Implementation of a collaborative environment to support the integration of 3rd generation MDO frameworks

B. Beijer

TU Delft

KE-works

AGILE
Aircraft 3RD Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts

Implementation of a collaborative environment to support the integration of 3^{rd} generation MDO frameworks

by

B. Beijer

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday July 5, 2017 at 13:30 PM

Student number:	1518836	
Thesis registration number:	136#17#MT#FPP	
Project duration:	August 1, 2016 – June 15, 2017	
Thesis committee:	Prof. ir. G. La Rocca,	TU Delft, Supervisor
	Ir. J. Berends,	KE-works, Supervisor
	Prof. dr. ir. P. Colonna	TU Delft, Chair Propulsion & Power
	Dr. ir. J. Guo	TU Delft, Space Systems Engineering

TUDelft

Acknowledgements

Hereby I would like to thank my main supervisors Gianfranco La Rocca and Jochem Berends for their inspiration during my research and taking the time to check my work. Next I would like to thank my other thesis committee members, Jian Guo and Piero Colonna, for making time available for being part of my defense. Thanks to Joost Schut and Stefan van der Elst for providing feedback and improve the value of my work. Thank you Kevin van Hoogdalem for providing me with a thesis research topic and being my first supervisor. Thanks to my colleagues at KE-works for the fun times and all the help. I really enjoyed the occasional table football games!

Thanks to Imco Van Gent and Zaoxu Zhu for being available for my green light review and providing feedback for my research. Next I would like to thank fellow (and recently graduated!) MSc students Menco Schuurman and Andreas Makus for our discussions and information exchange regarding the research topic. Thanks Pier Davide Ciampa, Thierry Levebvre, Benedict Aigner, Ton van der Laan and the AGILE consortium for providing feedback on the collaborative environment developed as the result of this research.

Finally I would like to thank my friends for supporting me throughout my thesis and providing me with very helpful feedback. And of course my girlfriend Jeannine, for her patience and support, especially during the stressful times.

The research presented in this paper has been conducted as part of European research project AGILE. This project has received funding from the European Union's Horizon 2020 research and innovation framework program under grant agreement No 636202.

B. Beijer
Delft, May 2017

Summary

Increasing complexity of engineering problems and corresponding modeling and analysis activities required to explore novel aircraft designs lead to interesting challenges in the domain of Multidisciplinary Design Optimization (MDO). As MDO frameworks, capable of performing overall aircraft design studies become too complex to be comprehended by a single team of experts, there is an increasing need to develop distributed analysis frameworks in which both tools and experts are integrated in a single network. This collaborative design in distributed teams of engineers and tools is characterized as the 3^{rd} generation of MDO.

This research is conducted as part of European research project AGILE. AGILE addresses the challenges implied with the three main phases involved in design and optimization processes: the setup, operation and solution phases. This research focuses mainly on the challenges and solutions existing in the setup phase. The aim of AGILE is to support MDO of realistic overall aircraft design tasks involving novel aircraft configurations using distributed frameworks in a multi-organization framework. During this research it was found that the distribution of the different actors such as architects, integrators, discipline specialists, collaboration engineers and the customer poses great challenges on the ability to control and collaborate on the setup and operation of these 3^{rd} generation MDO frameworks.

In order to enable better collaboration of all actors involved in the setup and operation phases of 3^{rd} generation MDO frameworks, the main objective of this thesis research is defined as follows: *Support the setup of 3^{rd} generation MDO frameworks by developing a collaborative environment in which the activities leading to an automated design workflow are integrated in the business process layer.*

A first step in the implementation of this collaborative environment is formalization of the main steps involved in the development of an MDO framework. During this research five steps were identified, which characterize the so-called MDO framework development process. These five steps can be seen in figure 1. This MDO framework development process is an integral part of any type of complex product development process in which Multidisciplinary Design Analysis and Optimization (MDAO) strategies are implied, such as the conceptual design of aircraft or automobiles.

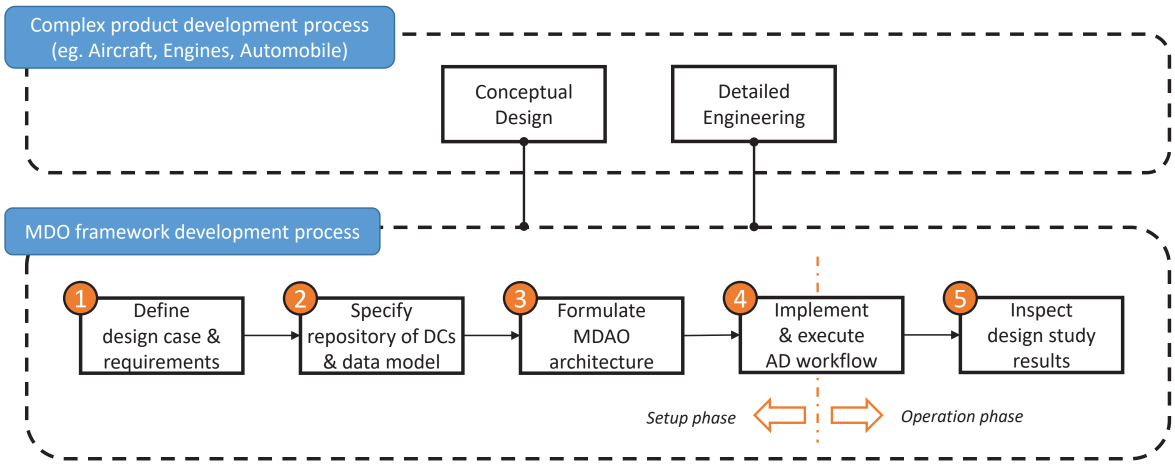

Complex product development process (eg. Aircraft, Engines, Automobile)
Conceptual Design
Detailed Engineering
MDO framework development process
1
Define design case & requirements
2
Specify repository of DCs & data model
3
Formulate MDAO architecture
4
Implement & execute AD workflow
5
Inspect design study results
Setup phase
Operation phase

Figure 1: MDO framework development process

In general, an MDO framework is composed of three layers, depicted in figure 2. These layers, from top to bottom are the Product Development (PD) layer, the Automated Design (AD) layer and Design Competence (DC) layer. The DC layer can be defined for simplicity as a collection of disciplinary tools. The AD layer contains executable simulation workflows, and the tools required to integrate or formulate those. Finally, the PD layer contains manual and automated tasks, which together define an executable business process. All layers have interfaces, which are required to exert control, or transfer information between layers. This thesis research focuses mainly on the PD layer and its interfaces with the lower layers in the knowledge architecture.

The main functions of the PD layer can be summarized as follows:

- Support the user in setup of a design problem and related simulation workflows
- Support the user during the execution of the required automated design process with manual interfaces to that process and by retrieving information on the simulation process performance and the design results.

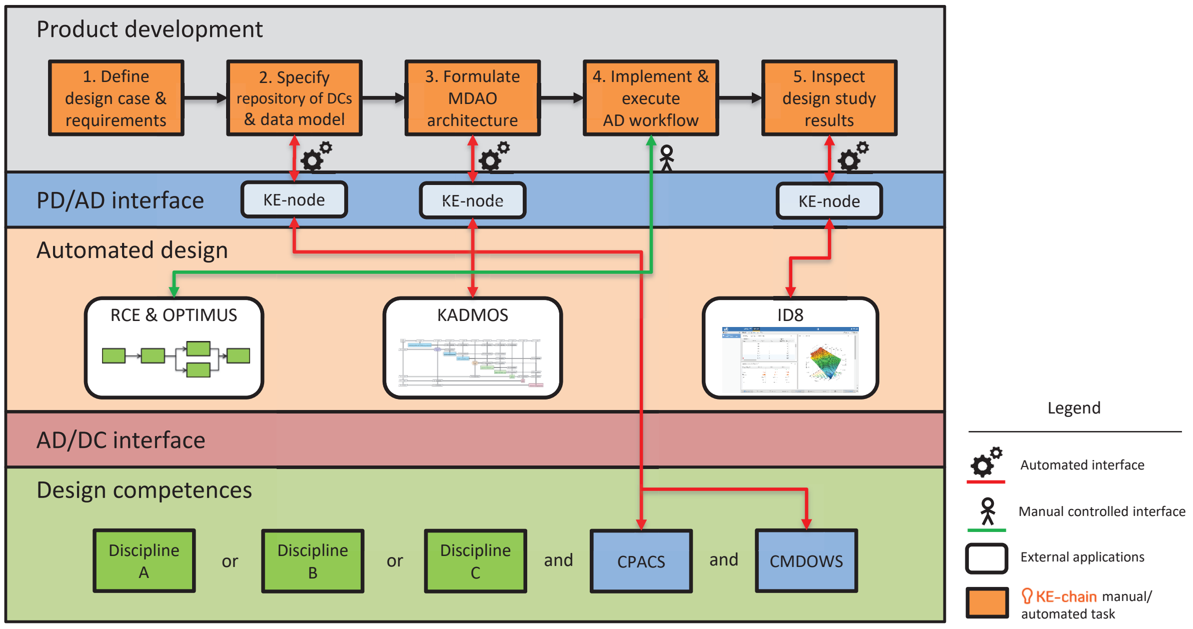

Product development
1. Define design case & requirements
2. Specify repository of DCs & data model
3. Formulate MDAO architecture
4. Implement & execute AD workflow
5. Inspect design study results
PD/AD interface
KE-node
KE-node
KE-node
Automated design
RCE & OPTIMUS
KADMOS
ID8
AD/DC interface
Design competences
Discipline A
or
Discipline B
or
Discipline C
and
CPACS
and
CMDOWS
Legend
Automated interface
Manual controlled interface
External applications
KE-chain manual/ automated task

Figure 2: Overview of the interfaces and components in the integrated MDO framework

During this thesis research, the author has developed a collaborative environment which extends KE-chain. KE-chain is a web-based information and Workflow Management (WFM) system developed by KE-works. KE-chain has been extended to support the required PD layer functionality and to enable the integration of interfaces between the PD layer and AD layer. These extensions can be summarized as: the development of methods to tailor the customized layout of the different tasks in the business process, a model verification module, a central knowledge library and the development of external scripts to support the integration of external tools which are in sync with a the implemented collaborative environment. These developed components are explained in more detail in the following sections.

The developed methods to extend task customization enable communication of information in a more understandable way to the end-users and aims to improve the flow of the business process. The developed model verification module is used in step two of the MDO framework development process. In this step the architect, together with the discipline specialist assemble a repository of DCs and data model which will be used in the formalization of an executable MDO workflow. In this repository, the input properties and output properties need to be mapped on a common data model. This common data model is defined according to CPACS (Common Parametric Aircraft Configuration Schema). CPACS describes the characteristics and attributes of aircraft, engines, climate impact, fleets and mission in a structured, hierarchical manner. Using a single CPACS-derived product model to couple all DCs ensures all DCs are able to communicate in an executable workflow as ambiguity in the exchange of information is removed through the use of a single modeling standard. The model verification module checks whether data flow is possible, if all properties defined as input have an origin and that input data has only a single origin.

The author has developed methods and services to enable a reuse of modeling knowledge through a central knowledge library which is accessible through KE-chain and through the importing of standardized documents using XML knowledge technologies. Both methods are used to instantiate product models and process models. This product model is a collection of parts and properties organized in an intuitive and maintainable hierarchical structure. This product model forms the blue-print of the final product which is to be developed. The process model is a collection of tasks or activities, which contain the functions or methods which need

to be executed to design the final product. Within the AGILE project, process modeling language is stored according to CMDOWS (Common Multidisciplinary Design and Optimization Workflow Schema). The developed methods described and implemented during this research enable the storing, manipulation and reuse of product and process modeling knowledge using CPACS and CMDOWS directly from the implemented collaborative environment.

The implemented collaborative environment is equipped with the methods described earlier and with developed interfaces to enable the formalization of MDAO architectures using KADMOS (Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System). This collaborative environment is tested and verified through the formalization of various small MDO workflows. A full-scale test has been performed during a workshop organized at the AGILE M21 meeting held on the 4th-5th-6th of April in Delft. During this workshop a total of 43 users, among which various actors such as integrators, architects, collaboration engineers, customers and discipline specialists, worked in distributed groups on the setup of an MDO framework to perform an aerodynamic optimization of an aircraft main wing.

From the workshop session and interviews with several experts it was concluded that the implemented collaborative environment was able to support the setup of complex MDO problems. The traditionally very complex and highly manual tasks of defining the problem, finding the right tools to solve the problem and connecting them with each other in the right way was made a lot easier for the various actors. The accessibility of information and integration of key components used during the formalization of MDO frameworks currently applied to the AGILE project such as CPACS, CMDOWS and KADMOS provide a powerful environment which can be used in any type design and optimization process. This becomes evident from the current number of useages in upcoming design cases within AGILE in which the collaborative environment is used, involving the conceptual design of novel aircraft configurations: open-rotor aircraft, strut-braced aircraft box-wing aircraft, Blended-Wing-Body (BWB) aircraft and Unmanned Aerial Vehicles (UAVs).

Although the developed collaborative environment has proven to enhance the operability, maintainability and support during the setup of 3^{rd} generation MDO frameworks, some limitations were acknowledged by the author. The developed methods and interfaces between the PD and AD layer are relatively slow during the setup of medium to large MDO problems (700+ properties). The main reason for this lies in the architectural backbone of the underlying KE-chain application used during the implementation. Hence a lot of performance related issues, which are currently out-of-scope for this thesis research, have been placed on the KE-chain development road-map. The author also recommends an extension of automated interfaces between the collaborative environment and software solutions which are able to run simulation workflows. Finally improvements are advised on the communication of under-the-hood automated design activities, which can be considered as a big open challenge. Improving the performance of the overall development process through integration of automated design activities must not impede with the user-friendless and sense of control exposed to the end-users.

Contents

List of Figures

List of Abbreviations

AD	Automated Design
AGILE	Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts
API	Application Programming Interface
BWB	Blended-Wing-Body
C3PRO	Complete Consistent & Compliant PROcess
CAD	Computer Aided Design
CLI	Command-Line Interface
CM	Capability Module
CMDOWS	Common Multidisciplinary Design and Optimization Workflow Schema
CPACS	Common Parametric Aircraft Configuration Schema
DC	Design Competence
DEE	Design Engineering Engine
DLR	German Aerospace Center
DSM	Design Structure Matrix
FFD	Functional Flow Diagram
GUI	Graphical User Interface
IDF	Individual Design Feasible
IQ	Information Quality
JSON	JavaScript Object Notation
KBE	Knowledge Based Engineering
KDP	Key Design parameter
KE	Knowledge Engineering
KPI	Key Performance Indicators
MDA	Multidisciplinary Design Analysis
MDAO	Multidisciplinary Design Analysis and Optimization
MDF	Multidisciplinary Design Feasible
MDO	Multidisciplinary Design Optimization
MMG	Multi-Model Generator
OAD	Overall Aircraft Design

PD	Product Development
PDP	Product Development Process
PIDO	Process Integration and Design Optimization
PIM	Product Information Model
RCG	Repository Connectivity Graph
REST	REpresentational State Transfer
SE	Systems Engineering
SIM	Service Integration Module
SWF	Simulation Workflow
UAV	Unmanned Aerial Vehicle
UML	Unified Modeling Language
WFM	Workflow Management
WIM	Workflow Information Model
WP	Work package
XDSM	Extended Design Structure Matrix
XML	Extensible Markup Language

1 Introduction

1.1. Future of Aerospace engineering

Over the past few decades there is a growing demand for environmentally friendly and more efficient aircraft capable of transporting a large number of passengers over long ranges at reduced direct operating costs [35]. However, as the conventional tube and wing design has almost reached the limit of its potential there is a renewed interest in unconventional aircraft configurations such as the Blended Wing Body (BWB) concept and the box-wing aircraft displayed in figure 1.1 [20]. New aircraft configuration deviates from the conventional distinction of aircraft components such as wings, fuselages, engines and tail as there is a lot more integration and interaction of components. This leads to interesting challenges and opportunities for Multidisciplinary Design and Optimization (MDO) to support Overall Aircraft Design (OAD) [44].

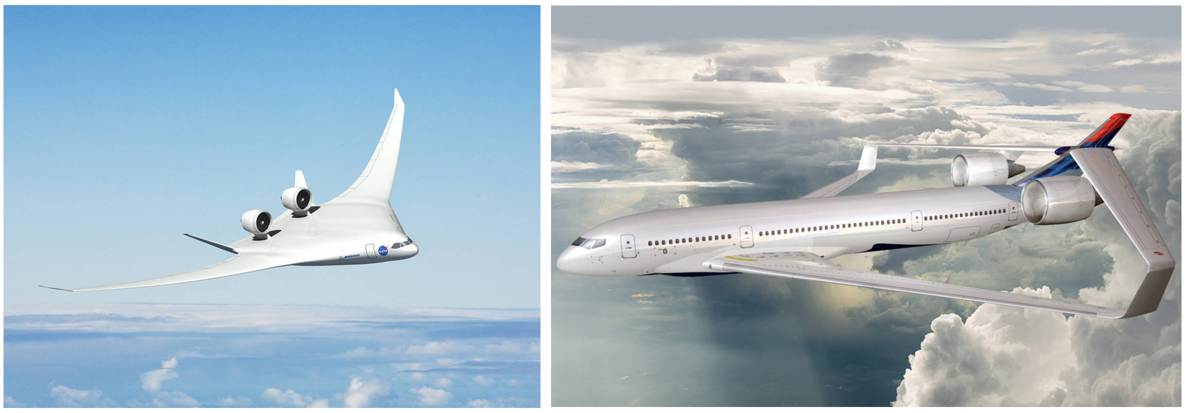

(a) Blended wing body concept (b) Box-wing concept

Figure 1.1: Illustrations of novel aircraft configurations (source: NASA)

Aircraft design is a multidisciplinary process due to the numerous domains, such as aerodynamics, structural analysis and flight mechanics, that need to be considered in an overall design process [47]. Integrating all domains in a single design process is a challenging task due to the complexity imposed by the information intensity, computing intensity and the amount of interwoven elements involved [4]. In order to cope with these challenging design processes MDO techniques are increasingly adopted in the conceptual design phase. However their use in novel aircraft development is still limited [23].

Due to this complexity, an MDO process can no longer be comprehended with a single team of specialists. This leads to challenges in processing the large amount of information gathered during the setup, operation and execution phases of an MDO process. The research presented in this report aims at improving the collaboration and control of the various actors such as architects and discipline specialists involved in the setup phase of the MDO framework to reduce project lead-time. The topic of MDO frameworks is explained in more detail in the next section.

1.2. Multidisciplinary Design Optimization frameworks

Increased computational capabilities due to modern computing power result in increasingly complex analysis tools. In order to operate these analysis tools, cross-organizational disciplinary specialists are required during the integration in any MDO framework. The purpose of a framework is to provide support during the development and execution of the MDO process [31]. This distribution of experts and tools requires a collaborative approach to support MDO in the conceptual design phase of novel aircraft design. The shift from locally implemented simulation workflows to support MDO to collaborative and distributed MDO is referred to as the shift from the 2^{nd} to the 3^{rd} generation of MDO [1, 8, 23]. The three generations of MDO which can be distinguished are displayed in figure 1.2. These generations are explained in more datail in the coming sections.

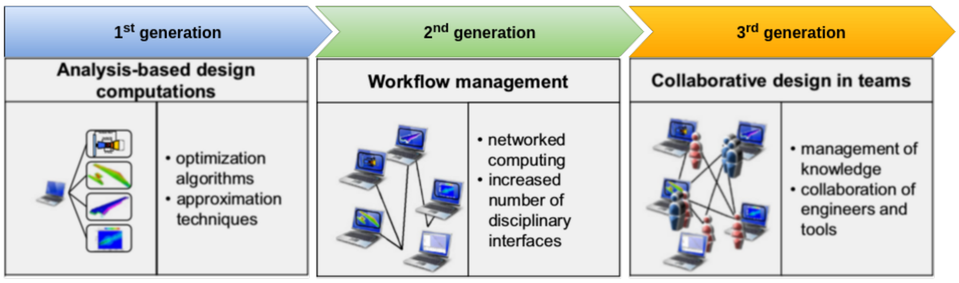

1st generation
Analysis-based design computations
• optimization algorithms
• approximation techniques
2nd generation
Workflow management
• networked computing
• increased number of disciplinary interfaces
3rd generation
Collaborative design in teams
• management of knowledge
• collaboration of engineers and tools

Figure 1.2: Evolving generations in MDO [23]

First generation

As can be seen in figure 1.2, the 1^{st} generation of MDO is characterized by an application in which disciplinary tools are tightly coupled in a monolithic system. A monolithic system in software engineering is a system in which all components and functions are interwoven, instead of being contained in architecturally separate components [36]. In a 1^{st} generation MDO framework a design process is deployed via direct interfaces between multiple design tools in an environment where are analysis modules are locally available [8].

Second generation

A system to support 2^{nd} generation MDO is characterized by distributed analysis modules each on a dedicated computer and a centralized design and optimization process [8]. In such a system different experts are responsible for their tool. With respect to the previous generation, an increased number of disciplinary interfaces are required over a distributed network. Flexibility of the design process is improved due to the fact that the system is not tightly coupled. This allows for easier exchange or addition of design modules. Due to the increasing number of interfaces data management and workflow management becomes increasingly important.

Third generation

In the quest to support MDO on the overall design of novel aircraft configurations an increasing number of systems and disciplinary analysis modules need to be integrated. These systems are too complex for a single user and require collaboration among hundreds of engineers, distributed among multiple specialized organizations. This collaborative design in distributed teams of engineers and tools is characterized as the 3^{rd} generation of MDO.

1.2.1. The design and optimization process

A typical design and optimization process consists of three phases: setup, operational and (convergent) solution phase (displayed in figure 1.3). The clear objective of the 3^{rd} generation of MDO is to reduce the lead-time of the design and optimization process. It is aimed to achieve this reduction in lead-time by enabling an increase of knowledge available in the early design stages. Moreover a reduction of abstraction is required, in order to minimize uncertainties during the design process.

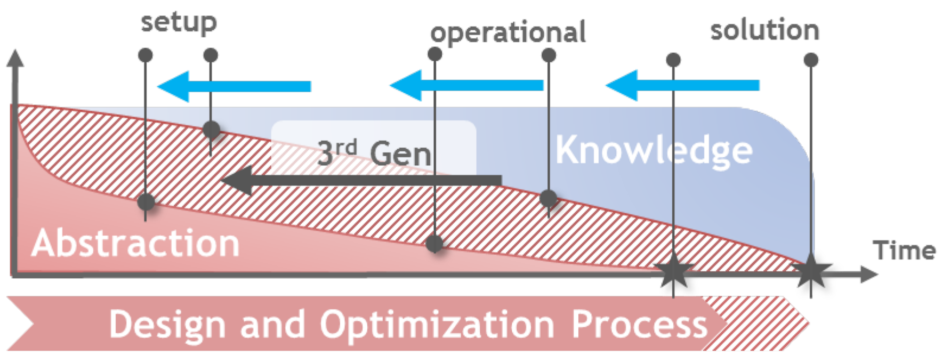


Figure 1.3: Different phases in the design and optimization process [8]

1.2.2. Example of an MDO framework architecture

To illustrate the traditional architectural components of an MDO framework the reader is referred to figure 1.4. This figure shows the Design and Engineering Engine (DEE) concept. The DEE is a modular, loosely integrated software system able to support MDO on the conceptual design of both conventional and novel aircraft configurations [18].

In the DEE the architectural elements can be distinguished as the initiator, the Multi-Model Generator (MMG), the Capability Modules (CMs) and a converger & evaluator. The initiator is used to define the initial set of parameter values for the required aircraft product model. The product model contains all main elements of the aircraft and the parameters describing these elements in a sufficiently detailed manner to support the design study defined in the requirements. In order to prepare for multiple disciplinary analyses, the MMG is used to generate multiple disciplinary models from a single product model required for each required disciplinary analyses. Finally all disciplinary analyses results need to be gathered and evaluated to obtain a (converged) design solution.

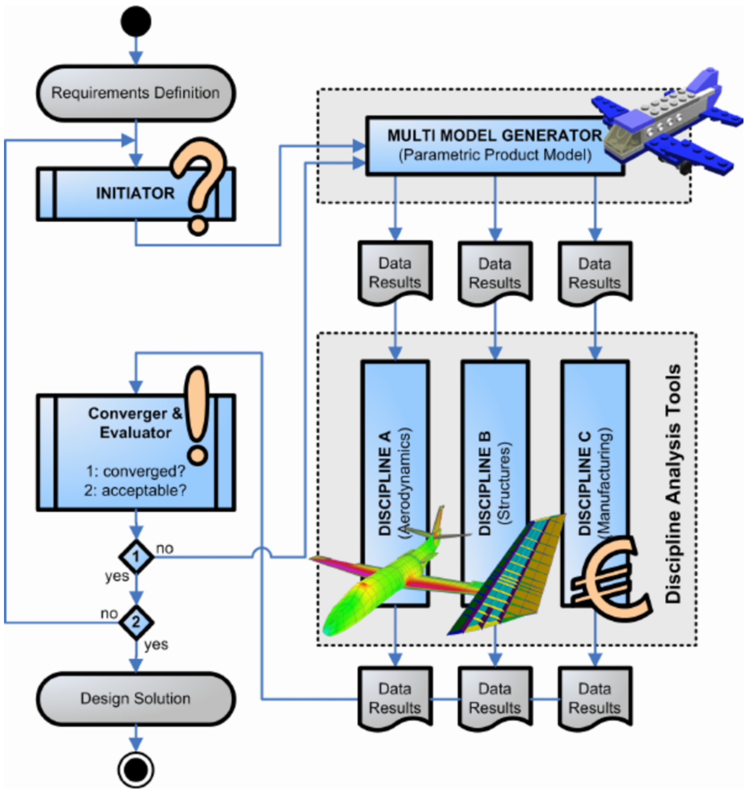


Figure 1.4: The concept of the Design Engineering Engine (DEE) to support MDO [4]

The DEE concept can be used in the development of either 1st, 2nd or 3rd generation MDO frameworks, depending on the distribution of disciplinary tools and/or experts. The general architecture and information access characteristics of an MDO framework should support a steering function during the design cycle, and provide the ability to monitor the progress and results during and after execution. Therefore the implementation of an intuitive Graphical User Interface (GUI) is a key requirement in the development of an MDO framework [4, 31]. Besides a GUI, modularity is another key requirement of an MDO framework which is required to be operable in a collaborative environment [27]. In this collaborative environment, intricate details quickly become too complex to understand for engineers with different disciplinary expertise.

1.3. AGILE research project

The aim of the AGILE research project is to improve on the current way of performing MDO on complex systems such as aircraft. AGILE stands for Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts [1]. AGILE targets multidisciplinary optimization using distributed analysis frameworks. The project is set up to proof a speed up of 40% for solving realistic MDO problems compared to today's state-of-the-art. The use-cases, displayed in figure 1.5, are realistic overall aircraft design tasks for conventional (top-left), open-rotor (top-center), strut-braced (top-right), box-wing (bottom-left), BWB (bottom-center) and UAV (bottom-right) configurations.

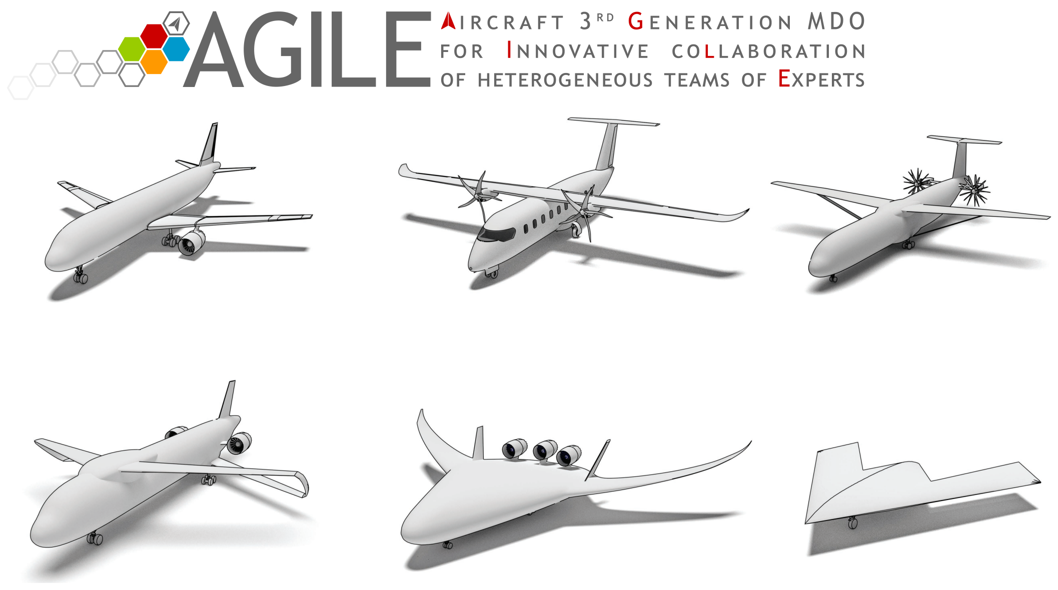

AGILE
Aircraft 3rd Generation MDO
for Innovative Collaboration
of Heterogeneous Teams of Experts

Figure 1.5: AGILE use-cases [8]

The project structure can be seen in figure 1.6. Each year represents a new design campaign. In the first design campaign a reference aircraft configuration is optimized using state-of-the-art techniques. The reference MDO problem is the end-result of the Initialization phase, referred to as Work Package (WP) 2. The reference MDO problem is used in design campaign to test novel optimization techniques in the MDO Test bench (WP3). Finally the most successful optimization approaches are used to explore novel aircraft configurations (WP4) in design campaign three. Three additional layers can be identified, which encompass all design campaigns. The first layer, Collaboration techniques (WP5), targets techniques to support distributed collaboration of specialists and tools. The second layer, Knowledge enabled information technologies (WP6), support the formalization and management of knowledge within an MDO process. Finally, the outer layer involves the coordination and dissemination of all design campaigns.

The work presented in this paper focuses mainly on WP6. At the time the author was involved in the AGILE project, the AGILE project was close to the half-way point of design campaign 2. Evaluation of design campaign 1 has shown that a lot of challenges regarding the setup of an executable design framework were tackled, such as standardized data handling, modular integration and accessibility through a distributed communication network. However, there is still room for improvement in terms of process formalization, knowledge

[1] http://www.agile-project.eu (accessed 11/03/2017)

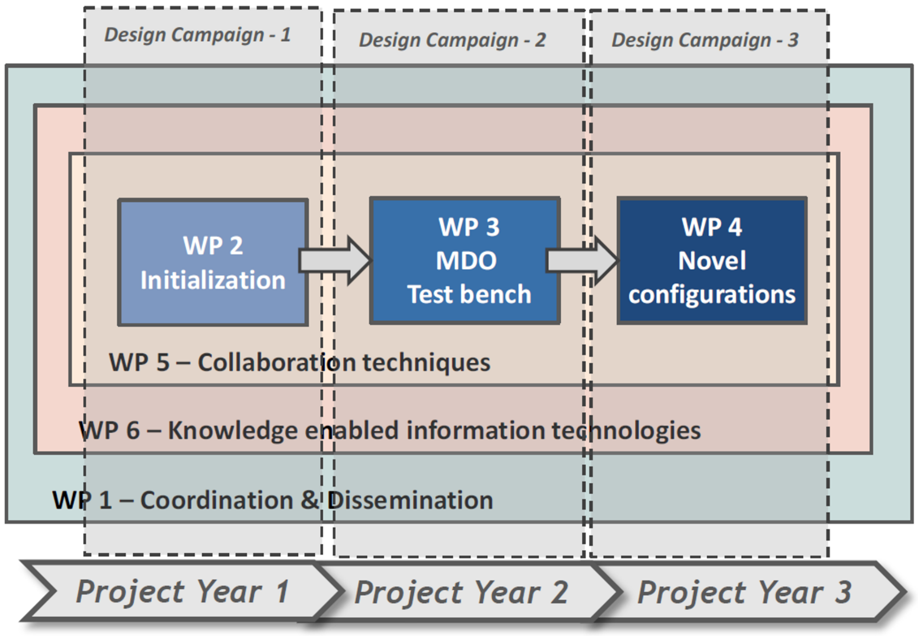

Design Campaign - 1
Design Campaign - 2
Design Campaign - 3
WP 2 Initialization
WP 3 MDO Test bench
WP 4 Novel configurations
WP 5 – Collaboration techniques
WP 6 – Knowledge enabled information technologies
WP 1 – Coordination & Dissemination
Project Year 1
Project Year 2
Project Year 3

Figure 1.6: AGILE project structure [8]

management and integration of systems in a collaborative design and optimization process. These are some of the challenges that will be addressed in this report. A more detailed decomposition of these challenges follows in section 1.5

1.4. Scoping and actors identified in a design and optimization process

The different actors that are involved in the development of an MDO framework in the context of AGILE are architects, integrators, discipline specialists, collaboration engineers and the customer. The roles of these actors can be described as follows:

- **Customer:** the primary user of the framework. Responsible for defining the design task, top-level requirements, and available development lead-time.
- **Architect:** responsible for specification of the design case in the framework, such as collecting the required tools, defining the design phases and the dimensionality of the design space to be explored.
- **Integrator:** responsible for the deployment of the design and optimization processes, and for the management of such processes within the MDO framework.
- **Discipline specialist:** Responsible for providing analysis tools within the framework, such as a simulation for a specific domain, or an optimization service.
- **Collaboration engineer:** Responsible for providing the integration and maintaining the interfaces within the framework, necessary to connect the various competences and making them accessible to the framework. It includes the secure integration of software apps in different networks.

The relation of the various actors with three domain layers in the MDO framework can be seen in figure 1.7. The level of abstraction on which the actors operate are indicated by the distinguished between the product and process model related activities. The integration of tools focus on the product scope. This product scope is characterized by the parametric and geometric properties defined in the product model. The upper layers focus more on the process model definition: the coupling of tools and activities in executable processes.

The amount of actors involved in the setup of 3^{rd} generation MDO frameworks lead to challenges in ensuring consistency of models and knowledge during the design and optimization process. This comes from

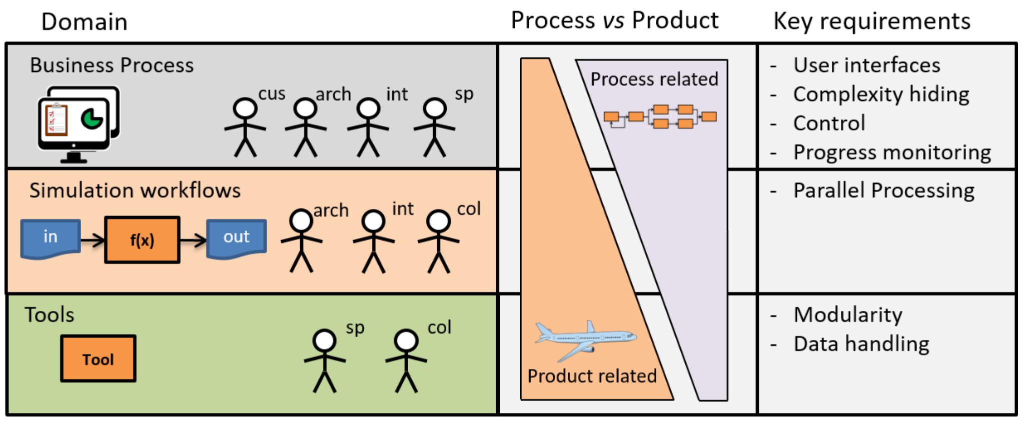


Figure 1.7: Different levels in the information domains, actors (customer, architect, integrator, discipline specialists, collaboration engineer) involved in the context of AGILE and key requirements of each level in MDO framework development (adapted from [4])

the fact that a large amount of people involved in a single development process increase the risk of having different interpretations on the same model or representation of knowledge. Moreover, an increasing number of actors involved in a single project requires a good subdivision of work and assigned roles in the design and optimization problem. This challenge and other challenges involving the setup of 3^{rd} MDO frameworks are further explained in the next section.

1.5. Challenges in setting up of 3^{rd} MDO frameworks

As can be read in the section 1.1, increasingly complex and collaborative MDO frameworks are required to create a breakthrough in novel aircraft development. This however surfaces a lot of challenges, presented in this section.

The success of MDO framework implementations not only lies in improving on the setup of executable MDO workflows, but also the ability to reuse the underlying models developed along the way. As supported by a critical analysis on various MDO frameworks by Padula and Gillian [27] it is found that a lot of the effort put in the process of producing a good design is often lost or discarded before a next level or design cycle was initiated.

As concluded from the setup of the first AGILE design framework much effort was required to identify all input/output relations during the integration of the various tools in the setup phase of the framework development process [8]. Moreover, the current methods to couple each tool to a common data model is not easily maintainable due to manual editing of input and output files using XML definitions. During this process of manually defining input/output relations of each tool it is currently difficult to answer questions such as: *do I have enough tools to solve the MDO problem? Is data flow possible? Is my tool using the correct product definition as used by other tools? What is the impact of introducing a new tool in the process model?* These challenges contribute to the fact that 80% of project lead-time is typically spent on the setup of the framework [8].

Given that there is only a small portion of time left after obtaining a first design iteration, puts a lot of stress on the design team. Moreover, the limited time left to run the simulation workflows for different design scenarios poses a large risk on the project [42].

A big challenge that remains after setting up an executable framework in the design and optimization process is a lack of understanding and flexibility. This lack of flexibility and understanding is most often the result of a missing system level overview, leading to difficulties in finding inconsistencies in the setup simulation workflow, introduction of new design competences, managing requirements and the making of design decisions [42]. Besides a lack of system level overview, collaboration is often hampered by working on distributed workflows from different locations. Hence, the making of design decisions is affected by geographical barriers.

Suboptimal collaboration between partners involved and integration of available capabilities within the consortium yields waste in the design and optimization process. Waste can be characterized as effort necessary to overcome difficulties in accessing, retrieving information or confirm and correct inaccurate information [14]. This additional effort is associated with an increase in man-hours. Hence, waste leads to additional costs and an extension of project lead-time.

This thesis report does not consider the challenges involved in achieving a convergent solution in the shortest time possible by addressing the architecture, technologies or computation techniques involved in solving MDO problems. However, an approach is followed to address the challenges of improving the integration of the various actors, tools and automated design processes in the setup and operation on 3^{rd} generation MDO frameworks. The challenges addressed during the research presented in this paper can be summarized as follows:

- A lack of formalization of the business process in the setup of MDO frameworks leads to inefficient integration of knowledge and tools.
- The complexity of collaborative design requires different abstractions of knowledge to coordinate the development process of the framework. This knowledge is currently not easily accessible and traceable during the development process.
- The current use of raw XML editing to support database management features are not easily maintainable. This leads to a lot of effort required to make disciplinary tools compliant to a common data model.
- Due to lack of formalization of the overall development process of the MDO framework maintainability and reuse in knowledge during and between projects is at risk.
- Feedback on the completeness and consistency of the framework for a particular design case is discovered in the later stages of implementation resulting in waste and undesired increase in lead-time of the development process.

1.6. Research Objectives

The research presented in this thesis aims to support the setup of collaborative MDO frameworks in the context of the AGILE research project. As presented in section 1.5 various various open challenges remain in the integration of a business process to assist the various actors in setting up such a framework in a distributed setting. For this purpose the work presented in this thesis aims to develop a collaborative environment to integrate the business process activities required to formulate and operate the design and optimization process. In order to investigate means to improve the integration of the various actors involved in the setup phase of the MDO framework, allow for more collaboration and control to reduce project lead-time the following research objective is defined:

Support the setup of 3^{rd} generation MDO frameworks by developing a collaborative environment in which the activities leading to an automated design workflow are integrated in the business process layer

In order to integrate the business process layer, tools and data model an interface is required to facilitate knowledge exchange. In this thesis research a collection of tools coupled to a common data model is defined as a process model and the data model as a product model. Together these models are used to define the foundation of the MDO framework. The status of integration is determined by completeness and consistency of the set of tools and data model to arrive at an executable MDO workflow. In order to measure and value the completeness and consistency of the MDO framework the following sub-objectives are defined:

- **Sub-objective 1:** Enable the automatic integration of tools and the data model by reuse of standardized product and process models.
- **Sub-objective 2:** Provide feedback to the architect during integration of the MDO framework components by enabling in-the-loop model verification.
- **Sub-objective 3:** Enable inspection on the MDO framework status of integration by developing a GUI to support MDO framework development in the AGILE project.

In order to test whether the goals are achieved, the developed collaborative environment is deployed and tested during workshop sessions. Moreover, the collaborative environment will be used to support in the formulation of WP4 use-cases introduced in section 1.3 to support the analysis of novel aircraft configurations.

1.7. How to read

The remainder of this report is composed of five chapters. In this five chapters the author would like to guide the reader through the undertaken steps taken during this thesis research. In chapter 2 the terminology used and all components present that describe the knowledge architecture of an MDO framework used within this report. Moreover, a formalized approach is presented which characterize the steps required to setup and operate an MDO framework. In chapter 3 all enabling methodologies and technologies researched and used in the context of this research are presented. Chapter 4 discusses the implementation of a collaborative implementation to support the setup and operation of 3^{rd} generation MDO frameworks. Next, in chapter 5 the results of this research are presented, followed by the conclusions and recommendations in chapter 6.

2 Design and optimization system

In this chapter the design and optimization process is analyzed to identify the main steps involved in the setup and operation phases. In order to develop a collaborative environment which acts as a single system to support in this design and optimization process this chapter tries to answer questions such as:

- *What are the main MDO framework requirements which need to be supported by the system?*
- *What are the main steps involved in the setup and operation phase of the design and optimization process?*
- *What are the distinguishable information layers and components in the MDO framework which require integration in the collaborative environment?*

First an overview of key requirements in the development of any MDO framework is presented in section 2.1. Integration of all components is key to provide an operable system to support in the design and optimization process. The challenges implied with system integration are presented in section 2.2. Next, in section 2.3 the main steps implied with an MDO-based development process are described. The MDO framework development process, derived from the MDO-based development process, drives the functionality that the collaborative environment developed by the author needs to support. Finally, in section 2.4 the knowledge architecture describing all layers and components which define an MDO framework is presented. This knowledge architecture plays an important role during the development and integration of the collaborative environment's main components and interfaces developed during this research.

2.1. MDO framework requirements

In order to support the various actors involved in the setup, operation and execution phases of a design and optimization process various requirements need to be satisfied. These requirements are imposed on the architectural design, problem formulation, problem execution and accessibility of information. The set of requirements imposed on a design and optimization system are presented in the following sections. These requirements extend the MDO framework requirements defined by Salas and Townsend [31].

Architectural design

Salas and Townsend [31] propose the following requirements on the architectural design of an MDO framework:

- **Requirement AD1** *Intuitive GUI*: The GUI should allow for end-users to quickly understand and learn to use the features of the framework.
- **Requirement AD2** *Design using Object-Oriented (OO) principles*: The strength of OO modeling is that it allows for flexible and re-usable programming [9]. OO modeling has various advantages in the context of MDO applications, such as switching between analysis or optimization methods at run-time and extends naturally into distributed computing.

- **Requirement AD3** *Extensible and support for developing the interfaces required to integrate new processes into the system*: Integration of new tools or discipline codes, optimization methods in the framework should be enabled for the end-users.

- **Requirement AD4** *Minimized overhead imposed on the optimization process*: Performance measurements should be provided to enable the user to identify time-consuming activities.

- **Requirement AD5** *Ability to handle large problem sizes*: In order to support design and optimization problems with a high level of fidelity the framework should support several thousands of design variables.

- **Requirement AD6** *Support collaborative design*: As stated in section 1 next generation MDO frameworks involve different disciplines and associated specialists. Therefore it is desired that the framework architecture allows for collaborative design in which multiple users have simultaneous access to problem data.

- **Requirement AD7** *Utilization of standards*: Use of standards ensures that data and messages used within the framework are unambiguous to disciplinary boundaries. A common language, for storing problem information and exchanged data is required to preserve invested knowledge. This lowers the maintenance cost of the framework.

Problem formulation

Salas and Townsend [31] propose the following requirements on the problem formulation capabilities of an MDO framework:

- **Requirement PF1** *A framework should allow the user to configure complex branching and iterative MDO problem formulations easily without low-level programming*: Using a higher level of abstraction enables the user to formulate an MDO problem faster with less risk of errors. This ideally is extended through a visual programming interface.

- **Requirement PF2** *Easy reconfiguration of MDO problem formulations*: This involves editing existing processes or adding new processes. This enables the user to explore alternative solution strategies of a problem without the need to build a problem from nothing.

- **Requirement PF3** *Support the user in incorporating legacy codes and proprietary codes into the MDO problem formulation*: Supporting the use of legacy and proprietary codes enables the users to continue working using familiar coding languages and improves productivity. To support this, the MDO framework should support wrapping functions that allow for tools and codes to communicate with each other. These wrapping functions should transform outputs of interest and inputs in appropriate files for each tool.

- **Requirement PF4** *Support several optimization methods*: As not a single optimization method is sufficient to support each optimization problem, various optimization methods should be provided.

- **Requirement PF5** *Ability to debug multiple processes executing on computers across a network*: The framework should provide feedback to the user to warn for improperly configured processes and allow the user to monitor progress during execution to step through intermediate results of several remote computations.

Problem execution

Salas and Townsend [31] propose the following requirements on the problem execution of an MDO framework:

- **Requirement PE1** *Automated execution of processes and the movement of data*: Traditionally discipline specialist rely on their own scripts to transform outputs of interest provided by other tools to their own desired format, this process should be automated. Preparing interfaces between tools in such a way that human transformation activities are required allow for faster transfer of data between processes.

- **Requirement PE2** *Parallel execution of processes*: Parallel execution, if possible, would be advantageous for computationally intensive MDO problems. The framework should provide the ability to identify and execute codes or subsystem optimization in parallel without compromising the result.
- **Requirement PE3** *Support execution distributed across a network of heterogeneous computers*: The framework should utilize the advantages of running tools and codes on computers optimized for a certain analysis available in the network. This is also important to remove license constraints, where tools can be remotely executed without the need to have the tool running locally.
- **Requirement PE4** *Support user interaction (steering) during the design cycle*: During the design cycle the user needs to be in the loop continuously to monitor progress and adjust the process if desired. In MDO problems this involves changing constraint or design variables, adjusting termination or convergence criteria or replacing analysis tools.
- **Requirement PE5** *Allow for batch operation*: Ideally, the users needs to be able to define a set of problems which can be executed after each other without user interaction. This would allow the user to experiment with multiple starting points for an optimization problem.

Accessibility of information

Salas and Townsend [31] propose the following requirements on the accessibility of information of an MDO framework:

- **Requirement AI1** *Provide database management features*: A central database to maintain data used by multiple disciplines is convenient for large design problems. The user should have the option to define which data are written to and read from the central database.
- **Requirement AI2** *Visualize intermediate and final optimization and analysis results*: Using the data stored in the central database, results should be made available. These results should provide the ability for the user to tack the behaviour of design variables, constraint or objective variables and the overall design.
- **Requirement AI3** *Provide the ability to monitor the status of the system and an ongoing execution*: The framework should provide visual feedback on which processes are currently executing. The ability to monitor progress allow for the user to be alerted of potential problems such that timely actions can be taken.
- **Requirement AI4** *Provide some mechanism for fault tolerance*: The framework should provide the ability to recover from a failed execution without the risk of losing previously generated data. The framework should allow the user to restart from a previous state.

The author proposes to extend requirement AI1 to provide for database management features on the problem definition level as well. This allows for storing a complete set of data leading to a certain result. This leads to improved transparency of the overall design and optimization process, but also enables part of the design chain to be reused in future processes.

During this research the focus lies on complying to the requirements regarding the architectural design, problem formulation and accessibility. These requirements drive the integration of the various components in the collaborative environment which will be explained in more detail in section 4.1.

2.2. System integration

A system that allows for the setup, operation and execution of the design and optimization process should comply to the requirements defined in section 2.1. In order to support the modular integration of new processes in the system (requirement AD3), utilize modeling standards (requirement AD7), support easy (re)configuration of MDO problem formulations (requirements PF1 and PF2), provide a steering function during the design cycle (requirement PE4) and provide an accessibility of information (requirements AI1-AI4) systems integration is required. Systems integration is the composition of a capability by assembling elements in a way that allows them to work together to achieve an intended purpose.

Within the AGILE consortium various capabilities are available to support in problem formulation, problem execution, visualization of results, standardization and distributed collaboration. As stated in section 1.5 an encompassing system to integrate all elements does not yet fulfill all framework requirements. In an approach to integrate all elements in a single system, vertical integration is required. Vertical integration is the process of integrating elements developed within a similar context to support one common purpose. Vertical integration of all elements reduces waste in the design and optimization process, as a lot of interfaces are automated.

2.3. MDO-based development process

In order to develop an MDO workflow, various steps should be taken into consideration. A typical MDO-based development process is displayed in figure 2.1. This typical MDO-based development process is split in two main phases: the formulation phase and the execution phase [43]. The formulation phase depicted on the left side of the figure is used to define the MDO problem and setup an inexecutable MDO process. This formulated MDO process is used as a blueprint for an executable simulation workflow. Before an executable simulation workflow is obtained, the neutral MDO process formulation needs to be translated into a formulation required by a selected PIDO (Process Integration and Design Optimization) application. After workflow execution an (optimized) design solution is obtained and the MDO-based development process can be repeated to obtain different design solutions.

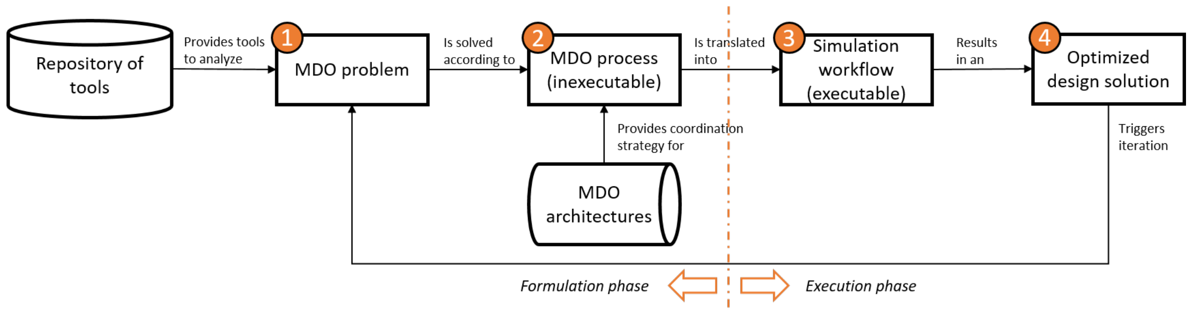


Figure 2.1: Typical MDO problem formulation and execution process overview and terminology (based on [43])

The MDO-based development process presented in figure 2.1 focuses mainly on problem formulation and problem terminology. What's missing in this figure is a layer of overhead. Behind an MDO problem, there is a customer who has a product he or she wants to (re)develop. Moreover, there is an architect who defines the premises of the MDO problem that needs to be specified. A layer of overhead to structure the MDO based development process is important if one wants to work collaboratively on the setup and operation of a distributed MDO framework. In collaboration of the author with members of the AGILE consortium, five steps were identified to formalize the activities required to develop such an MDO framework. The goal of this so-called *MDO framework development process* is to assist in the development of an integrated application used to solve a specific MDO or other complex design problem. The MDO framework development process can be seen in figure 2.2 along with its relative position w.r.t. complex PDPs.

Figure 2.2 displays five steps which describe the MDO framework development process. Although these steps are presented as sequential, previous experiences dictate that this process is highly concurrent in practise. The earlier defined MDO-based development process can be projected on the more complete MDO framework development process. Each of the five steps can be explained in more detail as follows:

1. **Define design case and requirements:** In the first step requirements on the design concept, required fidelity, available lead-time, required analyses, available competence providers, etc. are formalized. Next, a set of available design competences and key design parameters which need to be included in the design case are agreed upon with the discipline specialists, architect and customer(s).

2. **Specify repository of design competences and data model:** In the second step the underlying models of the design case must be defined. These underlying models are the product and the initial process model. This product model contains all parameters which are used to describe the product which is

[1] Based on: *AGILE, "D65 AGILE framework architecture," AGILE project (H2020-636202), 2016*

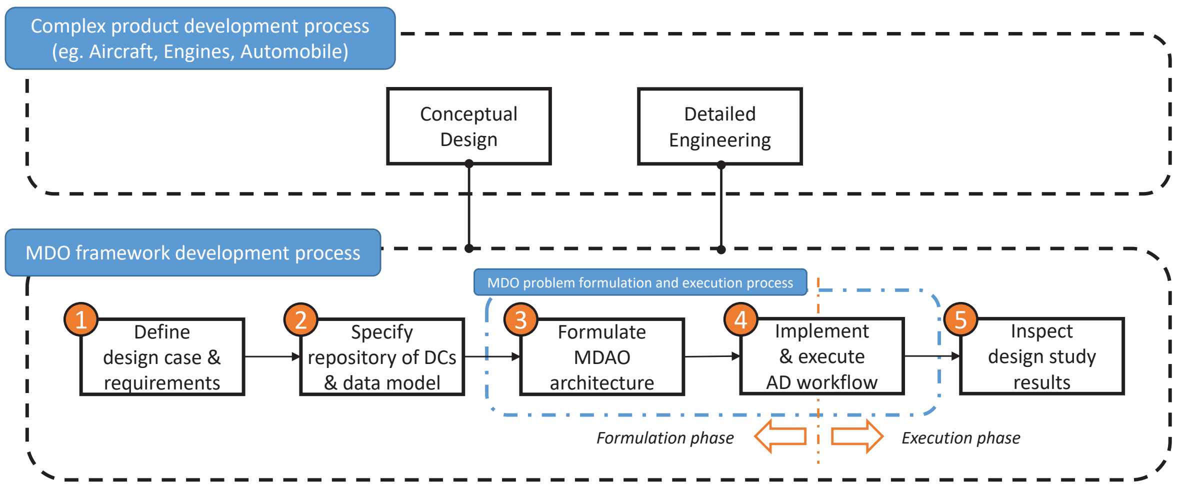

Complex product development process (eg. Aircraft, Engines, Automobile)
Conceptual Design
Detailed Engineering
MDO framework development process
MDO problem formulation and execution process
1 Define design case & requirements
2 Specify repository of DCs & data model
3 Formulate MDAO architecture
4 Implement & execute AD workflow
5 Inspect design study results
Formulation phase
Execution phase

Figure 2.2: MDO framework development process and its relation w.r.t. the product development process of an aircraft, engine, automobile, etc. [1]

to be designed. The initial process model contains all design competences, of which the inputs and outputs are defined. The design competences, coupled to a common data model form a case-specific repository, used to formulate the MDO process.

3. **Formulate Multidisciplinary Design Analysis and Optimization (MDAO) architecture:** In the third step of the development process the formalization stage of the design and optimization process is entered. In this step a suitable MDAO strategy and architecture is formulated to solve the design case.

4. **Implement and execute the Automated Design (AD) workflow:** In the forth step an Automated Design (AD) workflow is implemented in a selected PIDO application. The executable workflow uses the formulated MDAO architecture of the previous step.

5. **Inspect design study results:** Finally in the fifth step the design study results from the previous step can be inspected. This requires transformation of the raw data produced by the executed MDO workflows to graphs and figures which can be interpreted by the integrator and customer.

The main difference between the MDO-based development process defined in [43] and the MDO framework development process is the actor perspective. The MDO-based development process is formalized from the architect's perspective, whilst the five steps describe a process in which the integrators, architect, collaboration engineers, discipline specialists and the customer have a clear distinguishable goal. For example, assembling a repository of tools required to solve an MDO problem requires actions from various discipline specialists. The MDO architect is dependent on the discipline specialists to make sure their tools are available and compliant to a common data model, but ideally does not need to be involved in the first two steps of the development process. Having a clear distinction between roles and deliverables at each stage of the development process, helps the collaboration process and speeds up the overall design process by ensuring each actors operates in an environment which suits their abilities best.

2.4. MDO framework knowledge architecture

The methodological approach followed in this research works on the foundation of the MDO framework knowledge architecture defined in [42] which can be seen in figure 2.3.

The knowledge architecture contains three layers which represent an MDO framework from top to bottom: **Product Development** (PD) layer, **Automated Design** (AD) layer and **Design Competences**(DCs) layer. The figure also shows the required interfaces between the different layers. An extra data layer is added to control the integration of DCs in the PD layer. These layers and their components are explained in more detail in the next sections to illustrate the integration required in the collaborative environment that needs to be developed. This collaborative environment involves mainly the PD layer and the PD/AD interfaces.

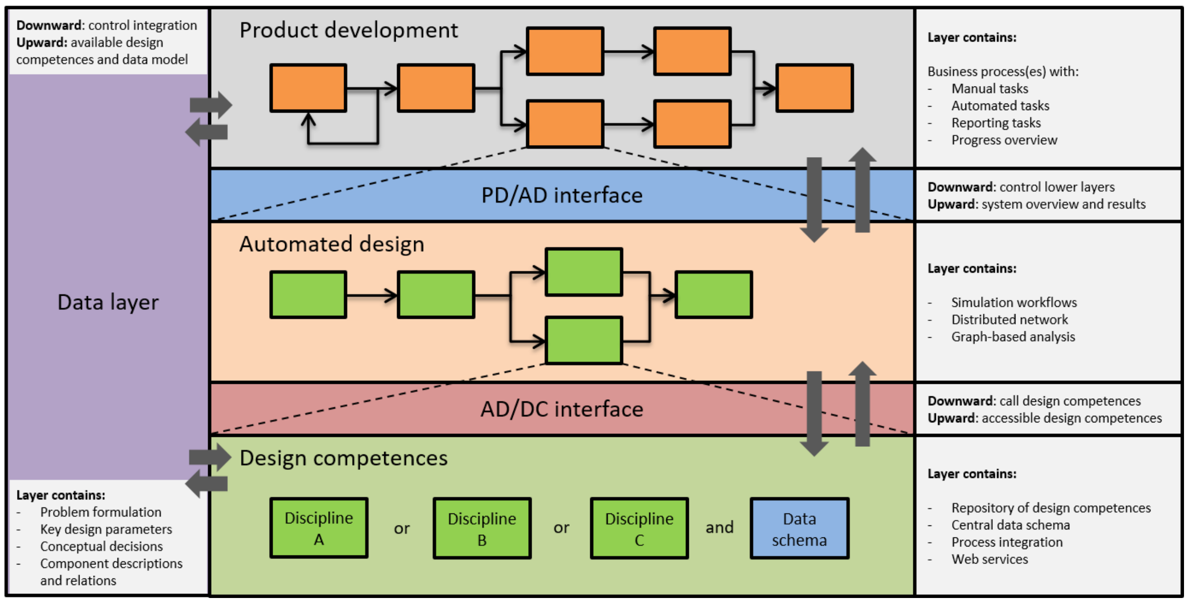


Figure 2.3: Knowledge architecture of the various components and information layers which can be identified in an integrated MDO framework (adapted from [42])

2.4.1. Product Development (PD) layer

The Product Development (PD) process is the top layer in the knowledge architecture. This layer contains both manual and automated tasks which contribute to the development of the product desired by the customer. The main purpose of the PD layer is to assist in the design of an optimal product which meets all requirements acquired from the customer. The manual and automated tasks in the PD define a business process, which is used to control the setup and execution of the design and optimization process. According to van Gent *et al.* [42] two primary functions of the business process are:

1. Support the user in setup of a design problem and related simulation workflows
2. Support the user during the execution of the required automated design process with manual interfaces to that process and by retrieving information on the simulation process performance and the design results.

In this thesis research a web-based information and Workflow Management (WFM) system KE-chain is used to integrate the PD layer of the MDO framework. KE-chain is developed by KE-works [2]. KE-chain provides access for multiple end-users to a single project through a user-based authentication system. KE-chain facilitates management of project data, integration of external applications in the business process and monitoring of progress. Based on these strengths KE-chain provides an excellent interface for the different actors operating in the PD layer.

The current state and required extensions of the application architecture of KE-chain is displayed in figure 2.4. Two main domains can be identified: the client and server domains. The client domain is composed of an Ext JS 6 [3] frontend. This frontend is accessible to any PC or tablet through the web-browser. A KE-chain client communicates to the backend application server through a user-authentication protected Django REST framework Application Programming Interface (API) [4]. The backend core's main component is called a scope. A scope is a container to store project related information. A scope has two information domains called the Product Information Model (PIM) and the Workflow Information Model (WIM). The domain of PIM contains all models and actions used to read and write product related information. The domain of WIM contains all models and actions to read and write process related information. The domains of PIM and WIM will be explained in more detail in sections 3.3.1 and 3.3.2 respectively.

[2]`http://www.ke-works.com(accessed11/03/2017)`
[3]`http://docs.sencha.com/extjs/6.2.0/` (accessed: 10/05/2017)
[4]`http://www.django-rest-framework.org/` (accessed 14-05-2017)

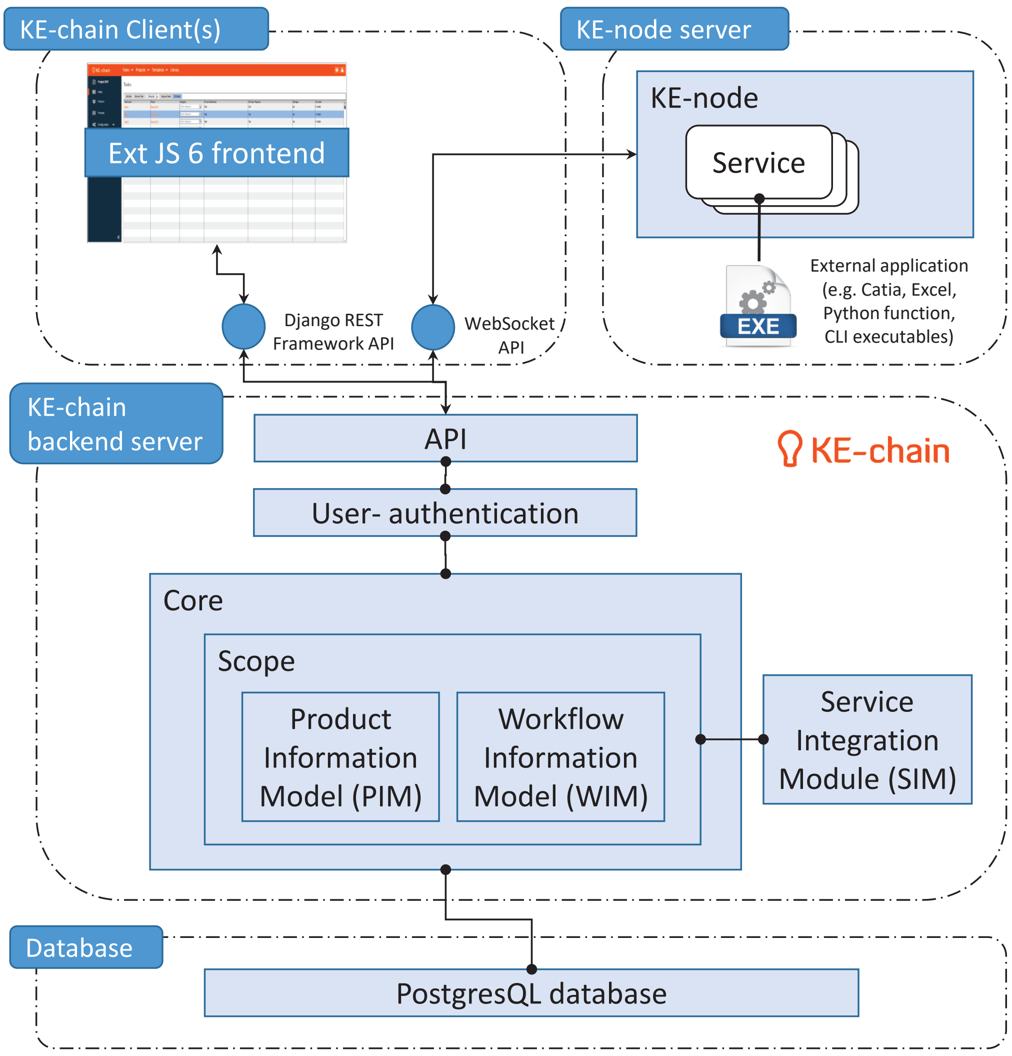

KE-chain Client(s)
Ext JS 6 frontend
Django REST Framework API
WebSocket API
KE-node server
KE-node
Service
External application (e.g. Catia, Excel, Python function, CLI executables)
EXE
KE-chain backend server
KE-chain
API
User- authentication
Core
Scope
Product Information Model (PIM)
Workflow Information Model (WIM)
Service Integration Module (SIM)
Database
PostgresQL database

Figure 2.4: Application architecture of KE-chain

In order to achieve the desired PD layer functionality to support in the setup of 3rd generation MDO frameworks improvements to the KE-chain architecture need to be made. The following improvements are required:

- Extend the core of the KE-chain backend server to enable reuse of modeling knowledge to support in the accelerated setup of new PDPs. The component developed by the author to improve the reuse of modeling knowledge is called the knowledge library and is further discussed in section 4.3.
- Extend the core of the KE-chain backend server by a module which extends the domains of PIM and WIM such that the coupling of both domains can be verified and monitored to feedback on the consistency of defined product and process models. The components developed by the author to monitor and verify the coupling and configuration of product and process models are called the model verification and model inspector module. Both developed components are further discussed in section 4.2.
- Extend the functionality of SIM with methods to support in the retrieval and writing of information which support the end-user in the inspection or sharing of product or process knowledge. The implementation of the extended functionality is further discussed in section 4.2.
- Extend the functionality of SIM such that it is able to throughput communicate the status of service execution. These improvements are further discussed in section 4.1

- Extend WIM to allow for customized task layout to improve user experience by changing the visualization of information contained in task-forms and improving navigation patterns. These improvements are further discussed in section 4.2.

All components enlisted above require an extension of both the KE-chain backend server and the frontend as displayed in figure 2.5. In this figure the introduction of new components are indicated in green and components and the extension of existing architectural components is indicated in orange. The backend needs to be extended to define all models and operations which are required to add the desired functions. Moreover, the API needs to be extended to enable exposure of the new functionality in the KE-chain frontend. Finally, the frontend needs to be extended to create the new views, dialogues and to make the required PD functionality available for the KE-chain client(s).

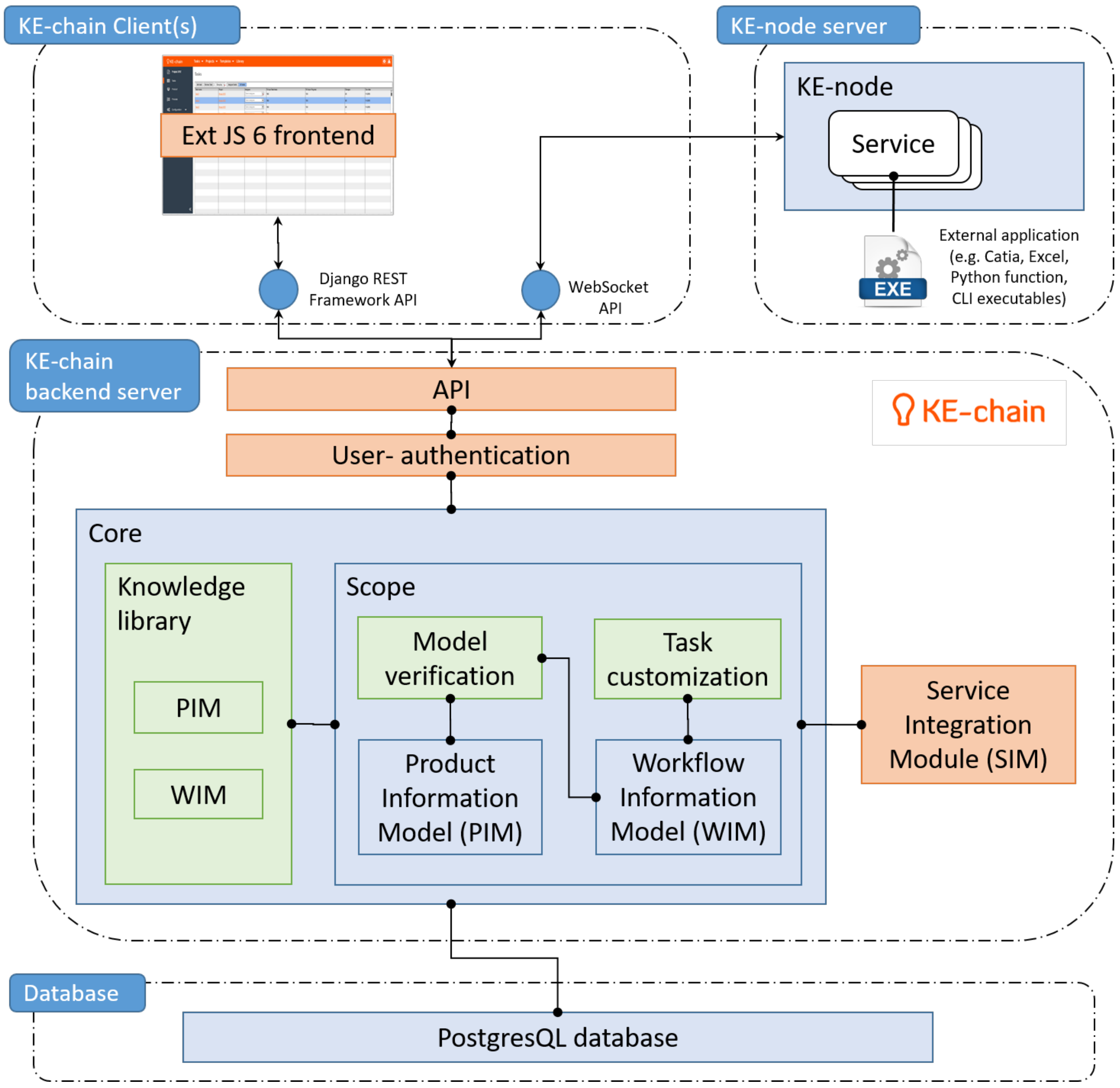


Figure 2.5: Required extensions to the application architecture of KE-chain to support in the required PD layer functionality. New components which need to be developed are indicated in green and components which require extension are indicated in orange.

To support in the integration of external applications KE-node is used. KE-node provides a remotely accessible server environment from which local services can be executed. These local services can range from Python scripts to external Computer Aided Design (CAD) applications such as Catia, PIDO applications and Microsoft Excel scripts which can be executed through the Command-Line Interface (CLI). KE-node communicates directly to KE-chain through a WebSocket (WS) API. This WS API communicates to the KE-chain server through a user-authentication system. The information which is exposed from KE-chain to the KE-node server is dependent on the configuration of special service tasks. A service task is schematically displayed in figure 2.6. A service task extends the human based tasks in WIM with additional functionality. A service task can be configured such that it reads and writes data to a KE-chain project through input and output mappers. These input and output mappers should comply with the input and output schemes defined on the KE-node service. Input and output schema validation ensures a service obtains a correct set of input data and writes a correct set of output data to KE-chain.

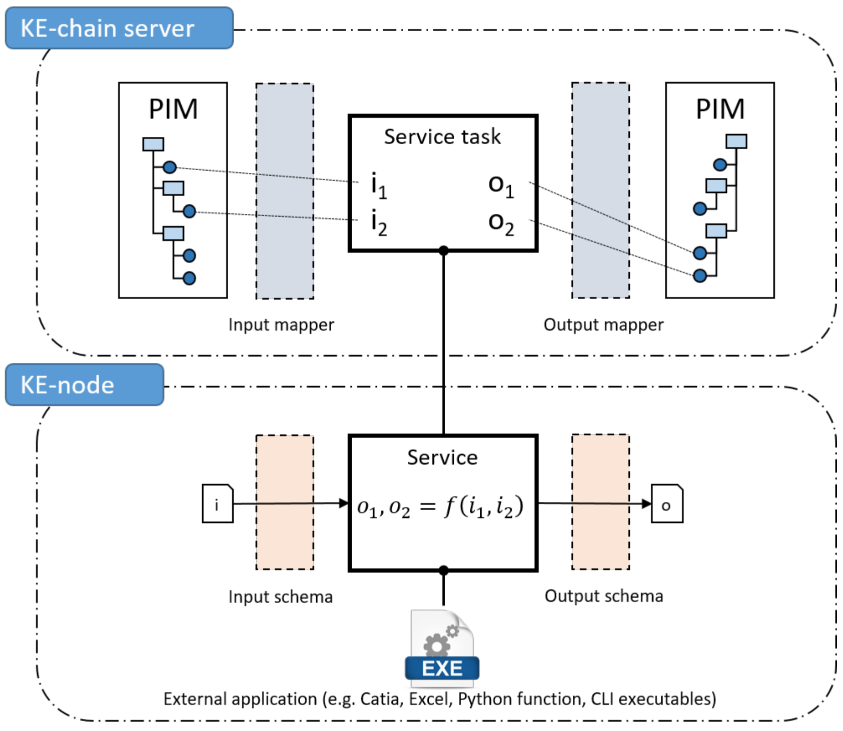


Figure 2.6: Schematic overview of KE-chain service task

Using KE-node services, various applications used throughout the MDO framework development process can be integrated. Applications used in the AD layer of the AGILE framework architecture are described in the next section.

2.4.2. Automated Design (AD) layer

The AD layer contains simulation workflows which can be executed in distributed networks. The simulation workflows are composed of several DCs which represent disciplinary analysis tools. The interaction between the PD and AD layer aims to control the execution but also the setup process of the MDO framework.

In AGILE, a graph-based system is in development called KADMOS (Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System) [43]. In order to integrate the PD and AD layer the Common Multidisciplinary Design and Optimization Workflow Schema (CMDOWS) is used. KADMOS and CMDOWS are explained in more detail in the following sections.

KADMOS

KADMOS aims at speeding up the setup phase of the MDO framework by assisting the architect in defining the strategy to solve the MDO problem, architecture and solution strategy. KADMOS is used to automate the formalization side of the MDO-based development process depicted earlier in figure 2.1. For this, KADMOS uses graph based analysis. The KADMOS process is schematically depicted in figure 2.7. This schematic overview shows a a representation of the KADMOS graphs associated with the various steps in the MDO-based development process and an example of the associated MDO workflows in a neutral format, the Extended Design Structure Matrix (XDSM) [19]. The example KADMOS graphs and XDSMs show the formalization of a simple MDO workflow defined as the Sellar problem[33]. This Sellar problem is composed for simple analytic functions representing disciplinary tools, constraint functions and an objective function. The Sellar problem is commonly accepted within the MDAO community to test various MDO architectures[5]. All steps of the KADMOS process are automated, and controllable through a Python CLI.

The starting point for KADMOS is a knowledge base. This knowledge base contains an abstract repository of all tools. This abstract repository contains knowledge on a tool's execution time or fidelity level and a set of all input and output parameters which originate from a common data model. Importing all tools and their connections to the common data model is transformed in a Repository Connectivity Graph (RCG). Next, redundant tools or functions are eliminated from the graph, leading to a Fundamental Problem Graph (FPG). The next step is to wrap an MDO architecture (e.g. MDF, IDF, DOE) around the problem. After completion of all steps a neutral formulation of the MDO process can be exported, such that it can be translated to specific Simulation Workflow (SWF) software. This neutral formulation is stored according to the CMDOWS standard, defined in the next section.

CMDOWS

CMDOWS contains meta-data on all DCs such as their required inputs and outputs as can be seen in figure 2.8. These inputs and outputs refer to parameters defined in a common data model. This is further explained in section 2.4.3. The parameters and DCs form the nodes in a graph that is used to define the MDO workflow. Parameters that have a specific role (design variable, objective or constraint) in the MDO architecture are stored as architecture elements. Finally the CMDOWS stores information on the problem definition and the MDO architecture. CMDOWS is currently under development within the AGILE consortium [6].

2.4.3. Design Competence (DC) layer

The DC layer contains the various disciplinary tools and functions necessary to solve a particular MDO problem. In a multi-site and collaborative approach to setup an MDO framework each DC has a discipline specialist responsible for maintenance of his/her tool. Moreover this discipline specialist is responsible for ensuring a tool is able to communicate with other tools. In order to enable integration of all DCs in the simulation workflows, they need to read and produce data which is in accordance with a central data model definition.

A central model approach is used to define a common namespace. The number of interfaces between analysis modules has a large effect on the efficiency in data exchange and flexibility of a design environment [5]. For this purpose a central model approach, depicted in figure 2.9, is a great enabler for reducing the number of interfaces. As disciplinary modules (indicated by blue circles) do not exchange data directly between one another, but trough a single data model, the number of interfaces is reduced by a factor of $\frac{(n-1)}{2}$. Here n is the number of disciplinary modules.

The central data model used in AGILE is CPACS. CPACS is acronym for Common Parametric Aircraft Configuration Schema is an XML-based data model developed at the German Aerospace Center (DLR) [23] [8]. CPACS, depicted in figure 2.10 describes the characteristics and attributes of aircraft, engines, climate impact, fleets and mission in a structured, hierarchical manner for various levels of fidelity. Integration of all DCs in an executable workflow is controlled by ensuring each DC reads input data stored in a CPACS file and writes output data back in that CPACS file. This way of standardizing the data flow ensures that a common language is used to work on a single product definition by removing ambiguity in what data is to be exchanged.

In this chapter the knowledge architecture describing an integrated MDO framework is presented. In this knowledge architecture various layers and components were identified which need to be integrated through appropriated interfaces. In this thesis research mainly the PD layer and PD/AD interfaces are considered. In this PD layer a high amount of information needs to be exchanged, acquired and visualized to support the actors in the setup and operation of an MDO framework.

In the next chapter lean information management principles are presented. Lean information management principles are investigated in the context of this thesis research in an approach to improve the performance of the overall PDP. The approach here is to reduce the design and optimization process lead-time through mitigation of waste. Moreover the next chapter discussed knowledge technologies to support in the information and knowledge interfaces which needs to be developed for the PD & AD layers. Finally the next chapter addresses the topic of product and process modeling. This plays an important role in the definition of the MDO framework's underlying models.

[5]Example of the Sellar problem formulated by the MDAO community, `http://openmdao.org/releases/0.2.5/docs/mdao/intro.html` (accessed: 06/06/2017)

[6]The CMDOWS data schema is in development by the AGILE project, `www.agile-project.eu`, to be published in 2017

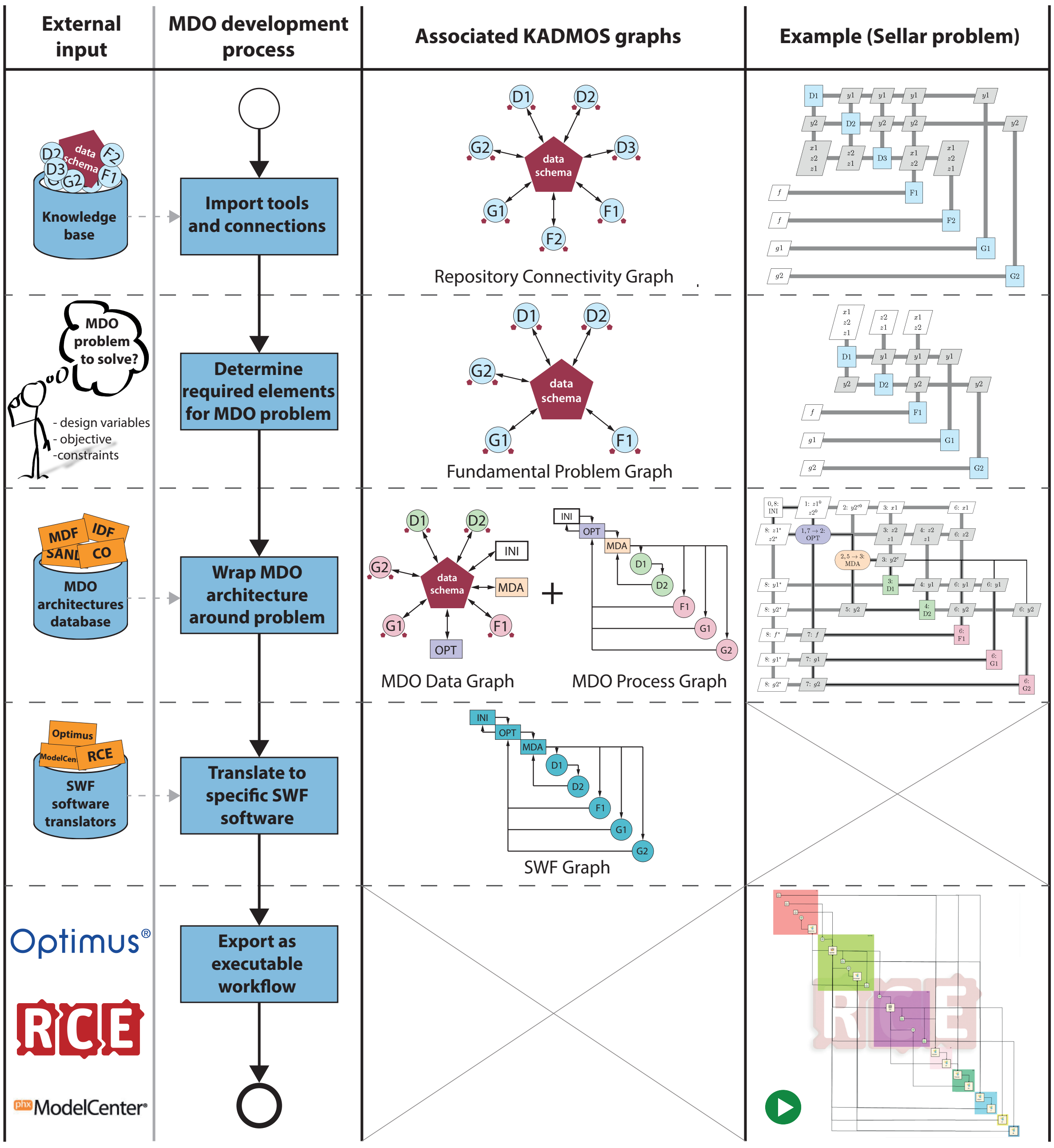


Figure 2.7: Schematic overview of the MDO development process in KADMOS [43]

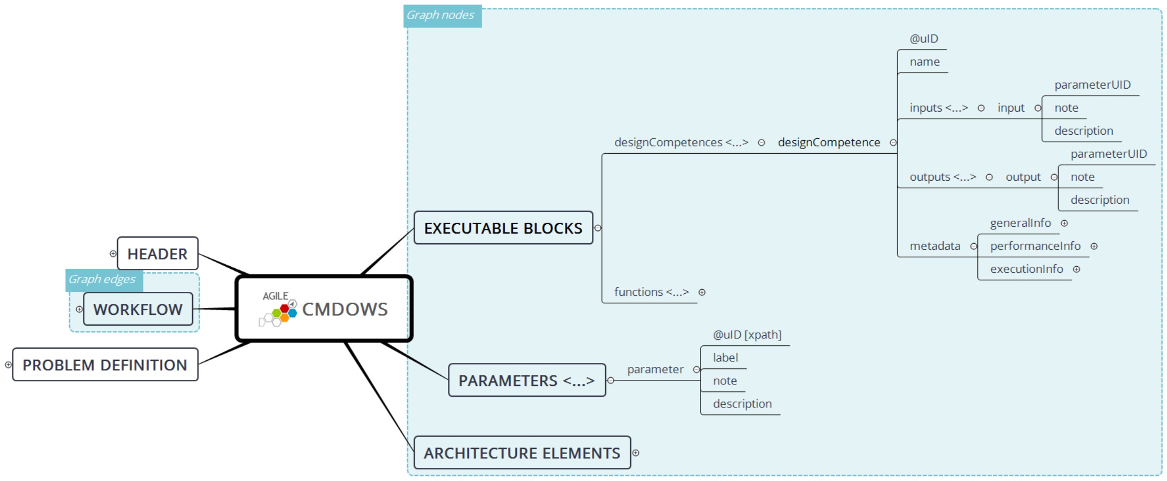


Figure 2.8: CMDOWS data structure for MDO workflows

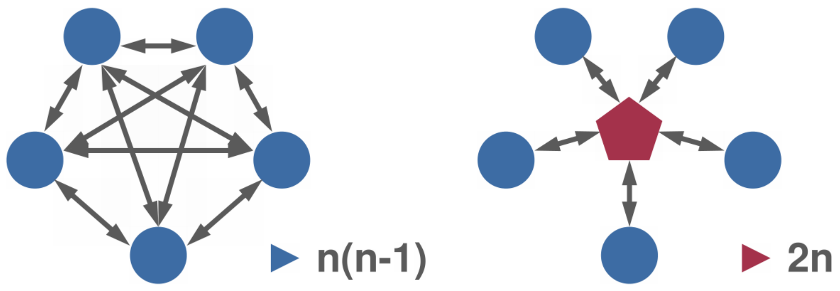


Figure 2.9: Central model approach [5]

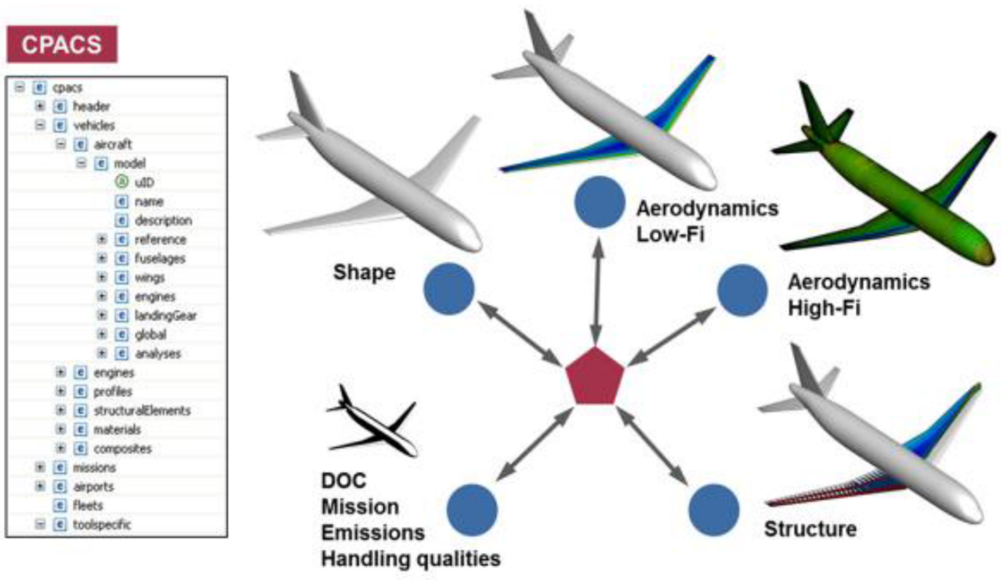


Figure 2.10: Centralized CPACS data structure for Multi-Disciplinary Frameworks [34]

3 Enabling Methodologies and Technologies

In this chapter enabling methodologies and technologies are presented and analyzed. The purpose of this chapter is to lay the foundations of the work that is done in this thesis. In this chapter the author will answer the following questions:

- *How can a reduction in the design and optimization process lead-time be achieved by incorporating lean information management strategies?*
- *What are the key principles the collaborative environment developed during this research must comply to such that waste in the business process activities is minimized?*
- *What knowledge technologies exist which enables the storing and writing of knowledge in an interchangeable format such that knowledge and information is able to 'flow' between different integrated systems?*
- *What is the best way to visualize, represent and share product and process modeling knowledge in the context of this research?*
- *What is the ontology used in the context of this research to model product and process models? And how is this ontology extendable to allow for model verification?*

This chapter first discusses the topic of information management in section 3.1. Next, in section 3.2 various methods to share, exchange and store knowledge are presented and analyzed. Finally in section 3.3 product & process modeling languages and ontology are discussed.

3.1. Information Management

During the design and optimization process a lot of knowledge is gathered and data is generated. But more importantly, all activities performed in the design and optimization process involve structure and coordination of many different actors in a cross-organizational environment. Given the high dependence of knowledge within the organization, the management of information is critical in order to operate efficiently and effectively [14]. This requires the development of an information system, capable of managing information.

Information management considers the activities involved with the creation, organization, maintenance, visualization, reuse, sharing, communication and disposal of information [14]. The generation of information can be considered valuable. Of course adding value within an organization is always bound by time and cost. Moreover product quality is a driving factor within an organization. Together time, cost and quality are considered as typical Key Performance Indicators (KPI) on a PDP level [24]. In order to minimize time and cost whilst designing a product with high quality lean principles can be used. Lean principles imply eliminating all waste and unnecessary actions in the steps which are used to add value. Waste can be categorized as activities which are required to overcome missing or wrong information. Moreover, waste can imply waiting on information, information excess and excessive activities needed to overcome for transport or bad exchange of information.

To demonstrate the concept of lean information management the value-flow model is shown in figure 3.1. This model can be applied to any information system or information processing activity. An example of an information processing activity is the visualization of an inspectable MDO workflow based on a certain set of data or the sharing, documenting and managing requirements. Both types of information are valuable to various end-users of the collaborative environment which is the goal of this research. The value-flow model describes the process of value flow which is induced by an end-user requesting information. This value flow describes the process of capturing raw data or information and transforming it in a way that it has maximal value to the end-user. Information management considers the value of information, the value stream, information pull and the flow of information. These can be described as follows:

- **Value**: Information and functionality must supply value to the information consumer or end-user. In a lean process, value needs to be defined from the customer's perspective. For example in a PDP a specific product needs to be developed within a specific time.
- **Value stream**: The value stream is the process of transforming information into value for the customer. In a lean process this value stream is formalized for a particular product or family of products. In the PD layer a lot of actors are involved. In a lean information process actors should be enabled to operate the framework by minimized waste. Waste can be reduced by integrating different design activities in a single system and automation of information intensive processes [14].
- **Value flow**: As information is seldom available in one simple step, a series of activities are required to transform information. In a lean process it is important that information flows efficiently between different systems and end-users. The value flow can be improved by ensuring that information is available real-time and up-to-date. An example of applying lean flow principles to PDPs is the shift from traditional sequential to concurrent processes. Concurrent engineering reduces waste through integration of multiple design disciplines and upstream and downstream functions involved in the product and process design activities [37].
- **Pull**: In a lean information system it is important that information only flows when it is required by an end-user. In the MDO framework development process, information flow is therefore triggered by a customer's specific need. All steps in the development process require information and trigger the generation of new information. When different actors in the development process perform their work they must pull only the information they need to perform their work. Other type of information implies waste.

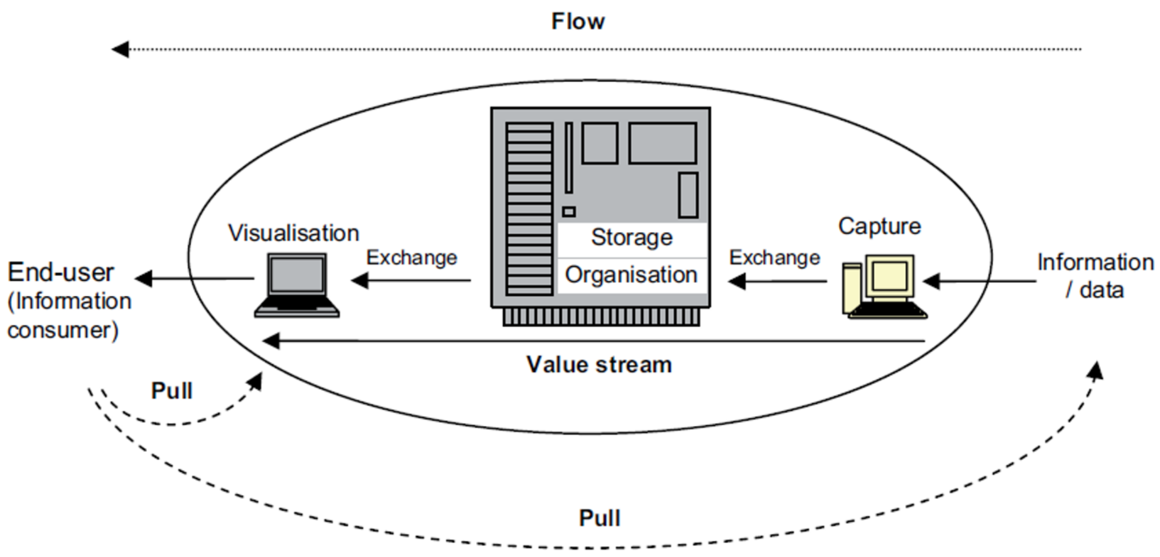


Figure 3.1: The value-flow model as applied to information management [14]

Studies in the field of information quality enable the evaluation of information quality and their underlying models. Examples of information quality categories are relevance, completeness, ease of understanding, interpretability, traceability and ease of operation [45]. For more examples and improved explanation of all information quality categories the reader is referred to appendix A.

Adopting lean principles in the PD layer of the collaborative environment to support the setup of collaborative and distributed MDO frameworks is essential. Mitigating waste and managing the flow of information adds value to the information generated in the design and optimization process. This thesis work aims to speed up the setup phase of the design and optimization process through mitigating waste in the information intensive processes. Mitigating waste through integration, formalization and automation of information management activities leaves more time for exploring different design solutions. To summarize, lean principles applied to the PD layer of the MDO framework knowledge architecture implies:

- Automate repetitive information processes, such as input/output coupling of design competences to a common data model in favor of manual XML editing.
- Integration of architectural components in which the information interfaces are automated.
- Formalization of knowledge through forms and information models, such that information is stored in a single format throughout development processes. This favors ease of understanding, interpretability, traceability and ease of operation.
- Improve the accessibility of information in the cross-organizational environment. Only relevant information should be provided when needed based on up-to-date information.
- Assist in the setup of correct information and data models. This helps to ensure information is complete and consistent.

3.2. Knowledge technologies

In this section knowledge technologies are discussed because of the need to formalize knowledge, share knowledge, transform and exchange knowledge in the development process of 3rd generation MDO frameworks. The first question which obviously arises is: *what is knowledge* and *how does it play a role in PDPs and the MDO framework development process*?

3.2.1. What is knowledge

Knowledge is considered to be a fundamental resource of every organization. Hence, it is generally agreed upon that knowledge is an enabler for businesses and organizations in the 21st century to retain their competitive advantage in the engineering market [16, 17]. Various definitions of knowledge exist as displayed in figure 3.2. From these definitions it becomes evident that knowledge plays an important role in decision making, solving of problems and to perform skillfully. In the collaborative environment which is developed during this research, these knowledge roles are just as apparent for the various actors involved in for example the development process of 3rd generation MDO frameworks as for other type of PDPs.

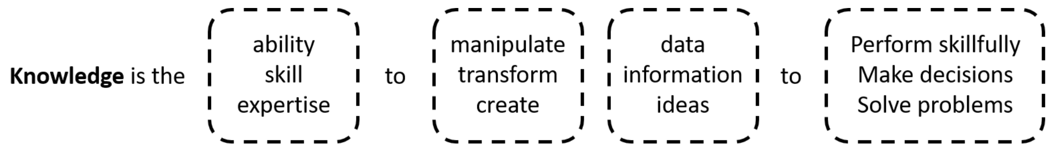


Figure 3.2: What is knowledge, some examples [22]

There is a big analogy and overlap with information management, discussed in section 3.1, and knowledge management. There is often not a clear understanding of the actual differences, mostly due to missing distinguishable definitions [6, 12]. Engineers working with design and optimization processes tend to be more prone on using terms coming from knowledge management. In the context of this research, information management involves the process of providing the interfaces for making information accessible to various end-users. Knowledge management in general focuses more on the know-how, know-why, and know-who compared to information management being more fact based. Therefore it can be said that knowledge management aims more and identifying rationale which can have a more lasting impact over the course of several projects. For this reason knowledge management is an important part of accelerating PDPs through front-loading, due to project to project knowledge transfer [38].

3.2.2. Semantic Web technology stack

Knowledge technologies is a summary of specifications and solutions used to store knowledge, transform knowledge and exchange knowledge. A good overview of the available knowledge technologies can be seen in the Semantic Web Technology Stack in figure 3.3. This overview shows various solutions for different concepts and abstractions. Usually an application only uses a subset of the visualized stack. The same goes for the application developed in this research.

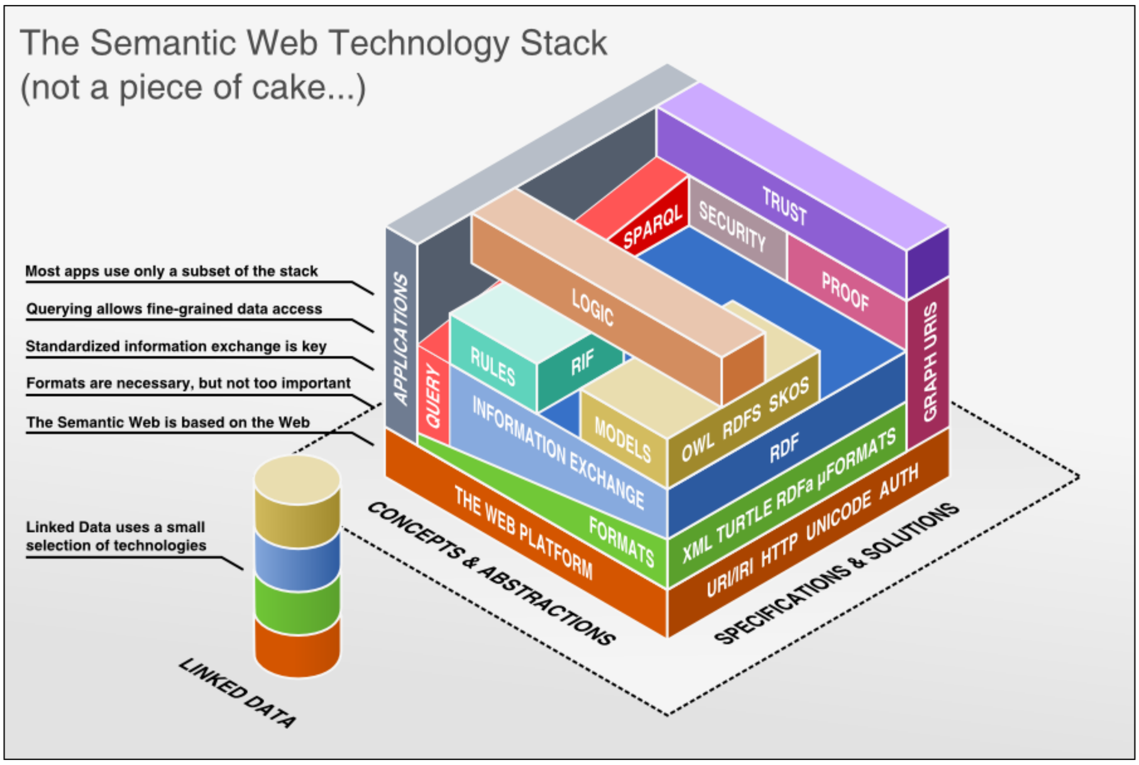

The Semantic Web Technology Stack
(not a piece of cake...)
Most apps use only a subset of the stack
Querying allows fine-grained data access
Standardized information exchange is key
Formats are necessary, but not too important
The Semantic Web is based on the Web
Linked Data uses a small selection of technologies
LINKED DATA
APPLICATIONS
QUERY
SPARQL
SECURITY
TRUST
PROOF
LOGIC
RULES
RIF
MODELS
OWL RDFS SKOS
RDF
INFORMATION EXCHANGE
FORMATS
XML TURTLE RDFa μFORMATS
THE WEB PLATFORM
URI/IRI HTTP UNICODE AUTH
GRAPH URIS
CONCEPTS & ABSTRACTIONS
SPECIFICATIONS & SOLUTIONS

Figure 3.3: Semantic Web Technology Stack [26]

During this research an integrated web-based application is developed extending KE-chain. This application operates on the PD layer of the MDO framework knowledge architecture. Integration of components from the AD layer of the knowledge architecture, require the use of knowledge technologies to enable knowledge transfer. Hence, solutions for various concepts in the Semantic Web Technology stack were explored on the concept of web platforms, knowledge formats, information exchange and models. The next section describes the various knowledge technologies in more detail.

3.2.3. Storing and writing knowledge

The research presented in this paper involves a lot about managing information and knowledge which is required for integrating various components in the PD layer of the framework knowledge architecture. Therefore this section introduces various solutions used to store and write knowledge. Below a summation of solutions coming from the Semantic Web Technology stack and others are presented. These technologies describe in more detail solutions to format information using XML, information exchange using Resource Descriptive Framework (RDF), storing information using Relational Databases (RDBs) or triple stores.

- **Extensible Markup Language (XML)**: According to Milton [22] 'XML is a web standard for storing meaning full information about a group of concepts or about the contents of a document'. XML documents enable capturing of hierarchical structures such as product models. The use of XML schemes enables the validation and generation of information structures for many different purposes. For example product and process related knowledge, ranging from abstract model information to meta-information about a DC's level of fidelity or execution time. An example using two DCs for which information is stored in XML format is given below:

```
<designCompetences>
    <designCompetence uID="Discipline1">
        <name> Aerodynamic Analysis </name>
        <metaData>
            <fidelity> 1 </fidelity>
            <executionTime> 16 </executionTime>
        </metaData>
    </designCompetence>
    <designCompetence uID="Discipline2">
        <name> Structural Analysis </name>
        <metaData>
            <fidelity> 2 </fidelity>
            <executionTime> 24 </executionTime>
        </metaData>
    </designCompetence>
</designCompetences>
```

Using this structure two DCs are defined which can be seen from two opening tags < **designCompetence** > and two closing tags < / **designCompetence** >. In each opening tag, attributes can be assigned. Attributes help in making sure information structures with similar categories can be distinguished from one another. In this example, an attributed name **uID** is added for each **designCompetence**. Using the logic of tags, attributes and nesting tags allow us to navigate and retrieve information using various XML parsing modules for example a Python module called ElementTree [1].

XML formatted documents also allow for direct querying of individual attribute values through XPaths. An XPath is a a path which follows the hierarchical structure of the XML document. In order to find for example the execution time of the Structural Analysis tool defined in the XML document above can be done using the XPath shown below. Note that the **uID** attribute is used to distinguish between the two DCs.

```
/designCompetences/designCompetences[Discipline2]/metaData/executionTime
```

- **Relational Databases (RDBs)**: A RDB is based on a relational model of data. It is organized in one or more tables, where each table represent an entity type. Every row of a table has a unique ID and corresponds to an instance of that entity type. A row of an entity is called a record, or tuple. The columns in the table can be used to assign values to attributes of an instance. A table can also be seen as a set of tuples which have the same attributes, hence a table is also called a relation. A RDB uses Structured Query Language (SQL) for querying.

- **Resource Descriptive Framework (RDF)**: RDF is a standard format of XML for describing resources. In RDF, triples are used to describe a statement. This triple consists of three parts: the subject, predicate and object. The triples of which RDF is build up can be illustrated using an RDF graph. All relations in this graph can be captured in the triple structure such as Batman (subject) lives in (predicate) Gotham City (object).

  RDF is part of a collection of non-relational database systems. This collection is referred to as NoSQL ("Not only SQL"). NoSQL solution can be categorized in four categories [3]: Key-value databases, document databases, wide-column databases and graph databases. RDF is the largest subset of the graph database NoSQL solution. According to Bendiken [3], RDF data maps well to object-oriented programming paradigms and to RESTful (REpresentational State Transfer) architectures.

- **Triple Stores**: In a semantic system, a triple store database is a form of knowledge base which has been optimized for storing triples. According to Milton [22], 'the triple store in a semantic system is

[1]`https://docs.python.org/2/library/xml.etree.elementtree.html` (accessed: 14-05-2017)

usually created and updated by compiling RDF to create the triples. Queries can be made on the triple store, or on the RDF, using a special language called SPARQL'. SPARQL provides a standardized and interoperable query language, which is understandable even for non-programmers, whilst have at least equal capabilities compared to SQL [3].

The developed collaborative environment to support in the setup of 3^{rd} generation MDO frameworks is build through extending KE-chain. KE-chain makes use of a PostgreSQL[2] RDB. This RDB enables the definition of relational models. This RDB is used to store all data in the KE-chain database using unique identifies for each record. Records of a model can refer to other models using foreign key objects. An example of such a foreign key relation, emphasizing the strength of the RDB, is storing information regarding a KE-chain task configuration. Here a task is related to various input and output properties. Each property has its own record of type Property, which is related through foreign keys to a task configuration. These relations are explained in more detail in section 3.3.3. Exploring these relations play an important role in the methods used to evaluate the coupling of the product model and process model. The product model is a collection of part models and property models which are all related to parent child relations. The process model is a collection of tasks.

When shifting our attention to processing information outside KE-chain, information formats are required which can be read by multiple information systems and applications. For this purpose XML formatted documents are used. The strength of XML is a lightweight format to store a large amount of information. In an approach to adopt principles of lean information management in the transfer of knowledge, standardization is a key aspect in minimizing waste. Using XML schema validation, a common language can be defined and used within the organization. In this research two schema validations are used: CPACS [3] and CMDOWS [4]. Schema validation and XML technologies are used during this research in the integration of external services using KE-node. KE-node enables writing services in Python scripts. Hence the use of Python's module ElementTree is used for reading, writing and validating XML CPACS and CMDOWS documents.

In the KE-chain application the Application Programming Interface (API) which formats all data stored in KE-chain makes use of a RESTful architecture called Django REST framework. Advantages of the Django REST framework is a web-based interface which enables a seamless and customize integration of KE-chain's Ext JS 6 front-end and Python backend. All developed extensions developed by the author require an extension of this REST API, to ensure all defined backend models can be used by the various users through the frontend of the KE-chain application.

In order to define information models its relations and concepts used need to be formalized. For this an ontology-based representation is well suited. An ontology can be used to express the terms, entities, classes, objects and the relationships between them for a particular domain and axioms to constrain the meaning of defined terms [40]. Hence, an ontology can be seen as a common language for representing knowledge in a domain. It is an enabling technology, which in the context of for instance Knowledge Based Engineering (KBE) can be used to define what knowledge to capture, and how to represent it [22]. According to Wang *et al.* [46] there are several reasons for developing context models, which define how context data is structured and maintained, based on ontology:

- **Knowledge Sharing**: The use of ontologies enables computational entities such as agents and services in pervasive computing environments to have a common set of concepts about context while interacting with one another.
- **Logic Inference**: Based on ontology, context-aware computing can exploit various existing logic reasoning mechanisms to deduce high-level, conceptual context from low-level, raw context, and to check and solve inconsistent context knowledge due to imperfect sensing.
- **Knowledge Reuse**: By reusing other well-defined Web ontologies, new context ontologies can be generated without the need to start from scratch.

A well established ontology-based modeling language is the Web Ontology Language (OWL). OWL is used for the creation and sharing of ontologies on the World Wide Web. OWL is part of the W3C's Semantic Web technology stack [13]. OWL is modeled using an object-oriented approach, using classes and properties to describe the structure of a domain. OWL can use the existing body of Description Logic (DL) reasoning including class consistency and consumption and other ontological reasoning [46].

[2]`https://www.postgresql.org/` (accessed: 14-05-2017)

[3]`https://github.com/DLR-LY/CPACS/blob/master/schema/cpacs_schema.xsd`

[4]`https://bitbucket.org/imcovangent/cmdows/raw/master/schema/0.4/cmdows.xsd` (accessed: 14-05-2017)

3.3. Product and process modeling

In a product driven design and optimization process various underlying abstraction models can be identified. These abstractions relate to the product that is to be designed or optimized and the workflow used to perform the analysis. Traditionally a set of attributes defined in a design and optimization problem is referred to as a data model. In a design and optimization process to support product development this data model consists of all properties describing a certain product. For example, an optimization problem in which a wing structure is analyzed based on aerodynamic and structural performance, the data model is composed of wing geometric parameters (such as: wing span, sweep, taper), analyses parameters (such as: lift & drag coefficients, bending moments, design masses), environmental variables (such as: cruise velocity, air density, temperature) and optional settings required to run the simulation. These optional settings might include load factors or for example boundary layer separation settings. As all these parameters together act as a blue print for the resulting optimized product. This blue print is referred to as the product model.

In the following sections first the topics of product modeling and process modeling are explained in more detail. This is necessary to introduce the concepts on enabling verification on the coupling of the product and process models.

3.3.1. Product modeling

The product model is a special type of data model which is a definition quite commonly used in the field of KBE. A product model is a central part in a KBE system. It contains geometry, configuration and engineering knowledge on how a certain product should be developed in the form of rules [32]. KBE systems are characterized by using Object-Oriented Modeling (OOM) techniques [9, 32]. OOM makes use of concepts defined as classes and objects. An example of a UML class-object diagram of a simple aircraft is shown in figure 3.4. This class-object diagram shows composition and specialization dependencies. For example, the aircraft is composed of a main wing, fuselage and a t-tail. This generic composition of the defined aircraft is captured in the class diagram. This class diagram shows the aircraft main components, and all attributes of these components. For example, a wing has an attribute span. Moreover it shows the amount of classes of which a class is composed. For example, it can be seen that a t-tail is composed of two wings, one for the vertical tail and one for the horizontal wing. Based on this class diagram, an object can be defined. The object follows the compositions defined in the class diagram and inherits all its attributes. The object however, now has values. Using these class-object relations allow us to define multiple versions of aircraft, based on a single “model”.

In the context of this research, product models are setup using a KE-chain definition. The definition of KE-chain product models is very much similar to the class-object diagram as can be seen in figure 3.4, however it uses different terminology adopted during this research. The KE-chain product model uses parent-child relations similar to the earlier defined composition relation. As the reader could have read in section 2.4.1 the product model is part of the PIM domain in the KE-chain application architecture. A simplified class definition of parts and properties used in the domain of PIM is illustrated in figure 3.5. In this figure a class definition of a part assembly. A part assembly is a composition of several parts. A parent part is composed of none or several children parts. A part is composed of zero or infinite properties. The different attributes of the part and property classes contain a Unique Identifier (ID) and the name of the part or property. The property class has additional attributes to allow for the specification of a property type such as float, integer or text property and an attribute to store a value. A part class has two other important attributes: category and multiplicity. This category is used to distinguish between a part *model* and a part *instance*.

A part model is used to define the product model, the blueprint of a product instance. A product instance in analogy with an object versus a class is derived from a product model definition. The multiplicity is used to indicate how many instances an instantiated part can potentially be composed of. This is similar to saying that a tail is composed of a multiplicity of two wing models. The relation between the product model and the product instance is shown in more detail in the next section. Finally a part object has a publisher attribute, which is an extension of the part class by the author used to perform the storing of modeling knowledge in a centralized library. This publisher concept however, will be explained in more detail in section 4.3

Using the parent child relations, a product model is constructed such that it follows a hierarchical or tree-like composition as can be seen in figure 3.6. This figure shows on the left side a product model definition of the simple aircraft defined earlier in the class-object diagram, and on the right side two instances of that aircraft model. Every instance of this aircraft model has a tail containing a horizontal tail and a vertical tail, this is captured in the product model definition. However, as the main wing and both the vertical and hor-

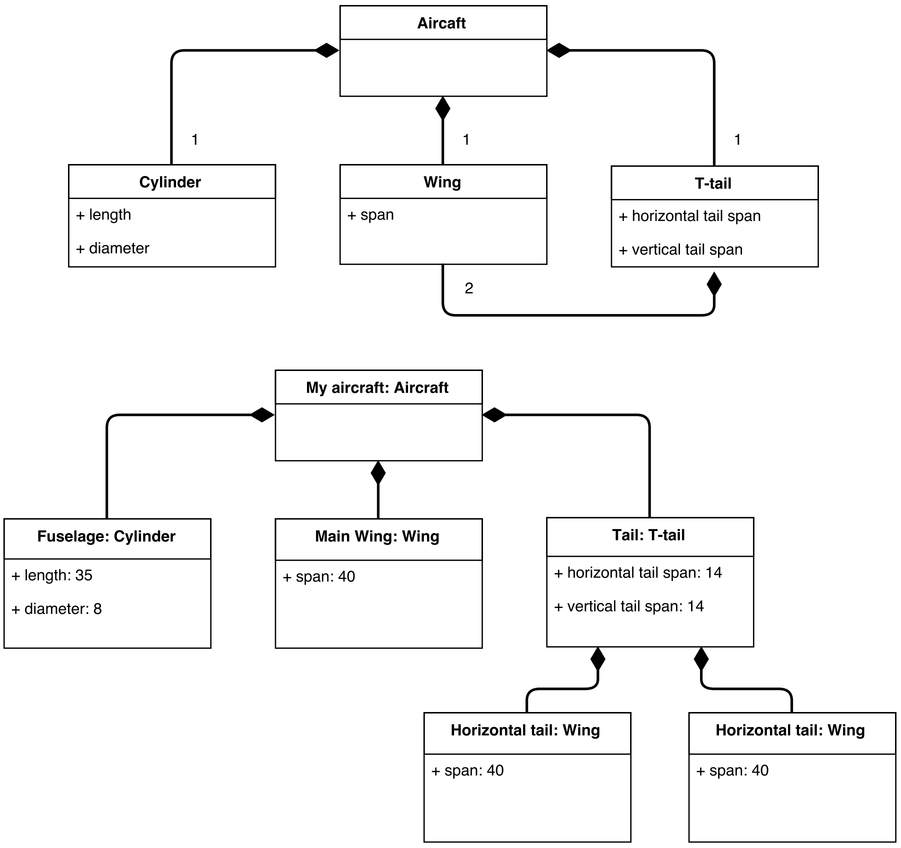


Figure 3.4: Example of a UML class (top) and object (bottom) diagram

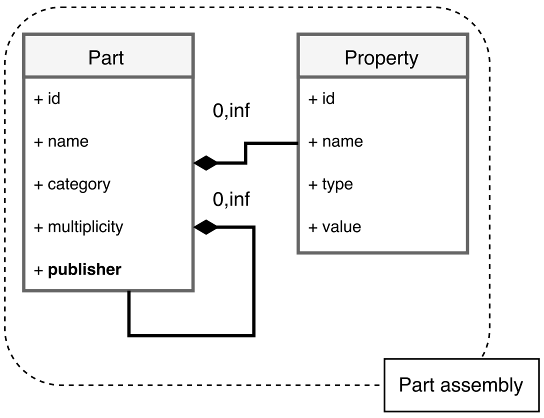


Figure 3.5: KE-chain ontology used to define parts and properties

izontal tail are derived from a single wing model, a proxy relation can be made. This proxy relation enables the generation of dependent models, which inherit all properties of a proxy model. The use of a proxy models shows great similarities with the concept called aggregation in UML.

The next step is to define a definition for the process model. This process model is used to define activities which are used to read and write information from the product instance. The process model is explained in further detail in the following section.

3.3.2. Process modeling

In order to capture the flow of data or information process models are used. A process model can be used to visualize a process, or assist in the execution of one. In the knowledge architecture presented in section 2.4

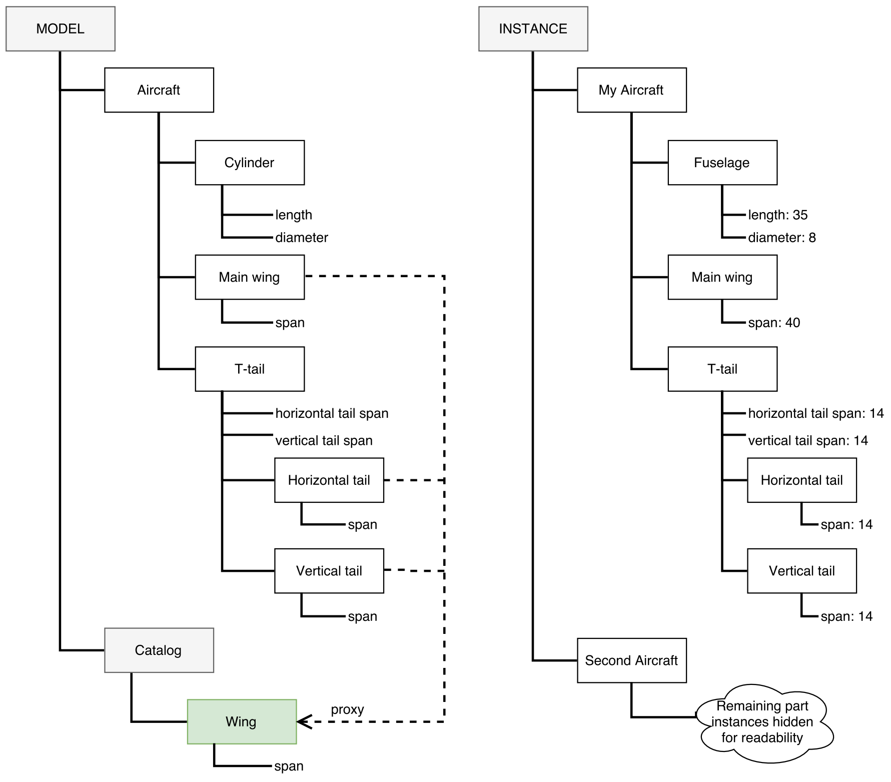


Figure 3.6: KE-chain ontology of defining a product model and its instances

different types of integrated processes can be identified. This follows from the hybrid implementation of the MDO framework with the subdivision between the PD layer and the AD layer. The PD layer, which is the main context of this thesis research, makes use of business processes. A business process is used to organize the organizational activities to control and analyze operational processes involving various actors (end-users), documents, applications and other sources of information [30]. However, also means are investigated to enable testing of compliance and consistency of a formulated process used to perform MDO. This requires investigation of different means to represent process related information for both the purpose of controlling and organizing organizational activities, but also more complex processes involving solely analysis tool integration. Below an overview of available representations of various type of process models is presented:

- **Business Processes**: Ability to show sequence flow of business processes. Here Business Process Modeling and Notation (BPMN) is currently regarded as the graphical standard when it comes to modeling business processes and focuses on process design [30].
- **Design Structure Matrix (DSM)**: One of the most commonly used way of representing a process model is the Design Structure Matrix (DSM), also referred to as N^2-charts. According to Browning [7], ‘a DSM displays the relationships between components of a system in a compact, visual, and analytically advantageous format’. A DSM succeeds in defining a framework which is able to handle dependencies and relations between elements, such as tasks within a PDP but also parts of which a product is decomposed.
- **Data Flow Diagram (DFD)**: A DFD is a graphical representation of the "flow" of data through an information system, modelling its process aspects. DFDs can be used to provide the end user with an idea of what the impact of changing information is with respect to the whole system. However, a DFD does not say anything about the order in which the activities, processes or sub-processes are executed. A DFD is a representation of the dependence of information as it currently is.
- **Graphs**: A process representation which consists of nodes and edges, which together form a graph. According to Pate *et al.* [28] ’one of the advantages of a graph theoretic approach is the standard algorithms which can be used to inform the user’s decisions, such as cycle detection, minimum spanning

trees, and shortest path algorithms'. Although a lot of ingredients such as properties and their dependence are stored in the coupled product and process model, a graphical interface using graphs quickly becomes complex.

- **Extended Design Structure Matrices (XDSM)**: The reasoning behind the XDSM comes from the need to develop a standard visual representation of an MDO solution process [19]. This has to be done in such a way that both the data connections as the MDO algorithm could be represented in a single diagram. The key guiding principles that the design appeals to are simplicity, clarity, information density and integration of mathematics. Strength of the XDSM is its ability to contain both the sequential order as well as data dependence. In process of integrated tools, this sequential order represents the sequence in which tools are executed. The XDSM is able to show both feed-forward as feed-backward loops.

In this thesis work, the functionality of the PD layer is developed using BPMN business processes. The collaborative environment which contains the PD layer functionality should support both the orchestration and choreography of the business process [29]. Orchestration represents control over the executable business process in which the different services are linked with one another. Choreography refers to the information which is exchanged between different services. The PD/AD layer interface handles the choreography, whilst the business process containing both human and automated tasks, facilitate in orchestration of the process.

Extending the BPMN business process, this research makes use of IDEF0 principles to extend the information which is being processed by each individual task. The IDEF0 (Integration DEFinition for Function) task description can be seen in figure 3.7. The IDEF0 is commonly used to provide extra information on the process model [25]. IDEF0 is a method to capture information on the input, output, control and mechanism of a task in the business process:

- **Input** refers to the information or data required to execute that task and produce a certain **output**. *For example: cooked spaghetti requires as input: water and raw spaghetti*

- **Controls** constrain and direct activities. *For example: the activity of cooking spaghetti is regulated by a prescribed portion per person and cooking time*

- **Mechanisms** define the actor, required tools and other supporting aspects of an activity. *For example: the cooking process requires a pan, heat, scale and a chef*

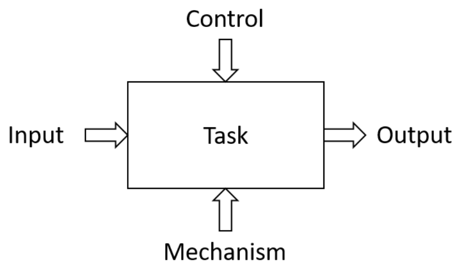


Figure 3.7: Basic IDEF0 representation of a task and its related information [11]

Besides a controllable business process to support the PD layer functionality in the MDO framework, another requirement is the visualization of intermediate design results. Part of the PD layer involves the formulation of the MDAO workflow. Hence this MDAO workflow needs to be visualized in the PD layer for continuous inspection of the selected MDO workflow used for execution. The use of DSMs and XDSMs is widely accepted in the field of MDO [19]. Hence, to support the architect to formalize an MDO workflow the PD layer of the MDO framework development process is supported with with DSM and XDSM previews. This requires extension of the control feature of tasks in the third step of the MDO framework development process to support the following:

- Provide a controllable interface in which formulated MDAO workflows can be formulated through the use of external KADMOS services.

- Extend the configuration of tasks to display the resulting DSMs and XDSMs in an inspectable and embedded preview container

- Develop a mechanism which allows for the updating and replacing of MDAO workflows visualizations when new input is provided.

- Enable the configuration of tasks to be maintainable by a collaboration engineer.

The requirements listed above are included in this research to improve PD functionality. The solutions and implemented methods to support these features are presented in section 4.2.1.

Finally the completeness and consistency of a repository of tools need to assessed prior to the definition of a knowledge base to support the formulation of MDAO workflows. This requires the definition of the inputs and outputs of tools related to a common data model and assessing whether tools are able to be connected in a single framework. This requires inspection of a process model on data level. The definition of a framework in which only the connectivity of tools is tested, sequential order should not be taken into account. For this purpose both directed graphs and DFDs are suitable. The implementation of a process model inspector is further explained in section 4.2.2.

Due to the close coupling of product and process models sought after in this research methods to achieve this are discussed in the following section.

3.3.3. Relations between product and process models

In this section the relations between product and process models in the context of this research is presented. Each task which requires transformation of input properties into output properties can be related to a uniquely defined product model. An example of such input and output mapping can be seen in figure 3.8. In this figure one can see on the left side a simple aircraft product model defined in the domain of PIM. In this product model various parts are defined of which the aircraft product model is composed of. For example, the aircraft model is composed of a wing, design masses and aerodynamics. Each part has an arbitrary amount of properties describing that part. The aircraft wing for example might have properties to describe its plan-form shape, or airfoil positions etc. On the other hand the aerodynamic characteristics of the aircraft can be defined using properties which describe the aerodynamic coefficients, load distribution etc. On the right side of the figure one can see the domain of WIM. In this domain two tasks are drawn: aerodynamic analysis and structural analysis. According to the IDEF0 definition, this tasks requires a set of properties as input and produces a set of properties as output. In a system in which the product and process models are coupled, this set of input properties and output properties is a subset of PIM. Mapping the inputs and outputs of a tasks to a subset of PIM enables the use of a a common product model in which uniqueness of all defined properties can be guaranteed. This makes it possible to evaluate the coupling of PIM and WIM, which is a great enabler of the model verification module, described in section 4.2.2, which is developed by the author in context of this research.

The coupling of PIM and WIM is a concept which is defined in KE-chain using the following class definition displayed in figure 3.9. In this class definition one can see the earlier defined class definition of PIM, extended by an association class. This association class object is used to bridge between PIM and WIM. An association is an object which defines an input property or output property for each activity object defined in the domain of WIM. Therefore an association object is a unique object which relates both to a property and an activity. Furthermore each project has a single process object.

Outside of the scope of KE-chain, the coupling of product and process models needs to be communicated with the other components operating on different levels in the knowledge architecture of the MDO framework. For example, the formation of MDAO workflows using KADMOS also requires the coupling of all properties of the product model to a formulated MDO process model. In the context of this research and AGILE the common data model used to connect all DCs and tools in the executable simulation workflow follows the CPACS schema. This CPACS schema can be used to generate a product model definition composed of various part compositions each with various properties. As for simulation workflows the hierarchy of product models is irrelevant, the hierarchy of the different exchangeable properties is stored using XPaths. As the author described in section 3.2 XPaths are used in XML technologies to refer to a unique attribute which is nested in an XML document. For example, the inputs and outputs of the simple aerodynamic tool described in figure 3.8 can be described by the following set in equation 3.1:

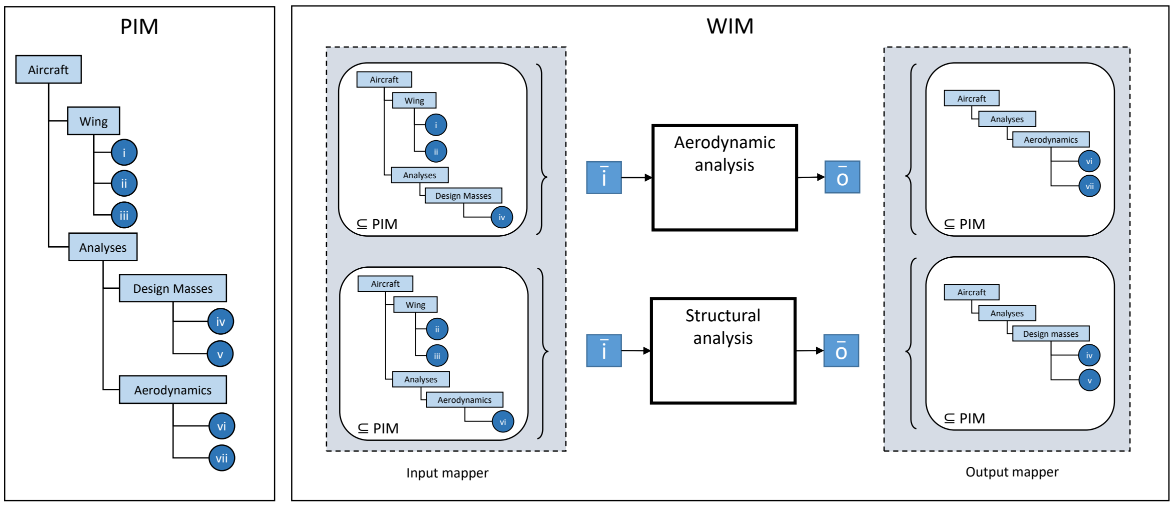


Figure 3.8: Coupling of the inputs and outputs of tasks in WIM with subsets of PIM

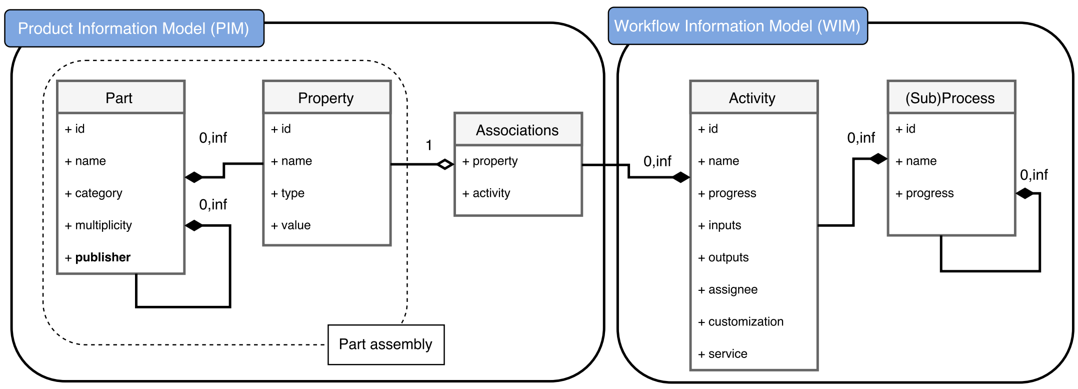


Figure 3.9: Coupling of PIM and WIM UML class definition

$$
\begin{aligned}
aerodynamicsTool_{input} &= \begin{bmatrix} /\mathrm{Aircraft}/\mathrm{Wing}/\mathrm{i}/ \\ /\mathrm{Aircraft}/\mathrm{Wing}/\mathrm{ii}/ \\ /\mathrm{Aircraft}/\mathrm{Analysis}/\mathrm{DesignMasses}/\mathrm{iv}/ \end{bmatrix} \\
aerodynamicsTool_{output} &= \begin{bmatrix} /\mathrm{Aircraft}/\mathrm{Analysis}/\mathrm{Aerodynamics}/\mathrm{vi}/ \\ /\mathrm{Aircraft}/\mathrm{Analysis}/\mathrm{Aerodynamics}/\mathrm{vii}/ \end{bmatrix}
\end{aligned}
\tag{3.1}
$$

Using XPaths as unique identifiers for properties or attributes exchanged by different tools, transformations between product and process models can be easily made depending on the system's requirements for the various integrated software solutions in the collaborative environment. The XPath definition to determine input/output properties is also used in the definition of the CMDOWS schema used to store the MDAO workflow definition in a neutral and interchangeable format.

All methods, technologies and definitions defined in this chapter form the foundation of the implementation of the collaborative framework to support the setup of 3rd generation MDO frameworks described in the next chapter.

4 Implementation

In this chapter the implementation of the solutions required to support the integration of all business process activities leading to the development of a 3^{rd} generation MDO framework in a collaborative environment is presented. The implementation focuses on extracting information from the lower layers in the knowledge architecture and providing an interface to the customer, system architect, specialists and integrators in the PD layer. This interface must be developed such that it follows the five main steps identified in the MDO framework development process. The implementation of this collaborative environment requires the development of the PD/AD interfaces and the development of KE-chain capabilities by the author following the required extensions presented in section 2.4.1 to support the PD layer functionality.

This MDO framework development process is part of the setup and operation phases of the design and optimization process. These five steps are considered as the blue print for the business process which needs to be implemented. Integration of the business process activities requires the development of PD/AD interfaces to ensure the business process is able to perform as a closed system. In this integrated system the functionality of the lower layers of the knowledge architecture can be controlled through the GUI. Finally the implementation of the collaborative environment requires the development of a GUI in which the completeness and consistency of the framework setup can be inspected. Determining the completeness and consistency of the framework setup aims to answer questions such as: *Are the different DCs introduced by the discipline specialists able to provide, or make an impact with a set of key design parameters specified by the customer? Are DCs able to be coupled to a common data model and able exchange data with other DCs?*

These main topics can be categorized as integration, support and reuse. In the coming sections the implemented solutions on the topics of integration, support and reuse are further discussed:

- **Integration** of all elements in the framework and information layers
- **Support** human inspection of the MDO framework setup through the development of user interfaces which provide the means to manage the MDO framework's underlying product and process models.
- **Reuse** to maintain knowledge and enable reuse to accelerate future MDO projects

4.1. Integration

In this section the integration of the information layers in the MDO framework is discussed. An overview of the components integrated in the MDO framework can be seen in figure 4.1. The PD layer is used to provide a GUI for the different actors during the setup and operation of the MDO framework. The PD layer contains manual tasks, to acquire knowledge or control exposed functions of the AD layer, and automated tasks. These automated tasks are connected to the components in the AD layer through KE-node interfaces. KE-node provides a remotely controlled interface to external applications through easy maintainable Python scripts as introduced in section 2.4.1. In the remainder of this section the PD layer components and developed PD/AD interfaces are explained in more detail.

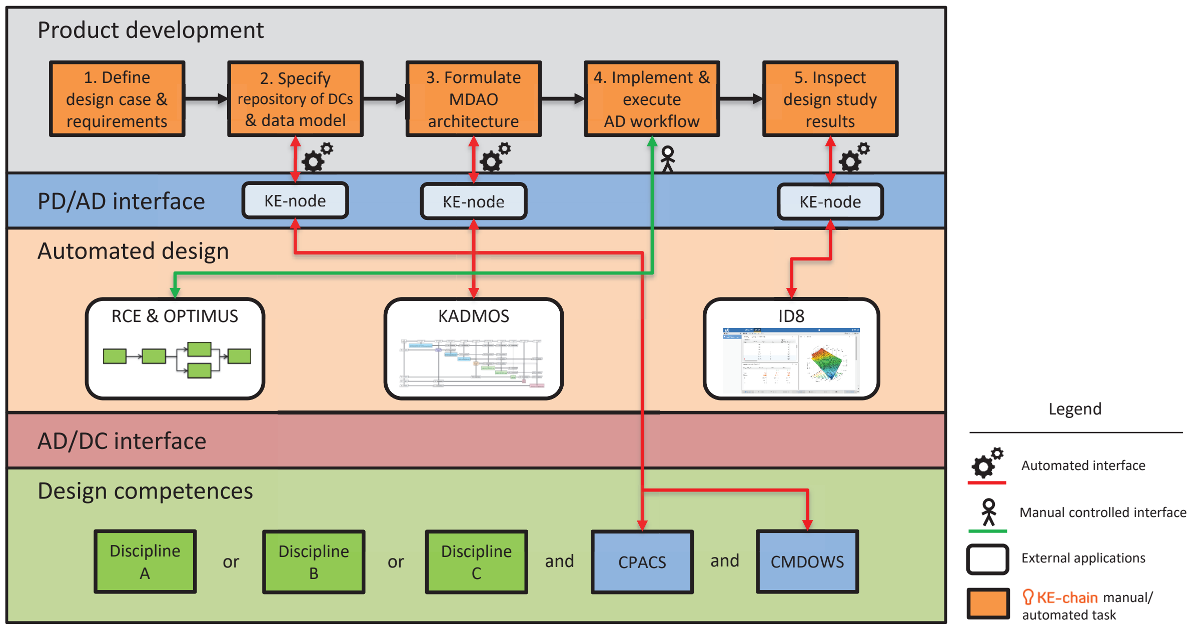

Product development
1. Define design case & requirements
2. Specify repository of DCs & data model
3. Formulate MDAO architecture
4. Implement & execute AD workflow
5. Inspect design study results
PD/AD interface
KE-node
KE-node
KE-node
Automated design
RCE & OPTIMUS
KADMOS
ID8
AD/DC interface
Design competences
Discipline A
or
Discipline B
or
Discipline C
and
CPACS
and
CMDOWS
Legend
Automated interface
Manual controlled interface
External applications
KE-chain manual/ automated task

Figure 4.1: Overview of the interfaces and components in the integrated MDO framework

To support the PD layer functionality a business process needs to be integrated in the PD layer which supports the actors in performing the five steps defined in the MDO framework development process. These five steps are each composed of various sub-tasks. An overview of the business process which is implemented by the author is displayed in figure 4.2. In this figure five sub-processes can be identified, each of which has various sub-tasks. Among these sub-tasks manual and automated tasks are included. Automated tasks can be identified by the gear icon in the bottom right corner of that task. The sub-process of step four is not displayed in further detail, as the automatic integration of executable simulation workflows was considered out-of-scope in this thesis research. A different representation of all business process activities integrated in KE-chain is visualized in appendix B in figures B.4 and B.5. Although the business process is displayed as a sequential process, in practise all activities are highly concurrent.

The formalization of this business process is the result of various design iterations between November 2016 and March 2017. The formalization of the business process steps occurred during a collaborative session held with a small selection of AGILE consortium members during a three day worksession hosted at the DLR in Hamburg on 7-8-9 November 2016. These committee members included the author, E.J. Schut [1], P. D. Ciampa [2] and T. Lefebvre [3]. The result of this meeting was the formalization of the five main steps identified as the AGILE framework development process [4]. In this report, these five steps are referred to as the MDO framework development process, as presented in section 2.3. In the following months the formalized steps were integrated by the author in KE-chain, a data model was setup to store all project information and the required PD/AD interfaces were developed.

[1]Chief Commercial Officer, KE-works
[2]Air Transportation Systems- Integrated Aircraft Design, German Aerospace Center (DLR)
[3]Department of System Design and Performance Evaluation, the French Aerospace Lab (ONERA)
[4]AGILE, "D65 AGILE framework architecture," AGILE project (H2020-636202), 2016

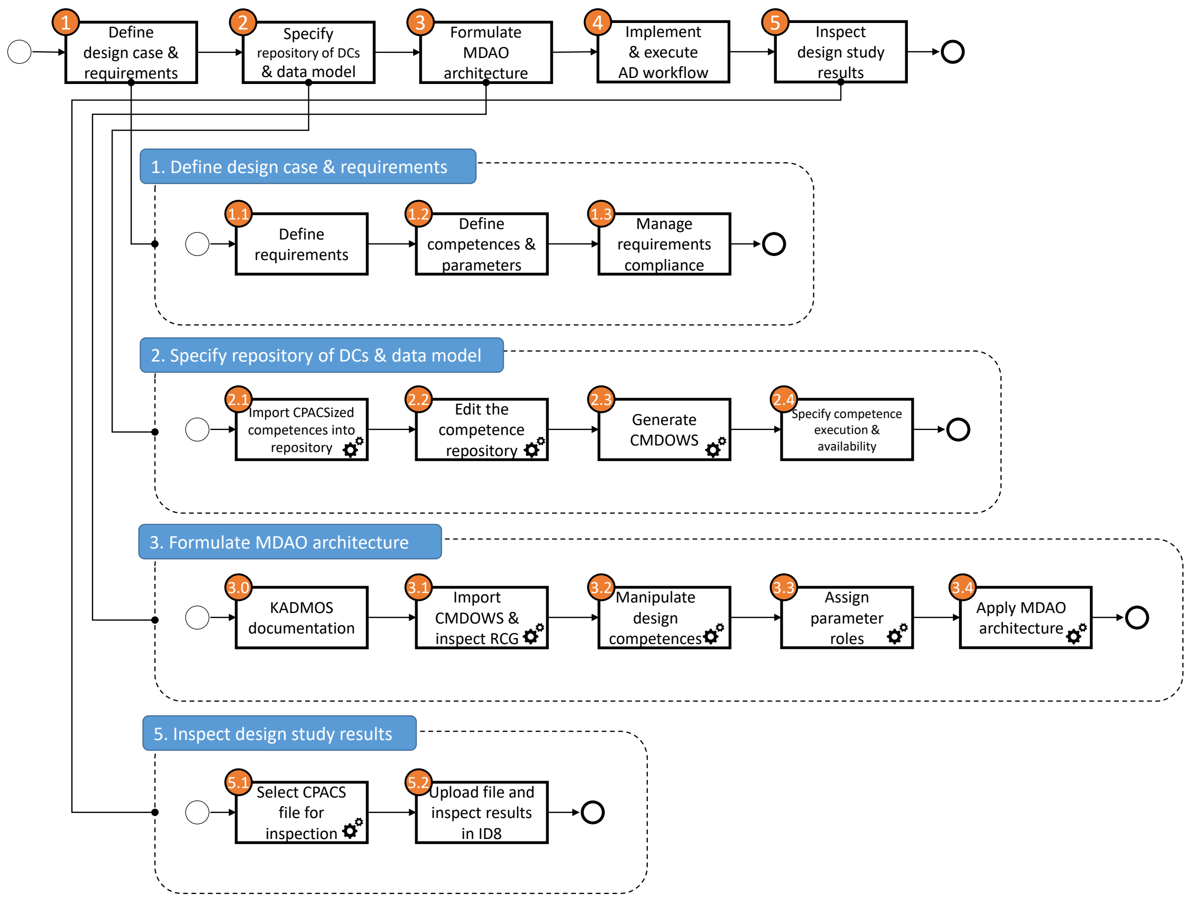

1 Define design case & requirements
2 Specify repository of DCs & data model
3 Formulate MDAO architecture
4 Implement & execute AD workflow
5 Inspect design study results
1. Define design case & requirements
1.1 Define requirements
1.2 Define competences & parameters
1.3 Manage requirements compliance
2. Specify repository of DCs & data model
2.1 Import CPACSized competences into repository
2.2 Edit the competence repository
2.3 Generate CMDOWS
2.4 Specify competence execution & availability
3. Formulate MDAO architecture
3.0 KADMOS documentation
3.1 Import CMDOWS & inspect RCG
3.2 Manipulate design competences
3.3 Assign parameter roles
3.4 Apply MDAO architecture
5. Inspect design study results
5.1 Select CPACS file for inspection
5.2 Upload file and inspect results in ID8

Figure 4.2: Detailed overview of the business process activities in the PD layer

The data model developed to store information generated during the business process activities can be seen in figure 4.3. This data model is composed of various classes, indicated as ellipsoids, and their attributes. Collections of classes are grouped according to the respective step in the MDO framework development process the information belongs to. For example, in step 1 the design case and system requirements are defined.

In the coming sections the business process is explained step by step. For each step the exposed PD layer functionality and required PD/AD interfaces are explained.

4.1.1. Step 1: Define design case & requirements

In step one of the MDO framework development process the design case needs to be defined. This is an operation which is performed by the architect together with the customer. An example of a design case definition formulated as a system requirement can be defined as can be seen in table 4.1. In this table one can see a set of requirements formalized together with a customer who wishes to setup an optimization workflow in which the maximum take-off weight of a benchmark aircraft is minimized by manipulation of the geometrical wing shape. In this case the benchmark aircraft is the optimized aircraft in the first AGILE design campaign introduced in section 1.3. Based on the involved discipline specialists a set of DCs to include in the optimization workflow are agreed upon with the customer.

The following three sub-tasks are integrated in the PD layer to support in defining the design case and requirements:

- **Task 1.1: Define requirements**
  Requirements are defined through knowledge acquisition forms. These forms follow the requirement model shown in figure 4.3. This requirement model is composed of the following attributes: description, type, responsible (actor), means of compliance and compliant. Each requirement which is added

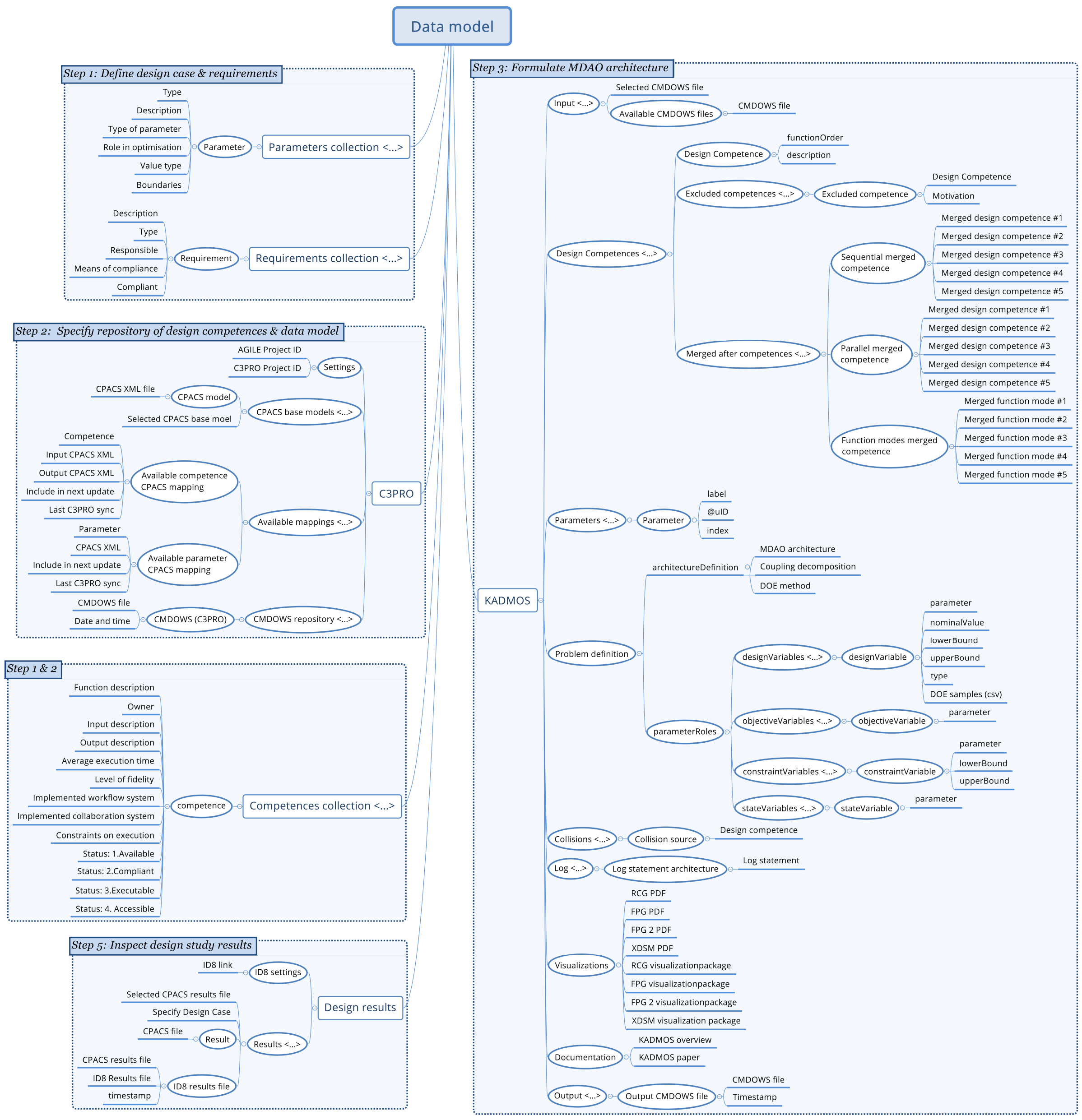


Figure 4.3: Defined data model used to store information generated during the MDO framework development process

in step 1.1. requires the customer to specify a requirement description, a requirement type (options are: objective, competence, concept, performance, deliverable), the actor responsible for validating the requirement and a means of compliance. This means of compliance is a procedure to assess whether the requirement is met.

The implemented task-form for task 1.1 can be seen in appendix B in figure B.6.

- **Task 1.2: Define competences & parameters**
  In the second sub-task of step 1 in the MDO framework development process the discipline specialists get access to the filled in requirements in task 1.1. Based on these requirements the available set of DCs can be made available to the different actors involved in the setup of the MDO framework. For each DC, the responsible discipline specialists provides information according to knowledge forms which follow the DC model shown in figure 4.3. The discipline specialist provides information on the function description, required tool inputs, produced tool outputs, the average execution time and the level of fidelity. Moreover, each specialist must indicate if their tools are available within their organization and

Table 4.1: Example of design case requirements specified for an aerodynamic and structural wing optimization

Requirement	Description
Initial data set	The initial data set contains all geometrical parameters and analytical results coming from design campaign 1. For this the CPACS base-file can be used
Design concepts to include	This design case requires an optimization study performed solely on the wing of the DC1 aircraft. Only the concept of the wing has to be included in the modelling.
Competences to include	The wing design study will require the use of structural and aerodynamic analysis tools, such as EMWET and Q3D. This entails a viscous and an inviscid aerodynamic analysis to be performed.
Objective functions	The objective is to minimize the maximum take-off weight of the full aircraft.
Modelling the wingless aircraft	Since the design problem is focused on the main wing geometry only, the rest of the aircraft is modeled by estimating a realistic weight, lift coefficient and drag coefficient of the wingless aircraft.
Design variables	The wing shape should be optimized by varying: root chord, taper ratios (per wing segment), dihedral angle, sweep (per wing segment), and wing span
Constraints	Two constraints have to be met: all fuel for the required range has to be stored in the main wing fuel tanks and the wing loading of the aircraft cannot be higher than the wing loading of the initial design point.

ready to use in a future simulation workflow.

The average execution time and level of fidelity is required to perform trade-offs on the available disciplinary analysis tools in the MDAO workflow architecture. Four levels of fidelity can be distinguished for each DC as can be seen in figure 4.4. The discipline specialist is able to rank the level of fidelity of their analysis tools from level 0 to level 3. Level 0 is used to indicate if a disciplinary tool only makes use of empirical methods, such as Torenbeek's famous conceptual design sizing rules [39]. On the other end level 3 disciplinary tools make use of the least amount of simplification of the physics phenomena. The knowledge on the tool's level of fidelity and average execution time enables the architect to make decisions on which disciplinary tools to include in a formalized MDAO architecture in the later stages of the MDO framework development process.

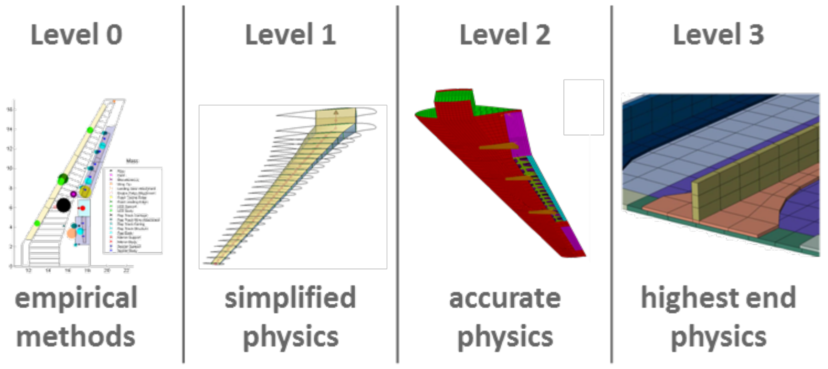


Figure 4.4: Disciplinary levels of fidelity [8]

- **Task 1.3: Manage requirements compliance**
  In the third sub-task of the first step in the MDO framework development process an overview of the requirements is provided. In task 1.3 the status of compliance of all requirements can be managed. This status can switch between compliant and non-compliant. This overview can be used throughout the development process to inspect to assess whether the MDO framework which is being setup full-fills to

the requirements defined at the beginning of the process. Compliance of requirements is tested manually. Some improvements in the future might imply the formulation of rules, to automatically evaluate requirements compliance. In the current implementation however, this ability is not supported.

4.1.2. Step 2: Specify repository of DCs & data model

In the second step the available DCs need to be made compliant to a common data model. This means that inputs and outputs of each DC needs to be mapped on a common data model according to the CPACS format. A large part of the second task is automated if a DC is already mapped to CPACS in previous design studies. Step two is composed of four sub-tasks. These sub-tasks and the developed PD/AD interfaces developed by the author are explained in more detail below:

- **Task 2.1: Import CPACSized DCs into repository**
  Through the user-interface, displayed in appendix B in figure B.8, discipline specialists are able to import their CPACSized competences into a repository. This so-called repository contains both product and process modelling knowledge. A DC is called CPACSized if all required inputs and outputs have been made compliant to the CPACS schema. For each already CPACSized DC this discipline specialist is able to upload two separate CPACS XML files which describe all input and output parameters of their DC(s).

  Based on the input and output XML files two automated transformations are performed: a product model is generated containing the complete set of input and output parameters of all DCs and a process model in which all DCs are automatically coupled. A Functional Flow Diagram (FFD) describing the chain of events in the KE-node service underlying task 2.1 is displayed in figure 4.5. As can be seen from the FFD the underlying KE-node function, the function starts with retrieving two KE-chain projects, one belonging to the main MDO framework development process project and an embedded KE-chain project called the C3PRO project. C3PRO is acronym for Complete, Consistent and Compliant process. The C3PRO project contains the product model and process model which is being assembled in KE-chain to solve the design case specified in step one.

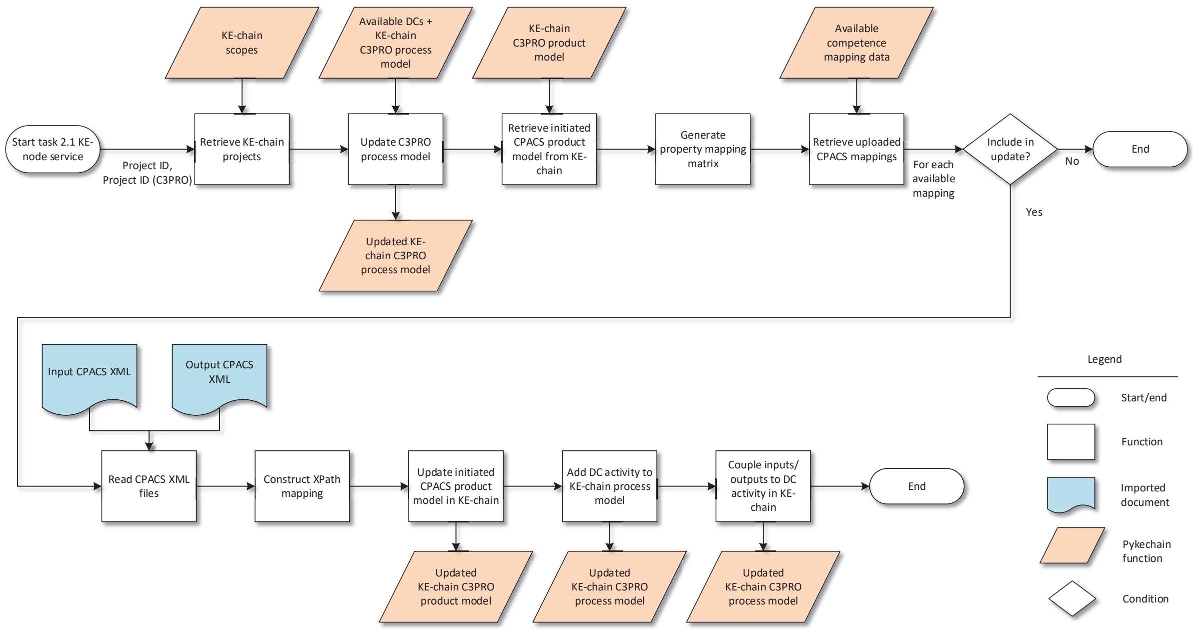

KE-chain scopes
Available DCs + KE-chain C3PRO process model
KE-chain C3PRO product model
Available competence mapping data
Start task 2.1 KE-node service
Project ID, Project ID (C3PRO)
Retrieve KE-chain projects
Update C3PRO process model
Retrieve initiated CPACS product model from KE-chain
Generate property mapping matrix
Retrieve uploaded CPACS mappings
For each available mapping
Include in update?
No
End
Yes
Updated KE-chain C3PRO process model
Input CPACS XML
Output CPACS XML
Read CPACS XML files
Construct XPath mapping
Update initiated CPACS product model in KE-chain
Add DC activity to KE-chain process model
Couple inputs/ outputs to DC activity in KE-chain
End
Updated KE-chain C3PRO product model
Updated KE-chain C3PRO process model
Updated KE-chain C3PRO process model
Legend
Start/end
Function
Imported document
Pykechain function
Condition

Figure 4.5: Functional Flow Diagram of the KE-node service underlying task 2.1

The author makes use of pykechain to retrieve and write information contained in the KE-chain projects. Pykechain [5] is a Python library which is able to connect and interact fully to all features of KE-chain. Pykechain is developed by KE-works and extended by the author and others by request during this

[5]Pykechain, a Python library used for advanced operations using KE-chain `https://pypi.python.org/pypi/pykechain/1.6.0` (accessed: 06/06/2017)

research. Initially pykechain was only supporting interaction with the features of PIM. During this research, to support the advanced operations required in step 2 in the development process, pykechain was extended to support interaction with features of WIM.

After retrieval of the MDO framework development process project and C3PRO project, the product model information is retrieved through pykechain functions. First the DCs made available by the discipline specialists in step 1.2 are synchronized with the process model in the C3PRO project. This function uses the extended pykechain functions to access the features of WIM such as reading and writing activities. Next, the C3PRO product model is retrieved. Based on this product model, a mapping matrix is defined. This matrix maps the XPaths of all properties, generated through the hierarchical structure of the product model, to the property objects themselves. This is used in a later stage to update and import the input and output XML files, add additional properties and couple the inputs and outputs of each DC on a single product model.

After retrieval of the property mapping matrix, the list of available input/output mappings is retrieved. If the discipline specialists uploaded the input and output CPACS XML files the specialist wants to synchronize, the C3PRO product and process models are updated. This starts with reading the XML files, using the Python module ElementTree, which is introduced in section 3.2. For the inputs and outputs the XPaths are retrieved which need to be configured as inputs and outputs respectively. Based on these XPaths, and the previously generated property mapping matrix first the C3PRO product model is updated, followed by the process model. This process is repeated for all uploaded and selected available CPACS mappings.

After the DCs are coupled to a CPACS-based product model in KE-chain, the functions which will be presented in section 4.2 can be used to validate on the completeness, consistency and compliance of all design competences to the product model based on CPACS. Moreover, a similar chain of functions can be used by the architect to import CPACS XML files for each key design parameter in task 1.2. In case not all DCs can be mapped directly on CPACS through XML technologies, the discipline specialists are able to manually reconfigure the product model and activities in sub-task 2.2.

- **Task 2.2: Edit the competence repository**
  In case a DC is not yet compliant to CPACS, the collaborative environment enables manual editing of the repository of design competences and data model. To support the discipline specialists to manually edit the input and output coupling of their DC the task-form of task 2.2 the GUI of KE-chain is used. This manual editing of the repository is performed in the C3PRO project. To maintain an easy working environment, the C3PRO project embedded in the task-form of task 2.2, similar to the embedded C3PRO project displayed in appendix B in figure B.8. This C3PRO project contains the product model and process model currently generated in task 2.1. In the embedded C3PRO project, the discipline specialist is able to manually map property inputs and outputs on the product model through KE-chain. An example of manual coupling of the competence repository is displayed in appendix B in figure B.9.

  In case the product model in the C3PRO project is not yet complete to couple all DCs, a KE-node service is developed by the author to automatically update the product model using a single CPACS XML file. The FFD of the KE-node service underlying task 2.2 is displayed in figure 4.6. The function starts similar to the underlying service of task 2.1: retrieving the KE-chain projects and gathering a property mapping matrix. Based on the uploaded and selected CPACS XML file, the C3PRO product model is updated with all missing properties. After a DC is coupled to the CPACS product model it is successfully CPACSized and compliant to CPACS.

- **Task 2.3: Generate CMDOWS**
  After successfully coupling all DCs to the data model a KE-node service is developed to automatically create a CMDOWS file. In task 2.3 the architect is able to automatically generate a CMDOWS file based on the repository of DCs and data model configured in tasks 2.1 and 2.2. The associated task-form designed in the collaborative environment to support the architect in performing task 2.3 is displayed in appendix B in figure B.10. A CMDOWS file is generated by an underlying KE-node service which follows the FFD displayed in figure 4.7.

  The displayed FFD starts with the retrieval of the KE-chain projects belonging to the MDO framework development process and the C3PRO project. First the information provided by all discipline specialists on the function description, average execution time and level of fidelity is retrieved from KE-chain.

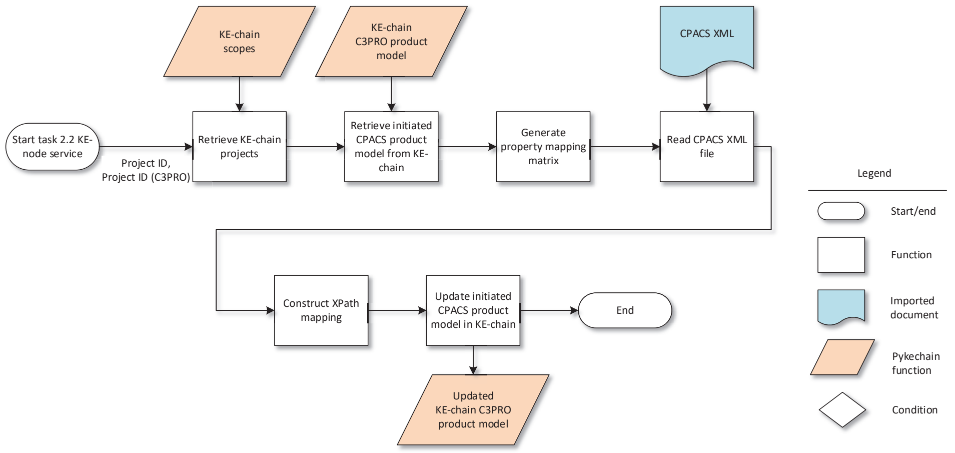

KE-chain scopes
KE-chain C3PRO product model
CPACS XML
Start task 2.2 KE-node service
Project ID, Project ID (C3PRO)
Retrieve KE-chain projects
Retrieve initiated CPACS product model from KE-chain
Generate property mapping matrix
Read CPACS XML file
Construct XPath mapping
Update initiated CPACS product model in KE-chain
End
Updated KE-chain C3PRO product model
Legend
Start/end
Function
Imported document
Pykechain function
Condition

Figure 4.6: Functional Flow Diagram of the KE-node service underlying task 2.2

Next, all activities are retrieved from the C3PRO project. Next a CMDOWS file is generated using the Python ElementTree module, which is introduced in section 3.2. The CMDOWS file is generated using all input and output properties of each DC activity, meta data and performance information. For each input and output property, the product model hierarchical structure is used to generate an XPath, the unique identifier for each property. These XPaths are used in the CMDOWS schema. Finally the generated CMDOWS is validated according to its schema and uploaded back to KE-chain. The same procedure is done for all individual DCs and key design parameters.

From KE-chain all CMDOWS files can be downloaded or used directly in step three of the development process. With respect to step three, the CMDOWS generated by KE-chain is referred to as the RCG. This RCG contains all nodes and edges of the initial repository of DCs and data model. This RCG stored in a CMDOWS file can be used by KADMOS to formulate an MDAO architecture as described in the next section.

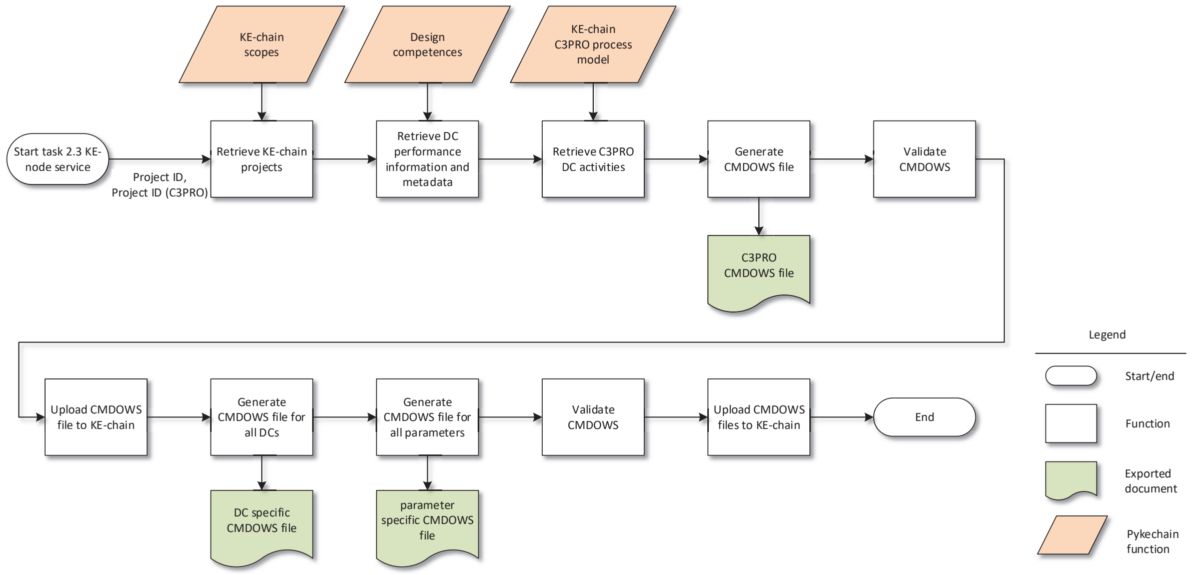

KE-chain scopes
Design competences
KE-chain C3PRO process model
Start task 2.3 KE-node service
Project ID, Project ID (C3PRO)
Retrieve KE-chain projects
Retrieve DC performance information and metadata
Retrieve C3PRO DC activities
Generate CMDOWS file
Validate CMDOWS
C3PRO CMDOWS file
Upload CMDOWS file to KE-chain
Generate CMDOWS file for all DCs
Generate CMDOWS file for all parameters
Validate CMDOWS
Upload CMDOWS files to KE-chain
End
DC specific CMDOWS file
parameter specific CMDOWS file
Legend
Start/end
Function
Exported document
Pykechain function

Figure 4.7: Functional Flow Diagram of the KE-node service underlying task 2.3

- **Task 2.4: Specify competence execution & availability**
  Finally in the fourth and final task of step two in the MDO framework development process additional information needs to be provided by the discipline specialists. In the associated task-form the discipline specialist are able to specify if their DC is executable in a local environment using CPACS input

& output files and if their DC is accessible in a distributed network. It is assumed that all activities required to ensure a DC is accessible and executable are performed according to already existing protocols specified within the AGILE consortium. It is considered out of scope for this research to facilitate in this task in more detail.

4.1.3. Step 3: Formulate MDAO architecture

In the third step of the MDO framework development process a suitable architecture needs to be formulated to solve the design optimization problem. During this research a controllable interface is developed in collaboration with the lead developer of KADMOS[6] to wrap various MDAO architectures around the current set of DCs. This collaboration was required to maximize to identify all required interfaces, required information and interaction with the actors which must be supported in the PD layer. To support the architect in the PD layer, PD/AD interfaces are required to use the KADMOS main functions. This requires the use of KADMOS exposed functions using KE-node services. First a RCG CMDOWS file generated by KE-chain is automatically imported by KADMOS. Next a series of sub-tasks enables graph manipulations described in the paper by van Gent *et al.* [43]. These graph manipulations can be summarized as follows:

- Manipulation of DCs. This involves editing the function order in which the tasks need to be executed, excluding of DCs and merging DCs.
- Assigning of parameter roles such as: design variables, constraint variables, objective variables and state variables.
- Apply MDAO architecture such as Multidisciplinary Design Feasible (MDF), Individual Design Feasible (IDF) or Design Of Experiments (DOE)[7]. The goal of this function is to provide a neutral formulation of the optimization workflow by storing it in an XDSM and CMDOWS file.

An impression of the developed task-forms integrated in the PD layer is displayed in appendix B in figures B.11 to B.17. These task-forms are developed to provide a user-interface to the sub-tasks displayed in figure 4.2. The sub-tasks and developed KE-node and KADMOS interfaces developed during this research are developed as follows:

- **Task 3.0 KADMOS documentation**:
  The first task in step three of the MDO framework development acts as an introduction to the exposed KADMOS functionality. This task-form makes use of the developed extension to the KE-chain task customization which is elaborated upon in section 4.2.1.
- **Task 3.1 Import CMDOWS & inspect RCG**:
  In task 3.1 the first PD/AD interface is integrated to support the import of the RCG CMDOWS file generated in task 2.3 by KE-chain in KADMOS. An overview of FFD of the developed PD/AD interface is displayed in figure 4.8. The FFD starts with the retrieval of the KE-chain project associated with the MDO framework development process. This is followed by a download of the selected RCG on the KE-node server. Next KADMOS is used to import and validate the CMDOWS file and generate a RCG. After the RCG is generated, KADMOS is used to generate a DSM in PDF format and an interactive KADMOS visualization package[8] which is explained in more detail in the paper by Aigner *et al.*[41]. Both the PDF preview and visualization package are uploaded to KE-chain. This allows the architect to inspect the imported RCG in KADMOS through the developed user-interface in the PD layer. Moreover, the imported DCs and parameters in KE-chain are updated in KE-chain according to the data model defined in figure 4.3. The list of imported DCs and parameters enable the architect to inspect if the import was performed successfully. Finally, the KADMOS RCG is saved on the KE-node server. Here it can be retrieved during the execution of the underlying service of task 3.2.
- **Task 3.2 Manipulate design competences**:
  Next, in task 3.2 the architect is able to manipulate the DCs trough the developed user-interface in the

[6]I. van Gent, PhD researcher faculty of Aerospace Engineering, Flight Performance & Propulsion department

[7]More information regarding MDAO architectures can be found on `http://openmdao.org/releases/0.5.0/docs/mdao/` (accessed: 07/06/2017)

[8]KADMOS visualization package, developed by I. van Gent (TU Delft) and B. Aigner (Aachen University) as part of the European AGILE project

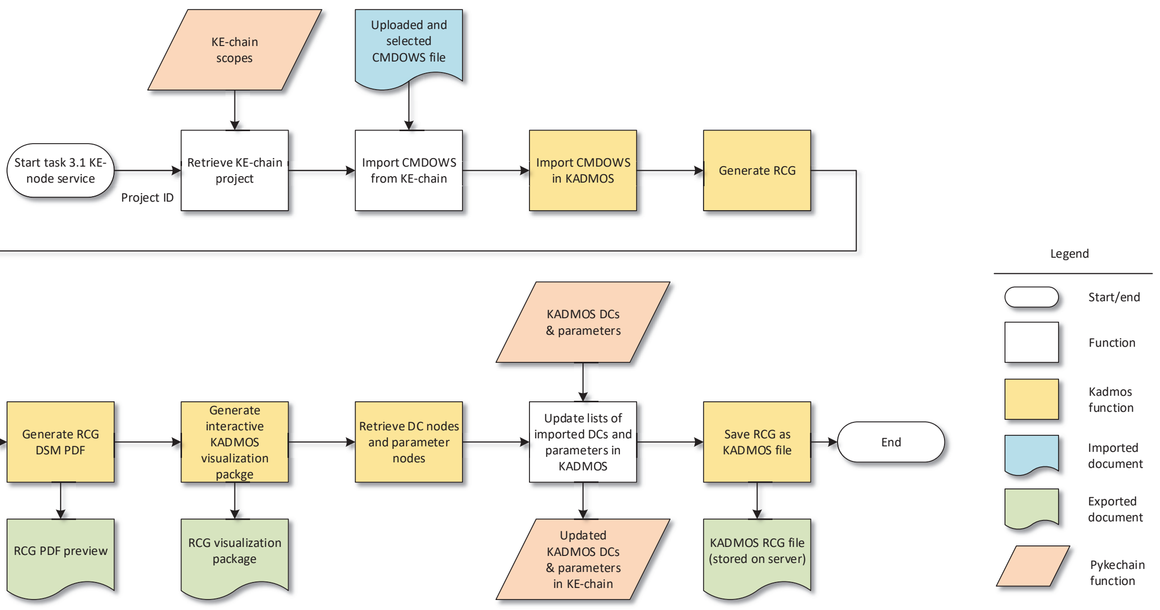

KE-chain scopes
Uploaded and selected CMDOWS file
Start task 3.1 KE-node service
Project ID
Retrieve KE-chain project
Import CMDOWS from KE-chain
Import CMDOWS in KADMOS
Generate RCG
KADMOS DCs & parameters
Generate RCG DSM PDF
Generate interactive KADMOS visualization packge
Retrieve DC nodes and parameter nodes
Update lists of imported DCs and parameters in KADMOS
Save RCG as KADMOS file
End
RCG PDF preview
RCG visualization package
Updated KADMOS DCs & parameters in KE-chain
KADMOS RCG file (stored on server)
Legend
Start/end
Function
Kadmos function
Imported document
Exported document
Pykechain function

Figure 4.8: Functional Flow Diagram of the KE-node service underlying task 3.1

PD layer. An impression of the associated task-form is displayed in appendix B in figure B.12. Through the developed task-form, the architect has control on the function order of all design competences, and the architect is able to exclude or merge DCs. In the KE-node service pykechain and KADMOS functions are integrated according to the FFD displayed in figure 4.9. This KE-node service has been developed to read information from KE-chain through pykechain and write back visualizations generated by KADMOS to KE-chain.

The KE-node service of task 3.2 starts with importing the stored KADMOS RCG file on the server, generated at the end of the of the executed KE-node service underlying task 2.1. Next, the data stored in KE-chain is retrieved. This information is stored in the KE-chain database according to the data model defined in figure 4.3. The imported KE-chain data is validated first in the developed KE-node interface. The data is checked for the following consistency:

– All design competences have a function order defined
– The function order of the design competence follows a sequential order such that the function order is in the range of $[1 : n]$ for n DCs.
– An excluded DC added by the the architect through the interface, needs to be linked to a DC imported by KADMOS in the RCG.
– A block of merged DCs needs to be linked to at least two DCs imported by KADMOS in the RCG
– A block of merged DCs needs to have an adjacent function order defined by the architect through the user-interface.

In case validation of the KE-chain data fails the service execution is stopped and assertions are send to KE-chain to provide feedback to the architect executing the underlying KE-node service. In case all validations pass, the underlying service calls KADMOS to exclude or merge the indicated DCs. This results in a FPG generated by KADMOS. The generate FPG is used to generate a DSM PDF and KADMOS' visualization package. The FPG PDF file and visualization package are uploaded to KE-chain, where they can be downloaded and inspected directly through an embedded previewer. Finally KADMOS is used to save the FPG as a KADMOS file. This KADMOS file is stored on the KE-node server, where they can be used in task 3.3 to assign parameter roles.

- **Task 3.3 Assign parameter roles**:
In task task 3.3 of the implemented MDO framework development process the architect is provided

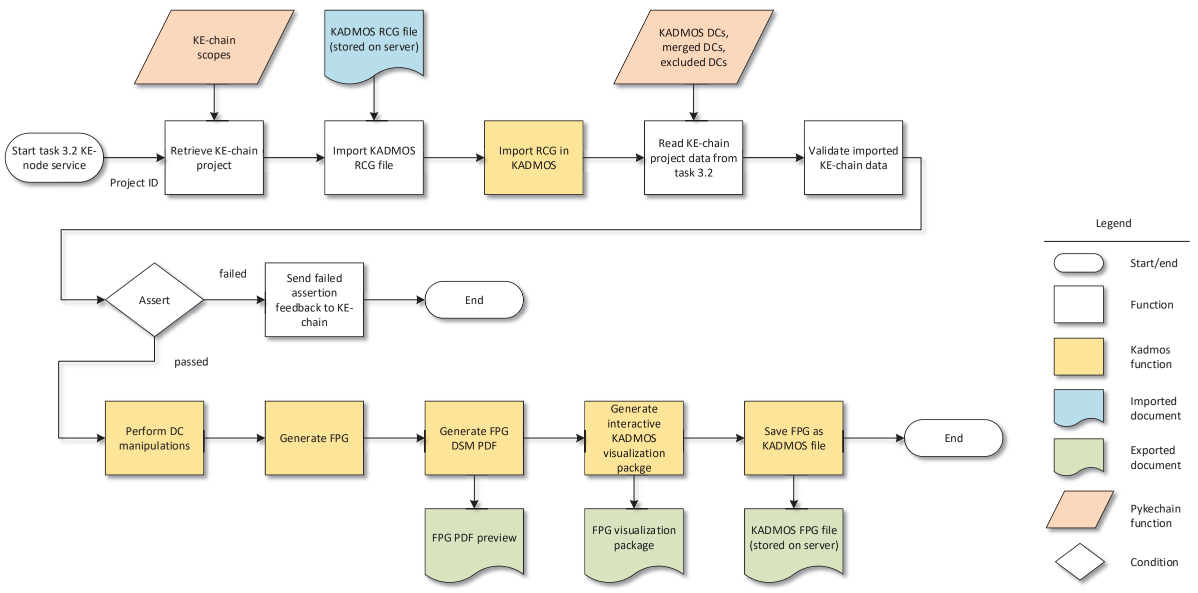

KE-chain scopes
KADMOS RCG file (stored on server)
KADMOS DCs, merged DCs, excluded DCs
Start task 3.2 KE-node service
Project ID
Retrieve KE-chain project
Import KADMOS RCG file
Import RCG in KADMOS
Read KE-chain project data from task 3.2
Validate imported KE-chain data
Assert
failed
Send failed assertion feedback to KE-chain
End
passed
Perform DC manipulations
Generate FPG
Generate FPG DSM PDF
Generate interactive KADMOS visualization packge
Save FPG as KADMOS file
End
FPG PDF preview
FPG visualization package
KADMOS FPG file (stored on server)
Legend
Start/end
Function
Kadmos function
Imported document
Exported document
Pykechain function
Condition

Figure 4.9: Functional Flow Diagram of the KE-node service underlying task 3.2

with a user-interface to assign parameter roles. An impression of the associated task-form is displayed in appendix B in figure B.14. An underlying KE-node service is developed by the author in collaboration with the lead developer of KADMOS to provide an PD/AD interface between KE-chain and KADMOS. In the KE-node service pykechain and KADMOS functions are integrated according to the FFD displayed in figure 4.10.

From the FFD it can be seen that the underlying KE-node service of task 3.3 starts with the retrieval of the KE-chain project associated with the MDO framework development process. Next the stored KADMOS FPG file on the server, generated at the end of the of the executed KE-node service underlying task 2.2, is imported using KADMOS. Next, the data stored in KE-chain is retrieved. This information is stored in the KE-chain database according to the data model defined in figure 4.3. The retrieved data is composed of the design variables, objective variables, constraint variables and state variables added to the problem definition by the architect. The imported KE-chain data is validated first, to check for the following consistency:

– Each indicated parameter role needs to be linked to an FPG parameter which is also available in the imported FPG. This FPG parameter is a unique parameter in the graph. The available FPG parameters might differ from the imported RCG in task 3.1 after DCs are excluded or merges have been performed.
– A design variable and constraint variable must have an upper and lower bound specified. The following rule must be validated: $lowerbound <= upperbound$.

In case validation of the KE-chain data fails the service execution is stopped and assertions are send to KE-chain to provide feedback to the architect executing the underlying KE-node service. After validation of the input data provided by the architect, the list of FPG parameters is updated in KE-chain using pykechain functions. This enables the architect to adapt the assignation of parameter roles to match the updated FPG. After this synchronization KADMOS is used to assign all parameter roles, validate the resulting graph and generate an updated FPG. Next KADMOS is used to generate an inspectable PDF preview of the resulting DSM and an interactive visualization package. Both generated files are uploaded to the KE-chain project where they can be accessed through the PD layer. Finally KADMOS saves the FPG on the server, where it can be used in the final task of step three.

- **Task 3.4 Apply MDAO architecture**:
To conclude step three in the MDO framework development process, the architect is able to apply an MDAO architecture around the defined set of DCs and parameters. An impression of the associated

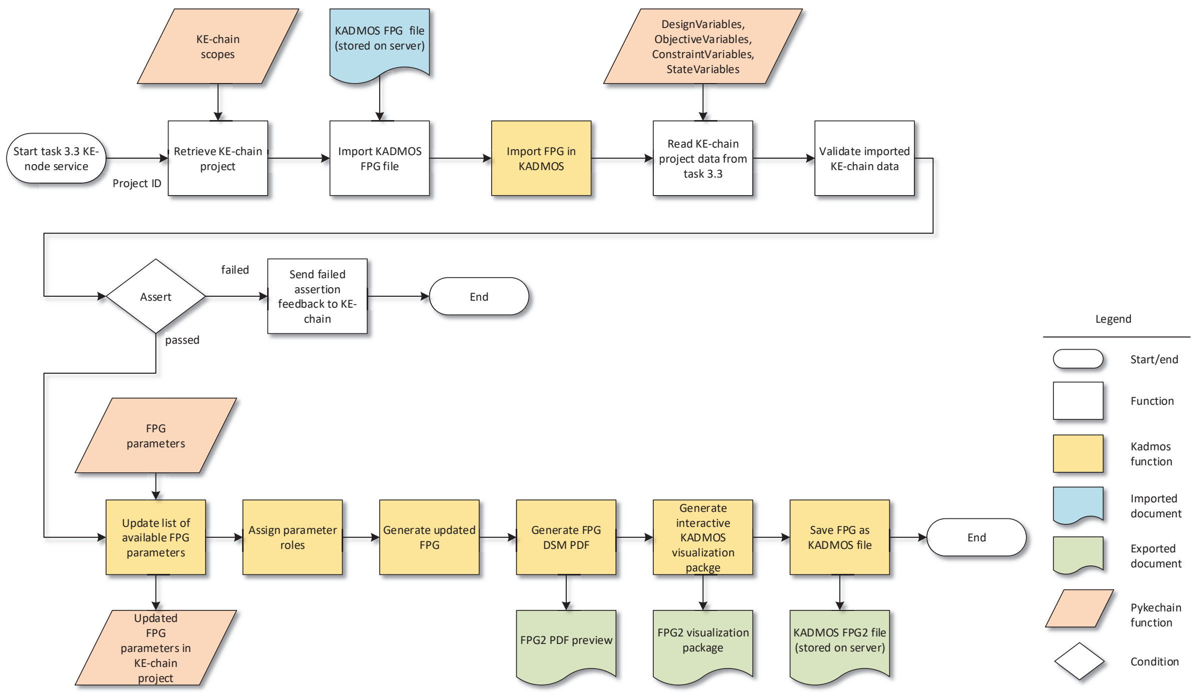

KE-chain scopes
KADMOS FPG file (stored on server)
DesignVariables, ObjectiveVariables, ConstraintVariables, StateVariables
Start task 3.3 KE-node service
Project ID
Retrieve KE-chain project
Import KADMOS FPG file
Import FPG in KADMOS
Read KE-chain project data from task 3.3
Validate imported KE-chain data
Assert
failed
Send failed assertion feedback to KE-chain
End
passed
FPG parameters
Update list of available FPG parameters
Assign parameter roles
Generate updated FPG
Generate FPG DSM PDF
Generate interactive KADMOS visualization packge
Save FPG as KADMOS file
End
Updated FPG parameters in KE-chain project
FPG2 PDF preview
FPG2 visualization package
KADMOS FPG2 file (stored on server)
Legend
Start/end
Function
Kadmos function
Imported document
Exported document
Pykechain function
Condition

Figure 4.10: Functional Flow Diagram of the KE-node service underlying task 3.3

task-form is displayed in appendix B in figure B.17. In the KE-node service pykechain and KADMOS functions are integrated according to the FFD displayed in figure 4.11.

This FFD again starts with the retrieval of the KE-chain project which contains the information of the MDO framework development process. This is followed by a KADMOS function which imports the KADMOS FPG file stored on the server. Next, the data stored in KE-chain is retrieved. This information is stored in the KE-chain database according to the data model defined in figure 4.3. The retrieved data is composed of the MDAO architecture selected by the architect, a type of coupling decomposition and a DOE method if applicable. The architect is able to select the following MDAO architectures: IDF, MDF, (un)converged MDA and (un)converged DOE. The user is able to select two type of coupling decomposition: Gauss-Seidel and Jacobi. In case the author selects Gauss-Seidel as decomposition method feedback coupling is removed, while Jacobi will remove both feed-forward and feedback coupling between multidisciplinary analyses. The imported KE-chain data is validated first in the developed KE-node interface. The data is checked for the following consistency:

- A DOE method must only be specified if the architects an (un)converged DOE architecture. For all other MDAO architectures, a DOE method should be set to empty.
- In case a user selects a DOE method, the defined design variables in the imported FPG should contain DOE sample files in a CSV format.

In case all validations pass, KADMOS is used to apply the selected MDAO architecture. Next KADMOS will validate the resulting graph. This implies checking if all parameter roles have been correctly applied to match the selected MDAO architecture. During this process a log is generated, which is formatted and sent to KE-chain using pykechain. This log can be inspected by the architect, such that fixes can be made on the formalized MDAO architecture. If the graph validation passes, KADMOS generates an XDSM. From this XDSM a PDF preview and inspectable visualization package is generated. Both are uploaded to KE-chain, where they can be inspected by the architect. Finally a CMDOWS file is generated in which the entire graph is stored, along with all information specified in KE-chain during step two of the development process. This CMDOWS file is ready to use for implementation in a suitable PIDO application in step four of the development process.

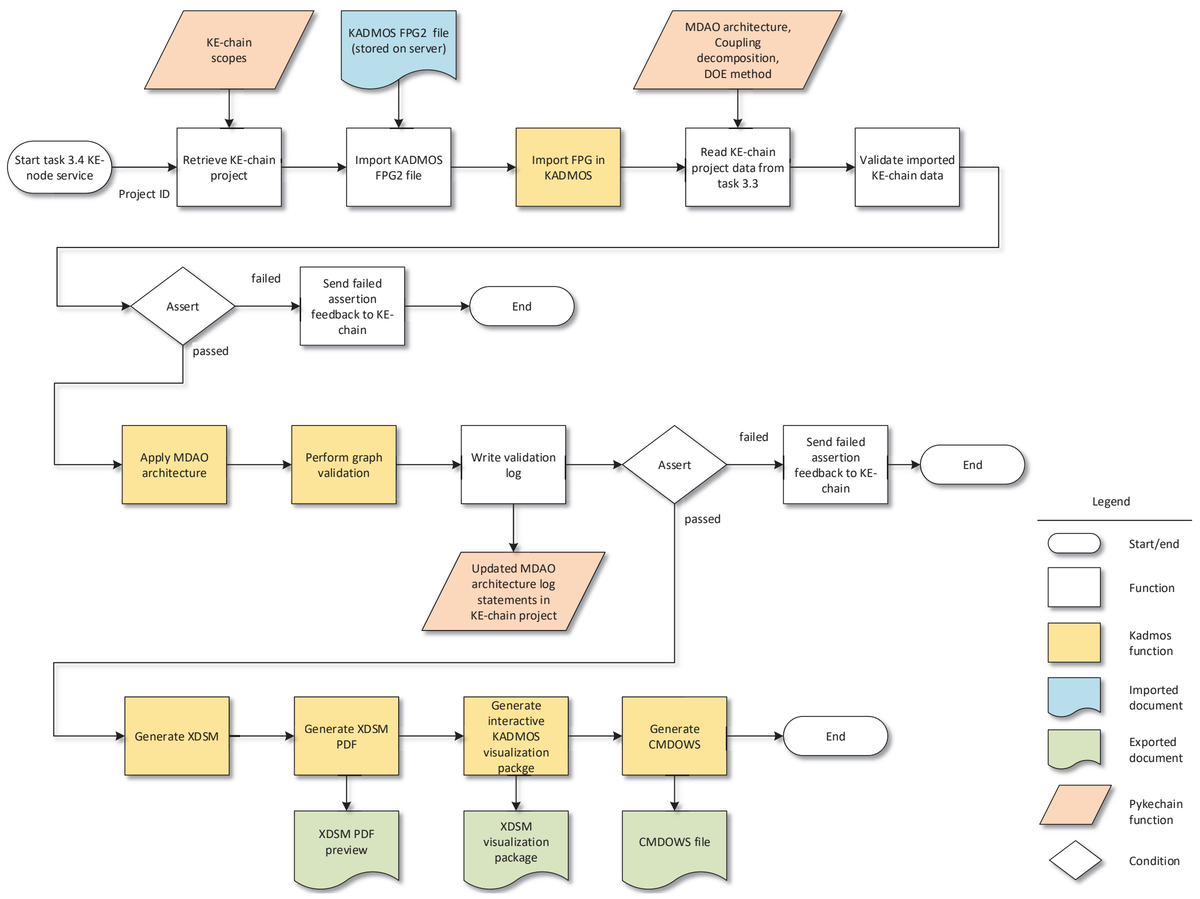

KE-chain scopes
KADMOS FPG2 file (stored on server)
MDAO architecture, Coupling decomposition, DOE method
Start task 3.4 KE-node service
Project ID
Retrieve KE-chain project
Import KADMOS FPG2 file
Import FPG in KADMOS
Read KE-chain project data from task 3.3
Validate imported KE-chain data
Assert
failed
Send failed assertion feedback to KE-chain
End
passed
Apply MDAO architecture
Perform graph validation
Write validation log
Assert
failed
Send failed assertion feedback to KE-chain
End
passed
Updated MDAO architecture log statements in KE-chain project
Generate XDSM
Generate XDSM PDF
Generate interactive KADMOS visualization packge
Generate CMDOWS
End
XDSM PDF preview
XDSM visualization package
CMDOWS file
Legend
Start/end
Function
Kadmos function
Imported document
Exported document
Pykechain function
Condition

Figure 4.11: Functional Flow Diagram of the KE-node service underlying task 3.4

4.1.4. Step 4: Implement & execute AD workflow

In the fourth step of the implemented development process in the PD layer a simulation workflow formulated in step three using KADMOS needs to be implemented in either RCE or Optimus. RCE is an open source distributed, workflow-driven integration environment developed at the DLR [9]. Optimus is a Process Integration and Design Optimization (PIDO) developed by Noesis [10]. Both RCE and Optimus are PIDO applications available in the AGILE consortium. The fourth step is currently not integrated automatically in the PD layer, hence a manual interface is required to initiate an executable workflow using the CMDOWS file produced at the end of step three and execute the simulation workflow.

4.1.5. Step 5: Inspect design study results

Finally the design results obtained in step four need to be made available for inspection. The design results can be inspected through Noesis Solutions ID8 software package [11]. A KE-node interface enables automatic interpretation of the design study results produced in step four. ID8 is embedded in the collaborative GUI environment of KE-chain in step five. An illustration of the implemented task-forms associated with step five in the collaborative environment can be seen in appendix B in figures B.19 and B.20.

As displayed in the business process overview displayed at the beginning of this section in figure 4.2, step five consists of two sub-tasks. For sub-task 5.1 an underlying KE-node service is required to import a CPACS results file generated after a successful simulation workflow execution in step four. The exposed PD layer functionality and developed PD/AD interface is explained in this section:

- **Task 5.1: Select CPACS file for inspection**

[9]http://rcenvironment.de/ (accessed 11/04/2017)
[10]http://www.noesissolutions.com/our-products/optimus (accessed 11/04/2017)
[11]http://www.noesissolutions.com/our-products/id8 (accessed 11/04/2017)

In task 5.1 the integrator is able to upload a CPACS result file of an executed simulation workflow. This result file is stored in KE-chain. Based on the list of result files the integrator is able to select a result file for inspection. After selection of a result file the integrator can execute the underlying KE-node service to read and convert the file. The underlying KE-node service is coupled to a KE-node function in which an AD/PD interface is made between an ID8 conversion package. The FFD of the underlying service is displayed in figure 4.12.

In this FFD one can see that the underlying KE-node service starts with the retrieval of the KE-chain project which contains the information of the MDO framework development process. Next, the uploaded and selected CPACS file is imported using pykechain. Next the specified design case for inspection selected by the integrator is retrieved. Based on the selected design case, the CPACS result file is imported and read using a python module developed by Noesis, to support the ID8 visualization package. This Python package is integrated in the underlying KE-node service. After importing the CPACS result file, the available design cases stored in that file are checked. In case an extended set of design cases are available, this set is updated in the KE-chain project. If the design case specified by the integrator is matched with the uploaded result file, a final Python function developed by Noesis is called which writes the ID8 results file. This ID8 results file is then uploaded back to KE-chain, where it can be downloaded and used for inspection in task 5.2.

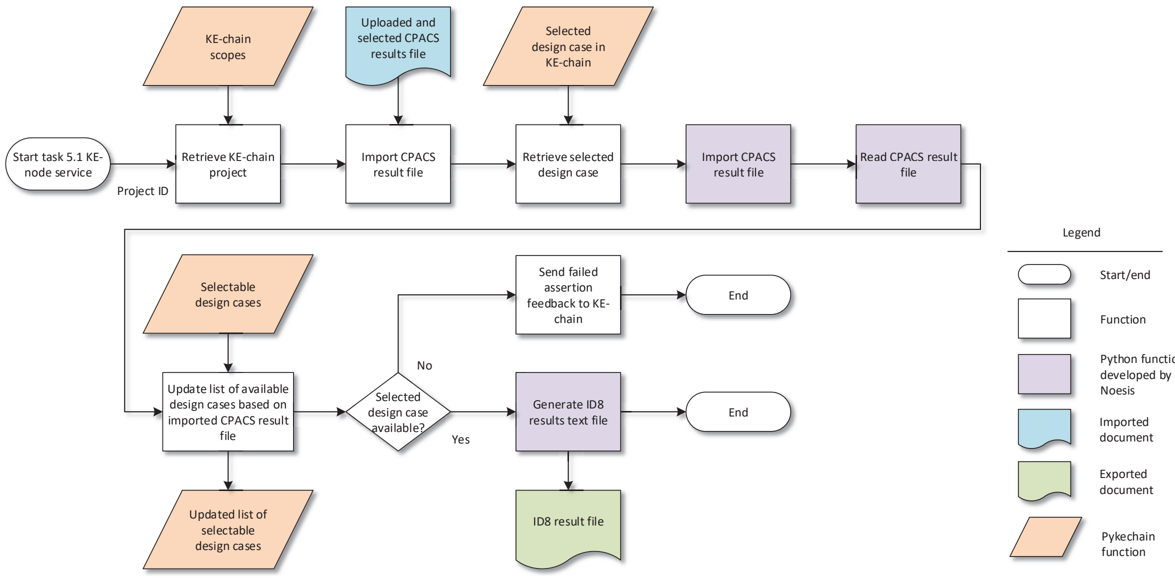

KE-chain scopes
Uploaded and selected CPACS results file
Selected design case in KE-chain
Start task 5.1 KE-node service
Project ID
Retrieve KE-chain project
Import CPACS result file
Retrieve selected design case
Import CPACS result file
Read CPACS result file
Selectable design cases
Update list of available design cases based on imported CPACS result file
Selected design case available?
No
Yes
Send failed assertion feedback to KE-chain
End
Generate ID8 results text file
End
Updated list of selectable design cases
ID8 result file
Legend
Start/end
Function
Python function developed by Noesis
Imported document
Exported document
Pykechain function

Figure 4.12: Functional Flow Diagram of the KE-node service underlying task 5.1

- **Task 5.2: Upload file and inspect results in ID8**
  Task 5.2 concludes the implemented MDO framework development integrated in the collaborative environment. This task contains an embedded ID8 environment. In this environment the integrator, together with the customer is able to inspect the design case study results through inspectable graphs developed by Noesis. An impression of the embedded ID8 environment can be found in appendix B in figure B.20.

4.1.6. Integration round-up

In this section the integration of the various components required to successfully operate in the PD layer of the defined knowledge architecture in section 2.4. A reflection can be made on the developed functionality and provided control over the MDO framework development process through an integrated business process. In a great extend the PD layer integrated in the collaborative environment addresses five main domains which can be identified in any PDP. According to Danilovic and Browning [10] these five domains involve:

- **The product system**: the desired outcome of the development process
- **The process system**: the work that needs to be done to achieve the desired development process outcome

- **The system organization**: the subdivision of actors in different groups, teams or departments
- **The system of tools**: implying information technology solutions and the tools the actors or process system use to perform the work
- **The system of goals**: the requirements, objectives and constraints which apply to all systems.

The business process activities required to operate in the PD layer form the process domain. The product domain is handled by the underlying data model, which provides a template for the information generated during the course of the process execution. A system organization is provided through the ability to form a specific development team involved in a particular design case. Among this development team the following actors can be identified: discipline specialists, architects, integrators, collaboration engineers and the customer. Design activities are grouped to provide a work domain for the different actors. A system of goals is provided in the collaborative environment through the specification of system level requirements. Using a system of goals, the different actors are able to reflect upon the work they've been doing. Finally a system of tool domain is provided by integration of the different components in the AD and PD layer of the knowledge architecture through knowledge technologies.

Moreover the implemented PD layer functionality aims to adhere to some of the MDO framework requirements specified in section 2.1. Especially the requirements originating from the domains of architectural design, problem formulation and accessibility of information have been adhered to in great extend during the integration of the various components and implementation of the collaborative environment. However, on the integration of the various components in the collaborative environment do not aim at incorporating the requirements on problem execution. Integration of applications to support workflow execution were considered out-of-scope during this research.

4.2. Support

In this section the developed methods to support human inspection of the MDO framework setup through the development of user interfaces which provide the means to manage the MDO framework's underlying product and process models. Moreover developed components which were able to support integration of the actors and components in the collaborative environment is presented. All developed components have been developed according to the already existing KE-chain platform. In this section the required extension to the existing KE-chain architecture, introduced in section 2.4.1, are explained in further detail. In this section the following topics are addressed:

- **Task customization**: The ability to customize a task-form such that is tailored specifically for the work which is supported by the task. For example: Information forms in grids, the integration of figures, embedded websites, interactive visualizations, buttons and text-panels.
- **Model verification**: The ability to verify and inspect the coupling of the product and process model.

4.2.1. Task customization

Task customization is an enabler to support in the configuration of KE-chain projects. Task customization is developed to support a very important, and currently not mentioned actor in the MDO framework development process. This actor is the configurator. The configurator ensures that prior to the actual execution of the development process, the data model and work breakdown structure are defined correctly. Moreover, the configurator is able to edit the layout of each task-form. The layout of a task-form plays an important role in the information intensive business processes. The large amount of information generated during the MDO framework development process needs to be inspectable and manageable by the respective actors. This is supported by the requirements adopted on the PD layer of the MDO framework described in section 2.1 defined by Salas and Townsend [31] and endorsed in [4, 27].

The author has extended the task customization of KE-chain task-forms to maximize the user-experience in the collaborative environment by the addition of buttons, embedded visualizations linked to project data and explanatory text panels. These added widgets and the method the configurator is able to use these widgets is explained in more detail in the remainder of this section.

Task customization requires manual editing of a JSON. This JSON can be edited directly through the KE-chain GUI by the configurator. Hence, the actors of the MDO framework development process are not exposed with the underlying task customization JSON. An example of this JSON is shown in appendix C in figure C.5. In this example, one can see the basic structure of task-customization: a list of JSON components. Each component has an "xtype" dictionary item. This xtype is used in the ExtJS 6 front-end to render the correct front-end component. These xtypes refer to a custom ExtJS component class defined by the author. An overview of the class hierarchy of the developed custom widgets can be seen in figure 4.13. As can be seen from this figure each widget has mandatory attributes (indicated by the + symbol) and optional attributes (indicated by the *o* symbol). These attributes are explained in more detail in the remainder of this section.

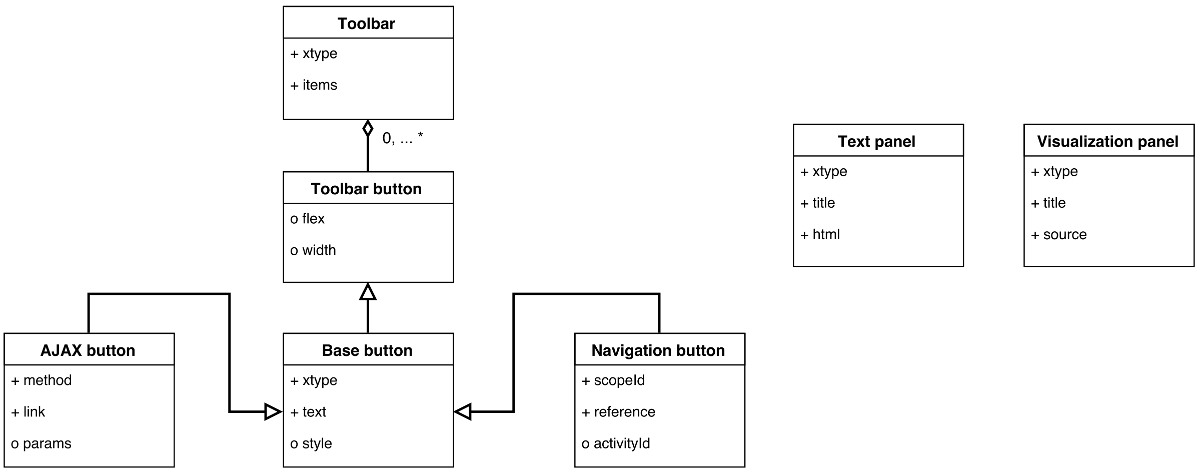


Figure 4.13: Available widget classes implemented by the author to support task-customization

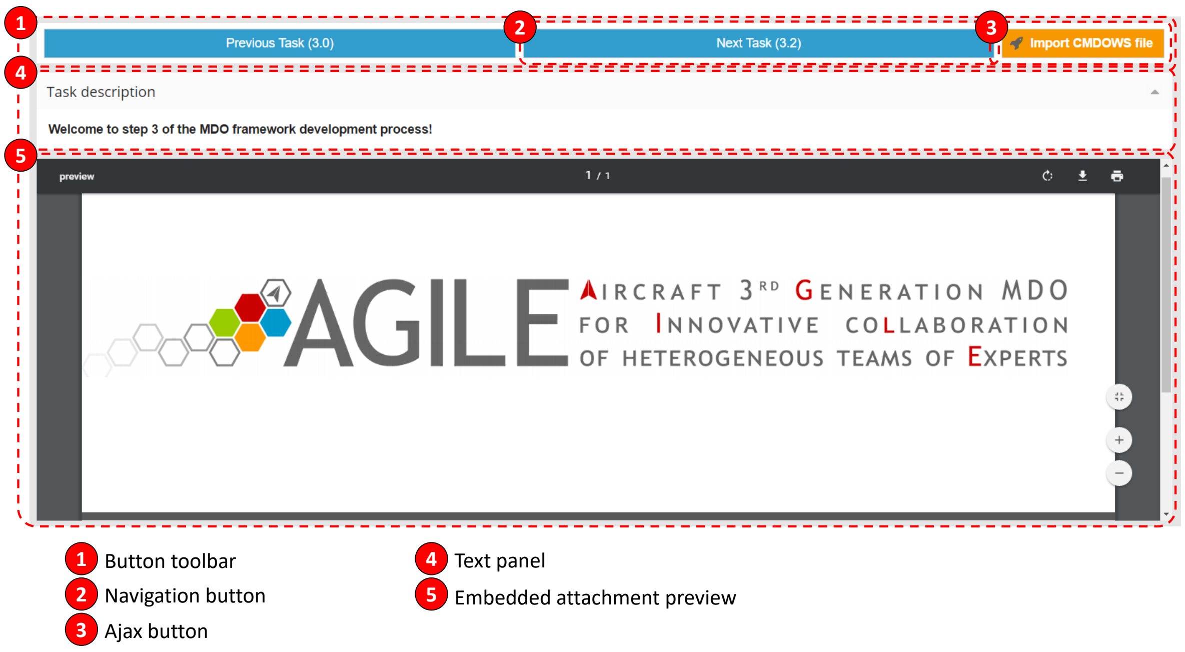

1
2
3
Previous Task (3.0)
Next Task (3.2)
Import CMDOWS file
4
Task description
Welcome to step 3 of the MDO framework development process!
5
preview
1 / 1
AGILE
AIRCRAFT 3RD GENERATION MDO FOR INNOVATIVE COLLABORATION OF HETEROGENEOUS TEAMS OF EXPERTS
1 Button toolbar
2 Navigation button
3 Ajax button
4 Text panel
5 Embedded attachment preview

Figure 4.14: Example of newly implemented type of task-customization objects in KE-chain

- **Configurable buttons**:
  To support quick navigation between task-forms in a project and the execution of underlying services, the author has developed configurable buttons which can be added by the configurator. A distinction can be made with two type of buttons: AJAX[12] request buttons and KE-chain navigation buttons. Both buttons are indicated in figure 4.14 by numbers 2 and 3 respectively. A navigation button requires the specification of a scope ID, corresponding to the KE-chain project one which to be redirected to. Additionally an activity ID needs to be provided, in case button selection should redirect to an activity. An AJAX request button requires the specification of a GET, PUT or POST method. Usually a GET method is used to retrieve information from the back-end, a PUT method is used to edit information in the back-end and a POST is used to impose an action exposed by the KE-chain API. In some cases, additional parameters need to be provided to perform an AJAX request, as indicated in the displayed UML diagram.

  The KE-chain navigation buttons are made available by the author to create a nice flow in the business process directly from the task-forms exposed to the user. Navigation buttons are integrated in the collaborative environment to allow the different actors to navigate backwards and forwards between preceding and succeeding task-forms. This creates a shortcut compared to the traditional navigation pattern provided by KE-chain. In this traditional navigation pattern KE-chain users are only able to navigate to the different task-forms through the work breakdown overview of which an impression is provided in appendix B in figure B.4. The AJAX request buttons have been implemented by the author to support the execution of underlying KE-node services, to make the PD/AD interfaces an integrated part of the provided user-interface in the PD layer.

- **Configurable text panels**:
  The author has enabled the configurator to embed text-panels in the task-forms. An example of a rendered text panel can be seen in figure 4.14 indicated by the number 4. The author experienced that the different type of actors in the collaborative environment (defined in section 1.4) often need assistance in performing the different steps defined in the business process. For this purpose text panels have been added to KE-chain, which can be configured freely by the configurator to document on the important steps. More explanation on the configurable text panels and the required HTML content which needs to be provided by the configurator is shown in appendix B.

[12]AJAX requests are explained in more detail on: `https://www.w3schools.com/xml/ajax_xmlhttprequest_send.asp`

- **Live rendered (and interactive) previews**:
  To conclude the task customization, the author has implemented a widget class to support in the visualization of important information inside a task-form. As can be seen in figure 4.14 indicated by number 5, an inspectable frame can be embedded in a KE-chain task-form by the configurator. Embedded previews are an essential part of the implemented collaborative environment to provide the desired PD functionality. Using embedded previews, the architect, discipline specialists and customers are able to directly investigate intermediate or final results of the work performed in the different steps of the development process. Some examples of the information provided to the actors through the developed preview components can be seen in appendix B in figure B.8 showing an embedded KE-chain C3PRO project (explained in more detail in section 4.1.2), and figures B.11 to B.17 showing embedded DSMs and an XDSM.

4.2.2. Model verification

To support the system architect in setting up a complete and consistent repository of DCs and data model a model verification module and model inspector is researched and developed by the author. The model verification module is based on the concept of reducing and eliminating waste in the process introduced in section 3.1.

The model verification module extends the existing domains of PIM and WIM in the KE-chain application as is graphically displayed earlier in section 2.4.1 in figure 2.5. PIM contains all models and function which apply to the product model. WIM contains all models and functions which apply to the process model definition. The model verification module quantifies and qualifies the product and process model on Information Quality (IQ) categories completeness, relevance and consistency. Completeness defines the breath, depth and scope of information [45]. Relevance is the extend to which information is applicable for and helpful for the task at hand [45]. Consistency implies an absence of contradictions, such that information can be exchanged within the framework in the correct representation. In a complete and consistent framework, only relevant information can be exchanged between tools. The model verification module provides an in-the-loop feedback report based on a set of rules to assist the system architect to mitigate waste in later stages of the PD process. This set of rules and associated wastes can be seen in table 4.2.

Table 4.2: Metric used to quantify and qualify the product and process model on Information Quality (IQ), associated wastes and the adopted color code in the model inspector

IQ category	Rule	Waste	Color in model inspector
Completeness	Property configured as input, but not provided as output in any task	No information flow possible, hence additional activities yielding an increase in man-hours and costs are necessary to overcome a lack of information	Red
Consistency	Property configured as input at least once, but configured as output multiple times	Risk of colliding information. Additional man-hours are necessary to overcome excessive information	Orange
Relevance	Property configured as output but not used as input in the process model.	Additional activities yielding an increase in man-hours and costs are required to produce redundant information and filter out irrelevant information produced by the system	Yellow
Relevance	Property is defined in the product model, but is not configured as either input or output in any activity	Additional activities yielding an increase in man-hours and costs are required to overcome a lack of information flow.	N/A
Consistency	Property configured as output once and configured as input at least once	No waste	Green

The model verification module is coupled to a model inspector. An example of the model inspector can be seen in figure 4.15. In this figure one can see various arbitrary activities (D1, D2, D3, G1, G2 and F) all of which have input properties and output properties, displayed in the left and right column of each activity block in the inspector respectively. The model inspector is embedded in the KE-chain application and uses the input/ output configuration of each activity to automatically generate a data flow diagram as defined in section 3.3.3. The model inspector is an interactive component extending the front-end of KE-chain. This component is developed using a JavaScript based GoJS library [13]. Different filters can be applied to visualize consistent flow, indicated by green parameters, and inconsistencies in the process model indicated by red, yellow and orange parameters. Selecting a parameter or activity highlights its data dependence w.r.t. other parameters and activities, indicated by green arrows in the model inspector.

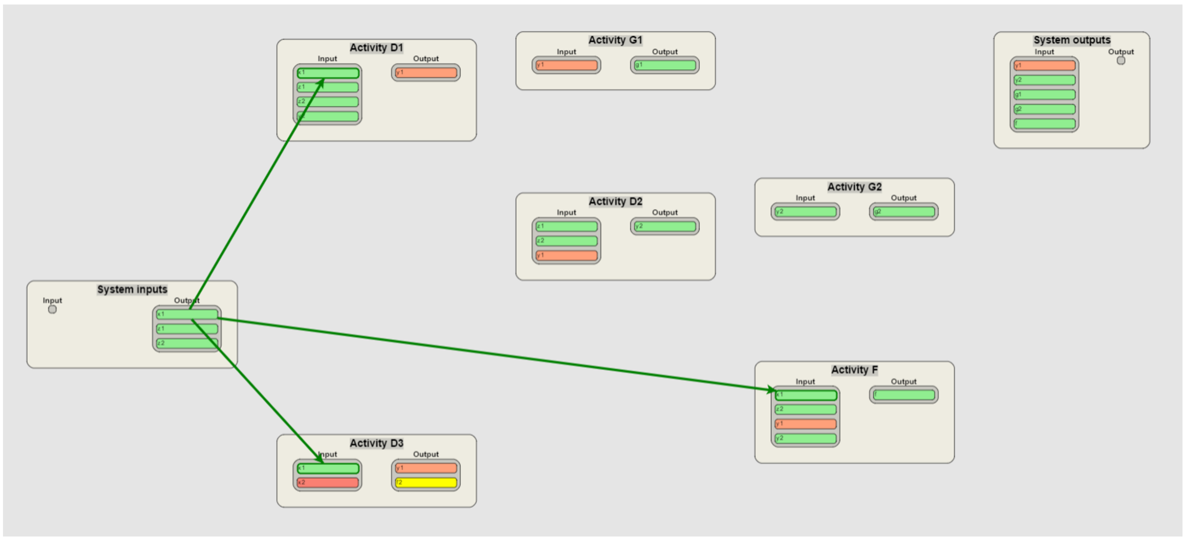


Figure 4.15: Interactive model inspector embedded in the GUI of KE-chain

A consistent data flow implies that all data which is exchanged between different activities has a source and target. As KE-chain is considered to be a closed system, uncoupled input and output properties lead to an inconsistent data flow. A closed and open system can be defined as follows:

- **Open system**: A system that has ongoing interactions with external environments or systems.
- **Closed system**: A system with no external inputs or outputs. Hence there is no interaction with an external environment or external systems.

KE-chain is a closed system due to the fact that all information is contained within the boundaries of KE-chain. Hence, if a certain function is integrated in KE-chain which requires a set of input parameters, they need to be defined within that environment. This is in contrast with the MDO framework architecture that is formalized during steps one to three in the MDO framework development process. In this MDO framework interactions between tools and functional blocks are contained within the system. On the other hand, an MDO framework depends on external parameters to start a design problem: the design variables. Without design variables, no optimization can take place. These design variables configured as input in at least a single tool or functional block. Otherwise, variation of the design variables has no impact on the final design. In the formalization of the MDO framework, it is however not specifically defined which function provides these input parameters.

To use the developed methods to support verification of the imported DCs and key design parameters in the C3PRO project in KE-chain the following schematic overview is used as depicted in figure 4.16. The methods to synchronize and import the DCs and key design parameters in the C3PRO project was introduced in section 4.1.2. In this schematic overview one can see three sub-processes: system inputs, design competences and system outputs. In this example, the system inputs is a single set of key design parameters

[13]http://gojs.net/latest/index.html (accessed: 15/3/2017)

referred to as geometry. These system inputs are declared as design variables in task 1.2 in the MDO framework development process. The system outputs contain a collection of output key design parameters such as objective variables and constraint variables. Using the imported key design parameters as separate activities, the input and output relations defined for each KE-chain activity, as described in section 3.3.3, the system of DCs of which the MDO framework will be composed of, can be wrapped in a closed system. Using this closed system representation, the model verification rules, introduced in this section apply.

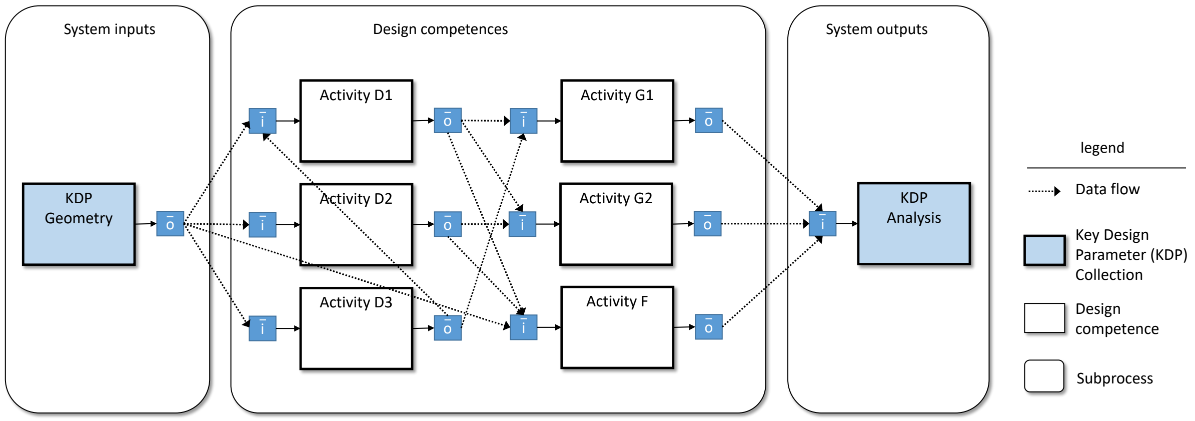


Figure 4.16: Schematic overview of the model verification system implemented in the MDO framework development process *(KDP = Key Design Parameter)*

4.3. Reuse

A reduction in project lead-time can be achieved through front-loading [38] [2]. Front-loading is a strategy which seeks to improve the performance of PDPs by shifting the identifications and solving of design problems to the early stages and even in front of the official start of the actual project. A key enabler for front-loading is reuse of standard solutions. To support reuse of product and process modeling knowledge from previous MDO frameworks, two methods are implemented by the author:

- Exporting and importing product and process knowledge using CPACS or CMDOWS XML documents
- A central knowledge library extending KE-chain.

Exchangeable CPACS and CMDOWS documents

In section 4.1 services to automatically generate a product and process model using CPACS XML files were presented. Additional services are implemented to import both product and process models in KE-chain using a CMDOWS file. This functionality can be used to quickly import and manipulate already existing workflow configurations. CMDOWS files can be generated at any stage during the PD process. Having the option to automatically generate a product model and couple tools defined in the CMDOWS file automatically enables the use of CMDOWS files to quickly switch to different workflow configuration during operation.

Central knowledge library

A central knowledge library enables easy access to standard solutions without the need for file exchange. The knowledge library is a new concept in KE-chain introduced by the author. Its position with respect to PIM and WIM in the application architecture of KE-chain can be seen in figure 2.4. Part assemblies and coupled activities can be retrieved from or saved to the knowledge library through a *publish* and *promote* action respectively. The *promote* and *publish* actions are shown in figure 4.17. A promotion enables a system architect to create a copy of the selected part and its children, which is stored in the library. The architect can reuse stored models in any project using a *publish* action. A published model can be used multiple time in a new project by creating a so-called proxy model. During the creation of a proxy model, the associated library activities can be published as well. This approach favours a building block approach to quickly setup new product and process models.

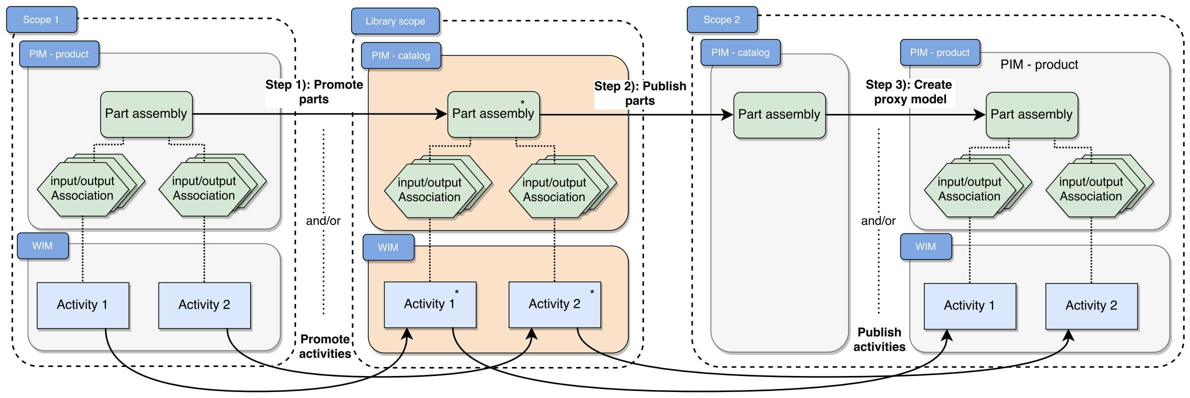


Figure 4.17: Writing and reading coupled part assemblies and activities with the publish and promote actions through a knowledge library

Both the central knowledge library and exchangeable documents enable a quick setup of the underlying MDO framework's product and process models. In the integrated MDO framework development process as introduced in section 4.1 the setup of the product and process model is done using the C3PRO project. The methods discussed in this section, apply in a great extend to this C3PRO project. In this project, the architect and discipline specialists would benefit from using methods to reuse predefined model configurations.

To conclude this chapter, various supporting methods have been developed by the author which extend KE-chain such that the implemented collaborative environment is able to fit in the PD layer of the MDO framework's knowledge architecture defined in section 2.4.

The implemented improvements aim to support the system architect and discipline specialists to integrate the DCs and data model through model inspection, verification and reuse of standardized models. All operations and functionalities are accessible through KE-chain, such that all actors can work collaboratively on setting up a complete and consistent MDO framework. The effort to undo mistakes in the setup of the underlying product and process model traditionally increase over time [15, 21]. Hence, the ability to detect flaws in the underlying product and process models of the framework at an early stage in the development process, aims to shorten project lead-time by minimizing these time-consuming feedback loops.

In the next chapter an application of the developed collaborative environment is presented. In this chapter the methodologies to support in the setup of 3rd generation MDO frameworks is tested for an aerodynamic optimization of an aircraft wing structure.

5 Results

In this chapter the results of this thesis research are presented. These results involve currently implemented use-cases, ongoing use-cases and intended future use-cases of the collaborative environment to support the integration of 3^{rd} generation MDO frameworks. Moreover, user-experience, performed verification and validation and a reflection on the impact of the developed solutions are presented. The contents of this chapter can be summarized as follows:

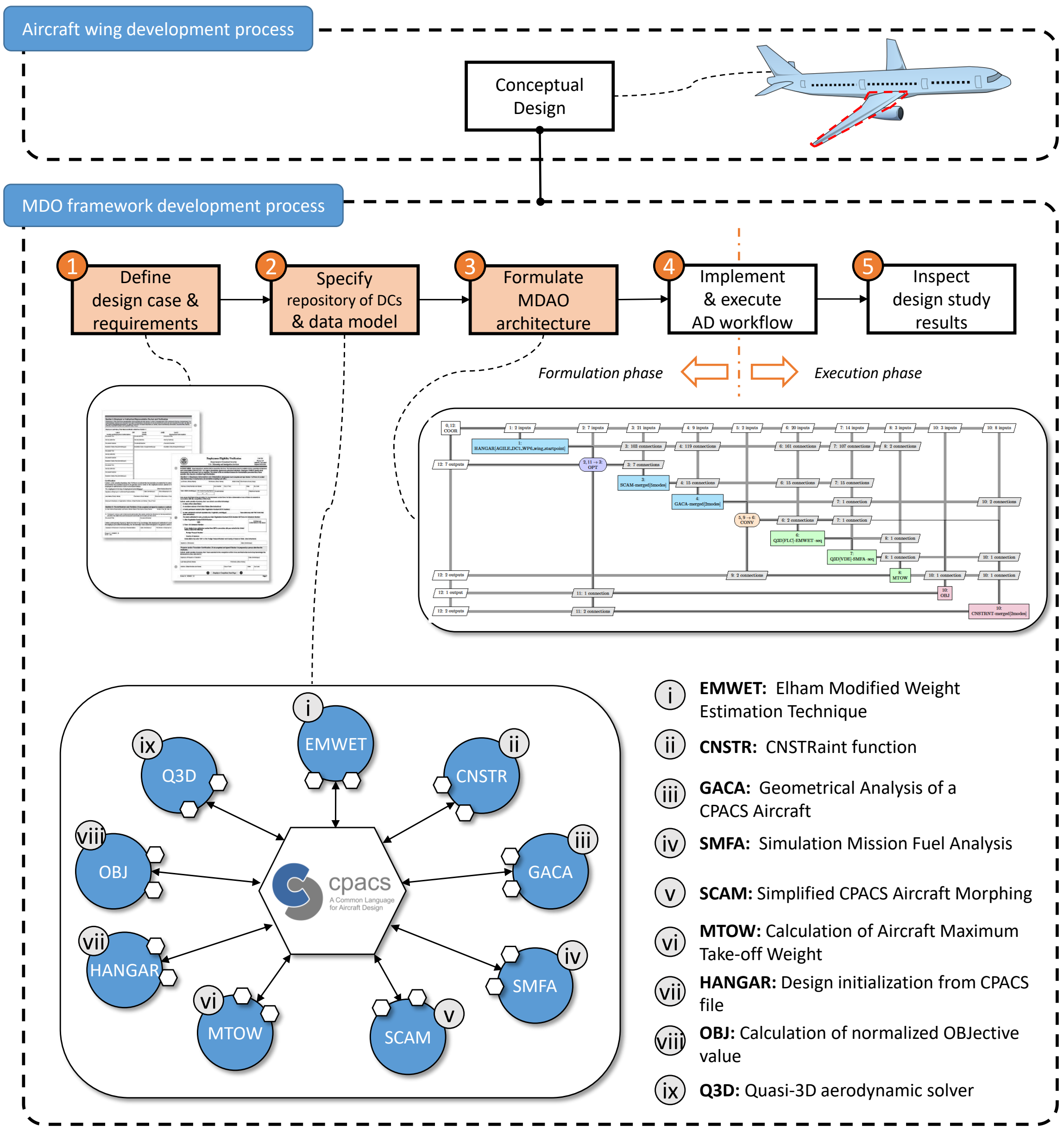
- First in section 5.1 the formalization of an MDO workflow to support geometrical, aerodynamic wing optimization during a workshop session as part of the AGILE project is presented. This workshop has been performed during the AGILE M21 meetings hosted in Delft. During this workshop, the developed collaborative environment and its PD/AD interfaces have been tested in a collaborative session involving multiple design teams involving 43 users.
- Next in section 5.2 verification and validation performed by the author to test various codes, methods and integrated components are explained in more detail.
- As the implemented collaborative environment is an integral part of the AGILE research project for the final work-package design activities. Current and future involvement of the developed solutions during this thesis research are highlighted in section 5.3.
- During the workshop session and current usages of the collaborative environment many users are involved. The results of this research can best be reflected by the actual actors using the implemented solutions. Hence, in section 5.4 some user-experiences are elaborated upon.

5.1. Use-case implementation: Aerodynamic wing optimization during the AGILE workshop sessions

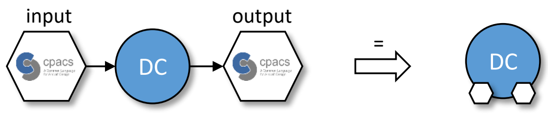
When this research was started, as concluded in the challenges presented in chapter 1, there is a need for improved functionality and integration of business process activities leading to the setup and operation of an executable MDO framework in the AGILE research project. This gap has been bridged by the development of a collaborative environment capable of managing and processing information by extending KE-chain. The developed application has been tested during the M21 AGILE meeting in Delft held on the 4th-5th-6th of April. For this meeting a workshop was prepared, in collaboration with the lead developer of KADMOS, I. van Gent[1]. A total of 43 users, among which various actors such as integrators, architects, collaboration engineers, customers and discipline specialists, worked in distributed groups on the setup of an MDO framework to perform an aerodynamic optimization of an aircraft main wing. Prior and during this workshop training was provided to make the different users familiar with the application.

The main focus of the workshop was to guide the various actors through steps 1 to 3 of the integrated MDO framework development process. This is illustrated in the workshop overview in figure 5.1. In this figure one

[1] I. van Gent, PhD researcher TU Delft, department of Flight Performance & Propulsion


Aircraft wing development process
Conceptual Design
MDO framework development process
1 Define design case & requirements
2 Specify repository of DCs & data model
3 Formulate MDAO architecture
4 Implement & execute AD workflow
5 Inspect design study results
Formulation phase
Execution phase
i EMWET
ii CNSTR
iii GACA
iv SMFA
v SCAM
vi MTOW
vii HANGAR
viii OBJ
ix Q3D
cpacs A Common Language for Aircraft Design
i EMWET: Elham Modified Weight Estimation Technique
ii CNSTR: CNSTRaint function
iii GACA: Geometrical Analysis of a CPACS Aircraft
iv SMFA: Simulation Mission Fuel Analysis
v SCAM: Simplified CPACS Aircraft Morphing
vi MTOW: Calculation of Aircraft Maximum Take-off Weight
vii HANGAR: Design initialization from CPACS file
viii OBJ: Calculation of normalized OBJective value
ix Q3D: Quasi-3D aerodynamic solver

(a) Impression of the main results per step in the MDO framework development process



(b) Explanation CPACSized (CPACS compliant) Design Competence (DC)

Figure 5.1: Illustrated overview of the wing design study during the workshop demonstrations

can see the five steps of the MDO framework development process as part of the conceptual design phase of an aircraft wing development process. In step 1, the administrative procedure of the development process is illustrated, this implies the definition of requirements, available & required DCs and key design parameters. A more detailed overview of all DCs and key design parameters incorporated in the MDO framework is presented in appendix D in tables D.1 to D.3. Next, in step 2 the architect and discipline specialist ensure a complete, consistent and compliant repository of DCs and a data model is defined. In this complete, compliant and consistent repository, all introduced tools and functions are coupled to a standardized CPACS model.

This implies that the input and output of each tool needs to be specified in CPACS parameters, as is indicated in figure 5.1b. During the workshop the actors were guided through the design activities summarized below:

1. The definition of MDO use-case and its requirements.
2. The introduction of DCs to the problem definition.
3. The integration of a DC in the repository of DCs and data model using the provided KE-node interfaces.
4. The manual coupling of a new DC to a CPACS based product model using the embedded C3PRO project in the developed collaborative environment.
5. The generation of a CMDOWS file of the generated repository of DCs and data model, which can be used to formulate the MDO framework architecture.
6. The importing of a CMDOWS file in KADMOS using the developed user-interface and PD/AD interface belonging to task 3.1 of the MDO framework development process.
7. Manipulation of the design competences and assigning of parameter roles using KADMOS through the developed user-interface and PD/AD interface associated to tasks 3.2 and 3.3 in the MDO framework development process.
8. Applying of various MDO architectures, such as MDF and IDF.
9. The automated generation of a CMDOWS file using KADMOS through KE-chain of the formulated MDO architectures.

Based on the experiences of various users it was found that the developed application and added functionality provided good control in the setup of the MDO framework. The developed interfaces to the various components in the lower layers of the framework's knowledge architecture allowed for easy manipulations of the product and process models. Although the KE-chain server experienced excessive loads during the workshop due to the amount of people working simultaneously, most users were able to experience and perform all operations exposed in the PD layer. Improvements on the performance are however required to cope with larger design and optimization projects. As the performance issues are caused by the chosen collaboration and integration platform KE-chain, it is currently considered out of scope to solve the performance issues during this research. However, all findings and limitations are included in the KE-chain road-map and worked on by the developers of KE-chain.

The result of this research is the implementation of an application extending KE-chain to provide control during the setup and operation phase of an MDO framework in the context of the AGILE research project. From the research it can be concluded that a collaborative approach in the setup of complex and distributed MDO frameworks shortens the design process lead-time. Traditionally a lack of control during the design process leads to time-consuming iterations in later stages of the development process. A gain in project lead-time is achieved by increased transparency and control during the setup of the underlying process and data model by providing automated interfaces to quickly make conceptual design decisions.

5.2. Verification & validation of the developed collaborative environment

In this section the results of verification and validation of the implemented solutions are presented. For this two methods were chosen: unit testing of the implemented codes and testing functionality to small test cases.

5.2.1. Unit testing

As the implementation of all solutions builds on the foundation of a functioning software application KE-chain, it is important to test whether the implemented extensions in KE-chain do not break existing functionality or codes. For this purpose unit tests have been executed before every new deployment of the KE-chain application with the developed extensions. A unit test is written to test small blocks of codes for which the outcome can be tested with an expected result. Although some pieces of developed code have not been tested, the current coverage of back-end functionality show a 100% pass of all 603 tests.

5.2.2. Test cases

In order to test the main functionality regarding import/export of standardized product and process models and integration of the business process, framework configuration and KADMOS several test cases have been implemented. For this purpose a Sellar problem has been implemented. The Sellar problem, described in [33] shows a small set of functions used to test MDO optimization techniques. The MDO framework has been passed through the entire chain of human and automated activities in the PD layer of the AGILE framework. As executing and setting up of the MDO architecture is out of scope in this research, testing of the resulting CMDOWS of the RCG is done using KADMOS. As it is now possible to setup the entire MDO framework for this small use-case, without any programming or XML-editing involved, in an accessible application through the Internet. Based on these test cases it can be concluded that the developed interfaces and services work as required.

5.3. Current and future use-cases

The implemented collaborative environment acts as a template which can be used for the setup and operation of a wide range of MDAO problems due to its generalized implementation. Currently a total of 22 projects are available, which follow the same template. Among these 22 projects are projects used by different members of the AGILE community, used to formalize their own MDAO workflows. Moreover, the project template will be used in WP4 of the AGILE research project. These projects involve the conceptual design of novel aircraft configurations: open-rotor aircraft, strut-braced aircraft box-wing aircraft, BWB aircraft and UAV. These novel aircraft configurations are illustrated in chapter 1 in figure 1.5.

5.4. Reflection and user-experiences

During the workshop session and ongoing projects in which the implemented collaborative environment is used, a lot of important feedback was gathered by the author. The overall user-experiences were good. During the workshop session not all members were familiar with KE-chain and the integrated MDO framework development process. However, due to the extensive task-descriptions and training provided most members were able to successfully complete the workshop in the allocated time. As concluded by T. Lefebvre [2] the currently deployed collaborative environment is ready to be called the first release of the intended collaborative environment to support in performing the steps identified as the MDO framework development process.

According to an AGILE member and extensive user of the framework during various projects B. Aigner [3] *"the implemented process is really helpful setting up complex MDO problems. The traditionally very complex and highly manual tasks of defining the problem, finding the right tools to solve the problem with and connecting them with each other in the right way got a lot easier. Doing this in a knowledge based software environment helps the MDO integrator to really focus on the important things and let the computer do the cumbersome tasks. It is very easy to do from the beginning because the integrator is guided nicely throughout the whole process with a step-by-step approach and very well explained tutorials for each step."*.

On the downside, there are still improvements which can be made in the future as concluded from the user-experiences. It was found that KE-chain, or the server on which KE-chain was running, has trouble in processing the large amount of information. This results in long waits, which can be considered as a "deal-breaker" in using the environment in the context of the large MDO use-cases which need to be implemented in WP4. This is acknowledged by AGILE member and lead developer of KADMOS, I. van Gent: *"adding new elements to the C3PRO model takes a long time. This is really a deal-breaker for performing collaborative MDO of large systems. It should take seconds to add hundreds of variables, not minutes (or hours)"*. The author acknowledges these issues in section 5.1. Feedback regarding such performance issues is important for the development team of KE-chain. Therefore all issues have been placed of the KE-chain road-map. This way solutions can proposed and implemented to fix these issues for future projects.

Another improvement based on user experience is the feedback users get when utilizing the developed PD/AD layer interfaces. As stated by B. Aigner: *"if something does not work out the way it should (e.g. creation of FPG from an RCG), it is sometimes hard to debug the problem"*. Based on this feedback the author has already implemented improved message logs, which are fed back to the users during service task execution. These new features however have not yet been deployed in the current collaborative environment, but show great promise of improving the user-experience in future use-cases.

[2]Department of System Design and Performance Evaluation, the French Aerospace Lab (ONERA)
[3]PhD Researher, Institute of Aerospace Systems, RWTH Aachen University

6 Conclusions and Recommendations

6.1. Introduction

Increasing complexity of engineering problems and corresponding modeling and analysis activities required to explore novel aircraft designs and configurations lead to interesting challenges in the domain of Multi-disciplinary Design Optimization (MDO). As MDO frameworks, capable of performing overall aircraft design studies become too complex to be comprehended by a single team of experts, there is an increasing need to develop distributed analysis frameworks in which both tools and experts are integrated in a single distributed network. This collaborative design in distributed teams of engineers and tools is characterized as the 3^{rd} generation of MDO. During this research it was found the distribution of the different actors such as architects, integrators, discipline specialists, collaboration engineers and the customer poses great challenges on the ability to control and collaborate on the setup and operation of these 3^{rd} generation MDO frameworks.

This thesis research has been performed as part of European research project AGILE. AGILE targets 3^{rd} generation MDO frameworks. The project is set up to proof a speed up of 40% for solving realistic MDO problems compared to today's state-of-the-art. In this approach AGILE tries to target the three main stages involved in any design and optimization process: the setup, operation and solution phases. This research aims to achieve a reduction in project lead-time, by mitigating challenges implied mainly with the setup and formalization of the MDO framework.

In this research an approach to tackle these challenges is made with the integration of a business process to assist the various actors in setting up of an MDO framework in a distributed setting. This resulted in the implementation of a collaborative environment to support in the setup of 3^{rd} generation of MDO frameworks. The implementation and the subsequent research has been driven by the following research objective: *Support the setup of 3^{rd} generation MDO frameworks by developing a collaborative environment in which the activities leading to an automated design workflow are integrated in the business process layer.* This yielded the following sub-objectives:

- **Sub-objective 1:** Enable the automatic integration of tools and the data model by reuse of standardized product and process models.
- **Sub-objective 2:** Provide feedback to the architect during integration of the MDO framework components by enabling in-the-loop model verification.
- **Sub-objective 3:** Enable inspection on the MDO framework status of integration by developing a GUI to support MDO framework development in the AGILE project.

These sub-objectives are explained in more detail in the sections 6.2 to 6.4. Finally a general conclusion of this research is presented in section 6.5.

6.2. Sub-objective 1

The developed application complies with CPACS and CMDOWS standards adopted within the AGILE research project. This enables actors to reuse and enrich already acquired knowledge regarding the integration of their tools in the MDO framework. Reuse of standardized models enables acceleration of current and future product development processes. This is a fundamental aspect to which any MDO framework should comply with, in order to be maintainable within an organization.

The application developed during this research facilitates the reuse of and manipulation of product models using CPACS XML formatted documents and process models using CMDOWS XML formatted documents. Moreover validated models can be stored and retrieved from an implemented knowledge library. Therefore it can be concluded that the first sub-objective has been accomplished.

6.3. Sub-objective 2

The developed model verification and model inspector modules generate real time verification on the completeness, relevance and consistency of the properties defined in the product model and the exchange of these properties in the process model. This supports the architect during the coupling of tools. The system modules assert when data is not able to flow through the system of tools. Unfortunately the currently implemented model inspector becomes less readable for frameworks with a large amount of properties. Hence, improvements on the current implementation is advised. However, as concluded from workshop sessions the current implementation of the model inspector enables an architect with limited knowledge on the actual system of tools to identify any inconsistencies in the initial framework setup. Hence, it can be concluded that the second sub-objective has been full-filled.

6.4. Sub-objective 3

This research shows the importance of a GUI in the setup and operation phases of any MDO framework. The application developed during this research presents a reusable and maintainable environment to support the setup of the MDO framework. This application complies to a formalized approach adopted within the AGILE research project. As concluded from the workshop sessions, integration of the MDO framework can be monitored and controlled during the different steps of the development process. Therefore it can be concluded that the third sub-objective has been met.

6.5. Conclusions

The results presented in chapter 5 show a reusable and maintainable environment to support the setup of MDO frameworks. This environment is tested for several small use-cases, but is currently used to setup a large MDO framework with 43 different project members. It can be seen that a trend has been followed in which a lot of information, usually only accessible by each individual engineer or specialist in their individual domain of operation, is integrated in the business process layer. From here, monitoring of progress on the framework setup and collaboration between distributed teams of experts is supported.

Although a lot of functionality has been developed in the context of the AGILE research project the developed functionality is likely to open up new possibilities not addressed in this paper. These possibilities might lie outside the domain of supporting the setup of 3^{rd} generation MDO frameworks, but on the management of other information intensive business processes. This can be concluded from the modular implementation and strong focus on maintainability and user-friendliness.

To conclude, an approach has been made to identify and improve on the missing functionality implemented in the business process layer in context of the AGILE research project. Hereby enabling support in the development of 3^{rd} generation MDO frameworks. The implementation of an MDO framework to support a wing design case study demonstrates the flexibility of the developed application. This is endorsed by various experts using the environment. The implemented collaborative environment is currently being used to setup various use-cases in the context of the AGILE research project. The continued use of the work that has been done during this thesis research makes the author very enthusiastic.

6.6. Limitations and recommendations

The current implementation is tested with various end-users within the context of the AGILE project. Some improvements are advised to cope with large design and optimization problems:

- Improved performance in response times of the application.
- Improved feedback on the status of execution of the underlying services.

During the workshop demonstrations, it was found that the performance of the developed application decreased as multiple end-users work simultaneously on different use-cases. In the current interface between the AD and PD layer, the execution status of underlying services is not communicated sufficiently well to the end-user.

Server time-outs, long loading times and incomplete feedback on the status of execution of underlying services diminish the control end-users are experiencing. Hence, it is recommended that effort is made to improve the overall performance of the developed application and feedback to the end-user on the execution status of automated tasks in the PD process. These recommendations however are all included in the KE-chain road-map and worked on by the developers of KE-chain.

Finally it is recommended to extend the implemented collaborative environment to allow for more automated interfaces in step four of the MDO framework development process. This has currently not been made possible due to the limited time and limited compliance of the CMDOWS schema in PIDO applications RCE & Optimus used within context of the AGILE research project. As ongoing improvements on CMDOWS, RCE and Optimus are currently made within the AGILE consortium to enable improved integration in the MDO framework development process, it is expected that an extension in automated PD/AD interfaces is possible in the near future. In this integration, a lot of the developed methods and interfaces during this research can be reused.

Appendices

A
Information Quality categories

As introduced in section 3.1, lean information management principles have the potential to accelerate any PDP or other type of information intensive processes. A lean process is characterized by a maximized value of information which is generated or retrieved with minimized waste. Waste is considered deadly in the field of information management. This so-called waste refers to activities which are required to overcome missing information, wrong information, a bad exchange of information or an excessive amount of information. Waste impacts the PDP through extended project duration and cost as additional resources such as man-hours need to be allocated.

Studies performed by Wang *et al.* [45] and Hicks [14] describe the value of information and the different categories which can be used to qualify information. In this research Information Quality (IQ) categories are used to identify potential areas which can be improved upon in the implementation of a collaborative environment with respect to the current state of collaboration and exchange of information within AGILE.

In table A.1 a list of Information Quality (IQ) categories defined by Wang *et al.* [45] are presented. This table shows twenty IQ categories. For each IQ category a description and example based on AGILE is provided. The author has analyzed the different IQ categories on their applicability within the scope of this thesis research and their applicability within the AGILE project. The implemented collaborative environment which is the result of this thesis research strives to maximize the value of information by addressing IQ categories relevancy, interpretability, ease of understanding, accessibility, completeness, traceability, representational consistency, variety of data and data sources, concise, appropriate amount of data and flexibility. All other IQ categories are either not in scope of this thesis research or applicable within the context of the AGILE project. A IQ category is not in scope of this research if there is no means to assess or manage the quality of information w.r.t this category. A IQ category which is not applicable with the context of the AGILE project is deemed to have a low impact on the overall performance of the design and optimization processes.

Table A.1: Different Information Quality (IQ) categories according to Wang *et al.* [45] and derived examples based on the AGILE project

- (*) - IQ category not within the scope of this thesis research and does not require additional attention in the AGILE project
- (**) - IQ category not within the scope of this thesis research but is interesting within the context of the AGILE project.
- (***) - IQ category important in context of AGILE and incorporated in this thesis research.

#	**IQ Category**	**Description**	**Example based on AGILE**
1	Believability (**)	The extent to which data are accepted or regarded as true, real and credible.	Believability is an important IQ category with respect to the context of AGILE. For example all (intermediate) results need to be believable. Believebabiliy is largely the result of: • Complete transparency and dependency of all intermediate steps that have let to (intermediate) results of a project cycle within AGILE. • Reduced black-box approach possible using a formalized development process in which all intermediate steps are documented and clear for all actors involved. • Extensive reporting functions and specialist reports + verification
2	Value-added (**)	The extent to which data are beneficial and provide advantages from their use.	Beneficial data is an important aspect of any development process. The same goes for the AGILE project. Beneficial project data needs to be directly deducible from the use-case definition and defined key design parameters by the customer. Unfortunately the IQ category value-added is difficult to assess in the implemented collaborative environment as it is highly subjective for the different actors involved in the development process.
3	Relevancy (***)	The extent to which data are applicable and helpful for the task at hand.	The extend to which data is helpful or applicable to the task at hand can be deduced directly from: • Top Level (Aircraft) Requirements • Key design parameters • Specific input for design competences • Design concept(s) that is/are to be subjected against an MDO problem
4	Accuracy (**)	The extent to which data are correct, reliable, and certified free of error.	In the final phases of the MDO framework development process design results are inspected. The accuracy of the MDAO results need to adhere to an accuracy and implied level of fidelity which is in satisfaction with the demands of the customer or use-case owner.

5	Interpretability (***)	The extent to which data are in appropriate language and units and the data definitions are clear.	In the MDO framework development process potential issues arise in step four, the implementation and deployment of an executable simulation workflow, if wrong interpretation of information and data sources are done in previous steps. For this purpose the collaborative environment should support in visualization of dependence between information and data. Clear documentation of information sources, accessible to the architect and all discipline specialists aim to improve interpretability.
6	Ease of understanding (***)	The extent to which data are clear without ambiguity and easily comprehended.	Whether data can be comprehended determines on the role of the person requesting the information. Within AGILE multiple types of actors are involved in a single design and optimization process. This increases the risk of information which is not easily understandable for all actors involved. Adhering to standards (CMDOWS, CPACS) aims to enhance the understanding of underlying methods and models. Moreover clear documentation and explanation of information intensive design activities should aim to improve the ease of understanding.
7	Accessibility (***)	The extent to which data are available or easily and quickly retrievable.	Accessibility is one of the core values of KE-chain. Availability and easy access to only the information one needs to perform its work is essential, also within the AGILE project. For all actors involved within the AGILE project it is expected that information is accessible for the actor to perform its job. Moreover, accessibility of data is of importance to disciplinary specialists who are required to integrate their disciplinary tools in a complete and consistent design process. Step two of the formalized MDO framework development process aims to contain functions to assess whether data required as input data is accessible for each disciplinary tool.
8	Objectivity (*)	The extent to which data are unbiased (unprejudiced) and impartial.	Information in KE-chain and within the scope of AGILE is to be highly subjective for different use-cases. Some objectivity however is required to ensure that the (intermediate) design results are generated using a similar approach. This way all design studies can be interpreted with a similar level of objectivity even though case-specific design results might not be comparable. For example, the output data of a low-fidelity box-wing aircraft design is not comparable to a high fidelity BWB design. The process leading to the case-specific design results however need to be be subjected to objectivity.

9	Timeliness (**)	The extent to which the age of the data is appropriate for the task at hand.	In order to do multi-site, multi-collaborative work up-to-date information is key. Whether information is useful depends on its dependence to earlier information. If this chain remained constant over time it can be assumed that this information is appropriate for the task at hand.
10	Completeness (***)	The extent to which data are of sufficient breadth, depth, and scope for the task at hand.	Completeness of the models used in the AGILE framework are highly use-case specific. The scope of work is determined in the formalized MDO development framework development process. The repository of design competence that is assembled by the architect and discipline specialists needs to be complete with respect to the use-case defined by the customer and architect. Iterations are required in order to define a complete set of design competences and data model that is able to solve the design problem defined by the customer.
11	Traceability (***)	The extent to which data are well documented, verifiable, and easily attributed to a source.	Traceability of data and information is very important given the fact that information is (re-)used throughout the different steps in the development process.
12	Reputation (*)	The extent to which data are trusted or highly regarded in terms of their source or content.	With the AGILE project, the reputation of disciplinary specialist, architects or integrator might impact the acceptance of information. Trustworthiness of information needs to be improved by using a standardized approach of acquiring information. Re-use of information and underlying models using predictable and standardized solutions during the MDO framework development process can be used to increase the reputation of data and information.
13	Representational consistency (***)	The extent to which data are always presented in the same format and are compatible with previous data.	Representational consistency is striven after by using modeling standards such as CMDOWS and CPACS. The use of modeling standards promotes the consistency of the information generated during a development process. Changes can be throughput more rapidly if the representation of data is consistent throughout different design and optimization processes.
14	Cost-effectiveness (**)	The extent to which the cost of collecting appropriate data is reasonable.	In AGILE one of the challenges that is to be overcome is to reduce the project lead-time required to setup a repository of design competences & data model which is consistent and complete with respect to a specific use-case and the formalization of an executable simulation workflow. All steps leading to the formalization of an executable simulation workflow, and the execution itself, need to be efficient in cost to enhance the available time and resources to explore the design space.

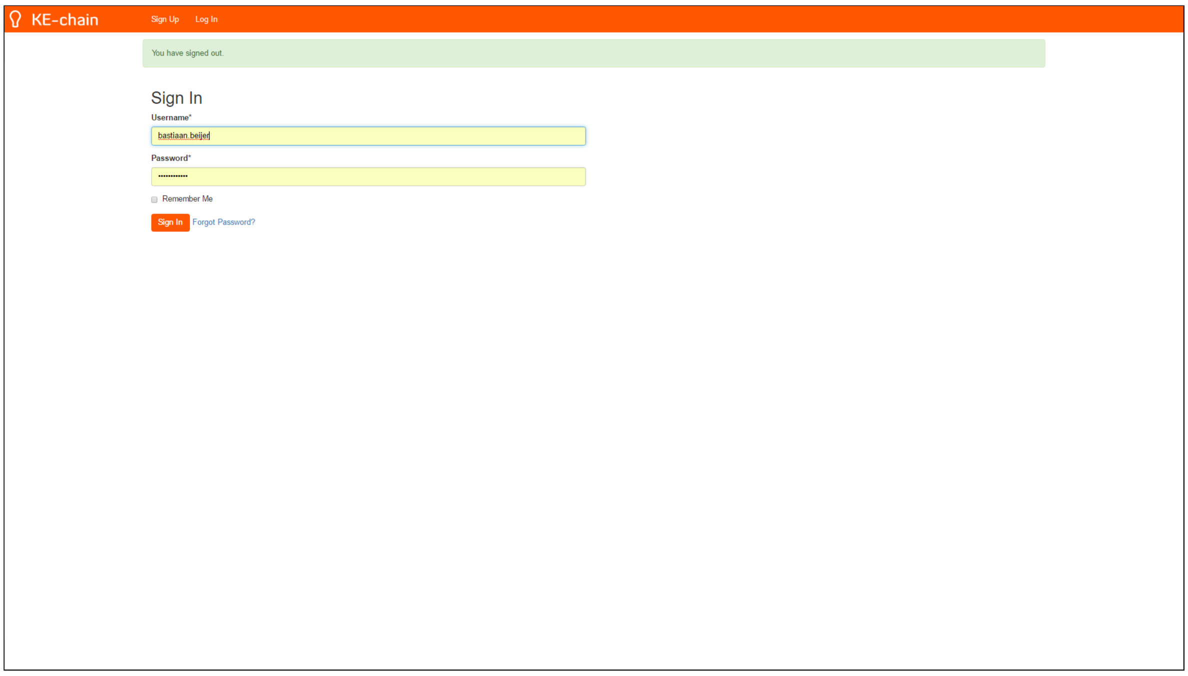
15	Ease of operation (***)	The extent to which data are easily managed and manipulated (i.e., updated, moved, aggregated, reproduced, customized).	The ease of operation important aspect of any MDO framework and the implemented collaborative environment by the author. Increased ease of operation enhances the flexibility of the setup and operation of the integrated MDO framework.
16	Variety of data and data sources (***)	The extent to which data are available from several differing data sources.	Within the different AGILE use-cases a large variety of data and data sources are encountered. For example: • Information used for top level management, defined by the customer and architect • Information regarding the various parameters and their role in the MDAO architecture • Information describing the design competences on a functional level. • Information regarding (intermediate) design study results • Information regarding standardized product and process models. Each type of information has a different source, for example: a disciplinary specialist, architect or external application generating that information.
17	Concise (***)	The extent to which data are compactly represented without being overwhelming (i.e., brief in presentation, yet complete and to the point).	One of the challenges in trying to visualize very large sets of information. It would be a real challenge to remain concise in showing only the actual information needed.
18	Access Security (**)	The extent to which access to data can be restricted and hence kept secure.	Handled by handing out permissions to the different partners involved within the project. This permission system and the used of sub-scopes within the entire project ensures one can manage the accessibility and editabilty of information. A large part of access security is handled by KE-chain through user-authentication.
19	Appropriate amount of data (***)	The extent to which the quantity or volume of available data is appropriate.	This highly depends on the level of fidelity and the concepts to be investigated. The appropriate amount of data is closely coupled to the completeness of the repository of design competences and data model.
20	Flexibility (***)	The extent to which data are expandable, adaptable, and easily applied to other needs	Flexibility is one of the goals within AGILE. Moreover the collaborative environment which is implemented during this research needs to provide flexibility to improve design space exploration and the ability to support different design use-cases.

B
AGILE collaborative environment

In this appendix an impression of the Agile collaborative environment developed by the author during this thesis research is presented. This chapter is composed of screenshots coming from a completed Agile MDO framework development process used during the workshop demonstrations held on the 4th-5th-6th of April. A detailed explanation is provided explaining the expected behavior of the various actors and end-users of the application.

B.1. User authentication

A user enters the Agile collaborative environment through the KE-chain login screen which is displayed in figure B.1. Each user can log in using personal log in credentials. Based on user-authentication the projects a user can see and the type of operations a user can perform are regulated. For example, a regular KE-chain user is able to only see projects to which he or she is assigned and a KE-chain superuser is able to see all projects.



You have signed out.

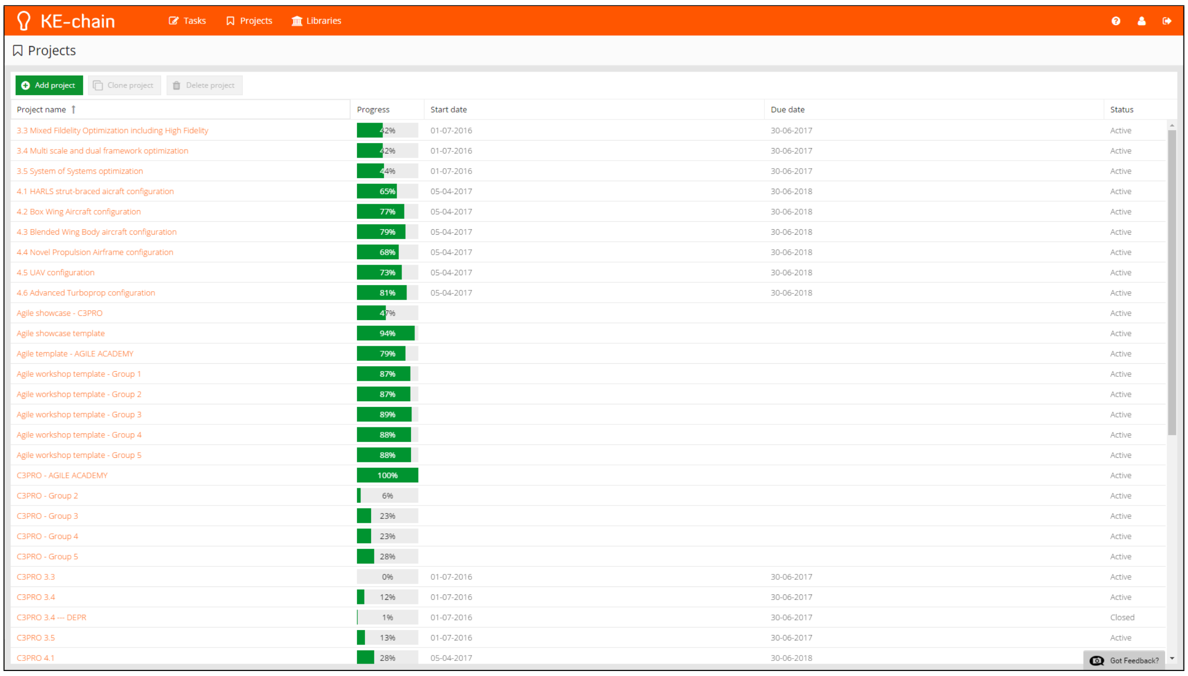
Username*: bastiaan.beijer

Password*: ••••••••••••

Figure B.1: KE-chain user-authenticated login screen

B.2. Implemented agile project overview

After a user is authenticated, an overview of available projects can be inspected. An overview of all type of KE-chain projects for various design cases are displayed in figure B.2. This overview shows the progress of each project and optionally a start and due date. The progress of each project is automatically calculated, based on the amount of information filled in. Selection of a project enables inspection of its contents, configuration of the project and participation in the project activities.



Project name	Progress	Start date	Due date	Status
3.3 Mixed Fildelity Optimization including High Fidelity	42%	01-07-2016	30-06-2017	Active
3.4 Multi scale and dual framework optimization	42%	01-07-2016	30-06-2017	Active
3.5 System of Systems optimization	44%	01-07-2016	30-06-2017	Active
4.1 HARLS strut-braced aicraft configuration	65%	05-04-2017	30-06-2018	Active
4.2 Box Wing Aircraft configuration	77%	05-04-2017	30-06-2018	Active
4.3 Blended Wing Body aircraft configuration	79%	05-04-2017	30-06-2018	Active
4.4 Novel Propulsion Airframe configuration	68%	05-04-2017	30-06-2018	Active
4.5 UAV configuration	73%	05-04-2017	30-06-2018	Active
4.6 Advanced Turboprop configuration	81%	05-04-2017	30-06-2018	Active
Agile showcase - C3PRO	47%			Active
Agile showcase template	94%			Active
Agile template - AGILE ACADEMY	79%			Active
Agile workshop template - Group 1	87%			Active
Agile workshop template - Group 2	87%			Active
Agile workshop template - Group 3	89%			Active
Agile workshop template - Group 4	88%			Active
Agile workshop template - Group 5	88%			Active
C3PRO - AGILE ACADEMY	100%			Active
C3PRO - Group 2	6%			Active
C3PRO - Group 3	23%			Active
C3PRO - Group 4	23%			Active
C3PRO - Group 5	28%			Active
C3PRO 3.3	0%	01-07-2016	30-06-2017	Active
C3PRO 3.4	12%	01-07-2016	30-06-2017	Active
C3PRO 3.4 --- DEPR	1%	01-07-2016	30-06-2017	Closed
C3PRO 3.5	13%	01-07-2016	30-06-2017	Active
C3PRO 4.1	28%	05-04-2017	30-06-2018	

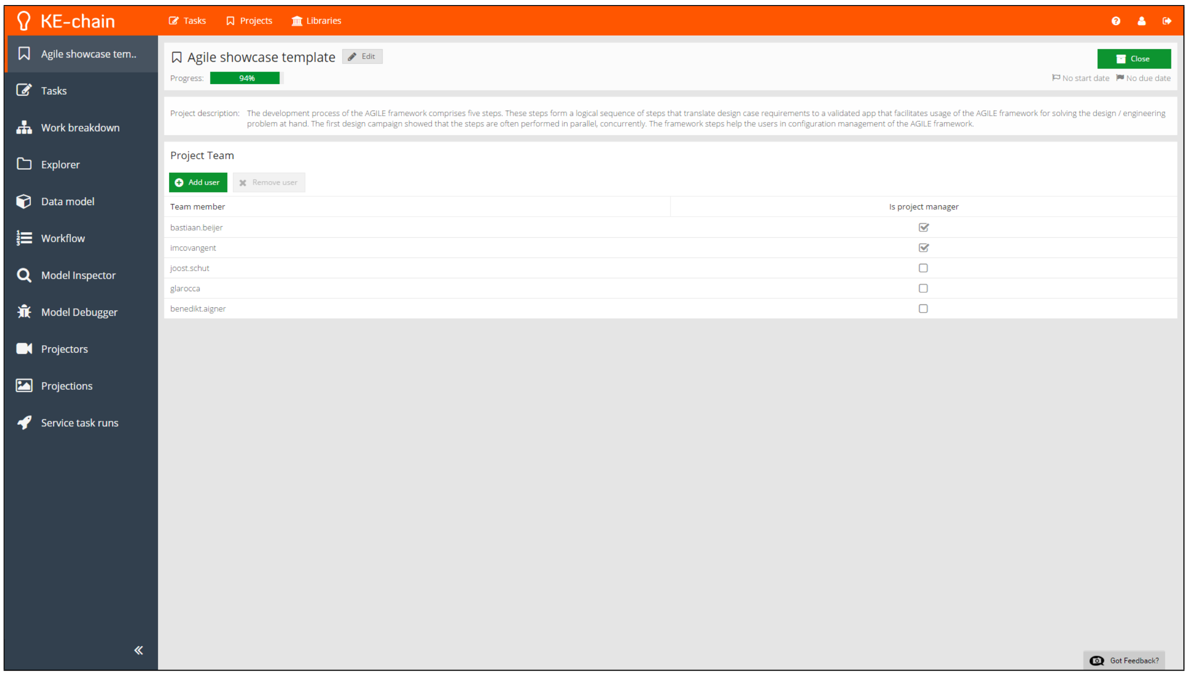
Figure B.2: Project overview

B.2.1. Agile MDO framework development process application home screen

The selection of a project navigates the user to the project's landing page displayed in figure B.3. This landing page shows a detailed overview of the project's description and its assigned project members. Multiple users can be assigned to a single project as member. All assigned project members are able to view and edit information contained in a project. A special type of user, the so-called project manager, is able to change the users assigned to the project and the configuration of all activities and underlying data model.

On the left side of the project view, a navigation bar can be seen. This navigation bar enables quick navigation between a project's main views and tools. The main views a user is able to navigate to are described as follows:

- **Details overview**: This view contains the project's landing page. This view shows an overview of the assigned project members and a detailed project's description.
- **Tasks overview**: This view contains a list of all tasks which are part in scope of this project. In this view, a project manager is able to assign project members to a specific task. Each member is able to filter all project's activities based on tasks which are specifically assigned to them. Assigning specific members to specific tasks enables distribution of responsibilities. In the context of the Agile MDO framework development process this feature is not yet used, as there is a shared responsibility for multiple tasks.
- **Work breakdown**: This view displays the project's tasks in a hierarchical fashion. The work breakdown is used to get a quick overview of each task's progress and navigation to a particular task
- **Explorer**: This view enables inspection of all data contained in the project. This project explorer shows the instances of a pre-configured data model.



Project description: The development process of the AGILE framework comprises five steps. These steps form a logical sequence of steps that translate design case requirements to a validated app that facilitates usage of the AGILE framework for solving the design / engineering problem at hand. The first design campaign showed that the steps are often performed in parallel, concurrently. The framework steps help the users in configuration management of the AGILE framework.

Team member	Is project manager
bastiaan.beijer	✓
imcovangent	✓
joost.schut	☐
glarocca	☐
benedikt.aigner	☐

Figure B.3: Agile MDO framework development process landing page

- **Data model**: This view enables inspection to the underlying data model defined for a project. This data model is used as a template for all information and data stored in this project.
- **Workflow**: This view enables inspection of the business process which can be defined for this project. A business process is composed of all tasks and sub-processes displayed in the work breakdown, however ontological order can be specified. This enables inspection in the business process activities.

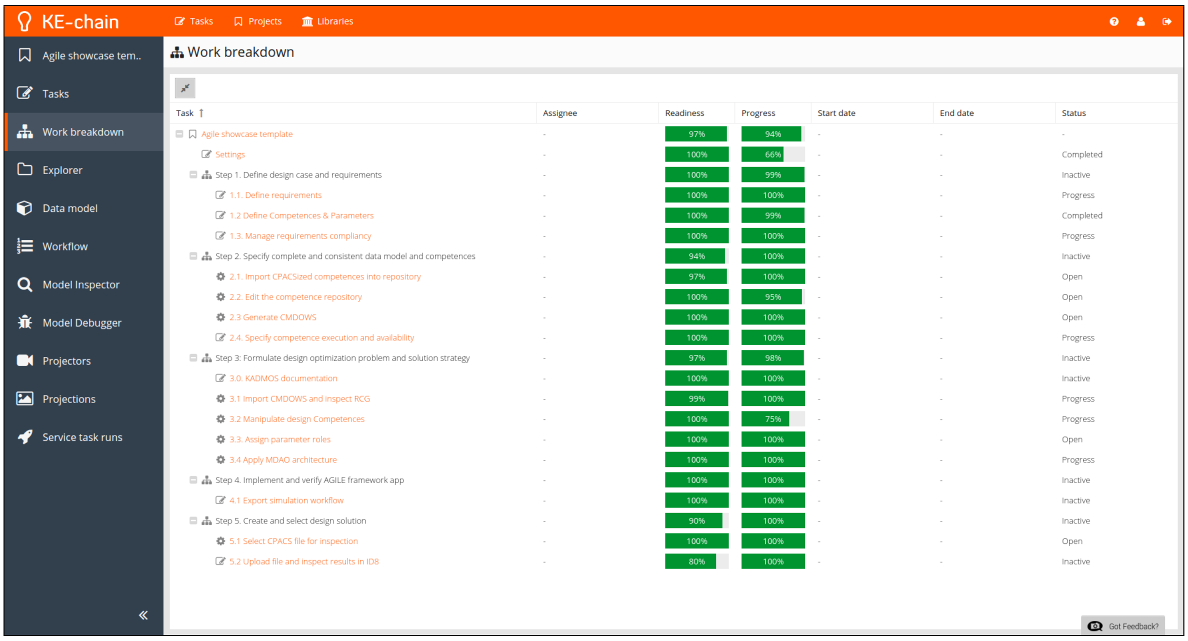
A project member also has access to various tools which allow for further inspection of the underlying process model and data model defined for a project. Moreover views are provided to control various projector scripts and view the execution status of service tasks. These service tasks are integrated in the business process and are coupled to external tools or python scripts using KE-node interfaces. The latter is explained in more detail in section 2.4.1. The provided tools can be summarized as follows:

- **Model inspector**: This view enables the inspection of the coupling between tasks in the business process and detailed exchange of information and data between tasks. This view is used to assist in the configuration management of a project. It assists the project manager in visualizing the flow of information, such that measures can be taken to mitigate waste in the development process: reducing the risk of information not being able to flow between tasks.
- **Model debugger**: This view gives an overview of potential bugs in the project's configuration. These bugs are categorized and collected in a list. For example, if information is required as input for task A, but it is not provided by any other task, there is a risk of flawed flow: information is not able to flow between the project's activities. This view assists the project manager in the configuration management of the project.
- **Projectors**: This view contains an overview of all projectors defined in the project and enables the creation of new projector scripts. Projectors are required to map input and output properties in a service task to the underlying service, or as stand-alone scripts which are used to visualize, share or write information in various formats such as: JSON, CSV, XML, HTML or latex.
- **Projections**: This view contains an overview of all executed projector scripts and provides the interface to make new projections. Moreover, projectors can be executed to unique URLs. These URLs can be retrieved from this view. These URLs can be used to create embedded visualizations in tasks forms, as discussed in section 4.2.1, or download the output content of projection.

- **Service task runs**: This view enables inspection in executed service tasks. An overview is provided to see the history log of service task executions or the status of currently running services.

B.2.2. Business process activities breakdown

In this section the project's work breakdown is described. An overview of the hierarchical overview of all business process activities can be accessed through this view as depicted in figure B.4. This view shows the progress of each individual task or sub-process. Progress is calculated based on the amount of information filled in for each task. Moreover this view shows the readiness of a task. The readiness describes the amount of input information which is provided, as a percentage of the required input information. The percentage of readiness acts as an incentive to perform work in that task.



Work breakdown

Task	Assignee	Readiness	Progress	Start date	End date	Status
Agile showcase template	-	97%	94%	-	-	-
Settings	-	100%	66%	-	-	Completed
Step 1. Define design case and requirements	-	100%	99%	-	-	Inactive
1.1. Define requirements	-	100%	100%	-	-	Progress
1.2 Define Competences & Parameters	-	100%	99%	-	-	Completed
1.3. Manage requirements compliancy	-	100%	100%	-	-	Progress
Step 2. Specify complete and consistent data model and competences	-	94%	100%	-	-	Inactive
2.1. Import CPACSized competences into repository	-	97%	100%	-	-	Open
2.2. Edit the competence repository	-	100%	95%	-	-	Open
2.3 Generate CMDOWS	-	100%	100%	-	-	Open
2.4. Specify competence execution and availability	-	100%	100%	-	-	Progress
Step 3: Formulate design optimization problem and solution strategy	-	97%	98%	-	-	Inactive
3.0. KADMOS documentation	-	100%	100%	-	-	Inactive
3.1 Import CMDOWS and inspect RCG	-	99%	100%	-	-	Progress
3.2 Manipulate design Competences	-	100%	75%	-	-	Progress
3.3. Assign parameter roles	-	100%	100%	-	-	Open
3.4 Apply MDAO architecture	-	100%	100%	-	-	Progress
Step 4. Implement and verify AGILE framework app	-	100%	100%	-	-	Inactive
4.1 Export simulation workflow	-	100%	100%	-	-	Inactive
Step 5. Create and select design solution	-	90%	100%	-	-	Inactive
5.1 Select CPACS file for inspection	-	100%	100%	-	-	Open
5.2 Upload file and inspect results in ID8	-	80%	100%	-	-	Inactive

Figure B.4: Impression of the developed KE-chain work breakdown structure

In the work breakdown displayed in figure B.4 shows five sub-processes. These five sub-processes follow the five steps of the MDO framework development process. A different representation of these five steps can be seen in figure B.5. This figure shows a business process representation of the MDO framework development process. Through the workflow view, the user is able to navigate to the various sub-processes. Using exclusive-or gateways and parallel gateways the execution order can be visualized. This view has been integrated by the author in KE-chain during this thesis research. This integration has been accelerated using already implemented workflow components in previous editions of KE-chain.

Selecting a task from the business process enables navigation to that task. The individual task forms belonging to each sub-task are presented in the following sections starting with the subtask of step 1 of the MDO framework development process.

B.3. Step 1: Define design case and requirements

In this section the different views associated with step 1 in the MDO framework development process are visualized. In figure B.6 the taskform associated with task 1.1 is displayed. This task is the starting point of the MDO framework development process. As can be seen from this view, the task's layout is customized such that it displays a grid of requirements. Requirements can be added by the customer, and completed by the architect.

In figure B.7 task 1.2 is displayed. In this taskform the requirements specified by the customer and architect can be inspected and design competences required in the design case can be added using the respective grid. Moreover, the key design parameters can be added through the provided interface. This task form is design such that information can be added through forms, and information is displayed in grids. After com-

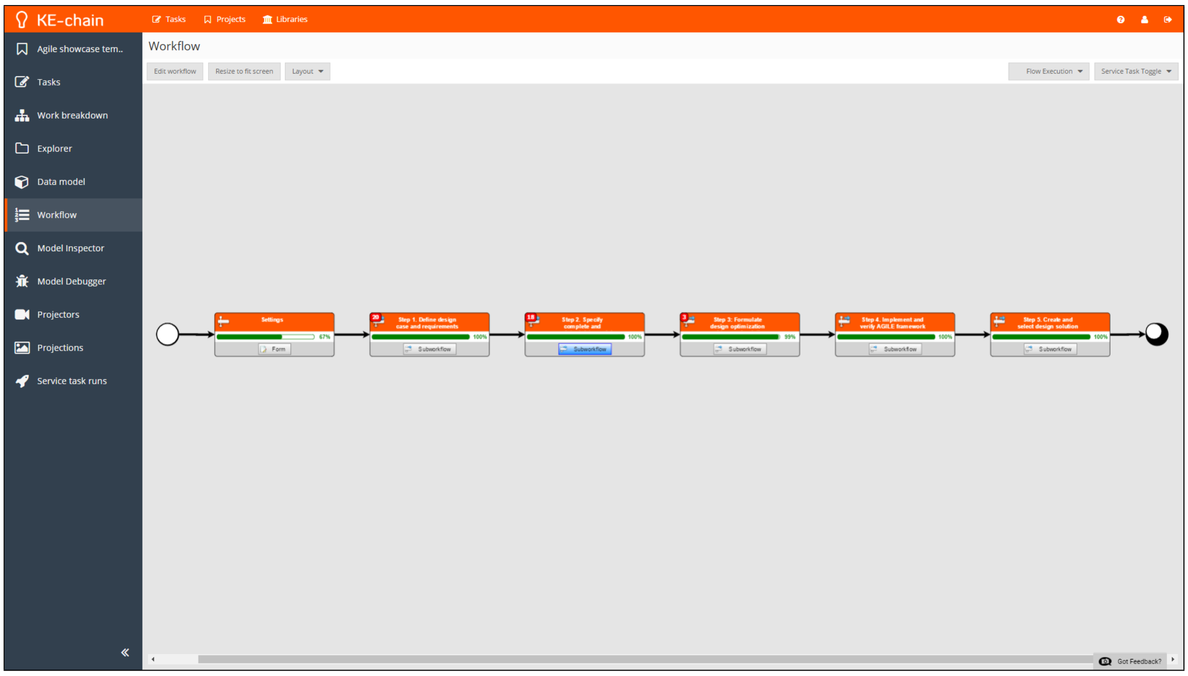
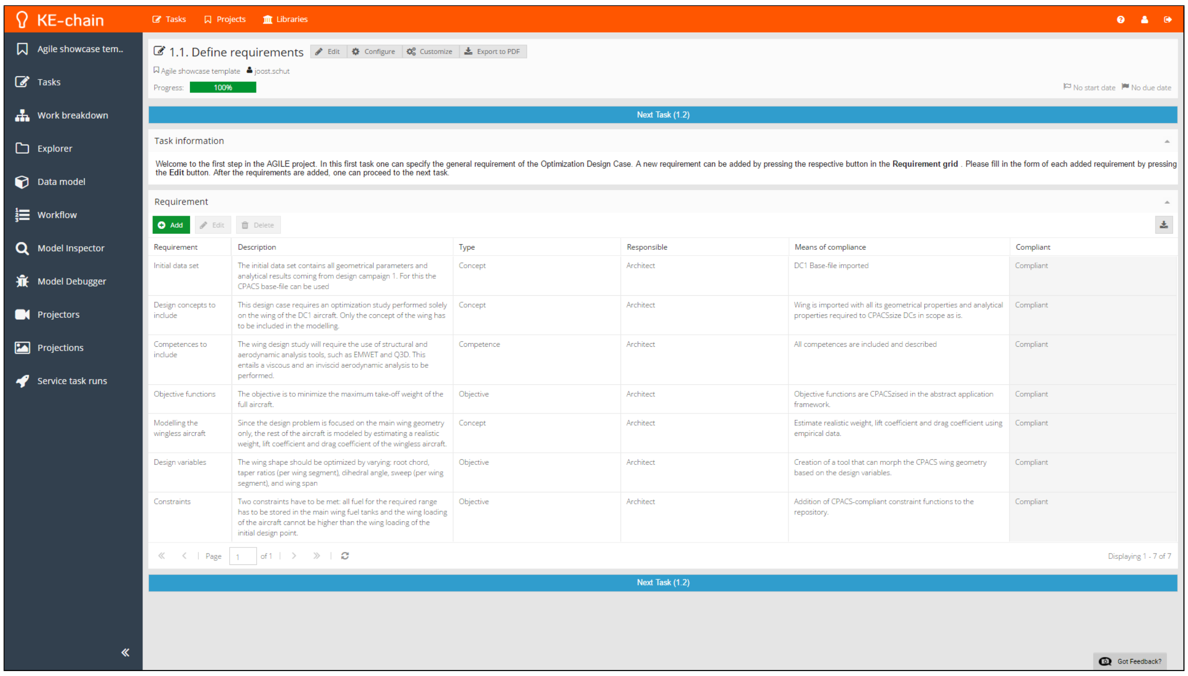


Figure B.5: Impression of the developed business process top level overview integrated in KE-chain



1.1. Define requirements

Agile showcase template joost.schut

Progress: 100%

No start date No due date

Next Task (1.2)

Task information

Welcome to the first step in the AGILE project. In this first task one can specify the general requirement of the Optimization Design Case. A new requirement can be added by pressing the respective button in the **Requirement grid** . Please fill in the form of each added requirement by pressing the **Edit** button. After the requirements are added, one can proceed to the next task.

Requirement

Requirement	Description	Type	Responsible	Means of compliance	Compliant
Initial data set	The initial data set contains all geometrical parameters and analytical results coming from design campaign 1. For this the CPACS base-file can be used	Concept	Architect	DC1 Base-file imported	Compliant
Design concepts to include	This design case requires an optimization study performed solely on the wing of the DC1 aircraft. Only the concept of the wing has to be included in the modelling.	Concept	Architect	Wing is imported with all its geometrical properties and analytical properties required to CPACSsize DCs in scope as is.	Compliant
Competences to include	The wing design study will require the use of structural and aerodynamic analysis tools, such as EMWET and Q3D. This entails a viscous and an inviscid aerodynamic analysis to be performed.	Competence	Architect	All competences are included and described	Compliant
Objective functions	The objective is to minimize the maximum take-off weight of the full aircraft.	Objective	Architect	Objective functions are CPACSzised in the abstract application framework.	Compliant
Modelling the wingless aircraft	Since the design problem is focused on the main wing geometry only, the rest of the aircraft is modeled by estimating a realistic weight, lift coefficient and drag coefficient of the wingless aircraft.	Concept	Architect	Estimate realistic weight, lift coefficient and drag coefficient using empirical data.	Compliant
Design variables	The wing shape should be optimized by varying: root chord, taper ratios (per wing segment), dihedral angle, sweep (per wing segment), and wing span	Objective	Architect	Creation of a tool that can morph the CPACS wing geometry based on the design variables.	Compliant
Constraints	Two constraints have to be met: all fuel for the required range has to be stored in the main wing fuel tanks and the wing loading of the aircraft cannot be higher than the wing loading of the initial design point.	Objective	Architect	Addition of CPACS-compliant constraint functions to the repository.	Compliant

Page 1 of 1

Displaying 1 - 7 of 7

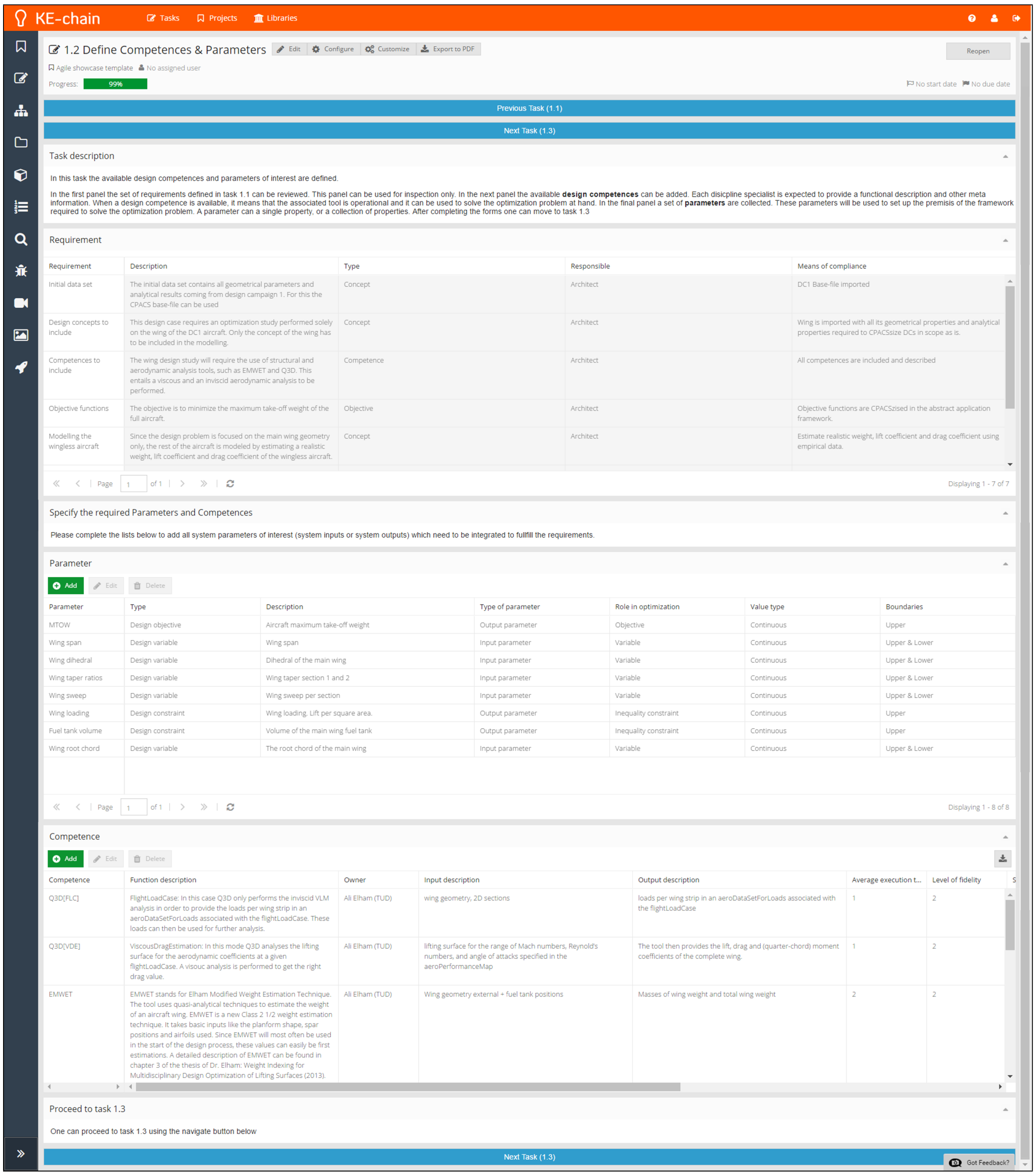
Next Task (1.2)

Figure B.6: Impression of the developed KE-chain task-form of task 1.1: Define requirements

pleting the associated forms of task 1.2. a complete description of the design case is defined.

B.3.1. Step 2: Specify complete and consistent data model and competences

In this section the KE-chain forms associated with the second step in the MDO framework development process is presented. First in figure B.8 the task form associated associated with task 2.1 is shown. This task is about coupling the DCs to a common CPACS data model using automated input/output mapping. The collection of DCs coupled to a common CPACS data model is a repository used as a knowledge-base for step 3.1.



1.2 Define Competences & Parameters

Agile showcase template No assigned user

Progress: 99%

No start date No due date

Task description

In this task the available design competences and parameters of interest are defined.

In the first panel the set of requirements defined in task 1.1 can be reviewed. This panel can be used for inspection only. In the next panel the available **design competences** can be added. Each disicpline specialist is expected to provide a functional description and other meta information. When a design competence is available, it means that the associated tool is operational and it can be used to solve the optimization problem at hand. In the final panel a set of **parameters** are collected. These parameters will be used to set up the premisis of the framework required to solve the optimization problem. A parameter can a single property, or a collection of properties. After completing the forms one can move to task 1.3

Requirement

Requirement	Description	Type	Responsible	Means of compliance
Initial data set	The initial data set contains all geometrical parameters and analytical results coming from design campaign 1. For this the CPACS base-file can be used	Concept	Architect	DC1 Base-file imported
Design concepts to include	This design case requires an optimization study performed solely on the wing of the DC1 aircraft. Only the concept of the wing has to be included in the modelling.	Concept	Architect	Wing is imported with all its geometrical properties and analytical properties required to CPACSsize DCs in scope as is.
Competences to include	The wing design study will require the use of structural and aerodynamic analysis tools, such as EMWET and Q3D. This entails a viscous and an inviscid aerodynamic analysis to be performed.	Competence	Architect	All competences are included and described
Objective functions	The objective is to minimize the maximum take-off weight of the full aircraft.	Objective	Architect	Objective functions are CPACSzised in the abstract application framework.
Modelling the wingless aircraft	Since the design problem is focused on the main wing geometry only, the rest of the aircraft is modeled by estimating a realistic weight, lift coefficient and drag coefficient of the wingless aircraft.	Concept	Architect	Estimate realistic weight, lift coefficient and drag coefficient using empirical data.

Page 1 of 1 Displaying 1 - 7 of 7

Specify the required Parameters and Competences

Please complete the lists below to add all system parameters of interest (system inputs or system outputs) which need to be integrated to fullfill the requirements.

Parameter

Parameter	Type	Description	Type of parameter	Role in optimization	Value type	Boundaries
MTOW	Design objective	Aircraft maximum take-off weight	Output parameter	Objective	Continuous	Upper
Wing span	Design variable	Wing span	Input parameter	Variable	Continuous	Upper & Lower
Wing dihedral	Design variable	Dihedral of the main wing	Input parameter	Variable	Continuous	Upper & Lower
Wing taper ratios	Design variable	Wing taper section 1 and 2	Input parameter	Variable	Continuous	Upper & Lower
Wing sweep	Design variable	Wing sweep per section	Input parameter	Variable	Continuous	Upper & Lower
Wing loading	Design constraint	Wing loading. Lift per square area.	Output parameter	Inequality constraint	Continuous	Upper
Fuel tank volume	Design constraint	Volume of the main wing fuel tank	Output parameter	Inequality constraint	Continuous	Upper
Wing root chord	Design variable	The root chord of the main wing	Input parameter	Variable	Continuous	Upper & Lower

Page 1 of 1 Displaying 1 - 8 of 8

Competence

Competence	Function description	Owner	Input description	Output description	Average execution t...	Level of fidelity	S
Q3D[FLC]	FlightLoadCase: In this case Q3D only performs the inviscid VLM analysis in order to provide the loads per wing strip in an aeroDataSetForLoads associated with the flightLoadCase. These loads can then be used for further analysis.	Ali Elham (TUD)	wing geometry, 2D sections	loads per wing strip in an aeroDataSetForLoads associated with the flightLoadCase	1	2	
Q3D[VDE]	ViscousDragEstimation: In this mode Q3D analyses the lifting surface for the aerodynamic coefficients at a given flightLoadCase. A visouc analysis is performed to get the right drag value.	Ali Elham (TUD)	lifting surface for the range of Mach numbers, Reynold's numbers, and angle of attacks specified in the aeroPerformanceMap	The tool then provides the lift, drag and (quarter-chord) moment coefficients of the complete wing.	1	2	
EMWET	EMWET stands for Elham Modified Weight Estimation Technique. The tool uses quasi-analytical techniques to estimate the weight of an aircraft wing. EMWET is a new Class 2 1/2 weight estimation technique. It takes basic inputs like the planform shape, spar positions and airfoils used. Since EMWET will most often be used in the start of the design process, these values can easily be first estimations. A detailed description of EMWET can be found in chapter 3 of the thesis of Dr. Elham: Weight Indexing for Multidisciplinary Design Optimization of Lifting Surfaces (2013).	Ali Elham (TUD)	Wing geometry external + fuel tank positions	Masses of wing weight and total wing weight	2	2	

Proceed to task 1.3

One can proceed to task 1.3 using the navigate button below

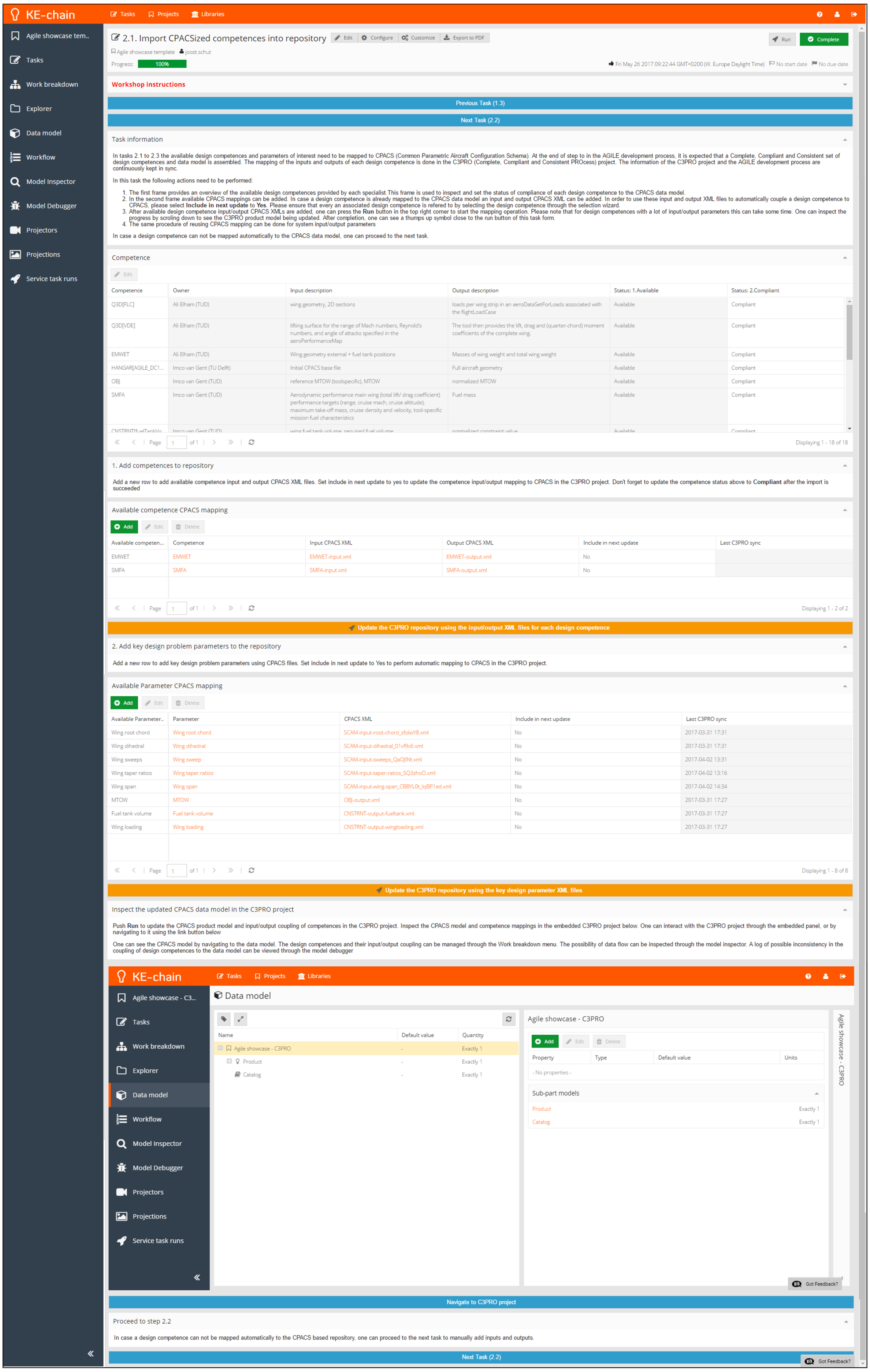
Figure B.7: Impression of the developed KE-chain task-form of task 1.2: Define competences and parameters

The discipline specialists are able to upload their CPACS input XML file and CPACS output XML file for each DC in the second grid displayed in the task-form. Pressing the orange button below it triggers the execution of the underlying service which reads the XML files and constructs the product model in an embedded KE-chain project. Moreover, it couples KE-chain activities in the C3PRO project. These coupled activities in KE-chain enable inspection of the coupled product and process models directly through KE-chain. This embedded KE-chain project is referred to as the C3PRO project in the collaborative environment. C3PRO is acronym for a Complete, Consistent and Compliant PROcess. The C3PRO project is visualized in the bottom of the taskform.

Besides the coupling of DCs inputs and outputs to a common data model, the architect is able to upload CPACS XML files for the key design parameters defined earlier in task 1.2. Coupled key design parameters appear in the C3PRO project as either input or output activities. Driving design parameters, such as design variables become input activities. Design parameters which are part of the expected outcome of a design study, such as objective and constraint variables, are coupled as output activities.

Together, the set of coupled DCs and key design parameters enable the architect to judge whether a complete, consistent and compliant process is defined which can be used in the remain steps of the MDO framework development process. In case not all DCs are compliant to the common data model, the discipline specialists are able to proceed to task 2.2 to manually map remaining input and output parameters. Manual input and output mapping is facilitated through the embedded KE-chain C3PRO project. An example of this manual input/output mapping can be seen in figure B.9. Through the widget displayed in this figure, one sees the complete product model with all its part and property models. The user is able to select input and output properties by selecting the box in the “view” and “edit” column respectively. Selecting a property also selects its ancestor part models. Figure B.9 for example shows a selection of the configuration of EMWET, a structural analysis tool developed by Ali Elham at the TU Delft. In this example the x, y & z coordinates of the root airfoil section are selected as input of EMWET.

The complete repository of DCs and data model needs to be prepared such that it can be used in step three of the MDO framework development process. Figure B.10 shows task 2.3 in the collaborative environment. In this task-form the architect is able to directly press the orange run button directly. Pressing this button generates a new CMDOWS file based on the repository of DCs and data model defined during tasks 2.1 and 2.2. The task-form of task 2.3 contains two additional grids: the first stores CMDOWS files for each individual DC and the second stores CMDOWS files for each key design parameter.



2.1. Import CPACSized competences into repository

Agile showcase template joost.schut

Progress: 100%

Fri May 26 2017 09:22:44 GMT+0200 (W. Europe Daylight Time) No start date No due date

Workshop instructions

Task information

In tasks 2.1 to 2.3 the available design competences and parameters of interest need to be mapped to CPACS (Common Parametric Aircraft Configuration Schema). At the end of step to in the AGILE development process, it is expected that a Complete, Compliant and Consistent set of design competences and data model is assembled. The mapping of the inputs and outputs of each design competence is done in the C3PRO (Complete, Compliant and Consistent PROcess) project. The information of the C3PRO project and the AGILE development process are continuously kept in sync.

In this task the following actions need to be performed:

1. The first frame provides an overview of the available design competences provided by each specialist.This frame is used to inspect and set the status of compliance of each design competence to the CPACS data model.
2. In the second frame available CPACS mappings can be added. In case a design competence is already mapped to the CPACS data model an input and output CPACS XML can be added. In order to use these input and output XML files to automatically couple a design competence to CPACS, please select **Include in next update** to **Yes**. Please ensure that every an associated design competence is refered to by selecting the design competence through the selection wizard.
3. After available design competence input/output CPACS XMLs are added, one can press the **Run** button in the top right corner to start the mapping operation. Please note that for design competences with a lot of input/output parameters this can take some time. One can inspect the progress by scroling down to see the C3PRO product model being updated. After completion, one can see a thumps up symbol close to the run button of this task form.
4. The same procedure of reusing CPACS mapping can be done for system input/output parameters

In case a design competence can not be mapped automatically to the CPACS data model, one can proceed to the next task.

Competence

Competence	Owner	Input description	Output description	Status: 1.Available	Status: 2.Compliant
Q3D[FLC]	Ali Elham (TUD)	wing geometry, 2D sections	loads per wing strip in an aeroDataSetForLoads associated with the flightLoadCase	Available	Compliant
Q3D[VDE]	Ali Elham (TUD)	lifting surface for the range of Mach numbers, Reynold's numbers, and angle of attacks specified in the aeroPerformanceMap	The tool then provides the lift, drag and (quarter-chord) moment coefficients of the complete wing.	Available	Compliant
EMWET	Ali Elham (TUD)	Wing geometry external + fuel tank positions	Masses of wing weight and total wing weight	Available	Compliant
HANGAR[AGILE_DC1...	Imco van Gent (TU Delft)	Initial CPACS base file	Full aircraft geometry	Available	Compliant
OBJ	Imco van Gent (TUD)	reference MTOW (toolspecific), MTOW	normalized MTOW	Available	Compliant
SMFA	Imco van Gent (TUD)	Aerodynamic performance main wing (total lift/ drag coefficient) performance targets (range, cruise mach, cruise altitude), maximum take-off mass, cruise density and velocity, tool-specific mission fuel characteristics	Fuel mass	Available	Compliant
CNSTRNT[fuelTankVo...	Imco van Gent (TUD)	wing fuel tank volume, required fuel volume	normalized constraint value	Available	Compliant

Page 1 of 1 Displaying 1 - 18 of 18

1. Add competences to repository

Add a new row to add available competence input and output CPACS XML files. Set include in next update to yes to update the competence input/output mapping to CPACS in the C3PRO project. Don't forget to update the competence status above to **Compliant** after the import is succeeded

Available competence CPACS mapping

Available competen...	Competence	Input CPACS XML	Output CPACS XML	Include in next update	Last C3PRO sync
EMWET	EMWET	EMWET-input.xml	EMWET-output.xml	No	
SMFA	SMFA	SMFA-input.xml	SMFA-output.xml	No	

Page 1 of 1 Displaying 1 - 2 of 2

2. Add key design problem parameters to the repository

Add a new row to add key design problem parameters using CPACS files. Set include in next update to Yes to perform automatic mapping to CPACS in the C3PRO project.

Available Parameter CPACS mapping

Available Parameter..	Parameter	CPACS XML	Include in next update	Last C3PRO sync
Wing root chord	Wing root chord	SCAM-input-root-chord_sfolwY8.xml	No	2017-03-31 17:31
Wing dihedral	Wing dihedral	SCAM-input-dihedral_01vf9v6.xml	No	2017-03-31 17:31
Wing sweeps	Wing sweep	SCAM-input-sweeps_QaOJINt.xml	No	2017-04-02 13:31
Wing taper ratios	Wing taper ratios	SCAM-input-taper-ratios_5Q3zhoO.xml	No	2017-04-02 13:16
Wing span	Wing span	SCAM-input-wing-span_CBBYL0t_IqBP1ed.xml	No	2017-04-02 14:34
MTOW	MTOW	OBJ-output.xml	No	2017-03-31 17:27
Fuel tank volume	Fuel tank volume	CNSTRNT-output-fueltank.xml	No	2017-03-31 17:27
Wing loading	Wing loading	CNSTRNT-output-wingloading.xml	No	2017-03-31 17:27

Page 1 of 1 Displaying 1 - 8 of 8

Inspect the updated CPACS data model in the C3PRO project

Push **Run** to update the CPACS product model and input/output coupling of competences in the C3PRO project. Inspect the CPACS model and competence mappings in the embedded C3PRO project below. One can interact with the C3PRO project through the embedded panel, or by navigating to it using the link button below

One can see the CPACS model by navigating to the data model. The design competences and their input/output coupling can be managed through the Work breakdown menu. The possibility of data flow can be inspected through the model inspector. A log of possible inconsistency in the coupling of design competences to the data model can be viewed through the model debugger

Data model

Name	Default value	Quantity
Agile showcase - C3PRO	-	Exactly 1
Product	-	Exactly 1
Catalog	-	Exactly 1

Agile showcase - C3PRO

Property	Type	Default value	Units
- No properties -			

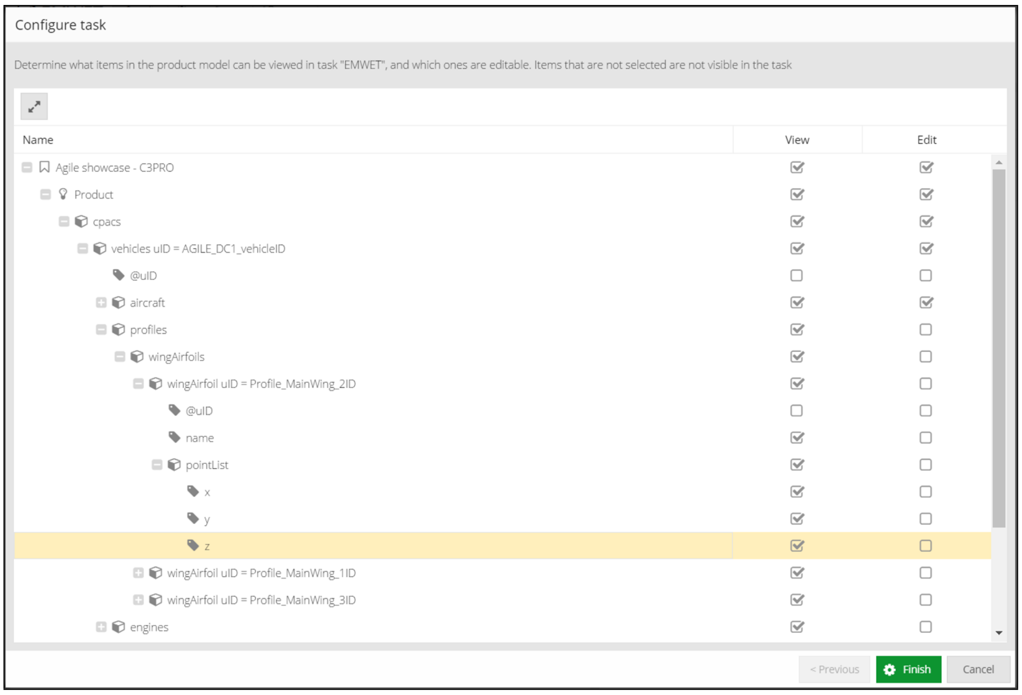
Sub-part models

Product	Exactly 1
Catalog	Exactly 1

Proceed to step 2.2

In case a design competence can not be mapped automatically to the CPACS based repository, one can proceed to the next task to manually add inputs and outputs.

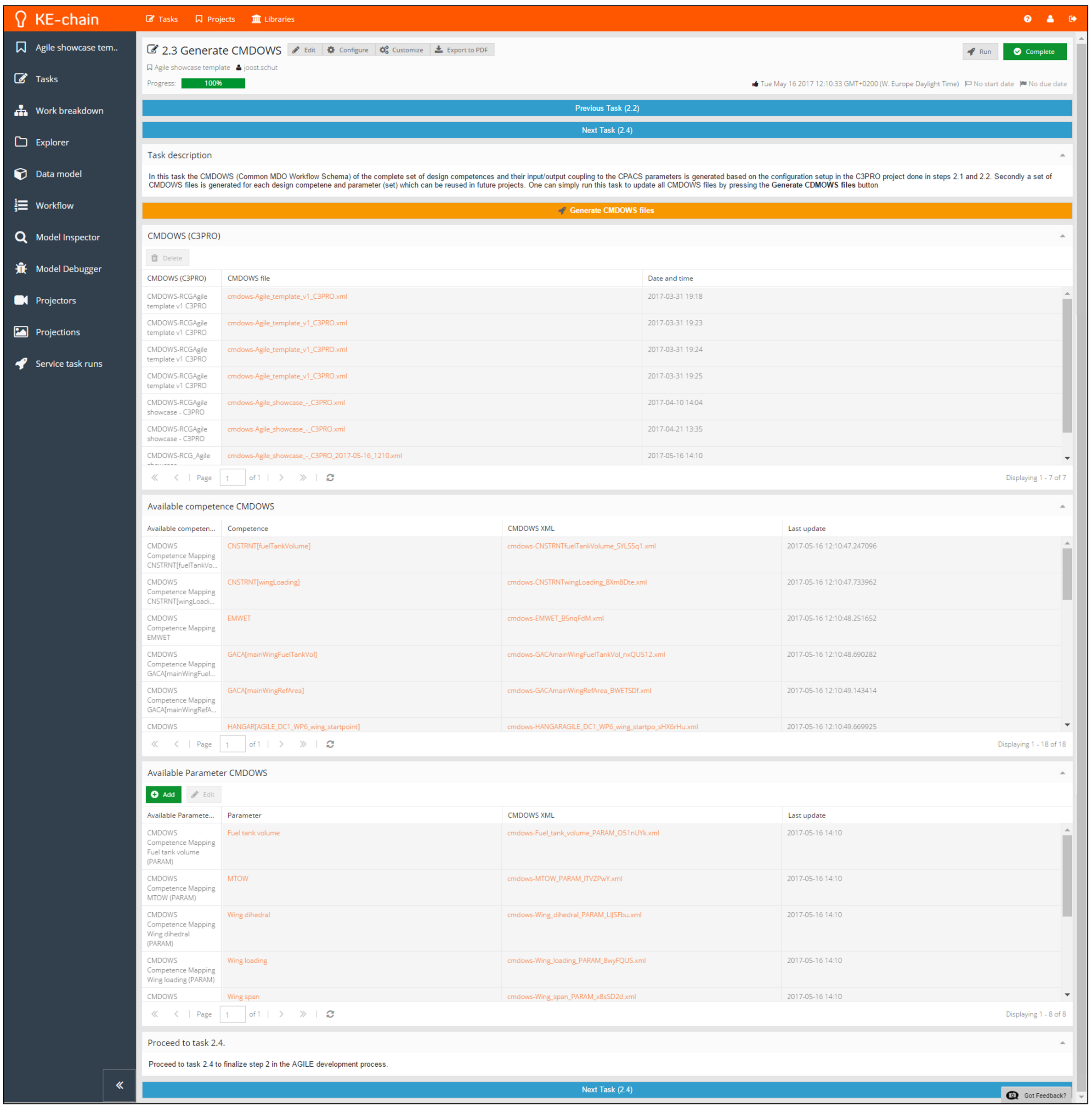
Figure B.8: Impression of the developed KE-chain task-form of task 2.1: Import CPACSized competences into repository


Configure task

Determine what items in the product model can be viewed in task "EMWET", and which ones are editable. Items that are not selected are not visible in the task

Name	View	Edit
Agile showcase - C3PRO	✓	✓
Product	✓	✓
cpacs	✓	✓
vehicles uID = AGILE_DC1_vehicleID	✓	✓
@uID	☐	☐
aircraft	✓	✓
profiles	✓	☐
wingAirfoils	✓	☐
wingAirfoil uID = Profile_MainWing_2ID	✓	☐
@uID	☐	☐
name	✓	☐
pointList	✓	☐
x	✓	☐
y	✓	☐
z	✓	☐
wingAirfoil uID = Profile_MainWing_1ID	✓	☐
wingAirfoil uID = Profile_MainWing_3ID	✓	☐
engines	✓	☐

Figure B.9: Manual mapping of inputs and outputs through the task configuration widget in KE-chain



2.3 Generate CMDOWS

Agile showcase template · joost.schut

Progress: 100%

Tue May 16 2017 12:10:33 GMT+0200 (W. Europe Daylight Time) · No start date · No due date

Task description

In this task the CMDOWS (Common MDO Workflow Schema) of the complete set of design competences and their input/output coupling to the CPACS parameters is generated based on the configuration setup in the C3PRO project done in steps 2.1 and 2.2. Secondly a set of CMDOWS files is generated for each design competene and parameter (set) which can be reused in future projects. One can simply run this task to update all CMDOWS files by pressing the **Generate CDMOWS files** button

CMDOWS (C3PRO)

CMDOWS (C3PRO)	CMDOWS file	Date and time
CMDOWS-RCGAgile template v1 C3PRO	cmdows-Agile_template_v1_C3PRO.xml	2017-03-31 19:18
CMDOWS-RCGAgile template v1 C3PRO	cmdows-Agile_template_v1_C3PRO.xml	2017-03-31 19:23
CMDOWS-RCGAgile template v1 C3PRO	cmdows-Agile_template_v1_C3PRO.xml	2017-03-31 19:24
CMDOWS-RCGAgile template v1 C3PRO	cmdows-Agile_template_v1_C3PRO.xml	2017-03-31 19:25
CMDOWS-RCGAgile showcase - C3PRO	cmdows-Agile_showcase_-_C3PRO.xml	2017-04-10 14:04
CMDOWS-RCGAgile showcase - C3PRO	cmdows-Agile_showcase_-_C3PRO.xml	2017-04-21 13:35
CMDOWS-RCG_Agile	cmdows-Agile_showcase_-_C3PRO_2017-05-16_1210.xml	2017-05-16 14:10

Page 1 of 1 · Displaying 1 - 7 of 7

Available competence CMDOWS

Available competen...	Competence	CMDOWS XML	Last update
CMDOWS Competence Mapping CNSTRNT[fuelTankVo...	CNSTRNT[fuelTankVolume]	cmdows-CNSTRNTfuelTankVolume_SYLS5q1.xml	2017-05-16 12:10:47.247096
CMDOWS Competence Mapping CNSTRNT[wingLoadi...	CNSTRNT[wingLoading]	cmdows-CNSTRNTwingLoading_8Xm8Dte.xml	2017-05-16 12:10:47.733962
CMDOWS Competence Mapping EMWET	EMWET	cmdows-EMWET_B5nqFdM.xml	2017-05-16 12:10:48.251652
CMDOWS Competence Mapping GACA[mainWingFuel...	GACA[mainWingFuelTankVol]	cmdows-GACAmainWingFuelTankVol_nxQU512.xml	2017-05-16 12:10:48.690282
CMDOWS Competence Mapping GACA[mainWingRefA...	GACA[mainWingRefArea]	cmdows-GACAmainWingRefArea_BWETSDf.xml	2017-05-16 12:10:49.143414
CMDOWS	HANGAR[AGILE_DC1_WP6_wing_startpoint]	cmdows-HANGARAGILE_DC1_WP6_wing_startpo_sHX6rHu.xml	2017-05-16 12:10:49.669925

Page 1 of 1 · Displaying 1 - 18 of 18

Available Parameter CMDOWS

Available Paramete...	Parameter	CMDOWS XML	Last update
CMDOWS Competence Mapping Fuel tank volume (PARAM)	Fuel tank volume	cmdows-Fuel_tank_volume_PARAM_O51nUYk.xml	2017-05-16 14:10
CMDOWS Competence Mapping MTOW (PARAM)	MTOW	cmdows-MTOW_PARAM_ITVZPwY.xml	2017-05-16 14:10
CMDOWS Competence Mapping Wing dihedral (PARAM)	Wing dihedral	cmdows-Wing_dihedral_PARAM_LIJSFbu.xml	2017-05-16 14:10
CMDOWS Competence Mapping Wing loading (PARAM)	Wing loading	cmdows-Wing_loading_PARAM_8wyFQUS.xml	2017-05-16 14:10
CMDOWS	Wing span	cmdows-Wing_span_PARAM_x8sSD2d.xml	2017-05-16 14:10

Page 1 of 1 · Displaying 1 - 8 of 8

Proceed to task 2.4.

Proceed to task 2.4 to finalize step 2 in the AGILE development process.

Figure B.10: Task 2.3: Generate CMDOWS

B.3.2. Step 3: Formulate MDAO architecture

In the third step of the MDO framework development process an MDAO architecture needs to be formulated using KADMOS. Formulation of the MDAO architecture starts with importing the CMDOWS file generated in task 2.3 of the development process. This is done through the import functions exposed in task 3.1. The associated task-form is shown in figure B.11. This task-form first displays a grid containing available CMDOWS files. These available CMDOWS files can be generated in task 2.3 but can also be added manually through the green add button. Below the available CMDOWS grid, the architect is able to select a CMDOWS file from the list. The selected CMDOWS file is imported after pressing the large orange button. The imported CMDOWS file is exported as a DSM, shown in an embedded PDF previewer. Moreover a KADMOS visualization package is generated. All files for visualization can be downloaded from the grid below the PDF previewer. Moreover a list of all imported DCs is displayed such that the architect is able to inspect if the import has been performed correctly.

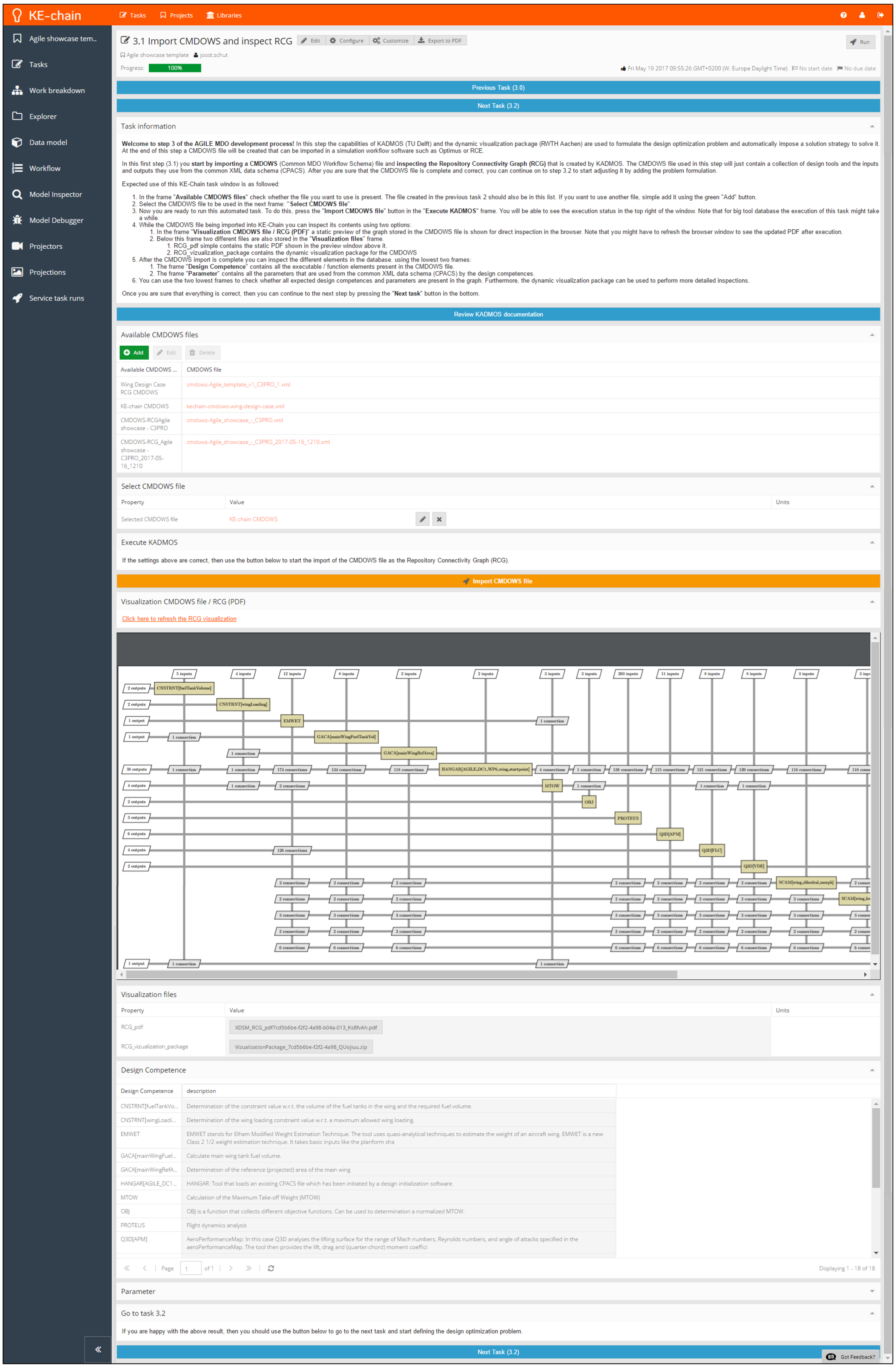
Navigation to task 3.2 directs the architect to the task-form displayed in figure B.12. This task is used to manipulate the DCs to make initial preparations for formulating an MDAO architecture. Manipulation of DCs is done through a series of grids. For example, the exclusion of a DC is done by creating a new record in the associated grid by pressing the green add button. After creating a new excluded DC record, the associated DC needs to be selected through a widget displayed in figure B.13. This selection widget enables the architect to filter through the list of available DCs. The exclusion of the DCs from the FPG can is performed after pressing the large orange button. Underlying functions of KADMOS manipulate the problem graph such that the excluded DC and all its input/output parameters are excluded from the graph. The resulting FPG is exported to a PDF file and KADMOS visualization package. The PDF file is displayed in an embedded PDF previewer in the task-form.

At the bottom of the task-form the architect is able to move to task 3.3 of the development process. Task 3.3 enables the architect to assign parameter roles. In the task-form the architect sees four grids. These four grids contain design variables, objective variables, constraint variables, and state variables. The architect is able to add for example a new design variable by adding a new record to the respective grid. Selecting the add button enables the architect to fill in the form displayed in figure B.15. This form is used to acquire a complete definition of the design variable, such that KADMOS is able to enrich the FPG. The architect is able to link the design variable to one of the parameter available in the FPG through the selection widget displayed in figure B.16. Through this widget the architect is able to filter the usually hundreds of parameters by querying on XPaths or parameter label names.

After the architect has successfully assigned all parameter roles required to solve the design case the FPG can be enriched by KADMOS after pressing the large orange button. The enriched graph is exported as a DSM displayed in an embedded PDF previewer in the task-form. The exported PDF and an exported KADMOS visualization package can be downloaded below the previewer.

In the final task of step 3 in the MDO framework development process the architect is able to apply an MDAO architecture. The associated task-form is shown in figure B.17. The architect is provided with an interface in the top of the task form to select the MDAO architecture, coupling decomposition and a DOE method if applicable. The available MDAO architectures are IDF, MDF, (un)converged MDA and (un)converged DOE. The user is able to select two type of coupling decomposition: Gauss-Seidel and Jacobi. Gauss-Seidel will only remove feedback coupling, while Jacobi will remove both feedforward and feedback coupling between multidisciplinary analyses.

After applying the MDAO architecture settings, the architect is able to execute the underlying KADMOS functions by pressing the large orange button. Executing the underlying KADMOS function manipulates the graph according to the selected architecture. During execution, first a log is generated in which a series of tests are reflected upon. If all tests pass, the XDSM is generated. The exported XDSM is displayed in the embedded PDF previewer at the bottom of the the task-form. Moreover KADMOS generates its visualization package and an CMDOWS file which contains the complete MDAO architecture which can be used in step four of the development process. These CMDOWS files are appended to a list displayed at the bottom of the task-form. The architect is able to download any CMDOWS XML file.



3.1 Import CMDOWS and inspect RCG

Agile showcase template joost.schut

Progress: 100%

Fri May 19 2017 09:55:26 GMT+0200 (W. Europe Daylight Time) No start date No due date

Previous Task (3.0)

Next Task (3.2)

Task information

Welcome to step 3 of the AGILE MDO development process! In this step the capabilities of KADMOS (TU Delft) and the dynamic visualization package (RWTH Aachen) are used to formulate the design optimization problem and automatically impose a solution strategy to solve it. At the end of this step a CMDOWS file will be created that can be imported in a simulation workflow software such as Optimus or RCE.

In this first step (3.1) you **start by importing a CMDOWS** (Common MDO Workflow Schema) file and **inspecting the Repository Connectivity Graph (RCG)** that is created by KADMOS. The CMDOWS file used in this step will just contain a collection of design tools and the inputs and outputs they use from the common XML data schema (CPACS). After you are sure that the CMDOWS file is complete and correct, you can continue on to step 3.2 to start adjusting it by adding the problem formulation.

Expected use of this KE-Chain task window is as followed:

1. In the frame "**Available CMDOWS files**" check whether the file you want to use is present. The file created in the previous task 2 should also be in this list. If you want to use another file, simple add it using the green "Add" button.
2. Select the CMDOWS file to be used in the next frame: "**Select CMDOWS file**".
3. Now you are ready to run this automated task. To do this, press the "**Import CMDOWS file**" button in the "**Execute KADMOS**" frame. You will be able to see the execution status in the top right of the window. Note that for big tool database the execution of this task might take a while.
4. While the CMDOWS file being imported into KE-Chain you can inspect its contents using two options:
    1. In the frame "**Visualization CMDOWS file / RCG (PDF)**" a static preview of the graph stored in the CMDOWS file is shown for direct inspection in the browser. Note that you might have to refresh the browser window to see the updated PDF after execution.
    2. Below this frame two different files are also stored in the "**Visualization files**" frame.
        1. RCG_pdf simple contains the static PDF shown in the preview window above it.
        2. RCG_vizualization_package contains the dynamic visualization package for the CMDOWS
5. After the CMDOWS import is complete you can inspect the different elements in the database. using the lowest two frames:
    1. The frame "**Design Competence**" contains all the executable / function elements present in the CMDOWS file.
    2. The frame "**Parameter**" contains all the parameters that are used from the common XML data schema (CPACS) by the design competences.
6. You can use the two lowest frames to check whether all expected design competences and parameters are present in the graph. Furthermore, the dynamic visualization package can be used to perform more detailed inspections.

Once you are sure that everything is correct, then you can continue to the next step by pressing the "**Next task**" button in the bottom.

Review KADMOS documentation

Available CMDOWS files

Available CMDOWS ...	CMDOWS file
Wing Design Case RCG CMDOWS	cmdows-Agile_template_v1_C3PRO_1.xml
KE-chain CMDOWS	kechain-cmdows-wing-design-case.xml
CMDOWS-RCGAgile showcase - C3PRO	cmdows-Agile_showcase_-_C3PRO.xml
CMDOWS-RCG_Agile showcase - C3PRO_2017-05-16_1210	cmdows-Agile_showcase_-_C3PRO_2017-05-16_1210.xml

Select CMDOWS file

Property	Value	Units
Selected CMDOWS file	KE-chain CMDOWS	

Execute KADMOS

If the settings above are correct, then use the button below to start the import of the CMDOWS file as the Repository Connectivity Graph (RCG).

Import CMDOWS file

Visualization CMDOWS file / RCG (PDF)

Click here to refresh the RCG visualization

Visualization files

Property	Value	Units
RCG_pdf	XDSM_RCG_pdf7cd5b6be-f2f2-4a98-b04a-013_Ks8fvAh.pdf	
RCG_vizualization_package	VizualizationPackage_7cd5b6be-f2f2-4a98_QUojiuu.zip	

Design Competence

Design Competence	description
CNSTRNT[fuelTankVo...	Determination of the constraint value w.r.t. the volume of the fuel tanks in the wing and the required fuel volume.
CNSTRNT[wingLoadi...	Determination of the wing loading constraint value w.r.t. a maximum allowed wing loading.
EMWET	EMWET stands for Elham Modified Weight Estimation Technique. The tool uses quasi-analytical techniques to estimate the weight of an aircraft wing. EMWET is a new Class 2 1/2 weight estimation technique. It takes basic inputs like the planform sha
GACA[mainWingFuel...	Calculate main wing tank fuel volume.
GACA[mainWingRefA...	Determination of the reference (projected) area of the main wing
HANGAR[AGILE_DC1...	HANGAR: Tool that loads an existing CPACS file which has been initiated by a design initialization software.
MTOW	Calculation of the Maximum Take-off Weight (MTOW)
OBJ	OBJ is a function that collects different objective functions. Can be used to determination a normalized MTOW.
PROTEUS	Flight dynamics analysis
Q3D[APM]	AeroPerformanceMap: In this case Q3D analyses the lifting surface for the range of Mach numbers, Reynolds numbers, and angle of attacks specified in the aeroPerformanceMap. The tool then provides the lift, drag and (quarter-chord) moment coeffici

Page 1 of 1 Displaying 1 - 18 of 18

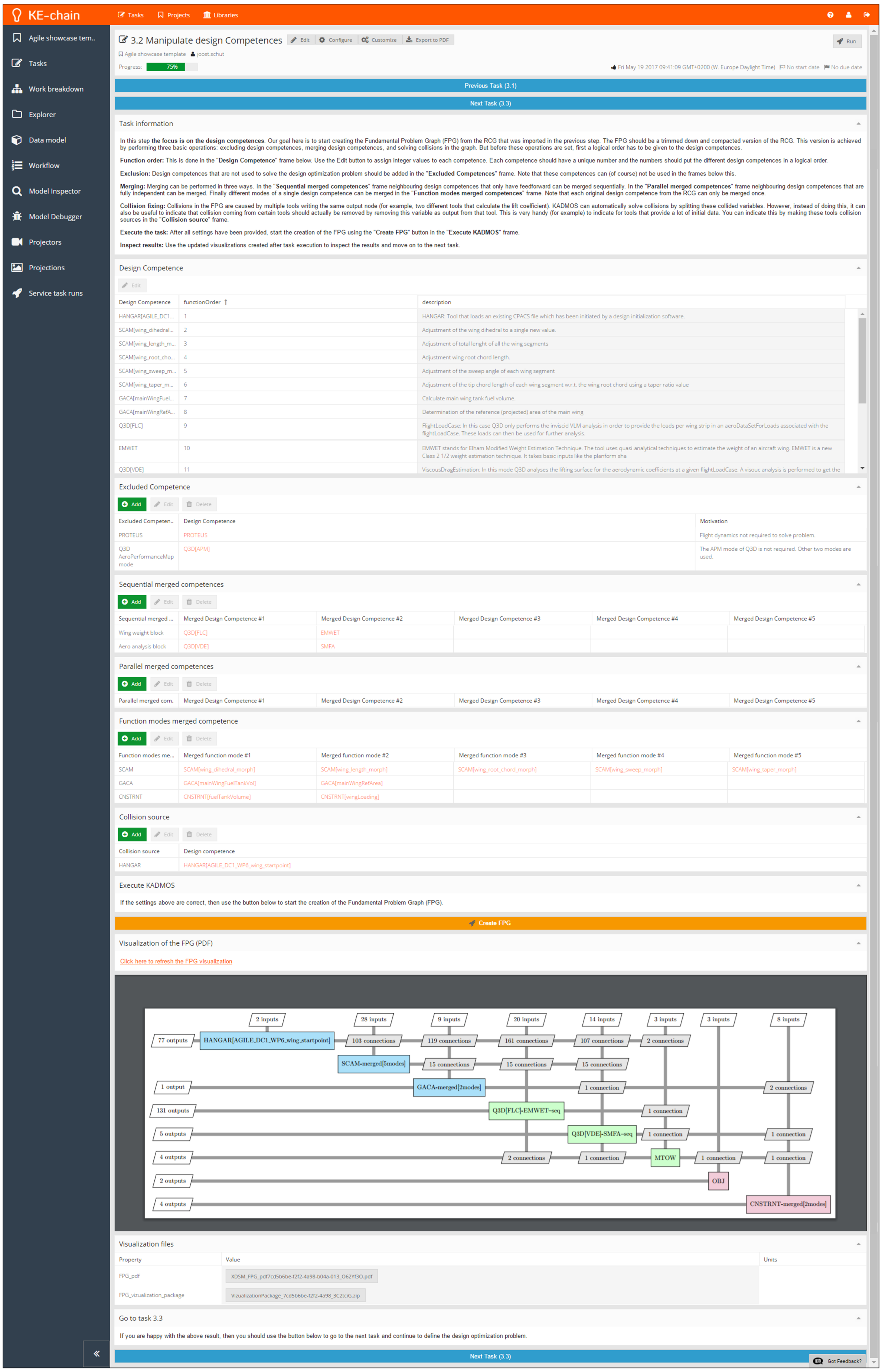
Parameter

Go to task 3.2

If you are happy with the above result, then you should use the button below to go to the next task and start defining the design optimization problem.

Next Task (3.2)

Figure B.11: Impression of the developed KE-chain task-form of task 3.1: Import CMDOWS and inspect RCG



3.2 Manipulate design Competences

Agile showcase template · joost.schut

Progress: 75%

Fri May 19 2017 09:41:09 GMT+0200 (W. Europe Daylight Time) · No start date · No due date

Previous Task (3.1)

Next Task (3.3)

Task information

In this step **the focus is on the design competences**. Our goal here is to start creating the Fundamental Problem Graph (FPG) from the RCG that was imported in the previous step. The FPG should be a trimmed down and compacted version of the RCG. This version is achieved by performing three basic operations: excluding design competences, merging design competences, and solving collisions in the graph. But before these operations are set, first a logical order has to be given to the design competences.

Function order: This is done in the "**Design Competence**" frame below. Use the Edit button to assign integer values to each competence. Each competence should have a unique number and the numbers should put the different design competences in a logical order.

Exclusion: Design competences that are not used to solve the design optimization problem should be added in the "**Excluded Competences**" frame. Note that these competences can (of course) not be used in the frames below this.

Merging: Merging can be performed in three ways. In the "**Sequential merged competences**" frame neighbouring design competences that only have feedforward can be merged sequentially. In the "**Parallel merged competences**" frame neighbouring design competences that are fully independent can be merged. Finally different modes of a single design competence can be merged in the "**Function modes merged competences**" frame. Note that each original design competence from the RCG can only be merged once.

Collision fixing: Collisions in the FPG are caused by multiple tools writing the same output node (for example, two different tools that calculate the lift coefficient). KADMOS can automatically solve collisions by splitting these collided variables. However, instead of doing this, it can also be useful to indicate that collision coming from certain tools should actually be removed by removing this variable as output from that tool. This is very handy (for example) to indicate for tools that provide a lot of initial data. You can indicate this by making these tools collision sources in the "**Collision source**" frame.

Execute the task: After all settings have been provided, start the creation of the FPG using the "**Create FPG**" button in the "**Execute KADMOS**" frame.

Inspect results: Use the updated visualizations created after task execution to inspect the results and move on to the next task.

Design Competence

Design Competence	functionOrder ↑	description
HANGAR[AGILE_DC1...	1	HANGAR: Tool that loads an existing CPACS file which has been initiated by a design initialization software.
SCAM[wing_dihedral...	2	Adjustment of the wing dihedral to a single new value.
SCAM[wing_length_m...	3	Adjustment of total lenght of all the wing segments
SCAM[wing_root_cho...	4	Adjustment wing root chord length.
SCAM[wing_sweep_m...	5	Adjustment of the sweep angle of each wing segment
SCAM[wing_taper_m...	6	Adjustment of the tip chord length of each wing segment w.r.t. the wing root chord using a taper ratio value
GACA[mainWingFuel...	7	Calculate main wing tank fuel volume.
GACA[mainWingRefA...	8	Determination of the reference (projected) area of the main wing
Q3D[FLC]	9	FlightLoadCase: In this case Q3D only performs the inviscid VLM analysis in order to provide the loads per wing strip in an aeroDataSetForLoads associated with the flightLoadCase. These loads can then be used for further analysis.
EMWET	10	EMWET stands for Elham Modified Weight Estimation Technique. The tool uses quasi-analytical techniques to estimate the weight of an aircraft wing. EMWET is a new Class 2 1/2 weight estimation technique. It takes basic inputs like the planform sha
Q3D[VDE]	11	ViscousDragEstimation: In this mode Q3D analyses the lifting surface for the aerodynamic coefficients at a given flightLoadCase. A visouc analysis is performed to get the

Excluded Competence

Excluded Competen..	Design Competence	Motivation
PROTEUS	PROTEUS	Flight dynamics not required to solve problem.
Q3D AeroPerformanceMap mode	Q3D[APM]	The APM mode of Q3D is not required. Other two modes are used.

Sequential merged competences

Sequential merged ...	Merged Design Competence #1	Merged Design Competence #2	Merged Design Competence #3	Merged Design Competence #4	Merged Design Competence #5
Wing weight block	Q3D[FLC]	EMWET			
Aero analysis block	Q3D[VDE]	SMFA			

Parallel merged competences

Parallel merged com.	Merged Design Competence #1	Merged Design Competence #2	Merged Design Competence #3	Merged Design Competence #4	Merged Design Competence #5

Function modes merged competence

Function modes me...	Merged function mode #1	Merged function mode #2	Merged function mode #3	Merged function mode #4	Merged function mode #5
SCAM	SCAM[wing_dihedral_morph]	SCAM[wing_length_morph]	SCAM[wing_root_chord_morph]	SCAM[wing_sweep_morph]	SCAM[wing_taper_morph]
GACA	GACA[mainWingFuelTankVol]	GACA[mainWingRefArea]			
CNSTRNT	CNSTRNT[fuelTankVolume]	CNSTRNT[wingLoading]			

Collision source

Collision source	Design competence
HANGAR	HANGAR[AGILE_DC1_WP6_wing_startpoint]

Execute KADMOS

If the settings above are correct, then use the button below to start the creation of the Fundamental Problem Graph (FPG).

Create FPG

Visualization of the FPG (PDF)

Click here to refresh the FPG visualization

Visualization files

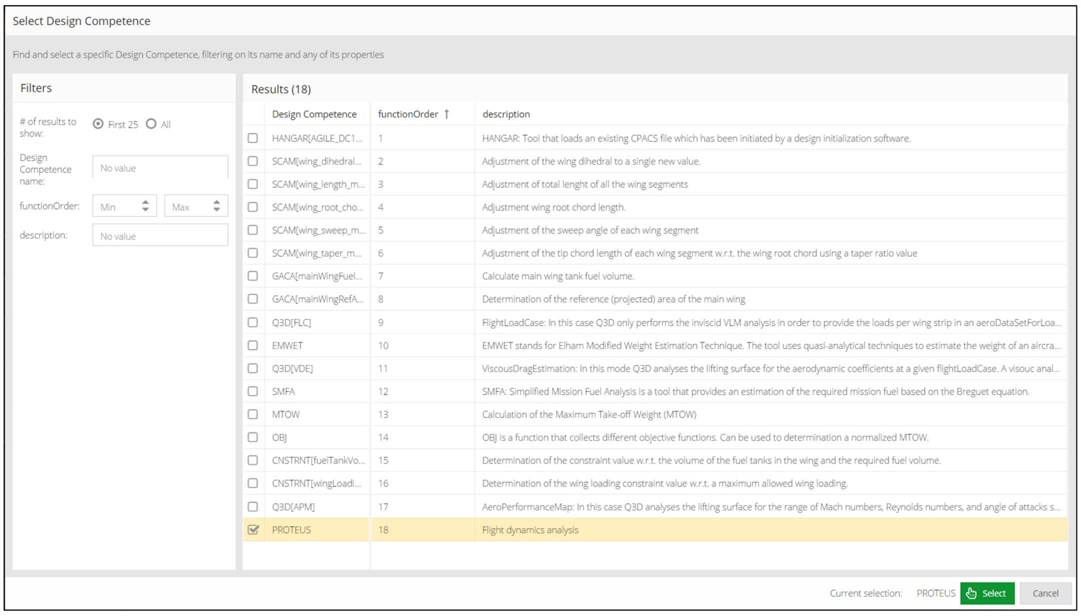
Property	Value	Units
FPG_pdf	XDSM_FPG_pdf7cd5b6be-f2f2-4a98-b04a-013_O62Yf3O.pdf	
FPG_vizualization_package	VizualizationPackage_7cd5b6be-f2f2-4a98_3C2tciG.zip	

Go to task 3.3

If you are happy with the above result, then you should use the button below to go to the next task and continue to define the design optimization problem.

Next Task (3.3)

Figure B.12: Impression of the developed KE-chain task-form of task 3.2: Manipulate design competences



Select Design Competence

Find and select a specific Design Competence, filtering on its name and any of its properties

Filters

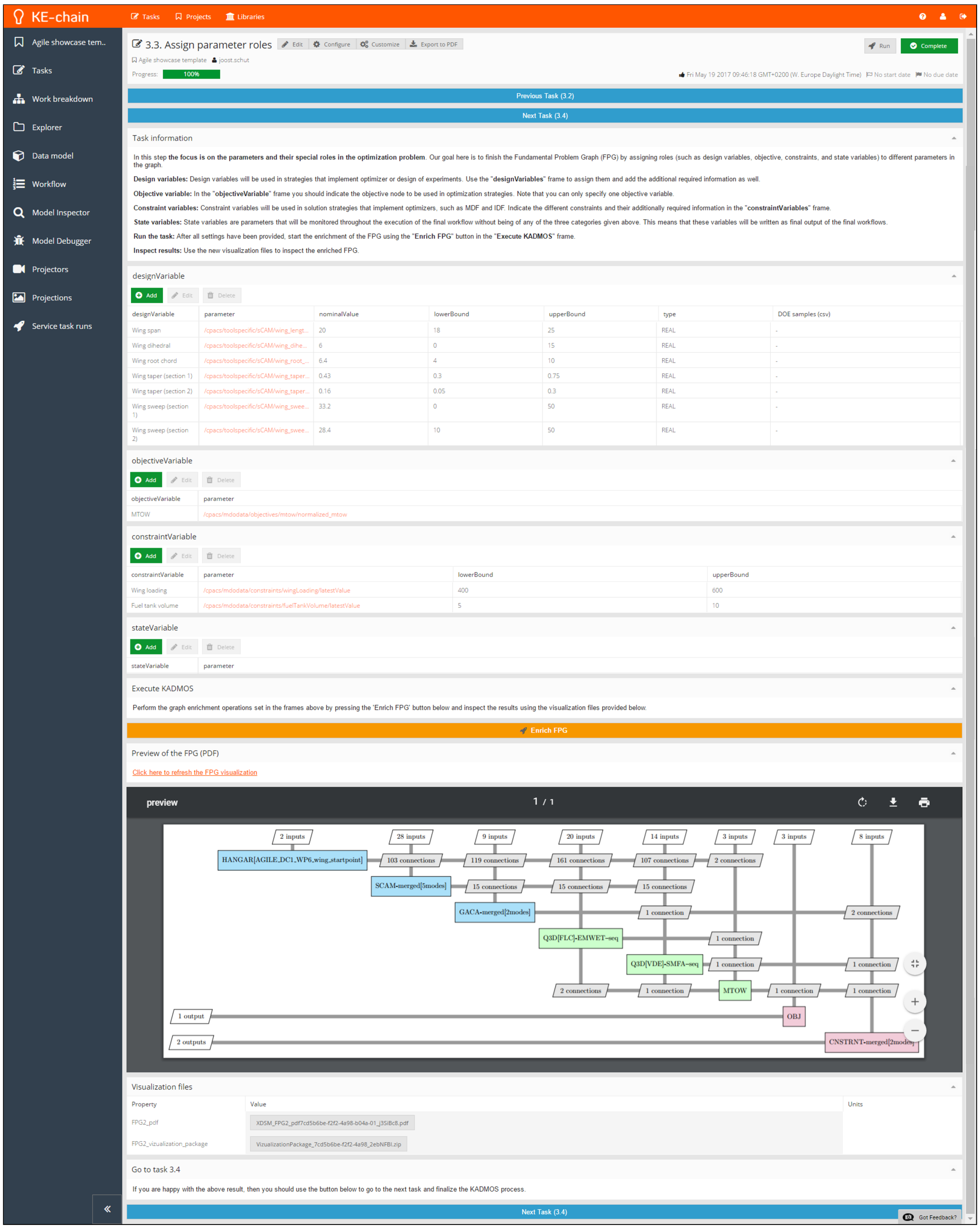
# of results to show:	[✓ First 25] All
Design Competence name:	No value
functionOrder:	Min / Max
description:	No value

Results (18)

	Design Competence	functionOrder ↑	description
☐	HANGAR[AGILE_DC1...	1	HANGAR: Tool that loads an existing CPACS file which has been initiated by a design initialization software.
☐	SCAM[wing_dihedral...	2	Adjustment of the wing dihedral to a single new value.
☐	SCAM[wing_length_m...	3	Adjustment of total lenght of all the wing segments
☐	SCAM[wing_root_cho...	4	Adjustment wing root chord length.
☐	SCAM[wing_sweep_m...	5	Adjustment of the sweep angle of each wing segment
☐	SCAM[wing_taper_m...	6	Adjustment of the tip chord length of each wing segment w.r.t. the wing root chord using a taper ratio value
☐	GACA[mainWingFuel...	7	Calculate main wing tank fuel volume.
☐	GACA[mainWingRefA...	8	Determination of the reference (projected) area of the main wing
☐	Q3D[FLC]	9	FlightLoadCase: In this case Q3D only performs the inviscid VLM analysis in order to provide the loads per wing strip in an aeroDataSetForLoa...
☐	EMWET	10	EMWET stands for Elham Modified Weight Estimation Technique. The tool uses quasi-analytical techniques to estimate the weight of an aircra...
☐	Q3D[VDE]	11	ViscousDragEstimation: In this mode Q3D analyses the lifting surface for the aerodynamic coefficients at a given flightLoadCase. A visouc anal...
☐	SMFA	12	SMFA: Simplified Mission Fuel Analysis is a tool that provides an estimation of the required mission fuel based on the Breguet equation.
☐	MTOW	13	Calculation of the Maximum Take-off Weight (MTOW)
☐	OBJ	14	OBJ is a function that collects different objective functions. Can be used to determination a normalized MTOW.
☐	CNSTRNT[fuelTankVo...	15	Determination of the constraint value w.r.t. the volume of the fuel tanks in the wing and the required fuel volume.
☐	CNSTRNT[wingLoadi...	16	Determination of the wing loading constraint value w.r.t. a maximum allowed wing loading.
☐	Q3D[APM]	17	AeroPerformanceMap: In this case Q3D analyses the lifting surface for the range of Mach numbers, Reynolds numbers, and angle of attacks s...
✓	PROTEUS	18	Flight dynamics analysis

Current selection: PROTEUS

Figure B.13: Select design competence widget



3.3. Assign parameter roles

Agile showcase template joost.schut

Progress: 100%

Fri May 19 2017 09:46:18 GMT+0200 (W. Europe Daylight Time) No start date No due date

Task information

In this step **the focus is on the parameters and their special roles in the optimization problem**. Our goal here is to finish the Fundamental Problem Graph (FPG) by assigning roles (such as design variables, objective, constraints, and state variables) to different parameters in the graph.

Design variables: Design variables will be used in strategies that implement optimizer or design of experiments. Use the "**designVariables**" frame to assign them and add the additional required information as well.

Objective variable: In the "**objectiveVariable**" frame you should indicate the objective node to be used in optimization strategies. Note that you can only specify one objective variable.

Constraint variables: Constraint variables will be used in solution strategies that implement optimizers, such as MDF and IDF. Indicate the different constraints and their additionally required information in the "**constraintVariables**" frame.

State variables: State variables are parameters that will be monitored throughout the execution of the final workflow without being of any of the three categories given above. This means that these variables will be written as final output of the final workflows.

Run the task: After all settings have been provided, start the enrichment of the FPG using the "**Enrich FPG**" button in the "**Execute KADMOS**" frame.

Inspect results: Use the new visualization files to inspect the enriched FPG.

designVariable

designVariable	parameter	nominalValue	lowerBound	upperBound	type	DOE samples (csv)
Wing span	/cpacs/toolspecific/sCAM/wing_lengt...	20	18	25	REAL	-
Wing dihedral	/cpacs/toolspecific/sCAM/wing_dihe...	6	0	15	REAL	-
Wing root chord	/cpacs/toolspecific/sCAM/wing_root_...	6.4	4	10	REAL	-
Wing taper (section 1)	/cpacs/toolspecific/sCAM/wing_taper...	0.43	0.3	0.75	REAL	-
Wing taper (section 2)	/cpacs/toolspecific/sCAM/wing_taper...	0.16	0.05	0.3	REAL	-
Wing sweep (section 1)	/cpacs/toolspecific/sCAM/wing_swee...	33.2	0	50	REAL	-
Wing sweep (section 2)	/cpacs/toolspecific/sCAM/wing_swee...	28.4	10	50	REAL	-

objectiveVariable

objectiveVariable	parameter
MTOW	/cpacs/mdodata/objectives/mtow/normalized_mtow

constraintVariable

constraintVariable	parameter	lowerBound	upperBound
Wing loading	/cpacs/mdodata/constraints/wingLoading/latestValue	400	600
Fuel tank volume	/cpacs/mdodata/constraints/fuelTankVolume/latestValue	5	10

stateVariable

stateVariable	parameter

Execute KADMOS

Perform the graph enrichment operations set in the frames above by pressing the 'Enrich FPG' button below and inspect the results using the visualization files provided below.

Preview of the FPG (PDF)

Click here to refresh the FPG visualization

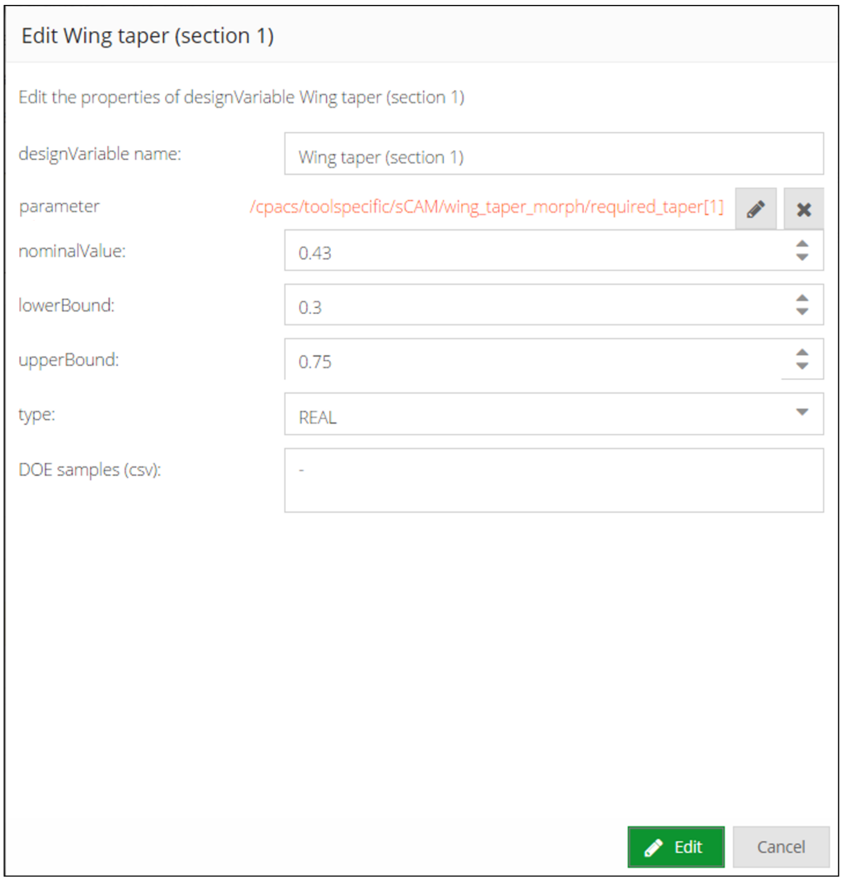
Visualization files

Property	Value	Units
FPG2_pdf	XDSM_FPG2_pdf7cd5b6be-f2f2-4a98-b04a-01_j3SiBc8.pdf	
FPG2_vizualization_package	VizualizationPackage_7cd5b6be-f2f2-4a98_2ebNFBl.zip	

Go to task 3.4

If you are happy with the above result, then you should use the button below to go to the next task and finalize the KADMOS process.

Figure B.14: Impression of the developed KE-chain task-form of task 3.3: Assign parameter roles

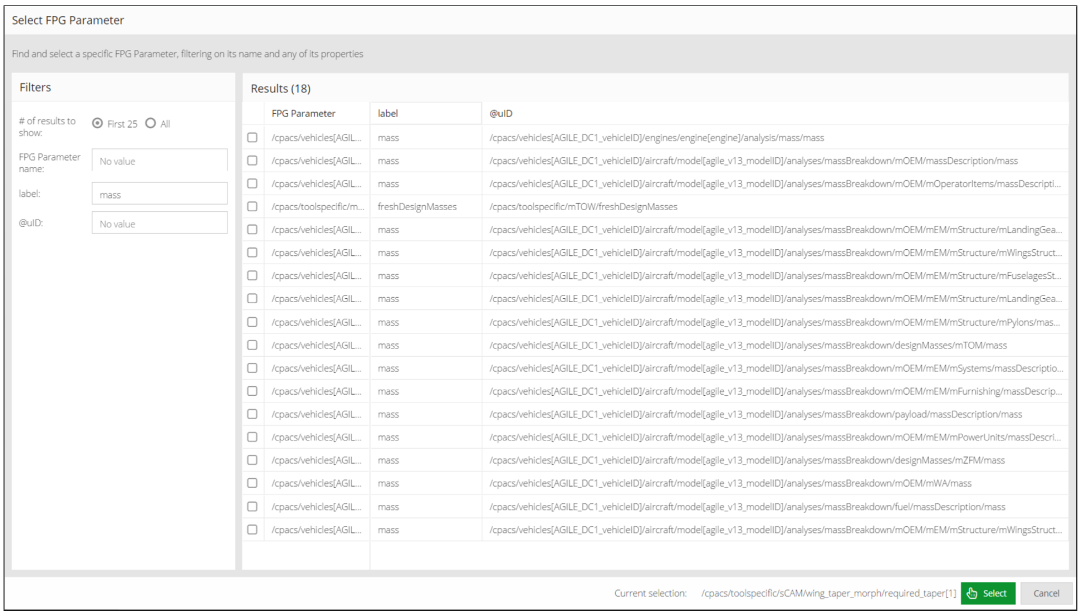


Edit Wing taper (section 1)

Edit the properties of designVariable Wing taper (section 1)

designVariable name:	Wing taper (section 1)
parameter	/cpacs/toolspecific/sCAM/wing_taper_morph/required_taper[1]
nominalValue:	0.43
lowerBound:	0.3
upperBound:	0.75
type:	REAL
DOE samples (csv):	-

Figure B.15: Edit design variable



Select FPG Parameter

Find and select a specific FPG Parameter, filtering on its name and any of its properties

Filters

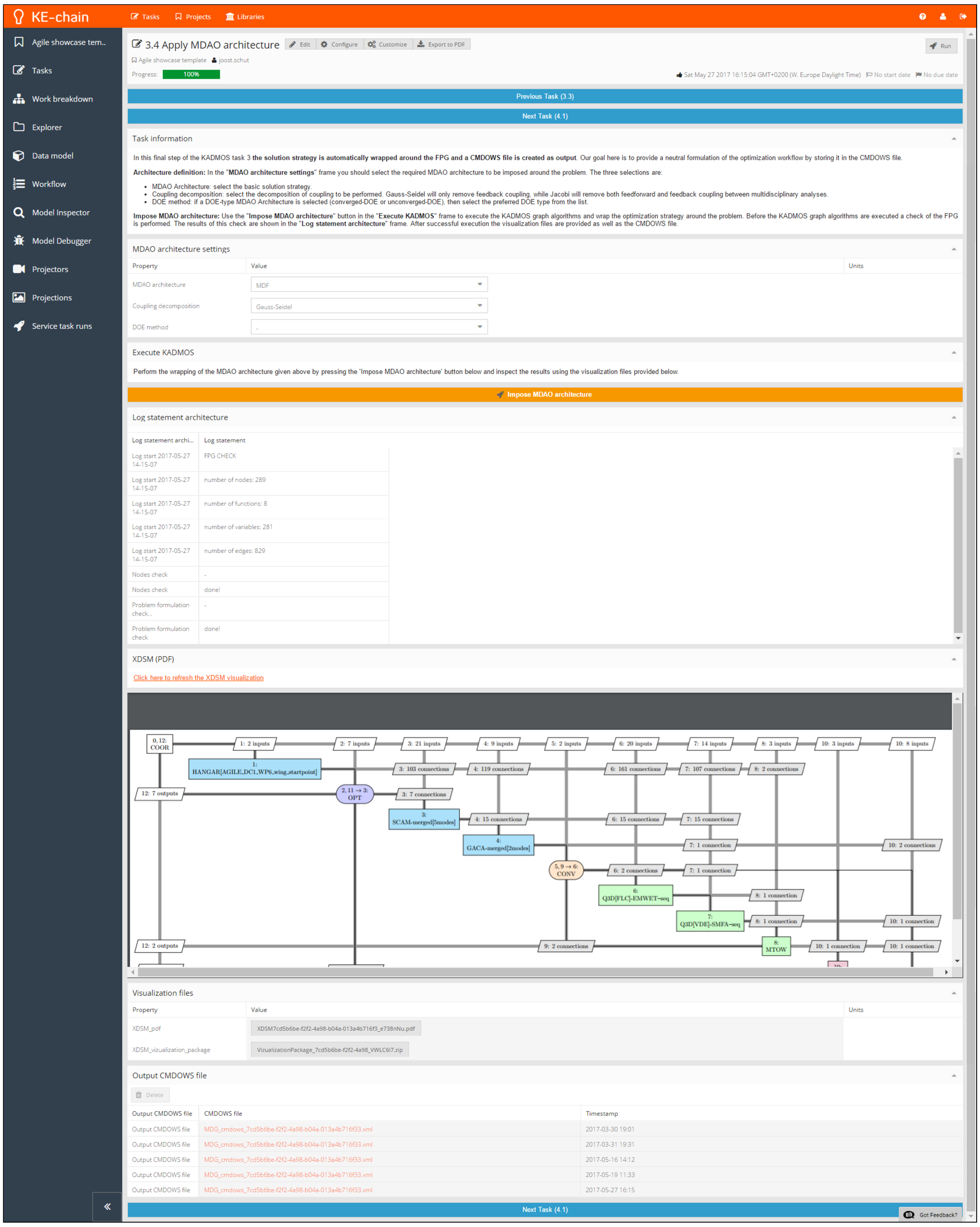
# of results to show:	[✓ First 25] All
FPG Parameter name:	No value
label:	mass
@uID:	No value

Results (18)

	FPG Parameter	label	@uID
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/engines/engine[engine]/analysis/mass/mass
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/mOEM/massDescription/mass
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/mOEM/mOperatorItems/massDescripti...
☐	/cpacs/toolspecific/m...	freshDesignMasses	/cpacs/toolspecific/mTOW/freshDesignMasses
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/mOEM/mEM/mStructure/mLandingGea...
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/mOEM/mEM/mStructure/mWingsStruct...
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/mOEM/mEM/mStructure/mFuselagesSt...
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/mOEM/mEM/mStructure/mLandingGea...
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/mOEM/mEM/mStructure/mPylons/mas...
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/designMasses/mTOM/mass
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/mOEM/mEM/mSystems/massDescriptio...
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/mOEM/mEM/mFurnishing/massDescrip...
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/payload/massDescription/mass
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/mOEM/mEM/mPowerUnits/massDescri...
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/designMasses/mZFM/mass
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/mOEM/mWA/mass
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/fuel/massDescription/mass
☐	/cpacs/vehicles[AGIL...	mass	/cpacs/vehicles[AGILE_DC1_vehicleID]/aircraft/model[agile_v13_modelID]/analyses/massBreakdown/mOEM/mEM/mStructure/mWingsStruct...

Current selection: /cpacs/toolspecific/sCAM/wing_taper_morph/required_taper[1]

Figure B.16: Select FPG parameter widget



3.4 Apply MDAO architecture

Agile showcase template joost.schut

Progress: 100%

Sat May 27 2017 16:15:04 GMT+0200 (W. Europe Daylight Time) No start date No due date

Task information

In this final step of the KADMOS task 3 **the solution strategy is automatically wrapped around the FPG and a CMDOWS file is created as output**. Our goal here is to provide a neutral formulation of the optimization workflow by storing it in the CMDOWS file.

Architecture definition: In the "**MDAO architecture settings**" frame you should select the required MDAO architecture to be imposed around the problem. The three selections are:

- MDAO Architecture: select the basic solution strategy.
- Coupling decomposition: select the decomposition of coupling to be performed. Gauss-Seidel will only remove feedback coupling, while Jacobi will remove both feedforward and feedback coupling between multidisciplinary analyses.
- DOE method: if a DOE-type MDAO Architecture is selected (converged-DOE or unconverged-DOE), then select the preferred DOE type from the list.

Impose MDAO architecture: Use the "**Impose MDAO architecture**" button in the "**Execute KADMOS**" frame to execute the KADMOS graph algorithms and wrap the optimization strategy around the problem. Before the KADMOS graph algorithms are executed a check of the FPG is performed. The results of this check are shown in the "**Log statement architecture**" frame. After successful execution the visualization files are provided as well as the CMDOWS file.

MDAO architecture settings

Property	Value	Units
MDAO architecture	MDF	
Coupling decomposition	Gauss-Seidel	
DOE method	-	

Execute KADMOS

Perform the wrapping of the MDAO architecture given above by pressing the 'Impose MDAO architecture' button below and inspect the results using the visualization files provided below.

Log statement architecture

Log statement archi...	Log statement
Log start 2017-05-27 14-15-07	FPG CHECK
Log start 2017-05-27 14-15-07	number of nodes: 289
Log start 2017-05-27 14-15-07	number of functions: 8
Log start 2017-05-27 14-15-07	number of variables: 281
Log start 2017-05-27 14-15-07	number of edges: 829
Nodes check	-
Nodes check	done!
Problem formulation check...	-
Problem formulation check	done!

XDSM (PDF)

Click here to refresh the XDSM visualization

Visualization files

Property	Value	Units
XDSM_pdf	XDSM7cd5b6be-f2f2-4a98-b04a-013a4b716f3_e738nNu.pdf	
XDSM_vizualization_package	VizualizationPackage_7cd5b6be-f2f2-4a98_VWLC6I7.zip	

Output CMDOWS file

Output CMDOWS file	CMDOWS file	Timestamp
Output CMDOWS file	MDG_cmdows_7cd5b6be-f2f2-4a98-b04a-013a4b716f33.xml	2017-03-30 19:01
Output CMDOWS file	MDG_cmdows_7cd5b6be-f2f2-4a98-b04a-013a4b716f33.xml	2017-03-31 19:31
Output CMDOWS file	MDG_cmdows_7cd5b6be-f2f2-4a98-b04a-013a4b716f33.xml	2017-05-16 14:12
Output CMDOWS file	MDG_cmdows_7cd5b6be-f2f2-4a98-b04a-013a4b716f33.xml	2017-05-19 11:33
Output CMDOWS file	MDG_cmdows_7cd5b6be-f2f2-4a98-b04a-013a4b716f33.xml	2017-05-27 16:15

Figure B.17: Impression of the developed KE-chain task-form of task 3.4: Apply MDAO architecture

B.3.3. Step 4: Implement & execute AD workflow

In the current state of the collaborative environment PIDO applications such as RCE or Optimus available with the Agile consortium are not integrated in step four of the development process. Hence, figure B.18 only shows an impression of an implemented AD workflow in RCE. In the current setup of the collaborative environment, the integrator or architect downloads a CMDOWS file containing a formulated MDAO workflow in step 3.4 and manually imports this CMDOWS file in their PIDO application. From the respective PIDO application, the architect or integrator is able to start the execution of the MDAO workflow locally. During workflow execution, a distributed network of DCs is used.

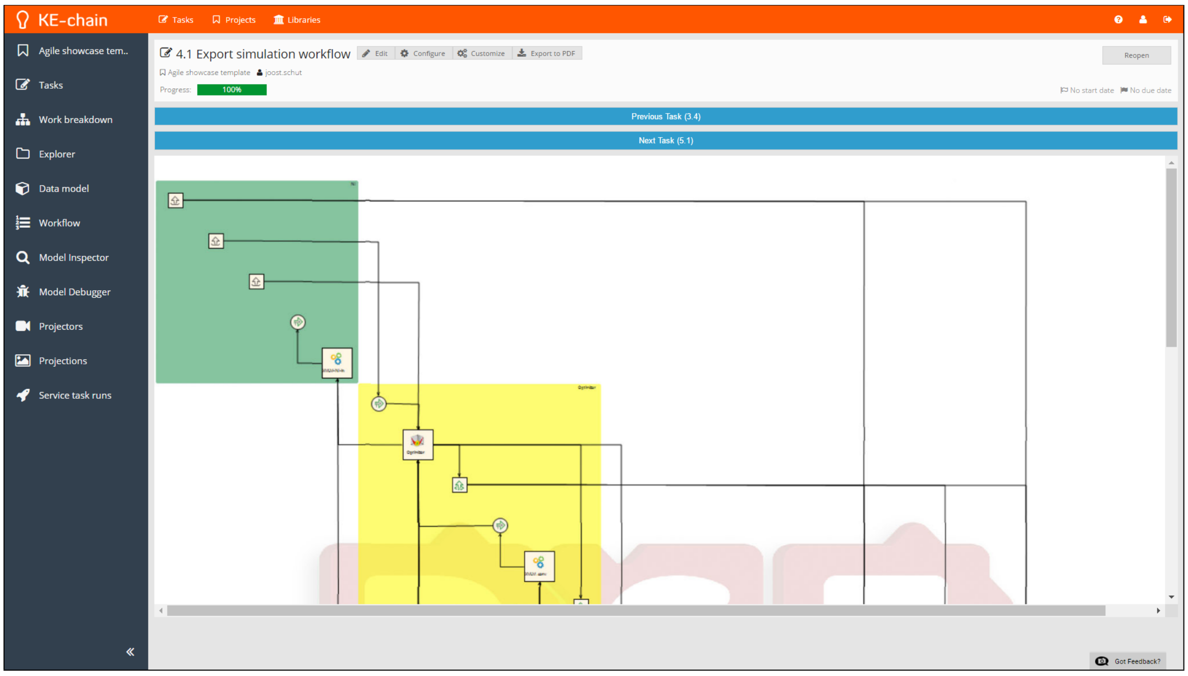


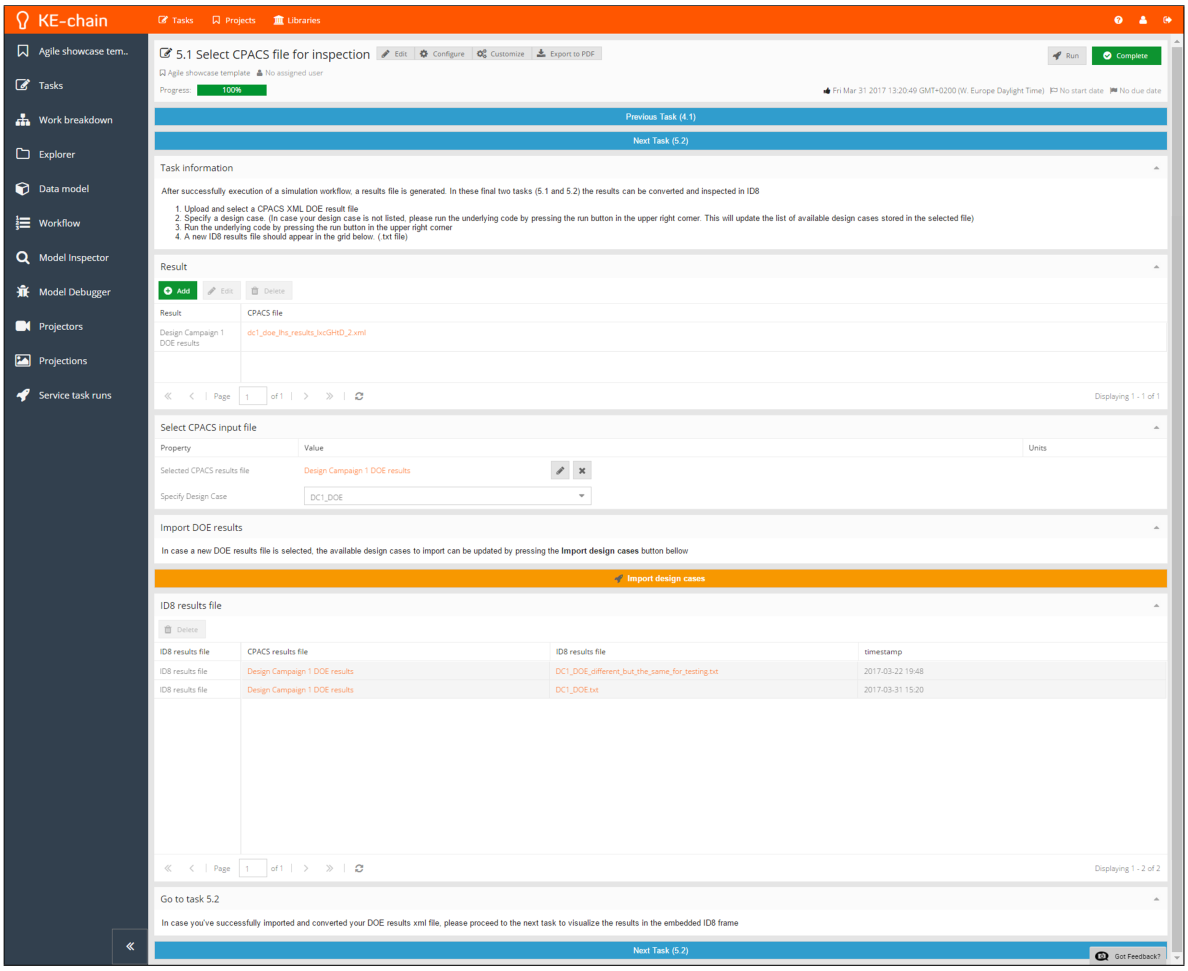
Figure B.18: Impression of the developed KE-chain task-form of task 4.1: Export simulation workflow

B.3.4. Step 5: Inspect design study results

Finally in step five of the MDO framework development process the integrator must prepare the results for visualization in task 5.1 and is able to share the design results with the customer in task 5.2 in the development process.

The task-form associated to task 5.1 is displayed in figure B.19. The first grid displayed in the task form contains a repository of available design study results files produced in step four. In this form the integrator is able to manually upload a CPACS XML results file trough the interface. Below the grid of available results file, the integrator is able to select a specific file he wants to prepare for visualization. After selection of a specific result files, an underlying service can be ran by pressing the large orange button. Execution of the underlying service reads the results XML file, and produces a formatted text file. This text file can be imported in the ID8 visualization environment embedded in the task-form of task 5.2 displayed in figure B.20.

The task-form of task 5.2 is configured such that it can be used directly by the customer. A selection of available formatted design results files, generated in the previous task, are displayed in a grid. The customer is able to download a result file, which can be easily inspected through the embedded ID8 environment provided by Noesis.



5.1 Select CPACS file for inspection

Agile showcase template · No assigned user

Progress: 100%

Fri Mar 31 2017 13:20:49 GMT+0200 (W. Europe Daylight Time) · No start date · No due date

Task information

After successfully execution of a simulation workflow, a results file is generated. In these final two tasks (5.1 and 5.2) the results can be converted and inspected in ID8

1. Upload and select a CPACS XML DOE result file
2. Specify a design case. (In case your design case is not listed, please run the underlying code by pressing the run button in the upper right corner. This will update the list of available design cases stored in the selected file)
3. Run the underlying code by pressing the run button in the upper right corner
4. A new ID8 results file should appear in the grid below. (.txt file)

Result

Result	CPACS file
Design Campaign 1 DOE results	dc1_doe_lhs_results_lxcGHtD_2.xml

Page 1 of 1 · Displaying 1 - 1 of 1

Select CPACS input file

Property	Value	Units
Selected CPACS results file	Design Campaign 1 DOE results	
Specify Design Case	DC1_DOE	

Import DOE results

In case a new DOE results file is selected, the available design cases to import can be updated by pressing the **Import design cases** button bellow

ID8 results file

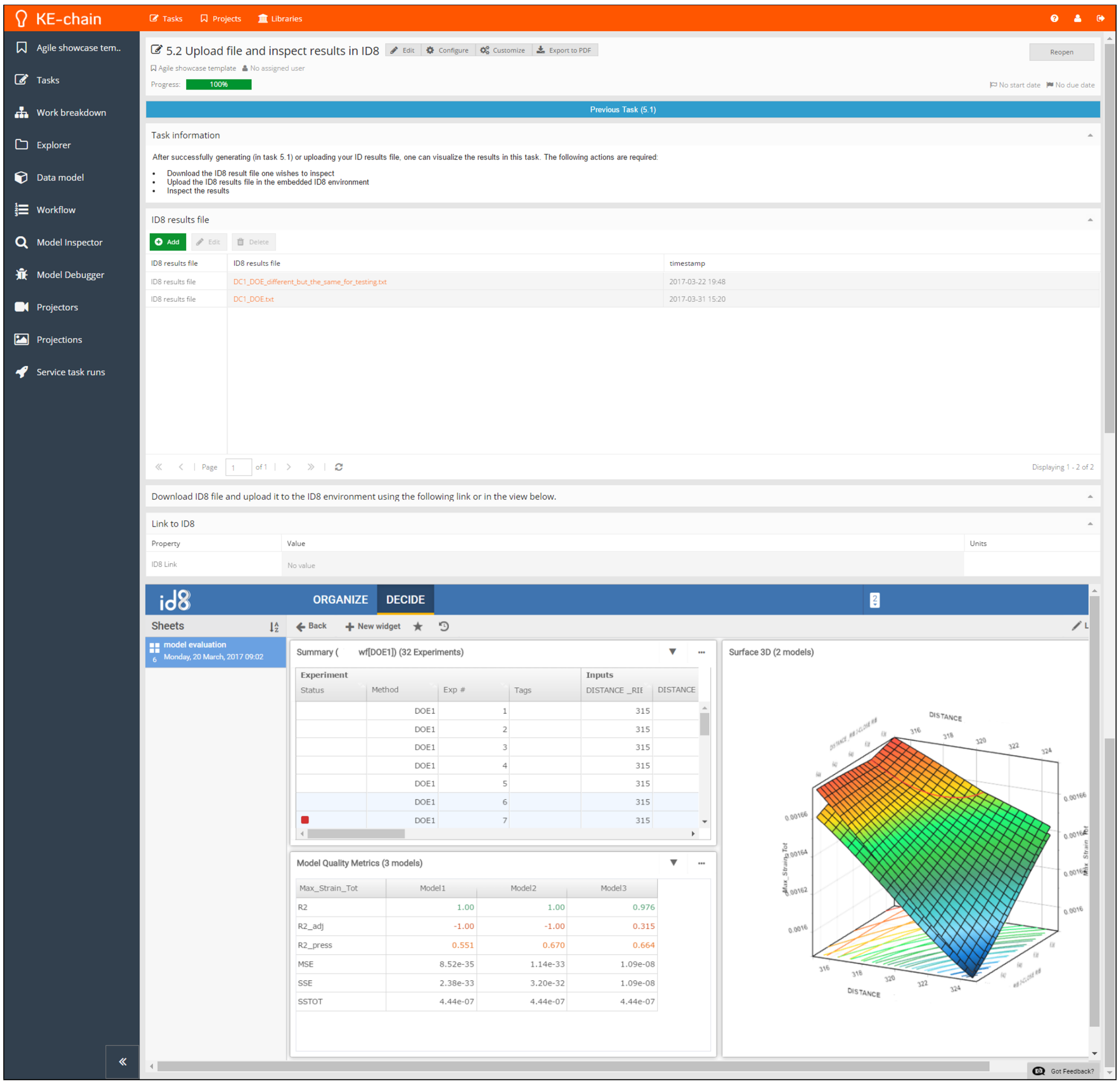
ID8 results file	CPACS results file	ID8 results file	timestamp
ID8 results file	Design Campaign 1 DOE results	DC1_DOE_different_but_the_same_for_testing.txt	2017-03-22 19:48
ID8 results file	Design Campaign 1 DOE results	DC1_DOE.txt	2017-03-31 15:20

Page 1 of 1 · Displaying 1 - 2 of 2

Go to task 5.2

In case you've successfully imported and converted your DOE results xml file, please proceed to the next task to visualize the results in the embedded ID8 frame

Figure B.19: Impression of the developed KE-chain task-form of task 5.1: Select CPACS file for inspection



5.2 Upload file and inspect results in ID8

Agile showcase template   No assigned user

Progress: 100%

No start date   No due date

Task information

After successfully generating (in task 5.1) or uploading your ID results file, one can visualize the results in this task. The following actions are required:

- Download the ID8 result file one wishes to inspect
- Upload the ID8 results file in the embedded ID8 environment
- Inspect the results

ID8 results file

ID8 results file	ID8 results file	timestamp
ID8 results file	DC1_DOE_different_but_the_same_for_testing.txt	2017-03-22 19:48
ID8 results file	DC1_DOE.txt	2017-03-31 15:20

Page 1 of 1   Displaying 1 - 2 of 2

Download ID8 file and upload it to the ID8 environment using the following link or in the view below.

Link to ID8

Property	Value	Units
ID8 Link	No value	

Sheets

model evaluation
Monday, 20 March, 2017 09:02

Summary (wf[DOE1]) (32 Experiments)

Experiment				Inputs	
Status	Method	Exp #	Tags	DISTANCE _RIE	DISTANCE
	DOE1	1		315	
	DOE1	2		315	
	DOE1	3		315	
	DOE1	4		315	
	DOE1	5		315	
	DOE1	6		315	
🟥	DOE1	7		315	

Model Quality Metrics (3 models)

Max_Strain_Tot	Model1	Model2	Model3
R2	1.00	1.00	0.976
R2_adj	-1.00	-1.00	0.315
R2_press	0.551	0.670	0.664
MSE	8.52e-35	1.14e-33	1.09e-08
SSE	2.38e-33	3.20e-32	1.09e-08
SSTOT	4.44e-07	4.44e-07	4.44e-07

Surface 3D (2 models)

Figure B.20: Impression of the developed KE-chain task-form of task 5.2: Upload file and inspect results in ID8

C
Task customization example

In this appendix the rationale behind the task-customization components which can be configured by the configurator. This appendix serves as an extension to section 4.2.1 in the main report. In this section the different object classes which can be configured as widget by the configurator are defined. In listing C.5 one can see an example of underlying JSON code, which solves the purpose of generating a customized taskform with the following components, displayed in figure C.1:

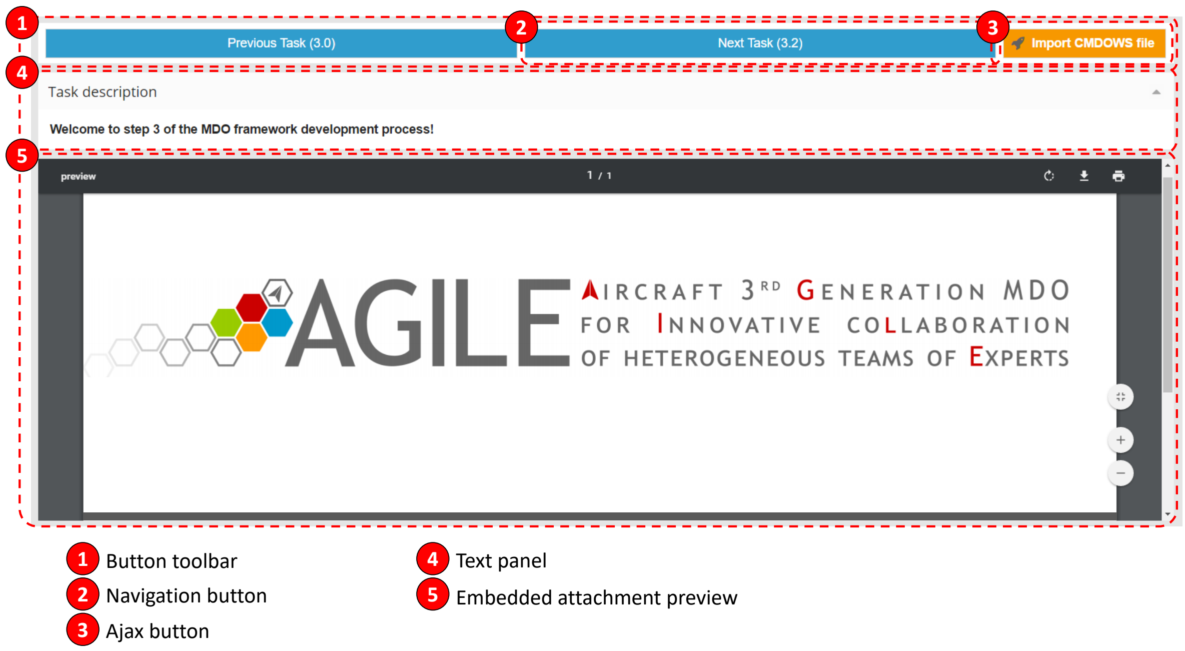
1. **Button toolbar**: Widget containing a list of buttons, which can be embedded in horizontal order.
2. **KE-chain navigation button**: Button which allows for navigation to a single task-form or panel inside KE-chain. For example: navigation to a previous or next task-form.
3. **Ajax button**: Button which enables the performing actions exposed by the KE-chain API, using POST, GET and PUT methods. For example: the execution of an underlying service.
4. **Text panel**: A panel widget can be embedded inside a task-form to display additional information or instructions on the information displayed in that task-form, or the actions a user must perform to complete a task.
5. **Embedded attachment previews**: A custom widget which enables the previewing of attachments uploaded into the KE-chain database. For example the DSMs generated by KADMOS are embedded in the task-forms in step three of the development process to show the intermediate formulation of the MDAO problem setup by the architect. Additional illustrations of embedded previews used in the collaborative environment are shown in appendix B in figures B.11 to B.17.

In the next sections the technical background, and methods the configurator can use to customize a task-form is explained in more detail.

Interactive button widgets

The example task-customization JSON shown in listing C.5 shows two KE-chain navigation buttons, and an AJAX button. Both can be recognized by the *navigatebutton* and *ajaxbutton* xtypes respectively. In this example the navigate buttons link enable the end-users to navigate directly to a task in KE-chain. The navigation button requires mandatory input parameters: *scopeId* and *reference*. The *scopeId* refers to a unique project scope id. The reference refers to a category one wishes to navigate to, for example activities or productmodel etc. In case navigation to a KE-chain activity is required, the *activityId* needs to be provided as well. All IDs can be copied from the URL displayed in the browser. An example of a URL of a certain task is shown in figure C.2.

The AJAX request button, requires the configurator to specify the URL exposing the action. In the example JSON code displayed in C.5 an orange button is integrated in the toolbar, which imposes a service task execution action. An example of the URL of the exposed API of KE-chain used for service task execution is



Welcome to step 3 of the MDO framework development process!

Figure C.1: Example of newly implemented type of task-customization objects in KE-chain

$$\text{https://agile.ke-chain.com/\#scopes/}\underbrace{\textbf{7cd5b6be-f2f2-4a98-b04a-013a4b716f33}}_{\textit{scopeId}}/\underbrace{\textbf{activities}}_{\textit{reference}}/\underbrace{\textbf{11cd28c8-f3e5-4f24-bd9a-163ec06250fd}}_{\textit{activityId}}$$

Figure C.2: Example of a URL used for navigation in KE-chain and its composition

displayed in figure C.3. From this URL, the final part of the URL (*/api/service_tasks/11cd28c8-f3e5-4f24-bd9a-163ec06250fd/execute*) needs to be added to the button configuration. In this URL, an activity ID is included to specify which service task is executed. Moreover a method needs to be specified. Service task execution requires a POST method. Finally a process ID needs to be specified, which can be provided by the KE-chain integrator. The author has developed methods in the KE-chain front-end which are able to read the button configuration, and generate a JavaScript AJAX request function, which is triggered every time the button widget is pressed.

$$\text{https://agile.ke-chain.com/api/service\_tasks/}\underbrace{\textbf{11cd28c8-f3e5-4f24-bd9a-163ec06250fd}}_{\textit{Activity ID}}\text{/execute}$$

Figure C.3: Example of a URL used for service task execution in KE-chain and its composition

The architect is able to edit the style of the button and add a unique text. The text can be styled through HTML formatting[1]. This enables the changing the text font, font-size, color etc. Moreover a style class can be applied on the button according to the ExtJS 6 style configuration methods[2]. Hence, the configurator is able to apply various type of styles to change for example the background color of a button as demonstrated in the displayed example in this chapter. Finally an icon can be attached to a button using the available icons provided by the Font Awesome[3] package. The KE-chain front-end is developed such that it is able to work with all icon classes provided by Font Awesome. In the example of displayed in figure C.1 a rocket icon is added to the AJAX button using the *x-fa fa-rocket* icon.

[1]Example of HTML text styling provided by w3schools.com, `https://www.w3schools.com/html/html_styles.asp` (accessed: 31/05/2017)
[2]Sencha's ExtJS 6 documentation on button styles, `http://docs.sencha.com/extjs/6.2.0/classic/Ext.button.Button.html#cfg-style` (accessed: 31/05/2017)
[3]Font Awesome website, `http://fontawesome.io/icons/` (accessed: 31/05/2017)

All buttons can be inserted in a toolbar, as is demonstrated in this example, or as isolated components. If a configured chooses to insert the buttons in a toolbar, a width or flex might be in the widget configuration. A flex button, automatically adjust the width such that it fills up the remaining toolbar space. A specified width fixes the button width. In case a button is not added in a toolbar, its width will flex over the entire width of the task-form.

Inspectable text panels

The inspectable text panels, indicated by the number 4 in the JSON code displayed in figure C.5 can be given a custom title. Additionally the configurator might want to change the title font or panel style using ExtJS custom classes similarly to the button styles discussed in the previous section. Next, the main text embedded in the panel needs to be specified. This panel widget enables the configurator to write custom text, links and embedded images using HTML formatting. The author used an open-source text-generator [4] to transform custom text to HTML. Using such a text editor, and copying the source code makes writing custom task descriptions a piece of cake.

Live rendered previews

To conclude this appendix on task-customization, live rendered attachment or image previews are discussed. As can be seen form the task-customization JSON in figure C.5 by the number 5 tag, the configurator is able to generate a custom attachment preview component in the task-form. The author uses embedded iframes[5] for this purpose. the configurator is able to change the height of the preview, and the source location of the preview. In case of an attachment preview, this source (indicated in the JSON by *src*) follows the unique URL displayed in figure C.4. This URL is exposed by the API and generates a locally hosted image, of an uploaded attachment to KE-chain. To embed an attachment, for example a PDF preview as displayed in figure C.1, the configurator needs to manually insert the correct property instance ID inside the URL. This property ID can be retrieved directly through KE-chain through the explorer environment as explained in more detail in section B.2.1.

$$\text{https://agile.ke-chain.com/api/properties/}\underbrace{\textbf{b9ff4343-7046-4a6b-9ae3-d33a69ea37c8}}_{\textit{Property instance ID}}\text{/preview}$$

Figure C.4: Example of a URL used for previewing attachments in KE-chain and its composition

Besides a KE-chain attachment preview the source of an iframe component can be direct to many other accessible file locations provided that they are hosted on the internet. For example, the configurator is able to embed external websites or embed interactive D3JS[6] diagrams.

[4]Open source HTML text editor: `http://www.html.am/html-editors/html-text-editor.cfm` (accessed: 31/05/2017)
[5]Detailed explanation of iframes by w3schools.net, `https://www.w3schools.com/tags/tag_iframe.asp` (accessed: 31/05/2017)
[6]D3JS: Data-Driven Documents, `https://d3js.org/` (accessed: 31/05/2017)

```
{
"components":[
    {
1       "xtype": "toolbar",
        "items": [
        {
            "xtype": "navigatebutton",
            "text": "<div style=’color: white; font:normal 14px arial;’>Previous Task (3.0)</div>",
            "style": {"background":"#309dcd"},
            "flex": 1,
            "activityId": "f38627c3-d186-41bc-883f-1c5a8758b722",
            "scopeId": "7cd5b6be-f2f2-4a98-b04a-013a4b716f33",
            "reference": "activities"
        },
        {
2           "xtype": "navigatebutton",
            "text": "<div style=’color: white; font:normal 14px arial;’>Next Task (3.2)</div>",
            "flex": 1,
            "style": {"background":"#309dcd"},
            "activityId": "a51ba25f-2164-49d0-a09b-6859548fa925",
            "scopeId": "7cd5b6be-f2f2-4a98-b04a-013a4b716f33",
            "reference": "activities"
        },
        {
3           "xtype": "ajaxbutton",
            "text": "<div style=’color: white; font:normal 14px arial;’><strong> Import CMDOWS file</strong></div>",
            "iconCls": "x-fa fa-rocket",
            "style": {"background":"#f79800"},
            "link": "/api/service_tasks/11cd28c8-f3e5-4f24-bd9a-163ec06250fd/execute",
            "method": "POST",
            "params": {"process": "dd633db6-dc29-4af0-942e-65198454e681"}
        }]
    },
    {
4       "xtype": "panel",
        "title": "Task description",
        "html": "<div style=’font:normal 14px arial; margin-left:13px;’><p><strong> Welcome to step 3 of the MDO framework development process!</strong></p></div>"
    },
    {
5       "xtype": "component",
        "height": 800,
        "html" :"<html><iframe name=’iframe’ src=\"/api/properties/b9ff4343-7046-4a6b-9ae3-d33a69ea37c8/preview\" width=\"100%\" height=\"800\" frameborder=\"0\" webkitallowfullscreen mozallowfullscreen allowfullscreen></iframe></html>"
    }
]}
```

Figure C.5: Example of a task-customization JSON

D

Detailed overview of integrated tools in the MDO framework used during the workshop sessions

During the workshop session an MDO framework was setup in which an aerodynamic and structural optimization of a wing. The starting point of the wing design case was taken from the optimized aircraft during design campaign 1, as introduced in section 1.3 This reference aircraft is a conventional aircraft with a wing aspect ratio of 9.5. The aircraft is designed to transport 90 passengers with a total payload mass of 9180kg over 3500km. The MDO framework requirements used during the demonstrated MDO framework development process are shown in table D.1. The key design parameters defined for the design case are shown in table D.2. Finally a detailed description of the 18 available DCs is shown in table D.3.

The results of step of the MDO framework development process demonstrated during the workshop sessions is a CPACSized set of DCs, which together form a RCG[43] is shown in a neutral DSM in figure D.1a. An impression of a formulated MDO workflow according to the an MDF architecture and Gauss-sidel coupling decomposition is demonstrated in the neutral XDSM format in figure D.1b. Note that the results of step 3 are generated through the development collaborative environment by the various actors operating the system, using KADMOS's exposed functions.

Table D.1: MDO Use-case requirements defined for the demonstrated workshop session

Requirement	**Description**	**Type**	**Responsible**	**Means of compliance**
Initial data set	The initial data set contains all geometrical parameters and analytic results coming from design campaign 1. For this the CPACS base-file can be used	Concept	Architect	DC1 Base-file imported
Design concepts to include	This design case requires an optimization study performed solely on the wing of the DC1 aircraft. Only the concept of the wing has to be included in the modelling.	Concept	Architect	Wing is imported with all its geometrical properties and analytic properties required to CPACSsize DCs in scope as is.
Competences to include	The wing design study will require the use of structural and aerodynamic analysis tools, such as EMWET and Q3D. This entails a viscous and an inviscid aerodynamic analysis to be performed.	Competence	Architect	All competences are included and described
Objective functions	The objective is to minimize the maximum take-off weight of the full aircraft.	Objective	Architect	Objective functions are CPACSzised in the abstract application framework.
Modelling the wingless aircraft	Since the design problem is focused on the main wing geometry only, the rest of the aircraft is modeled by estimating a realistic weight, lift coefficient and drag coefficient of the wingless aircraft.	Concept	Architect	Estimate realistic weight, lift coefficient and drag coefficient using empirical data.
Design variables	The wing shape should be optimized by varying: root chord, taper ratios (per wing segment), dihedral angle, sweep (per wing segment), and wing span	Objective	Architect	Creation of a tool that can morph the CPACS wing geometry based on the design variables.
Constraints	Two constraints have to be met: all fuel for the required range has to be stored in the main wing fuel tanks and the wing loading of the aircraft cannot be higher than the wing loading of the initial design point.	Objective	Architect	Addition of CPACS-compliant constraint functions to the repository.

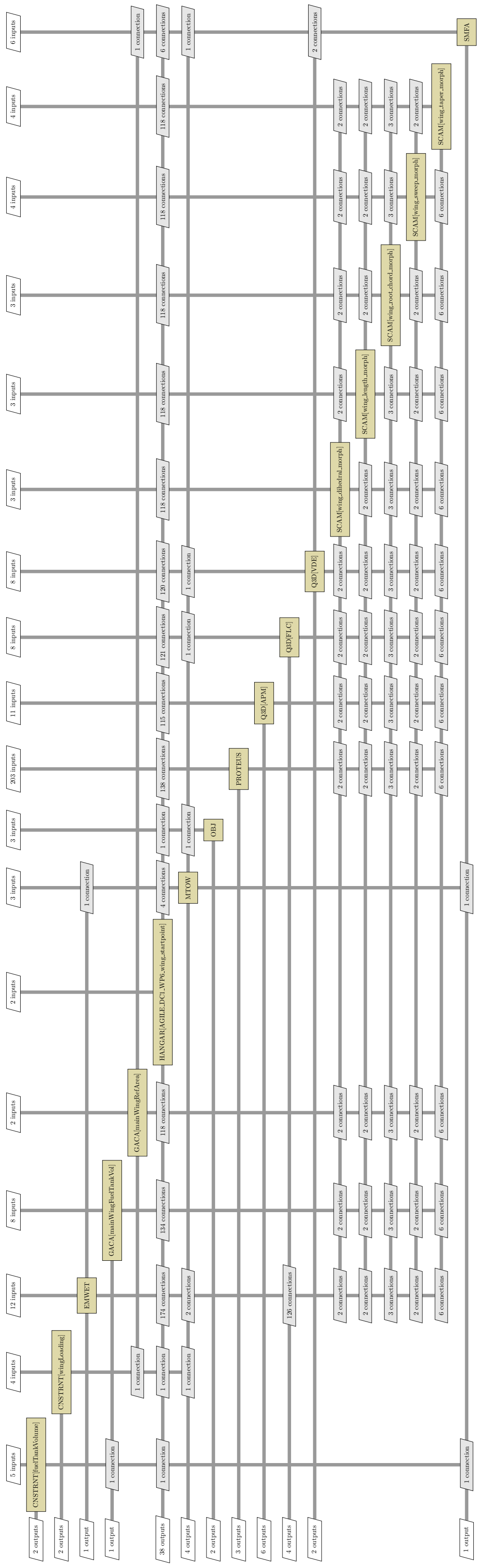
Table D.2: MDO Use-case key design parameters defined for the demonstrated workshop session

Parameter	**Type**	**Description**	**Type of parameter**	**Role in optimization**
MTOW	Design objective	Aircraft maximum take-off weight	Output parameter	Objective
Wing span	Design variable	Wing span	Input parameter	Variable
Wing dihedral	Design variable	Dihedral of the main wing	Input parameter	Variable
Wing taper ratios	Design variable	Wing taper section 1 and 2	Input parameter	Variable
Wing sweep	Design variable	Wing sweep per section	Input parameter	Variable
Wing loading	Design constraint	Wing loading: Lift per square area.	Output parameter	Inequality constraint
Fuel tank volume	Design constraint	Volume of the main wing fuel tank	Output parameter	Inequality constraint
Wing root chord	Design variable	The root chord of the main wing	Input parameter	Variable

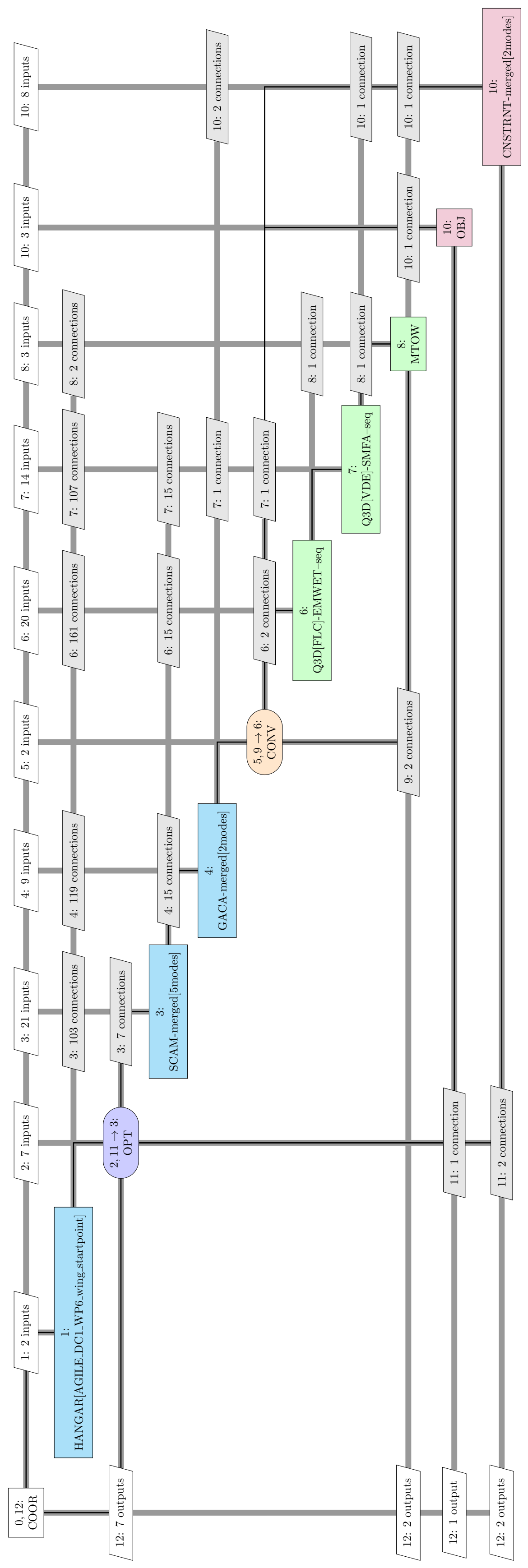
Table D.3: MDO Use-case available Design Competences (DCs) used during the demonstrated workshop session

Competence	Function description	Input description	Output description
Q3D[FLC]	FlightLoadCase: In this case Q3D only performs the inviscid VLM analysis in order to provide the loads per wing strip in an aeroDataSetForLoads associated with the flightLoadCase. These loads can then be used for further analysis.	wing geometry, 2D sections	loads per wing strip in an aeroDataSetForLoads associated with the flightLoadCase
Q3D[VDE]	ViscousDragEstimation: In this mode Q3D analyses the lifting surface for the aerodynamic coefficients at a given flightLoadCase. A visouc analysis is performed to get the right drag value.	lifting surface for the range of Mach numbers, Reynold's numbers, and angle of attacks specified in the aeroPerformanceMap	The tool then provides the lift, drag and (quarter-chord) moment coefficients of the complete wing.
EMWET	EMWET stands for Elham Modified Weight Estimation Technique. The tool uses quasi-analytical techniques to estimate the weight of an aircraft wing. EMWET is a new Class 2 1/2 weight estimation technique. It takes basic inputs like the planform shape, spar positions and airfoils used. Since EMWET will most often be used in the start of the design process, these values can easily be first estimations. A detailed description of EMWET can be found in chapter 3 of the thesis of Dr. Elham: Weight Indexing for Multidisciplinary Design Optimization of Lifting Surfaces (2013).	Wing geometry external + fuel tank positions	Masses of wing weight and total wing weight
HANGAR [AGILE_DC1_WP6_wing_startpoint]	HANGAR: Tool that loads an existing CPACS file which has been initiated by a design initialization software.	Initial CPACS base file	Full aircraft geometry
OBJ	OBJ is a function that collects different objective functions. Can be used to determination a normalized MTOW.	reference MTOW (tool-specific), MTOW	normalized MTOW
SMFA	SMFA: Simplified Mission Fuel Analysis is a tool that provides an estimation of the required mission fuel based on the Breguet equation.	Aerodynamic performance main wing (total lift/ drag coefficient) performance targets (range, cruise mach, cruise altitude), maximum take-off mass, cruise density and velocity, tool-specific mission fuel characteristics	Fuel mass

CNSTRNT [fuelTankVolume]	Determination of the constraint value w.r.t. the volume of the fuel tanks in the wing and the required fuel volume.	wing fuel tank volume, required fuel volume	normalized constraint value
GACA [mainWingFuelTankVol]	Calculate main wing tank fuel volume.	wing geometry including fuel tank description	fuel tank volume
MTOW	Calculation of the Maximum Takeoff Weight (MTOW)	zero-fuel mass, operative empty mass, fuel mass	MTOW
GACA [mainWingRefArea]	Determination of the reference (projected) area of the main wing	wing geometry	wing reference area
CNSTRNT [wingLoading]	Determination of the wing loading constraint value w.r.t. a maximum allowed wing loading.	wing geometry, maximum allowed wing loading	normalized constraint value
PROTEUS	Flight dynamics analysis	control derivatives, aircraft geometry, engine characteristics	flight dynamics characteristics
Q3D[APM]	AeroPerformanceMap: In this case Q3D analyses the lifting surface for the range of Mach numbers, Reynolds numbers, and angle of attacks specified in the aeroPerformanceMap. The tool then provides the lift, drag and (quarter-chord) moment coefficients of the complete wing	Wing parametrization	lift, drag and (quarter-chord) moment coefficients of the complete wing
SCAM [wing_sweep_morph]	Adjustment of the sweep angle of each wing segment	Sweep angle	Morphed wing geometry
SCAM [wing_dihedral_morph]	Adjustment of the wing dihedral to a single new value.	Dihedral angle	Morphed wing geometry
SCAM [wing_taper_morph]	Adjustment of the tip chord length of each wing segment w.r.t. the wing root chord using a taper ratio value	Wing segments, chord lengths	Morphed wing geometry
SCAM [wing_root_chord_morph]	Adjustment wing root chord length.	Root chord, wing parametrization	Morphed wing geometry
SCAM [wing_length_morph]	Adjustment of total lenght of all the wing segments	Wing segments, airfoils	Morphed wing geometry



(a) Wing design study Repository Connectivity Graph (RCG)



(b) Wing design study Extended Design Structure Matrix (XDSM) corresponding to a MDF-GS architecture

Bibliography

[1] J. Agte, O. De Weck, J. Sobieszczanski-Sobieski, P. Arendsen, A. Morris, and M. Spieck. MDO: assessment and direction for advancement—an opinion of one international group. *Structural and Multidisciplinary Optimization*, 40(1-6):17–33, 2010.

[2] A.M. Belay, T. Welo, and P. Helo. Approaching lean product development using system dynamics: investigating front-load effects. *Advances in Manufacturing*, 2(2), 2014.

[3] A. Bendiken. How RDF databases differ from other NoSQL solutions. `http://blog.datagraph.org/2010/04/rdf-nosql-diff`, 2010/04/22. Accessed: 2016-05-17.

[4] J.P.T.J. Berends and M.J.L. van Tooren. Multi-agent task environment framework to support multidisciplinary design and optimization. *Journal of Aerospace Information Systems*, 10(6):258–267, 2013.

[5] D. Böhnke, F. Dorbath, B. Nagel, and V. Gollnick. Multi-fidelity wing mass estimations based on a central model approach. 2012.

[6] F. Bouthillier and K. Shearer. Understanding knowledge management and information management: the need for an empirical perspective. *Information research*, 8(1):8–1, 2002.

[7] T.R. Browning. Applying the design structure matrix to system decomposition and integration problems: A review and new directions. *IEEE Transactions on Engineering Management, VOL. 48, NO. 3, AUGUST*, 2001.

[8] P.D. Ciampa and B. Nagel. Towards the 3^{rd} generation MDO collaborative environment. *30th International Congress of the Aeronautical Sciences*, 2016.

[9] D. Cooper and G. La Rocca. Knowledge-based techniques for developing engineering applications in the 21st century. *7th AIAA ATIO Conference, AIAA, Belfast, Northern Ireland*, 2007.

[10] M. Danilovic and T.R. Browning. Managing complex product development projects with design structure matrices and domain mapping matrices. *International Journal of Project Management 25 (2007) 300–314*, 2006.

[11] J.M. Dorador and R.I.M. Young. Application of IDEF0, IDEF3 and UML methodologies in the creation of information models. *International Journal of Computer Integrated Manufacturing*, 2000.

[12] A. Frost. Information management vs knowledge management. `http://www.knowledge-management-tools.net/IM_vs_KM.html`, 2014. Accessed: 2017-05-14.

[13] OWL Working Group. Web ontology language (OWL). `https://www.w3.org/2001/sw/wiki/OWL`, 2012/12/11. Accessed: 2016-05-17.

[14] B.J. Hicks. Lean information management: Understanding and eliminating waste. *International journal of information management*, 27(4):233–249, 2007.

[15] T. Knothe, R. Jochem, and N. Wintrich. Enforcing front-loading in engineering processes through product-process integration. *R. Poler et al. (eds.), Enterprise Interoperability V: Shaping Enterprise Interoperability in the Future Internet, Proceedings of the I-ESA Conferences 5, DOI 10.1007/978-1-4471-2819-9_7, Springer-Verlag London*, 2012.

[16] G. La Rocca. The challenge of managing knowledge - yet another contribution to the longstanding data-information-knowledge discussion. *TU Delft, Delft University of Technology, Working paper, DOI: 10.13140RG.2.1.4241.8963*, 2016.

[17] G. La Rocca and M.J.L. van Tooren. Development of design and engineering engines to support multidisciplinary design and analysis of aircraft. *Delft Science in Design - A congress on Interdisciplinary Design, Faculty of Architecture, ISBN 90-5269-327-7, Delft, NL*, 2005.

[18] G. La Rocca, T.H.M. Langen, and Y.H.A. Brouwers. The design and engineering engine. towards a modular system for collaborative aircraft design. *28th International Congress of the Aeronautical Sciences*, 2012.

[19] A.B. Lambe and J.R.R.A. Martins. Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes. *Structural and Multidisciplinary Optimization*, 46(2):273–284, 2012.

[20] R. H. Liebeck. Design of the blended wing body subsonic transport. *Journal of aircraft*, 41(1):10–25, 2004.

[21] H.C. Martinez Leon, J.A. Farris, and G. Letens. Improving product development performance through iteration front-loading. *IEEE Transactions on Engineering Management, Vol 50, No. 3, August*, 2013.

[22] N.R. Milton. *Knowledge technologies*, volume 3. Polimetrica sas, 2008.

[23] E. Moerland, R. Becker, and B. Nagel. Collaborative understanding of disciplinary correlations using a low-fidelity physics-based aerospace toolkit. *CEAS Aeronaut J, 6:441-454, DOI 10.1007s13272-015-0153-4*, 2015.

[24] A Mulder. A methodological approach for optimisation of product development processes by application of design automation. 2015.

[25] O. Noran. UML vs IDEF: An ontology-oriented comparative study in view of business modelling. *Conference: 6th International Conference on Enterprise Information Systems (ICEIS), At Porto, Portugal, Volume: 3*, 2004.

[26] B. Nowack. Semantic web technology stack. `http://bnode.org/blog/2009/07/08/the-semantic-web-not-a-piece-of-cake`. Accessed: 2016-06-17.

[27] S. Padula and R. Gillian. Multidisciplinary environments: a history of engineering framework development. In *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 7083, 2006.

[28] D.J. Pate, J. Gray, and B.J. German. A graph theoretic approach to problem formulation for multidisciplinary design analysis and optimization. *Structural Multidisciplinary Optimization 49: 743-760, DOI: 10.1007s00158-013-1006-6*, 2014.

[29] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[30] K.L. Ryan, S.S.G. Ko, and E.W.L. Lee. Business process management (BPM) standards: a survey. *Business Process Management Journal Vol. 15 No. 5*, 2009.

[31] A. Salas and J. Townsend. Framework requirements for MDO application development. In *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, page 4740, 1998.

[32] M. Sandberg. *Knowledge based engineering: in product development*. Luleå tekniska universitet, 2003.

[33] R. Sellar, S. Batill, and J. Renaud. Response surface based, concurrent subspace optimization for multidisciplinary system design. In *34th Aerospace Sciences Meeting and Exhibit*, page 714, 1996.

[34] P. Shiva Prakasha, P. D. Ciampa, and B. Nagel. Collaborative systems driven aircraft configuration design optimization. *30th International Congress of the Aeronautical Sciences*, 2016.

[35] H Smith. College of aeronautics blended wing body development programme. In *Icas 2000 congress*, 2000.

[36] R. Stephens. *Beginning software engineering*. John Wiley & Sons, 2015.

[37] J. Stjepandic, N. Wognum, and W.J.C. Verhagen. *Concurrent Engineering in the 21st Century*. Springer International Publishing Switzerland ISBN 978-3-319-13775-9 DOI 10.1007/978-3-319-13776-6, 2015.

[38] S.S.S. Thomke. Effect of ‘front-loading’ problem-solving on product development performance. *Journal of Product Innovation Management 17(2):128-142*, 2000.

[39] E. Torenbeek. Fundamentals of conceptual design optimization of subsonic transport aircraft. *Delft University of Technology, Department of Aerospace Engineering, Report LR-292*, 1980.

[40] V. Trehan, C. Chapman, and P. Raju. Informal and formal modelling of engineering processes for design automation using knowledge based engineering. *Journal of Zhejiang University-SCIENCE A (Applied Physics & Engineering) ISSN 1673-565X (Print); ISSN 1862-1775 (Online)*, 2015.

[41] I. van Gent, B. Aigner, G. La Rocca, E. Stumpf, and L.L.M. Veldhuis. Using graph-based algorithms and datadriven documents for formulation and visualization of large MDO systems. *Abstract submitted to the 6th CEAS Air and Space Conference*, 2016.

[42] I. van Gent, P.D. Ciampa, J. Schut, G. La Rocca, B. Aigner, and J. Jepsen. Knowledge architecture supporting collaborative MDO in the AGILE paradigm. *Abstract of paper to be submitted to the 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 5-9 June 2017, Denver, Colorado, USA*, 2016.

[43] I van Gent, G La Rocca, and L.L.M. Veldhuis. Composing MDO symphonies: graph-based problem formulation to enable automated execution for large MDO systems, 2016. Abstract of paper to be submitted to the 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 5-9 June 2017, Denver, Colorado, USA.

[44] S. Wakayama and I. Kroo. The challenge and promise of blended-wing-body optimization. In *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, page 4736, 1998.

[45] R.Y. Wang and D.M. Strong. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, 12(4):5–33, 1996.

[46] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology based context modeling and reasoning using owl. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 18–22. IEEE, 2004.

[47] T. Zill, P.D. Ciampa, and B. Nagel. Multidisciplinary design optimization in a collaborative distributed aircraft design system. In *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, page 553, 2012.