played!

All solution strategies have been tested rigorously and proven to provide the same optimization results as presented in other papers using the Sellar problem.[14,31] However, it is not so much the results (or even the performance) of the optimization process that is of interest here, but rather the successful implementation of a system that can support the formal specification of an MDAO model from disciplinary tools in a knowledge base to executable workflows in a SWF software package. Thanks to the formal specification provided by KADMOS the set-up of distributed executable workflows, such as the one shown on the right side of Figure 13, becomes a matter of minutes instead of hours. The system implementation will make the design team more agile in the set-up and execution of MDAO problems, as is shown in the second case study in the next section.

2. Wing design

The wing design study is a case study that considers the use of CPACS-compatible aircraft design tools for KADMOS. In this study several in-house wing design tools from the Delft University of Technology

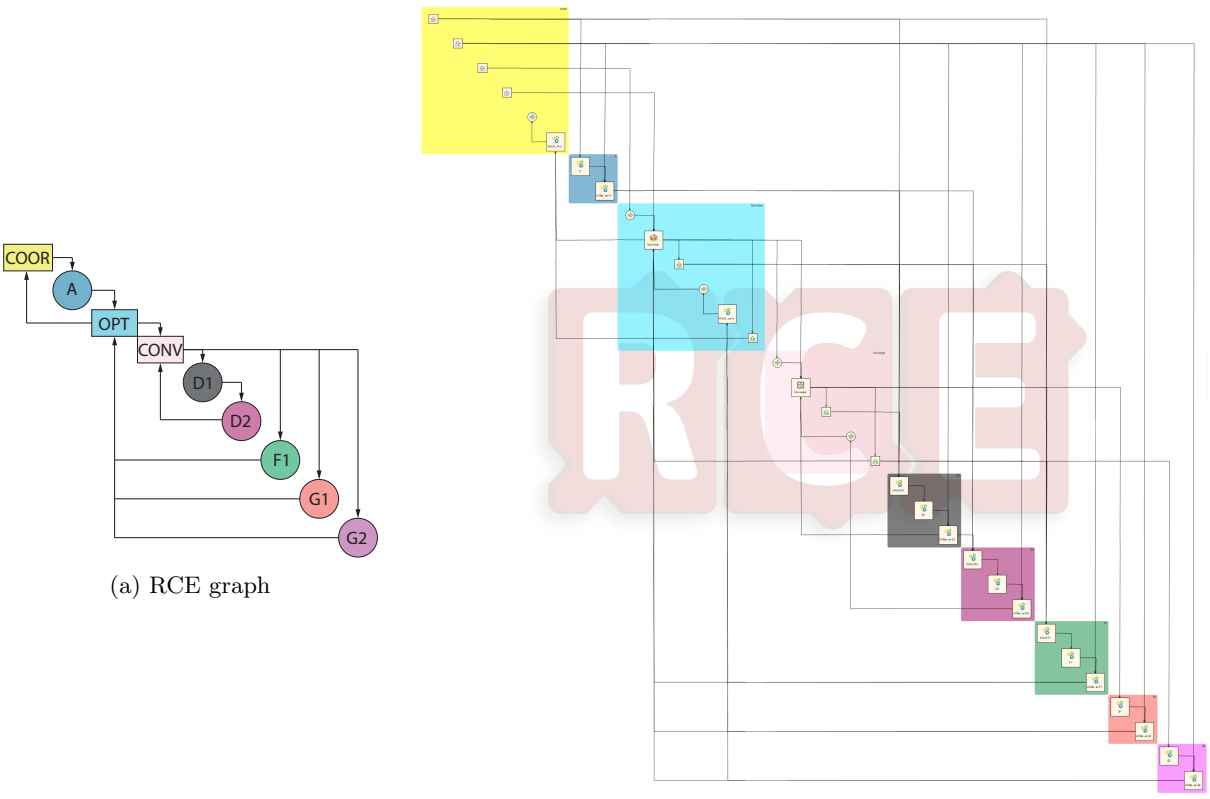


(a) Top-level MDG

(b) Top-level MPG

(c) XDSM

Figure 12: Formal specification of the MDF with Gauss-Seidel convergence architecture for the Sellar case: (a) and (b) MDG and MPG with contracted data schema, and (c) superimposed MDG and MPG in XDSM



(a) RCE graph

(b) Screenshot of the executable RCE workflow

Figure 13: Two different instances of the RCE-specific implementation of the formal specification depicted in Figure 12: (a) RCE graph and (b) RCE workflow file for the Sellar case

are used to assess the use of the CPACS as a central data schema. In-house design tools are used as they are more convenient to quickly test the system, instead of being dependent on the execution of third-party tools. However, a more heterogeneous tool repository would not change anything on the KADMOS side of the process, as it would only impact the additional elements required for collaborative workflows which are created after the formal specification provided by KADMOS, see also Section II.B.4. The results of this case study are summarized in Figure 14. There are 29 tools in the repository, such as those to initialize a CPACS aircraft geometry, to morph the wing based on design variables, and to perform disciplinary analyses such as quasi-3D aerodynamic wing analysis with Q3D[32] and wing mass estimation using EMWET.[33] The MDAO problem concerns the optimization of a wing by adjusting its airfoil shapes at three sections (root, kink, tip) and its planform (root chord length, taper, dihedral, and twist) of an existing aircraft to minimize the maximum take-off weight for a single design mission.

The wing design case provides a proof-of-concept concerning the scalability of KADMOS, as well as the handling of a complex data schema like CPACS. The results of the study are summarized in Figure 14, where the full RCG (>28000 nodes) is shown in the top. Using the RCG an FPG is constructed that can be used to impose many different architectures. For different architectures, one would only have to change the assigned problem role attributes in the FPG (e.g. a tool chain test only uses state variables, while the MDF requires design variables, contraints, and an objective). KADMOS then automatically determines the MDG and MPG for each architecture, as visualized by the XDSMs in the figure. Finally, the accompanying RCE workflows can also be created and executed (not shown in the figure). These workflows, which involve hundreds of connections between the different execution blocks, are a tedious and time-consuming task to create manually. In addition, the manual creation of these complex workflows is prone to human error, but using KADMOS' formal specification they can now be created fully automatically within a time span which is an order of magnitude lower (minutes instead of hours). In conclusion, the wing design case has proven that KADMOS is the MDAO symphony composition tool that can make design teams more agile than before.

V. Conclusions and future work

The current status of a graph-based MDAO system called KADMOS has been presented. The system is under development within the EU project AGILE. Analysis of the current system led to the following conclusions:

- KADMOS is able to fully support the formulation phase of the MDAO-based development process shown in Figure 1 from tool definition to executable distributed workflows for a classical MDAO problem.
- KADMOS can automate a lot of steps that currently have to be performed manually when defining the MDAO model, such as checking of inputs, outputs, and tool couplings.
- KADMOS can provide a formal specification of the MDAO model using two graphs: the MDAO data graph and the MDAO process graph. The superposition of these graphs would result in an XDSM.
- The formal specification of the MDAO model is currently a cumbersome task and should be supported by software tools like KADMOS as MDAO problems grow in size and complexity.
- The formal specification of the MDAO model enables the automatic creation of executable, collaborative workflows, thereby bringing the required time to set-up such workflows down an order of magnitude (from hours to minutes).
- KADMOS has successfully been tested for scalability and CPACS compatibility using a wing design optimization problem of industrial relevance.

In conclusion, KADMOS has shown the possibility for design teams to become more agile when composing MDAO symphonies, as well as the ability to actually play the symphony with the orchestra. Future developments will aim at extending the ability to compose and play larger and more complex pieces. These developments will focus on three different aspects of the system:

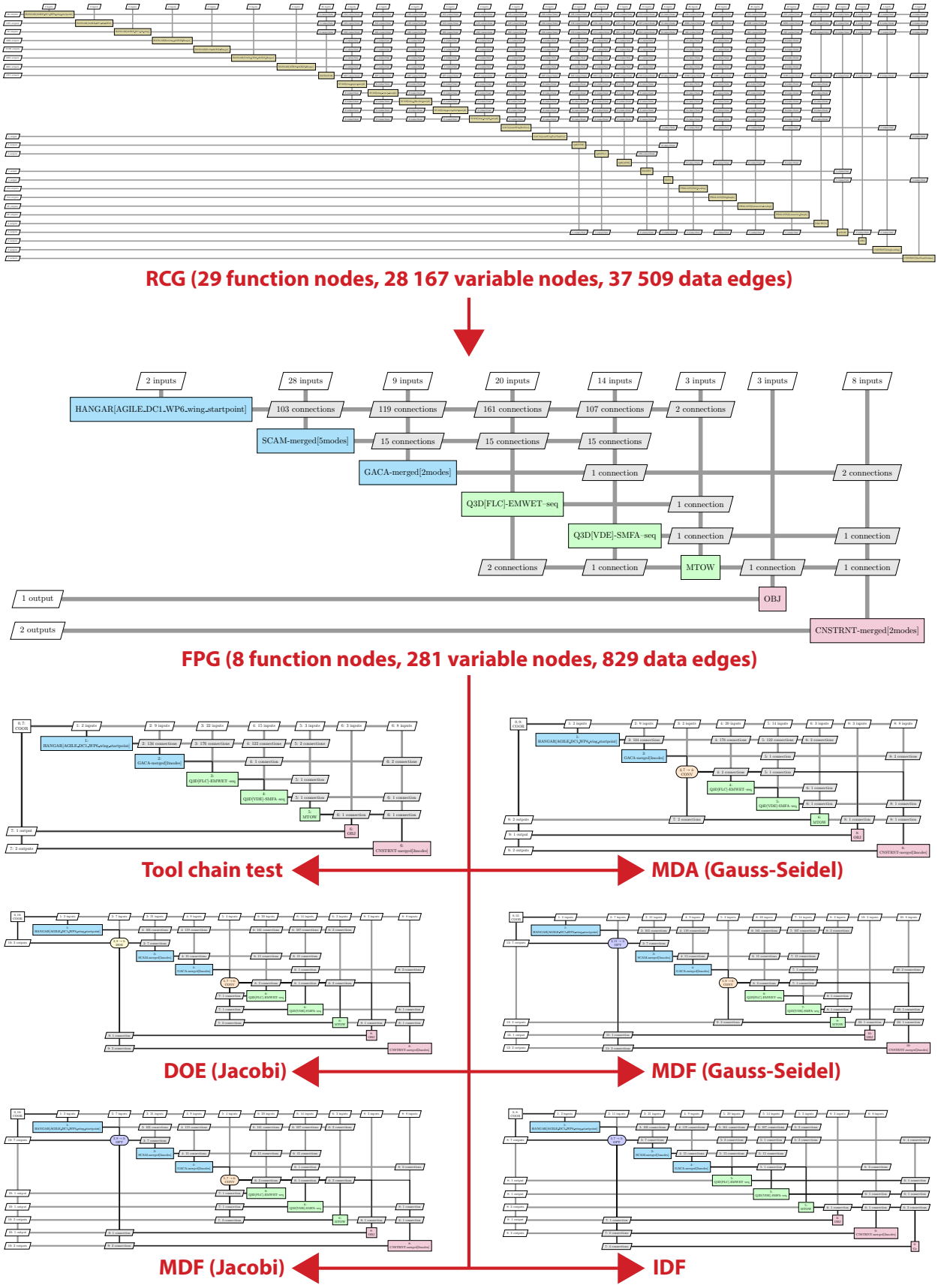


Figure 14: Summary of the wing design case: for a large repository of design tools (top) a relatively small FPG is crafted to solve a specific MDAO problem (middle) on which different MDAO architectures can be imposed automatically (bottom)

- Extension of the current graph manipulation algorithms to include more complex MDAO architectures (e.g. BLISS, CSSO, CO, etc.) and to enable the definition of user-defined architectures using the modularized methods.
- Definition of an import and export file standard to support exchangeability of the KADMOS graphs with other MDAO developments, e.g. the exported file could be opened by platforms like RCE and Optimus to assemble the executable workflow inside the SWF platform instead of having KADMOS provide the SWF-specific files. The work on this standard has recently been initiated in AGILE and it is called CMDOWS (Common MDO Workflow Schema)[34,e].
- Testing of KADMOS in future MDAO design studies that will be performed in the final year of AGILE (June 2017-2018).

The goal is to make KADMOS a publicly available MDAO support system, so that other MDAO projects can also benefit from its capabilities in the future.

Acknowledgments

The research presented in this paper has been performed in the framework of the AGILE project (Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts) 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 contribution and feedback.

References

[1]Bowcutt, K. G., "A perspective on the future of aerospace vehicle design," *12th AIAA International Space Planes and Hypersonic Systems and Technologies, Norfolk, VA, AIAA Paper*, No. 2003-6957, 2003.

[2]Vandenbrande, J. H., Grandine, T. A., and Hogan, T., "The search for the perfect body: Shape control for multidisciplinary design optimization," *44th AIAA Aerospace Science Meeting and Exhibit, Reno, NV*, No. 2006-928, 2006.

[3]La Rocca, G., *Knowledge based engineering techniques to support aircraft design and optimization*, Delft University of Technology, 2011.

[4]Agte, J., De Weck, O., Sobieszczanski-Sobieski, J., Arendsen, P., Morris, A., and Spieck, M., "MDO: assessment and direction for advancement - an opinion of one international group," *Structural and Multidisciplinary Optimization*, Vol. 40, No. 1-6, 2010, pp. 17–33.

[5]Belie, R., "Non-technical barriers to multidisciplinary optimisation in the aerospace industry," *9th AIAA/ISSMO Symposium of Multidisciplinary Analysis and Optimisation*, 2002, pp. 4–6.

[6]Shahpar, S., "Challenges to overcome for routine usage of automatic optimisation in the propulsion industry," *Aeronautical Journal*, Vol. 115, No. 1172, 2011, pp. 615.

[7]Simpson, T. W. and Martins, J. R. R. A., "Multidisciplinary design optimization for complex engineered systems: report from a national science foundation workshop," *Journal of Mechanical Design*, Vol. 133, No. 10, 2011, pp. 101002.

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

[9]Ciampa, P. D. and Nagel, B., "Towards the 3rd generation MDO collaboration environment," *30th Congress of the International Council of the Aeronautical Sciences*, 2016.

[10]Martins, J. R. R. A. and Lambe, A. B., "Multidisciplinary Design Optimization: A Survey of Architectures," *AIAA Journal*, Vol. 51, No. 9, 2013, pp. 2049–2075.

[11]Gray, J., Moore, K. T., Hearn, T. A., and Naylor, B. A., "A standard platform for testing and comparison of MDAO architectures," *8th AIAA multidisciplinary design optimization specialist conference (MDO), Honolulu*, 2012, pp. 1–26.

[12]Flager, F. and Haymaker, J., "A comparison of multidisciplinary design, analysis and optimization processes in the building construction and aerospace industries," *24th international conference on information technology in construction*, 2007, pp. 625–630.

[13]van Gent, I., Aigner, B., La Rocca, G., Stumpf, E., and Veldhuis, L. L. M., "Using graph-based algorithms and data-driven documents for formulation and visualization of large MDO systems," *6th CEAS Air and Space Conference*, 2017.

[14]Sellar, R. S., Batill, S. M., and Renaud, J. E., "Response surface based, concurrent subspace optimization for multidisciplinary system design," *AIAA paper*, Vol. 714, 1996, pp. 1996.

[15]Nagel, B., Böhnke, D., Gollnick, V., Schmollgruber, P., Rizzi, A., La Rocca, G., and Alonso, J. J., "Communication in aircraft design: Can we establish a common language?" *28th International Congress Of The Aeronautical Sciences, Brisbane*, 2012.

[16]DLR website, "CPACS – A Common Language for Aircraft Design," May 2016.

[e]CMDOWS is publicly available at: `http://cmdows-repo.agile-project.eu`

[17]Böhnke, D., Nagel, B., and Gollnick, V., "An approach to multi-fidelity in conceptual aircraft design in distributed design environments," *Aerospace Conference, 2011 IEEE*, IEEE, 2011, pp. 1–10.

[18]Milton, N. R., *Knowledge technologies*, Vol. 3, Polimetrica S.a.s., 2008.

[19]Makus, A., *Development of a knowledge-enabled tool repository to support automated generation of multidisciplinary design optimization workflows*, Master's thesis, Delft University of Technology, 2017.

[20]Diestel, R., "Graph theory," *Graduate Texts in Mathematics*, Vol. 173, 2010.

[21]Gray, J., Hearn, T. A., Moore, K. T., Hwang, J. T., Martins, J. R. R. A., and Ning, A., "Automatic Evaluation of Multidisciplinary Derivatives Using a Graph-Based Problem Formulation in OpenMDAO," *Proceedings of the 15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Atlanta, GA*, 2014.

[22]Lu, Z. and Martins, J. R. R. A., "Graph partitioning-based coordination methods for large-scale multidisciplinary design optimization problems," *ISSMO multidisciplinary analysis optimization conference, Indianapolis*, 2012, pp. 1–13.

[23]Alexandrov, N. M. and Lewis, R. M., "Reconfigurability in MDO problem synthesis, part 1," *Proceedings of the 10th AIAA/ISSMO multidisciplinary analysis and optimization conference, AIAA paper*, Vol. 4307, 2004.

[24]Alexandrov, N. M. and Lewis, R. M., "Reconfigurability in MDO problem synthesis, part 2," *Proceedings of the 10th AIAA/ISSMO multidisciplinary analysis and optimization conference, AIAA paper*, Vol. 4307, 2004.

[25]Balachandran, L. and Guenov, M., "Object-Oriented Framework for Computational Workflows in Air-Vehicle Conceptual Design," *9th AIAA Aviation Technology, Integration, and Operations Conference (ATIO) and Aircraft Noise and Emissions Reduction Symposium (ANERS)*, 2009, p. 7117.

[26]Hoogreef, M. F. M., *Advise, Formalize and Integrate MDO Architectures - A Methodology and Implementation*, Ph.D. thesis, Delft University of Technology, 2017 (to be published).

[27]Seider, D., Fischer, P. M., Litz, M., Schreiber, A., and Gerndt, A., "Open source software framework for applications in aeronautics and space," *Aerospace Conference*, IEEE, 2012, pp. 1–11.

[28]Baalbergen, E., Kos, J., and Lammen, W., "Collaborative multi-partner modelling & simulation processes to improve aeronautical product design," *4th CEAS Air & Space Conference*, 2013.

[29]Hagberg, A., Schult, D., and Swart, P., *NetworkX Reference - Release 1.11*, January 2016.

[30]Lambe, A. B. and Martins, J. R. R. A., "Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes," *Structural and Multidisciplinary Optimization*, Vol. 46, No. 2, 2012, pp. 273–284.

[31]Perez, R. E., Liu, H. H. T., and Behdinan, K., "Evaluation of multidisciplinary optimization approaches for aircraft conceptual design," *AIAA/ISSMO multidisciplinary analysis and optimization conference, Albany, NY*, 2004.

[32]Mariens, J., Elham, A., and van Tooren, M. J. L., "Quasi-Three-Dimensional Aerodynamic Solver for Multidisciplinary Design Optimization of Lifting Surfaces," *Journal of Aircraft*, Vol. 51, No. 2, 2014, pp. 547–558.

[33]Elham, A. and van Tooren, M. J. L., "Beyond Quasi-Analytical Methods for Preliminary Structural Sizing and Weight Estimation of Lifting Surfaces," *56th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2015, p. 0399.

[34]van Gent, I., Hoogreef, M. F. M., and La Rocca, G., "CMDOWS: A Proposed New Standard to Formalize and Exchange MDO Systems," *6th CEAS Air and Space Conference*, 2017.