Knowledge architecture supporting the next generation of MDO in the AGILE paradigm

van Gent, Imco; Aigner, Benedikt; Beijer, Bastiaan; Jepsen, Jonas; La Rocca, Gianfranco

DOI
10.1016/j.paerosci.2020.100642

Publication date
2020

Document Version
Final published version

Published in
Progress in Aerospace Sciences

Citation (APA)
van Gent, I., Aigner, B., Beijer, B., Jepsen, J., & La Rocca, G. (2020). Knowledge architecture supporting the next generation of MDO in the AGILE paradigm. *Progress in Aerospace Sciences*, *119*, Article 100642. https://doi.org/10.1016/j.paerosci.2020.100642

Progress in Aerospace Sciences 119 (2020) 100642

Contents lists available at ScienceDirect

Progress in Aerospace Sciences

journal homepage: http://www.elsevier.com/locate/paerosci

Knowledge architecture supporting the next generation of MDO in the AGILE paradigm

Imco van Gent [a,1], Benedikt Aigner [b,2], Bastiaan Beijer [c,3], Jonas Jepsen [d,4], Gianfranco La Rocca [a,*,5]

[a] Delft University of Technology - Faculty of Aerospace Engineering, Kluyverweg 1, 2629 HS, Delft, the Netherlands
[b] Institute of Aerospace Systems, RWTH Aachen University, Wuellnerstrasse 7, 52062, Aachen, Germany
[c] KE-works, Molengraaffsingel 12, 2629 JD, Delft, the Netherlands
[d] Integrated Aircraft Design Dpt., German Aerospace Center (DLR), ZAL TechCenter, Hein-Saß-Weg 22, 21129, Hamburg, Germany

ARTICLE INFO

Keywords:
MDO
Knowledge architecture
AGILE
Framework
Standardization

ABSTRACT

After almost three decades of evolution, it is not yet possible to apply MDO in collaborative projects within large, heterogeneous and distributed teams of experts, whilst nowadays necessary for the development of any complex product. The H2020 project AGILE took the challenge of devising a novel paradigm to swiftly set up and deploy large distributed MDO systems, that are easy to (re)configure and monitor during the whole process, from requirements definition to data post-processing. The main outcome is an advanced set of tools and methods contributing to a 3rd generation MDO environment, specifically tailored to the aerospace industry. The AGILE paradigm is built on top of two main pillars, the so-called *knowledge architecture* and the *collaborative architecture*. The former, which is the main focus of this paper, provides a structured approach and the related workbench to formulate and inspect any automated design process, including fully formalized MDO systems. The latter includes the tools and methods to translate these formulations into executable workflows and deploy them across distributed networks. Although AGILE aims specifically at aircraft MDO, the proposed knowledge architecture provides a general conceptual framework that is suitable for the development of any complex product. The knowledge architecture has a multi-level hierarchical structure, consisting of four layers: development process, automated design, design competences and data schemas. Interfaces between the various layers are defined to achieve a fully.interconnected development process. This paper provides first a description of the knowledge architecture as a generalized paradigm to formulate collaborative and distributed MDO systems. Then, the specific implementation of such a paradigm within the AGILE project is illustrated: four knowledge architecture applications and two data schemas are described in detail. Finally, the whole approach is demonstrated by means of a realistic aircraft design case. This implementation proved successful in multiple aspects. First of all, in allowing heterogeneous teams of experts to generate complete and correct MDO system formulations involving large amount of distributed disciplinary tools, while maintaining full control and systematic overview of the complete system archi- tecture. Second, in offering the necessary agility to adjust and reconfigure the formulated MDO systems, such to support the iterative and evolutionary nature of their development process. Finally, by dramatically accelerating the setup time of the MDO system, thanks to the automation of the complex, lengthy and repetitive operations involved in the partitioning and coordination process, and to the effective support in inspecting and resolving the eventual inconsistencies in the data flow, arising every time tools are added or modified, or different solution strategies are implemented.

* Corresponding author.
E-mail address: g.larocca@tudelft.nl (G. La Rocca).
[1] Ph.D. Student.
[2] Ph.D. Student.
[3] Knowledge Engineer.
[4] Researcher Engineer.
[5] Associate Professor.

https://doi.org/10.1016/j.paerosci.2020.100642
Received 2 June 2020; Accepted 13 June 2020
Available online 24 September 2020

Nomenclature

ADF	AGILE Development Framework
AGILE	Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts
API	Application Programming Interface
AR	Aspect Ratio
ATR	Average Temperature Response
BLISS	Bi-Level Integrated System Synthesis
CA	Collaborative Architecture
CIAM	Central Institute of Aviation Motors (Russia)
CMDOWS	Common MDO Workflow Schema
CO	Collaborative Optimization
CPACS	Common Parametric Aircraft Configuration Schema DLR German Aerospace Center
DOC	Direct Operating Cost
DOE	Design Of Experiments
DUT	Delft University of Technology
IDEaliSM	Integrated & Distributed Engineering Services framework for MDO
IDF	Individual Discipline Feasible
ISA	International Standard Atmosphere
KA	Knowledge Architecture
KADMOS	Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System
MDAO	Multidisciplinary Design Analysis and Optimization
MDO	Multidisciplinary Design Optimization
MDF	MultiDisciplinary Feasible
MLM	Maximum Landing Mass
MTOM	Maximum Take-Off Mass
PAX	Passengers
PIDO	Process Integration and Design Optimization
PoliTo	Politecnico di Torino
RCE	Remote Component Environment
RWTH	Rheinisch-Westfälische Technische Hochschule
S	Wing surface area
SL:	Sea Level
TOFL:	Take-Off Field Length
TsAGI	Central Aerohydrodynamic Institute (Russia)
UID	Unique IDentifier
VISTOMS	VISualization TOol for MDO Systems
XDSM	eXtended Design Structure Matrix
XML:	eXtensible Markup Language
Λ	Wing sweep

1. MDO challenges and the AGILE paradigm

Multidisciplinary Design Optimization (MDO) has been a promising design methodology for decades. Despite its promise, MDO is not as widely applied as was expected at its birth more than three decades ago [1–3]. Both technical and non-technical challenges have hampered its successful exploitation, and eventually reduced its scope to cases involving either a limited amount of disciplines or the application of low-fidelity analysis tools. At date, success stories of MDO application to full aircraft design in a true industrial setting are not available. Still the community acknowledges the large benefits such design methodology can potentially deliver, especially when applied to novel aircraft configurations, for which design drivers and multidisciplinary synergies are still unknown or unexploited. Through a systematic effort of identification and resolution (or at least mitigation) of the aforementioned challenges, the MDO (or MDAO) community has succeeded in evolving its arsenal of tools and methods throughout two generations of MDO environments [4,5].

First generation environments included software applications and strategies limited to the setup of monolithic computational systems operated by a single user. Second generation environments have improved workbenches, enabling the distribution of the analysis capabilities on dedicated computational facilities, under the coordination of a centralized design and optimization process. A third generation MDO environment is currently being shaped within AGILE (Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts) [6], an EU research project under the funding scheme Horizon 2020, of which this paper presents a key contribution. AGILE's goal is to extend the current set of applications, strategies and data schemas such to enable distributing all the tasks involved in the formulation and operation of an MDO system, thereby enabling a truly collaborative environment for discipline experts, system architects and final users.

In this paper, by *MDO environments* we intend the ecosystems in which MDO systems are assembled using *MDO frameworks* as workbenches. Hence, the framework presented in this paper is one of the implemented workbenches to be used within the broader 3rd generation environment. By *MDO system* we intend a representation of the set of design competences (tools), exchanged data and process relations necessary to perform multidisciplinary design, analysis and/or optimization of a given product. In this sense, the term MDAO (Multidisciplinary Design Analysis and Optimization) might be more suitable to express the fact that some of the MDO systems addressed in this paper do not necessarily include optimization, and could be limited to sole multidisciplinary analysis. Yet, the term MDO will be used here to maintain consistency with existing literature.

According to the AGILE paradigm, an MDO system evolves throughout five stages, as further elaborated later in this paper, involving the following three main phases in the utilization of the MDO environment (see Fig. 1):

Each phase presents specific challenges and, accordingly, puts different demands on the MDO environment that is used for developing the given MDO system. The goal of the setup phase is to gather into a coherent and consistent repository different design competences (e.g. in the form of disciplinary design tools, or pre-assembled workflows of tools), which are often provided by different departments within the same organization, or even by multiple organizations. The first technical challenge here is to "let the design competences speak to each other", hence to enable the necessary input/output data flow. This is typically a challenging task even when using design competences that are available in the same design team. Its complexity grows exponentially when the tools to connect are distributed across large and heterogeneous teams, not geographically collocated. This is not only difficult and time-consuming, but intellectual property protection, tool accessibility and security issues can make any technical solution practically unfeasible.

In the operation phase the MDO environment is used by the design team to define first the MDO problem to be solved, then to determine the right strategy to solve it and, finally, to implement such strategy as an executable workflow. This second phase too presents a mix of technical and non-technical challenges. A first main technical challenge, obviously, concerns the ability to formulate a complex MDO system involving a large number of design competences. The second concerns the integration of the MDO system formulation into an ex- ecutable computational process (e.g. using a PIDO tool). A third technical challenge, specifically addressed by AGILE, is the homonym *agility* challenge, that is the ability to reconfigure a previously assembled MDO system, such to support designers exploring new insights acquired after the first computation runs, to include new or modified design

requirements, or to change or add some design competences to the already established automated design process.

A fundamental non-technical challenge originates from the loss of top-level overviews the various MDO system stakeholders may suffer, as a consequence of the high workflow complexity. This can make it hard to find possible in- consistencies in the automated design process and hampers the identification of design trends and decision making. On top of that, the fact that disciplinary tools and other design competences involved in a distributed MDO system are used outside the direct control of the disciplinary experts, can undermine the trust on the reliability of the obtained results and, eventually, on the benefit of the MDO approach, at all.

A survey of recent MDO-oriented projects performed within DLR (German Aerospace Center) as well as in the context of other European research initiatives, highlighted that an astonishing 60–80% of the overall project time is used for assembling the first automated chain of multidisciplinary analyses [7], hence to reach the end of the operation phase in Fig. 1 for the first project iteration, without any reconfiguration. A similar conclusion was drawn by Refs. [8]; who compared the traditional design method used in the aerospace industry with the novel MDO approach and found that, using MDO, the first design iteration took 133% more time. Although the MDO approach was able to produce tenfold more iterations and in a very short time, it was noted that the time increase to achieve the first result was putting a huge burden on the designer and dramatically increasing the risks involved in the project.

The solution phase challenges are mostly of technical nature and concern the capability to reach convergence of the executable workflow and identify robust optima within the allocated time. New optimization algorithms and MDO architectures are continuously developed to address these issues [9–11]. Recently, quite some developments are happening also concerning the computational infrastructure, for which software solutions are being devised to *cloudify* computationally expensive workflows [12]. Intellectual property, software licensing policies and security issues, again, hamper the practical usability of such technical solutions. After the third phase the design project is finished and needs to be stored in a systematic manner to be useful for future reference or if modifications turn out to be necessary.

The excessive time to formulate and integrate an MDO system, the lack of reconfiguration agility during deployment, and the struggle to maintain overview and control have been identified by AGILE as the main limitations of the first two generations of MDO environments. To address these fundamental challenges, AGILE is proposing a new methodological approach, the so-called *AGILE paradigm*, which is built on top of two main cornerstones: the *knowledge architecture* (KA) and the *collaborative architecture* (CA). The KA provides the structured approach and workbench to formulate, (re)configure and inspect any automated design process, including fully specified MDO systems that are ready to be converted into executable computational systems. The CA includes the methods and tools to assemble and deploy executable MDO workflows across distributed networks. More specifically, it provides the means to connect simulation tools in a service-oriented scenario, including solutions for the cross- human (e.g. disciplinary specialists need to stay in control of their own tools) and cross-organizational (e.g. intellectual property restrictions and firewalls) issues occurring in a collaborative distributed process. Fig. 2 schematically illustrates the whole AGILE paradigm and provides a qualitative representation of the targeted time reductions in the three phases of the MDO process: a 40% reduction in time needed to configure a multidisciplinary system by a team of heterogeneous specialists in the setup and operation phases and a further reduction of 20% in time to find an (optimized) design solution.

For a detailed description of the AGILE project, its background, objectives and organization, the reader is referred to Ref. [7]. The CA within the AGILE paradigm is fully elaborated in Ref. [13]. The KA is the main focus of this paper. First a description of the KA, as a non- domain specific methodology to architect collaborative and distributed MDO systems, is provided in Section 2. Then, its specific implementation to support the AGILE aircraft design campaigns is discussed in Section 3. Finally, the whole approach is demonstrated in Section 4, by means of a realistic aircraft design case.

2. Knowledge architecture: conceptual description

Fig. 3 provides a schematic of the Knowledge Architecture (KA), as defined within the overall AGILE paradigm. The KA features a hierarchical four- layer structure. In the top layer *development process* all the tasks required to define, monitor and manage an MDO system are compiled into one business pro- cess. Hence, this development process layer serves as “the cockpit” of the KA from which lower layers are controlled and all other applications are used “under the hood”. The intermediate layer *automated design* provides the means to formalize the computational architecture of the automated design system. The bottom layer *design competences* hosts the actual synthesis and analysis tools contributed by the various discipline experts involved in the collaborative design process. A fourth transverse layer *data schemas* provides the other three with one product and one workflow data schema to support, respectively, the input/output data exchange between the various design competences and the progressive integration of the overall MDO system. Interfaces guarantee cohesion between the layers, as clarified in the forthcoming subsections.

In the spirit of supporting the collaborative work of heterogeneous teams of experts, five different agents have been defined within the AGILE paradigm. The identification of these agents, with their specific needs, competence and responsibility in the various utilization phases of the MDO environment is one of the key aspects of the AGILE paradigm.

Customer: The target user/beneficiary of the MDO system to be developed within the MDO environment. The customer is responsible for specifying the top-level requirements for the product to be designed/analyzed/optimized (including performance indicators to be maximized, constraints to be respected, etc.), as well as key limitations on its development process, for example the expected scope and lead time. The customer is also responsible for providing feedback on the results produced by the developed MDO system and, if required, for revising the initial requirements and scope.

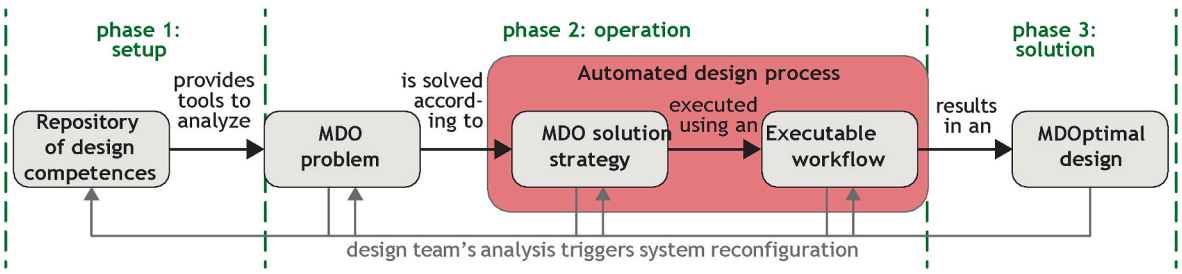


Fig. 1. The five different stages (grey blocks) an MDO system can have within an MDO environment, their relation (black arrows), their position in different project phases (green text and lines) and the required reconfigurations (grey arrows) for a typical MDO project.

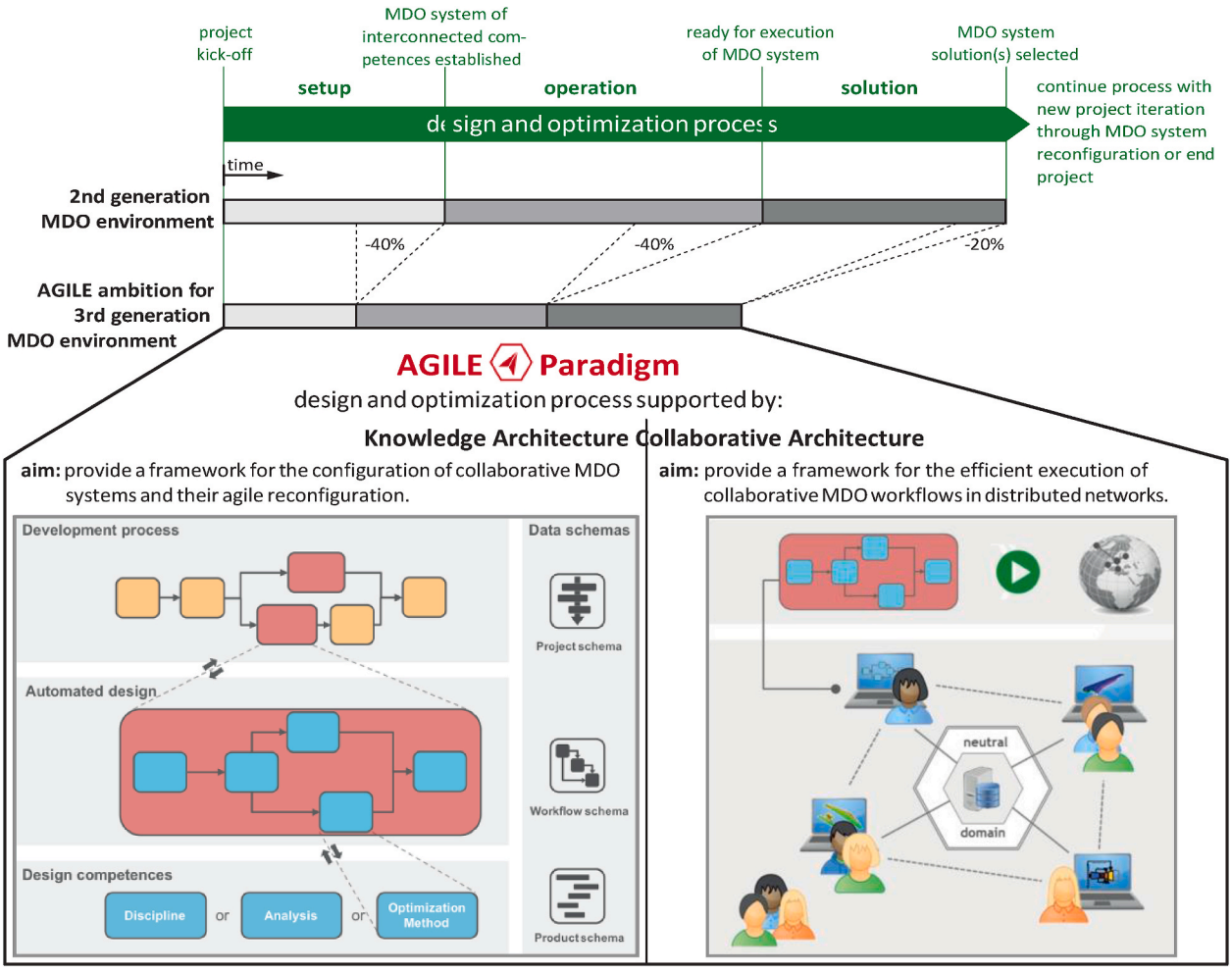


Fig. 2. AGILE Paradigm - Conceptual overview.

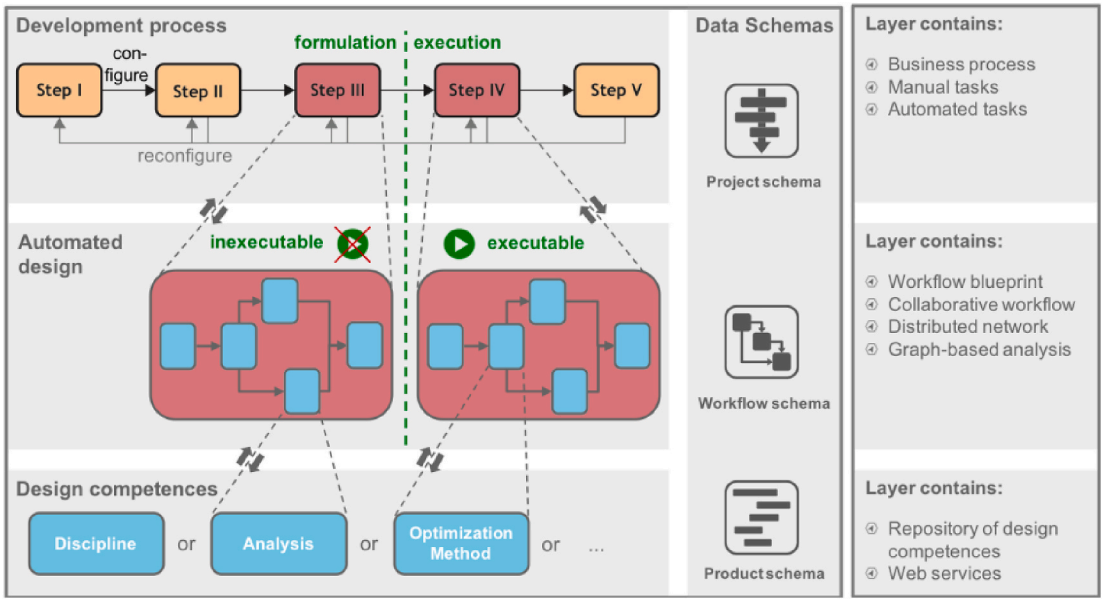


Fig. 3. The AGILE Knowledge Architecture (KA) to support automated design in large, heterogeneous teams of experts.

Architect: This is the agent responsible to define a suitable automated design system architecture to fulfill the customer's needs. Thus the architect is responsible to translate the MDO problem defined by the customer into a fully formalized computational architecture, containing the necessary design competences.

Integrator: This is the agent responsible to convert the MDO system formulation provided by the architect, into an executable computational workflow to be deployed across the distributed computational infrastructure. Thus, this agent is the technical manager responsible for encoding the neutral MDO system formulation into the Process Integration and Design Optimization (PIDO) platform of choice and for testing the obtained executable. The integrator is also responsible for the integration of design competences within the KA (hence for coupling inputs and outputs of the various design competences), which is actually the first step towards the set up of any MDO system.

Competence specialist: Typically multiple such agents are involved, each one responsible for the functionality, availability and usability of one or more of the design competences to be integrated in the MDO system, such as design synthesis tools, disciplinary analysis tools and optimization services.

Collaborative engineer: Responsible throughout the project phases for providing technical support for the integration of design competences. In the setup phase, the collaborative engineer supports the competence specialist in making their design competence compliant to the requirements for integration. Also, this agent provides solutions to make competences accessible and executable in the MDO workflow. This includes the secure integration of design competences from different networks. In addition, the collaborative engineer is the solution provider for the intellectual property protection and data transfer security.

The specific roles of the various agents in the KA layers are discussed in more detail in the forthcoming sections and examples are provided in the demonstrator Section 4.

The identification of the four key concepts mapped to the KA layers, namely, organization, simulation workflow, tool and data, as fundamental aspects of any collaborative design process, is not new in literature [14]. However, the way they are brought together into one comprehensive methodological approach, including the key operating agents, is original of the AGILE and IDEaliSM projects [15]. The latter is a recent, almost concurrent, predecessor of AGILE, involving several common partners.

The four layers of the KA with their relative interfaces are discussed in the next subsections.

2.1. Development process layer

The development process layer can be seen as the cockpit or decision room where the setup and execution of a new multidisciplinary design problem take place. A business type of process is defined here to allow all the aforementioned agents to collaborate and interact during the two main phases of the MDO system development: formulation and execution. This process, named the AGILE development process, is organized in five main steps, which are illustrated in Fig. 4 and listed below:

Step I: **Define design case and requirements.** Information and requirements are collected concerning the product to design/analyze/optimize, the MDO system to be developed and the available design compe- tences (tools and experts) from the design competences layer (Section 2.3).

Step II: **Specify complete and consistent product model and design competences.** The consistency of the collected information is verified and validated. The design competences are linked to the common product model from the data schemas layer (Section 2.4) and the connections are verified.

Step III: **Formulate design optimization problem and solution strategy.** This is the formal link to the automated design layer (Section 2.2), where the automated design process is formalized based on the requirements and design competences identified in the preceding steps.

Step IV: **Implement and verify collaborative workflow.** This is the link between the KA and CA of the AGILE paradigm. The common workflow schema from the data schemas layers (Section 2.4) is used to enable the translation of the formulated (inexecutable) automated design process into an executable workflow for the PIDO platform of choice.

Step V: **Execute collaborative workflow, select design solution (s) and/or go back to an earlier step for reconfiguration.** This is the final phase of the development process, where results are generated and, in case, a reconfiguration of the development process is triggered in light of the obtained insights.

Next to the five steps of the AGILE development process shown in Fig. 4, the specific involvement of the aforementioned five agents and the software applications developed and/or selected in AGILE to support the proposed methodology are also depicted. These so-called KA applications, namely KE-chain.

KADMOS, VISTOMS, cpacsPy, provide the necessary functionality for the execution of the five-step approach. For example, KE-chain is the main application in the development process layer and provides the platform to model and manage the execution of the five steps; KADMOS is a graph-based package used for the formulation of automated design processes (step III); and VISTOMS is a visualization tool (steps II and III). In step IV and V the CA applications Optimus and RCE are PIDO platforms that are used for the integration of executable computational workflows. All these applications are not domain specific, thus endow the AGILE paradigm with the necessary neutrality to allow its application to different engineering areas. The KA applications involved in the first three steps will be further discussed in Section 3, where the implementation of the AGILE paradigm in the aircraft design domain is discussed. For details on the last two steps and the involved CA applications, the reader is referred to Ref. [13].

2.2. Automated design layer

As illustrated in Fig. 3, two instances of the automated design process reside in this layer: the inexecutable formulation of the automated design process defined in step III of the development process and the executable collaborative workflow that is instantiated in step IV.

This automated design process is assembled by invoking multiple design competences from the design competences layer and orchestrating them according to the specific MDO architecture selected in the development process. This can be the architecture of an automated design process for design convergence studies, design space explorations or full fledged MDO. It may regard the overall design synthesis of a complete product (e.g. an aircraft), or the optimization of a specific component (e.g. a wing). The non-executable formulation (Fig. 3, left) represents the blueprint of the automated design process and can be generated by the KA application KADMOS. Such a blueprint contains the formalization of the data and process flow of the automated design process. It is a *neutral* representation of the MDO system, in the sense that it does not contain any specific method for instantiating the actual workflow using some specific PIDO platform. In the MDO community, a widely used visualization standard for this sort of formalization is the eXtended Design Structure Matrix (XDSM), originally proposed by Ref. [16]. As shown later in this paper, the XDSM is regularly used within AGILE, not only to visualize MDO architectures, but any form of automated design process. VISTOMS is the KA application developed in AGILE to provide the necessary visualizations. Based on the automated design process blueprint, the executable counterpart (Fig. 3, right) is instantiated automatically using a PIDO platform (e.g. Optimus, RCE, OpenMDAO, etc.). This automated link between formulation and execution is a key feature of the KA. Without this link the formulation performed in steps I-III would be disconnected from the execution in Step V, meaning that any reconfiguration of the automated design process would require manual adjustments of the executable workflows.

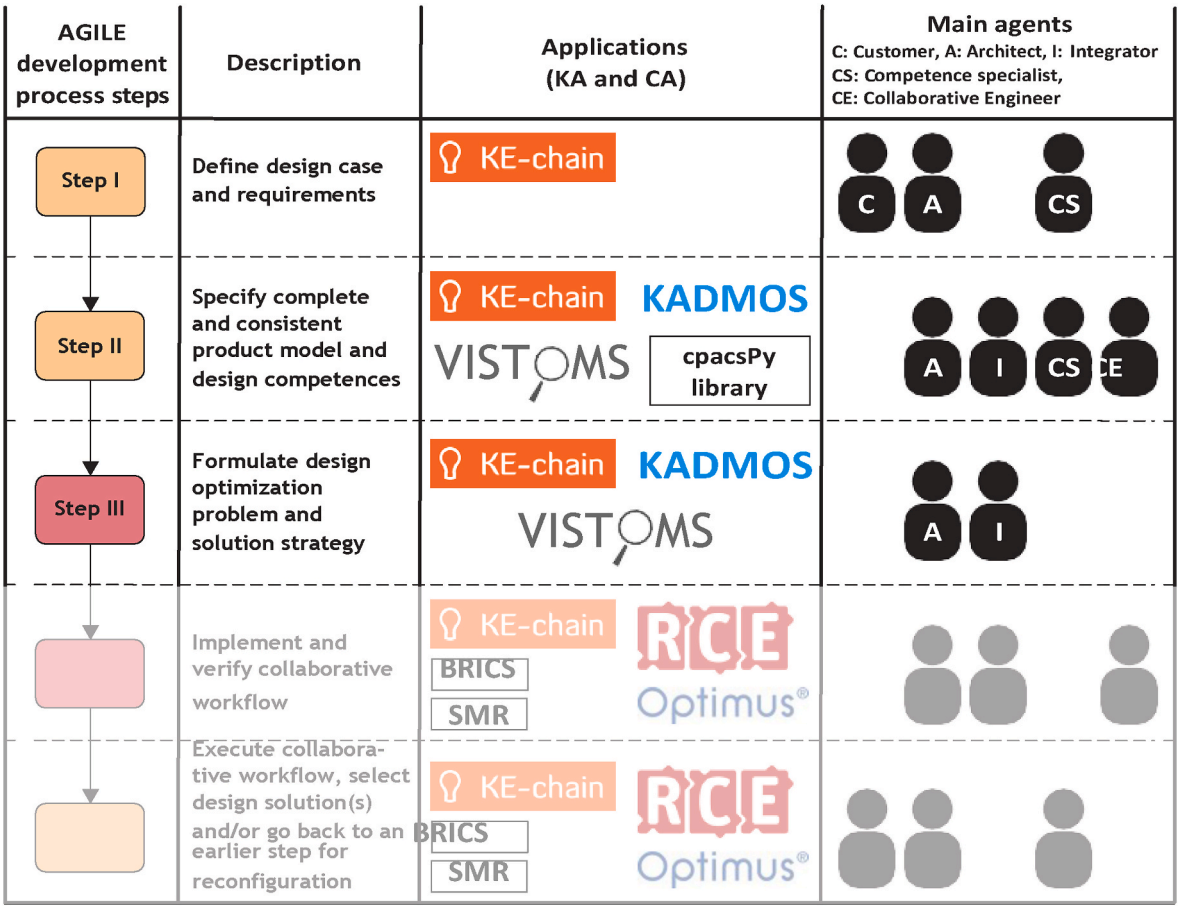


Fig. 4. Five-step AGILE development process with the first three steps highlighted as they are enabled by the KA applications. The final two steps are mainly involving CA applications.

The link between formulation and execution of the automated design process, which takes place in Step IV, is enabled through the aforementioned common workflow data schema, from the data schemas layer, which is discussed in more detail in Section 2.4.

2.3. Design competences layer

The design competences layer of the KA includes the actual design and analysis tools contributed by the various competence specialists. In order to take part in this layer, hence to become available to the other layers, all design competences must comply with the following guidelines:

- All design competences should make use of the shared product schema defined in the data schemas layer (details in Section 2.4) to extract all the necessary inputs and to store all their outputs. This requires the adopted shared product schema to be sufficiently comprehensive or extensible to allow all design competences to exchange all relevant data with it.
- Competence specialists are expected to wrap the shared product schema around their tools with support from the collaborative engineer. Thus, the service provided by a competence team should use and produce data with respect to the shared product schema [6].
- Competence specialists are expected to provide their tools together with a standard set of information, such as service description, availability, remote access details and fidelity level. These tool metadata are necessary information for the system integrator that, in Step IV, has to plug the given design competence in the computational workflow.

At the same time, design teams are granted the following rights:

- Competence specialists can bring in the design competences layer their tools of choice, either commercial or self developed.
- Competence specialists can bring in the design competences layer also workflows of tools, (pre-)assembled using any integration system of their choice.
- Competence specialists can keep their tools running on their own systems/networks and only expose them as a fully controlled web service to protect their intellectual property.

In step II of the AGILE development process all the design competences contributed by the competence specialists are stored in a virtual repository.

2.4. Data schemas layer

Data schemas are used in all layers of the KA (see Fig. 3). The project schema used in the development process layer is outside the scope. The other two schemas are briefly discussed here, namely the workflow schema used in the automated design layer and the common product schema used in the design competences layer. These two schemas are used to facilitate the exchange of data between design competences and KA applications, respectively. The use of such schemas is based on the

[6] The effort required to wrap a disciplinary tool, such to transform it in a useable design.

"common model integration" approach which enables the efficient integration of different components with a minimal amount of interface links, as is illustrated for the product schema in Fig. 5a and b. Both schemas will ultimately enable a faster integration of the automated design process in the development process layer.

Within the AGILE paradigm, all design competences require a common product schema, such that they all can read from and write to a single file. This makes any ad-hoc and direct couplings between design competences unnecessary and results in a minimal amount of interfaces. The concept of the common product schema is visualized in Fig. 5b. In addition to facilitating design competence integration, the use of a common schema keeps the domain-specific knowledge within the design competences layer, meaning that the layers built on top of it are completely unaware of the type of product that is under consideration. Hence, it does not matter for the automated design layer whether competence for the MDO system can vary significantly, from almost no effort up to many weeks of work. It depends on the tool's compatibility with the selected product schema, but, more importantly, on the developers' experience with the product schema and the creation of wrappers for it.

The product under consideration is a car, an aircraft, a satellite, or a wind turbine, as long as their design competences are integrated with a common product schema. Thereby, the AGILE KA does not only account for the heterogeneity of a team within a certain (knowledge) domain, but also the heterogeneity of product type for which automated design processes might be of interest.

Within the aircraft design community, a dedicated product schema is available, called the Common Parametric Aircraft Configuration Schema (CPACS) by Ref. [17],[7] which is further discussed in Section 3.2. In the constructions domain, BIM[8] is a widely used common data schema for buildings and infrastructure in general. A similar development is currently performed for offshore wind farms and turbines in a collaborative wind energy project by the International Energy Agency.[9]

The second schema of interest is the workflow schema. This schema enables the storage of the inexecutable automated design process (i.e. MDO solution strategy) in a neutral format. It is not only the completed automated design process that can be stored using the workflow schema; the different stages (Fig. 1) of the MDO system before the MDO solution strategy can be stored as well. This means that the repository of design competences of step II can also be stored, as well as the MDO problem formulated in step III on the basis of the available competences. This storage is required in order to be able to benefit from the different KA applications, such as multiple design competence repositories, automated design process formulation tools, visualization packages, development process integration environments and PIDO platforms. As is depicted in Fig. 5c, a workflow schema can facilitate the exchange of the same MDO system between a heterogeneous set of KA applications. Where the workflow exchange between an automated design process formulation tool and a PIDO platform is possibly the most relevant achievement in view of bridging the gap between the formulation and execution of any MDO system.

Although the two schemas have the same goal of improving interoperability (between design competences in the case of the product schema and between KA applications in the case of the workflow schema), there is a fundamental difference in their nature. A product schema is always domain specific (e.g. CPACS for aircraft, BIM for buildings), while a workflow schema for MDO systems is completely product domain independent. The workflow schema used in AGILE is called the Common MDO Workflow Schema (CMDOWS) by [36] [10]. The use of CMDOWS within an AGILE paradigm implementation is more elaborately discussed in Section 3.4.

Although the use of data schemas at the different layers of the KA matches well with the heterogeneity of teams, products, and workflows, it could also be seen as a constraint that is put on the three KA layers. This is the *paradox of standardization*: while the data schemas help the integrator tremendously in his task, individual competence specialists might feel they are loosing some freedom in defining their own competence, since any data that needs to be exchanged between members of the heterogeneous teams will have to meet schema definitions. However, the schema compliance was found to be crucial for enabling the definition and execution of collaborative workflows in large teams and therefore in the AGILE paradigm its benefits are assumed to outweigh the burden. To lighten the burden of schema compliancy, every data schema that is used should be accompanied by an ecosystem of tools, such as a schema operations library (also indicated in Fig. 5), as further discussed in Section 3.

2.5. Development process/automated design interface

The interface between the development process and automated design layer is indicated in Fig. 3 using dashed lines. This interface needs to be bidirectional. In downward direction (development process automated design) the development process tasks can control the automated design process by changing settings (e.g. a change in design requirements or a tool replacement). In upward direction (automated design development process) the specification of the automated design process needs to be brought to the development process layer, to give the agents operating in that layer the opportunity to inspect, validate and discuss the automatically generated MDO solution strategies.

2.6. Automated design/design competences interface

The bidirectional interface between the automated design and design competences layers is also shown in Fig. 3 using dashed lines. In downward direction (automated design competences) the automated design process should be able to call the different design competences. To respect the competence domain of the discipline specialist, the automated design process should only be allowed to ask for the execution of a design competence, while leaving the actual execution and feedback up to the competence specialist. This interface will prevent the automated design process to demand the execution of a design competence in ways that are outside the "comfort zone" of the design competence team.

The upward direction in this interface (design competences automated design) entails that the full definition of all design competences is made available to the automated design layer. This repository of design competences should contain the product schema, the information used and produced by each design competence with respect to this schema, and metadata on the design competence that might be relevant for the automated design process definition (e.g. how to call, average execution time, fidelity level, accuracy).

Within the AGILE paradigm it is of key importance to make this interface robust in order to "get things right", since the automated design process that is created will be large and complex, and it is difficult to find any mistakes or inconsistencies that it might contain. Firstly, an example of such a hidden inconsistency is the situation that all design competences are basing their analysis on the same product schema, but they might interpret its contents differently. A simple example being the assumed unit of the analysis results stored in the schema. Secondly, to handle the size and complexity of the automated design process a system

[7] CPACS is an open-source initiative, publicly available at: http://cpacs.de (accessed: May-2018).

[8] https://www.nationalbimstandard.org/(accessed: May-2018).

[9] http://windbench.net/iea37 (accessed: May-2018).

[10] CMDOWS is an open-source initiative, publicly available at: http://cmdows-repo. Agile-project.eu (accessed: May-2018).

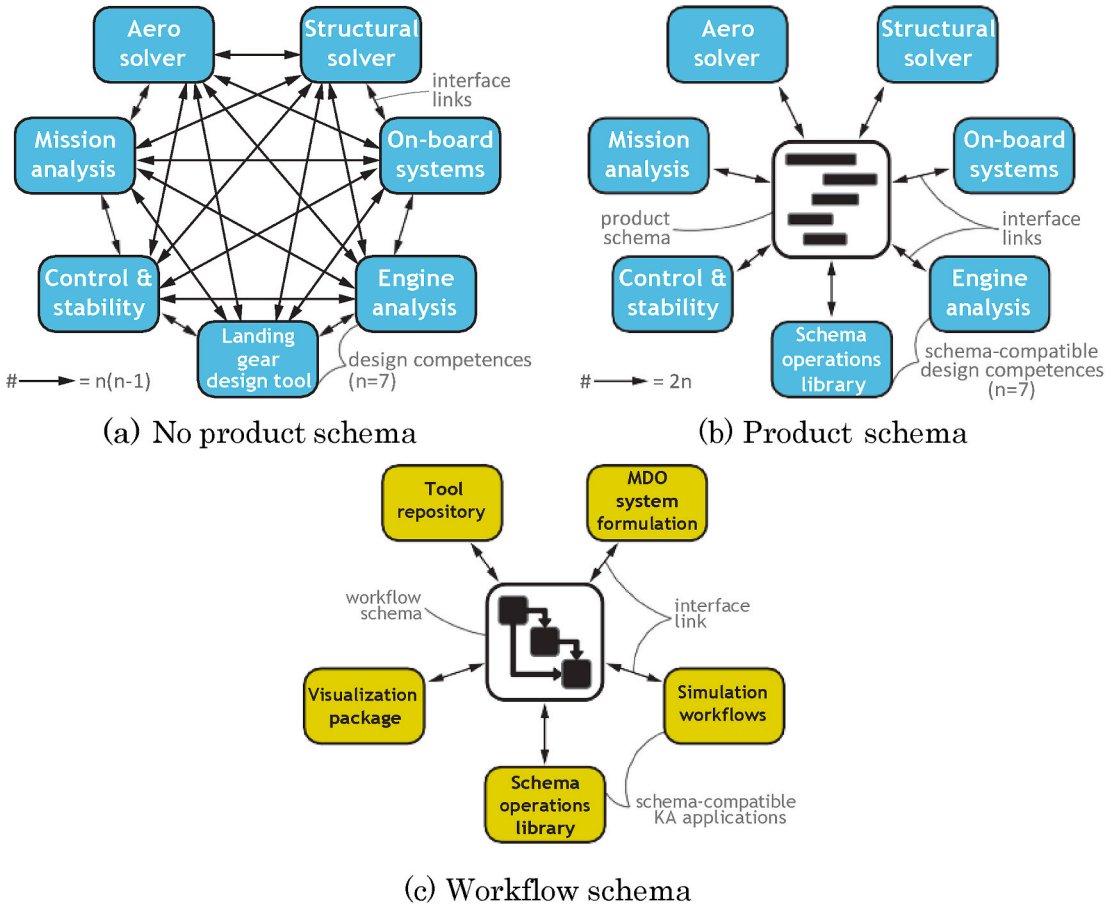


Fig. 5. Conceptual visualization of the two main data schemas used in the AGILE KA. The product schema in (b) enables the integration of an arbitrary amount of design competences (typical aircraft design tools are given as examples) using a significantly lower amount of interface links than without a schema as illustrated in (a). Similarly, the workflow schema in (c) enables the efficient connection of different types of KA applications.

needs to be available that can represent the repository of coupled design competences, such that this representation can be used to inspect the repository, and can also be manipulated to formulate the MDO problem and solution strategy stages of the MDO system. The KA applications developed in AGILE that tackle these type of issues are discussed in the next section.

2.7. Relevance of the KA

A coherent and comprehensive formalization of the collaborative MDO process is something that has been on the wish list of the MDO community for a long time, as expressed in the report on the 2011 MDO workshop [18] and the more recent workshop on Complex Systems Integration presented at the ICAS 2016 [19]. In particular, the shift from structured processes to knowledge modeling is envisioned as a key enabler for the development of the next generation of aerospace products. The panel of experts present in two aforementioned workshops have estimated a required development time between ten and twenty years. The KA proposed within the AGILE paradigm and its specific implementation in the aircraft domain represent a first fundamental advancement towards such envisioned conceptual model.

This section concludes the description of the AGILE paradigm and its KA, as a generic methodological approach to support MDO. Its specific implementation in the aircraft MDO domain, called the AGILE Development Framework (ADF), is described in Section 3. The functionality of the ADF will be demonstrated in Section 4 through the formulation of a design space exploration workflow for a passenger jet aircraft.

3. Knowledge architecture implementation: the AGILE development framework for aircraft MDO

The Knowledge Architecture described in the previous section, together with the Collaborative Architecture, represents one of the fundamental concepts of the AGILE paradigm. This paradigm provides the abstract formalization of a generic methodology to develop MDO systems, independently of the application field. Within the AGILE project, such methodology has been specifically implemented to develop a dedicated MDO framework for aircraft design, the so-called *AGILE Development Framework* (ADF), which is the focus of this section. The ADF, whose architecture is visualized in Fig. 6, is built on top of the following technology enablers and data schemas, all developed or extended during the course of the AGILE project (the institute/company that developed the item is indicated between square brackets):

- KE-chain [KE-works]: a commercial web-based platform providing the development process environment.
  - •Common Parametric Aircraft Configuration Schema (CPACS) product schema [DLR]: an open-source, de-facto standard data schema for conceptual and preliminary aircraft design.
  - •cpacsPy [DLR]: a dedicated operation library to support the use of CPACS (only available within AGILE and the DLR).
  - •Common MDO Workflow Schema (CMDOWS) [Delft University of Technology (DUT)]: a proposed new open-source standard schema to store and exchange MDO systems.
  - •Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System (KADMOS) [DUT]: an open-source, graph-

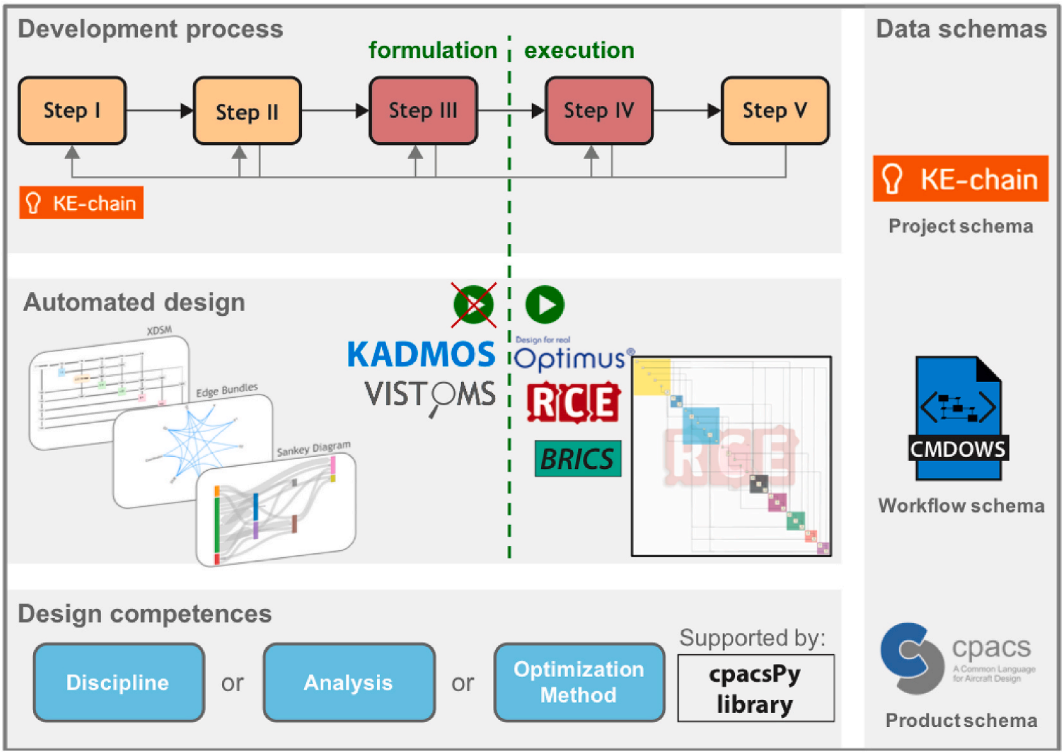


Fig. 6. Overview of the agile development framework (ADF).

based package used to enable the configuration, formalization and manipulation of MDO systems.

•VISualization TOol for MDO Systems (VISTOMS) [RWTH Aachen University]: a web-based package to enable the visualization and inspection of large MDO systems.

These key enablers, together with the interfaces between the ADF layers are discussed in detail in the following subsections and later in Section 4.

3.1. Development process environment: KE-chain

KE-chain is a web-based collaborative environment, developed by KE-works[11] to support the integration of collaborative and hybrid processes, thus combining business-type processes that require manual input and user interaction with fully automated engineering simulation workflows. The environment provides access to a single project for multiple end users through a user-based authentication system, offers functionality to facilitate the management of project data, the integration of automation solutions within the business process, and monitoring of progress. Because of these characteristics, KE-chain provides the most suitable solution for the implementation of the KA development process layer.

(Fig. 3). Through its web-based graphical user interface (see screenshot in Fig. 7), KE-chain provides control over the setup of the MDO system, based on the five-step approach discussed in Section 2.1. In other words, it provides the ADF cockpit, where the users interactively define all the aspects related to the MDO system development and constantly monitor both the development progress and the operation of the MDO system.

To achieve the aforementioned functionality, KE-chain exploits its native project model where both data and process information are stored. When the ADF is deployed, five main KE-chain steps are defined within the project, thus one for each step in the aforementioned five-step approach. The screenshot in Fig. 7 shows an overview of the defined steps broken down in substeps. Each design step has its own custom view, which allows users to edit and view data defined in the project's data model.

In step I, substeps are defined to gather the design competences available through the various competence specialists in the team and to collect the customer's requirements. In step II, substeps are defined to set up the repository of available design competences and its input/output relations with the aircraft product model, based on the CPACS schema. In steps III and IV, other sub- steps are defined to trigger and operate KADMOS (Section 3.5), which is in fact the main application in the overall automated design layer of the ADF for defining the MDO solution strategy and supporting the integration of the executable MDO workflow. In these steps, through the integration of the VISTOMS application (Section 3.6), it is also possible to visualize and inspect the automatically generated MDO system formulations. In step V, substeps are de- fined to allow inspecting the generated design results, through the integration of Noesis's post-processing tool id8[12]. In Section 4, the use of these design tasks is explained in more detail for the first three steps.

3.2. Product schema: CPACS

The Common Parametric Aircraft Configuration Schema (CPACS) is the common product schema used in the ADF. It is developed and supported by DLR and distributed as open source.[13] CPACS is based on an XML schema definition, which makes it both human and machine readable, and provides a hierarchical structure to store all the relevant aspects (geometry, performances, flight conditions, etc.) used in conceptual and preliminary aircraft design. Since the main purpose of CPACS is to provide one common data model for multiple analysis tools,

[11] http://www.ke-works.com (accessed: May-2018).

[12] https://www.noesissolutions.com/our-products/id8 (accessed: May-2018).

[13] https://github.com/DLR-LY/CPACS (accessed: May-2018).

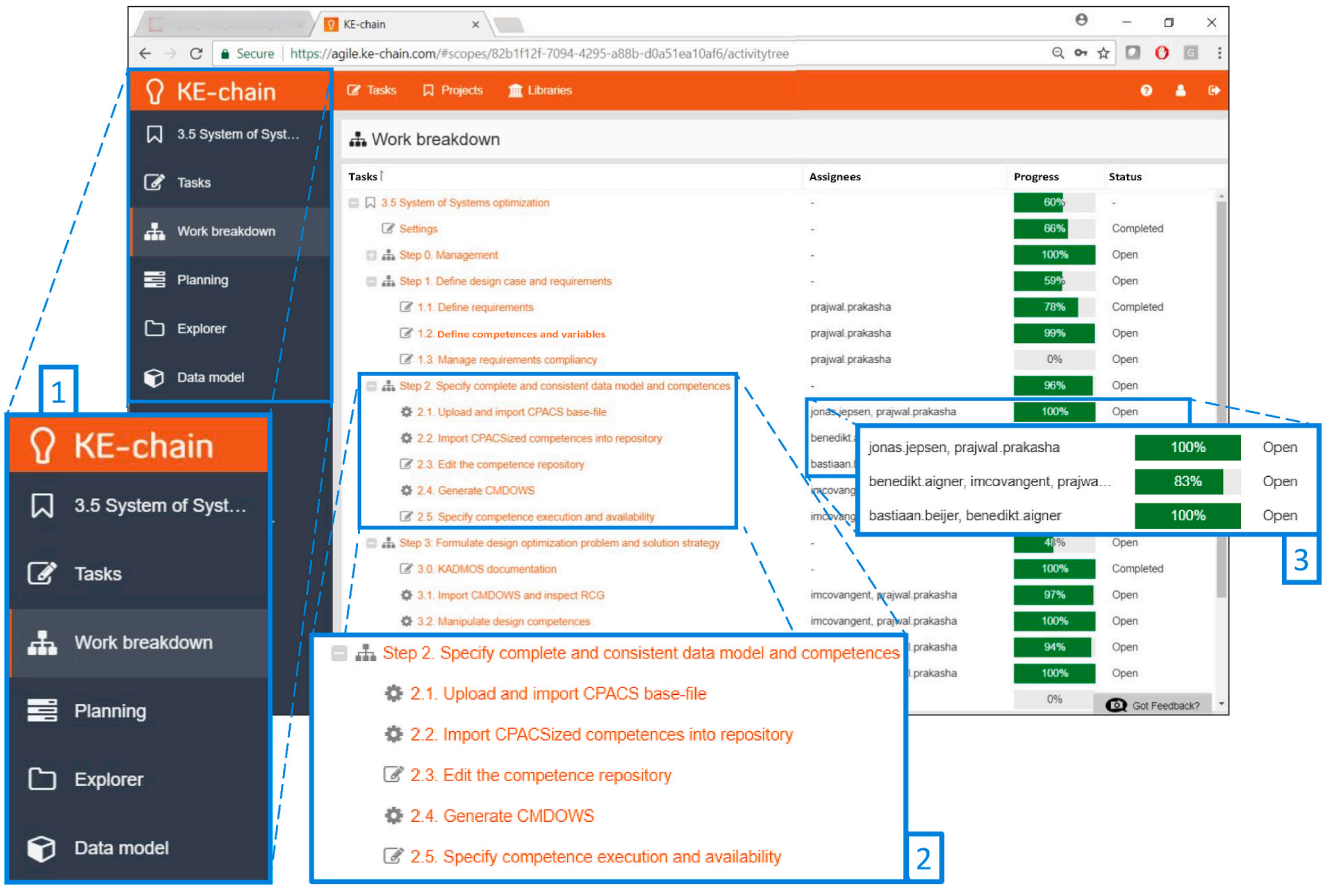


Fig. 7. Screenshot of KE-chain's web-based GUI with some key features: (1) project's navigation panel to view tasks, the work breakdown overview (currently displayed view), data model, etc. (2) breakdown of manual and automated development process tasks and sub-tasks, (3) progress, status and user assignment overview.

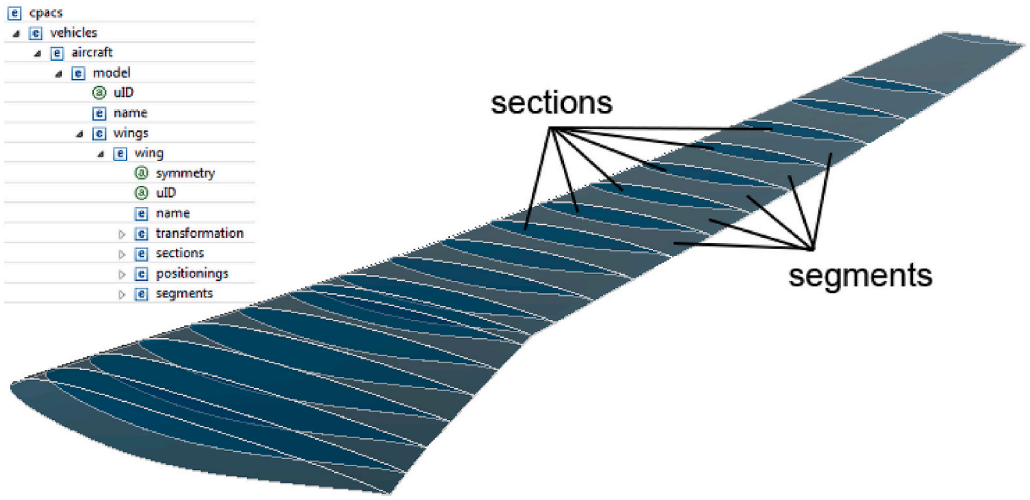


Fig. 8. CPACS wing segments and sections.

as illustrated in Fig. 5b, the data in the schema were selected by the CPACS developers in order to provide a product representation that is acceptable for a large variety of disciplines, while avoiding any data redundancy. An example of the way CPACS stores aircraft data is depicted in Fig. 8.

Multiple aircraft can be defined in a single CPACS file and each of them can include an arbitrary number of wings. Each wing is defined by sections and segments. Sections are aerodynamic profiles (the normalized airfoil profiles are stored in another part of the schema) which are scaled, rotated and positioned in space, whereas segments define the connection of two individual sections to form a lofted wing surface.

While this generic geometry definition allows describing a large range of wings, typical high-level parameters such as span or aspect ratio, both commonly used in aircraft conceptual and preliminary design, are actually not stored in CPACS to avoid redundancy. The

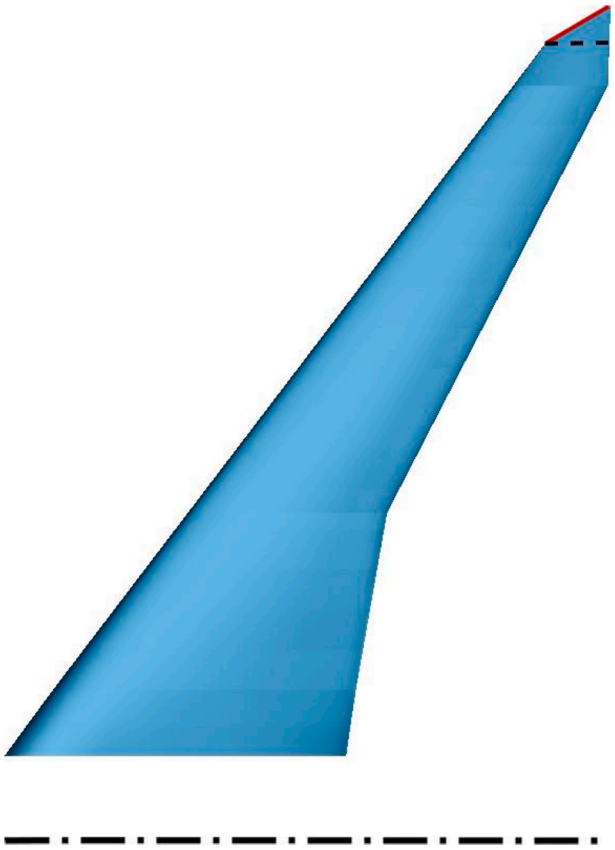


Fig. 9. CPACS wing with rotated tip section. The span-wise location of the reference point of the tip section is not necessarily the most outboard point defining the wing span and design competences. When done properly even the gap between design competence of different fidelity can be closed easily.

responsibility to determine their values on the basis of the actual CPACS wing definition (e.g. using the definition of the various sections) is left to the competence specialists. Any "translation" of the CPACS representation into high-level design parameters (and vice versa), apart from.

Requiring some data processing effort, can introduce errors and inconsistencies into the design process if not properly done [20]. For example, if one would want to evaluate the high-level parameter wing span by computing the distance between the innermost and outermost wing section stored in CPACS, this would lead to a wrong value in case, for example, the root or tip sections are rotated, or the root section is not defined at the fuselage center line (as conventionally) but at the wing-fuselage intersection, as shown in Fig. 9.

In order to exploit the benefits of the central data model approach while avoiding any design inconsistency caused by possible different interpretations of a CPACS file, a dedicated library of so-called *design concepts* has been developed. These are discussed in the next subsection.

3.3. Design concepts: cpacsPy

When exchanging data in collaborative design processes, it is fundamental that the provider and recipient of information have the same understanding of transferred data. Both need to make sure they interpret the shared data based on the same *concepts*,[14] otherwise inconsistencies can arise and invalidate the obtained results. In AGILE the Python library cpacsPy, developed at DLR, was used to access CPACS

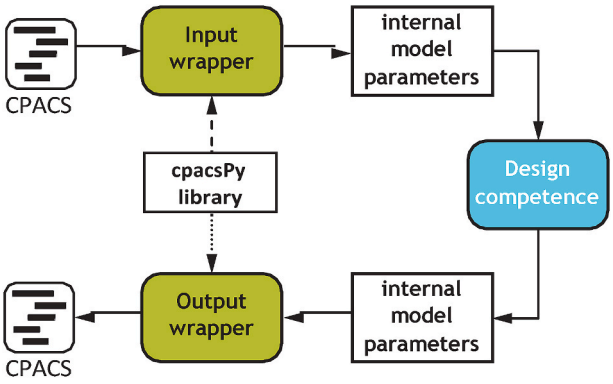


Fig. 10. CPACS wrappers.

data through predefined concepts, such as the use of trapezoid wing segments to determine top-level design parameters like sweep, taper ratio, aspect ratio and wing area. CpacsPy can simply be used as an external wrapper component for data conversion as illustrated in Fig. 10. The main role of cpacsPy is to simplify the exchange of data between CPACS.

Within AGILE different design studies have been performed involving wing planform parameters as design parameters for the optimization. While the optimization parameters were typically high-level, such as wing span and wing aspect ratio, the analysis methods involved in the design process required a much more detailed description of the wing shape. Therefore it was decided to generate a detailed shape from a baseline configuration and to morph that shape using the planform parameters typically used in conceptual design. For this task a simple wing concept from cpacsPy was used which allows to access a CPACS wing model using parameters such as span, aspect ratio, area, sweep, root chord, tip chord and taper ratio and to modify the detailed baseline wing shape accordingly, by mapping the concept parameters onto the detailed CPACS model. A description of the cpacsPy capability to support this specific task is given in Section 4.

Due to its modular structure additional concepts can be added as individual modules to cpacsPy, provided that all assumptions made for the mapping are documented in the concepts module description.

3.4. Workflow schema: CMDOWS

CMDOWS is an open-source,[15] XML-based schema, developed at DUT. As CPACS enables communication among the various design competences, similarly CMDOWS enables the interoperability of the different KA applications within the ADF, as illustrated in Fig. 5c. The main KA applications connected through CMDOWS are the aforementioned KE-Chain, plus KADMOS and VISTOMS (Fig. 6), which will be addressed in the next two subsections. In practice, CMDOWS serves as a dynamic storage schema where CMDOWS instances contain the evolving definition of the MDO system during its development process throughout the five-step approach described in Section 2.1. In particular, CMDOWS instances are generated from Step II to Step IV, as described below and illustrated in Fig. 11:

Step II: At the end of step II, KE-chain produces a first instance of a CMDOWS files containing the list of design competences contributed by the various competence specialists, together with their input and output, expressed as links to the specific CPACS nodes.

Step III.1/2/3 First, KADMOS receives the CMDOWS file; second, based on the definition of the optimization problem specified in KE-chain by the user, it removes the unnecessary design competences from CMDOWS; third, it enriches it by adding the MDO problem

[14] [34] define concepts as: "structured mental representations that encode a set of necessary and sufficient conditions for their application, if possible, in sensory or perceptual terms".

[15] Available at: http://cmdows-repo.agile-project.eu (accessed: Jul-18).

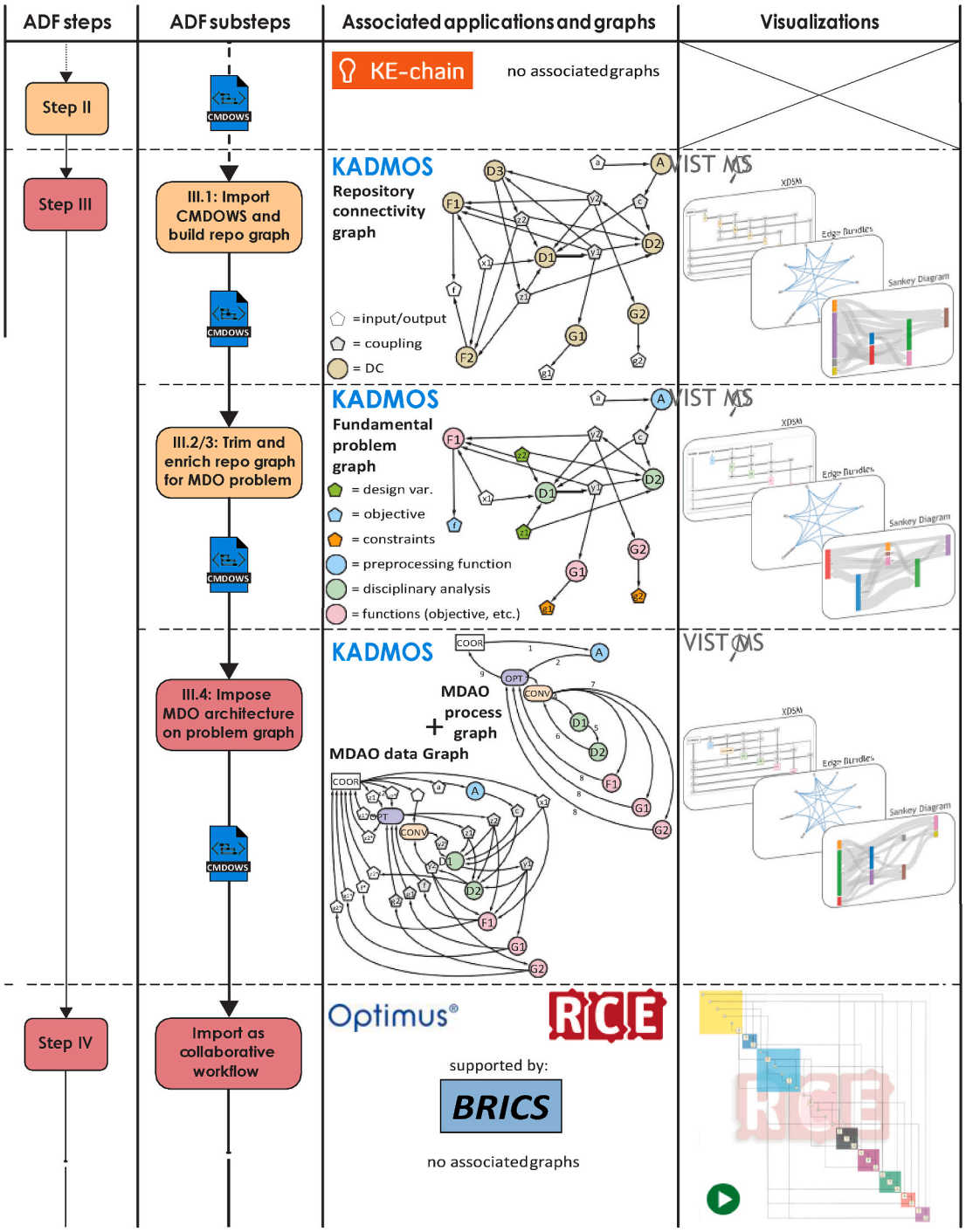


Fig. 11. Top-level overview of KADMOS and its relation to the five-step approach in the development process (all visualizations are based on the Sellar problem [21]) not (fully) machine-readable and thereby cannot be used for any further automated analysis, integration, or manipulation of the MDO system. Concerning the representation of workflows and computational frameworks in general, some of the commonly used visualizations methods include N2 charts [22], functional dependency tables [23] and (extended) design structure matrices (XDSM) [16,24]. While, in principle, a visualization method such as the XDSM offers a comprehensive method to capture the full description of an MDO system, including system couplings and service execution order, in practice it is not readily applicable to large and complex problems. A drawback of the aforementioned matrix-based representations is the fact that, due to their static form, they are not arbitrarily scalable to large system representations, thus their readability degrades with the size.

definition (e.g. design variables, objective, constraints), as specified in KE-chain by the architect.

Step III.4: Such modified CMDOWS file is further processed by KADMOS, to formalize the fundamental optimization problem according to the available MDO architecture, selected by the architect, still via KE-chain.

Step IV: The CMDOWS file, now containing the complete description of the MDO system, is passed to the Collaborative Architecture block of the AGILE environment, where it is used to trigger the automatic generation of the executable MDO workflow, in one of the selected PIDO platforms (e.g. Optimus or RCE).

At each (sub-)step, the CMDOWS file can be accessed by VISTOMS (Section 3.6) to generate convenient visualizations of the MDO system throughout its progressive development. This will offer all ADF

stakeholders the possibility to inspect and maintain oversight of MDO systems of any complexity. From Step IV description, it becomes evident that CMDOWS is not only playing the role of hub for the various ADF applications, but it is also the bridge between the formulation and the execution phases of the AGILE paradigm, as indicated in Figs. 3 and 6.

The neutral representation of the MDO system stored in CMDOWS is used to automatically generate executable workflows in PIDO platforms, by means of CMDOWS parsers specifically developed for these platforms. These parsers translate the neutral elements and processes defined in CMDOWS to platform- specific elements and processes, resulting in an executable workflow. This parsing process is further discussed in a companion paper by Ref. [13] and in earlier work by [37].

The description of the full CMDOWS is discussed in detail in [36]. The root of the schema and its six main elements are shown in Table 1. Their use for the different MDO system stages is also shown in the table, including a basic description. At the final stage of the formulation phase of the MDO system (i.e. MDO solution strategy), the full schema is used to describe the data and process flow required to solve the given MDO problem. The definition of CMDOWS is structured based on the notion that any MDO system can be represented as a directed graph. A directed graph is a mathematical construct in which nodes and connections are used to describe a system and where additional information can be added as well, either on the graph itself, or its nodes and edges. This relation between CMDOWS and directed graphs is indicated in the last column of Table 1 and will be further discussed in the next section.

3.5. Graph-based formulation support: KADMOS

KADMOS [25] is an open-source[16] Python package, developed at DUT to tackle the challenge of formulating large collaborative MDO systems, while providing users with necessary overview to stay in control of their complexity [26] and offer the agility to adjust and scale them [17] to achieve that, KADMOS supports the following three tasks:

The use of KADMOS within the ADF is depicted in Fig. 11. In this figure the small MDO benchmark “Sellar problem” [21] is used in the third and fourth column to clarify the ADF steps and substeps. KE-chain is used to define the repository of design competences in step II and passes that information to KADMOS through a CMDOWS file. At the beginning of step III, KADMOS establishes the repository connectivity graph (henceforth referred to as repo graph), which represents the system of coupled design competences in a directed graph. The repo graph can be checked and manipulated and the updated repository stored again as a CMDOWS file. Based on the MDO problem definition passed again by KE-chain, the repo graph is transformed into the fundamental problem graph (henceforth referred to as problem graph) using KADMOS methods to, for example, remove unnecessary design competences from the graph, indicate which of the elements from the product schema have special roles (design, objective, constraint), and establish a basic order of execution for the design competences. Again, a new CMDOWS file can be generated and used also for inspection work. Four main differences between the implementations are: KADMOS.

(1) supports the central product schema approach.
(2) supports the import and export of the graphs to the neutral CMDOWS format,
(3) is based on different graph-theoretic conditions and
(4) has a more sophisticated handling of complex situations (e.g. variable collisions) in the graph.

As final transformation step, KADMOS can automatically impose an MDO architecture on the problem graph, leading to the generation of two new graphs, namely the MDO data and process graph. These two graphs together provide the complete formulation of the automated design process to be executed: the MDO solution strategy. They are stored again in the form of a complete CMDOWS file, ready to be translated into executable workflows (as well as for inspection). A key advantage of KADMOS is that it fully separates the formulation of the MDO problem and the solution strategy used to solve it. Hence, on the same optimization problem definition, different MDO architectures can be imposed, provided that the problem itself contains characteristics required for the architecture that is selected, e.g. some distributed architectures expect both local and global design variables to be present. Currently, KADMOS supports the formulation of the automated design process for two monolithic MDO architectures [11], Multidisciplinary Feasible (MDF) and Individual Discipline Feasible (IDF) and two distributed MDO architectures, Collaborative Optimization (CO) and surrogate-based Bi-Level Integrated System Synthesis (BLISS-2000). Besides, it can also impose MDA architectures such as design point convergence and design of experiments (DOE).

It should be noted that in the ADF, KADMOS is executed through the user interface provided by KE-chain. Hence, in Fig. 3 KADMOS provides the upward design competences/automated design interface for formal workflow specification. In other words, whereas KE-chain provides the ADF cockpit, KADMOS is the engine running under the hood. The graphs created and manipulated by KADMOS use or affect all layers of the KA. The creation and manipulation of the graphs by KADMOS are a matter of seconds for the majority of cases, with a maximum evaluation time of up to 1 min for very large MDO systems in combination with the most complex manipulations the package can perform. Where there would normally be a gap between the specification of the automated design process to be executed and the actual executable collaborative workflow, KADMOS and CMDOWS enable a design team to go, in an agile way, from a repository of design competences to a fully configured workflow in Optimus or RCE, as depicted in the last row of Fig. 11.

3.6. Visualization of large, complex MDO systems: VISTOMS

The number of design competences and especially the typical amount of exchanged data within a collaborative MDO system can be so large that is nearly impossible to comprehend all underlying, relevant information at a glance. As a consequence, keeping oversight and control (hence trust) on the overall MDO system and the ability to inspect and debug it, is an extremely challenging job for the integrator. Rendering the relevant information in a human-intelligible way, by means of effective visualizations is of paramount importance. In current practice, different approaches are used. For example, a mix of spreadsheets and text documents are used to describe the various design competences, with all their input and output. However, these need to be updated manually and are difficult to keep consistent. Moreover, such manually created overviews are not (fully) machine-readable and thereby cannot be used for any further automated analysis, integration, or manipulation of the MDO system. Concerning the representation of workfows and computational frameworks in general, some of the commonly used visualizations methods include N2 charts [22], functional dependency tables [23] and (extended) design structure matrices (XDSM) [16]. While, in principle, a visualization method such as the XDSM offers a comprehensive method to capture the full description of an MDO system, including system couplings and service execution order, in practice it is not readily applicable to large and complex problems. A drawback of the aforementioned matrix-based representations is the fact that, due to their static form, they are not arbitrarily scalable to large system representations, thus their readability degrades with the size.

Therefore, within the ADF, a web-based visualization package called VISTOMS (VISualization TOol for MDO Systems) has been created by RWTH Aachen University, which is able to translate any MDO system

[16] Available at: http://kadmos-repo.agile-project.eu/(accessed: Jul-18).
[17] Conceptually, the graph-based approach in KADMOS is based on the approach proposed by Ref. [26]; though the technical implementation varies significantly from their.

Table 1
Tabular top-level overview of the CMDOWS schema for MDO system representations.

- **Create:** Enabling the specification of large, complex MDO systems by means of graphs, starting from the definition of the design competence repository, to the formulation of the MDO problem, and concluding with the complete MDO solution strategy, corresponding to the first three stages in Fig. 1, respectively.
- **Inspect & debug:** Enabling different experts (i.e. competence specialists, integrators) to inspect sections of the system that are relevant to them, in order to validate it, thereby increasing the level of trust in the system. In addition, KADMOS provides automated validation functions to check whether a valid MDO system is specified based on strict conditions on the graph construct (e.g. all design competences should have at least one output, all nodes should be connected, etc.).
- **Manipulate:** Automatically manipulating the generated MDO system specification using graph-based analysis and dedicated algorithms. These computerized manipulations, apart from providing drastic time reductions, also reduce the chances of errors and inconsistencies in the system by eliminating repetitive error-prone human tasks. Examples of simple manipulations are the processing of changes in the input or output variables of a design competence. Examples of more advanced manipulations are the automated transformations required to go from one MDO system stage to the next, i.e. from a repository of design competences to an op- timization problem definition and from that to a complete formulation of the MDO solution strategy according to a selected MDO architecture.

Schema			Description	Used to describe MDO system stage (Fig. 1)			Graph element type
Root	elements			repo	problem	strategy	
	header		Metadata about the file (e.g. creator, creation date, version, etc.)	✓	✓	✓	information
	Problem Definition		Definition of the MDO problem to be solved (design competences used, design variables, constraints, etc.)	×	✓	✓	
	executableBlocks	Design Competences	Function definition (inputs, outputs, metadata) of executable blocks that are not simple mathematical relations.	✓	✓	✓	nodes
		Mathematical Functions	Function definition (inputs, outputs, equations) of executable blocks that can be described by a set of mathematical relations.	✓	✓	✓	
	parameters		All elements from the product schema that are used as inputs or outputs of the executableBlocks.	✓	✓	✓	
	Architecture Elements		Additional elements that are required to solve an MDO problem according to a certain architecture, such as initial guesses, copy variables for convergers, etc.	×	×	✓	
	workflow	Data Graph	All data input (e.g. parameter x → executableBlock F) and output (e.g. executableBlock F → parameter y) connections of an MDO system	✓	✓	✓	connections
		Process Graph	Full process description of an MDO solution strategy using connections and step numbers between exectable Blocks (e.g. F –(step 3)-→ G)	×	×	✓	

information, as stored in a CMDOWS file, into dynamic, interactive and human-readable plots and diagrams, in a web-based interface. The open-source[18] tool is accessible via any web browser and can be used without installing additional software. In VISTOMS, visualization techniques such as XDSMs [16,27], Sankey diagrams [28–30] and hierarchical edge bundles [31,32] are utilized and enhanced with help of D3.js, an open-source JavaScript library. D3.js is specifically tailored for the dynamic visualization of any kind of data [32] and therefore enables the desired capabilities for the ADF. Large and complex XDSMs can, for instance, be expanded or collapsed in order to set the focus on certain aspects of the MDO system. Couplings between services or variable interdependencies can be analyzed in human-readable diagrams, in which it is possible to dynamically reveal or hide detailed information, for example via mouse clicking or hovering. This enables insight into information that could either not be visualized at all, or too confusing to comprehend when visualized all at once. Thus, VISTOMS can be used as an effective debugging environment, in which an MDO architects and integrators are able to examine the created solution strategies (stored as CMDOWS instances), review the steps of the product development process and detect possible issues. Several examples of produced visualizations can be found in the next section on the ADF demonstrator and in Ref. [33].

4. AGILE Development Framework demonstrator: formulation of an aircraft MDO system problem

The demonstrator of the ADF is based on the work performed in one of the design campaigns of the AGILE project. The design campaign targeted the conceptual and preliminary design of a medium-range twin-engine jet airliner (Figure 12). A heterogeneous collection of CPACS-compliant design competences has been used to set up a distributed automated design process. In this section, the first three steps (formulation) of the AGILE development process (Fig. 4) will be used to demonstrate how the ADF comes into play in a realistic design case. The last two steps (execution) will be addressed only briefly, details can be found in Ref. [13]. For each step, the following information will be provided: description of the main function, associated deliverable, deployed KA applications and main agents involved.

4.1. Step I: define design case and requirements

Function: The design (and optimization) case is identified and the requirements, both for the system to be developed and the development process itself, are defined and managed. This step includes also the definition of all the design competences contributed by the various discipline specialists in the team.

Agents: Customer, Architect, Competence specialists.

KA applications: KE-chain.

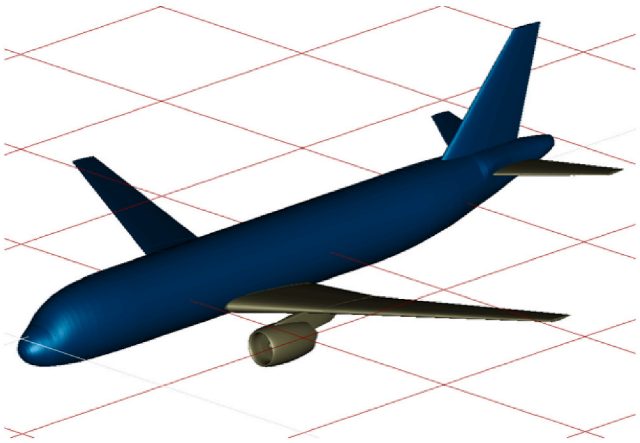
Deliverable: A description of the design case stored, editable, and inspectable in KE-chain.

Substeps: The step is divided into three substeps, in which the following actions are performed:

Step I.1 Definition of the MDO system requirements. The main information on the design case is collected from the involved customer, and translated by the architect into a set of requirements. These include performance specifications for the product and process, and corresponding means of compliance. The main product-related requirements specified in this demonstrator are the top-level aircraft requirements for the design of a medium-range jetliner (see table in Fig. 12). The main requirement on the development process included in the demonstrator concerns the scope of the design study, which in this case is the conceptual design of the full aircraft, with special attention to the wing-engine integration. A second example of requirement on the development process regards the target lead time for the project.

Step I.2 Listing the design competences and variables. A set of competences available for the resolution of the design case is collected

[18] The source code of VISTOMS is available inside the KADMOS repository [6]: bitbucket.org/imcovangent/kadmos (accessed: Jul-18).



Top-level aircraft requirements

Requirement	Value	Unit
Entry into service	2020	-
Range	3500	km
Design payload	9180	kg
Max. payload	11500	kg
PAX (@ 102kg)	90	-
MLM (%MTOM)	90%	-
Long range cruise Mach	0.78	-
Initial climb altitude	11000	m
Max. operating altitude	12500	m
Residual climb rate	91	m/min
TOFL (ISA, SL, MTOM)	1500	m
Vref (ISA, SL, MLM)	<130	kts
Dive Mach number	0.89	-
Fuel reserves	5%	-

Fig. 12. Impression of the medium-range twin-engine jet airliner under consideration for the demonstrator (left) including the top-level aircraft requirements (TLARs) listed in a table (right).

by the architect and the competence specialists. The information provided (e.g. fidelity level, expected runtime) supports the selection of design competences in a later stage, and the identification of any missing ones. At this stage, also a set of special variables relevant to the design process is assembled, as well as their roles within the design and optimization task (e.g. objective of the design and design variables). In a later stage these variables are mapped to elements of the product schema and become the variables of the MDO problem. Eleven relevant design competences were gathered from the team, see Table 2. The selected design variables were wing area, sweep and aspect ratio. The objective variables were the two key performance indicators direct operating cost (DOC) and the Average Temperature Response (ATR), used to measure the environmental impact of the design.

Step I.3 Requirements management. The architect keeps track of the requirements compliance during the execution of the entire development process. This step is continuously updated throughout the project execution.

4.2. Step II: specify complete and consistent product model and design competences

Function: The purpose of the second step is to define a complete, consistent and compliant repository of design competences. Therefore, the variables and competences gathered in the first step are assembled such that at least one complete process can be obtained. The assembly of the variables and competences is based on the product schema CPACS. Hence, the definition of the design competences with respect to that schema plays an important role in this step. Furthermore, special variables such as design variables and objective/constraints listed in Step I have to be mapped to the corresponding elements in CPACS.

Agents: Architect, Integrator, Competence specialists, Collaborative engineer.

KA applications: KE-chain, VISTOMS, KADMOS.

Deliverable: A CMDOWS file containing a complete and consistent repository.

Substeps: Five substeps have been identified, namely:

Step II.1 First, a so-called *base file* is instantiated by the integrator. This base file is a CPACS file that contains the definition of the aircraft geometry under consideration and the elements from the schema that are expected to be used to store analysis results (e.g. aerodynamic coefficients, weights). This base file is created to assure that all competence specialists use and produce data with respect to the expected XML elements of the full CPACS.

Step II.2 The competence specialists should develop (or check existing) CPACS-compliant design competences using the base file and provide the integrator with one CPACS input and one CPACS output file for each design competence. In this task, they are supported by the collaborative engineer. The input file should contain only the elements that a design competence needs for its execution. The output file should contain only the specific elements from the schema that the design competence is adjusting or creating. The integrator then imports these CPACS-compliant competences into a repository in KE-chain. Using

Table 2
Design competences used in the demonstrator. All the design competences have intellectual property restrictions and can therefore not be made available on one server environment that is shared with other partners.

Design competence Name	Description	Tool Provider	
Aircraft Synthesis	Aircraft initialization & overall aircraft synthesis	DLR, Germany	Hamburg,
Morphing	Aircraft geometry morphing	DLR, Germany	Hamburg,
Aerodynamics	Calculation of aerodynamic performance map	DLR, Germany	Hamburg,
Structural Mass	Analysis of structural masses	DLR, Germany	Hamburg,
On-Board Design Systems	Design and analysis of on-board system architecture	PoliTo, Turin, Italy	
Engine	Detailed engine design & performance map calculation	CIAM, Russia	Moscow,
Nacelle Design Aero Integration	Design & integration of engine nacelles. Aerodynamic analysis of nacelles	TsAGI, Zhukovsky, Russia	
Mission Simulation	Performance analysis of aircraft flight mission	DLR, Germany	Hamburg,
Mass Budget	Update of mass breakdown for consistency of data set	DLR, Germany	Hamburg,
Cost Analysis	Calculation of non-recurring, recurring & operational costs	RWTH, Germany	Aachen,
Emission Analysis	Calculation of exhaust emissions & climate metrics	RWTH, Germany	Aachen,

Of design competences which are compliant with CPACS.

these CPACS input and output files, KE-chain builds up a native project model containing the design competences and product model definitions.

Step II.3 The architect can now edit the design competence repository in consultation with the competence specialists to fix any inconsistencies that might have been found in the coupling between design competences. In practice, this step is used to fix inconsistencies in the project model from step II.2 that are found in a later stage, for example when inspecting the model using VISTOMS.

Step II.4 The integrator can now generate the CMDOWS file to enable other KA applications to access the definition of the tool repository.

Step II.5 Finally, the competence specialist should specify the competence execution method and availability (i.e. as a local tool that can be installed and run locally or a remote service that needs to be called through a service-oriented architecture). This final step is meant to prepare all information required for execution and inspection of the MDO solution strategy in steps IV and V and it is added to the CMDOWS file once it becomes relevant in step IV.

4.2.1. Use of KE-chain in step II of the demonstrator

To assist the integrator and competence specialists in the definition of a complete, consistent and compliant repository of design competences and product model, KE-chain provides engineering services to easily build, manipulate and inspect the initial available design competences. The first engineering service, used in step II.1, enables the integrator to upload a single or multiple CPACS base-files, which contain the full aircraft product parametrization and import this into the KE-chain platform. Importing the CPACS data model in KE-chain provides full inspectability and manipulation of attributes in a non-programmer friendly environment. Based on the CPACS base-file, the integrator defines input and output CPACS files for each available design competence. Alternatively, the integrator can also upload a CMDOWS file containing the input and output definition of a design competence.

In step II.2, the integrator is supported by an engineering service which parses the uploaded CPACS or CMDOWS file using XML technologies and uses them to configure the imported data schema elements as input and/or output to construct an initial MDO system using pykechain, KE-chain's Python API[19]. This MDO system, a network of imported design competences coupled through the various data schema elements, can be fully inspected or manually manipulated in KE-chain as is illustrated in the lower half of Fig. 13. The configuration of each design competence can be manually edited further in step II.3.

In step II.4, an engineering service enables the integrator to generate a CMDOWS file of the repository model defined in the previous steps. KE-chain provides an interface to the integrator in which versions of the generated CMDOWS file can be managed, and the connectivity of the different design competences can be inspected in more detail through the integration of VISTOMS, described in more detail in the next section. The VISTOMS visualization of the design competence repository is automatically created from KE-chain, and can be downloaded for local use, or inspected in more detail directly in the browser. Finally, step II is concluded by an editable table, in which information is gathered on the execution details of each design competence which will be used in step IV.

4.2.2. Use of VISTOMS in step II of the demonstrator

In this early stage of the problem formulation the MDO system is represented by the repo graph, i.e. the graph containing the design competences, product model elements, and the links between them, and does not yet take into account a specific design problem having a dedicated tool order, DOE or optimization behind it. A part of the repo graph for the presented design case is shown in Fig. 14 using the XDSM visualization of VISTOMS.

Note that the actual repo graph for the presented example is significantly larger than the extract shown in the figure. The blue overlay boxes (not present in the actual VISTOMS package) have been added in the figure to emphasize some of its visualization capabilities. One of them is the option to detect element collisions, i.e. elements which are written by multiple design competences simultaneously. Overlay frames 1 and 2 in Fig. 14 display for instance a so-called collided circular coupling of the main wing. This occurs due to the fact.

The dynamic visualization capabilities combining different visualization elements such as XDSMs for the general layout, hierarchical tree views for the data model (including different parameter categorizations), and the capability to highlight certain points of attention in the MDO problem, together enable an efficient and effective visualization of all the information stored in an MDO system that would normally be implicit and almost impossible to access.

4.2.3. Use of cpacsPy package in step II of the demonstrator

By inspecting the connections between the available design competences, a missing link was identified between the high level design variables (area, aspect ratio and sweep) selected to control the wing geometry and the more detailed CPACS geometry used by some of the design competences. In order to resolve this parametrization gap, a new design competence, called the Morphing tool (see XDSM in Fig. 14) was developed. The Morphing tool uses a simple description of a wing, based on a single trapezoid element, whose planform is controlled by the aforementioned high-level design variables. In order to cre- ate the detailed geometry eventually required by the other design competences, a baseline CPACS file was used, containing an initial detailed description of a wing. The otherwise cumbersome transformations required to adjust the detailed CPACS wing geometry representation (based on multiple sections and segments), using few conceptual planform parameters, was completely automated by the morphing tool making use of the *trapezoidal wing concept*, available in the cpacsPy library.

The mapping between the concept variables and the CPACS model is defined as follows:

- The single trapezoid wing area is the sum of all segment areas. $S = \Sigma S_i$ where S_i is the area of the i-th wing segment.
- The single trapezoid wing sweep is the mean sweep of all segments weighted with by their length. $\Lambda = {}^{\Sigma(\Lambda i \cdot li)}$ where Λ_i is the planform sweep of the i-th segment, and l_i is the length of the i-th segment.
- The single trapezoid wing aspect ratio is defined as $AR = {}^{\Sigma l_{i,spanwise}{}^2}$ where $l_{i,spanwise}$ is the span-wise length of the i-th segment and S is the wing area as defined above.

The cpacsPy trapezoidal wing concept allows users to define the start and end segment of any wing defined in CPACS. This enables the Morphing tool to exclude from the wing overall sweep definition the eventual rectangular segment spanning from the fuselage center line to the fuselage-wing intersection (see Fig. 15).

Because of the availability of the trapezoidal wing concept in the cpacsPy library, the development effort for the Morphing tool was minimal. The demonstrator proved that this new design competence enables the quick and reliable modification of a large number of CPACS parameters, based on a few high-level design parameters.

4.3. Step III: formulate design optimization problem and solution strategy

Function: The goal of this step is to go from a repository of design competences to an automated design process formulation of the collaborative workflow to be executed in step IV.

Agents: Architect, Integrator.

KA applications: KADMOS (accessed via KE-chain interface), VISTOMS.

[19] pykechain is a Python library for advanced users and KE-chain configurations to connect and interact fully to all features of KE-chain http://pykechain.readthedocs.io/en/latest/(accessed: Jul-18).

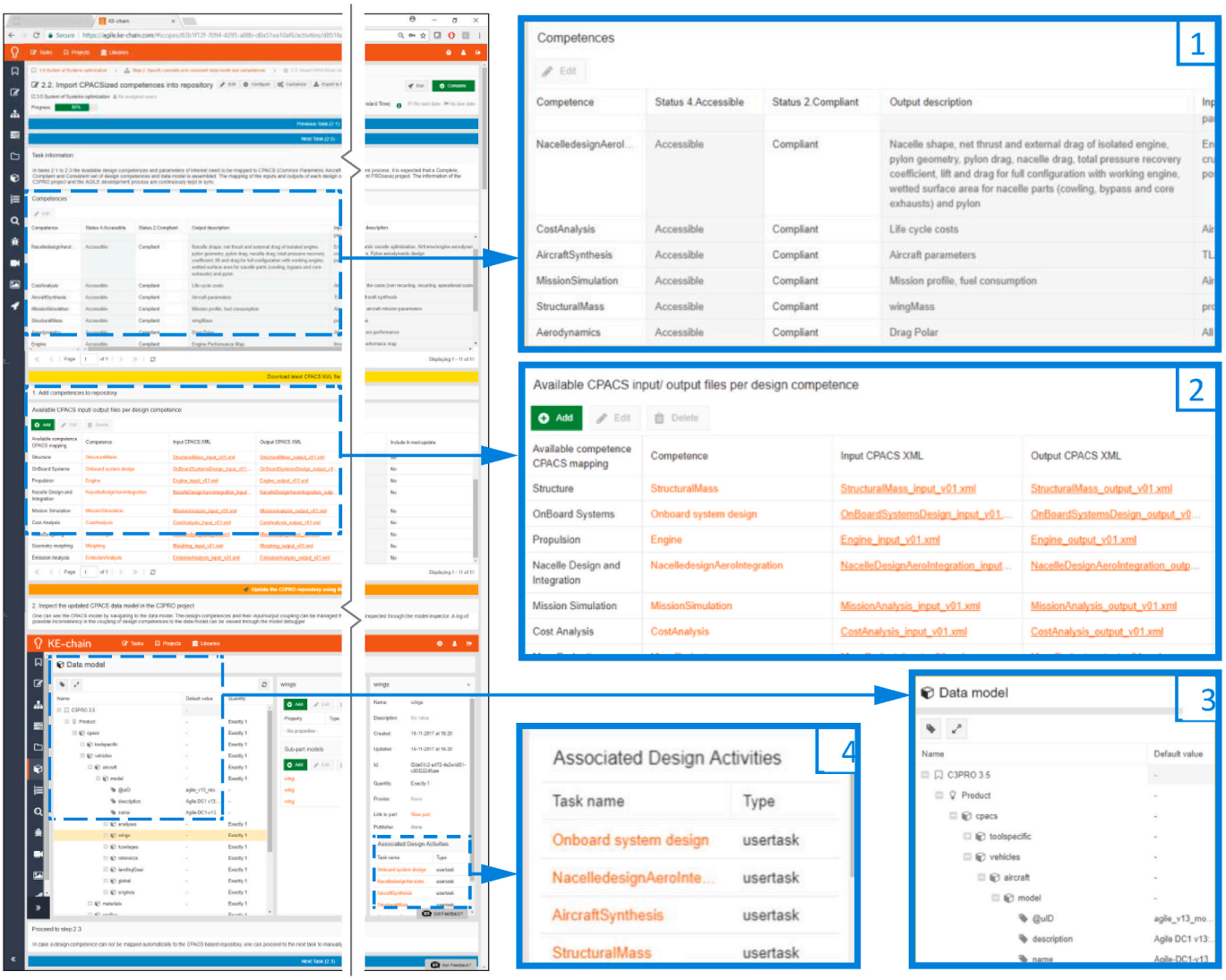


Fig. 13. Screenshot of the KE-chain web-based interface for Step II.2: (1) inspection of available design competences, (2) management of input/output CPACS or CMDOWS files, (3) live inspection of the aircraft data model, (4) inspection of coupled design competences using KE-chain on-platform utilities.

Deliverable: A CMDOWS file, with visualizations, containing the full specification of the automated design process.

Substeps (see also Fig. 11): All substeps are performed by the architect unless indicated otherwise:

Step III.1 First the repo graph is constructed in KADMOS using the CMDOWS file from step II. This is simply a translation of the XML-based CMDOWS format to the KADMOS native graph format.

Step III.2 The repo graph needs to be manipulated in order to arrive at the problem graph. The goal of this process is to get the smallest possible graph containing only those specific design competences, with their data connections, that are strictly necessary to solve the MDO problem at hand. To this purpose, KADMOS automatically removes from the repo graph all the unnecessary design competences and multiple design competences can be merged into a single service by the integrator or architect.

Step III.3 The problem graph specification is finalized by assigning certain CPACS elements a special role in the MDO problem, such as design variables, objective values and constraint values.

Step III.4 The MDO architecture of choice is finally imposed on the problem graph. In this demonstrator a DOE architecture is chosen, see Fig. 18 and 19.

The formal mathematical definition of the MDO problem being solved in this demonstrator is:determine: direct operating cost (DOC), recurring cost, average temperature response (ATR) and fuel mass based on: a latin hypercube sampling method.

with respect to: AR, Λ, S

with: $9.00 \leq \mathrm{AR} \leq 11.0$ $25.0^{\circ} \leq \Lambda \leq 31.0^{\circ}$

$75\ m \leq S \leq 110\ m$

Based on the problem graph, KADMOS automatically sorts the different design competences and includes the additional blocks, as required by the given MDO architecture, e.g., the Coordinator, DOE and Converger blocks in Fig. 18. This problem graph manipulation results in two directed graphs: the MDO data graph containing the specification of data exchanged by all the blocks in the MDO system, and the MDO process graph with the specification of the blocks execution order. Together these two graphs provide the complete automated design process definition, which is stored in a CMDOWS file and checked by the integrator.

4.3.1. *Use of KADMOS in step III of the demonstrator*

The CMDOWS file from step II contains eleven design competences (Table 2) involving around 3700 unique elements from CPACS as in- and outputs. The repo graph that is constructed by KADMOS in step III.1 is primarily useful to build visualizations with VISTOMS. Furthermore, KADMOS categorizes all the CPACS elements (e.g. inputs, outputs, couplings, collisions, etc.) based.

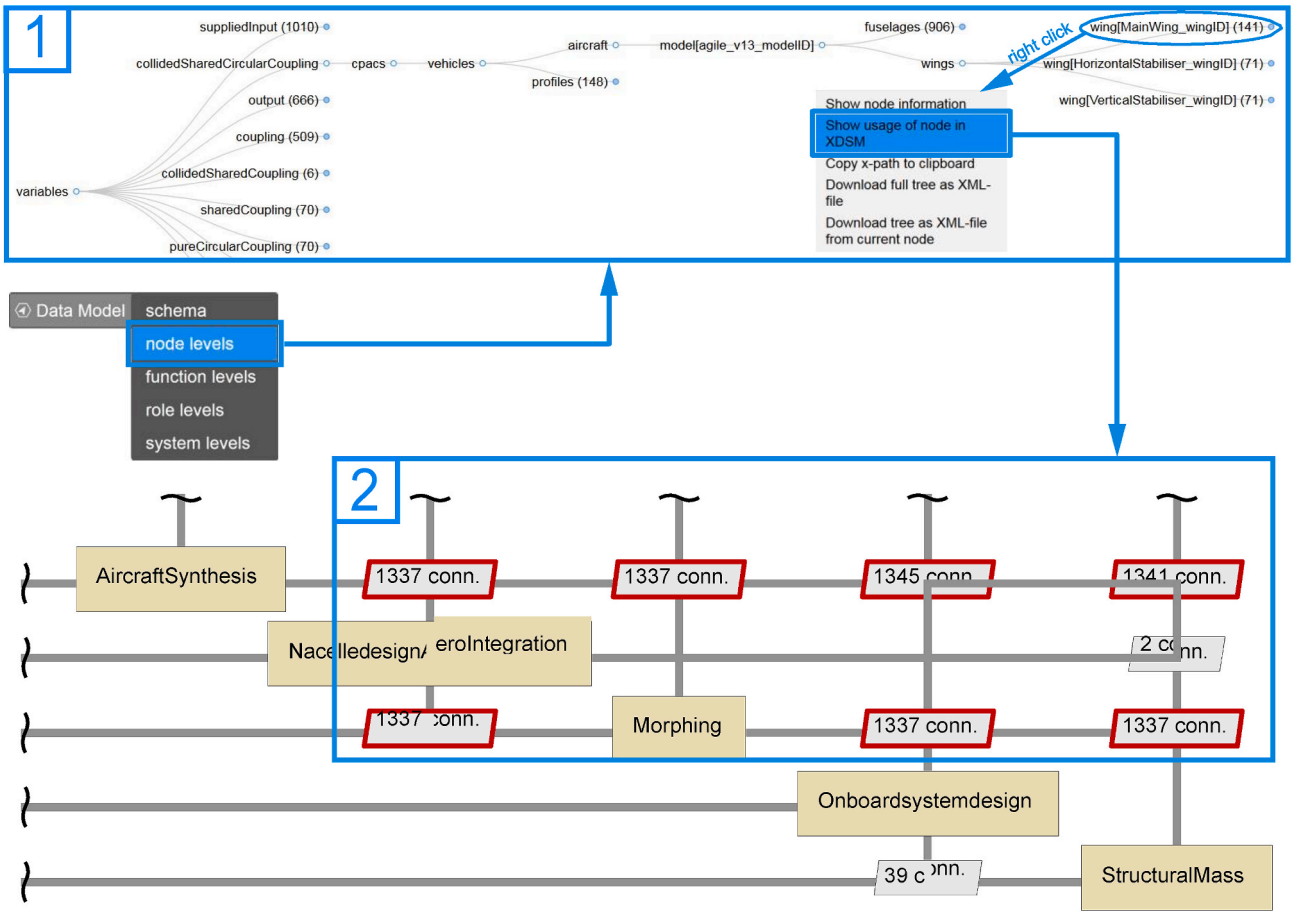


Fig. 14. Repo graph of the presented design case as VISTOMS XDSM view including data tree that two competences (AircraftSynthesis and Morphing) both modify the wing geometry. These kinds of collisions have to be resolved (or at least noticed) by the integrator in order to keep the underlying product model (CPACS file) consistent, which can be done with KADMOS in the subsequent step of the development process.

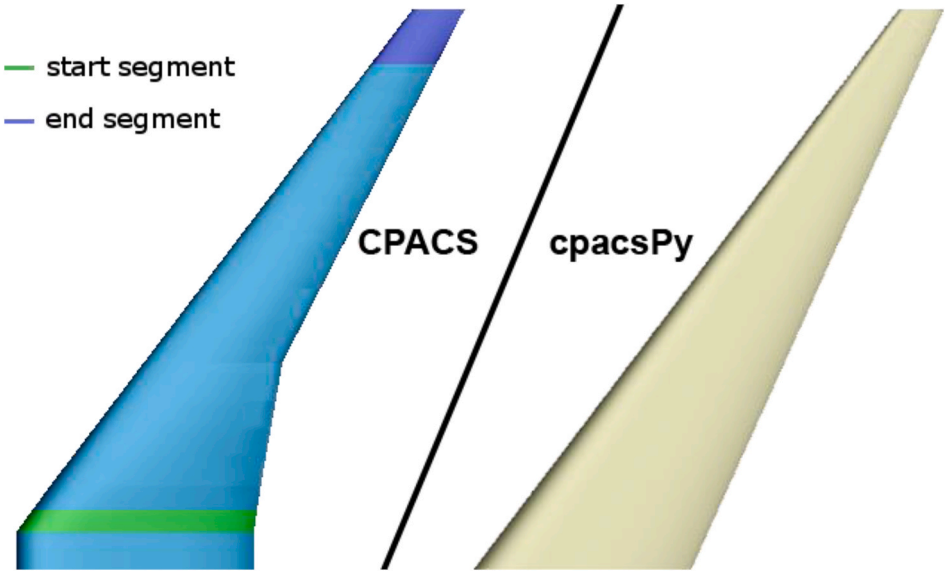


Fig. 15. The actual triple multi-segment wing defined in the CPACS file (left) with the selection of start and end wing segments indicated for the wing sweep determination and the equivalent single trapezoidal segment wing definition in cpacsPy (right) which has an equivalent planform area, mean sweep, and aspect ratio with respect to the multi-segment wing.

The results of the DOE will provide the design team with an indication of the most influential variables, based on which a surrogate model could be constructed to perform an optimization, however, these additional reconfigurations are discussed more elaborately in other AGILE design case papers and considered out of scope for the work presented here. on the amount of connections and the design team can use these categorizations for closer inspection in the VISTOMS visualization of the design compe- tence repository, as was mentioned already in Section 4.2.2. For example, in the demonstrator the CPACS data element pointing to the Maximum Take-off Mass (MTOM) is a collision, as it is both an output of the AircraftSynthesis and the MassBudget design competences. It is then up to the design team to decide which design competence is meant to provide this value. In this case the MassBudget tool was set to provide the MTOW value; however, the less accurate value produced by AircraftSynthesis could be used as initial guess for MTOM, if MassBudget is part of a convergence loop with MTOM as feedback variable.

In step III.2 the design competences are arranged according to a

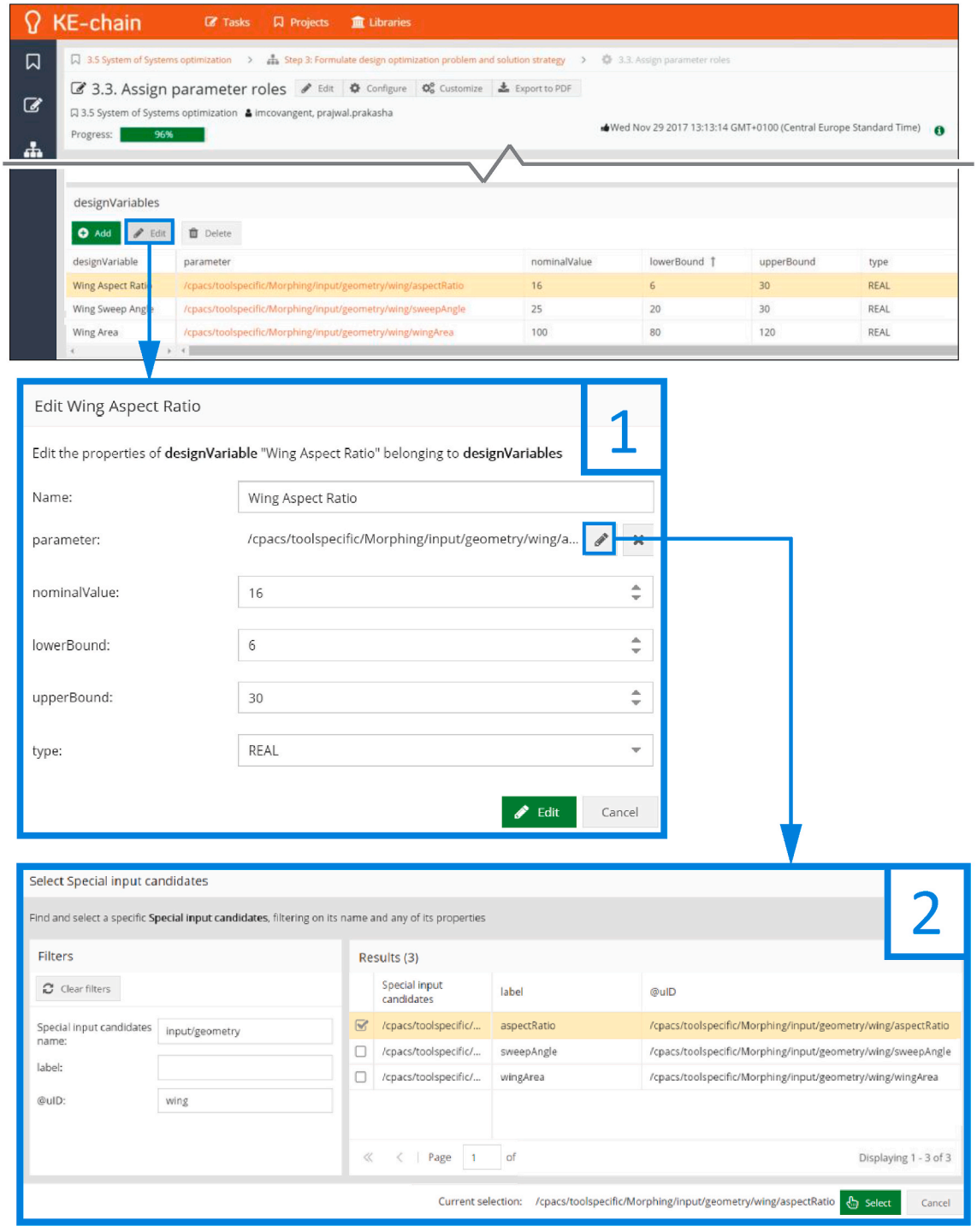


Fig. 16. KE-chain provides an interface for KADMOS to the architect in step III.3 to support in assigning special roles to CPACS elements. This example shows that the architect edits design variable *wing aspect ratio* (1) and selects its corresponding CPACS element through a search dialog (2) that filters through the entire set of 4371 elements.

convenient execution order and remaining issues on the data connections are solved. One such issue is visible in Fig. 14, where the wing geometry (1337 connections, hence 1337 CPACS elements are used to define the aircraft geometry and coupled as indicated in the figure) is output of both AircraftSynthesis and the Morphing design competences, and many design competences, including the Morphing tool itself, are using the same geometry as input. This is not a case of redundant design competences playing the same role of geometry provider. In this case, a first instance of the wing geometry should be given to the Morphing tool by the AircraftSynthesis tool. Then the Morphing tool should adjust this geometry and provide a second instance of the wing geometry, to be used by all the other tools that need geometry as input. KADMOS configures this data flow by creating instances. A first instance of the wing geometry is created by AircraftSynthesis and provided as input to Morphing. In its turn, Morphing creates a second instance of the wing geometry to be used by the following design competences. These two geometry data instances are also visible in Figs. 18 and 19 where the wing definition is included in the data connection blocks with 1337 connections. In step III.3 KADMOS assigns the special roles to selected elements, according to the MDO architecture selected by the architect. The architect is able to assign element roles through the interface in KE-chain as is shown in Fig. 16. In case of the demonstrator DOE, wing area, aspect ratio, and sweep have been assigned the role of design variables. Those are indeed the top-level wing variables used as input by the Morphing tool (see Fig. 18). All the other variables the design team wants to keep track of in the overall process, are assigned the role of 'quantities of interest'. These quantities could later become the objective and constraints of an optimization process. The quantities of interest selected for the demonstrator were: DOC and recurring cost from CostAnalysis, ATR from Emission Analysis, and fuel mass from Mission Simulation.

Finally, the DOE architecture is imposed on the problem graph in step III.4. The resulting XDSM is shown in Figs. 18 and 19. Two main blocks are added on the diagonal: the DOE regulator and a converger. The DOE regulator provides the design points to be analyzed and collects the quantities of interest of the converged design. Inside the DOE loop, every experiment provided by the DOE is converged based on the feedback variables MTOM, total lift and drag coefficients ($C_{f,x}$ and $C_{f,z}$) of the aircraft. The specification of this automated design process is the final result in step III that is stored in a CMDOWS file and visualized using VISTOMS (Figs. 18 and 19).

4.3.2. Use of VISTOMS in step III of the demonstrator

The various CMDOWS files used to store the evolving definition of the MDO system, from repo graph, to problem graph and finally the combination of MDO data and MDO process graphs, can all be loaded and visualized through the VISTOMS web-based graphical user interface, as shown in Fig. 17. In Fig. 18 the CMDOWS file containing the formulation of the demonstrator MDO system is visualized using a dynamic XDSM.

In Fig. 18, it can be observed how the competences have been organized by KADMOS in a meaningful order, compared with the unorganized repo graph from step II (Fig. 14). The process execution order is indicated in the XDSM by the thin black lines and the numbers written inside each block. The wing design variables selected for demonstrator DOE are provided to the MDO workflow by the DOE block (Fig. 18, overlay frame 1) and then translated into a full parametric CPACS geometry by the Morphing competence. Subsequently, the wing geometry is processed to the downstream competences such as Aero- dynamics and CostAnalysis. Here, it can be observed how the AircraftSynthesis competence only provides an initialized aircraft geometry, while the Morphing competence adjusts this geometry according to the given DOE inputs. The four main quantities of interest of the performed DOE can be examined in overlay frame 3, namely total recurring costs, fuel mass of the specified flight mission, ATR, and DOC.

The hierarchical tree views displayed in overlay frames 1–3 of Fig. 18 can be accessed within VISTOMS via a right mouse-click operation on the highlighted edges in the XDSM. In fact, there are multiple other inspection options implemented in VISTOMS, which cannot be discussed in detail in this paper. The interested reader is referred to Ref. [33] for further information.

In Fig. 19 a closer insight to the design competence operating inside the convergence loop is given, where the three previously mentioned convergence variables MTOM, $C_{f,x}$ and $C_{f,z}$ are displayed. The MTOM convergence is a typical iteration procedure in aircraft design, whereas the iteration of the aerodynamic coefficients is added to account for the

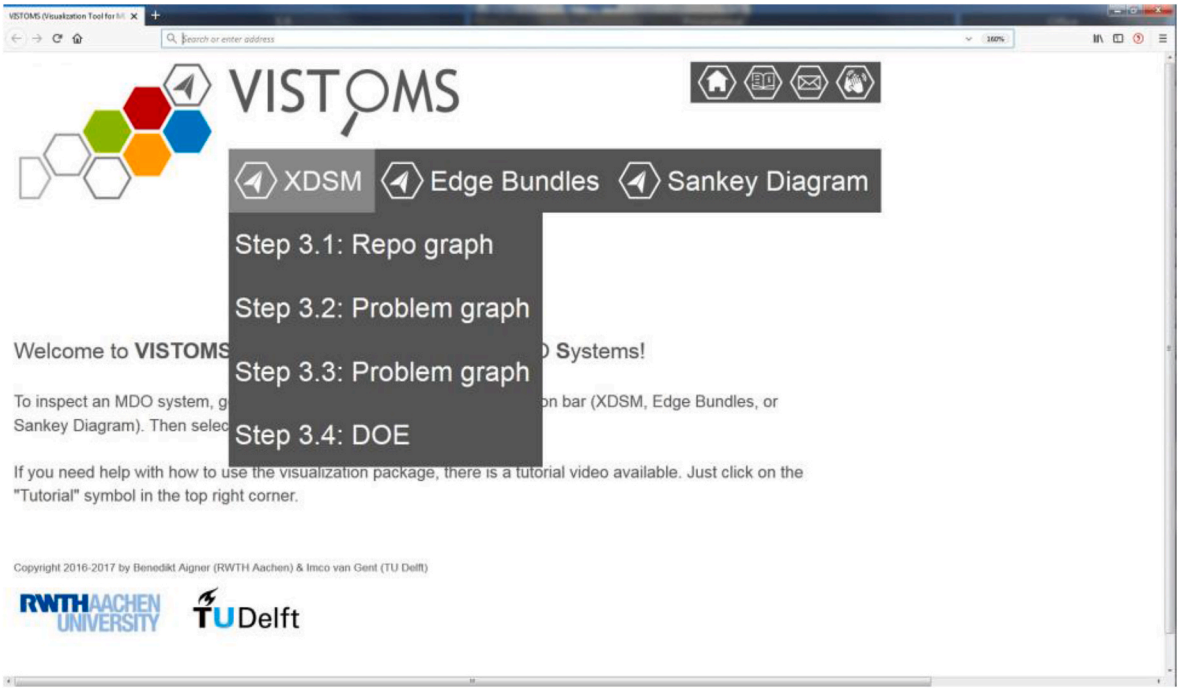


Fig. 17. Home page of the VISTOMS graphical user interface.

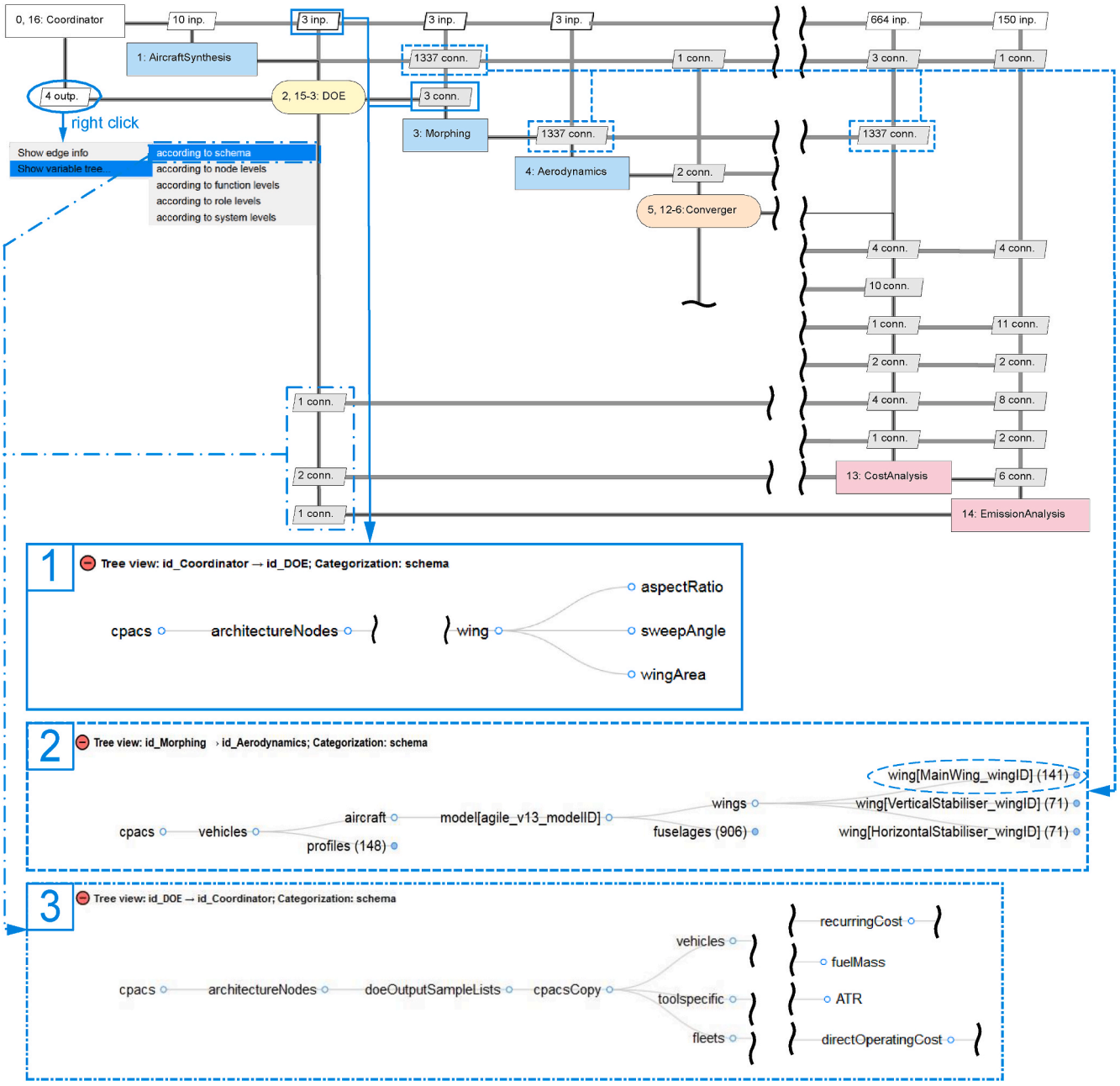


Fig. 18. Extract of the KADMOS MDO data and process graphs for the presented design case as VISTOMS XDSM view.

adjustment of these coefficients by the nacelle design and integration design competence (NacelledesignAeroIntegration).

4.4. Steps IV and V

The CMDOWS file from step III is used here to bridge the gap between formulation and execution by automatically translating the neutrally stored MDO solution strategy into an executable workflow for a PIDO platform of choice, see Fig. 20. The implementation details of Steps IV and V are outside the scope of this paper, but can be found in Refs. [13] and [25].

In a realistic design case, the found design solutions in the DOE will probably trigger an iteration which goes back to one of the earlier steps in the development process. Additional requirements might be acquired (Step I), additional tools might need to be integrated (Step II), or the collaborative workflow might need to be reconfigured using a different MDO architecture (Step III). Since all steps are integrated within the top-level development process environment KE-chain, the process that has been set up in the development process environment acts as a custom-made 'AGILE framework app' that can be used to collaboratively reconfigure the project.

4.5. ADF impact and limitations

Recently, a survey was conducted [35] to quantify the impact of the ADF presented in this paper as one of the cornerstones of the 3rd generation MDO environment. Thirty users with different roles in multiple AGILE design cases positively reviewed the framework and provided feedback on its individual components. An average time reduction of 39% to establish the first executable workflow in collaborative MDO projects was estimated by the respondents to be contributed by the ADF in comparison to previous MDO environment generations. Of the individual components, KADMOS and VISTOMS were most highly valued with time reductions of respectively 49% and 36%. KE-chain was

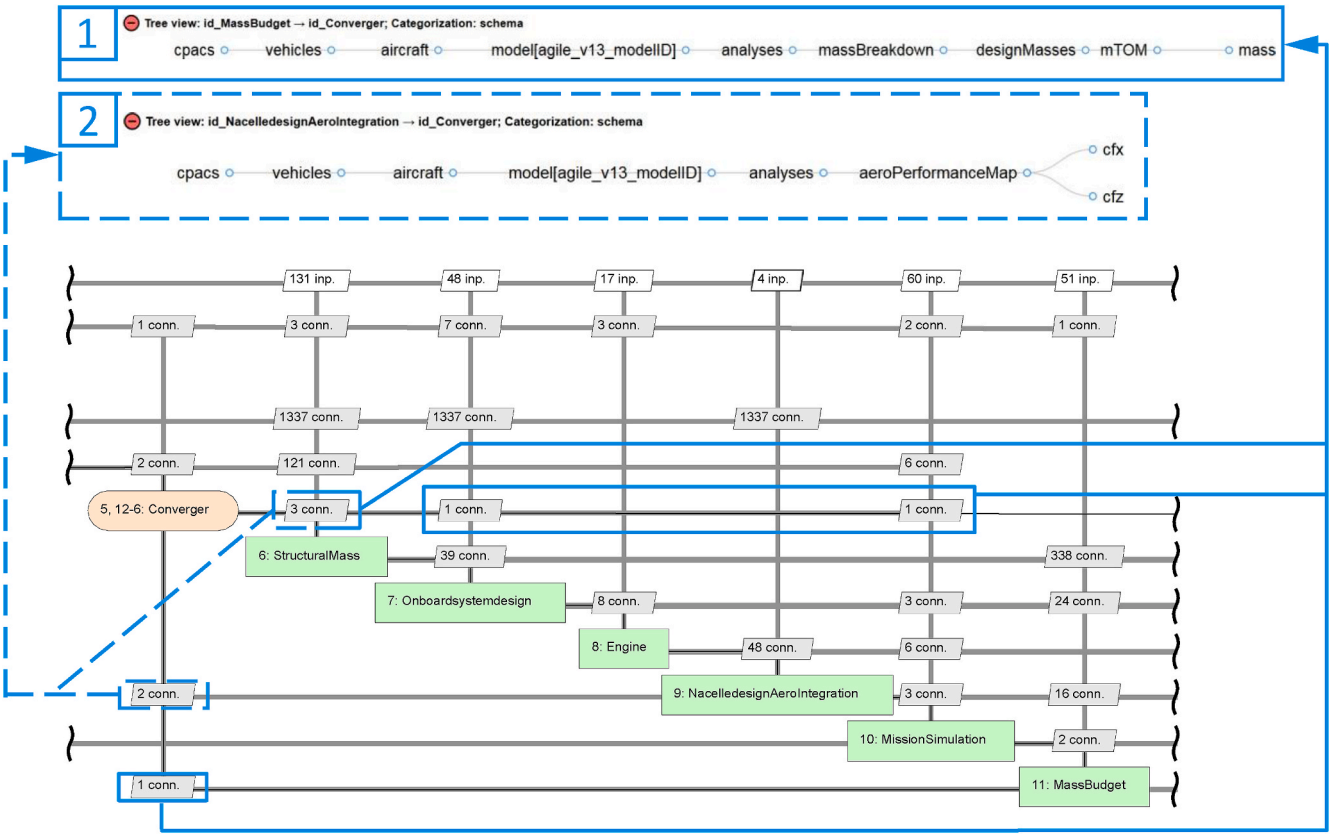


Fig. 19. VISTOMS generated XDSM showing details of the DOE system convergence loop.

estimated to have a lower impact (17%), mainly because of the low technology readiness level of the research-based KE-chain interface compared to interfaces of other commercial software packages in which most respondents work. Other components had estimated impacts between the KE-chain en VISTOMS scores.

Main limitations of the ADF that came forward in the survey were related to two main topics: use of gradients and high-fidelity tools. The use of gradients as output of the design competences has been considered outside the scope of work from the start. The reason for this is that the approach in AGILE was to use design competences as they were available at the start of the project in order to spend as little time as possible on tool development. It was found that the majority of the design competences that could be provided within the consortium did not provide any gradient information. This is actually rather representative of reality in industry, where the majority of tools (often commercial) are “black boxes”, unable to directly provide gradients. The ability to account for gradients would require extensions for several of the framework components.

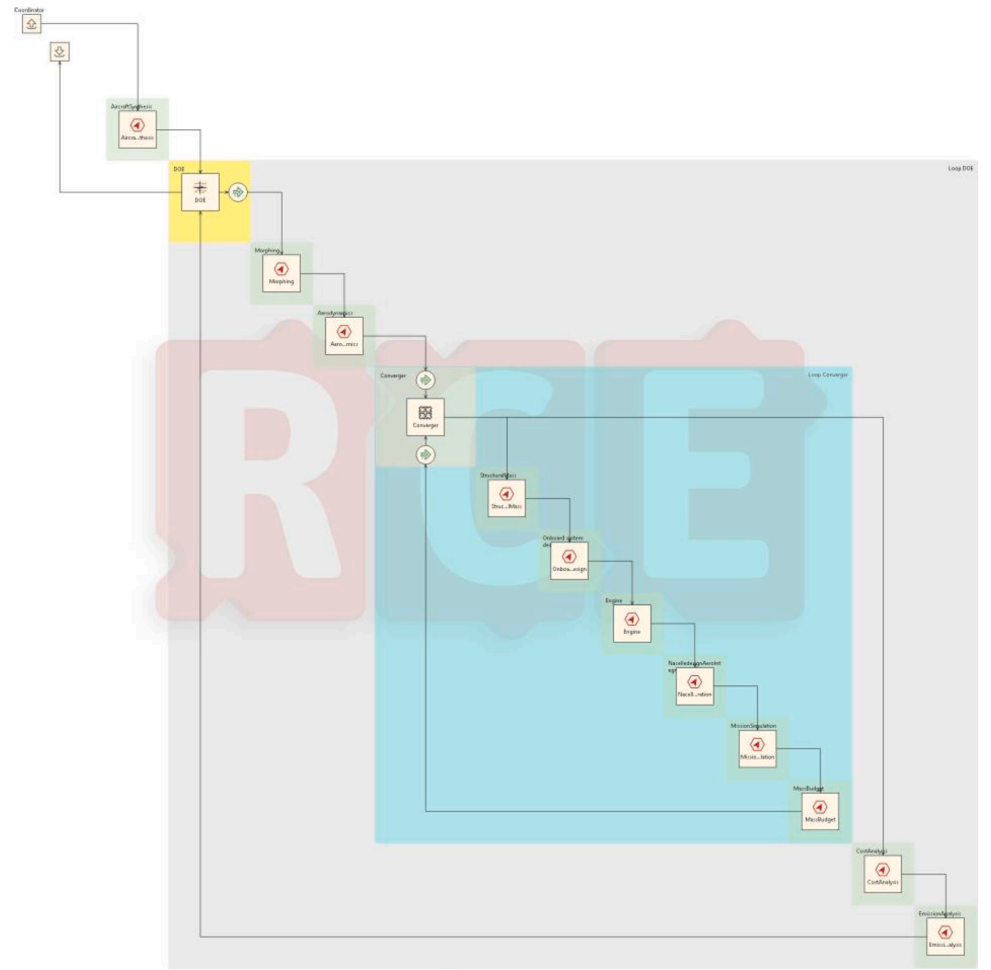
Secondly, the use of high-fidelity tools is currently also problematic, mainly because most high-fidelity tools require a very precise geometrical definition of the product under consideration. However, the employed CPACS data schema was originally developed to support the conceptual and preliminary design phase of the aircraft, therefore lacks support for a very detailed and accurate geometrical definition. Despite this limitation, high-fidelity MDO can follow the same approach and use the majority of the ADF without any large modifications, but would clearly require an extended or altogether different product schema.

5. Conclusion

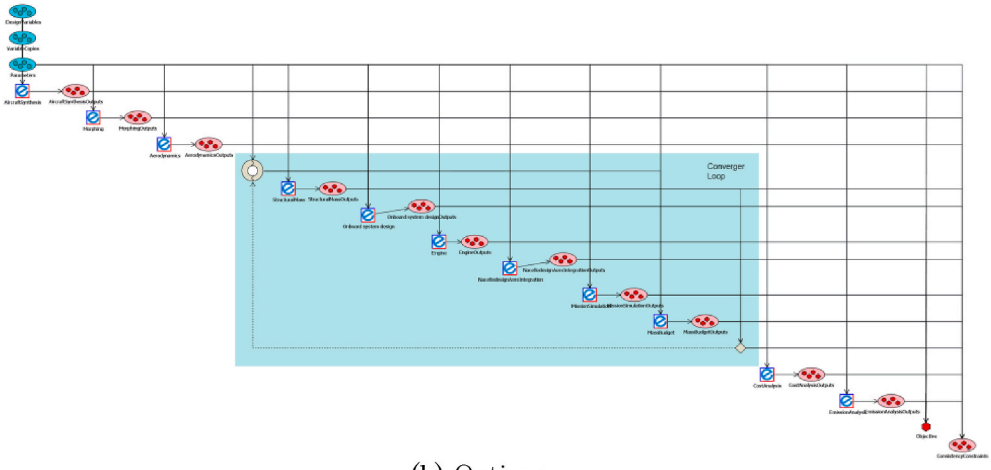
This paper has presented one of the main conceptual elements of the AGILE paradigm: the Knowledge Architecture (KA). Together with the Collaborative Architecture (CA), the KA is a cornerstone of a novel methodology to support collaborative design and optimization within large, heterogeneous and distributed teams of experts, as nowadays necessary for the development of any complex product. The focus on the knowledge-based approach to enable actual collaborative design makes the AGILE paradigm extend the boundaries of current MDO and sets the beginning of a 3rd generation MDO environment.

The AGILE paradigm is based on a five-step development process, encompassing the main stages of the formulation and execution phases of any automated design system. The KA is specifically supporting the formulation phase by means of a hierarchical structure, based on four main layers, namely the development process layer, the automated design layer, the design competences layer, and the data schemas layer which transverses the first three. Through their interfaces, these layers offer a comprehensive framework that brings rationale and cohesion in the complex endeavour of formulating a multidisciplinary design system, where different actors are involved, including discipline specialists, system architects and integrators.

The first layer provides an interactive user friendly environment to integrate and streamline design aspects that are fundamentally human-based with others that are based on design automation. The automated design solutions are formulated in the second layer, making use of the design competences available in the third layer. The fourth layer enables data exchange between the various design competences and the means to store the formalization of the complete MDO system, as ready to be translated into an executable computational workflow (by means of the CA). In its whole, the proposed KA addresses the need for formulating complex design automation solutions, while granting designers full oversight of the design system being developed and leaving competence specialists in full control of their own tools, to guarantee their correct utilization within the distributed multidisciplinary workflow. The Knowledge Architecture (as well as the whole AGILE paradigm) is proposed as a generic methodology to be applied to any engineering domain that deals with the development of complex products (e.g. aerospace,



(a) RCE



(b) Optimus

Fig. 20. Screenshots of executable workflow instances for two different PIDO platforms. Both workflows are based on the same CMDOWS file that were provided by KADMOS in Step III. [46].

automotive and wind energy), where automation is necessary to enable trade-off studies, design space explorations, up to full multidisciplinary optimizations in a collaborative environment.

One of the innovative aspects of the KA is provided by the introduction of the automated design layer, where a neutral formulation of the MDO system is generated first, before moving to the automated integration of the computational system in a PIDO platform. Without this conceptual layer, IT-wise one would still be able to create an automated design process, but would quickly loose oversight of process intricacies and the flexibility to adapt and reconfigure the system. Thus, without this intermediate formal specification step, any automated design process has the risk of becoming a large, complex black box that none of the involved design team members can grasp, inspect, validate and adjust. The definition of this intermediate layer improves the agility of the design team in three ways:

- reduce the setup time for large and complex automated design processes,
- enable systematic inspection and debugging of automated design processes using a visualization tool,
- allow the manipulation of the automated design formulation so that the creation and reconfiguration of optimization strategies can be automated.

Yet, only a partial increase in agility could be gained without the other main innovation: the workflow schema CMDOWS to store and communicate MDO formulations. The possibility to translate such neutral formulation into a fully executable workflow, effectively streamlines the formulation and execution phase of any MDO process. Dramatic reductions in lead time can be achieved, as well as more flexibility in the selection of the PIDO system.

Within the AGILE project, the proposed methodology has been specifically applied to the aircraft design domain, resulting in the so-called AGILE development framework (ADF). To this purpose a number of software applications and two data schemas have been developed, either from scratch (KADMOS, VIS- TOMS and CMDOWS) or by extending existing solutions (KE-chain, CPACS, cpacsPy). Each one of these applications contributed to the realization of the four KA layers. The ADF functionality has been demonstrated by applying the development process to an MDO case of representative complexity for the aeronautic industry, concerning the design space exploration of a conventional passenger aircraft. To this purpose a repository of eleven CPACS-compliant design competences has been used, contributed by five partners, operating in five different locations in Germany, Italy and Russia. The ADF proved able to dramatically speed up the setup of the distributed multidisciplinary system, by reducing the effort required to inspect and eventually resolve eventual inconsistencies in the data flow, and by enabling a quick and fully automated formulation (and eventually reconfiguration) of the design process.

The KE-chain platform provided an intuitive and flexible environment to let customers, architects and discipline specialists collaborate during the first three steps of the developed approach. KADMOS, under the hood, applied its efficient knowledge- and graph-based approach to fully automate the formulation of the MDO system, starting from the set of design competences and design process requirements specified by the system architect in KE-chain. VISTOMS enabled the discipline specialists, integrator and the architect to continuously inspect the system, throughout all the development phases, by means of dynamic and scalable visualizations, effective despite the sheer size of data to be displayed. The initial investment for making all the design competences CPACS-compliant paid off by practically enabling a plug & play approach for all the synthesis and analysis tools. The development of the cpacsPy library was however necessary to guarantee a coherent interpretation of the CPACS data by the various discipline specialists. CMDOWS provided the medium to store the MDO system definition during its evolution from repository of design competences to complete MDO solution strategy. Moreover, it played the role of hub for all the other KA applications and provided the actual bridge between the formulation and the execution phase of the MDO system.

A survey was conducted to establish the impact and limitations of the current framework based on the design cases performed in the AGILE project. The survey results, containing responses from thirty users of the ADF, indicated an estimated 39% time reduction in the setup and operation phases compared to earlier MDO environment generations. The two main limitations of the framework are its lack of support for using gradients in the design competences and the problematic use of high-fidelity tools. The latter is related to the use of the current CPACS that was developed for conceptual and preliminary design. These limitations are at the top of the priority list for the future development plans of the framework.

Declaration of competing interest

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

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 no. 636202. The authors are grateful to the partners of the AGILE consortium for their contribution and feedback.

References

[1] J. Agte, O. De Weck, J. Sobieszczanski-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] 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.
[3] S. Shahpar, Challenges to overcome for routine usage of automatic optimisation in the propulsion industry, Aeronautical Journal 115 (1172) (2011) 615.
[4] P.D. Ciampa, B. Nagel, Towards the 3rd generation MDO collaboration environment, in: 30th Congress of the International Council of the Aeronautical Sciences, 2016.
[5] I. Kroo, V. Manning, Collaborative optimization: status and directions, in: 8th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, 2000.
[6] AGILE consortium. https://www.agile-project.eu, 2017.
[7] P.D. Ciampa, B. Nagel, The AGILE Paradigm: the next generation of collaborative MDO, Prog. Aero. Sci. (2018). In this issue.
[8] 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.
[9] A. Dener, P. Meng, J. Hicken, G.J. Kennedy, J. Hwang, J. Gray, Kona: a parallel optimization library for engineering-design problems, in: 57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 2016, p. 1422.
[10] N. Bartoli, T. Lefebvre, S. Dubreuil, R. Olivanti, N. Bons, J. Martins, M. Bouhlel, J. Morlier, An adaptive optimization strategy based on mixture of experts for wing aerodynamic design optimization, in: 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2017, p. 4433.
[11] 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.
[12] E. Heydari Beni, B. Lagaisse, W. Joosen, December, Adaptive and reflective middleware for the cloudification of simulation & optimization workflows, in: ARM 2017 - 16th Workshop on Adaptive and Reflective Middleware, 2017.
[13] E. Moerland, E.H. Baalbergen, others, Collaborative architecture supporting the next generation of MDO in the AGILE paradigm, Prog. Aero. Sci. (2021). In this issue.
[14] J. Berends, M. Van Tooren, E. Schut, Design and implementation of a new generation multi-agent task environment framework, in: - (Ed.), 49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 2008.
[15] S. van der Elst, E. Moerland, J. Lugtenburg, L.M. Hootsman, S. Deinert, M. Motzer, C. Fernandez, D2.3: industrial service-oriented process methodology, Tech. rep., IDEaliSM project (2017).
[16] 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.
[17] 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, 2012 (Brisbane).
[18] 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.
[19] S.X. Ying, Report from ICAS workshop on complex systems integration in aeronautics, 30th Congress of the International Council of the Aeronautical Sciences (2016).
[20] J. Jepsen, P.D. Ciampa, B. Nagel, Avoiding inconsistencies between data models in collaborative aircraft design processes, in: 65. Deutscher Luft- und Raumfahrtkongress, 2016.
[21] R.S. Sellar, S.M. Batill, J.E. Renaud, Response surface based, concurrent subspace optimization for multidisciplinary system design, 1996. AIAA paper 714, 1996.
[22] R. Lano, The N^2 chart, Tech. rep., TRW. (1977).

[23] T.C. Wagner, P.Y. Papalambros, General Framework for Decomposition Analysis in Optimal Design, vol. 65, ASME DES ENG DIV PUBL DE., ASME, NEW YORK, NY (USA), 1993, pp. 315–325.
[24] D.V. Steward, The design structure system: a method for managing the design of complex systems, Engineering Management, IEEE Transactions on (3) (1981) 71–74.
[25] I. van Gent, G. La Rocca, Formulation and integration of MDAO systems for collaborative design: A graph-based methodological approach, Aerospace Science and Technology 90 (2019) 410–433. https://doi.org/10.1016/j.ast.2019.04.039.
[26] 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.
[27] R. Lafage, XDSMjs Package - XDSM Diagram Generator Written in Javascript, 2016. https://github.com/OneraHub/XDSMjs.
[28] H.R. Sankey, The thermal efficiency of steam engines, Minutes of the Proceedings of the Institution of Civil Engineers (1896) 182–212.
[29] M. Schmidt, Der Einsatz von Sankey-Diagrammen im Stoffstrommanage- ment. Beiträge der Hochschule Pforzheim, Hochsch. Pforzheim. (2006). https://books.google.de/books?id=zhBbMgAACAAJ.
[30] N. Atkinson, bihisankey.js Package - BiDirectional Hierarchical Sankey Diagram, 2015. https://github.com/Neilos/bihisankey.
[31] D. Holten, Hierarchical edge bundles: visualization of adjacency relations in hierarchical data, IEEE Trans. Visual. Comput. Graph. 12 (5) (2006) 741–748.
[32] M. Bostock, D3.js website. https://d3js.org/, 2015.
[33] 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 Aeronautical Journal (2018).
[34] E. Margolis, S. Laurence, Concepts: Core Readings, MIT PR, 1999.
[35] I. van Gent, B. Aigner, 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.
[36] I. van Gent, G. La Rocca, M.F.M. Hoogreef, CMDOWS: a proposed new standard to store and exchange MDO systems, CEAS Aeronautical Journal (2018) 607–627.
[37] I. van Gent, R. Lombardi, G. La Rocca, R. d'Ippolito, A Fully Automated Chain from MDAO Problem formulation to Workflow Execution, EUROGEN 2017 (2017).