Agile MDAO Systems
A Graph-based Methodology to Enhance Collaborative Multidisciplinary Design

van Gent, Imco

DOI
10.4233/uuid:c42b30ba-2ba7-4fff-bf1c-f81f85e890af

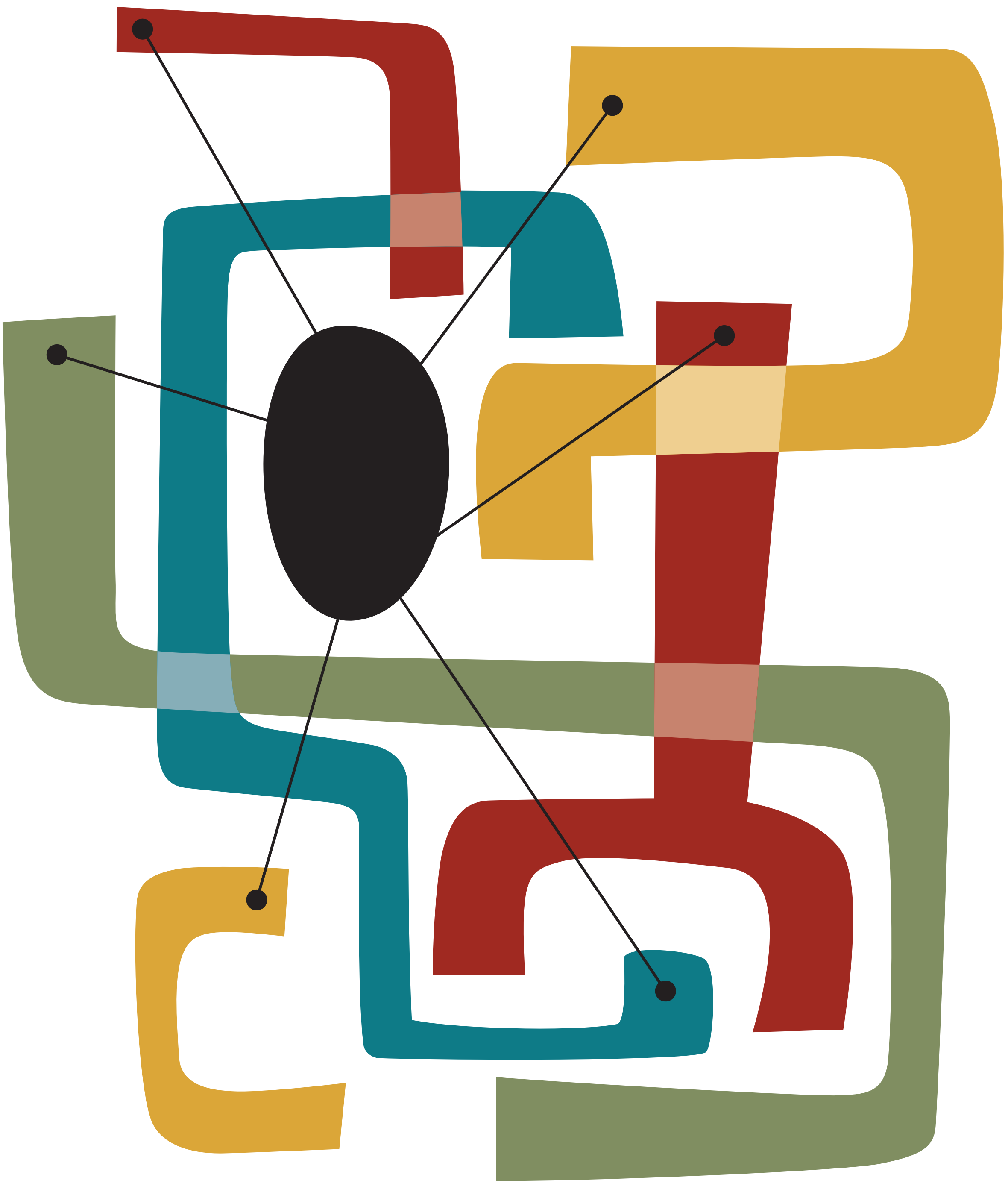
Publication date
2019

Document Version
Final published version

Citation (APA)
van Gent, I. (2019). *Agile MDAO Systems: A Graph-based Methodology to Enhance Collaborative Multidisciplinary Design*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:c42b30ba-2ba7-4fff-bf1c-f81f85e890af

AGILE MDAO SYSTEMS

A GRAPH-BASED METHODOLOGY TO ENHANCE COLLABORATIVE MULTIDISCIPLINARY DESIGN



Imco van Gent

Agile MDAO Systems

A Graph-based Methodology to Enhance Collaborative Multidisciplinary Design

Agile MDAO Systems

A Graph-based Methodology to Enhance Collaborative Multidisciplinary Design

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen
chair of the Board for Doctorates,
to be defended publicly on
Thursday 3 October 2019 at 10:00 o'clock

by

Imco van Gent

Master of Science in Aerospace Engineering,
Delft University of Technology, the Netherlands

born in Rotterdam, the Netherlands

This dissertation has been approved by the promotor.

Composition of the doctoral committee:

Rector Magnificus	chairperson
Prof. dr. ir. L. L. M. Veldhuis	Delft University of Technology, *promotor*
Dr. ir. G. La Rocca	Delft University of Technology, *copromotor*
Independent members:	
Prof. dr. J. Morlier	ISAE-SUPAERO, France
Prof. dr. S. Ricci	Politecnico di Milano, Italy
Prof. dr. P. Krus	Linköping University, Sweden
Dr. ir. L. F. P. Etman	Eindhoven University of Technology
Dr. ir. C. J. Simao Ferreira	Delft University of Technology
Prof. dr. ir. P. Colonna	Delft University of Technology, *reserve member*

This research was funded by the European Union's Horizon 2020 Framework Programme for Research and Innovation through the AGILE project (Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts) and has received funding from the European Union Horizon 2020 Programme (H2020-MG-2014-2015) under grant agreement n° 636202.

Keywords: MDO, MDAO, aircraft design, collaborative design

Printed by: Ipskamp Printing

Graphic design: H-SWAEP illustratie en ontwerp.

ISBN 978-94-6384-067-5

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

voor Nina
doctoranda joie-de-vivrisme

Summary

The need to continuously design better aircraft is challenging on multiple fronts: the existing tube-and-wing aircraft are difficult to improve further, while more creative configurations prove more cumbersome to design due to the lack of empirical knowledge and the (usually) more highly integrated components. Multidisciplinary Design Analysis and Optimization (MDAO) can offer major benefits to tackle these issues. However, MDAO is not yet widely applied in the industrial context due to two major hurdles:

configuration hurdle the set-up of the first executable MDAO workflow takes too long — namely 60-80% of the available project time — in collaborative projects in which a large team of heterogeneous experts needs to operate a large, complex MDAO system;

reconfiguration hurdle once the executable system has been established, it lacks agility and cannot be reconfigured easily to address 'what if scenarios' or answer new questions arising from the insights obtained from previously configured systems. Typical reconfigurations include the removal, addition, or modification of disciplinary tools, adjustment of the MDAO problem to be solved (e.g. new design variables, different objective, additional constraints), or the alternation of the underlying strategy (i.e. design convergence, design of experiments, optimization) that needs to be implemented as an executable workflow.

This dissertation defines the MDAO-based development process to be performed in five stages, where each stage contains a transformed version of the MDAO system. The first three stages occur in the formulation phase and consist of the tool repository, MDAO problem, and MDAO solution strategy. The final two stages reside in the execution phase, containing the executable workflow and the MDAOptimal design. Generally, this development process is iterative and the design team needs to be able to reconfigure the MDAO system regularly as described in the reconfiguration hurdle above, thereby switching from one stage (and one phase) to another continuously. A particularly challenging transformation — called the implementation gap — occurs between the formulation and the execution phases, when a formulated solution strategy needs to be instantiated as an executable workflow.

Based on these observations, this dissertation is motivated by the following research question:

> How can the state-of-the-art methodology for performing MDAO in collaborative projects — involving a large team of heterogeneous experts that needs to operate a large, complex MDAO system — be enhanced (or replaced) by a novel methodological approach that lowers the con- and reconfiguration hurdles encountered nowadays?

To answer this question the goal was set *to develop, implement, and assess a new methodology, specifically addressing the collaborative aspects of performing MDAO, that enables formalized MDAO systems in all stages of the formulation phase, such to lower the abovementioned hurdles.*

The pursuit of this goal resulted in three major developments representing the core of the novel methodology:

1. A graph-based formalization of MDAO systems was developed by expanding earlier work on this subject. This formalization allows MDAO systems to be represented as graphs in the three stages of the formulation phase:

   Tool repository The repository is represented by the 'repository connectivity graph', which describes the collection of disciplinary tools and how they are coupled.

   MDAO problem The 'fundamental problem graph' is used to specify the MDAO problem that should be addressed. This graph is defined by the design team based on the repository graph, and should meet strict graph-theoretical conditions so that the final stage of the design process can be automatically instantiated.

   MDAO solution strategy This stage is represented by two graphs: the MDAO data and the MDAO process graph. Together, these graphs constitute the blueprint of the eventual workflow in the execution phase. The graphs are automatically created by imposing the selected MDAO architecture on the fundamental problem graph.

   The graph-theoretic foundation developed in this work contains new concepts, such as 'instances' and the 'circularity index', to handle particular issues of MDAO systems, and forms the core of the methodological approach. This theoretical development was implemented as an open-source Python package called KADMOS (Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System).
2. Different members of the design team need the KADMOS graph objects for a variety of purposes, such as system visualization, management, and the creation of executable workflows. Therefore, a new standard was proposed to store and exchange the graph-based system formulations in a application-agnostic format. The standard's implementation as XML schema is called CMDOWS (Common MDO Workflow Schema).
3. With the MDAO solution strategy — also called workflow blueprint — formalized as graphs in KADMOS and stored as XML files based on CMDOWS, the opportunity presented itself to bridge the implementation gap by automatically instantiating the executable workflow in different Process Integration and Design Optimization (PIDO) platforms. This capability was developed for the platforms RCE, Optimus, and OpenMDAO (the first two as native plug-ins and the latter as a separate Python package called OpenLEGO).

Using the prototyped software implementations, the novel methodology has been verified and validated based on two MDAO benchmark problems: Sellar problem and NASA's supersonic business jet case. This validation included the complex distributed MDO architectures 'collaborative optimization' and 'BLISS-2000'. Additionally, the comparison of the development process with the state-of-the-art approach indicated a 57% time decrease for system (re)configuration.

The three core components were developed in the context of the next-generation MDAO framework project AGILE. In AGILE, an international group of nineteen partners worked together to establish and test a novel MDAO framework generation: the AGILE development framework. This framework combines a range of technologies and has a multilayered structure called the knowledge architecture. The methodological approach original to this dissertation (and their respective implementations) lie at the heart of the AGILE framework and have been assessed in four collaborative aircraft design projects. In these projects large, heterogeneous, internationally operating teams used the framework to perform the MDAO of four unconventional aircraft configurations: strut-braced wing, box-wing, blended-wing body, and turboprop. Individual framework components were assessed through a questionnaire from which the following conclusions were drawn concerning this dissertation's developments:

- CMDOWS demonstrated its capability to act as a data standard to integrate the heterogeneous collection of applications deployed in the AGILE framework: a user interface to define the MDAO system with KADMOS under the hood, a visualization tool, multiple PIDO platforms, and multiple tool repository databases.
- KADMOS played a central role in the formulation phase of the design projects and its ability to represent the complex MDAO system was highly appreciated. Survey respondents estimated that the use of KADMOS for systems formulation leads to an average time reduction of 49%.
- Automated workflow creation with the RCE and Optimus plug-ins was also considered valuable with an estimated time reduction of 33%, though the created workflows were found to perform less efficiently because of overhead introduced by additional components in the resulting workflows.

The complete AGILE framework — in which the developments original to this thesis played a central role — was estimated to positively impact the MDAO development process with an associated time reduction of 39%.

The graph-based methodological approach was also tested outside the AGILE project context. In this additional study a state-of-art aircraft design toolbox, called the Initiator, was restructured using this dissertation's approach and code base. The approach helped to 'de-spaghettize' a large computational toolbox and resulted in a well-structured, agile framework in which disciplinary components and modules can be easily integrated, connected, and used in a variety of MDAO studies. After the restructured framework was established, the MDAO system was configured and iteratively reconfigured to perform the design of a typical airliner by applying different strategies: design convergence, design of experiments, and optimization. The restructured toolbox demonstrated how the same methodology — originally developed and demonstrated in AGILE — is generic enough to be applied in other frameworks to improve their scalability and flexibility.

Two new concepts were proposed based on this work as an outlook and road map for future developments: composite architectures and MDAO bots. The former addresses the need of designers to define and use their own architectures that can handle the peculiarities of their specific MDAO problem. This need was identified by AGILE users based on the automation potential offered by the combination of KADMOS, CMDOWS, and CMDOWS-based workflow creation. The latter concept — MDAO bots — was introduced to suggest a road map for future research. The bot is defined as an autonomous program on a network which can interact with an MDAO system and the design team members, to behave like a human agent that answers a design question by (re)configuring and executing the MDAO system. The automated approach established in this work is considered to constitute the first MDAO bot. Several bots have been proposed to unburden the design team of repetitive and error-prone tasks that are part of the MDAO development process, in order to further lower the con- and reconfiguration hurdles that originally motivated this work. The actual development of these bots is left to researchers participating in future collaborative MDAO projects.

SAMENVATTING

De behoefte om continu betere vliegtuigen te ontwerpen is op meerdere fronten uitdagend: de bestaande buis-en-vleugelvliegtuigen zijn moeilijk te verbeteren, terwijl creatievere configuraties vaak lastiger te ontwerpen zijn vanwege het gebrek aan empirische kennis en de (vaak) meer geïntegreerde componenten. Multidisciplinaire Ontwerp Analyse en Optimalisatie (MOAO) kan grote voordelen bieden om deze belemmeringen weg te nemen. MOAO wordt echter nog niet breed toegepast in de industriële context vanwege twee grote hordes/hindernissen:

configuratie-horde het opzetten van de eerste uitvoerbare MOAO-workflow duurt te lang — namelijk 60-80% van de beschikbare projecttijd — in projecten in samenwerkingsverband waarbij een groot team bestaand uit heterogene specialisten een groot, complex MOAO-systeem moeten bedienen;

herconfiguratie-horde zodra een uitvoerbaar system is opgezet, mist het behendigheid en kan het niet makkelijk worden hergeconfigureerd om ‘wat als scenario's’ te adresseren of nieuwe vragen te beantwoorden die voortvloeien uit de opgedane inzichten die zijn verkregen uit eerder geconfigureerde systemen. Typische herconfiguraties omvatten de verwijdering, toevoeging, of aanpassing van disciplinaire tools, aanpassing van het MOAO-probleem dat opgelost moet worden (bijv. nieuwe ontwerpvariabelen, ander optimalisatie-objectief, extra beperkingen), of de afwisseling van de onderliggende strategie (bijv. ontwerpconvergentie, ontwerp met experimenten, optimalisatie) die geïmplementeerd moeten worden als een uitvoerbare workflow.

Dit proefschrift definieert het ontwikkelproces gebaseerd op MOAO als zijnde uitgevoerd in vijf stadia, waarbij elk stadium een getransformeerde versie van het MOAO-systeem bevat. De eerste drie stadia vinden plaats in de formuleringsfase en bestaan uit de tool opslag, MOAO-probleem, en MOAO-oplossingsstrategie. De laatste twee stadia zetelen in de uitvoeringsfase, die de uitvoerbare workflow en het MOAOptimale ontwerp bevat. In het algemeen is het ontwikkelproces iteratief en het ontwerpteam moet het MOAO-systeem regelmatig kunnen herconfigureren zoals eerder beschreven in de herconfiguratie-horde, daarbij continu wisselend tussen het ene stadium (en ene fase) en de andere. Een bijzonder uitdagende transformatie — genaamd de implementatiekloof — doet zich voor tussen de formulerings- en uitvoeringsfases, wanneer een geformuleerde oplossingsstrategie geïnstantieerd moet worden als uitvoerbare workflow.

Gebaseerd op deze observaties, is dit proefschrift gemotiveerd met de volgende onderzoeksvraag:

> Hoe kan de state-of-the-art-methodologie voor het verrichten van MOAO bij projecten in samenwerkingsverband — waarbij een groot team van heterogene specialisten betrokken is dat een groot, complex MOAO-systeem moet bedienen — verbeterd (of vervangen) worden door een nieuw uitgevonden methodologische aanpak die de huidige con- en herconfiguratiehordes verlaagt.

Om deze vraag te beantwoorden is het doel gesteld *om een nieuwe methodologie te ontwikkelen, implementeren en evalueren, die het aspect van samenwerking in het verrichten van MOAO specifiek adresseert, en die geformaliseerde MOAO-systemen mogelijk maakt in alle stadia van de formuleringsfase, om daarbij de eerder genoemde hordes te verlagen.*

Het najagen van dit doel heeft geresulteerd in drie belangrijke ontwikkelingen die de kern van de nieuw uitgevonden methodologie vertegenwoordigen:

1. Een op grafen gebaseerde formalisering van MOAO-systemen is ontwikkeld door het uitbreiden van eerder werk op dit gebied. Deze formalisering maakt het mogelijk om MOAO-systemen te representeren als grafen in de drie stadia van de formuleringsfase:
   Tool-opslag De opslag wordt gerepresenteerd door de 'opslag connectiviteitsgraaf' die de collectie van disciplinaire tools beschrijft en hoe ze zijn verbonden.
   MOAO-probleem De 'fundamentele probleemgraaf' wordt gebruikt om het MOAO-probleem te specificeren dat geadresseerd moet worden. Deze graaf wordt gedefinieerd door het ontwerpteam, gebaseerd op de opslaggraaf, en moet voldoen aan strikte graaf-theoretische condities zodat het laatste stadium van het ontwikkelproces automatisch geïnstantieerd kan worden.
   MOAO-oplossingsstrategie Dit stadium wordt gerepresenteerd door twee grafen: de MOAO-data en de MOAO-procesgraaf. Samen vormen deze grafen de blauwdruk voor de uiteindelijke workflow in de uitvoeringsfase. Deze grafen worden automatisch gecreëerd door de geselecteerde MOAO-architectuur op te leggen aan de fundamentele probleemgraaf.
   De graaf-theoretische fundering, ontwikkeld in dit onderzoek, bevat nieuwe concepten, zoals 'instanties' en de 'cirulariteitsindex', waardoor om kan worden gegaan met specifieke problemen in MOAO-systemen, en vormt de kern van de methodologische aanpak. Deze theoretische ontwikkeling is geïmplementeerd als een open-source Python-pakket genaamd KADMOS (Knowledge- and graph-based Agile Design voor Multidisciplinaire Optimalization System).
2. Verschillende leden van het ontwerpteam hebben de KADMOS graafobjecten nodig voor een verscheidenheid aan doeleinden, zoals systeemvisualisaties, management, en het creëren van uitvoerbare workflows. Daarom is er een nieuwe standaard voorgesteld om de op grafen gebaseerde systeemformuleringen op te slaan en te delen in een applicatie-onafhankelijk format. De implementatie van deze standaard als XML-schema is genaamd CMDOWS (Common MDO Workflow Schema).
3. Met de MOAO-oplossingsstrategie — ook wel workflow blauwdruk — geformaliseerd als grafen in KADMOS en opgeslagen als XML-bestand gebaseerd op CMDOWS, kwam de mogelijkheid om de implementatiekloof te overbruggen door de uitvoerbare workflow automatisch te instantiëren in verschillende Process Integratie en Ontwerp Optimalisatie (PIOO) platformen. Deze mogelijkheid is ontwikkeld voor de platformen RCE, Optimus, en OpenMDAO (de eerste twee als interne plug-ins en de laatste als een afzonderlijk Python-pakket genaamd OpenLEGO).

Gebruikmakend van deze prototype-software-implementaties, is de nieuw uitgevonden methodologie geverifieerd en gevalideerd op basis van twee MOAO-benchmark-problemen: Sellar probleem en NASA's supersonische zakenjet casus. Deze validatie bevatte ook de complex gedistribueerde MOO-architecturen 'collaborative optimization' en 'BLISS-2000'. Bovendien wees een vergelijking van het ontwikkelproces met de state-of-the-art aanpak op een 57% tijdsvermindering voor (her)configuratie van het systeem.

De drie kerncomponenten zijn ontwikkeld in de context van het volgende-generatie MOAO-raamwerk project AGILE. In AGILE werkte een internationale groep van negentien partners samen om een nieuwe MOAO-raamwerk-generatie op te zetten en te testen: het AGILE ontwikkelraamwerk. Dit raamwerk combineert een scala aan technologieën en heeft een meerlaagse structuur, genaamd de kennisarchitectuur. De methodologische aanpak die voortkwam uit dit proefschrift (en de gerelateerde implementaties) liggen in het hart van het AGILE-raamwerk en zijn geëvalueerd in vier vliegtuigontwerpprojecten uitgevoerd in samenwerkingsverband. In deze projecten hebben grote, heterogene, internationaal opererende teams het raamwerk gebruikt om MOAO te verrichten voor vier onconventionele vliegtuigconfiguraties: strut-braced wing, blended-wing body, box-wing, en turboprop. Individuele raamwerkcomponenten zijn geëvalueerd door middel van een enquête op basis waarvan de volgende conclusies getrokken konden worden met betrekking tot de ontwikkelingen beschreven in dit proefschrift:

- CMDOWS heeft aangetoond in staat te zijn om als een datastandaard te fungeren voor de integratie van de heterogene applicatie-collectie ingezet in het AGILE-raamwerk: een gebruikers-interface om het MOAO-systeem te definiëren met KADMOS op de achtergrond, een visualisatie-tool, meerdere PIOO-platformen, en meerdere tool-opslagdatabases.
- KADMOS speelde een centrale rol in de formuleringsfase van de ontwerpprojecten en zijn vermogen om het complexe MOAO-systeem te representeren werd zeer gewaardeerd. Respondenten van de enquête schatten in dat het gebruik van KADMOS voor systeemformulering tot een gemiddelde tijdsvermindering leidt van 49%.
- Geautomatiseerde workflow-creatie met de RCE en Optimus CMDOWS plug-ins werd ook als waardevol gezien met een ingeschatte tijdsvermindering van 33%, hoewel de gecreëerde workflows wel als minder efficiënt ondervonden werden vanwege overhead die geïntroduceerd wordt door extra componenten in de resulterende workflows.

Het complete AGILE raamwerk — waarin de ontwikkelingen beschreven in dit proefschrift een centrale rol spelen — werd ingeschat als een positieve impact op het MOAO-ontwikkelproces met een tijdsvermindering van 39%.

De op grafen gebaseerde methodologische aanpak is ook getest buiten de AGILE-project context. In deze bijkomende studie is een state-of-the-art vliegtuigontwerp-toolbox, genaamd de Initiator, geherstructureerd gebruikmakend van de aanpak en code base uit dit proefschrift. De aanpak hielp om een grote computational toolbox te ontwarren en resulteerde in een goed gestructureerd, behendig raamwerk waarin disciplinaire componenten en modules gemakkelijk geïntegreerd, gekoppeld, en gebruikt kunnen worden in een verscheidenheid aan MOAO studies. Nadat het hergestructureerde raamwerk was vastgesteld, is het MOAO-systeem geconfigureerd en meermaals geherconfigureerd om het ontwerp van een typische airliner uit te voeren op basis van verschillende stategiën: ontwerpconvergentie, ontwerp met experimenten, optimalisatie. De hergestructureerde toolbox liet zien hoe dezelfde methodologie — oorspronkelijk ontwikkeld en gedemonstreerd in AGILE — algemeen genoeg is om te worden toegepast in andere raamwerken teneinde hun schaalbaarheid en flexibiliteit te verbeteren.

Twee nieuwe concepten zijn voorgesteld op basis van dit werk als een vooruitzicht en wegenkaart voor toekomstige ontwikkelingen: composiete architecturen en MOAO-bots. De eerste komt tegemoet aan de behoefte van ontwerpers om hun eigen architecturen te definiëren en te gebruiken die om kunnen gaan met de eigenaardigheden van hun specifieke MOAO-probleem. Deze behoefte werd geïdentificeerd door AGILE-gebruikers op basis van het automatiseringspotentieel mogelijk gemaakt door de combinatie van KADMOS, CMDOWS, en op CMDOWS gebaseerde workflowcreatie. Het laatste concept — MOAO-bots — is geïntroduceerd om een wegenkaart voor te stellen voor toekomstig onderzoek. De bot is gedefinieerd als een autonoom programma op een netwerk, dat een wisselwerking heeft met een MOAO-systeem en de leden van het ontwerpteam, en zich gedraagt als een lid van het team dat ontwerpvragen beantwoordt door het (her)configureren en uitvoeren van het MOAO-systeem. De geautomatiseerde aanpak gevestigd in dit onderzoek wordt gezien als de eerste MOAO-bot. Verscheidene bots zijn al voorgesteld om het ontwerpteam te ontlasten van repetitieve en foutgevoelige taken die onderdeel zijn van het MOAO-ontwikkelproces, zodat de con- en herconfiguratiehordes, die oorspronkelijk dit werk motiveerden, nog verder kunnen worden verlaagd. De daadwerkelijke uitwerking van deze bots wordt overgelaten aan onderzoekers die meedoen aan toekomstige MOAO-projecten in samenwerkingsverband.

CONTENTS

Nomenclature

Glossary

MDAO architecture A scheme (or recipe) for executing a certain MDAO problem. Well-known examples for optimization are ‘multidisciplinary feasible’ (MDF) and ‘individual discipline feasible’ (IDF).

MDAO development process The collaborative process in an engineering project aimed at configuring and reconfiguring (hence developing) the MDAO system to answer design questions.

MDAO framework A collection of support applications (e.g. planning tool, workflow software, visualization package) assembled together into a coherent set that is used by an engineering team to manage the group and the MDAO system.

MDAO problem The multidisciplinary analysis or optimization to be solved to answer a specific design question.

MDAO solution strategy The blueprint of the workflow to be executed. The solution strategy provides a formalization to solve the given MDAO problem based on a given architecture.

MDAO system The computational system containing the multidisciplinary analyses required to design a complex product. This system is a dynamic object that needs to be built, debugged, executed, and reconfigured continuously based on progressive insights of the engineering team.

MDAO system management All handling of an MDAO computation system by a engineering team, such as composition, representation, manipulation, and framework integration.

MDAO workflow The executable process in a process integration and design optimization platform matching a given MDAO solution strategy.

tool repository The collection of interlinked disciplinary components (i.e. design competences, surrogate models, mathematical relations) available for execution to the engineering team.

Acronyms

AAO	All-At-Once
ADF	AGILE Development Framework
AGILE	Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts
AMR	Architecture-specific Mathematical Relation
API	Application Programming Interface
ATR	Average Temperature Response
AVL	Athena Vortex Lattice
BIM	Building Information Modeling
BLI	Boundary Layer Ingestion

BLISS Bi-Level Integrated System Synthesis
BWA Box-Wing Aircraft
BWB Blended-Wing Body

CA Collaborative Architecture
CAD Computer-Aided Design
CDS Central Data Schema
CMDOWS Common MDO Workflow Schema
CO Collaborative Optimization
CPACS Common Parametric Aircraft Configuration Schema
CSSO Concurrent Subspace Optimization

DLR German Aerospace Center
DOC Direct Operating Cost
DOE Design Of Experiments
DSM Design Structure Matrix
DUT Delft University of Technology
DVD Design Variable Dependent
DVI Design Variable Independent

FAR Federal Aviation Regulations
FDT Functional Dependency Table
FM Fuel Mass
FPG Fundamental Problem Graph
FrEACs Future Enhanced Aircraft Configurations

GEMS Generic Engine for MDO Scenarios
GUI Graphical User Interface

I/O input and output
IDEaliSM Integrated & Distributed Engineering Services Framework for MDO
IDF Individual Discipline Feasible
InFoRMA Integration, Formalization and Recommendation of MDO Architectures
IPACS Initiator Parametric Aircraft Configuration Schema

JSON JavaScript Object Notation

KA Knowledge Architecture
KADMOS Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System

MAUD Modular Analysis and Unified Derivatives
MCG Maximal Connectivity Graph
MDA Multidisciplinary Design Analysis
MDAO Multidisciplinary Design Analysis and Optimization
MDF MultiDisciplinary Feasible
MDG MDAO Data Graph
MDK Metis-based Decomposition of KADMOS graphs
MDO Multidisciplinary Design Optimization
MPG MDAO Process Graph
MTO Maximum Take-Off

MTOM Maximum Take-Off Mass

NLR Royal Netherlands Aerospace Centre

OEM Operational Empty Mass
OEM Original Equipment Manufacturer
ONERA The French Aerospace Lab
OpenLEGO Open-source Link between AGILE and OpenMDAO

PIDO Process Integration and Design Optimization
PSG Problem Solution Graph
PSN Process Step Number

QOI Quantity Of Interest

RCE Remote Component Environment
RCG Repository Connectivity Graph
REMS REconfigurable Multidisciplinary Synthesis
RMSE Root-Mean-Square Error

SAND Simultaneous Analysis aNd Design
SBW Strut-Braced Wing
SM Surrogate Model
SMR Surrogate Model Repository
SQP Sequential Quadratic Programming
SSBJ SuperSonic Business Jet

TLAR Top-level aircraft requirement
TPA Turbo-Prop Aircraft

UID Unique IDentifier
UML Unified Modeling Language

VISTOMS VISualization TOol for MDO Systems

XDSM eXtended Design Structure Matrix
XML eXtensible Markup Language
XSD XML Schema Definition

ZF Zero-Fuel

VARIABLES

AR	=	aspect ratio
C_f	=	skin friction coefficient
D	=	data graph
D	=	drag
DT	=	non-dimensional throttle setting
E	=	edges
ESF	=	engine scale factor

F	=	objective function
$\mathcal{F}$	=	fundamental problem graph
FF	=	fuel fraction
G	=	constraint function
$\mathcal{G}$	=	graph
Gc	=	consistency constraint function
L	=	lift
M	=	Mach
$\mathcal{M}$	=	MDAO graph
N_z	=	max. load factor
$\mathcal{P}$	=	process graph
$\mathcal{R}$	=	repository connectivity graph
R	=	range
S	=	surface area
S_{ref}	=	wing reference area
SFC	=	specific fuel consumption
T	=	throttle (SSBJ case)
T	=	thrust
$Temp$	=	temperature
V	=	vertices
V	=	volume
W	=	weight
b	=	wing span
c_D	=	drag coefficient
c_L	=	lift coefficient
c_r	=	root chord length
$dpdx$	=	pressure ratio
e	=	edge
f	=	factor
fin	=	fineness ratio (L/D)
gc	=	consistency constraint
h	=	altitude
k	=	scale factor
m	=	mass
n	=	number of
s	=	distance
t	=	time
t/c	=	thickness-to-chord ratio
v	=	node
x_{sec}	=	wingbox section height
Γ	=	dihedral angle
Λ	=	sweep angle
Θ	=	wing twist
λ	=	taper ratio
ξ	=	spar location
σ	=	stress

Subscripts, superscripts, and oversets

BE	=	baseline engine
E	=	engine
F	=	fuel
FS	=	front spar
FT	=	fuel tank
L	=	landing
MTO	=	maximum take-off
RS	=	rear spar
T	=	total
TO	=	take-off
WA	=	wingless aircraft
WL	=	wingloading
c	=	copy
cr	=	cruise
f	=	function
i	=	instance
p	=	process
$prop$	=	propulsion
s	=	sample vector
v	=	variable
0	=	initial guess value
$\sim$	=	scaled
$*$	=	final value

I

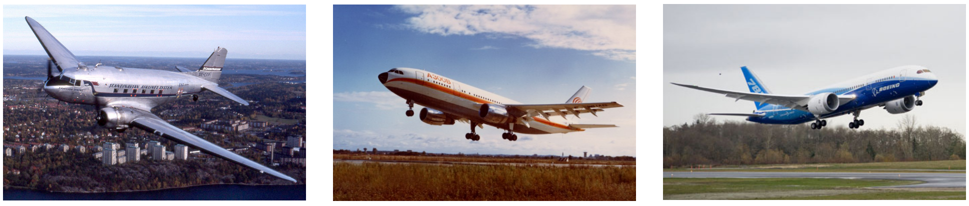
Background

1

Introduction

Designing better aircraft is not a straightforward task. The aircraft flying around the world today are based on decades of development and experience in the aerospace industry. Today's aircraft are significantly more efficient and sustainable than the machines of past decades, thanks to major developments in separate disciplinary fields impacting their design, such as propulsion, aerodynamics and structures. However, the conservative nature of the aerospace industry, where safety concerns have resulted in stringent certification requirements, in combination with decades of developments drawing on the same configuration makes it increasingly hard to significantly improve today's aircraft even further.

The configuration of today's aircraft provides a clear indicator of the innovation hurdle that designers are facing; airliners have been based upon the same basic layout since the 1930s, i.e. the tube-and-wing design (see Fig. 1.1). There is good reason for choosing this configuration: designing an aircraft requires the difficult integration of multiple engineering disciplines into one vehicle, the tube-and-wing configuration somewhat relieves this burden by separating the main functions of an aircraft as different components. One can carry passengers with the fuselage (tube), create lift with the wings, propel the vehicle using engines, and ensure stability & control using the tail. Due to decades of development and manufacturing, Original Equipment Manufacturers (OEMs) are also completely geared towards this configuration, and in their "divide and conquer" approach they have structured their organizations accordingly.



a) Douglas DC-3 (1935).[§] b) Airbus A300 (1972).[¶] c) Boeing 787-8 (2009).[‖]

Figure 1.1: Three airliners that were top of the bill in their time (year of first flight indicated between brackets). Like all current airliner families manufactured by companies such as Airbus, Boeing, Bombardier, and Embraer, every design uses the tube-and-wing layout. (For credit footnotes, see next page.)

The compartmentalized organizational structure at large organizations results in a highly unautomated and decoupled aircraft design process [1, 2]. Disciplinary analyses are performed in isolation by separated departments, so that specialists maintain ownership and control of their field, with synthesis only taking place manually through sporadic sharing of new information about an adjusted design discipline. This lack of automation greatly restricts the number of design iterations that can be performed, thereby hampering the potential improvements that could be made if more iterations were allowed in a shorter amount of time [3]. In addition, the lack of automated and coupled disciplinary analyses complicates the impact assessment of local changes on the overall design.

The impact of this unautomated and decoupled process can be felt in all three categories of design approaches described by Gero and Maher [4]: routine, innovative and creative (see Fig. 1.2). In routine design, the known design space is used to find a new solution, whereas with innovative design, the design space is extended around its known boundaries. Finally, with creative design, a completely new design space is explored to find an answer to the task at hand. Over the past decades, OEMs have only taken routine and innovative design approaches to updating the tube-and-wing aircraft, with creative design posing too high a risk for the conservative attitude of the industry.

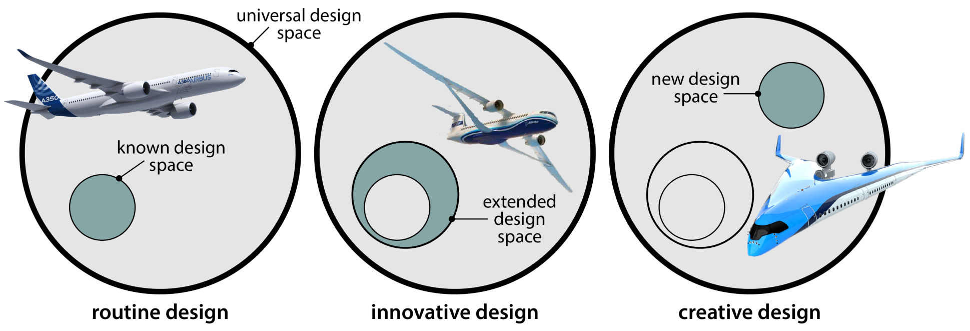


Figure 1.2: Three different approaches to performing a design task (adapted from [4])

Each design approach is restricted in its own way by the highly unautomated and decoupled design practice in industry. Performing routine design would mean attempting to further improve the tube-and-wing aircraft in its known design space. With decades of experience inside this space, manually finding a better design of such a complex and coupled vehicle is almost impossible. Therefore, the only way to squeeze out a high-level performance improvement for these designs would be to couple and automate the disciplinary analyses and perform thousands of iterations using a design space exploration algorithm.

With innovative design, the tube-and-wing aircraft would be improved by supplementing it with a new technology. Such a technology, can only be assessed properly if its impact on the full aircraft can be appraised by coupling all the analyses and if a new optimal design can be synthesized around it. Hence, automating the multidisciplinary analysis of the full aircraft is of key importance to correctly assess technological innovations that fall outside the routine design space.

§Credits: Flygande Veteraner

¶Credits: Airbus S.A.S.

‖Credits: The Boeing Company

With creative design, the design team is presented with a very different challenge. In routine and innovative design, the analyses can be based on empirical knowledge gathered in previous projects; Methods based on statistics and experience are used to pursue the typical design evolution from conceptual, to preliminary, to detailed design. Contrarily, the very nature of creative design means that the team is looking at a previously unexplored design space, thereby making it necessary to apply higher fidelity physics-based analyses earlier in the process. In addition, many configurations considered in this approach have highly integrated components (e.g. blended-wing body or flying-V plane, see right side of Fig. 1.2 for the latter concept) where 'everything affects everything'. Designing such configurations using physics-based models will only work if the different disciplinary analyses are coupled together so that their interactions are modeled correctly. The often highly integrated concepts also prevent the design team from changing the design intuitively, meaning that automatic design exploration methods become a necessity.

In short, the following challenges are identified in the aircraft design practice:

- Uncoupled multidisciplinary synthesis limits high-level performance improvements for routine design.
- Creative design (and also some innovative design) cannot rely on empirical knowledge and therefore requires high-fidelity coupled analyses earlier.
- Highly integrated concepts in creative design require design intuition to be augmented with design exploration methods.

Looking at these challenges in the different design approaches, it is clear that Multidisciplinary Design Analysis and Optimization (MDAO) can offer major benefits. With MDAO, a multidisciplinary system is created by directly coupling the different design disciplines so that the full system can be synthesized automatically. The disciplinary analyses in this system are fully automated, enabling a team to perform a tremendous number of design iterations and providing the ability to link design space exploration algorithms (i.e. optimization) as well. MDAO is not a new idea; it has been a field of research for many decades [5–8]. In the early 2000s, Boeing Phantom Works scientists estimated that MDAO can offer 8-10% gains for innovative design and even 40-50% gains for the creative design of radically new and undeveloped concepts [9, 10].

Despite this huge potential, MDAO has not yet been widely incorporated into the everyday design process that can be seen at OEMs today. If MDAO is applied, then it is generally limited to the conceptual and preliminary design stages only incorporating classical MDAO disciplines, such as aerodynamics and structures [11]. The true potential of MDAO for the aerospace industry would only be unlocked if all disciplines comprising an aircraft design could be coupled together, such as onboard systems, engine design and integration, and environmental impact. But for several reasons, MDAO cannot be utilized in the practical reality of the aerospace industry yet.

Both technical and non-technical challenges [1, 2, 12, 13] are currently hampering the full exploitation of MDAO in industry. Technically, one needs to be able to couple, maintain and execute in a single system a large variety of disciplinary analyses residing in different parts of a large, heterogeneous organization; one must also have the algorithms and computational resources available to work with that system. Non-technically, MDAO presents a radically different approach to designing in groups that requires a different organizational structure, and thus a more open and collaborative mindset, within the heterogeneous team of disciplinary experts.

The multidisciplinary computational system that needs to be built and executed collaboratively will be referred to in this dissertation as the *MDAO system*. This system is a dynamic object that needs to be built, debugged, executed, and reconfigured continuously based on progressive insights of the engineering team. Fig. 1.3 depicts how the system transforms through five different stages in a typical MDAO-based project. This MDAO development process is divided in two main phases: formulation and execution.

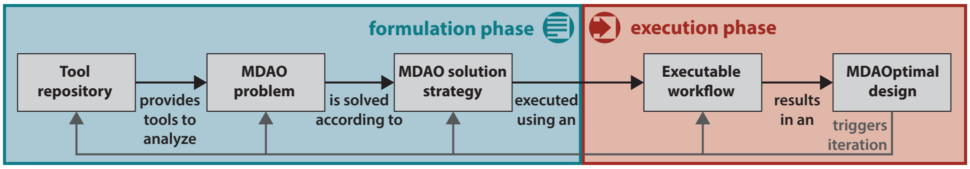


Figure 1.3: The five main stages of an MDAO system within a typical MDAO-based project and their relation

In the formulation phase the MDAO system is set up in three stages. First the tools (i.e. disciplinary analyses) required to analyze the design are collected in a tool repository. Then the MDAO problem is defined based on that repository, i.e. design variables, constraints and an objective are specified. Finally, different strategies can be implemented to solve the problem at hand. The strategy defines the coordination of the execution process and specifies the interactions between the disciplinary analyses and other components, such as optimizers and convergers. In MDAO, these strategies are called *architectures*, with classical examples being MultiDisciplinary Feasible (MDF) and Individual Discipline Feasible (IDF) [14]. Imposing an architecture on a problem results in the MDAO solution strategy, which represents the blueprint of the executable workflow integrated in the next phase.

In the execution phase this workflow has to be built based on this blueprint. This is a cumbersome task, especially for large, complex blueprints of systems involving a large number of disciplines and couplings. In the industrial practice, this workflow is often built using a PIDO (Process Integration and Design Optimization) platform, such as ModelCenter Integrate [S1], Optimus [S2], or RCE [S3]. When the workflow is executed, this will (hopefully) lead to the final design, or a range of design options in case of design space exploration. However, more realistically the found design option is food for thought and triggers the team to improve the MDAO system based on new insights. For example, a new tool might be required to analyze an extra aspect of the vehicle; or the MDAO problem needs to be adjusted to include additional constraints and/or new design variables; or a new strategy is chosen to improve workflow performance.

The need for *continuous reconfiguration* lies at the core of any design practice. This is well illustrated by Piperni et al. [11], who describe the adopted approach at Bombardier to perform multidisciplinary preliminary design of aircraft:

1. Perform a Design Of Experiments (DOE) to filter for significant design variables.
2. Create approximation model(s) based on the DOEs of previous step.
3. Perform a surrogate-based optimization using the approximation methods.
4. Perform a full optimization using exact models, starting from the optimum solution of the previous step and only using the most significant variables.

In this approach, at least three workflows have to be generated, and the team is using intermediate results to configure the next workflow. Looking at Fig. 1.3, the adopted approach thereby requires the team to perform three iterations of the MDAO development process. Hence, in any design practice the MDAO system needs to be reconfigured continuously based on the adopted design approach and the needs of the design team, and one is required to move between the formulation and execution phases repeatedly.

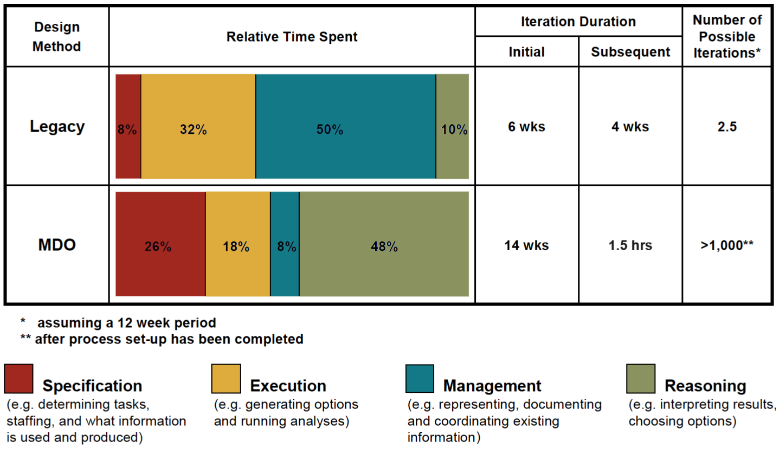
Two major hurdles are present in the current MDAO development process from Fig. 1.3, here referred to as the *configuration* and *reconfiguration* hurdles. The first, which combines technical and non-technical challenges, is that before a first workflow can be run, a fully automated, highly coupled MDAO system has to be formulated ready for execution. This is not a trivial task as the multidisciplinary analysis workflow (which is a necessary condition for MDAO) is only possible if all the connections between the different analyses are created correctly, down to the smallest detail. As Pate et al. [15] point out: the formulation of these problems has become increasingly complex as the number of analysis tools and design variables included in typical studies has grown. In this context the problem of determining a feasible data flow between tools to produce a specified set of system-level outputs is combinatorially challenging. Especially when complex and high-fidelity tools need to be included, the cost and time requirements to integrate the MDAO system can easily approach the cost and time requirements of creating any of the discipline analyses themselves.

In past collaborative MDAO projects, a large team would spend around 60-80% of the project time to configure this first version of the executable MDAO system [16]. This leaves little time for performing design iterations or for further improving the system progressively based on intermediate results. Hence, taking the MDAO approach puts the heavy burden of setting up this complex system on the design team, while the first results will only emerge after 60-80% of the project time has already been spent. In other words, the long set-up time makes MDAO riskier than the classical design approach.

The configuration hurdle in the MDAO development process is effectively summarized by Flager and Haymaker[3] in Fig. 1.4, which is the result of a comparison study between the Boeing legacy design method and an MDAO-based process for the development of a hypersonic vehicle [9, 10]. The main hurdle identified for the MDAO approach was the long set-up time of the workflow (14 weeks), which was more than double the time necessary to deliver the first design using the legacy method (6 weeks), with the consequent risk associated to the late availability of the first results. The potential of the MDAO-based process, however, is also clear: once the set-up of the workflow is complete, an enormous number of iterations can be performed (1000 vs. 2.5 for the legacy method), leaving much more time to the interpretation of the results rather than the coordination of the generated information. In industrial design situations, where deadlines are strict and a first conceptual design is normally already available within a short time, the postponement of the initial design, in combination with the time-consuming detailed specification, is experienced as a risk.

Configuring large, heterogeneous MDAO systems more quickly would already be an achievement, but at that point, the following requirements also have to be met: the system has to stay dynamic, comprehensible, and its stages need to be linked such that a change in an earlier stage can progress automatically to a later one. This is not trivial and constitutes the (second) reconfiguration hurdle. Gallard et al. [17] refer to this hurdle as the "combinatorial issue" in the MDAO development process. Fig. 1.5 depicts a small graph with different choices on use case, strategy, algorithm and PIDO platform. Gallard et al. point out:

> There are 99 paths in this graph: each one represents a possible process [ex-



Design Method	Relative Time Spent	Iteration Duration		Number of Possible Iterations*
		Initial	Subsequent	
Legacy	🟥 8% · 🟨 32% · 🟦 50% · 🟩 10%	6 wks	4 wks	2.5
MDO	🟥 26% · 🟨 18% · 🟦 8% · 🟩 48%	14 wks	1.5 hrs	>1,000**

* assuming a 12 week period
** after process set-up has been completed

🟥 **Specification** (e.g. determining tasks, staffing, and what information is used and produced)
🟨 **Execution** (e.g. generating options and running analyses)
🟦 **Management** (e.g. representing, documenting and coordinating existing information)
🟩 **Reasoning** (e.g. interpreting results, choosing options)

Figure 1.4: Legacy and MDAO-based design process metrics for the design of a hypersonic vehicle [3]

ecutable workflow] [...]. In addition, each of these items is implemented in a separate software package, which evolves in time. For three different versions of each software, this leads to $3^{99} \approx 10^{47}$ potential processes. This is a major issue for maintenance and industrial deployment of MDO methods.

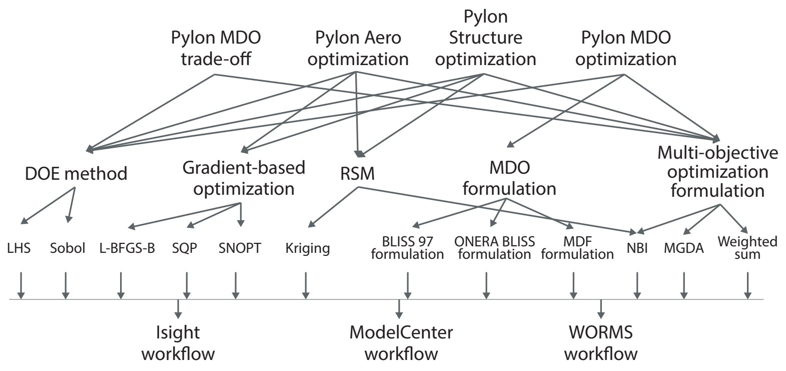


Figure 1.5: Graph with different options at each level for configuring the required executable workflow [17]

In past MDAO projects, the stages in Fig. 1.3 were decoupled, meaning that there are gaps between stages that need to be filled manually. For example, defining the tool repository and determining how different tools are coupled together is done using spreadsheets (e.g. Microsoft Excel [S4]), while the solution strategy is defined separately in a flowchart software (e.g. Microsoft Visio [S5]), and the executable workflow is built through the Graphical User Interface (GUI) of the selected PIDO platform (e.g. Optimus [S2]). With such decoupled stages, reconfiguring the MDAO system gets complicated very easily. For example, if one tool is modified or substituted by a new one requiring an extra input parameter, then the team might have to add an additional tool able to provide that parameter, and would need to independently update their spreadsheets, flowchart and workflow accordingly. Alternatively, one could also just update the workflow, but this would lead to a mismatch between the formulation and the execution phases, ultimately making the executable workflow a large, incomprehensible, black box. Hence, to support continuous reconfiguration, the formulation and execution stages of the system need to be synchronized semi-automatically and supported by computer automation as much as possible in order to keep the design team agile and guarantee a high level of trust in the final results.

Of the gaps between the system stages in Fig. 1.3, a particularly challenging transformation is performed from MDAO solution strategy to executable workflow. Here, the system switches from the formulation phase to the execution phase. As mentioned, the executable workflow is generally built manually using the GUI of a PIDO platform. This workflow creation method entails a cumbersome, error-prone task for the design team. Additionally, this method is completely decoupled from the formulation phase, since the formulated system is incompatible with the way executable workflow are generated in PIDO platforms. This gap between formulation and execution is called the *implementation gap*[*] and bridging this gap constitutes a major challenge to achieve (re)configurable MDAO systems.

The culmination of the configuration and reconfiguration hurdles just discussed, have hampered the implementation of large MDAO systems in industry. Generally, if MDAO is used, tailor-made solutions are developed for a specific application. Thereby any developed system and the applied methodology to establish it, are too problem-specific and difficult to reapply in a different context (i.e. a new design project).

A possible solution to these hurdles would be to more strictly formalize the MDAO system in the formulation phase. At the moment, setting up the executable workflow is generally done through a trial-and-error process, where each disciplinary specialist is asked to add their analysis to the overall system, and the implementation is checked by running and debugging the system itself. This is, however, a very inefficient method to build a multidisciplinary system, especially a large and heterogeneous one. Instead, it would be more advantageous to first properly formulate the MDAO system, so that the design team has the right blueprint before implementing the executable MDAO system itself. With the right formalization, the different stages can be tightly coupled and the workflow could be generated automatically. To draw an analogy with a completely different field of work: the current way of building MDAO systems is like trying to play a symphony with a large orchestra in which no one has the score. It would take a long time for a conductor to get through the full symphony if no one knew beforehand what to play and when to play it.

Based on these observations, the following research question motivates this dissertation:

> How can the state-of-the-art methodology for performing MDAO in collaborative projects — involving a large team of heterogeneous experts that needs to operate a large, complex MDAO system — be enhanced (or replaced) by a novel methodological approach that lowers the con- and reconfiguration hurdles encountered nowadays?

Different formalizations of MDAO systems can be found in literature (and will be discussed in the next chapter). However, a comprehensive methodology and implementation to test these formalizations with the aim of reducing the set-up time in realistic collaborative design projects does not exist. This dissertation presents the first such implementation. In addition, the developments shown in this dissertation are also tested in realistic projects, as this work is part of a broader effort that was started in the MDAO com-

[*] N.B. The implementation gap has a strong analogy with the gap between formulated policies and their actual implementation, which inspired the term to be used here.

munity through the AGILE (Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts) project [18]. With respect to the AGILE project, this dissertation constitutes several key components of a novel-generation MDAO framework for collaborative design.

The main research question is addressed by pursuing the goal:

> to develop, implement, and assess a new methodology, specifically addressing the collaborative aspects of performing MDAO, that enables formalized MDAO systems in all stages of the formulation phase, such to lower the aforementioned hurdles.

This goal is broken down in several subgoals:

- Develop methodology concerning:
  - formalization of MDAO systems in the formulation phase;
  - storing and sharing the formulated system.
- Implement a:
  - software solution to formulate MDAO systems;
  - storage solution to share system definitions;
  - solution for the implementation gap between formulation and execution.
- Assess implemented methodology in:
  - reference MDAO test cases for verification and validation purposes;
  - the AGILE collaborative MDAO project to measure impact on the design process in a collaborative, realistic environment;
  - an unrelated design environment to assert the general applicability of the methodology outside the AGILE context in which it was developed.

Fig. 1.6 depicts the relation between the MDAO-based development process (Fig. 1.3) and the chapters of this dissertation. This study began by assessing the state of the art of the formalization and implementation of MDAO systems in collaborative environments, and by identifying the remaining challenges in this field (Chapter 2). Subsequently, a new formalization of MDAO systems and its implementation are presented in Chapter 3, followed by the accompanying storage standard in Chapter 4. The automatic generation of executable workflows to bridge the implementation gap is discussed in Chapter 5. This dissertation's developments are integrated in the MDAO framework developed in the AGILE project to assess its impact on the broader collaborative MDAO process with multiple design projects in Chapter 6. Additionally, the methodology has also been applied in an alternative aircraft design toolbox unrelated to the AGILE project in Chapter 7 to demonstrate its generic applicability outside the context in which it was developed. Finally, an outlook on future opportunities will be presented in Chapter 8.

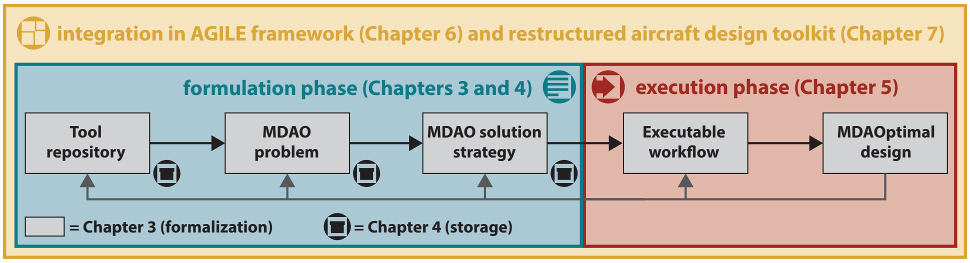

integration in AGILE framework (Chapter 6) and restructured aircraft design toolkit (Chapter 7)
formulation phase (Chapters 3 and 4)
execution phase (Chapter 5)
Tool repository
MDAO problem
MDAO solution strategy
Executable workflow
MDAOptimal design
= Chapter 3 (formalization)
= Chapter 4 (storage)

Figure 1.6: Mapping between MDAO-based development process and the chapters of this dissertation

2 State of the art in MDAO system management

Performing MDAO collaboratively can be a challenging ordeal full of technical and non-technical hurdles, as was illustrated in the previous chapter. A complex computational system needs to be set up and executed in a large, heterogeneous team. This team contains different stakeholders, such as disciplinary specialists, project managers, and decision makers from upper management. Each team member needs to have a proper understanding of the MDAO system to defend their interests in the project. Within such a complicated environment, useful executions of the computational system can only be achieved efficiently if the system is managed properly. The handling and management of collaborative computational systems has been a topic of research for many decades, where researchers have treated different aspects of these systems, ranging from composition and representation to execution and framework integration. In this chapter, each section will discuss one of the following aspects of the state of the art in MDAO system management:

Composition: how a collection of disciplinary tools is assembled into a single MDAO system.

Representation: how the same MDAO system can be described in multiple ways (e.g. visually, mathematically) to serve different goals.

Manipulation: how MDAO systems are manipulated to advance from one stage in the MDAO development process to another (see Fig. 1.3).

Coordination: how the execution of the MDAO system can be coordinated based on different architectures to solve the same underlying design problem.

Execution: how MDAO systems are executed using different PIDO platforms.

Framework integration: how the handling of MDAO systems is integrated in collaborative frameworks in recent projects.

Each of the above aspects will be addressed in a dedicated section. The chapter will end with a section on the next-generation MDAO framework and the identification of its needs that are addressed in this work.

2.1. Composition

The composition of the MDAO system, i.e. the way tool inputs and outputs (I/Os) are connected to form a single system, is an unreported topic in most work. Generally, a convenient approach is adopted without the need to consider alternatives, as the approach meets the requirements for the particular system, is used in a non-collaborative environment, and the system is only composed once. With respect to the MDAO development process in Fig. 1.3, system composition is part of the first process stage in which the tool repository is constructed. Though it might seem trivial, depending on the approach, the assembly of all components (i.e. tools in the repository) can be a cumbersome task for large MDAO systems, with a considerable impact on the time spent in the formulation phase. This is especially true when a large, heterogeneous design team is involved, since the composition process is a collaborative effort that requires contributions by and agreement between all tool specialists.

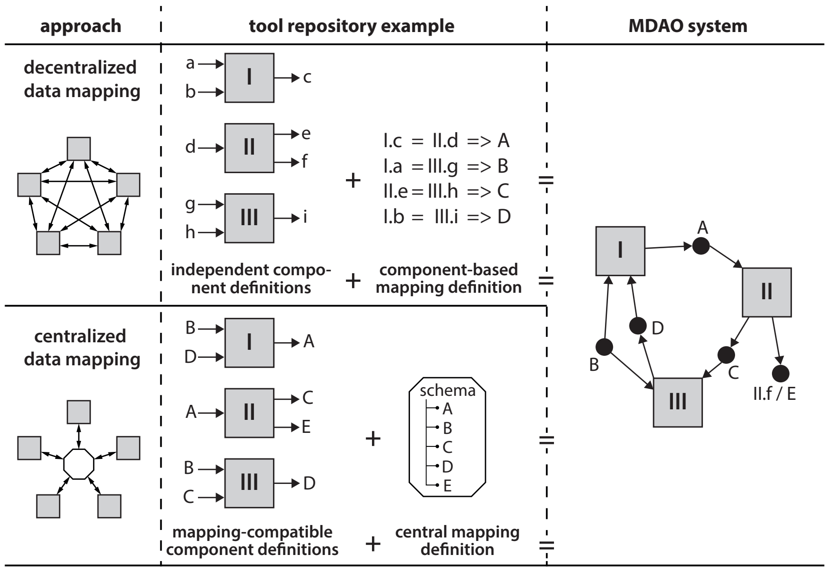


Figure 2.1: Schematic summary of the two main MDAO system composition approaches found in literature [19–27].

A key aspect of system composition concerns the determination whether variables are only related to a single component or shared with and/or coupled to multiple components. In this work, different composition methods found in literature are classified in two basic approaches:

- decentralized data mapping
- centralized data mapping

Both approaches are schematically summarized in Fig. 2.1. With a decentralized approach a component's I/Os are defined independently and connected through a separate mapping definition directly between the components. The centralized approach requires components to have I/Os that match a central mapping definition. Although both approaches would eventually result in the same MDAO system shown on the right in Fig. 2.1, the reason to elaborately discuss these approaches here is that each one has a different workload and scalability. Furthermore, the approaches also differ in their collaborative dynamic, as responsibilities and workload change for team members providing disciplinary analyses (tool specialists) and members responsible for the integration of the analyses into one system (system integrators).

2.1.1. Decentralized data mapping approach

Tosserams et al. [19] have a decentralized data mapping approach in their Ψ language for problem partitioning (the division of a system into multiple subsystems to enable distributed analysis). In their language each component can be defined independently and it is the responsibility of the component provider to define local and global input variables. When the system is assembled, the integrator has the responsibility to manually define the coupled I/Os between components, that is to identify which globally defined variables are actually referring to the same component values. A similar approach can be found in object-oriented frameworks such as πMDO [20] and OpenMDAO [21–23] [S6]. In OpenMDAO the user can provide different naming conventions for the same variable in each component and then explicitly state that the components are actually referring to the same variable by “connecting” them.

The decentralized data mapping approach leads to a high workload for the system integrator, proportional to the size of the MDAO system. The size of the system is measured by the number of components and the number of I/Os. The mapping definition (see Fig. 2.1) explicitly connects coupled variables. This definition grows with the number of tools and variables, and thereby the tasks to modify tool mappings or add new tools to the system become cumbersome. The maximum number of unidirectional tool interfaces equals $n(n-1)$, where n stands for the number of tools, and grows quadratically with each new tool in the repository. A clear advantage of this approach is the freedom left to the various tool providers, as they can decide their own variable names as they see fit. Hence, the MDAO system integrator is assigned extra responsibilities, in exchange of a non-intrusive approach towards the tool providers. Furthermore, as connections are handcrafted by the integrator, a valid system can be instantiated directly, without the need to fix problematic nodes or connections (e.g. multiple tools providing the same input value for another tool) post hoc.

2.1.2. Centralized data mapping approach

In the centralized data mapping approach, a system-wide naming convention is used to establish component connectivity. This system-wide naming convention is called a schema. Fig. 2.2 summarizes the mapping approaches for system composition. Key advantage of this approach is that the maximum number of unidirectional tool interfaces equals $2n$ and scales linearly (instead of quadratically for the decentralized approach) with the number of tools in the repository. The schema of the centralized approach can be either

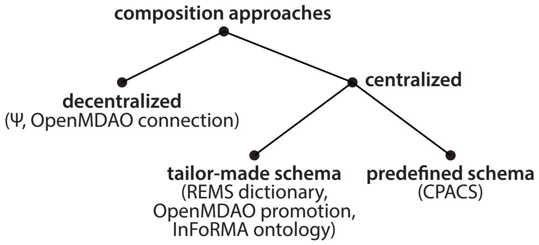


Figure 2.2: Breakdown of composition approaches

tailor-made or fixed. The tailor-made schema is more flexible and defined 'on the fly'. Alexandrov and Lewis [24, 25] use this method in their linguistic approach called REMS (REconfigurable Multidisciplinary Synthesis). With REMS, component providers define their I/O variables according to a shared dictionary of variable names, which needs to be expanded with each new system component that introduces new variables. The integrator is thus relieved of the mapping burden, at the expense of an extra management task for the dictionary and a limited freedom in variable naming for the tool providers. This approach can also be used in OpenMDAO by 'promoting' variable names to a common shared variable name within the model, thereby automatically connecting these variables. Also Hoogreef [26] uses the tailor-made centralized data mapping approach in his ontology-based MDAO-support platform InFoRMA (Integration, Formalization and Recommendation of MDO Architectures). With this platform the user builds and assembles the system using an interactive N^2 chart to add components and define couplings. Each coupling is added to the system definition by extending the tailor-made schema of the underlying ontology.

The other variant of the centralized approach (Fig. 2.2) would be to adopt a predefined Central Data Schema (CDS). The adoption of a CDS avoids the issue of updating the dictionary (REMS) or list of promoted names (OpenMDAO) for each system composition. Nagel et al.[27] have proposed a standard eXtensible Markup Language (XML)-based schema for aircraft design, called Common Parametric Aircraft Configuration Schema (CPACS) [S7]. In practice, CPACS (pronounced as: seapacks) provides an extensive predefined standard dictionary, which is meant to contain all the typical I/O data of the analysis tools; this concerns both geometrical definitions (e.g. airfoils, wings, engines) and design analysis data (e.g. component masses, mission definition, lift-drag polar) typically used in conceptual and preliminary aircraft design. When using CPACS, each disciplinary tool in the MDAO system takes a CPACS file instance as input and has to write its results to a new CPACS file. All data is then gathered by merging the different complementary CPACS files. Thus, once all component providers have 'made the investment' of rendering their tool CPACS-compatible, assembling even large MDAO systems becomes a relatively easy task.

Fig. 2.3 visualizes the high-level elements of CPACS. A fundamental idea in the schema definition is that information is only stored once in the CPACS file at its designated location. This information can then be used by other elements by adding a Unique IDentifier (UID) to it and referring to that identifier in the other element. For example, the material properties of Aluminum-2024 would be defined under the 'materials' element (level 3 in Fig. 2.3) and one could use the UID of this material to indicate which parts of the wings and fuselages are made out of this material by referencing that UID in the appropriate segments of the 'wings' and 'fuselages' elements (level 4 in Fig. 2.3). A similar example from the CPACS documentation is provided in Fig. 2.4.

A key advantage of using a CDS is that one can build an *ecosystem* of support applications around the structured schema definition to facilitate the design team integrating an MDAO system. For example, the German Aerospace Center (DLR) developed a suite of open-source software packages to enable the assembly of MDAO systems for aircraft design, based on CPACS:

TiXI [S8]: Read and write XML files with CPACS-specific functionality (e.g. check UIDs).
TiGL [S9]: Library to geometrically handle and visualize CPACS files.

[†]Adapted from: `https://cpacs.de/#contents`, accessed: May 27, 2019

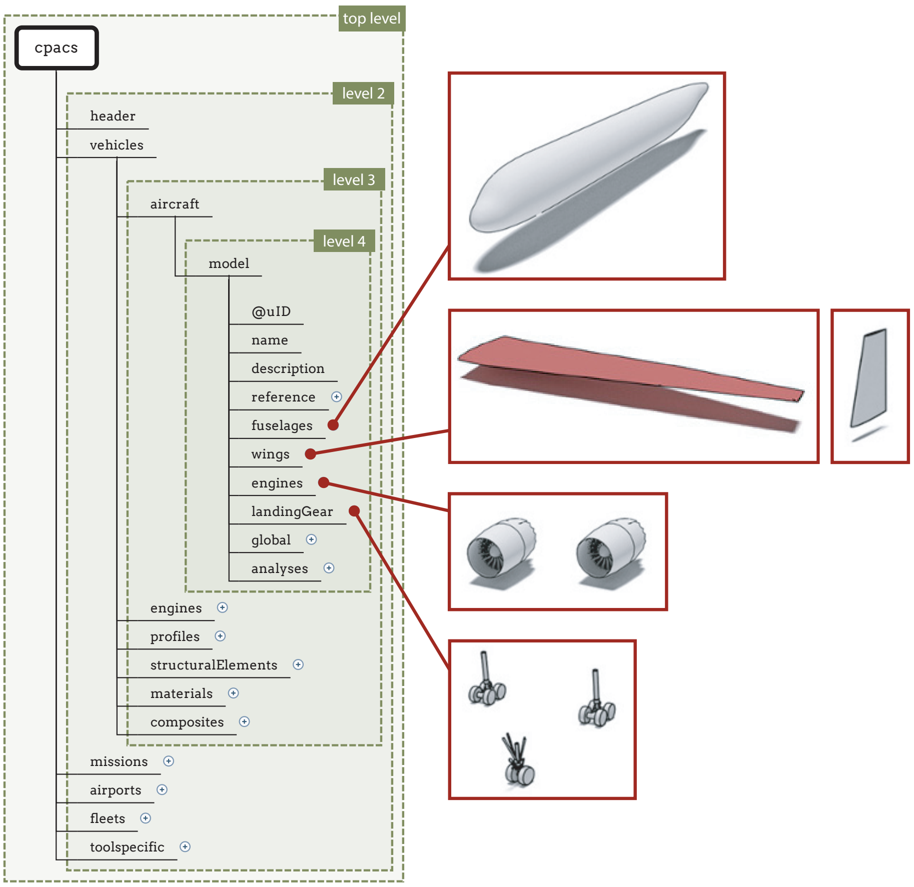


Figure 2.3: High-level elements of CPACS and the geometries they represent.†

```
<fuselage uID="ATTAS fuselage">...
```

Such a node with a UID is then typically referred to by a subnode, like:

```
<wing>
    <name>ATTAS main wing</name>
    <parentUID isLink="True">ATTAS fuselage</parentUID>
    ...
```

Figure 2.4: Example of UID referencing to define a fuselage-wing intersection [28]

RCE [S3]: PIDO platform with CPACS-specific functionality.

The CPACS definition has so far been used in multiple collaborative projects on aircraft MDAO [29–32] and has proven its value facilitating the assembly of large and distributed MDAO systems. Though CPACS is aircraft specific, the approach of a CDS would prove its value in any other application domain, given a CDS is made available a priori. The approach is gaining momentum in the MDAO community. For example, Dykes et al. [33] developed a similar standardization for the wind energy sector to connect the range of tools involved to perform wind farm MDAO. This schema also uses file I/O to exchange data (YAML-based instead of XML).[*] A well-established CDS in another domain is BIM (Building Information Modeling), which is used as a central digital representation of designs in civil engineering and architecture.

The choice for adopting one of these fixed schemas in a project impacts the collaborative dynamic by shifting responsibilities and workload. Instead of being free in defining their tool I/Os, the tool specialists first have to get acquainted with the CDS, and then have to make their tools compatible with it (also called tool wrapping or, in case of CPACS: tool CPACSization). Hence, the specialists are forced to comply with the storage format of the schema, or have to plead for an adjustment of the schema that is supported by the whole team.[†] The integrator has to manage the CDS and make sure that all required I/Os are actually included. Furthermore, the integrator has to determine how tools are coupled based on their I/Os with respect to the schema.

2.1.3. Comparison and conclusion

Decentralized data mapping approaches offer more control over the correctness of the assembled system, but bring forth the burden of mapping the variables used and produced by each component. This approach leaves the most freedom to the tool specialists, but it is does not scale well due to the required mapping. A centralized approach with a tailor-made schema removes the mapping burden, but at the risk of losing control over the system consistency and at the expenses of making all tools compatible with the I/O variable mapping definition and creating/updating such definition for each different MDAO system. The adoption of a fixed schema can remove this last issue for the specific domain the standard is defined (e.g. CPACS for aircraft conceptual and preliminary design), but it comes with three limitations:

1. The schema might not include all the required I/O elements and needs extensions.
2. One can easily loose oversight on the system's connectivity. For example, if a tool providing the values for certain CPACS data nodes is removed, it is difficult to determine whether some other tool is affected by the lack of those input data.
3. Contrary to the handcrafted connections in the decentralized approach, the indirect component couplings of the schema-based approach can easily lead to a system with problematic variables and connections (e.g. multiple tools writing the same schema element as output).

[*]It is worth noting that, conceptually, a CDS approach does not necessarily depend on file-based data exchange (e.g. a string-based naming convention or Python dictionary), but in practice a broadly recognized file standard (i.e. XML, YAML) is used for convenience.

[†]To release some of this rigidness with a fixed schema, CPACS includes a special element called 'toolspecific' (see level 2 in Fig. 2.3). Specialists can put any missing I/Os in that element, provided that these I/Os are only relevant for the tool itself and not coupled to other tools.

Hence, a dedicated formulation system is required to provide system oversight, check for problematic variables and connections, and fix them.

In conclusion, one could see the centralized mapping with a CDS as a good approach to share the burden of composing an MDAO system in a large, heterogeneous design team: the tool specialist 'only' has to make sure to comply to the CDS while the integrator 'only' has to ensure a complete schema. Though both tasks are not as trivial as they might seem in the collaborative engineering reality.

2.2. REPRESENTATION

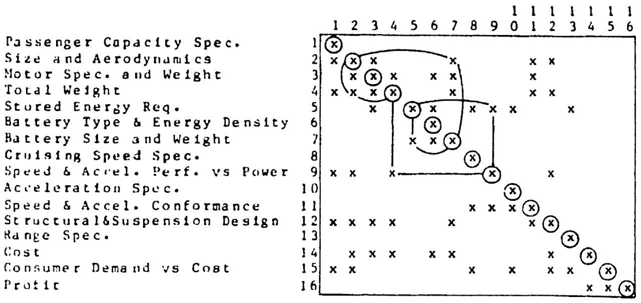
The system representations in this section are grouped by their intended use. Representations aimed at providing a human-readable overview of the MDAO system are covered first in §§2.2.1. Subsequently, machine-interpretable representations geared towards performing specific automated operations (e.g. partitioning) are discussed in §§2.2.2.

2.2.1. HUMAN-READABLE

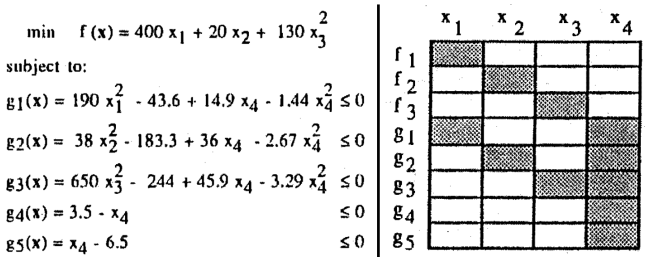
Well-known human-readable representations of complete systems are the N^2 chart [34] / Design Structure Matrix (DSM) [35] and the Functional Dependency Table (FDT) [36], see Fig. 2.5a and Fig. 2.5b for examples. The DSM approach, which only specifies the data coupling between different components, was extended by Lambe and Martins [37] into the eXtended Design Structure Matrix (XDSM) to include also information concerning the process to be executed. Today, the XDSM notation can be considered a defacto standard within the MDAO community for providing human-readable overviews of MDAO systems, independent from any proprietary PIDO platform formalization. Since the readability of an XDSM as a static document degrades with the size of the represented computational system, web-based dynamic XDSM visualization tools have been developed by Gazaix et al. [38] and Aigner et al. [39], with their respective developments of the XDSMjs [S10] package and VISTOMS (VISualization TOol for MDO Systems) [S11].

The XDSM notation from Lambe and Martins has served as a basis for all MDAO system visualizations shown in this dissertation. A small example of an XDSM is shown in Fig. 2.5c. The executable components of the MDAO system are positioned on the diagonal. Iterative components (e.g. optimizers and convergers) have an elliptical shape, while non-iterative components (e.g. disciplinary analyses and mathematical relations) are visualized as rectangles. Data dependencies between components are denoted by off-diagonal parallelograms, where grey-colored shapes indicate couplings and white-colored ones indicate system I/Os. Note that the data dependencies should be read in a clockwise direction. Data links between components are accentuated with thick grey lines. Finally, the execution process of the MDAO solution strategy is indicated using numbers inside the blocks in combination with thin black process connection lines.

All commercial PIDO platforms (which will be discussed in more detail in §2.5) provide a GUI to assemble and display executable MDAO systems according to dedicated representation approaches. Some, like ModelCenter® Integrate [S1], offer N^2-like representation, others, like Optimus [S2], offer more freedom in the representation of workflows, including multi-level visualization, but do not display the optimization modules.



a) N^2 chart / Design Structure Matrix [35]



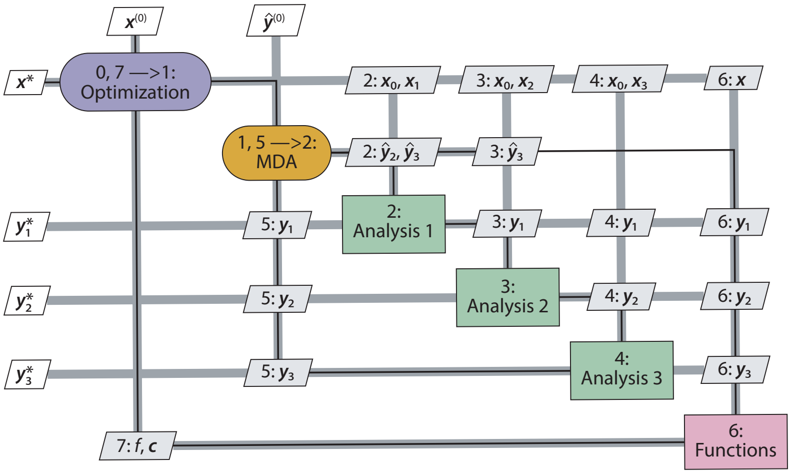
$$\min \quad f(\mathbf{x}) = 400\,x_1 + 20\,x_2 + 130\,x_3^2$$

subject to:

$$g_1(\mathbf{x}) = 190\,x_1^2 - 43.6 + 14.9\,x_4 - 1.44\,x_4^2 \leq 0$$
$$g_2(\mathbf{x}) = 38\,x_2^2 - 183.3 + 36\,x_4 - 2.67\,x_4^2 \leq 0$$
$$g_3(\mathbf{x}) = 650\,x_3^2 - 244 + 45.9\,x_4 - 3.29\,x_4^2 \leq 0$$
$$g_4(\mathbf{x}) = 3.5 - x_4 \leq 0$$
$$g_5(\mathbf{x}) = x_4 - 6.5 \leq 0$$

	x_1	x_2	x_3	x_4
f_1	▨			
f_2		▨		
f_3			▨	
g_1	▨			▨
g_2		▨		▨
g_3			▨	▨
g_4				▨
g_5				▨

b) Functional Dependency Table [36]



c) XDSM [37]

Figure 2.5: Examples of different representations of multidisciplinary systems.

2.2.2. MACHINE-INTERPRETABLE

In the background, the PIDO platforms represent the workflows in a machine-interpretable format, often based on XML or proprietary standards. Object-oriented frameworks for MDAO have been described in scientific literature that create machine-interpretable representations of given computational systems by constructing programming objects [20, 22, 23]. The packages then include methods to execute the MDAO system the best way possible (e.g. using different optimization strategies or taking benefit from parallel computing methods).

In other work, machine-interpretable representations of MDAO systems are created by defining a language. Examples of these linguistic representations are the χ [40], Ψ [19] and REMS [24, 25] languages. These languages allow an integrator to compose the MDAO system in a relatively straightforward way (once familiar with the syntax), so that algorithms built upon the language can perform system manipulations, like problem decomposition and coordination. A radically different representation of MDAO systems was investigated by Hoogreef, who makes use of ontologies to model and store MDAO systems in his InFoRMA platform [26]. By representing the MDAO system as a 'meaningful' (semantic) web of data, semantic reasoning engines [41] are used in InFoRMA to assess and manipulate the system. Although representing the MDAO system as a semantic web initially had potential and enabled the use of reasoning engines, Hoogreef also found that "their modeling becomes too complex for actual implementation". Therefore, the use of semantic web technologies was not further investigated in this work.

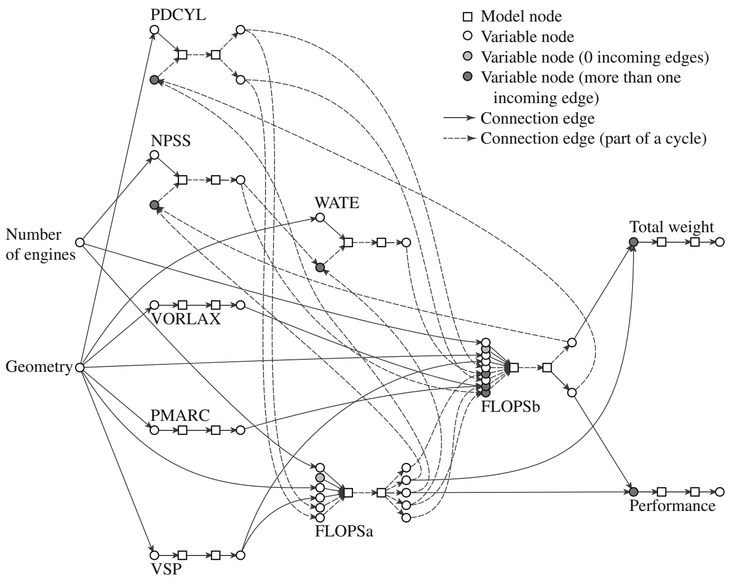
Pate et al. realized that many of the MDAO system representations can be brought back to the basic mathematical construct of (directed) graphs [15]. This is true for some languages such as REMS and InFoRMA's ontologies, but also for purely graphical representations like the N^2 chart and the DSM. Based on this realization, Pate et al. defined a graph-based syntax to represent and manipulate MDAO systems. Three types of directed graphs are defined in Pate et al.'s graph syntax that map one-on-one to the three stages of the MDAO development process illustrated in Fig. 1.3:

Maximal Connectivity Graph (MCG): represents the tool repository containing all available design and analysis tools and how they are interconnected through shared or coupled variables, see Fig. 2.6a.

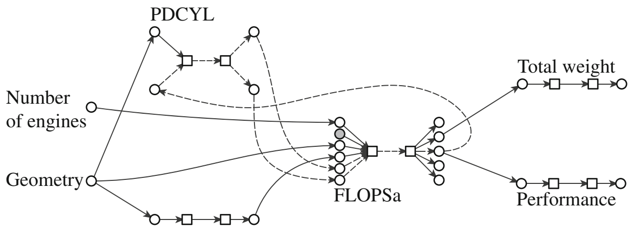
Fundamental Problem Graph (FPG): represents the MDAO problem as a subgraph of the MCG containing a subset of the available tools and marked design variables, constraints, and objective, see Fig. 2.6b.

Problem Solution Graph (PSG): represents the MDAO solution strategy for the aforementioned MDAO problem by integrating driver components (i.e. optimizers and convergers) and including data and process connections (similar to the XDSM notation discussed earlier), see Fig. 2.6c.

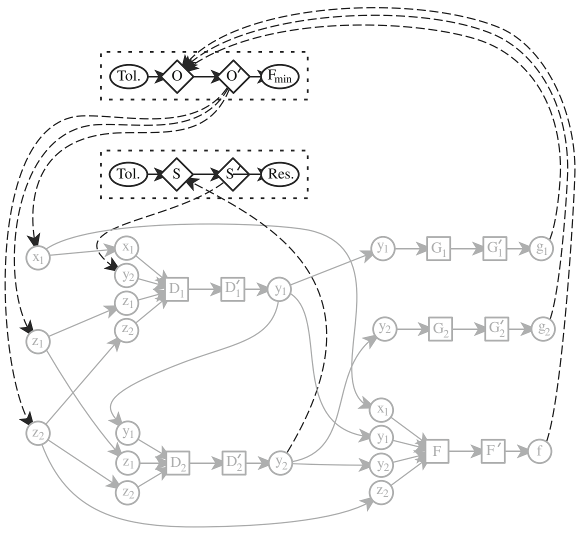
These graph definitions and their manipulation algorithms are mainly focused on the challenge of finding possible combinations of design and analysis tools to solve a given MDAO problem. For example, the FPG shown in Fig. 2.6b is the graph with the least number of cycles. Alternatively, an FPG could also be constructed by ranking tools based on its properties (e.g. accuracy, runtime) or the design team's preferences. Thus their main objective is the automatic determination of the FPG, starting from the larger MCG. In addition, the graph-based formalization enables a classification of the variable and function nodes. This classification can be used to check the graph (e.g. Do all functions have outputs? Are all variables written by only one function?), fix issues where necessary,



a) MCG (Tool repository)



b) FPG (MDAO problem)



c) PSG (MDAO solution strategy)

Figure 2.6: The three different graphs defined by Pate et al. [15]

and automatically identify variables that play a special role, such as design variables, constraints, and objective.

In summary, a large variety of system representations can be found in literature. The XDSM notation has become a de-facto standard within the MDAO community to create human-readable overviews for communication and discussion within the design team. Graph-based representations in different fashions (e.g. semantic webs, directed graphs) are adopted to represent systems in machine-readable form for analysis and manipulation.

2.3. Manipulation

All the machine-interpretable representations discussed in the previous section are set up to enable various forms of computerized manipulation of the MDAO system. These manipulations are generally related to the transformation of the system across subsequent stages in the formulation phase (Fig. 1.3) and aim at automating (part of) the work required to advance from one stage to the other. The main manipulations performed at each link are depicted in Fig. 2.7.

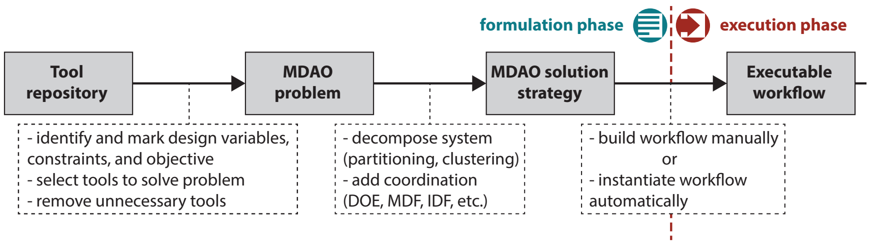

formulation phase
execution phase
Tool repository
MDAO problem
MDAO solution strategy
Executable workflow
- identify and mark design variables, constraints, and objective
- select tools to solve problem
- remove unnecessary tools
- decompose system (partitioning, clustering)
- add coordination (DOE, MDF, IDF, etc.)
- build workflow manually
or
- instantiate workflow automatically

Figure 2.7: The main manipulations performed on the MDAO system to advance stages in the MDAO development process

The first transformation is from tool repository to MDAO problem. In general, automation in the first link involves the creation of methods to 1) identify potential design variables, objective values and constraints, and mark them as such, 2) find valid combinations of tools to analyze the problem and, finally, 3) remove unnecessary functions (tools) and variables. The work of Pate et al. [15] addressed in §2.2 focuses specifically on this area. They offer a graph syntax and suggest algorithms to determine different possible MDAO problems (FPG) based on a tool repository definition (MCG). Also in the REMS language, the necessary manipulations to achieve the MDAO system formulation are based on the full graph of available functions and coupled variables.

The second transformation, from MDAO problem to MDAO solution strategy, is more complex and handled in many different ways in earlier work. In this link the MDAO problem has to be decomposed to make it computationally tractable. This is generally achieved by means of partitioning and clustering methods. These methods aim at grouping tools in smaller sets that belong together. This grouping can be intended to reduce the number of couplings that need to be converged between sets of tools or to enable efficient parallel computing. The Ψ language is an example of a manipulation language specifically developed for system decomposition. Other system decomposition methods are available in literature, based on DSM [42–44] and FDT [45, 46].

After decomposition, an MDAO architecture has to be imposed on the MDAO problem to coordinate its execution. Different MDAO architectures exist in literature, each one providing a different recipe to coordinate the computational system. Martins and Lambe [14] have provided an extensive review of the most commonly used, including both monolithic (e.g. MDF, IDF), and distributed types (e.g. CO (Collaborative Optimization) [47], BLISS (Bi-Level Integrated System Synthesis)-2000 [48], CSSO (Concurrent Subspace Optimization) [49]). MDAO architectures will be more elaborately discussed in §2.4.

Some authors have taken advantage of graph-based formulations to support the creation of alternatively organized MDAO solution strategies. For example, Lu and Martins [50] applied graph partitioning methods on directed graphs to automatically coordinate large MDAO problems using a hybrid MDF-IDF architecture. Thereby tightly integrating the decomposition and coordination. Another example is provided by Gray et al. [51], who took advantage of graph-based methods in OpenMDAO to automatically evaluate multidisciplinary derivatives.

In the third transformation – from MDAO solution strategy to executable workflow – two approaches are generally used to build the executable workflow: manual creation or automatic instantiation. Manually created workflows can be built in a variety of PIDO platforms, which will be discussed more elaborately in §2.5. These workflows are generated from scratch, in the sense that the user is not offered any template to integrate a certain MDAO system according to an MDAO architecture of choice. As a consequence, there is generally a gap between the execution and formulation phase. In collaborative MDAO projects, manual workflow creation is the standard, leading to a process largely based on trial-and-error to establish the executable workflow, thereby relying on the experience of the PIDO platform operator. Large, complex MDAO systems require the coupling of hundreds of variables, a non-trivial and error-prone task to perform manually, especially when a complex solution strategy, such as one based on a distributed architecture, has to be implemented.

Concerning automatic instantiation, the fact that the same MDAO problem can be solved using different solution strategies was one of the reasons to develop the πMDO [20], OpenMDAO (V0) [22] [S6], GEMS (Generic Engine for MDO Scenarios)[17] [S12] and InFoRMA [26] platforms, so that different architectures could be tested quickly on the same problem by automatically (re)configuring the executable workflow. These platforms either tightly integrate the executable workflow and the formulation of the system, as is the case for πMDO, OpenMDAO, and GEMS, or include methods to establish an automated transformation to external PIDO platforms. The latter is the case of InFoRMA, where a built-in mechanism parsing the system knowledge base creates the matching executable workflow in Optimus. Similarly, Ψ provides an export module to enable the creation of executable workflows in MATLAB [S13].

Both approaches to create executable workflows are limited in their capability to support collaborative MDAO projects where system formulation is a team effort and workflows need to be executed in a distributed server environment. A framework like OpenMDAO offers excellent possibilities to set up efficient single-user executable MDAO workflows, but is limited in its capability to support collaborative MDAO system formulation and execute distributed workflows. Some PIDO platforms available on the market allow the integration of multilevel and distributed MDAO workflows through manual “drag&drop” manipulations via their GUI, but, at date, none provides actual formulation capabilities

[52, 53] or the automatic instantiation of large, complex MDAO solution strategies (i.e. no commercial PIDO platform offers options to build an executable workflows based on different MDAO architectures, such as MDF, IDF, and BLISS).

In conclusion, a diverse set of methods and platforms is available to manipulate MDAO systems and transform them from one stage to the other in the MDAO development process. However, the state of the art lacks a comprehensive platform to support the entire formulation of collaborative MDAO systems in a large, heterogeneous design team, where the formulated MDAO solution strategy is also linked to a PIDO platform of choice.

2.4. COORDINATION

The creation of the executable workflows generally follow a certain recipe. Especially for Multidisciplinary Design Optimization (MDO), many different strategies can be applied to solve the same optimization problem. Such recipes are called MDAO architectures. Martins and Lambe [14] created a comprehensive overview and categorization of architectures for MDO. They state the standard multidisciplinary optimization problem as follows [14]:

$$\begin{aligned} \text{minimize} \quad & f_0(\boldsymbol{x}, \boldsymbol{y}) + \sum_{i=1}^{N} f_i(\boldsymbol{x}_0, \boldsymbol{x}_i, \boldsymbol{y}_i) & \\ \text{with respect to} \quad & \boldsymbol{x}, \hat{\boldsymbol{y}}, \boldsymbol{y}, \bar{\boldsymbol{y}} & \\ \text{subject to} \quad & \boldsymbol{c}_0(\boldsymbol{x}, \boldsymbol{y}) \geq 0 & \\ & \boldsymbol{c}_i(\boldsymbol{x}_0, \boldsymbol{x}_i, \boldsymbol{y}_i) \geq 0 & \text{for } i = i, \dots, N \\ & \boldsymbol{c}_i^c = \hat{\boldsymbol{y}}_i - \boldsymbol{y}_i = 0 & \text{for } i = i, \dots, N \\ & \boldsymbol{R}_i(\boldsymbol{x}_0, \boldsymbol{x}_i, \hat{\boldsymbol{y}}_{j \neq i}, \bar{\boldsymbol{y}}_i, \boldsymbol{y}_i) = 0 & \text{for } i = i, \dots, N \end{aligned} \tag{2.1}$$

in which f denotes objective, $\boldsymbol{x}$ design variables, $\bar{\boldsymbol{y}}$ state variables, $\boldsymbol{y}$ coupling variables, $\hat{\boldsymbol{y}}$ variable copies, $\boldsymbol{c}$ and $\boldsymbol{c}^c$ denote design and consistency constraints, $\boldsymbol{R}$ analysis constraints (e.g. residuals), and the subscripts 0 and i refer to shared and disciplinary data respectively.

2.4.1. MONOLITHIC ARCHITECTURES

If a single optimizer is employed to solve this problem, four different architectures could be implemented. The relation between these monolithic architectures is shown in Fig. 2.8. In this figure, the arrows indicate how three of the architectures can be derived from the one in the top left:

All-At-Once (AAO) This architecture describes the most exhaustive approach for solving any MDAO problem, but is merely meant as a theoretical scheme from which the other three practical architectures are derived. With AAO, the optimization would be set up using all elements of the standard problem statement in (2.1).

Simultaneous Analysis aNd Design (SAND) This architecture is derived from AAO by removing the consistency constraints $\boldsymbol{c}^c$ (and thereby also the variable copies $\hat{\boldsymbol{y}}$ that they require). This simplification is allowed as the consistency constraints are implicitly satisfied by meeting the analysis constraints $\boldsymbol{R} = 0$ based on the coupling and state variables ($\boldsymbol{y}$ and $\bar{\boldsymbol{y}}$) provided by the optimizer. With SAND, the optimizer is both designing the system and analyzing it, where the analysis component is

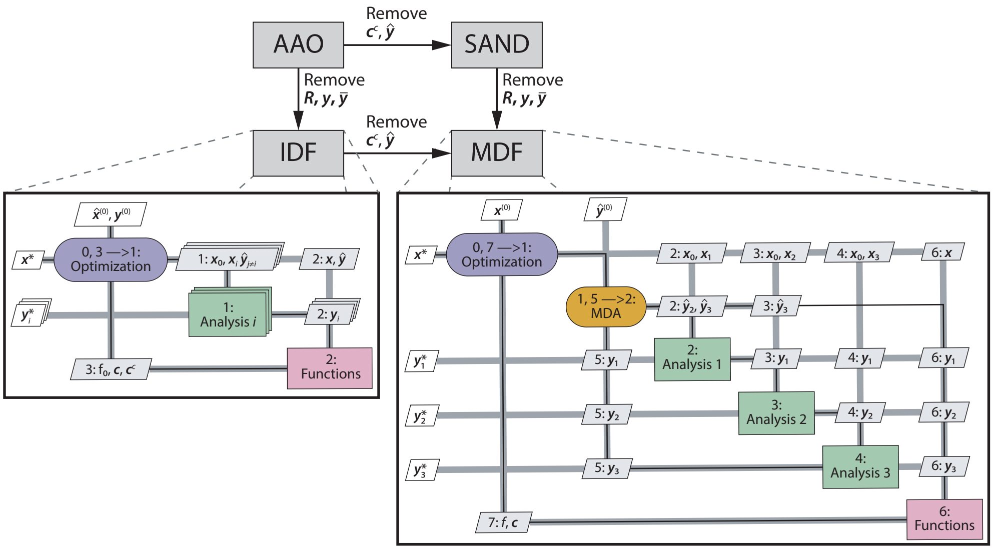


Figure 2.8: The relation between monolithic architectures with respect to the standard optimization problem definition (top). XDSM representation for the IDF (bottom left) and MDF (bottom right) architectures [14].

represented by the requirement to satisfy $\boldsymbol{R} = 0$. These analysis constraints also represent the major drawback of this architecture, as it means that all disciplinary tools have to provide their residuals. In practical engineering design, most tools will internally calculate and satisfy these residuals (e.g. an aerodynamic or structural solver), and act as black boxes in the multidisciplinary system. This handicap of SAND is addressed by the remaining two monolithic architectures.

IDF This architecture is derived from AAO by eliminating the analysis constraints $\boldsymbol{R} = 0$, see Fig. 2.8 (bottom left) for the XDSM. The consistency constraints are kept and the variable copies $\hat{\boldsymbol{y}}$ are added as design variables to the optimizer. Since estimated values of the couplings are available in the form of extra design variables, the disciplinary analyses can be parallelized. A handicap of IDF is that the design will only be feasible when all the consistency constraints are satisfied, which is usually only true when the optimization is finished successfully.

MDF As depicted in Fig. 2.8, MDF can be derived either from SAND or IDF. The key addition in this scheme is the Multidisciplinary Design Analysis (MDA) loop inside the optimizer. The MDA converges each design option that is provided by the optimizer based on the coupling variables, thereby eliminating the analysis constraints $\boldsymbol{R} = 0$ from the SAND point of view and the consistency constraints $\boldsymbol{c}^c = 0$ with respect to IDF. The strong point of MDF is that a feasible design is available for each iteration provided by the optimizer, but this comes at the additional cost of performing an MDA each optimization iteration to converge the system.

Of the four monolithic architectures, the IDF and MDF schemes (or hybrid combinations) are the only two that are useful in the practical engineering design environment, given the black-box behavior of most disciplinary analyses if existing software packages are used.

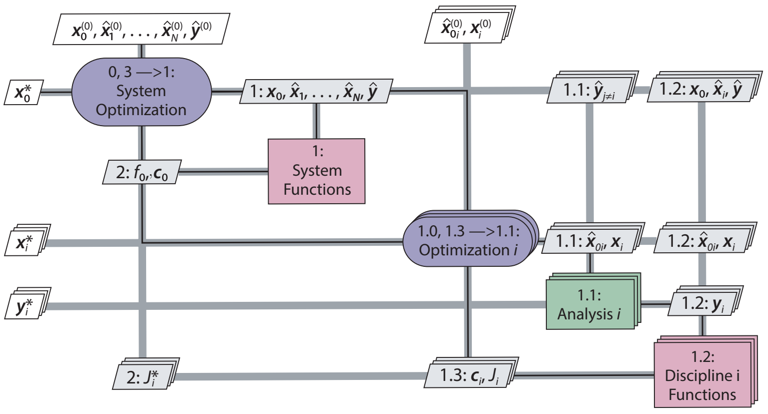
Many other architectures have been devised to handle MDO problems that, in opposition to monolithic schemes, distribute the optimization task over multiple subsystems. Conceptually, this distribution is motivated by the typical arrangement of the disciplinary specialists in departments at industrial organizations. Distributing the optimization allows each department to retain ownership of their field and work on their analysis in isolation, while at the same time the system-level optimization is performed more efficiently by accounting for specific characteristics of the disciplinary analysis (e.g. there might be a large difference in the execution times per discipline). The wide range of distributed architectures is elaborately discussed by Martins and Lambe [14] and summarized by Hoogreef [26]. Here, two architectures will be discussed in more detail as they will play a role later in this dissertation: Collaborative Optimization (CO) and BLISS-2000.

2.4.2. Distributed architecture: Collaborative Optimization

The CO architecture, which was formulated by Braun [47], is depicted in Fig. 2.9. Note that the CO$_2$ variation is used here, as it is the most widely used version of the architecture. With CO, the system is distributed based on shared and disciplinary variables. Each discipline gets to handle its disciplinary design variables and constraints with its own optimizer (step 1.0 in Fig. 2.9). Shared design variables are also added at this level as copies. At the system level, the optimizer is only concerned with system functions, such as shared constraints and the system objective. In addition, copies of the disciplinary design variables and coupling variable copies are added at the top-level. The following key function connects the system- and disciplinary-level optimizers and ensures consistency:

$$J_i = \|\hat{\boldsymbol{x}}_{0i} - \boldsymbol{x}_0\|_2^2 + \|\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i\|_2^2 + \|\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i(\hat{\boldsymbol{x}}_{0i}, \boldsymbol{x}_i, \hat{\boldsymbol{y}}_{j\neq i})\|_2^2 \quad \text{for } i = 1, \ldots, N \tag{2.2}$$

This function is used as objective (step 1.3 in Fig. 2.9) for every disciplinary optimization,



$x_0^{(0)}, \hat{x}_1^{(0)}, \ldots, \hat{x}_N^{(0)}, \hat{y}^{(0)}$
$\hat{x}_{0i}^{(0)}, x_i^{(0)}$
x_0^*
0, 3 —>1: System Optimization
1: $x_0, \hat{x}_1, \ldots, \hat{x}_N, \hat{y}$
1.1: $\hat{y}_{j\neq i}$
1.2: $x_0, \hat{x}_i, \hat{y}$
2: f_0, c_0
1: System Functions
x_i^*
1.0, 1.3 —>1.1: Optimization i
1.1: $\hat{x}_{0i}, x_i$
1.2: $\hat{x}_{0i}, x_i$
y_i^*
1.1: Analysis i
1.2: y_i
2: J_i^*
1.3: c_i, J_i
1.2: Discipline i Functions

Figure 2.9: CO architecture [14]

while for the system optimization the same value acts as equality constraint. Hence, the disciplinary optimizations aim at minimizing system inconsistency, while the system optimization is actually minimizing the design objective.

CO is an advantageous architectures if the MDAO system mainly contains disciplinary design variables and constraints and a small amount of shared data, as the disciplinary data is handled at their own level. The main advantage of the architecture is the distribution of the disciplinary analyses in fully separated suboptimizations, potentially matching the separation of departments in the organizational structure. Unfortunately, CO is also known to have poor computational performance in practice.

2.4.3. Distributed architecture: BLISS-2000

The BLISS-2000 architecture [54] is a radically different successor of the initial BLISS formulation [48], both devised by Sobieski. The scheme is depicted in Fig. 2.10 and contains three main loops:

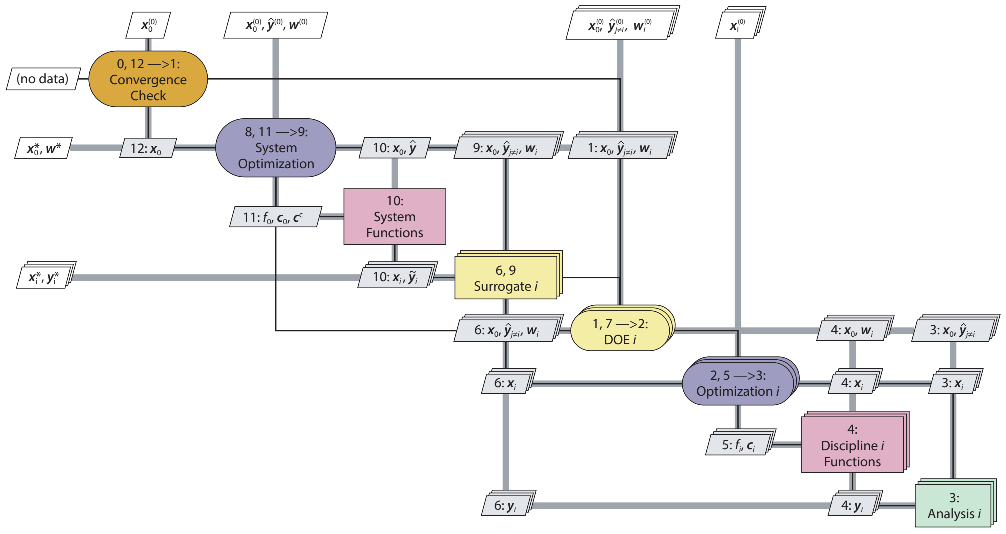

$x_0^{(0)}$
$x_0^{(0)}, \hat{y}^{(0)}, w^{(0)}$
$x_0^{(0)}, \hat{y}_{j\neq i}^{(0)}, w_i^{(0)}$
$x_i^{(0)}$
(no data)
0, 12 —>1: Convergence Check
x_0^*, w^*
12: x_0
8, 11 —>9: System Optimization
10: $x_0, \hat{y}$
9: $x_0, \hat{y}_{j\neq i}, w_i$
1: $x_0, \hat{y}_{j\neq i}, w_i$
11: f_0, c_0, c^c
10: System Functions
x_i^*, y_i^*
10: $x_i, \tilde{y}_i$
6, 9 Surrogate *i*
6: $x_0, \hat{y}_{j\neq i}, w_i$
1, 7 —>2: DOE *i*
4: x_0, w_i
3: $x_0, \hat{y}_{j\neq i}$
6: x_i
2, 5 —>3: Optimization *i*
4: x_i
3: x_i
5: f_i, c_i
4: Discipline *i* Functions
6: y_i
4: y_i
3: Analysis *i*

Figure 2.10: BLISS-2000 architecture. [14]

1. At the system level (steps 8-11 in Fig. 2.10), the optimization is performed based on system functions and surrogate models of optimal disciplinary design options.
2. In order to run this quick surrogate-based optimization at the system level, the data to build the surrogate models need to be gathered first. Therefore, a DOE is performed first (steps 1-7 in Fig. 2.10) for each discipline, in which the disciplinary analyses are optimized with respect to the disciplinary design variables and constraints.
3. The third component of the BLISS-2000 scheme is the convergence check (steps 0-12 in Fig. 2.10). This check is required, because the use of surrogate (approximation) models at the system level introduces an error between the found and actual optimum. The magnitude of this error depends on the precision of the surrogate models. Therefore, after each system optimization the design space used for the DOE (for which the surrogate models are built) is shrunk to reduce the approximation error at the system level. This process is repeated until the found optimum converges. The design space aspect of the scheme is not clearly indicated in the XDSM in Fig. 2.10: it could be added by introducing a data dependency between the convergence check and the system optimization and DOE elements, with the convergence check providing new bounds for the system-level design variables.

The main element still requiring definition for the BLISS-2000 scheme, is the link between the disciplinary and system level. Two key elements are introduced in the scheme for this purpose. The first element is the introduction of a weight coefficient ($\boldsymbol{w}$ in Fig. 2.10) for each coupling. These coefficients control the priority of different state variables in each disciplinary optimization. The second element uses these weight coefficients in the objective function for each disciplinary analysis (step 5 in Fig. 2.10):

$$f_i = \boldsymbol{w}_i^T \boldsymbol{y}_i(\boldsymbol{x}_0, \boldsymbol{x}_i, \hat{\boldsymbol{y}}_{j\neq i}) \tag{2.3}$$

The weight coefficients are added as design variables in the system optimization. Similar to IDF, constraints are also added at the system level to ensure consistency between the coupling variables from the optimizer and their approximated values coming from the surrogates.

BLISS-2000 is a scheme that can be matched well with a departmentalized organizational structure: each department can provide their own optimization results for the given design space, or even directly provide their own surrogate model. Alternatively, a collaborative process involving an OEM and its Tier 1 and Tier 2 suppliers could also be defined using BLISS-2000. In addition, the scheme allows a high degree of parallelization. At the disciplinary level, the different DOEs can be performed in parallel and each DOE could also perform multiple experiments in parallel. At the system level, the IDF-like separation of the surrogate-based disciplines also supports parallel execution.

2.5. Execution

Once the preferred solution strategy has been determined, the next step is to create a workflow that can actually execute the blueprint from the formulation phase (see Fig. 1.3). A multidisciplinary workflow can get very large and complex, so that its manual integration by means of some general-purpose language (i.e. Python, MATLAB) can quickly become an impractical approach. For this reason, several PIDO platforms (commercial and open-source) have been developed to support the creation and execution of these workflows, each with their own methods with respect to the following characteristics:

User interaction How the integrator is supposed to build the workflows.

Workflow concept What is the underlying concept for workflow creation. How disciplinary components are combined with special elements, such as optimizers and DOEs.

Component integration How well the platform supports the integration of heterogeneous components built in different software packages (e.g. a workflow containing a Python-based aerodynamic solver, Abaqus for finite-element analysis and a MATLAB-based mission analysis).

Distributed execution The execution of the executable workflow on a distributed server environment (e.g. each disciplinary specialist makes their analysis available on their own server).

Derivatives The support for incorporating partial derivative calculation and their automatic combination into full derivatives required for a gradient-based optimizer.

Convergence The methods available for converging circular dependencies in the MDAO system with solvers.

The overview here is limited to those platforms that play a major role in MDAO-related work. A large variety of platforms is available in this subset, ranging from open-source to commercial solutions, either script- or GUI-based. Their main characteristics are summarized in Tab. 2.1.

Table 2.1: Key characteristics of PIDO platforms used in MDAO projects

platform	ref.	open-source?	inter-action	workflow concept*	comp. integr.	distr. exec.	deriva-tives	conver-gence**
RCE	[S3]	Y	GUI	AC	+/-	++	-	FPI
Optimus	[S2]	N	GUI	C+M	++	+	+/-	FPI
Isight	[S14]	N	GUI	AC	++	++	-	FPI
ModelCenter	[S1]	N	GUI	AC	++	++	-	FPI
HEEDS	[S15]	N	GUI	C+M	++	++	-	FPI
OpenMDAO	[S6]	Y	scripts	OOC+M	-	-	++	FPI,NFPI
GEMS	[S12]	N	scripts	OOC+M	+	+	+	FPI,NFPI

* AC: all components, C+M: Components + methods, OOC+M: object-oriented components + methods
** FPI: Fixed-point iteration, NFPI: Non-fixed-point iteration

2.5.1. RCE

Initially developed for a shipbuilding project, the DLR has created its own open-source PIDO platform for aerospace research and development: Remote Component Environment (RCE). [55, 56] [S3] The platform creation was motivated by a need to better meet organizational requirements with respect to distributed workflow execution and scientific data management. The former is enabled in RCE by supporting teams in configuring their own server network. Disciplinary specialists can publish their tools on this network and thereby make them available for execution from their own server as part of a larger distributed workflow running at another location. The latter is supported specifically for aircraft design with components and options to easily integrate and handle CPACS-compatible tools (see §2.1 for more information on CPACS). Within DLR, this has led to a large network of aircraft design tools that can be easily connected in project-specific design workflows.

RCE has been used in a range of aircraft MDAO projects, such as SpaceLiner [57], FrEACs [30, 32], DigitalX [58], and IDEaliSM [59]. In RCE, workflows are created through the GUI and all aspects of the workflow are added as separate components, including elements like optimizers, convergers, and DOEs. These elements are typically integrated from other open-source packages. For example, design analysis methods (i.e. convergers, DOEs, optimizers) from the Dakota package [60] are incorporated in RCE and available as separate components. There is no specific support in RCE to combine or handle partial derivatives for gradient-based optimization, though it is possible to include the full derivatives as input to the optimization components.

2.5.2. Optimus

Optimus [S2] is a commercial package developed by Noesis Solutions. A GUI is used to let users create workflows (although a Python interface is also available to offer script-based workflow creation to advanced users) and the package includes a large variety of component types to easily integrate different disciplinary analyses developed on other platforms, such as MATLAB, Abaqus, Excel, in a single simulation workflow. The workflow set-up is based on a separation between this simulation workflow, containing the disciplinary analyses that provide the values of interest, and 'design methods', such as an optimization or DOE. Thanks to this separation the same simulation workflow can be used to impose different design methods, for example to first simply analyze a design at a couple of locations in the design space before performing a full-scale optimization. Optimus supports the addition of derivatives as outputs of individual disciplines and is able to combine these analytic derivatives with finite-difference values to calculate the gradients required by the optimizer. However, the derivatives support is quite primitive, which also indicates that the use of derivatives is not standard practice in the industrial context (including automotive and aerospace) where Optimus is used.

As mentioned in §2.3, the Optimus platform was used by Hoogreef [26] to automatically instantiate executable MDAO workflows from the ontology-based InFoRMA platform. Augustinus [61] describes the inner workings of this automated workflow generation based on the InFoRMA formulation. On the Optimus side, the automatic instantiation is supported by the Python-Application Programming Interface (API). With respect to collaborative MDAO projects, Optimus has been used to create workflows in the IDEaliSM project. Another application of Optimus in the MDAO context is described by Liu et al. [62].

2.5.3. Isight

Isight [S14] is the commercial PIDO platform of Dassault Systèmes. Similar to Optimus, Isight supports workflow creation through the GUI and a large set of standardized components is available to integrate disciplinary tools. Contrary to Optimus, Isight's workflow concept is fully component-based, meaning that the design method used has to be added explicitly to the workflow (as with RCE). Recently, Isight has been combined with the Simulia Execution Engine (SEE) to support execution across distributed compute resources. Isight does not support derivatives for gradient-based optimization. The platform is used in numerous MDAO studies and projects [63–65]. Most notably, Isight is the PIDO platform of choice for Bombardier's framework discussed in §2.6.

2.5.4. ModelCenter Integrate

Phoenix Integration developed the commercial ModelCenter Integrate [S1] platform. In its setup this platform is nearly identical to Isight: a GUI to create fully component-based workflows. ModelCenter is used in a number of MDAO studies by organizations like NASA [66], Lockheed Martin [67], and Boeing [68]. The recently developed GEMS framework [17] (which will be discussed in §2.6) also employs this platform within the broader framework.

2.5.5. HEEDS

HEEDS [S15] is a commercial PIDO platform developed by Siemens. Its workflow concept is similar to that of Optimus: a combination of components to create simulation workflows on which methods for design space exploration and optimization can be executed. The HEEDS platform is a popular choice in a range of industries (e.g. aerospace & defense, automotive, marine, and oil & gas). Strong points of HEEDS are the support to set up cluster and cloud computing and the wide range of options for post-processing and visualization.

2.5.6. OpenMDAO

OpenMDAO [S6] is an open-source platform developed by NASA.[22, 23, 51] Its development was motivated by the need to implement more efficient MDAO techniques, with respect to contemporary PIDO platforms, to enable the solution of large-scale and more complex optimization problems. Two main techniques lacking in other PIDO platforms are implemented in OpenMDAO: advanced algorithms for the solution of coupled systems and methods for derivative computation. With respect to solving the coupled system, other PIDO platforms treat every component as an explicit function evaluation, while OpenMDAO provides support for including tools as implicit functions as well, thereby opening up a range of possibilities for solving systems using different techniques (e.g. Newton, Krylov) than the fixed-point convergers (e.g. Gauss-Seidel, Jacobi) generally implemented. The support for derivative computation is also very limited in other platforms, whereas it is a key feature of OpenMDAO. In the latest versions of OpenMDAO (V1 and V2), derivative computation is based on the Modular Analysis and Unified Derivatives (MAUD) architecture [69]. Thanks to this architecture it is possible to build an executable workflow containing both a variety of components types, such as explicit and implicit functions, and a variety of methods to compute the component's derivatives, such as finite-difference, complex step or analytic [70]. OpenMDAO is a Python-based platform and requires its users to build their executable workflow using scripts. The integration of components built in different software environments is the responsibility of the user, who has to make sure this component can be executed as a Python-based OpenMDAO component.

The potential of OpenMDAO comes forward in the wide range of applications in which the platform has been used to solve engineering design optimization problems: Hwang et al. [71] used OpenMDAO to design a small-scale satellite involving over 25,000 design variables and 2.2 million state variables, Chung et al. [72] performed structural topology optimizations, and Gray and Martins [73] solved a coupled aeropropulsive design optimization problem with OpenMDAO. Many more applications are summarized in [23] showing the unprecedented scale and complexity of MDAO problems that can be solved with OpenMDAO. Despite the potential of the platform, it is currently mainly used in a research-related context, which can be attributed to the lower level of user friendliness (scripts instead of GUI) and the more invasive and cumbersome tool integration required to benefit from the platform's advanced methods. Morever, the availability of derivatives, which is exploited well by OpenMDAO, is uncommon in industrial workflows involving commercial, black-box analysis tools.

2.5.7. GEMS

GEMS (Generic Engine for MDO Scenarios) [S12] is a proprietary Python package developed at the IRT Saint Exupéry research institute by Gallard et al. [17]. The package is developed within the MDA-MDO project (described in more detail in §2.6) and is strongly influenced by OpenMDAO. However, there are key differences between GEMS and OpenMDAO: whereas the former focuses on monolithic architectures and the integration of "stripped" tools to access derivatives and residuals, GEMS also includes multilevel formulations and provides interfacing for integrating off-the-shelf disciplinary analyses. The integration of tools is supported in two ways by GEMS: either a generic wrapper is created for a specific workflow engine or a GEMS base class is used for direct Python integration. The inputs and outputs of tools are described using JSON (JavaScript Object

Notation) schemas.

GEMS is not only a PIDO platform, but is also able to reconfigure an MDAO system with a minimal amount of programming effort. Detailed information on the methods used for formulating and reconfiguring MDAO systems is unfortunately not available in open literature.

2.6. Framework integration

If a team or organization integrates their set of support applications into one coherent structure, an MDAO framework emerges. In this dissertation, the term (MDAO) framework is used to refer to a collection of integrated support applications.* Recently, many frameworks have been developed within large, collaborative projects. Here, the discussion is limited to four recent projects.

2.6.1. FrAECs

DLR's FrEACs (Future Enhanced Aircraft Configurations) project [30, 32] aimed "to quantify uncertainties in the design process for new aircraft and to apply them to the design of two unconventional configurations." The two configurations developed were the strut-braced wing and the blended-wing body. Eleven DLR departments spread across the country were involved in the project providing technologies and analysis capabilities. In this project, the distributed design framework developed over the years was used to perform physics-based analyses at increasing levels of fidelity.

The FrEACs framework for collaborative MDAO is shown in Fig. 2.11. The framework contains three layers. The CPACS ecosystem at the bottom lies at the foundation of the tool repository. The TiXI and TiGL libraries are available to tool specialists to help them make their tool compatible with the common language. These libraries make it easy to read, write, and check XML files (and CPACS files in particular) in different programming languages (TiXI) and also provide standard methods to analyze the contents of a CPACS file (TiGL), for example to get the number of wings specified for a given model or the span of a certain wing. The CPACS-compatible tools are made available for a collaborative workflow by publishing them on the server network through RCE. Thanks to this network environment, an integrator can build a complex workflow combining tools executed all over the country. See

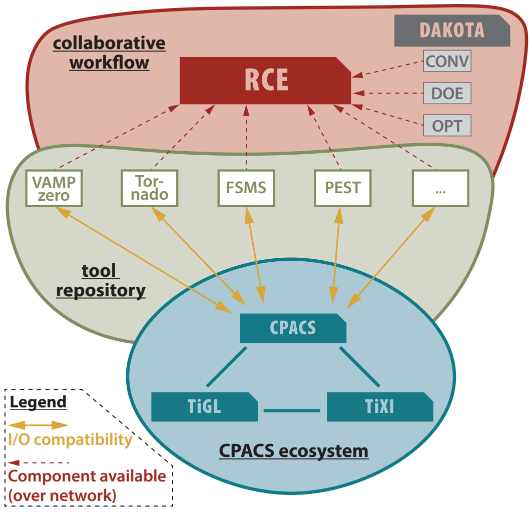


Figure 2.11: FrEACs framework overview (based on [30, 32])

*Other sources use the term framework to describe PIDO platforms such as OpenMDAO and Isight, which were discussed in §2.5.

§2.5 for more information on the RCE platform.

The formulation of the MDAO system has to be performed manually in this framework. Once tools are available on the server, it is not directly clear how its I/Os relate to other tools in the workflow. Therefore, building larger collaborative workflows is a cumbersome trial-and-error process. During the project, interactive workshops with tool specialists were organized to define the system, this definition was then digitized by integrators and further refined through additional group discussions. Visualizing the tool repository, MDAO problem, and solution strategy was done manually. Hence, the FrEACs framework supports the execution of a distributed workflow on an institutional network well, but this workflow had to be built through a manual trial-and-error process. Project lead Erwin Moerland stated after the project that “one of the key lessons learnt was that the setup of the collaborative workflows including leaping over the associated hurdles along the way consumed a large part of the available project time”.*

2.6.2. IDEaliSM

In the IDEaliSM (Integrated & Distributed Engineering Services Framework for MDO) project fourteen European partners worked together on a framework for the automotive and aerospace industry. The aim of the project was “to reduce the time-to-market and development cost of high-tech structures and systems through a change in the product development process by enabling the continuous integration of distributed and highly specialized development teams” [26]. The project delivered a framework and methodology to support the distributed, flexible, and service-oriented development process integrating people, process, and technology [74].

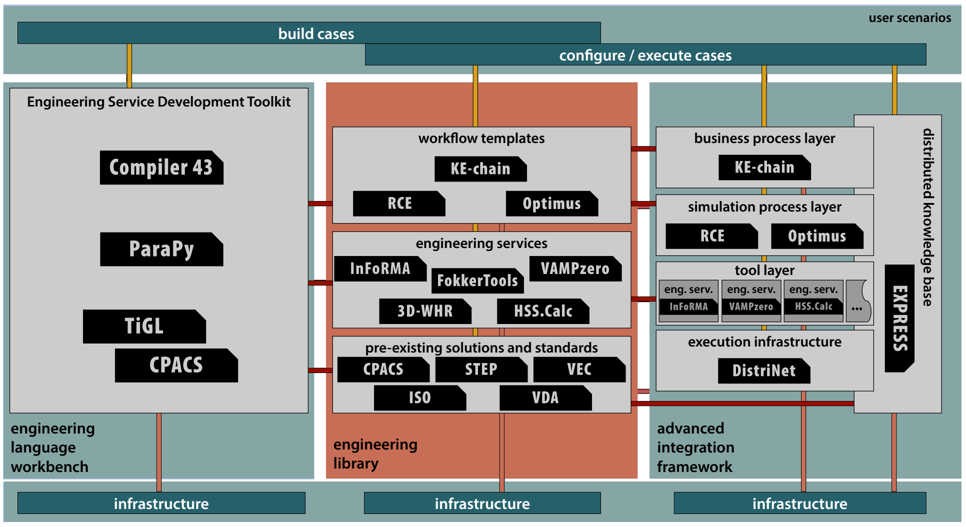


Figure 2.12: IDEaliSM framework overview (based on [75])

The IDEaliSM framework is depicted in Fig. 2.12. The framework contains three main blocks: engineering language workbench, engineering library, and advanced integration framework. Looking at the user scenarios in the top, the first two blocks are used and

*Personal communication with FrAECs project lead Erwin Moerland, April 4, 2019.

developed in "build cases", where a team or a specialist is preparing general workflow templates and engineering services that could be used in "configure / execute cases" to actually run an engineering project. The three main blocks contain the following:

Engineering language workbench This block consists of different development toolkits that can be used to create an engineering service. Three IDEaliSM-native toolkits are shown in the figure, however, these could also be other toolkits that allow the creation of an analysis module, such as MATLAB and Python. Note how the CPACS ecosystem from the FrEACs project is considered one of the possible toolkits here (DLR is one of the project partners). Other toolkits are the knowledge-based engineering [76] platform ParaPy [S16] and the Design Compiler 43 platform [S17].

Engineering library This block connects the build and configure/execute cases. Here, different generic templates, engineering services and data standards are developed (in build cases) or available (for configure/execute cases) for use in MDAO projects. The lowest layers contains data standards that could be adopted in the project. The engineering services subblock contains a range of support applications and analysis modules. InFoRMA (discussed in §2.3) is one of the services to support a team in formulating the MDAO system. Other services are tools used for design analysis and therefore this block can also be considered to be the tool repository. The top subblock contains templates for PIDO platforms (see §2.5) and for the business process management platform KE-chain as well. These templates make it easier to set up and reconfigure a (new) project.

Advanced integration framework The final framework block represents the instantiation of elements from the library to perform an engineering design project. The integration of people is supported in the business process layer, where each user can access a central platform to perform their tasks. The simulation process layer contains the executable workflow that is built from tools in the layer below. At the foundation an execution infrastructure could be used for cloud computing. Data storage at all these layers can be handled by a distributed knowledge base.

Use cases in the IDEaliSM project have shown that it is possible to build and execute collaborative workflows by combining a set of supporting technologies into one framework. In one use case, the InFoRMA platform was used to automatically create the executable Optimus workflow, showing a significant reduction in time required to build large workflows of over 90%. In another use case, the design of a rudder hinge system was performed by establishing a connection between the full aircraft design of the OEM and the detailed rudder design tool at supplier Fokker Aerostructures [77].

2.6.3. MDA-MDO PROJECT

The MDA-MDO project [38] was lead by French research institute IRT Saint Exupéry and includes partners from industry, research institutes, and universities. The project aimed to develop efficient capabilities to enable the deployment of MDAO in industry by increasing the technology readiness level. The framework developed in the project is depicted in Fig. 2.13. The core of the framework is the formulation and PIDO platform GEMS [17] (discussed in §§2.5.7). Through GEMS, a framework is assembled by interfacing a collection of disciplinary analyses with their native workflow engine. The design team can visualize formulated MDAO solution strategies using the XDSMjs package [S10] developed by The French Aerospace Lab (ONERA).

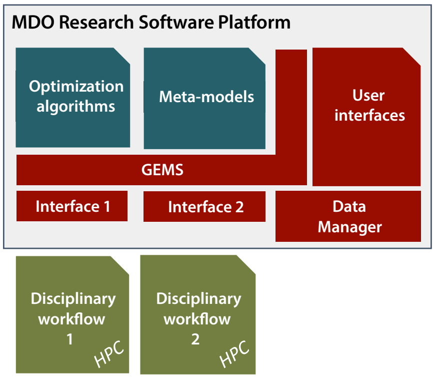


Figure 2.13: Framework of the MDA-MDO project [17]

A demonstrator of the project includes a supersonic business jet design in which four analyses, integrated in different platforms or languages (namely ModelCenter, a proprietary Airbus workflow engine, and Python), are all executed within a single executable workflow through GEMS. A second demonstrator of the project concerns the high-fidelity aerostructural optimization of an engine pylon. A differentiating aspect of the MDA-MDO framework, with respect to the other projects discussed here, is that it is focused on multi-level architectures that match well with the distributed nature of the industrial organization and the capability to integrate black-box off-the-shelf disciplinary tools already available at the company. Except for the visualization package, the majority of the MDA-MDO project targets the infrastructural aspects of executing MDAO system within the industrial context, rather than the collaborative ones.

2.6.4. Bombardier's MDO project

Piperni et al. [11] describe the multilevel MDAO capability developed at Bombardier. The framework is divided in three subframeworks matching the three phases of aircraft design: conceptual, preliminary, and detailed. The first two subframeworks are described in the paper. The conceptual MDAO framework is shown in Fig. 2.14. In this framework the disciplinary analyses are of low fidelity, either empirical-based for conventional configurations or physics-based for unconventional configurations. The configuration database at the center of the framework can be either purely based on numbers through an Excel [S4] sheet or an interactive CAD-based application built in CATIA [S18] can be used. The disciplinary tools are man-

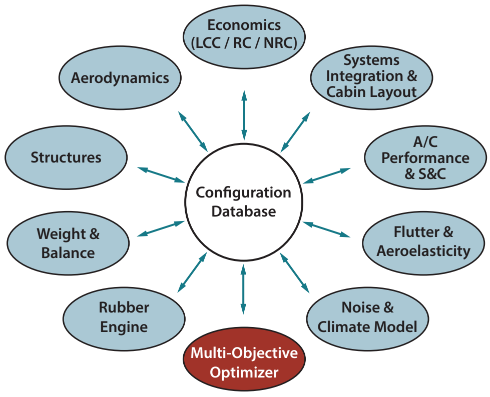


Figure 2.14: Overview of the conceptual MDAO framework at Bombardier [11]

ually integrated in Isight (discussed in §§2.5.3) and the multi-objective optimizer is taken from the PIDO platform. The conceptual design framework is used for broad and thorough design space exploration, including the analysis of aircraft-family concepts, mission requirements, and engine architectures.

In the preliminary MDAO framework, major aircraft features and parameters are already fixed and the team focuses on the design of both the detailed aerodynamic shape and preliminary structural sizing of the wing. This framework can be used for single-level or multilevel formulations, where a bi-level setup is shown in Fig. 2.15. This subframework is also integrated using Isight. All subframeworks at Bombardier are assembled manually and specialized for aircraft design. In this respect, the Bombardier framework shows how performing MDAO in an industrial setting requires the cumbersome task of assembling frameworks which, as Piperni et al. [11] note, are still fully uncoupled with respect to the different design phases. The authors indicate a need to couple these subframeworks together, thereby achieving a single reconfigurable MDAO system where the fidelity levels of tools can be changed dynamically to match the design phase.

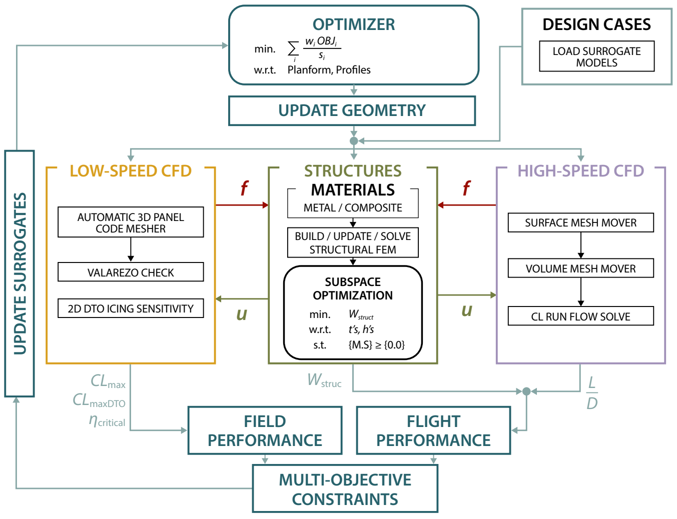


Figure 2.15: Overview of the bi-level preliminary MDAO framework at Bombardier [11]

2.7. Towards a new framework generation

This section sets the stage for the developments presented in the rest of this thesis. First, the categorization of frameworks in different generations is introduced, as this motivated the creation of a new generation, which provided the context for this work in the form of a collaborative MDAO project. Finally, current limitations of the state of the art are summarized to identify the needs for this new framework generation to succeed.

2.7.1. MDAO framework generations

The challenge of applying MDAO methods collaboratively in large, heterogeneous organizations requires multiple dedicated solutions for the range of tasks performed by the engineering team. For example, the MDAO system needs to be composed by establishing a tool repository (§2.1), represented for visual inspection and discussion (§2.2), manipulated to formulate the right problem and solution strategy (§2.3 and §2.4), and executed using a PIDO platform (§2.5). The proper support for each of these tasks generally justifies the development of a stand-alone application, probably written in a programming language or created on a platform best suited for the specific task at hand.

In the previous section (§2.6), four projects were discussed that integrated different applications and standards in one framework, see Figs. 2.11 to 2.15. Ciampa and Nagel [16] (based on [78] and [30]) categorize such frameworks in three generations, which are visualized in Fig. 2.16:

integrated analyses (first) generation (Fig. 2.16a) This framework generation refers to environments that tightly integrate disciplinary capabilities and an optimizer as a monolithic system. All analysis modules are directly available to the design team lead, and the design process is deployed via direct interfaces among the multiple design capabilities.

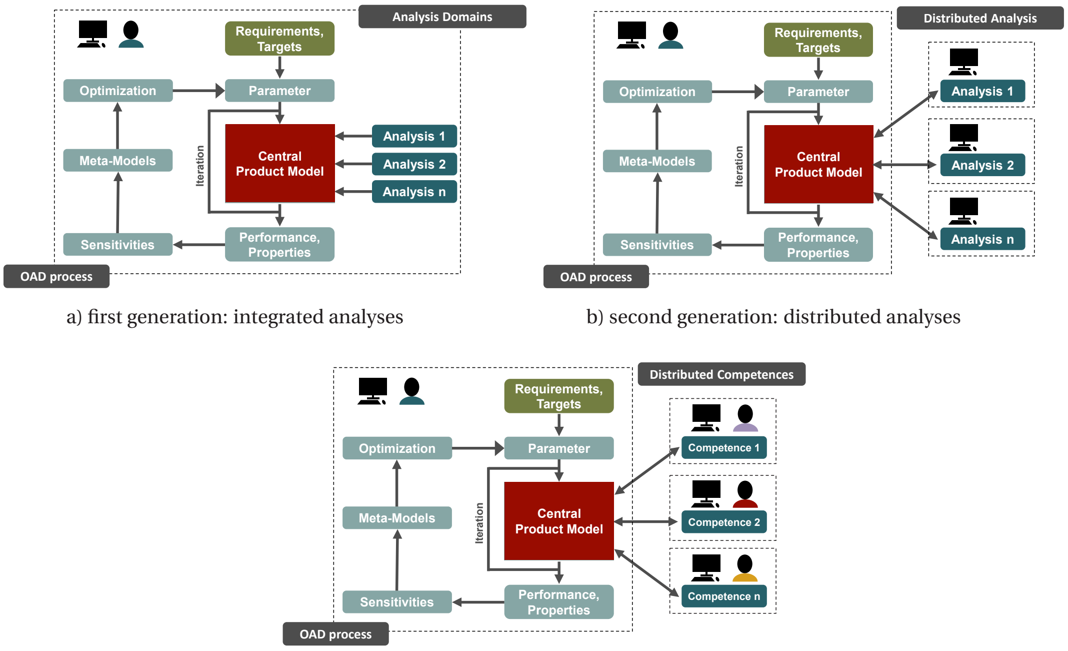
distributed analyses (second) generation (Fig. 2.16b) This generation is characterized by the distribution of the analyses on dedicated computational facilities. Disciplinary specialists are in charge of providing the analyses, but have to hand them over to the process integrator for execution. The analyses are called by a centralized design and optimization process. The team lead assumes the role of process integrator and is both owner and operator of the MDAO system. The analysis modules need to exchange data between themselves and with the centralized optimization components, requiring multiple interfaces to transform the various data formats and an effective data management system to limit data transfer overhead.

distributed design (third) generation (Fig. 2.16c) In this envisioned generation the distribution does not only concern the analysis modules, but rather all tasks involved in assembling and operating the MDAO system. One of the priorities in this framework generation is to support human judgment and collaboration in large, heterogeneous teams, with the aim of exploiting the full available spectrum of expertise offered by the disciplinary specialists. This is done by tackling a number of non-technical barriers for MDAO [1] that ranges from matching the company's organizational structure with the system's setup and execution, handling large data sets for result interpretation, to streamlining communication between parties.

Elements of the envisioned third-generation framework were already present in some developments of the IDEaliSM and FrEACs projects. However, the creation of a comprehensive third-generation framework was the focus of another collaborative MDAO project: AGILE. This dissertation presents key contributions to the AGILE project that were used at the core of the established framework.

2.7.2. AGILE PROJECT DESCRIPTION

The AGILE (Aircraft 3rd Generation MDO for Innovative Collaboration of Heterogeneous Teams of Experts) project was an international collaboration of nineteen partners from the aerospace industry that started in June 2015 and finished in December 2018. The novel framework generation developed in AGILE represents a new *paradigm* for collaborative MDAO: a doctrine based on a set of fundamental theories, assumptions, methodologies, and applications on how collaborative MDAO should be performed. In AGILE [16], performing collaborative MDAO is divided in three phases, see also Fig. 2.17:



a) first generation: integrated analyses

b) second generation: distributed analyses

c) third generation: distributed design

Figure 2.16: Three generations of MDAO frameworks by Ciampa and Nagel [16]

1. setup
2. operation
3. solution

Each phase presents specific challenges and, accordingly, puts different demands on the framework that is used to develop the given MDAO system. The goal of the set-up phase is to gather into a coherent and consistent repository different design competences (e.g. in the form of disciplinary design tools, or pre-assembled workflows of tools), which are often provided by different departments within the same organization, or even by multiple organizations. The first technical challenge here is to 'let the design competences speak to each other', hence to enable the necessary I/O data flow. This is typically a challenging task, even when using design competences that are available in the same design team. Its complexity grows exponentially when the tools to connect are distributed across large and heterogeneous teams, not geographically collocated. This is not only difficult and time-consuming, but intellectual property protection, tool accessibility and security issues can make any technical solution practically unfeasible. This is the *configuration hurdle* that was already identified in Chapter 1.

In the operation phase the MDAO framework is used by the design team to define first the MDAO problem to be solved, then to determine the right strategy to solve it and, finally, to implement such strategy as an executable workflow. This second phase too presents a mix of technical and non-technical challenges. A first main technical challenge, obviously, concerns the ability to formulate a complex MDAO system involving a large number of design competences. The second concerns the integration of the system formulation into an executable computational process (e.g. using a PIDO tool). A third

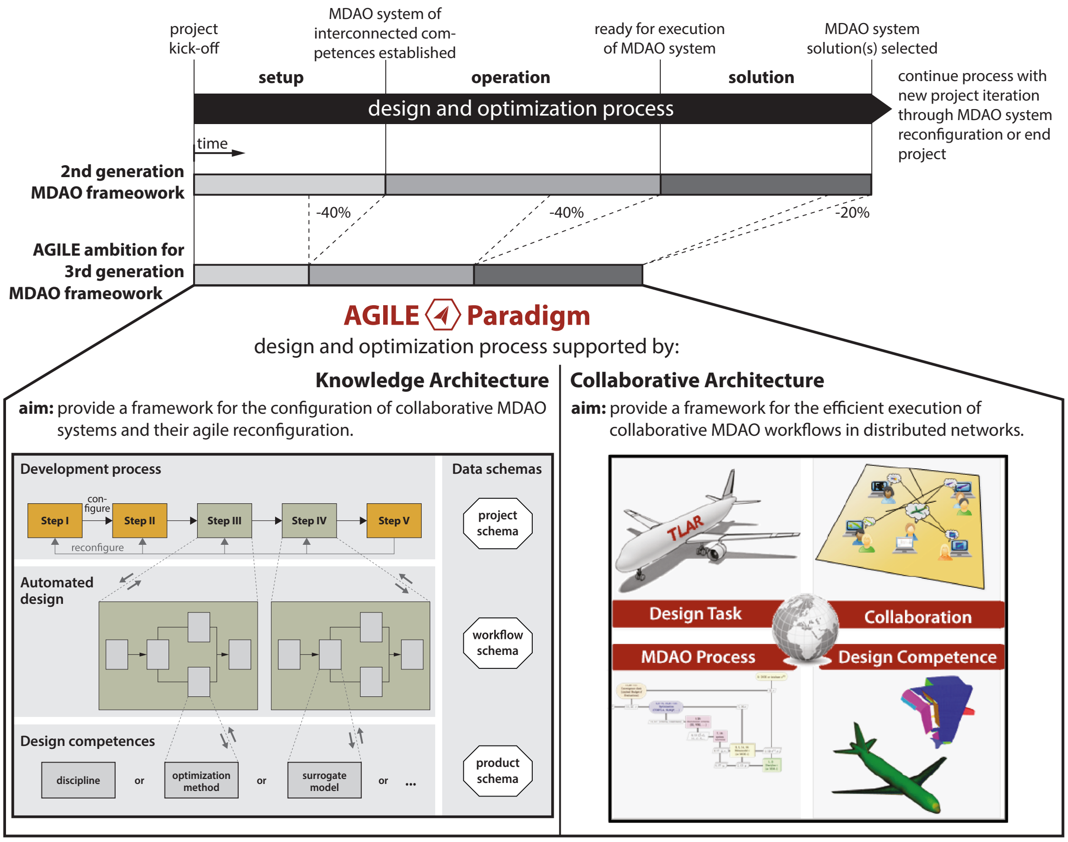

project kick-off
MDAO system of interconnected competences established
ready for execution of MDAO system
MDAO system solution(s) selected
setup
operation
solution
design and optimization process
continue process with new project iteration through MDAO system reconfiguration or end project
time
2nd generation MDAO frameowork
-40%
-40%
-20%
AGILE ambition for 3rd generation MDAO frameowork
AGILE Paradigm
design and optimization process supported by:
Knowledge Architecture
aim: provide a framework for the configuration of collaborative MDAO systems and their agile reconfiguration.
Collaborative Architecture
aim: provide a framework for the efficient execution of collaborative MDAO workflows in distributed networks.
Development process
Step I
configure
Step II
Step III
Step IV
Step V
reconfigure
Automated design
Design competences
discipline
or
optimization method
or
surrogate model
or
...
Data schemas
project schema
workflow schema
product schema
TLAR
Design Task
Collaboration
MDAO Process
Design Competence

Figure 2.17: AGILE Paradigm - Conceptual overview

technical challenge, specifically addressed by AGILE, is the homonym *agility* challenge; the ability to reconfigure a previously assembled system, such to support designers exploring new insights acquired after the first computation runs, to include new or modified design requirements, or to change or add some design competences to the already established automated design process. This need for agility matches with the *reconfiguration hurdle* identified in the introduction of this dissertation.

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

The solution phase challenges are mostly of technical nature and concern the capability to reach convergence of the executable workflow and identify robust optima within the allocated time. New optimization algorithms and MDAO architectures are continuously developed to address these issues [14, 79, 80]. Recently, quite some developments are happening also concerning the computational infrastructure, for which software solutions are being devised to cloudify computationally expensive workflows [81]. Intellectual property, software licensing policies and security issues, again, hamper the practical usability of such technical solutions. After the third phase the design project is finished and needs to be stored in a systematic manner to be useful for future reference or if modifications turn out to be necessary.

The excessive time to configure an MDAO system, the lack of reconfiguration agility during deployment, and the struggle to maintain overview and control have been identified by AGILE as the main limitations of the first two framework generations. To address these fundamental challenges, a new methodological approach, the so-called AGILE paradigm, is built on top of two main cornerstones: the *Knowledge Architecture (KA)* and the *Collaborative Architecture (CA)*. The KA provides the structured approach and workbench to formulate, (re)configure and inspect any design process, including fully specified MDAO systems that are ready to be converted into executable computational systems. The CA includes the methods and tools to assemble and deploy executable workflows across distributed networks. More specifically, it provides the means to connect simulation tools in a service-oriented scenario, including solutions for the cross-human (e.g. disciplinary specialists need to stay in control of their own tools) and cross-organizational (e.g. intellectual property restrictions and firewalls) issues occurring in a collaborative distributed process. Fig. 2.17 schematically illustrates the whole AGILE paradigm and provides a qualitative representation of the targeted time reductions in the three phases of the MDAO development process: a 40% reduction in time needed to configure a multidisciplinary system by a team of heterogeneous specialists in the set-up and operation phases and a further reduction of 20% in time to find an (optimized) design solution.

The project was divided in three design campaigns and six work packages, see Fig. 2.18. The developments presented in this dissertation are key components of the final AGILE framework and were developed under the overarching work package 6 (WP6), in which the knowledge architecture was developed. These developments were integrated in the full framework and tested in the second and third design campaigns. Both the framework and these design campaigns will be elaborately discussed in Chapter 6.

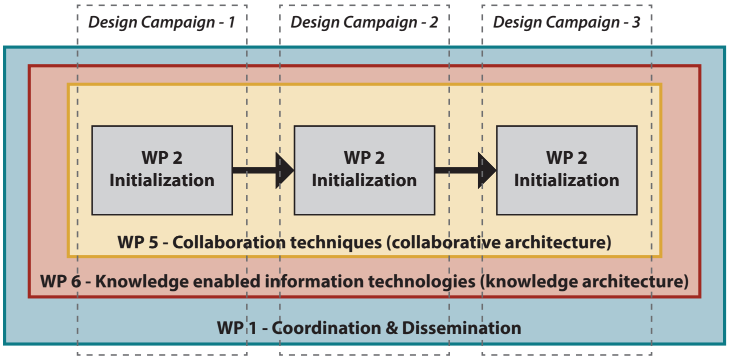

Design Campaign - 1
Design Campaign - 2
Design Campaign - 3
WP 2 Initialization
WP 2 Initialization
WP 2 Initialization
WP 5 - Collaboration techniques (collaborative architecture)
WP 6 - Knowledge enabled information technologies (knowledge architecture)
WP 1 - Coordination & Dissemination

Figure 2.18: AGILE design campaigns and work packages [16]

2.7.3. Current limitations and future needs

In this chapter, the state of the art concerning six different aspects of handling MDAO systems was discussed. Here, the current limits are summarized and six needs to move forward are identified.

In §2.1, two methods were identified for composing the tool repository of the system: centralized and decentralized mapping. Within the centralized mapping approach the idea of using a fixed Central Data Schema (CDS) is gaining momentum for collaborative engineering in the MDAO community and is the preferred approach for the AGILE framework. Two challenges with this approach, where tools are connected indirectly via the schema, are the loss of system oversight and the presence of problematic variables and connections, hence:

> *The fixed schema approach calls for a dedicated composition platform to provide oversight and the ability to check and fix the system.*

In §2.2, the range of representations for MDAO systems was reviewed. A clear distinction can be made between representations aimed at human-readability (e.g. FDT, XDSM) and machine-readability (e.g. semantic webs, programming objects). A middle way between these two was formalized by Pate et al. [15] using graphs. However, their graph-based approach was limited to automatically identifying the right set of tools to solve a given MDAO problem and thus:

> *The graph-based approach represents a solid conceptual basis, but requires a rigorous revision and extension to match the needs of the third-generation framework.*

The manipulation section (§2.3) listed a broad collection of methods and applications to transform the system to subsequent stages in the MDAO development process (Fig. 2.7). Here, the state of the art lacks a comprehensive methodology (and associated platform) to support the entire formulation phase for a large, heterogeneous design team that is also able to bridge the gap between formulation and execution, so:

> *The envisioned collaborative MDAO framework requires a methodology (and*

prototype implementation) to handle the MDAO system at all stages of the development process, and needs to support all manipulations required to transform it.

§2.4 showed how the solution strategy for the problem at hand can be coordinated using monolithic or distributed schemes. In the industrial practice, performing MDAO is based on exploiting different strategies fitting the status of the project (i.e. what decision needs to be made) and the team's understanding of the design space. Monolithic strategies are the most straightforward ones, but do not necessarily match the distributed nature of the departmentalized organization well. Therefore, distributed architectures, such as CO and BLISS-2000, might prove their value for these cases, consequently:

Both monolithic and distributed architecture types need to be supported throughout the development process.

The actual creation and execution of the workflows can be done in a range of PIDO platforms, which were summarized in §2.5. Each of these platforms has their strengths and weaknesses leading to the conclusion that no single platform would fit every project. Hence, the formalization applied by the methodology in the formulation phase needs to be PIDO platform agnostic. In addition, the manual creation of the workflows with these platforms is a cumbersome and error-prone task and prototypes automating this task have shown huge potential time benefits. Hence, the needs concerning workflow execution for the third-generation framework are stated as follows:

The formulated MDAO solution strategy needs to be platform-agnostic and the creation of the executable workflow has to be automated.

Finally, the section on framework integration (§2.6) discussed four recent collaborative MDAO projects. These projects showed that it is getting easier to technically set up a distributed, executable MDAO system, however, a reconfigurable process is not yet part of the state of the art and the tremendous resource investment required to configure the first executable workflow remains a hurdle. Hence, as anticipated in the introduction:

The configuration and reconfiguration hurdles need to be lowered to establish a novel MDAO framework generation.

The developments presented in the next part of this dissertation address the six needs identified above to enable a novel third-generation MDAO framework.

II

Developments

3 Graph-based methodological approach for MDAO system development

THE analysis of the state of the art, provided in Chapter 2, revealed the lack of a comprehensive methodology to formulate and manipulate MDAO systems throughout their iterative development process in large, collaborative engineering projects. This shortcoming is fixed here by a comprehensive formalization of such systems. In addition, this formal specification is also implemented to test and finetune it. This chapter presents the theoretic foundation of a novel graph-based methodology, followed by its demonstration in two case studies. The requirements for the formalization are stated in §3.1, followed by its theoretical development in §3.2-3.5. In §3.6 and §3.7, two case studies demonstrate the implemented formalization in KADMOS: the open-source Python package that was written based on the theoretical foundation presented here.

3.1. Functional requirements

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

The contents of this chapter have been adapted from [82].

of large, distributed and collaborative MDAO systems in the formulation phase, as indicated in Fig. 3.1. To this purpose, the following top-level requirements were set:

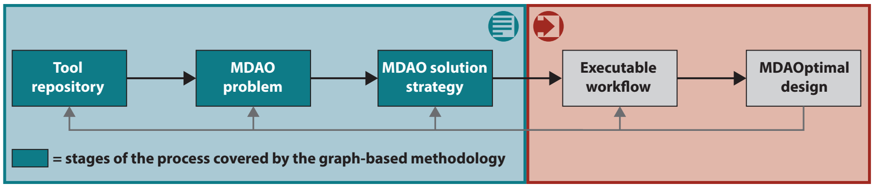


Figure 3.1: The coverage of Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System (KADMOS) in the MDAO development process

I **System composition:** The methodology should be based on the CDS approach for I/O data exchange. When a large number of heterogeneous design tools is involved, the overhead of making them all compatible to a CDS is negligible with respect to the challenge a system integrator would face by manually mapping all their I/Os, especially when the MDAO system is supposed to be frequently adjusted and reconfigured.

II **System representation:** The syntax should be based on the graph-theoretic foundation for MDAO systems initiated by Pate et al., but should also provide the human-readability of XDSMs and be extended to cover all stages of the MDAO development process.

III **System manipulation:** The methodology should support automated formulation of MDAO solution strategies including MDA, DOE and MDO architectures. MDO-type architectures should include both monolithic and distributed schemes.

IV **PIDO platform independence:** The formulated MDAO solution strategy should be portable to a range of PIDO platforms, while maintaining complete independence from them.

V **Controlled automation:** The approach should automate all the repetitive, non-creative tasks necessary to advance from one formulation stage to the other, while keeping the design team in control of all settings and strategic decisions that require engineering judgment (e.g. to define the problem, to pick the architecture).

VI **Tool heterogeneity:** The system should support a broad range of design tool types ranging from simple mathematical relations, to more complex surrogate model relations, up to complex disciplinary tools to be executed as black boxes on separate server domains because of intellectual property constraints.

VII **Scalability:** The system should be able to handle systems of any size and complexity.

3.2. Graph syntax and main graph classes

This section provides a formal definition of the adopted graph syntax adopted. Two main graph classes are defined as well: the data and process graph. The syntax follows the notation of Diestel [83] and Pate et al. [15]. Key concepts of graph theory are briefly revisited here for convenience, followed by the definition of nodes (§§3.2.1), edges (§§3.2.2), and main graph classes (§§3.2.3).

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

$$G = (V, E)$$

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

An example digraph is shown in Fig. 3.2. This graph is defined with two sets:

$$V = \{n, o, p, q, r, s, t\}$$
$$E = \{(n, n), (n, q), (o, n), (o, r), (q, o), (q, p), (r, o), (r, t), (t, r)\}$$

A *loop* in a digraph is defined as an edge with $init(e) = ter(e)$, see node n in Fig. 3.2 for an example. Loops are not allowed, meaning that only *simple digraphs* are used. A *looped pair* is allowed in simple digraphs. Looped pairs are defined as pairs of nodes $\{v, w\}$ for which there is an edge in both directions, hence $(E^+(v) \supset e \mid ter(e) = w) \wedge (E^-(v) \supset e \mid init(e) = w)$. See the pairs $\{o, r\}$ and $\{r, t\}$ in Fig. 3.2. The number of such looped pairs of one node v with respect to its neighbours is referred to as the *circularity index*: $cir(v)$.

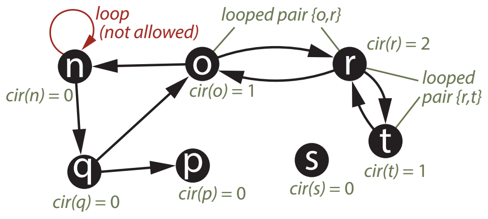


Figure 3.2: Example of a directed graph illustrating some key concepts.

A *path* $Q = (V, E)$ from v_0 to v_k in graph G is a subgraph of G ($Q \subseteq G$) with $V = \{v_0, v_1, ..., v_k\}$ and $E = \{(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)\}$. See as example the path with $V = \{t, r, o, n, q, p\}$ in Fig. 3.2. A *cycle* C is a path for which $v_0 = v_k$, e.g. the cycle with $V = \{o, n, q, o\}$ in Fig. 3.2.

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

To combine graphs, a notation for the union of a set of sets is required. If we define I to be a non-empty set such that for each $i \in I$ there is a corresponding set A_i, then the set of sets $\mathscr{A} = \{A_i \mid i \in I\}$ is called an indexed family of sets with index i and indexing set I [84]. The union of this family of sets can be denoted in different ways:

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

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

3.2.1. Node definitions

Graph nodes are enriched using different attributes. The attribute values are used to inspect and manipulate the graphs. The different node attributes are listed in Tab. 3.1 and explained in the following subsections.

Table 3.1: List of attributes for graph nodes

attribute name	notation	used for	typical values
category	*cat*()	V	function, variable
subcategory	*sct*()	V	input, coupling, collision, hole (see Tab. 3.2)
instance	*ins*()	V	0, 1, 2, 3
problem role	*pr*()	V	design variable, objective, coupled
architecture role	*ar*()	V	initial guesses, copies, optimizer (see Tab. 3.8)
mode	*mode*()	V_f	viscous, inviscid, 1, 2, 3, A, B, C
process step number	*psn*()	V_f	$\{0,7\}$, $\{1\}$

Category

The main attribute of a node is its *category*, denoted as $cat(v)$, which can have the two following values:

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

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

Subcategory

A more refined *subcategorization* of all the nodes in a graph can be defined based on their indegree, outdegree, and circularity index. This categorization can be inferred automatically based on these three properties. All possible *subcategories*, denoted by *sct*(v), are listed in Tab. 3.2. The subcategories play an important role in different graph classes, as some subcategories are not allowed in certain stages of the MDAO system, while others require a specific treatment in the graph manipulation algorithms. For example, the hole and collision node subcategories are generally considered problematic and need to be removed or fixed. Circular variable nodes, which are related to the circularity index introduced in the previous subsection, also require a specific treatment in certain graphs, as will be discussed in §§3.4.2 and §3.7.

Instance

All nodes can get an *instance* specified. A node instance attribute is used to allow multiple instances of a node that refer to the same tool in the tool repository, or to the same element in the CDS. A node instance is an integer value denoted by $ins(v)$, where the default instance value is zero. A practical example of the use of node instances for a variable is illustrated in Fig. 3.3. Here, the two functions A and B have the same variable b

Table 3.2: Subcategory definition of graph nodes and references to example nodes

category	subcategory	δ^-	δ^+	cir	Fig. 3.5a
variable	hole	0	0	0	a
	supplied input	0	1	0	b
	supplied shared input	0	>1	0	c
	output	1	0	0	d
	collision	>1	0	0	e
	coupling/	1	1	0	f
	pure circular coupling			1	g
	shared coupling/	1	>1	0	h
	shared circular coupling			1	i
	collided coupling/	>1	1	0	j
	collided circular coupling			1	k
	collided shared coupling/	>1	>1	0	-
	collided shared circular coupling			≥1	-
function	hole	0	0	0	A
	source	0	>0	0	B
	sink	>0	0	0	C
	complete	>0	>0	≥0	D

as output, but the values written by those tools are required at different moments in the MDAO system execution by tools *C* and *D*. In such a case, it could be required that *A* first determines the value of *b* ($ins(b) = 0$) which is used by tool *C*, and *B* later overwrites this value ($ins(b) = 1$) in the schema file so that it can be used used by tool *D*. Hence, node instances are required in the graph to be able to indicate which functions are using and producing a particular node instance. In the aircraft design practice, a typical example would be the use of two weight estimation tools (class I and class II) that both provide the maximum take-off mass value as output.

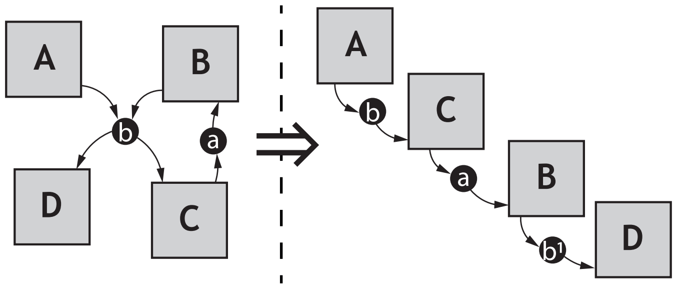


Figure 3.3: Illustration of the application of the instance attribute to split a variable.

Problem and architecture role

Two key attributes of both function and variable nodes are defined to describe the role they play in a certain graph:

- problem role (denoted $pr(v)$)
- architecture role (denoted $ar(v)$)

The *problem role* is used on variable nodes ($pr(v_v)$) to indicate special variables, such as design variables, constraints, objective, and quantities of interest (QOIs). The problem role of a function ($pr(v_f)$) is related to its position within the MDAO system and will be further explained in the next section.

The *architecture roles* are used in the MDAO Data Graph (MDG) and MDAO Process Graph (MPG). These roles indicate special variable ($ar(v_v)$) and function nodes ($ar(v_f)$) required by MDAO architectures. For variables, these are roles such as initial guess, copy variable, and final value. For functions, the architecture role can be optimizer, converger, and consistency constraint function. A full list of architecture roles is provided in Tab. 3.8 and is further discussed in §3.4.

Mode

An attribute specific for function nodes is the *mode* attribute, denoted by mode(v). This attribute is used to indicate that the same tool from the repository can be executed in different operational modes (e.g. an aerodynamic analysis tool can operate both in viscous or inviscid mode), using different I/O branches in the CDS. By using the keyword 'mode' the I/Os of the different function modes can be filtered automatically, as shown in Fig. 3.4. The advantage of using the mode attribute is that a tool with multiple execution modes can still be stored in the repository as one tool, with only one mapping definition to the CDS.

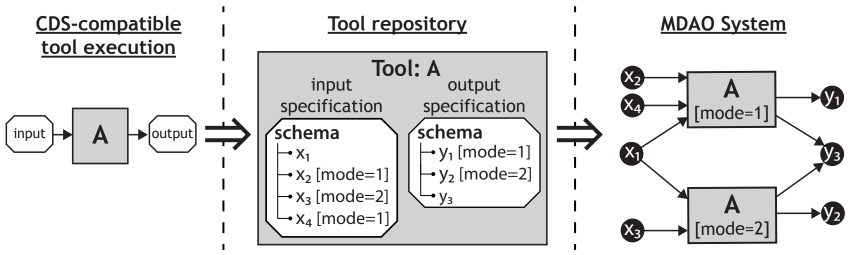


Figure 3.4: Illustration of the use of the attribute *mode*. Tool A can operate in two modes, using/producing different I/O values (right). Tool A is stored in the repository as one tool (center), requiring only one mapping to the CDS accounting for both modes (center, left).

Process step numbers

Another function-node-specific attribute is the ordered set containing the *process step numbers* (PSNs), denoted by $psn(v)$. These step numbers are integers used to specify the execution order of the function nodes in a process graph (Fig. 3.5b). The process numbering adheres to the XDSM convention. A function node can have multiple PSNs to allow for cycles. The maximum and minimum PSNs present in a graph G are denoted by respectively $maxpsn(G)$ and $minpsn(G)$.

3.2.2. Edge definitions

Two edge types are defined, called data and process edge, each belonging to a different graph class. The attributes used for edges are summarized in Tab. 3.3 and the two types are defined as follows:

data edge (e_d): represents the relation between a variable node and a function node. All data edges in a graph are denoted by $E_d = \{e \in E \mid cat(e) = \text{data}\}$. A data edge is always defined using one variable node and one function node. If $ter(e_d) \in V_f$ then the edge is an *input edge* and, vice versa, if $ter(e_d) \in V_v$ then the edge is an *output edge*. The data edges are indicated in Fig. 3.5a.

process edge (e_p): represents the relation between function nodes in a process graph. The subset of process edges in graph G is $E_p = \{e \in E \mid cat(e) = \text{process}\}$. Just like function nodes, the process edge can have a PSN attribute, denoted by $psn(e)$. For process edges the PSN is not an ordered set, but a single integer value, since each process edge can only represent a single step, see Fig. 3.5b for an example.

Table 3.3: List of attributes for graph edges

attribute name	notation	used for	typical values
category	*cat*()	E	data, process
process step number	*psn*()	E_p	1, 2, 3

3.2.3. Main graph class definitions

Two main graph classes are defined using the node and edge definitions from the previous sections: the data and the process graph.

Data graph

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

1.	$\lvert V_f\rvert + \lvert V_v\rvert = \lvert V_D\rvert$	(nodes are functions or variables)
2.	$\lvert E_d\rvert = \lvert E_D\rvert$	(edges are data edges)

Hence, the data graph connects function nodes with variable nodes, as shown in Fig. 3.5a.

Process graph

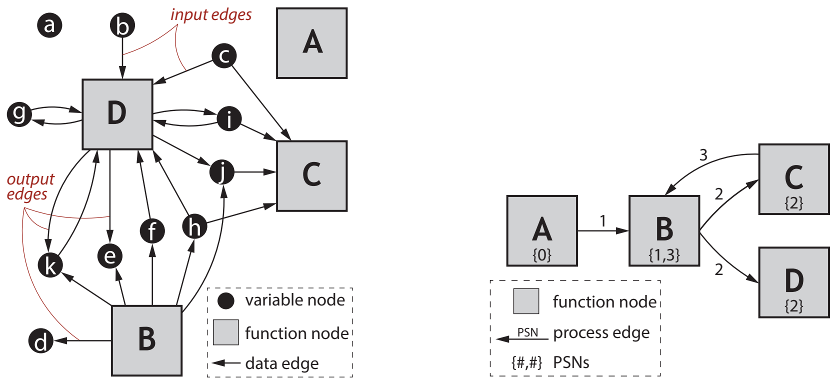
Instances of the `ProcessGraph` class are digraphs $P = (V_P, E_P)$ meeting the following conditions:

1.	$\lvert V_f\rvert = \lvert V_P\rvert$	(nodes are functions)
2.	$\lvert E_p\rvert = \lvert E_P\rvert$	(edges are process edges)
3.	$\forall v \in V_P : \delta^-(v) + \delta^+(v) \geq 1$	(each node is connected by at least one edge)
4.	$minpsn(P) = 0$	(process starts at PSN of zero)
5.	$\forall e \in E_P : psn(e) \in \mathbb{Z}^>$	(edge PSNs are positive integers larger than zero)
6.	$\forall v \in V_P : \lvert psn(v)\rvert > 0 \wedge$ $\forall i \in psn(v) : i \in \mathbb{Z}^\geq$	(all nodes have at least one PSN which are all larger or equal to zero)

7. $\forall e \in E_P : psn(e) \in psn(ter(e))$ (every edge target node contains the edge's PSN)

8. $\forall e \in E_P :$ $psn(init(e)) \ni x \mid x < psn(e)$ (every edge source node contains a PSN smaller than the edge PSN)

The last two conditions enforce process continuity by demanding the nodes around a process edge to have corresponding PSNs. For example, a process edge e with $psn(e) = 4$ should have an $init(e)$ that contains a PSN of 3 or lower and a $ter(e)$ containing a PSN of 4). Fig. 3.5b depicts a small process graph example.



a) Data graph instance b) Process graph instance

Figure 3.5: Example instances for each of the two main graph classes.

3.3. MDAO SYSTEM GRAPH TYPES

The two graph types described in the previous section, data and process graph, were taken as the base for four specialized graph classes that represent the MDAO system throughout the entire formulation process depicted in Fig. 3.1:

Data graphs

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

Fundamental Problem Graph (FPG) The second graph type matches the second stage of the process: MDAO problem. The FPG is an enriched (i.e. containing additional attributes) subset (i.e. due to removal of unnecessary functions and variables) of the RCG. It is created by performing graph manipulations on an RCG to define a graph that represents a valid (in terms of strict graph-theoretic conditions discussed in the next sections) MDAO problem. Thus, the graph includes the tools from the repository that should be used to solve the problems, and the indication of key problem variables, such as design variables, objective, and constraints. Once established, the graph is used to impose an MDAO architecture on it.

MDAO Data Graph (MDG) The MDAO solution strategy stage is represented by two graphs: a data and a process graph. The MDG is the graph that stores the data exchanged by the executable blocks and the CDS nodes that are required to solve the MDAO problem, according to the selected architecture.

The executable blocks in the MDG include both the repository tools from the FPG and the *architecture elements*, necessary to implement the MDAO architecture at hand, such as convergers, optimizers, and Architecture-specific Mathematical Relations (AMRs) (i.e. consistency constraint functions).

Process graphs

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

The developed methodology has also been implemented as an open-source Python package: Knowledge- and graph-based Agile Design for Multidisciplinary Optimization System (KADMOS) [S19]. The class diagram of this package is shown in Fig. 3.6. All KADMOS graphs are subclasses of the `DiGraph` class from the NetworkX [85] package. The main graph classes `DataGraph` and `ProcessGraph` were defined in the previous section, while the detailed description of the four MDAO system graphs listed above is provided in the next section, supported by a small illustrative example.

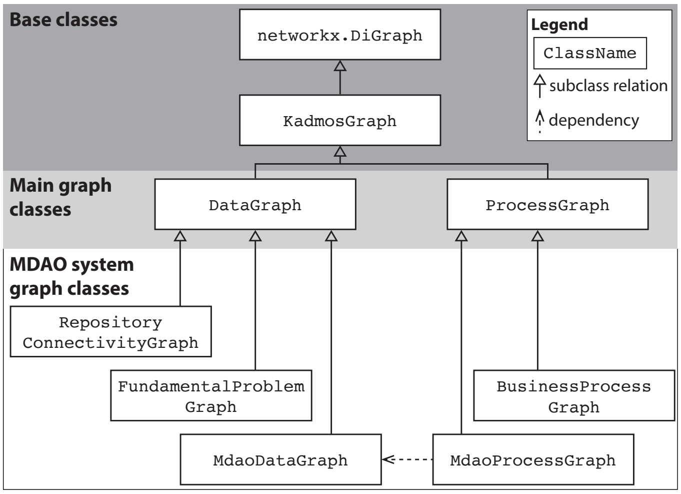


Figure 3.6: Class diagram of the KADMOS package. The `BusinessProcessGraph` class is added to the diagram for illustrative purposes.

3.4. Definition of MDAO system graphs

In this section we make use of a simple analytical MDAO problem to clarify the determination and use of the four graph classes introduced in the previous section: the Sellar problem [86]. Additionally, this small problem serves as the first demonstration, verification, and validation case for the other developments presented in this part of the dissertation. The Sellar problem can be described by the following tools, where the tools indicated with D represent the actual disciplines, F the objective function, and G the constraints:

$$\begin{aligned}
\text{D [mode=1]} \quad &\Rightarrow \quad y_1 = c \cdot (z_1^2 + x_1 + z_2 - 0.2 \cdot y_2) \\
\text{D [mode=2]} \quad &\Rightarrow \quad y_2 = c \cdot (\sqrt{y_1} + z_1 + z_2) \\
\text{F} \quad &\Rightarrow \quad f = x_1^2 + z_2 + y_1 + e^{-y_2} \\
\text{G [mode=1]} \quad &\Rightarrow \quad g_1 = \frac{y_1}{3.16} - 1 \\
\text{G [mode=2]} \quad &\Rightarrow \quad g_2 = 1 - \frac{y_2}{24.0}
\end{aligned}$$

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

$$\begin{aligned}
\text{A} \quad &\Rightarrow \quad b = \mathrm{f}(a) \\
\text{B} \quad &\Rightarrow \quad b = \mathrm{f}(b) \\
&\Rightarrow \quad z_1 = \mathrm{f}(b) \\
&\Rightarrow \quad z_2 = \mathrm{f}(b) \\
\text{C} \quad &\Rightarrow \quad c = \mathrm{f}(b) \\
\text{E} \quad &\Rightarrow \quad y_1 = \mathrm{f}(b, z_1, z_2) \\
&\Rightarrow \quad y_2 = \mathrm{f}(b, z_1, z_2) \\
\text{H} \quad &\Rightarrow \quad x_1 = \mathrm{f}(x_0)
\end{aligned}$$

Before the first graph in the KADMOS approach, the RCG, can be created, the tools have to be made compatible to a single CDS and stored in the tool repository, as illustrated in Figures 3.7a and 3.7b.

3.4.1. Repository connectivity graph

The RCG is a specific type of data graph (see §§3.2.3) that is used to represent the first stage of the MDAO development process: tool repository. The RCG $R = (V_R, E_R)$ can be built by combining the data graphs that represent individual functions. For each unique function/mode combination $i \in I_r$ stored in the tool repository $I_r = \{1, 2, \ldots, m\}$ a data graph can be constructed:

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

Where the nodes are:

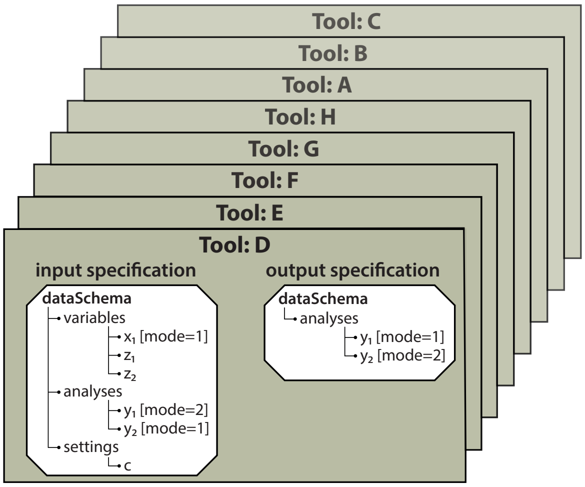
$$V_{D,i} = v_{f,i} \bigcup V_{v,I,i} \bigcup V_{v,O,i}$$

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

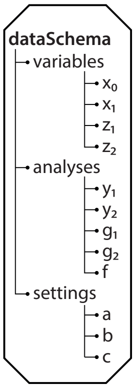
$$E_{D,i} = E^{-}(v_{f,i}) \bigcup E^{+}(v_{f,i})$$

where:

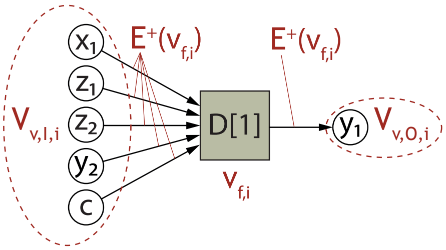
$$\begin{aligned}
E^{-}(v_{f,i}) &= \{(v_v, v_{f,i}) \mid v_v \in V_{v,I,i}\} \\
E^{+}(v_{f,i}) &= \{(v_{f,i}, v_v) \mid v_v \in V_{v,O,i}\}
\end{aligned}$$



a) Storage of tool D in the repository



b) Sellar CDS



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

Figure 3.7: Extended Sellar problem tool repository

The data graph of the tool D[1] ([1] denotes *mode*(D) = 1) is shown in Fig. 3.7c. The nodes and edges of the RCG can be written as:

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

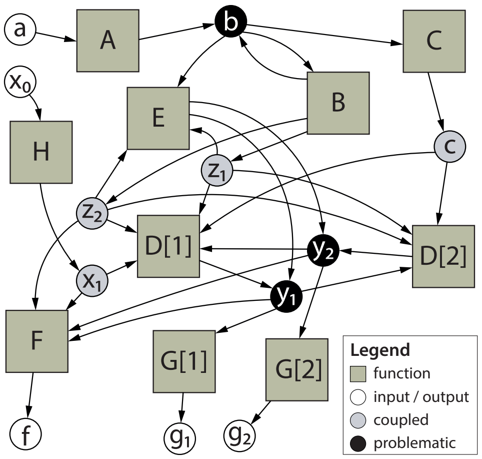


Figure 3.8: RCG of the extended Sellar tool repository

The RCG shown in Fig. 3.8 is automatically instantiated by KADMOS based on the Sellar tool repository defined in Fig. 3.7. After instantiation, the subcategory of the nodes (Tab. 3.2 in §§3.2.1) can be determined to filter the graph nodes, leading to the identification of the four main node subcategories for each variable node v, as illustrated in Fig. 3.8 legend:

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

The handling of these main node subcategories will be further discussed in the next section. Note that for all nodes in an RCG $ins(v) = 0$.

3.4.2. FUNDAMENTAL PROBLEM GRAPH

The FPG contains the definition of the fundamental MDAO problem to be solved and represents the second stage of the system development process in Fig. 3.1. The graph is a subset of the RCG, containing only the tools that are strictly necessary to solve the optimization problem at hand, plus extra information on the specific role of the tools (which one is a coupled discipline, which the objective function, etc.) and involved variables (which ones are design variables, which fixed parameters, etc.). The FPG is the starting point to impose one of the MDAO architectures in the third stage of the formulation phase.

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

1. $\forall v \in V_{F,f}$:
   $v \in V_{R,f} \vee v$ is merged based on $\subseteq V_{R,f}$ (functions are in RCG or else are merged from RCG nodes)
2. $\forall v \in V_{F,v}$:
   $v \in V_{R,v}$ if $ins(v) = 0$ else
   $\exists w \in V_{R,v} : w = v$
   except $ins(w) \neq ins(v) \wedge ins(w) = 0$ (variables are in RCG or higher instances of RCG variables)
3. $\forall v \in V_{F,v}$:
   $cir(v) = 0 \wedge \delta^{-}(v) + \delta^{+}(v) \geq 1 \wedge \delta^{-}(v) \leq 1$ (variables cannot be holes, collided, or circular, see Tab. 3.2)
4. $\forall v \in V_{F,f}$:
   $\delta^{-}(v) \geq 0 \wedge \delta^{+}(v) > 0$ (functions have to be source or complete, see Tab. 3.2)
5. $\forall v \in V_{F,f}$:
   $pr(v) \in \{$uncoupled-DVI, uncoupled-DVD, coupled, post-coupling$\}$ (all functions have a problem role)
6. $\forall v \in V_{F,v}$: if $pr(v) =$ design variable $\Rightarrow \delta^{-}(v) = 0$ (design variables are inputs)
7. $\forall v \in V_{F,v}$: if $pr(v) =$ QOI $\Rightarrow \delta^{-}(v) = 1$ (QOIs are couplings or outputs)
8. $\forall v \in V_{F,v}$:
   if $pr(v) =$ objective $\vee$ constraint $\Rightarrow \delta^{+}(v) = 0$ (objective and constraints are outputs)
9. $\forall v \in V_F$:
   $\exists$ path $Q = (V_Q, E_Q)$ where $v \in V_Q \;\wedge\; V_Q \ni w$
   where $pr(w) \in \{$QOI, objective, constraint$\}$ (all nodes eventually lead to at least one QOI, objective, or constraint)

The last condition demands that, for all nodes, at least one path can be created that leads to a QOI, objective, or constraint. Hence, nodes not included in any of those paths can be removed. This is a useful condition to trim a graph once the key variables have been indicated, since any node not falling under this condition can be safely removed.

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

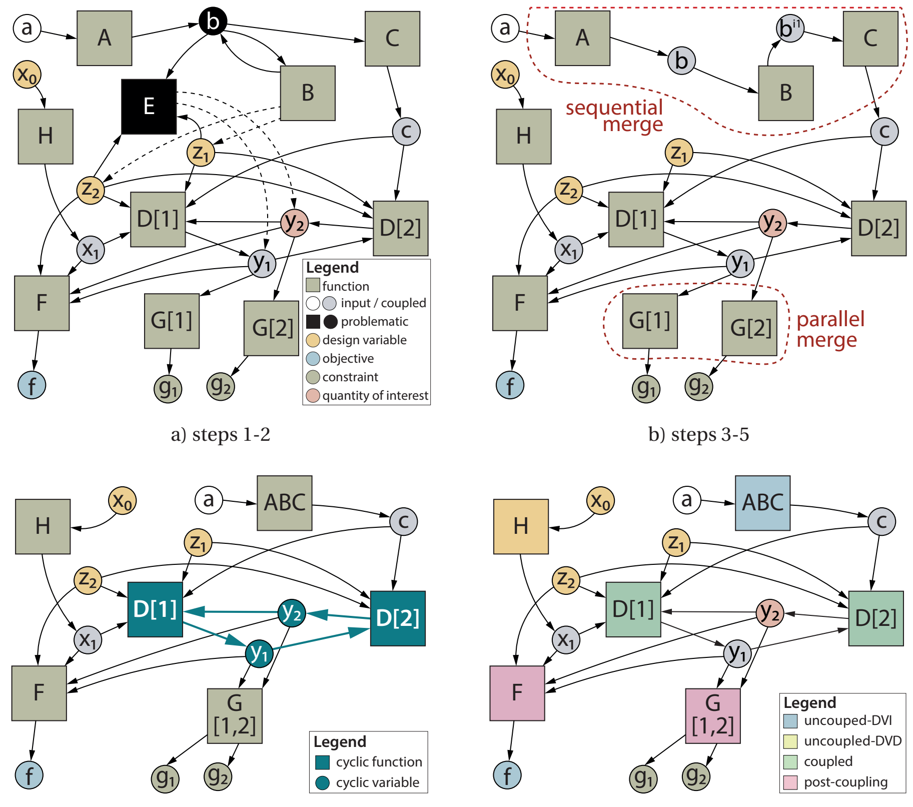
The FPG composition process for the Sellar problem is illustrated in Fig. 3.9. This process starts from the RCG in Fig. 3.8. The anticipated architecture type (step 1) is MDO. In step 2 (Fig. 3.9a) x_0, z_1, and z_2 are marked as design variables. In order to make z_1 and z_2 valid design variables the incoming edges (B, z_1) and (B, z_2) must be removed. The objective f and constraints g_1 and g_2 can directly be marked as such, since these are already of the valid subcategory *output*. Finally, if the design team is interested in keeping track of the coupling variable y_2, this has to be defined as a QOI. Therefore, the preferred function to determine y_2 has to be selected (between tool D and E). In this case, tool D is manually selected by the design team, also for the variable y_1, thus the edges (E, y_1) and (E, y_2) are removed.

The only problematic nodes left in the graph (step 3) are tool E and variable b. After the previous edge removal, tool E is now a sink and can safely be removed from the graph. Node b expresses a special case, in which a tool takes an initial value and then

Algorithm 1: FPG composition process

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

Legend
A: automated step
M: manual step supported by KADMOS scripting commands



a) steps 1-2

b) steps 3-5

c) step 6 - cycles

d) step 6 - function problem roles

Figure 3.9: Sellar problem FPG composition

updates that same value. Hence, the input and output point to the same location of the schema. This is common practice for systems using a CDS approach, when tools take initial guesses or update the values (e.g. the aircraft weight) stored in a schema file. Similar to the example in Fig. 3.3, this type of node is made valid by splitting it in two instances using a KADMOS method resulting in b ($ins(b) = 0$) and b^{i1} ($ins(b) = 1$). Fig. 3.9b depicts the FPG of the Sellar problem after the execution of the previous steps.

In step 4 a KADMOS check method is executed which returns a positive result. Then the team can merge functions that belong together and that can be executed in sequence (i.e. without any feedback coupling) or in parallel (i.e. without any coupling). Hence, in step 5 the function sequence $\langle \text{A}, \text{B}, \text{C} \rangle$ is merged as one function ABC, and the set of parallel functions $\{\text{G[1]}, \text{G[2]}\}$ is merged as G[1,2]. See Fig. 3.9c for the resulting graph. These function mergers are useful to declutter the system (and maintain oversight) by clustering more disciplines into fewer "macro-discipline" blocks.

Step 6 of the FPG composition process consists in the determination of the problem role of the various functions. The function problem roles in an FPG belong to one of four main groups: uncoupled-DVI, uncoupled-DVD, coupled, and post-coupling, where the suffix of the two uncoupled types indicates whether the functions are Design Variable Dependent (DVD) or Design Variable Independent (DVI). The coupled functions are the set of functions that are involved in cycles in the FPG. These cycles indicate that a method is required to converge the system, since any order of executing the tools will always require some feedback loop. The single cycle present in the Sellar FPG is depicted in Fig. 3.9c. Hence, tools D[1] and D[2] have the problem role 'coupled'. The uncoupled functions are all the functions on an incoming path with respect to the cycles, for example the function ABC on the path $\{a, \text{ABC}, c, \text{D[1]}\}$. The default problem role for uncoupled functions is uncoupled-DVI. However, if the FPG contains design variables, then some of the uncoupled functions could also be DVD and should be marked as such, see tool H in this example. This distinction is required later to correctly position the uncoupled functions either outside (if DVI) or inside (if DVD) the main cycle handling design variables (i.e. DOE or optimizer block). The post-coupling functions are simply all the remaining functions. The problem roles are shown in Fig. 3.9d.

In step 7 the description of the fundamental MDAO problem is completed by specifying the sequence of the functions. The sequences for each problem role are determined automatically in KADMOS. The uncoupled and post-coupling sequences are straightforward, since no cycles are present in those function sets. Any sequence of functions that constitutes a topological order of the nodes is valid, where for the uncoupled functions the most optimal sequence is a sequence where the DVD functions are required as late as possible, hence $\langle \text{ABC}, \text{H} \rangle$. The sequencing of the coupled functions is more challenging as the number of feedback variables can depend on the sequence given. For this purpose, sequencing algorithms have been implemented that search for the optimal sequence by minimizing the number of feedback couplings. These algorithms will be discussed in §§3.5.3. However, for the Sellar problem both possible sequences have the same number of feedback variables, thus the sequence $\langle \text{D[1]}, \text{D[2]} \rangle$ was arbitrarily selected. The post-coupling sequence is set to $\langle \text{F}, \text{G[1,2]} \rangle$.

If a distributed MDO architecture needs to be implemented, then the coupled functions in the FPG need to be grouped to indicate how the system needs to be distributed. This grouping can be determined automatically using a decomposition algorithm, which is discussed in more detail in §§3.5.4. In the small Sellar example, the two groups are simply the two coupled functions. The other functions in the FPG will be grouped automatically when the distributed MDO architecture is imposed, as further discussed in §§3.5.2. The FPG in Fig. 3.9d has been defined with the MDO architecture types in mind, but it is possible to use nearly the same FPG also for MDA and DOE by simply reassigning variable problem roles based on step 2 of Algorithm 1.

3.4.3. MDAO DATA GRAPH

The MDG, together with the MPG in the next section, belongs to the last set of graphs produced by KADMOS to enable the third stage of the formulation phase. The MDG is constructed automatically by a KADMOS graph manipulation algorithm that transforms the previously generated FPG into the MDAO solution strategy based on the architecture selected in step 1 of Algorithm 1. The MDG $M_D = (V_{M_D}, E_{M_D})$ has to meet the following conditions:

1. $V_F \subset V_{M_D}$ (graph contains all FPG nodes)
2. $\exists! v \in V_{M_D} : ar(v) = \text{coordinator} \Rightarrow v = \text{COOR}$ (graph has a single coordinator)
3. $\forall v \in V_{M_D}$:
   $\exists$ cycle $C = (V_C, E_C)$ where $v \wedge \text{COOR} \in V_C$ (all nodes are on a cycle that includes the coordinator)
4. $\forall v \in V_{M_D,f} \cap V_{F,f} : ar(v) \in \{\text{1st col. Tab. 3.8}\}$ (architecture role FPG functions)
5. $\forall v \in V_{M_D,f} \setminus V_{F,f}$:
   if $(\exists w \in V_{F,f}$ where $w = v$ except
   $ins(w) \neq ins(v)) \vee (v$ is an AMR)
   then $ar(v) \in \{\text{1st col. Tab. 3.8}\}$
   else
   $ar(v) \in \{\text{2nd col. Tab. 3.8}\}$
   (architecture role of new functions: FPG function instances, AMRs, and others)
6. $\forall v \in V_{M_D,v} \setminus V_{F,v}$:
   $(ar(v) \in \{\text{3rd col. Tab. 3.8}\}) \vee$
   $(\exists w \in V_{F,f}$ where $w = v$ except $ins(w) \neq ins(v)) \vee$
   $(v \in \{ter(e) \mid e \in E^+(\text{AMR})\})$
   (architecture role of new variables)

The coordinator node that is required by condition 2 provides the system-level inputs and collects system-level outputs. Conditions 4 and 5 concern the architecture roles of function nodes. Conditions 5 states that all nodes from the FPG, their instances, and AMRs get an architecture role from the same set as the architecture roles in condition 4. Other new function nodes also have an architecture role, but from a different set, see Tab. 3.8. Condition 6 describes that new variables generally get architecture roles assigned, except when they are added as instances or to serve as outputs of AMRs by the algorithm that imposes the MDAO architecture.

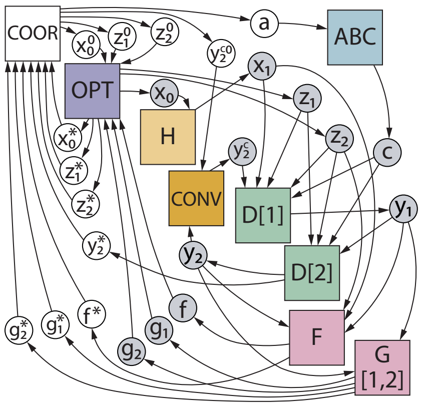
Based on the Sellar problem FPG given in Fig. 3.9d, KADMOS can impose any MDO architecture. Here, as an example, we illustrate the implementation of the MDF architecture, with a Gauss-Seidel convergence scheme, which requires the addition of a converger block to drive the MDA convergence cycle, see Algorithm 2 and Fig. 3.10a. More architectures are available in KADMOS, as discussed in §§3.5.2.

Table 3.8: Lists of possible architecture roles for function and variable nodes in an MDG

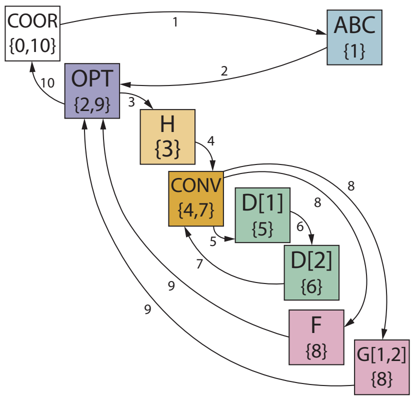
existing functions and AMRs	new functions	new variables
uncoupled-DVI	coordinator	final design variable
uncoupled-DVD	optimizer	final output
coupled	converger	final coupling
post-coupling	DOE	DOE output samples
	Surrogate Model (SM)	initial guess design variable
		initial guess coupling variable
		DOE input samples
		coupling copy
		design variable copy
		coupling weight
		SM approximate

Algorithm 2: MDG algorithm for MDF with Gauss-Seidel convergence scheme (Refer to Fig. 3.10a for the final graph and Fig. 3.10c for the XDSM)

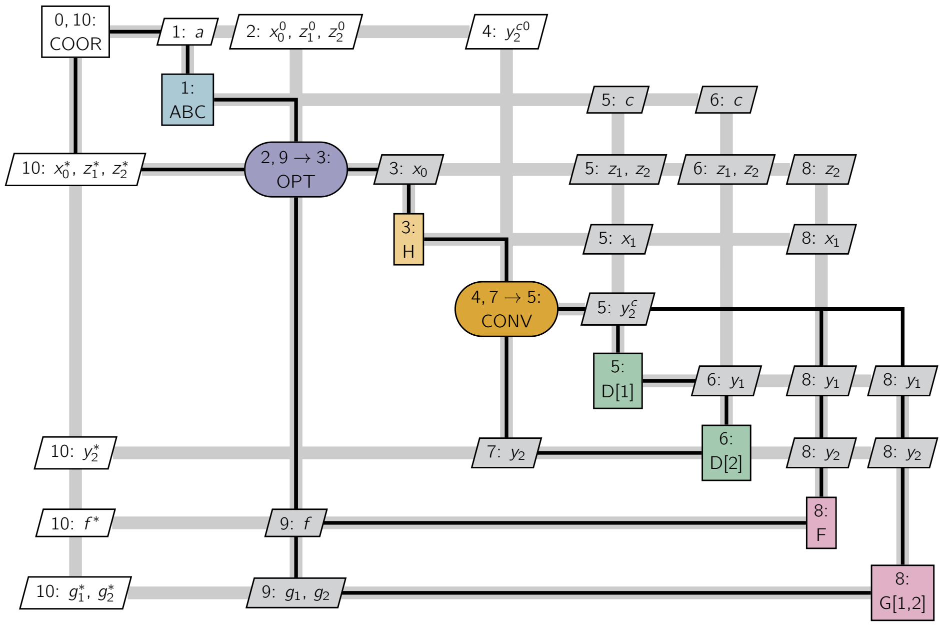
```
1.  Check FPG based on graph conditions.
2.  Copy the FPG as a starting point for the MDG.
3.  Add coordinator block (COOR).
4.  Add and connect the converger block (CONV) to the coupled functions.
 a: Remove feedback coupling between coupled functions ⇒ edge ($y_2$, D[1]) in the FPG.
 b: Connect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



a) MDG



b) MPG



c) Automatically generated XDSM of the combined MDG+MPG

Figure 3.10: MDAO graphs and XDSM view of the Sellar problem using the MDF architecture with a Gauss-Seidel convergence scheme

Algorithm 3: MPG algorithm for MDF with Gauss-Seidel convergence scheme (see Fig. 3.10b for the final graph and Fig. 3.10c for the XDSM)

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

3.4.4. MDAO PROCESS GRAPH

The MPG is the only process type graph discussed in this dissertation. This graph defines the process steps that are required to solve the MDAO problem based on the selected architecture, hence it contains the execution sequence of all the blocks in the architecture, including the necessary iteration cycles, their nesting, etc. An MPG is always based on an MDG. The MPG $M_P = (V_{M_P}, E_{M_P})$ should meet the following additional conditions with respect to the `ProcessGraph` class defined in §§3.2.3:

1. $V_{M_P} = V_{M_D,f}$ (graph nodes are MDG functions)
2. $\exists! v \in V_{M_P} : psn(v) = minpsn(M_P) \Rightarrow v = \text{COOR}$ (process starts at coordinator)
3. $\exists! v \in V_{M_P} : psn(v) = maxpsn(M_P) \Rightarrow v = \text{COOR}$ (process ends at coordinator)
4. $\forall v \in V_{M_P} :$
   $\exists$ cycle $C = (V_C, E_C)$ where $v \wedge \text{COOR} \in V_C$ (all nodes are part of a cycle that contains the coordinator)
5. $\exists$ cycle $C = (V_C, E_C) \in M_P$ for which
   $\{psn(e) \mid e \in E_C\} = [1, maxpsn(V_C)]$ (there is at least one cycle with continuously increasing step numbers)

The MPG corresponding to the earlier discussed MDG is automatically determined by the KADMOS algorithm provided in Algorithm 3.

3.5. AUTOMATED CAPABILITIES FOR GRAPH-BASED SYSTEMS

In this section, four novel capabilities that exploit the graph-based formalization of MDAO systems are briefly discussed: XDSM visualization, architecture reconfiguration, sequencing, and partitioning.

3.5.1. XDSM VISUALIZATION OF SYSTEM GRAPHS

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

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

The result is shown in the example in Fig. 3.10c. This XDSM is automatically created using the Python graph objects MDG and MPG and a graph-based extension of the LaTeX-based XDSM writer [S20] by Lambe and Martins [37]. The same method can also be used to solely visualize data graphs, which would result in 'XDSM data flow' visualizations. Without process information, such an XDSM would be equivalent to an N^2 chart.

Of the two graphs that describe an XDSM, the MDG will grow in size more quickly as the MDAO system becomes larger and more complex. Since the growing amount of off-diagonal information to be displayed would affect the readability of the XDSM visualization, KADMOS can be set to visualize just the number of connections, rather than the full list of exchanged I/O data. Hence instead of '4: x_1, z_1, z_2' as shown in Fig. 3.10c, KADMOS would only state '4: 3 connections'. This concise notation will be used in the case study (§3.7).

As discussed in the introduction, the formulation of an MDAO system in large, collaborative design projects can be severely impaired by a lack of proper visualizations. Debugging, documenting, exchanging information and, even more, maintaining oversight of the whole systems would easily become impossible, thereby compromising the success of the very MDAO initiative. The graph-based formalization provided by KADMOS appears to be a key enabler to such visualization, even beyond the KADMOS native XDSM generation ability. As discussed later in this dissertation (Chapter 6), its compact, structured and rigorous syntax provided the opportunity to develop an advanced dynamic visualization tool, called VISTOMS [S11], able to interactively display in the browser XDSMs of any size (as well as other type of useful visualizations) and inspect any single exchanged data, by toggling and expanding every XDSM block. Details on VISTOMS are provided by Aigner et al. [39].

3.5.2. Architecture reconfiguration

One of the motivations for describing the different stages of MDAO systems using a graph syntax, is that the system can be reconfigured very easily. Multiple algorithms are available in KADMOS to automatically determine the reconfigured data connections and process necessary to solve the optimization problem based on various MDO architectures. Hence, the same FPG in Fig. 3.9d can be used to reconfigure to MDF with a Jacobi convergence schema, or to impose IDF, or apply distributed architectures. Other architecture types (i.e. without optimization) would require small adjustments of the FPG, for example, a DOE strategy would be based on design variables, but instead of objective and constraint variables, it can only have QOIs. The following MDAO architectures are currently supported:

- MDA
    - without convergence, hence simply running a set of tools in sequence.
    - with convergence, based on either Gauss-Seidel or Jacobi scheme.
- DOE with one of the MDA set-ups embedded
- MDO
    - monolithic

        - ◇ MDF with a Gauss-Seidel or Jacobi convergence scheme
        - ◇ IDF
    - – distributed
        - ◇ CO
        - ◇ BLISS-2000

This section discusses the reconfiguration with CO for the Sellar case. In Fig. 3.10 the monolithic MDF architecture has been implemented on the FPG shown in Fig. 3.9d. Alternatively, the radically different CO strategy could also be used to solve the same problem. MDF and CO are just two of the standard MDAO architectures currently available in KADMOS, but new ones can be defined, possibly requiring the addition of new FPG graph manipulation methods. When CO was added to KADMOS, for example, its algorithms were built using a combination of the methods previously developed for monolithic architectures and newly developed methods, specific for distributed architectures (e.g. determination of global and local design variables and constraints, addition of new AMRs, etc.). With a growing library of MDAO architectures, the number of basic graph manipulation methods included in KADMOS also grows, making it easier to add new architectures (or variations of architectures) to the package.

The basic steps of the MDG creation for CO are summarized in Algorithm 4. The combination of MDG and MPG for the CO strategy is depicted in Fig. 3.11. Multiple concepts from the KADMOS graph syntax come forward in this small example. For example, many instances and copies of variables are created, especially for the global design variables z_1 and z_2, as these are used by functions in all three optimization cycles. In addition, also a function instance is created for the function H (see: H^{i1} in the sub-OPT-0 cycle), which is done in this case to keep the suboptimization cycle independent of the functions in the main Sys-OPT cycle. The algorithm also adds the AMRs J0 and J1, which are the consistency constraint/objective functions that are part of the CO formulation. Also note that all functions within the optimization cycles now have the architecture role 'uncoupled-DVD' functions, since the distribution of D[1] and D[2] through separate optimization cycles means that no convergence cycle is required within any optimizations, making the categories 'coupled' and 'post-coupling' irrelevant in this situation.

How other architectures from the aforementioned list can be used will be shown in the case studies in the two upcoming sections. These sections are dedicated to studies that validate (§3.6) and demonstrate (§3.7) KADMOS's ability to support the formulation phase for cases of representative complexity.

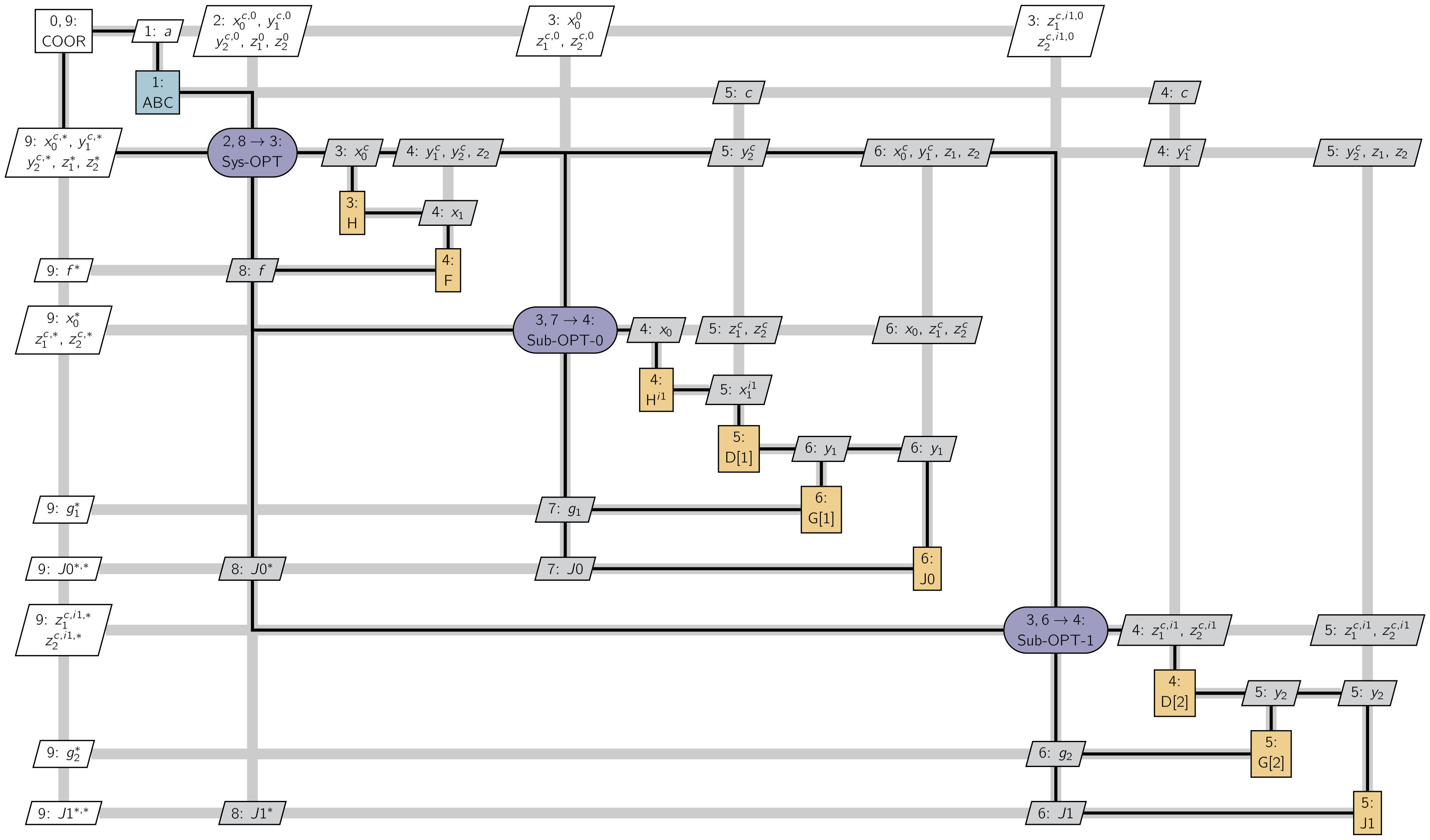

0, 9: COOR
1: a
1: ABC
2: $x_0^{c,0}$, $y_1^{c,0}$, $y_2^{c,0}$, z_1^0, z_2^0
3: x_0^0, $z_1^{c,0}$, $z_2^{c,0}$
3: $z_1^{c,i1,0}$, $z_2^{c,i1,0}$
5: c
4: c
9: $x_0^{c,*}$, $y_1^{c,*}$, $y_2^{c,*}$, z_1^*, z_2^*
2, 8 → 3: Sys-OPT
3: x_0^c
4: y_1^c, y_2^c, z_2
5: y_2^c
6: x_0^c, y_1^c, z_1, z_2
4: y_1^c
5: y_2^c, z_1, z_2
3: H
4: x_1
9: f^*
8: f
4: F
9: x_0^*, $z_1^{c,*}$, $z_2^{c,*}$
3, 7 → 4: Sub-OPT-0
4: x_0
5: z_1^c, z_2^c
6: x_0, z_1^c, z_2^c
4: H^{i1}
5: x_1^{i1}
5: D[1]
6: y_1
6: y_1
9: g_1^*
7: g_1
6: G[1]
9: $J0^{*,*}$
8: $J0^*$
7: $J0$
6: J0
9: $z_1^{c,i1,*}$, $z_2^{c,i1,*}$
3, 6 → 4: Sub-OPT-1
4: $z_1^{c,i1}$, $z_2^{c,i1}$
5: $z_1^{c,i1}$, $z_2^{c,i1}$
4: D[2]
5: y_2
5: y_2
9: g_2^*
6: g_2
5: G[2]
9: $J1^{*,*}$
8: $J1^*$
6: $J1$
5: J1

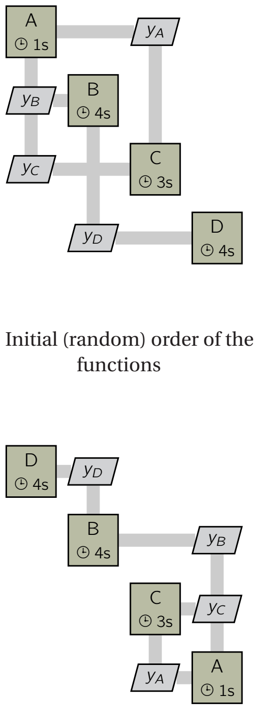
Figure 3.11: Automatically generated XDSM of the CO architecture imposed on the FPG shown in Fig. 3.9d

Algorithm 4: MDG algorithm for CO (Refer to Fig. 3.11 for the corresponding XDSM)

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

3.5.3. Sequencing algorithms

In step 7 of the FPG composition process (Algorithm 1), the execution sequence of the functions in the problem graph has to be specified. This is a trivial task for a small system involving a small number of functions and couplings. However, for larger systems that have a considerable number of interconnections, the best function order (where 'best' in this section means a minimum number of feedback variables) is not obvious neither unique. An automated capability to obtain a convenient function order saves time compared to adjusting that order manually, even for small systems. Therefore, sequencing algorithms have been implemented and tested in KADMOS [87].



a) Initial (random) order of the functions

b) Optimal order for minimum feedback

Figure 3.12: Illustrative example to explain sequencing algorithms

Two execution sequence properties are considered in the developed algorithms:

- number of feedback variables;
- execution time of a single sequence.

Fig. 3.12 depicts a small example of a sequencing problem. The order of the functions on the diagonal is assumed to be the execution order of the tools. In Fig. 3.12a a random order is provided. If this system needs to be converged, then three feedback variables need to be handled by the solver. A better sequence is provided in Fig. 3.12b, since only one feedback variable is left. The estimated execution times of the four functions are also indicated in the figure. In Fig. 3.12a, if the feedback variables are ignored, then functions A, B, and D can be executed directly and simultaneously; C can run after tool A has provided its required input. Hence, the total execution time of a single iteration sequence is 4 seconds. In Fig. 3.12b, D and C are started in parallel, followed by B and A. Thus the execution time is larger and equal to 9 seconds. Hence, the execution time of a single iteration is larger, but the sequence in Fig. 3.12b is still considered optimal, since feedback reduction is prioritized over iteration time minimization.

The goal of the sequencing algorithms is to *find the tool sequence that has the lowest number of feedback variables in combination with the shortest possible execution time of a single iteration for that number of feedback variables.* The following seven algorithms have been developed for this purpose:

- **brute-force:** determines all possible sequences and checks them individually.
- **branch-and-bound:** performs a smart tree search to find the best sequence.
- **swapping:** starting from a random sequence, iteratively adjusts the sequence based on a given swap type (see three options below) until the solution cannot be improved anymore. The three swap options are:
  - **single-swap:** swaps a single function node to a new location between two other functions.
  - **two-swap:** swaps the position of two function nodes.

- **hybrid-swap:** first performs two-swap until converged, then single-swaps to refine.
- **genetic algorithm:** employs a genetic algorithm implementation to solve the sequencing problem. This algorithm is used as a benchmark solution for large systems.

The sequencing algorithms were validated using a scalable MDAO system with randomized couplings, based on the variable complexity problem described by Zhang et al. [88]. Hundreds of MDAO systems were created automatically and sequenced using the seven algorithms above, changing the number of disciplines and couplings (the latter is also called the 'coupling density'). After validation, the performance of the different algorithms was assessed, which is shown in Fig. 3.13. These tests were performed for systems ranging from five to fifty disciplines. For each test setting (a combination of number of disciplines and coupling density), 200 randomly created MDAO systems were analyzed.

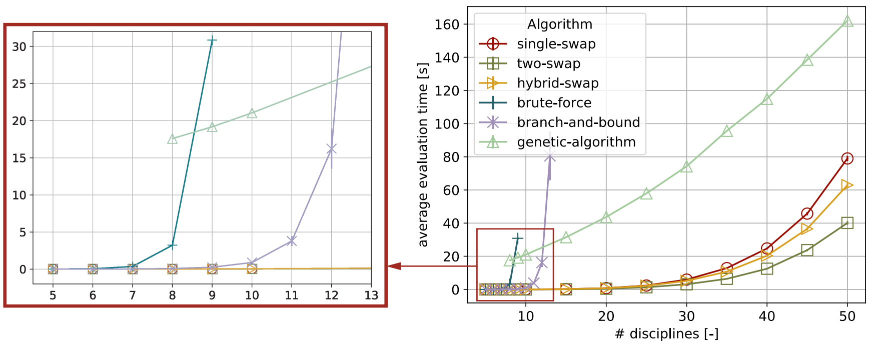


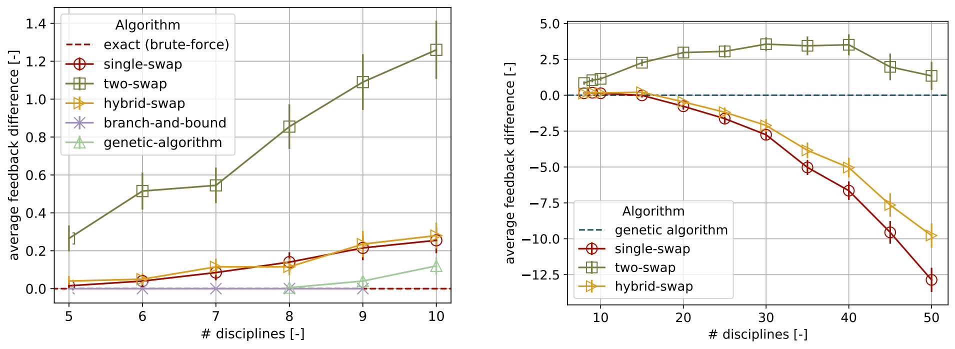
Figure 3.13: Performance of the different sequencing algorithms in terms of evaluation time [87]

Fig. 3.13 shows that the evaluation times increase quickly around ten disciplines for both the brute-force and branch-and-bound algorithms. This is to be expected, as the number of sequence options grow tremendously with each additional discipline. In the brute-force approach, the algorithm scales with $n_f!$, where n_f is the number of disciplines. Hence, at ten functions there are already $\sim 3.6 \cdot 10^6$ options to consider. The branch-and-bound algorithm reduces the number of options with the smart tree search, but the execution time still increases exponentially. However, the two aforementioned algorithms are still advantageous for small systems, since they will always provide the exact solution for the sequencing problem. Thus, for these systems the algorithms are still valuable in practice and they are also useful to benchmark the other algorithms.

The swapping algorithms offer a more scalable performance. All three swapping approaches are quite close in Fig. 3.13. The two-swap algorithm is fastest, the single-swap slowest, while the hybrid-swap performs between them. Though the performance of the swapping algorithms is more scalable with system size, their main disadvantage is the fact that they will not necessarily return the global optimum. From their starting sequence, these algorithms will keep swapping nodes until no improvement can be achieved anymore. Thereby, the nature of these algorithms does not guarantee that the best sequence is actually encountered in the iterative process. Hence, the next question to answer is which of these algorithms will provide the best results by obtaining the solution with the lowest number of feedback variables and lowest execution time of an iteration.

The sixth algorithm, genetic algorithm, was implemented to assess the results for large systems where brute-force and branch-and-bound methods cannot be used. As shown in Fig. 3.13, this algorithm clearly lacks a performance advantage, but it carries out a more distributed search of the design space with its evolving population of solutions. This makes the algorithm a good candidate to check the optimality of the swapping algorithms results.

The results of the sequencing algorithms for a range of systems are summarized in Fig. 3.14. Fig. 3.14a depicts the average feedback difference for small systems, where the brute-force algorithm is taken as the benchmark. Each point in this plot represents 200 analyses of randomized systems. The average feedback difference is calculated by taking the mean of the difference in the number of feedback variables obtained by a certain algorithm and the exact solution of the benchmark for the 200 systems that are analyzed per point. As expected, the branch-and-bound approach always finds the benchmark solution and the average feedback difference is zero. The swapping algorithms are not guaranteed to find the same solution, with the two-swap approach performing worst. The two-swap algorithm pays for it better evaluation time (Fig. 3.13) by returning less optimal (i.e. larger number of feedback variables) solutions (Fig. 3.14). This is attributed to the fact that the algorithm always has to swap two functions, meaning that it is not always able to find the same solution as the single-swap approach.



a) small MDAO systems

b) large MDAO systems

Figure 3.14: Benchmarking results for small (left) and large (right) MDAO systems [87]

Fig. 3.14b shows the average feedback difference for large systems. Here, the genetic algorithm is used as the benchmark. The brute-force and branch-and-bound solutions cannot handle this system size, so only the swapping algorithms are assessed. The two-swap algorithm is not able to find the same or better solutions in terms of feedback variable minimization as the genetic approach, while the single- and hybrid-swap approaches actually outperform it. Thus, for larger systems, the single- and hybrid-swap approaches are advantageous compared to a genetic algorithm, with respect to both performance and result optimality.

In conclusion, based on these results which are representative of the broader algorithm assessment (more details can be found in [87]), the sequencing method implemented in KADMOS automatically selects the algorithm to be used as follows:

- ≤ 10 coupled disciplines $\rightarrow$ branch-and-bound
- > 10 coupled disciplines $\rightarrow$ single-swap

The sequencing method is used to automatically determine the function order with minimal feedback variables and lowest execution time of a single iteration for systems up to fifty disciplines. In addition, the method is also used in the decomposition algorithm, as will be discussed in the next section.

3.5.4. Decomposition algorithm

If the computational environment support this (i.e. multi-core processing), it might be advantageous to decompose the functions of a large MDAO system in several smaller parallel function groups. Distributed MDO architectures also require the system to be decomposed to obtain a convenient distribution of the disciplinary functions. The graph-based system representation obtained by KADMOS lends itself very well for such decomposition using partitioning algorithms.

Together with Bruggeman, a decomposition method for KADMOS was developed [87] using the Metis graph partitioning algorithm by Karypis and Kumar [89]. The method is called Metis-based Decomposition of KADMOS graphs (MDK). Metis itself cannot be used directly, since additional graph analysis and manipulations are required to take care of the mismatch between the FPG used by KADMOS to describe the MDAO problem and the graph that is required by Metis to perform partitioning. Additionally, the specific decomposition objective for the KADMOS graphs also has to be translated to the right Metis format. This decomposition objective is:

> *to find the best decomposition of the system that balances the required coordination between partitions (i.e. minimize the couplings between partitions) and the execution time of each partition.*

This objective is translated to the following function that needs to be minimized:

$$f = f_b \cdot \frac{n_{ce} + n_{fb}}{n_c} + (f_b - 1) \cdot \frac{t_{sys}}{\sum_{i=1}^{n_f} t_i} \tag{3.1}$$

in which:

f_b :	balance factor ($0 \leq f_b \leq 1$) to prioritize coupling or runtime minimization
n_{ce} :	total number of edges cut by the partitioning algorithm
n_{fb} :	total number of feedback variables within the partitions
n_c :	total number of couplings in system
t_{sys} :	total runtime when partitions are executed in parallel
$\sum_{i=1}^{n_f} t_i$:	total runtime of all functions summed individually

The first term in the objective function represents the number of couplings in the decomposed system and the second term is the normalized execution time of the decomposed system. The balance factor is used for both terms and is set by the user to have a value between zero and one. This factor controls whether the decompositioning will aim at balancing the couplings between partitions ($f_b = 1$, second term in Eq. (3.1) is always zero), at balancing the runtime of the partitions ($f_b = 0$, first term in Eq. (3.1) is always zero), or a hybrid aim in between.

The process flow of the MDK method is depicted in Fig. 3.15 accompanied by a small example. The following steps are performed in the method:

1. Inputs have to be provided to start the process flow:
   - FPG to be decomposed.
   - MDAO architecture that the system has to be decomposed for. Depending on the architecture, different functions from the FPG are selected for decompositioning, see step 2.
   - Balance factor (f_b) to be used.
   - Number of partitions that is required. This should match with the number of cores that is available in the computational environment.
2. Based on the provided architecture, MDK determines which functions need to be included in the partitioning graph. For example, with an MDF architecture only the functions with the problem role 'coupled' would be included, while for IDF both the 'coupled' and 'post-coupling' functions need to be part of the partitioning. In the example provided in Fig. 3.15 all functions are coupled and need to be part of the partitioning.
3. The FPG is translated to a graph that can be handled by Metis. Metis can only handle simple, undirected graphs. The partitioning can be influenced by adding weights to the nodes and edges, where Metis will try to balance the total node and edge weights of each partition. The following two steps result in a Metis graph:
   (a) Function nodes are copied to an empty undirected graph. The execution time is added as the node weight.
   (b) The couplings between two functions are summed and an edge is created if there is at least one coupling. The edge weight is set to the number of couplings between the two functions.

   A translation example from FPG to Metis graph is shown in the figure.
4. The graph is partitioned using Metis. In the example, two partitions are requested and the edge between functions D3 and D7 is cut.
5. The provided partition needs to be analyzed further. First, the functions in each partition are sequenced using the sequencing method from §§3.5.3.
6. The objective value can be calculated. Note that due to the applied sequencing the actual execution time of a partition can be lower than the sum of the node weights, since some of the functions can be run in parallel, such as D3 and D2 in the example.
7. MDK checks for convergence by considering the last x iterations. This is a setting of the method, which defaults to 3.
8. Since Metis is not aware of the function order and considers summed node weights to represent the execution time of a partition, something needs to be done to take parallel execution of the functions into account. This is why parallel nodes are merged at the beginning of the second iteration. The node weight is then set to the largest execution time among the parallel nodes.
9. As shown in the last column of Fig. 3.15, the merged parallel nodes result in a new graph to be partitioned, which has an improved objective ($f = 0.40$ instead of 0.42).
10. This process is iterated until the objective can no longer be improved.

The MDK method was verified and validated using the same randomized scalable problem used for the sequencing algorithms in the previous section. A benchmark method was developed based on a brute-force approach: this method determines all possible partitions for a given system and calculates the objective value for each option. The

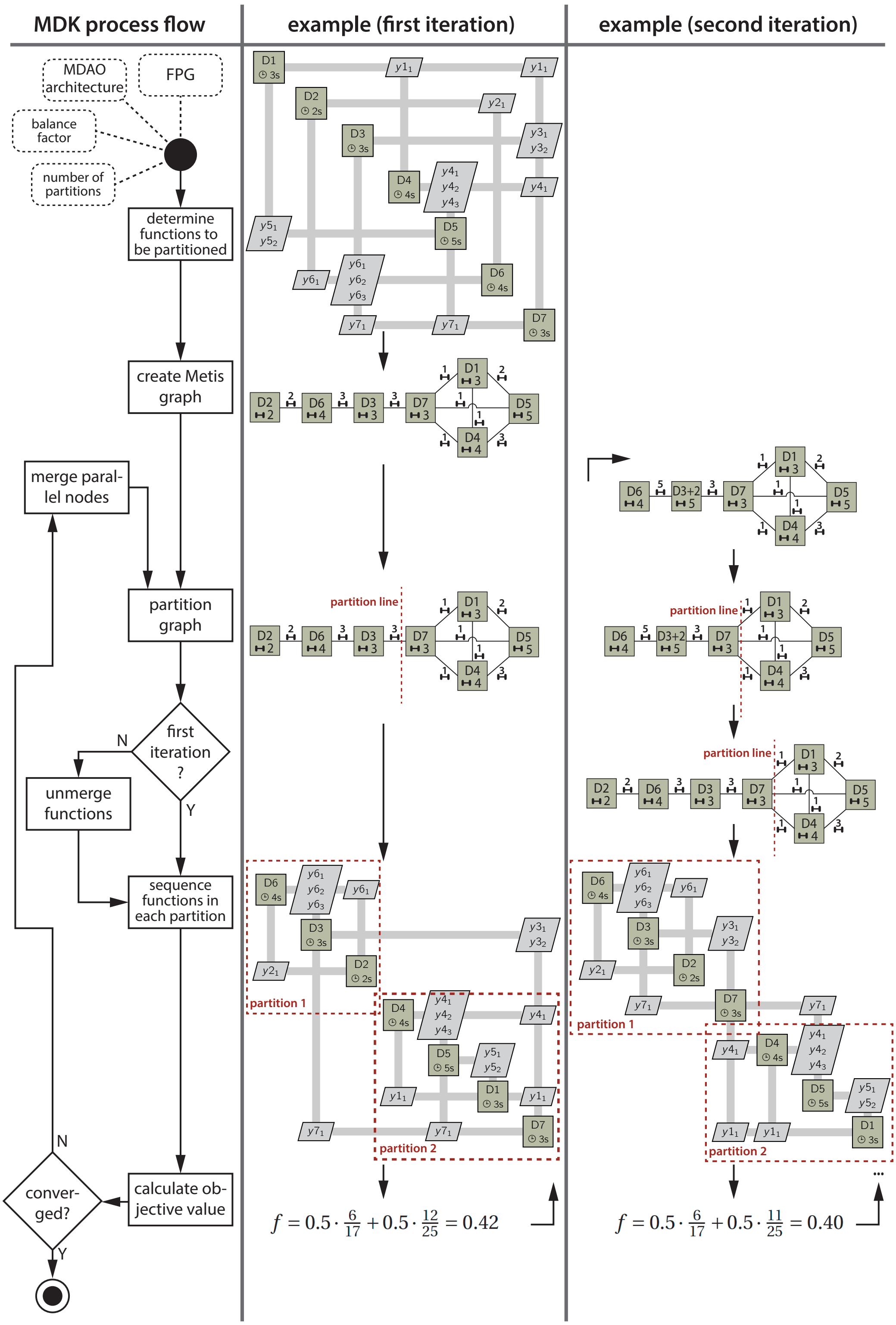
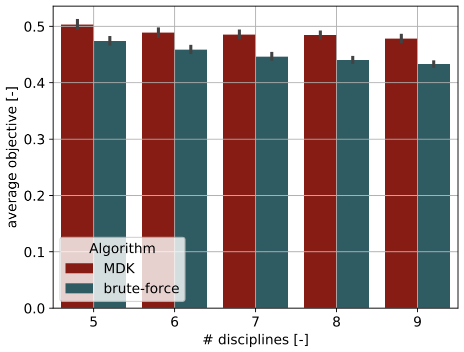
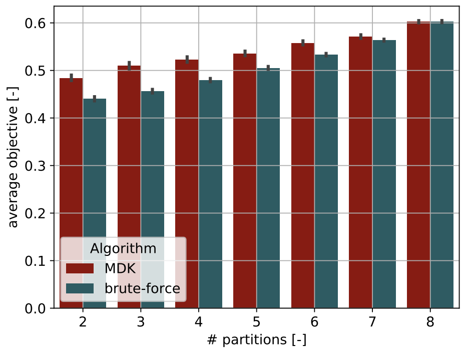

MDK process flow
example (first iteration)
example (second iteration)
MDAO architecture
FPG
balance factor
number of partitions
determine functions to be partitioned
create Metis graph
merge parallel nodes
partition graph
first iteration ?
N
Y
unmerge functions
sequence functions in each partition
calculate objective value
converged?
partition line
partition 1
partition 2
$f = 0.5 \cdot \frac{6}{17} + 0.5 \cdot \frac{12}{25} = 0.42$
$f = 0.5 \cdot \frac{6}{17} + 0.5 \cdot \frac{11}{25} = 0.40$

Figure 3.15: Process flow (left) of the MDK decomposition method implemented in KADMOS with a small illustrative example for two iterations (middle and right)

brute-force approach can only be used for small systems up to nine disciplines, as the number of options grows exponentially with the number of system functions. The comparison between MDK and the brute-force approach is shown in the bar plots in Fig. 3.16. For each bar 200 randomized systems have been decomposed and the average objective value is shown. The bar plot in Fig. 3.16a shows the results when a increasing number of disciplines is decomposed in two groups, while in Fig. 3.16b the number of partitions is varied for systems with eight disciplines. The MDK method is not able to achieve the same objective values as the brute-force approach, which is related to the aforementioned limitations of translating the FPG to a Metis graph (for more details, see [87]), but it is much faster.



a) varying the number of disciplines that are partitioned in two groups



b) varying the number of partitions for a system with eight functions

Figure 3.16: Typical validation bar plots for the MDK method with respect to a brute-force approach for small systems [87]

This performance gain of the MDK method is shown in Fig. 3.17. The method is able to handle much larger MDAO systems, up to fifty functions, wheres the brute-force approach cannot handle systems with more than nine. In conclusion, MDK will not guarantee to find the most optimal decomposition of a system, but it is able to provide convenient decompositions for a wide range of system sizes.

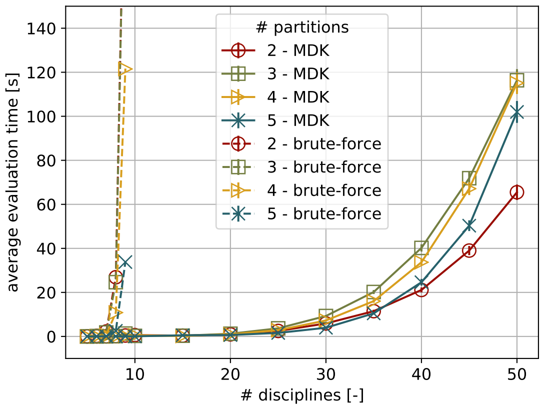


Figure 3.17: Performance of the MDK and brute-force methods for a range of MDAO system sizes [87]

3.6. Validation case study: supersonic business jet

The SuperSonic Business Jet (SSBJ) case is a small MDAO system developed by Sobieski et al. [48] to test different BLISS architecture schemes. The system is used to perform the conceptual design of a business jet by optimizing its cruise segment accounting for four disciplines: structures, aerodynamics, propulsion, and performance. In this dissertation, the SSBJ case is used to verify new developments at each stage of the collaborative MDAO design process, including executable workflow instances in multiple PIDO platforms (to be discussed in Chapter 5). The SSBJ disciplines, described in [90] and originally implemented as MATLAB code, were rewritten and published in Python by Dubreuil and Lafage [S21]. Subsequently, this Python code was used to create a CDS-compatible version of the SSBJ case, which has been published as open-source code [S22] as well. This version of the SSBJ system is used throughout the rest of the dissertation.

3.6.1. Stage I: Tool repository

The connectivity of the repository is depicted in Fig. 3.18 based on the RCG. Of the four main disciplines, three have circular dependencies (structures, aerodynamics, propulsion). In addition, several mathematical relations are added to serve as constraints and objective functions in optimization problems. An additional discipline was added (Dpdx) by extracting the pressure ratio (dpdx) calculation from the aerodynamic analysis, as this value only depends on a single (global) variable (t/c), which is important for correctly positioning it when distributed architectures are implemented. The following mathematical relations are included in the system:

$$\begin{aligned}
\mathrm{G}_\sigma &\Rightarrow \tilde{\sigma}_i = \sigma_i - 1.09 \quad \text{for } i = [1,5] \\
\mathrm{G}_\Theta &\Rightarrow \tilde{\Theta} = \Theta - 1.04 \\
\mathrm{G}_{\mathrm{dpdx}} &\Rightarrow \tilde{dpdx} = dpdx - 1.04 \\
\mathrm{G}_{\mathrm{prop}} &\Rightarrow \tilde{ESF} = ESF - 1.5 \\
&\Rightarrow \tilde{Temp} = Temp - 1.02 \\
&\Rightarrow \tilde{DT} = DT \\
\mathrm{G}_{L\sim W_T} &\Rightarrow \tilde{g}_{L\sim W_T} = \frac{L - W_T}{k_{L\sim W_T}} \\
\mathrm{F}_R &\Rightarrow \tilde{R} = -\frac{R}{k_R}
\end{aligned}$$

In which k denotes a scaling factor. Multiple MDAO problems can be formulated based on a single repository, as will be shown for the other case study in §3.7. Here, only the original optimization problem is considered.

3.6.2. Stage II: MDO problem

The SSBJ optimization problem, used for validating the developments in this dissertation, can be stated as follows:

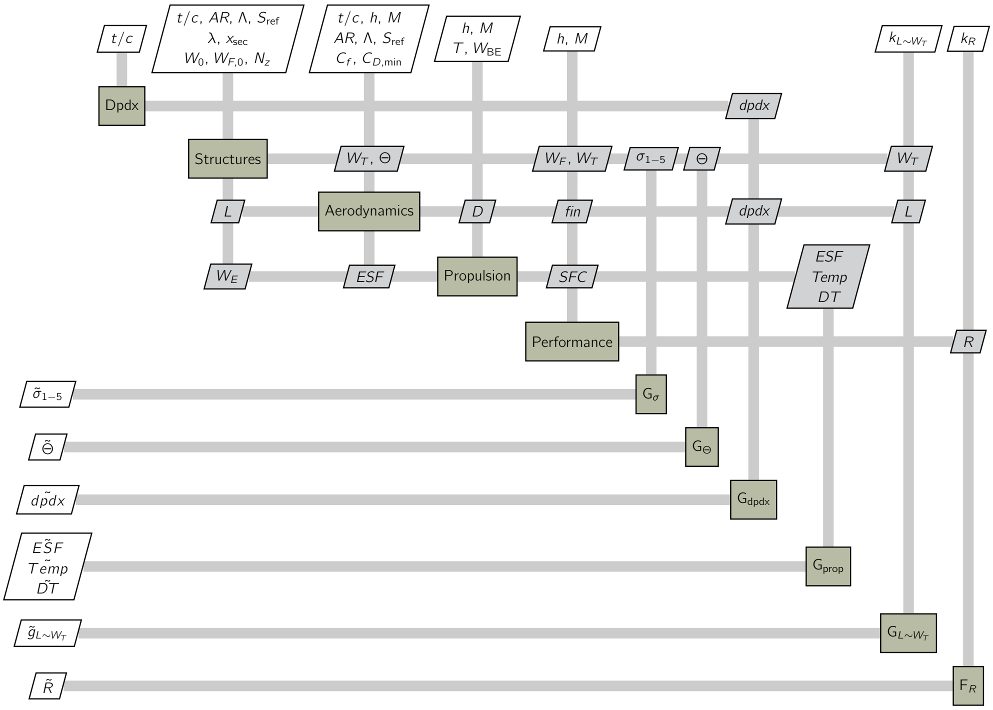


Figure 3.18: Automatically generated XDSM data flow (N^2 chart) visualization of the RCG for the SSBJ case

$$
\begin{aligned}
\text{minimize: } & \tilde{R} \\
\text{with respect to: } & t/c, h, M, AR, \Lambda, S_{\text{ref}}, \lambda, x_{\text{sec}}, C_f, T \\
\text{subject to: } & \tilde{\sigma}_i \leq 0.0 \quad \text{for } i = [1,5] \\
& 0.08 \leq \tilde{\Theta} \leq 0.0 \\
& d\tilde{p}dx \leq 0.0 \\
& -2.0 \leq E\tilde{S}F \leq 0.0 \\
& Te\tilde{m}p \leq 0.0 \\
& \tilde{DT} \leq 0.0 \\
& \tilde{g}_{L\sim W_T} = 0.0
\end{aligned}
$$

The last constraint is only used to ensure consistency with distributed architectures. The XDSM data flow of the FPG is depicted in Fig. 3.19.

3.6.3. Stage III: Solution strategies

Using KADMOS, different MDO architectures can be imposed on the problem depicted in Fig. 3.19. Two options are shown in Fig. 3.20 and Fig. 3.21: MDF with a Jacobi convergence scheme and IDF. Solution strategies based on other architectures are shown in Appendix A.

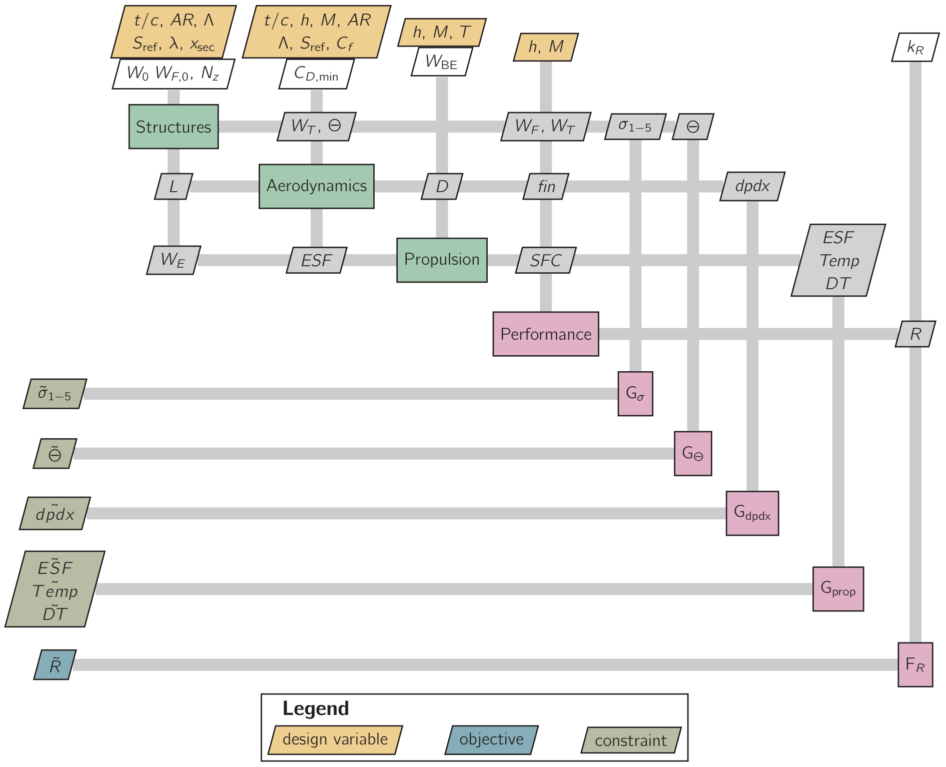
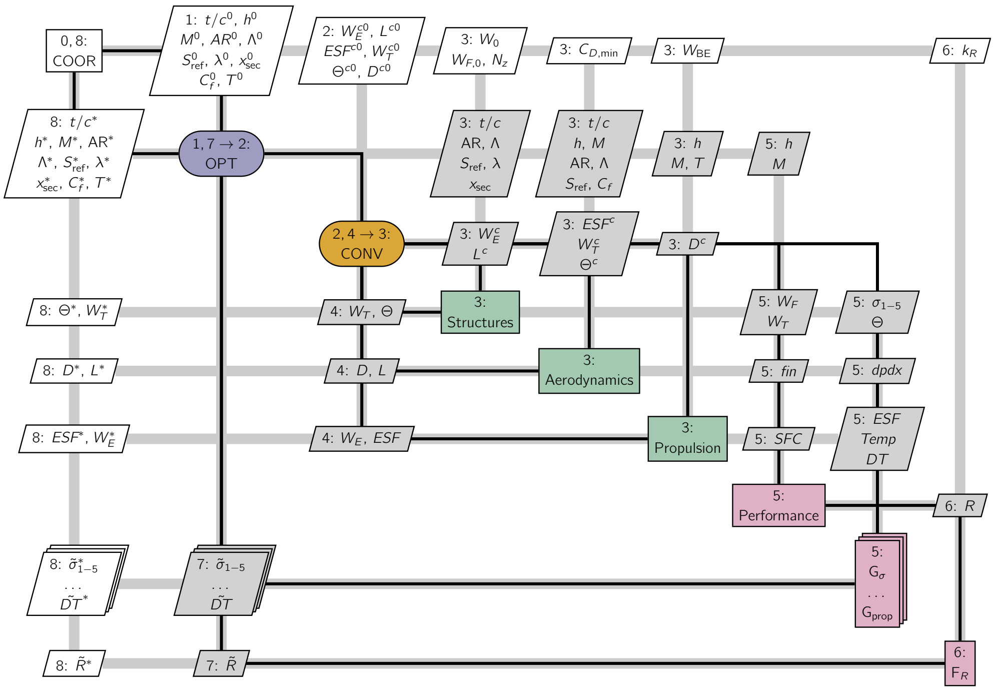
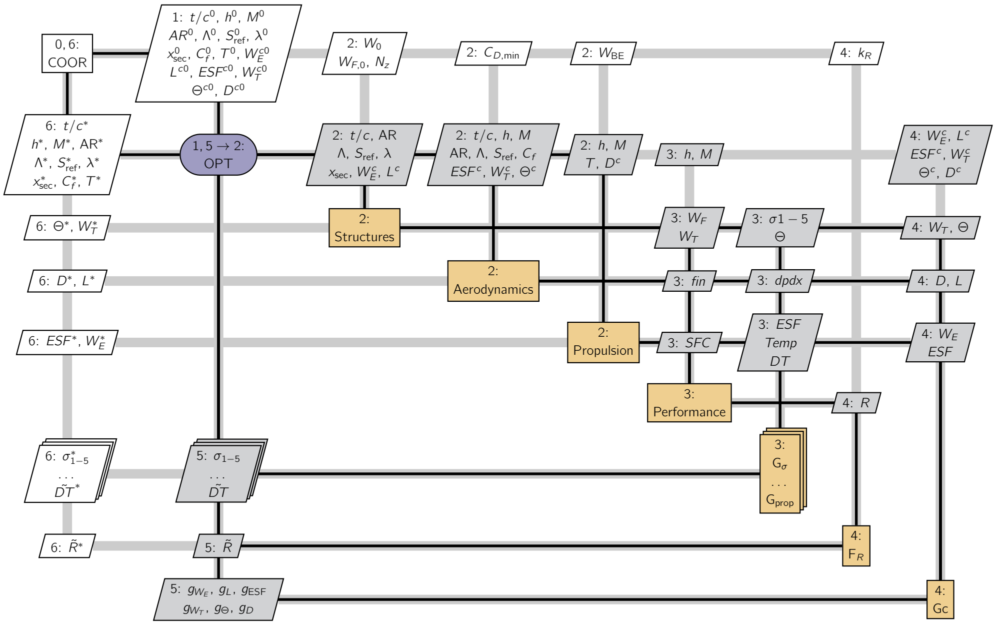


Figure 3.19: XDSM data flow (N^2 chart) visualization and marked variables of the FPG for the SSBJ optimization problem



0, 8: COOR
1: t/c^0, h^0 M^0, AR^0, Λ^0 S^0_{ref}, λ^0, x^0_{sec} C^0_f, T^0
2: W^{c0}_E, L^{c0} ESF^{c0}, W^{c0}_T Θ^{c0}, D^{c0}
3: W_0 $W_{F,0}$, N_z
3: $C_{D,\text{min}}$
3: W_{BE}
6: k_R
8: t/c^* h^*, M^*, AR^* Λ^*, S^*_{ref}, λ^* x^*_{sec}, C^*_f, T^*
1, 7 → 2: OPT
3: t/c AR, Λ S_{ref}, λ x_{sec}
3: t/c h, M AR, Λ S_{ref}, C_f
3: h M, T
5: h M
2, 4 → 3: CONV
3: W^c_E L^c
3: ESF^c W^c_T Θ^c
3: D^c
8: Θ^*, W^*_T
4: W_T, Θ
3: Structures
5: W_F W_T
5: σ_{1-5} Θ
8: D^*, L^*
4: D, L
3: Aerodynamics
5: fin
5: $dpdx$
8: ESF^*, W^*_E
4: W_E, ESF
3: Propulsion
5: SFC
5: ESF $Temp$ DT
5: Performance
6: R
8: $\tilde{\sigma}^*_{1-5}$ . . . $\tilde{DT}^*$
7: $\tilde{\sigma}_{1-5}$ . . . $\tilde{DT}$
5: G_σ . . . G_{prop}
8: $\tilde{R}^*$
7: $\tilde{R}$
6: F_R

Figure 3.20: XDSM visualization of the solution strategy MDF with a Jacobi convergence scheme for the SSBJ optimization problem



0, 6: COOR
1: t/c^0, h^0, M^0 AR^0, Λ^0, S^0_{ref}, λ^0 x^0_{sec}, C^0_f, T^0, W^{c0}_E L^{c0}, ESF^{c0}, W^{c0}_T Θ^{c0}, D^{c0}
2: W_0 $W_{F,0}$, N_z
2: $C_{D,\text{min}}$
2: W_{BE}
4: k_R
6: t/c^* h^*, M^*, AR^* Λ^*, S^*_{ref}, λ^* x^*_{sec}, C^*_f, T^*
1, 5 → 2: OPT
2: t/c, AR Λ, S_{ref}, λ x_{sec}, W^c_E, L^c
2: t/c, h, M AR, Λ, S_{ref}, C_f ESF^c, W^c_T, Θ^c
2: h, M T, D^c
3: h, M
4: W^c_E, L^c ESF^c, W^c_T Θ^c, D^c
6: Θ^*, W^*_T
2: Structures
3: W_F W_T
3: $\sigma 1-5$ Θ
4: W_T, Θ
6: D^*, L^*
2: Aerodynamics
3: fin
3: $dpdx$
4: D, L
6: ESF^*, W^*_E
2: Propulsion
3: SFC
3: ESF $Temp$ DT
4: W_E ESF
3: Performance
4: R
6: $\tilde{\sigma}^*_{1-5}$ . . . $\tilde{DT}^*$
5: σ_{1-5} . . . $\tilde{DT}$
3: G_σ . . . G_{prop}
6: $\tilde{R}^*$
5: $\tilde{R}$
4: F_R
5: g_{W_E}, g_L, g_{ESF} g_{W_T}, g_Θ, g_D
4: Gc

Figure 3.21: XDSM visualization of the solution strategy IDF for the SSBJ optimization problem

3.7. Demonstration case study: wing design

In §3.4 the simple Sellar problem was used as a means to explain the KADMOS functionality, its graph syntax and definitions. However, the real KADMOS value stands in its ability to support the formulation and reconfiguration of large scale (in terms of number of tools and the size of the design team involved) MDAO systems, of realistic complexity and relevance for the industry. This is the goal of the aerostructural wing design case study presented in this section. This test case uses a multidisciplinary, distributed tool repository constituted by a mix of proprietary design tools, mostly working as black boxes (i.e. aerodynamic solver, weight estimation tool, etc.), which have been linked together using the CDS approach. The aircraft configuration considered in this case study is a conventional passenger jet used in one of the AGILE project design campaigns. In this case study it is assumed that an MDAO integrator is creating a script for the design team to formulate and reconfigure the MDAO solution strategy using KADMOS methods. These reconfigurations are representative of a typical MDAO development process, as described by Piperni et al. [11] and discussed in §2.4, where an initial design point is determined first through a multidisciplinary convergence study, then design space exploration is performed using a DOE to assess the sensitivity of the design to some parameters of interest, to finally set up the actual optimization problem.

3.7.1. Tool repository

The collection of tools available in the database is summarized in Tab. 3.10. This includes twelve disciplinary tools developed by different experts, some of which featuring multiple execution modes, thus representing a truly heterogeneous repository. All tools have been made compatible with the CDS standard CPACS. As long as this compatibility is respected, any extra tool can be added to the repository in a straightforward manner. After the database has been imported, an RCG is created by KADMOS containing 2,909 nodes and 12,068 edges. By taking advantage of the mode attribute (see Fig. 3.4) the twelve tools in the database lead to 25 function nodes in KADMOS. Due to the sheer number of nodes and edges, the obtained RCG cannot be visualized as a graph because of obvious readability issues. The XDSM data flow with summarized connections can be used instead, as depicted in Fig. 3.22, although limited to a subset of the database.

Table 3.10: Tool repository

tool name	description	execution method	modes	mode description
HANGAR	Tool loads CPACS file with aircraft geometry. Used to have distinction between tool settings and aircraft design in XDSM.	local	AGILE AC	Conventional aircraft design created in the AGILE project
			AGILE AC wing	Adjusted wing of the AGILE conventional design for case study.
INITIATOR [91]	TLAR-based aircraft design creation.	local	main	-
SCAM	Simplified CPACS Aircraft Morphing. Adjust aircraft geometry in different ways.	local	λ	Wing taper morph
			Λ	Wing sweep morph
			Γ	Wing dihedral morph
			c_r	Wing root chord morph
			b	Wing length morph
			ξ	Wing spar position change
GACA	Geometrical Analysis of CPACS Aircraft components.	local	S_{wing}	Calculate wing reference area
			V_{FT}	Calculate wing fuel tank volume
Q3D [92]	Quasi-3D Aerodynamic solver.	remote	VDE	Viscous Drag Estimation
			FLC	Flight Load Case (vortex lattice meth.)
			APM	Aeroperformance Map
EMWET [93]	Wing mass estimation tool.	remote	main	-
SMFA	Simplified Mission Fuel Analysis.	remote	main	-
PHALANX [94]	Flight dynamics toolbox.	remote	Full Lookup	Full = full dynamic model
			Full Simple	Simple = empirical engine deck
			Symm. Lookup	Symm. = only longitudinal dynamics
			Symm. Simple	Lookup = external engine deck
PROTEUS [95]	Aeroelastic wing analysis tool.	remote	main	-
MaCal	Mass Calculation for MTO and ZF.	remote	main	-
OBJ	Normalized objective function.	script	main	$f_{\text{MTOM}} = m_{\text{MTO}} / m_{\text{MTO,ref}}$
CNSTRNT	Constraint value analysis.	script	WL (wing loading)	$c_{\text{WL}} = (m_{\text{MTO}} / S_{\text{wing}}) - \text{WL}_{\text{ref}}$
		script	FTV (fuel tank vol.)	$c_{\text{FT}} = m_{\text{fuel}} / (\rho_{\text{fuel}} \cdot \eta_{\text{FT}}) - V_{\text{FT}}$

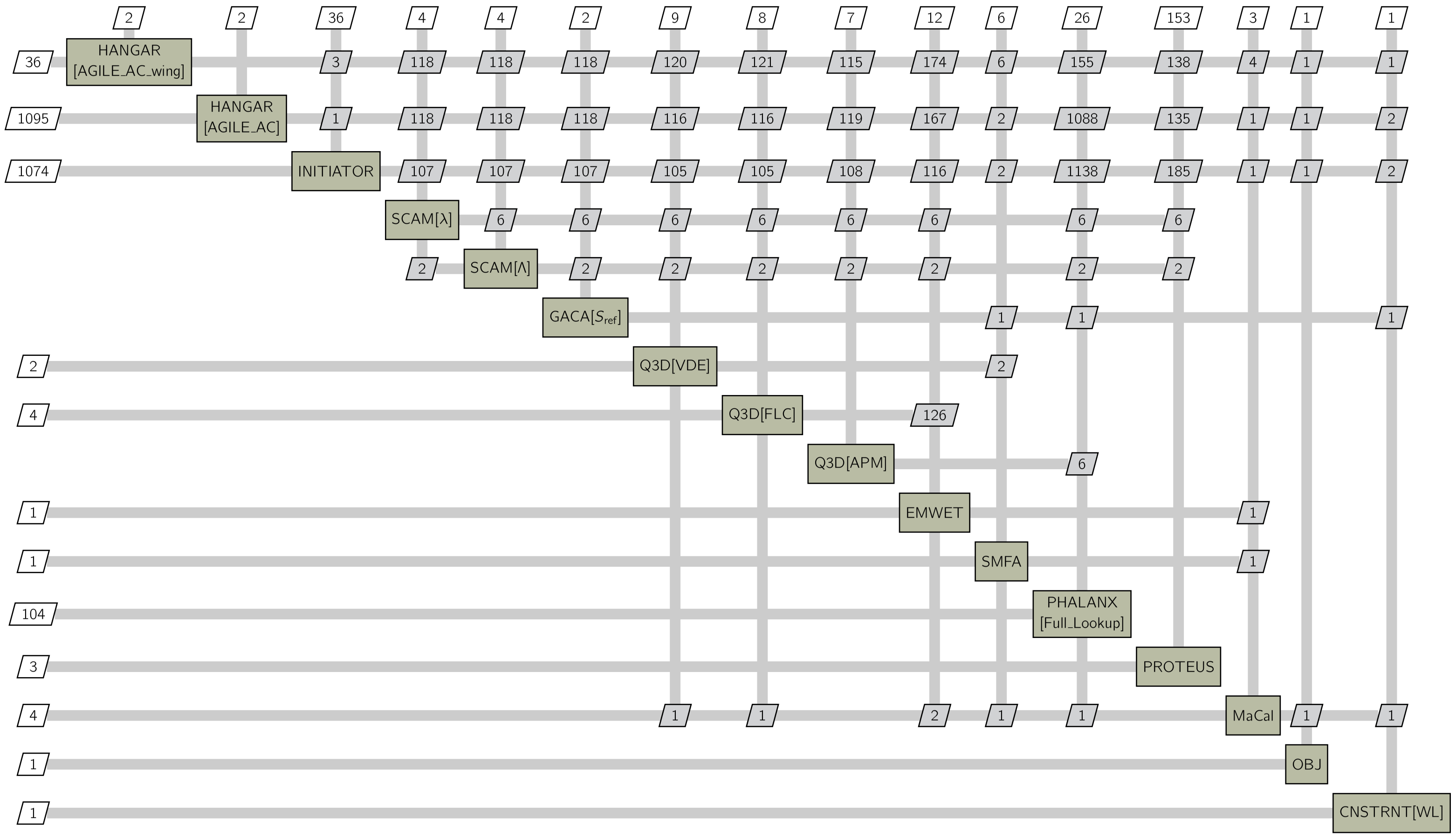


Figure 3.22: Automatically generated XDSM data flow (N^2 chart) visualization of a subset of the RCG (not all tool modes are shown to maintain readability). Numbers inside parallelograms indicate the number of inputs (top row), outputs (left column) and connections (off-diagonal)

3.7.2. Initial design point (design convergence study)

In a realistic design project the team would not directly jump to setting up the aerostructural optimization that they have in mind. Rather, the tools need to be tested first and a converged initial design point is required to start any optimization study. As discussed in §3.1, a design convergence study is one of the possible MDAO architectures supported by KADMOS. Following the algorithm suggested in §§3.4.2, the steps leading to the generation of the FPG are given here below, while the generated graph is depicted in Fig. 3.23a:

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

3a. All functions that do not provide any QOI and that are not coupled to tools providing them are automatically removed. In this case functions such as PHALANX, PROTEUS, OBJ, CNSTRNT, Q3D[APM] are removed from the RCG.

3b. For the initial geometry, the HANGAR function with mode AGILE_AC_wing is selected. The other HANGAR mode and the INITIATOR function are both removed.

3c. The HANGAR and SCAM functions still cause collisions, since both write to the same wing geometry elements. At this stage, the HANGAR tool is selected and SCAM is removed.

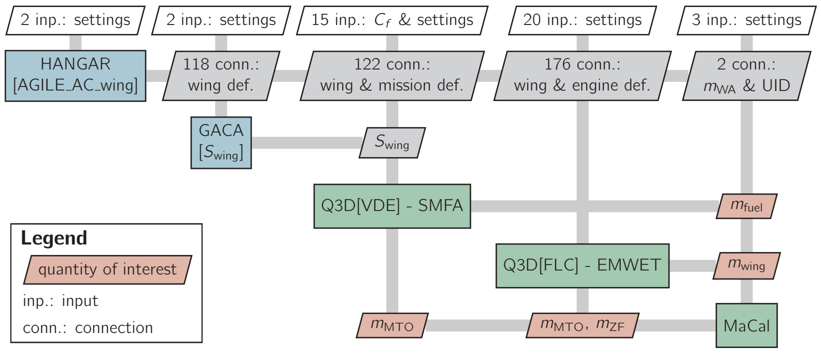
4. Graph is found to be valid by KADMOS on all necessary FPG conditions (see §§3.4.2).

5a. Q3D[VDE] and SMFA are merged sequentially to one function node.

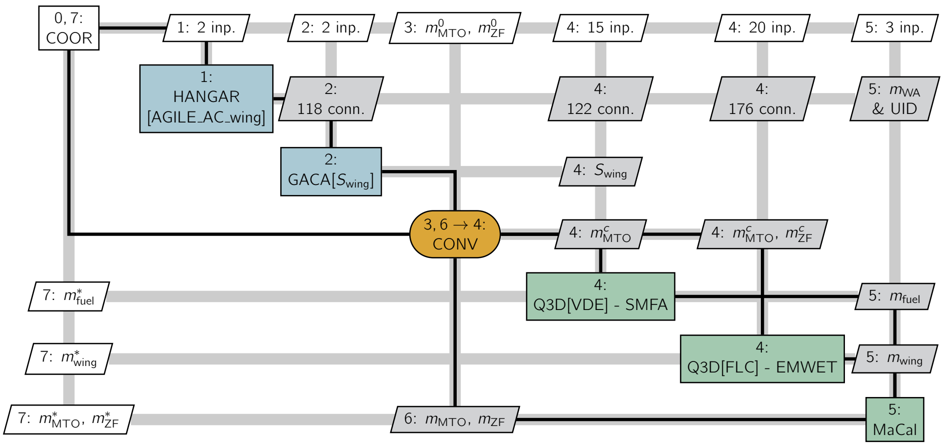
5b. Q3D[FLC] and EMWET are merged sequentially to one function node.

6-7. Function problem roles and order are set as shown in Fig. 3.23a.

Imposing the MDA with a Gauss-Seidel convergence scheme architecture on this FPG results in the solution strategy shown in Fig. 3.23b.



a) XDSM data flow of the FPG for design convergence study



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

Figure 3.23: Automatically generated XDSM visualizations of the KADMOS graphs for the first MDA convergence study

3.7.3. Design space exploration (DOE)

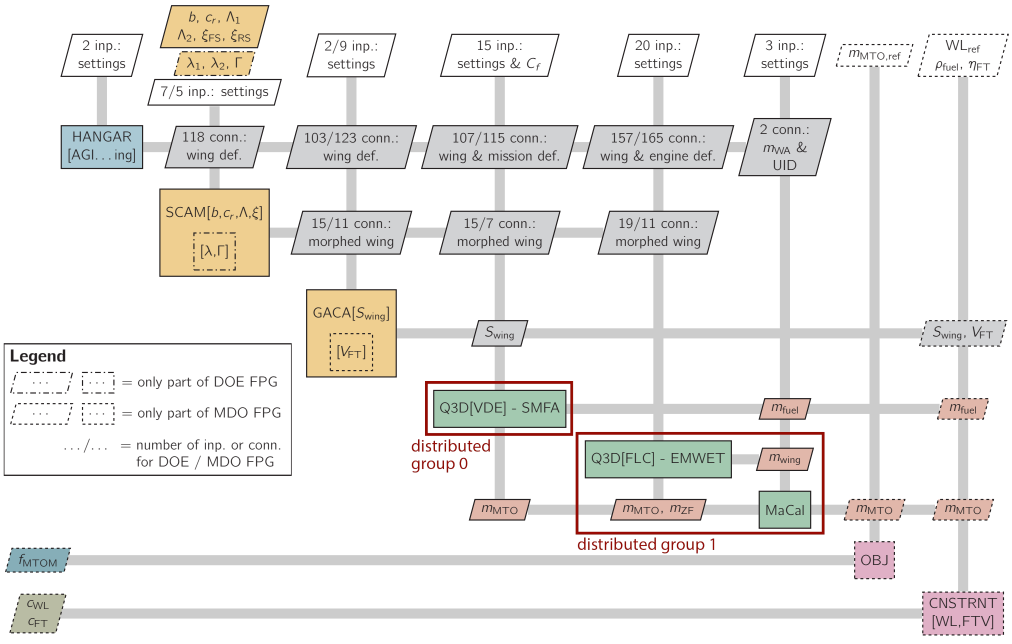
All the stages of the MDAO development process shown in Fig. 3.1 were covered by the MDA convergence study discussed in the previous section. At this point an iteration is triggered to reconfigure the MDAO system such to perform design space exploration through DOE. The following steps have to be taken to reconfigure the previously generated FPG:

1. Change to MDAO architecture type DOE.
2. QOIs remain and the following design variables are selected:
   b and c_r: wing span and root chord length
   λ_1 and λ_2: taper ratios of two wing segments
   Λ_1 and Λ_2: wing sweep of two wing segments
   ξ_{FS} and ξ_{RS}: wing front and rear spar loc.
   Γ: dihedral of the wing
   C_f: Friction coefficient
   The SCAM tool is added, because of its ability to adjust some design variables, which are not explicitly stated in CPACS.
3. Collisions are now caused by HANGAR and SCAM writing to the same wing elements. Furthermore, SCAM introduces circular variables as it has the wing geometry entries as input and output. The collided and circular variables are fixed by creating variable instances.
4. Graph validity is checked against all necessary FPG conditions (see §§3.4.2).
5. The five modes of the SCAM function are merged into one function node.

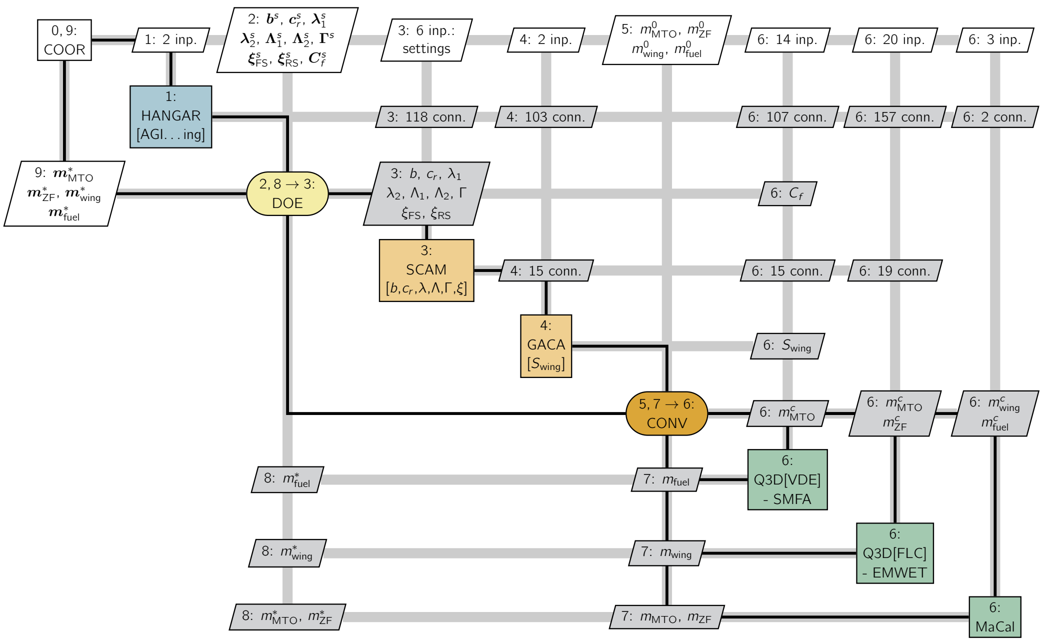
6-7. Function problem roles and order are set as shown in Fig. 3.24a.

The obtained FPG is shown in Fig. 3.24a. Two key concepts of the KADMOS graph syntax are used at step 3 of FPG creation process: the circularity index and instances. Initially, some of the wing definition nodes in the FPG are problematic (in a similar fashion as node b in Fig. 3.9a). This is because the HANGAR function provides a full initial wing definition that is then used and changed by the SCAM function. Fifteen values are changed by SCAM to adjust the wing based on top-level parameters and these fifteen values are initially of the subcategory 'collided shared coupling' with a circularity index of one. These collisions cannot be solved by simply removing connections, since the wing definition is supposed to be updated by SCAM and the initial geometry should come from HANGAR. Therefore, KADMOS automatically solves the collision by creating two instances of these variables, one instance before the SCAM function and one after. In this way, the collision is solved while the two instances still refer to the same position in the CDS, as is required.

The generated XDSM for the DOE solution strategy is shown in Fig. 3.24b. The Jacobi scheme was selected to test the effect of parallelizing all three disciplinary groups. The graph manipulation algorithm in KADMOS is able to position and connect the nested iterative elements (i.e. DOE and CONV) such that each design point to be analyzed is converged within the DOE block. Additional data elements that are required for the design variables and QOIs are also added and connected, for example, the new vector with samples for the design variables (e.g. $\boldsymbol{b}^s$) at PSN 2 and the vectors with final values of the QOIs (e.g. $\boldsymbol{m}^*_{\mathrm{MTO}}$) at PSN 9.



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



0, 9: COOR
1: 2 inp.
2: $\boldsymbol{b}^s$, $\boldsymbol{c}_r^s$, $\boldsymbol{\lambda}_1^s$ $\boldsymbol{\lambda}_2^s$, $\boldsymbol{\Lambda}_1^s$, $\boldsymbol{\Lambda}_2^s$, $\boldsymbol{\Gamma}^s$ $\boldsymbol{\xi}_{FS}^s$, $\boldsymbol{\xi}_{RS}^s$, $\boldsymbol{C}_f^s$
3: 6 inp.: settings
4: 2 inp.
5: m_{MTO}^0, m_{ZF}^0 m_{wing}^0, m_{fuel}^0
6: 14 inp.
6: 20 inp.
6: 3 inp.
1: HANGAR [AGI…ing]
3: 118 conn.
4: 103 conn.
6: 107 conn.
6: 157 conn.
6: 2 conn.
9: $\boldsymbol{m}_{MTO}^*$ $\boldsymbol{m}_{ZF}^*$, $\boldsymbol{m}_{wing}^*$ $\boldsymbol{m}_{fuel}^*$
2, 8 → 3: DOE
3: b, c_r, λ_1 λ_2, Λ_1, Λ_2, Γ ξ_{FS}, ξ_{RS}
6: C_f
3: SCAM [b,c_r,λ,Λ,Γ,ξ]
4: 15 conn.
6: 15 conn.
6: 19 conn.
4: GACA [S_{wing}]
6: S_{wing}
5, 7 → 6: CONV
6: m_{MTO}^c
6: m_{MTO}^c m_{ZF}^c
6: m_{wing}^c m_{fuel}^c
8: m_{fuel}^*
7: m_{fuel}
6: Q3D[VDE] - SMFA
8: m_{wing}^*
7: m_{wing}
6: Q3D[FLC] - EMWET
8: m_{MTO}^*, m_{ZF}^*
7: m_{MTO}, m_{ZF}
6: MaCal

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

Figure 3.24: Automatically generated XDSM visualizations of the KADMOS graphs for the development of the design space exploration system. Note that the FPG is also used for the MDO strategies, be it with small changes as indicated.

3.7.4. MDO STUDY

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

$$\begin{aligned}\text{minimize: } & f_{\text{MTOM}} = \frac{m_{\text{MTO}}}{m_{\text{MTO,ref}}}\\ \text{with respect to: } & b, c_{\text{r}}, \Lambda_1, \Lambda_2, \xi_{\text{FS}}, \xi_{\text{RS}}, C_f\\ \text{subject to: } & c_{\text{WL}} = \frac{m_{\text{MTO}}}{S_{\text{wing}}} - \text{WL}_{\text{ref}} \leq 0\\ & c_{\text{FT}} = \frac{m_{\text{fuel}}}{\rho_{\text{fuel}} \cdot \eta_{\text{FT}}} - V_{\text{FT}} \leq 0\end{aligned}$$

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

1. Select the MDAO architecture type MDO.
2. Design and QOIs remain from the previous FPG, except Γ, λ_1 and λ_2, and the following objective and constraints are selected:
   f_{MTOM}: output of function OBJ, hence this function from the RCG is added to the FPG
   c_{WL}: output of function CNSTRNT
   c_{FT}: output of function CNSTRNT
   In addition, the mode V_{FT} of the GACA tool is added, since the fuel tank volume is required for the constraint calculation.
5. The GACA modes are merged into one block as well as the two CNSTRNT modes.
6-7. Function problem roles and order are set as shown in Fig. 3.24a.
8. The coupled functions are distributed in two groups, as indicated in Fig. 3.24a.

On this FPG different MDO architectures can be imposed, with the resulting XDSMs for IDF and BLISS-2000 shown in Fig. 3.25 and Fig. 3.26. Of these two architectures, the distributed BLISS-2000 (Fig. 3.26) renders the idea of the extensive analysis and manipulation of the FPG necessary for the automatic formulation of a complex solution strategy. Other possible MDAO system reconfigurations, which could become interesting after the first MDO study, might include the replacement of some tool in the repository, a modification of the objective function or the addition of extra constraints, or a change in the MDO strategy. All of them could be easily accommodated and implemented in very short time (given a CPACS compatible tool repository): minutes instead of hours or even days, as would be required with the conventional manual approach.

This case study demonstrated how the KADMOS syntax and algorithms can enable a design team to quickly formulate and reconfigure an MDAO system, starting from a CDS-

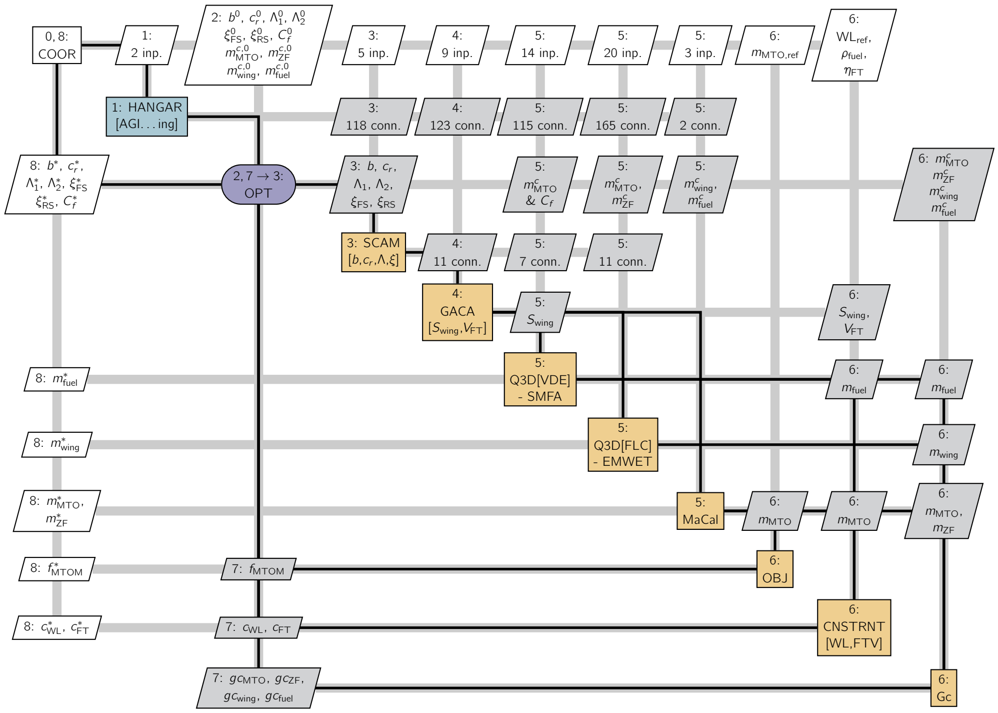
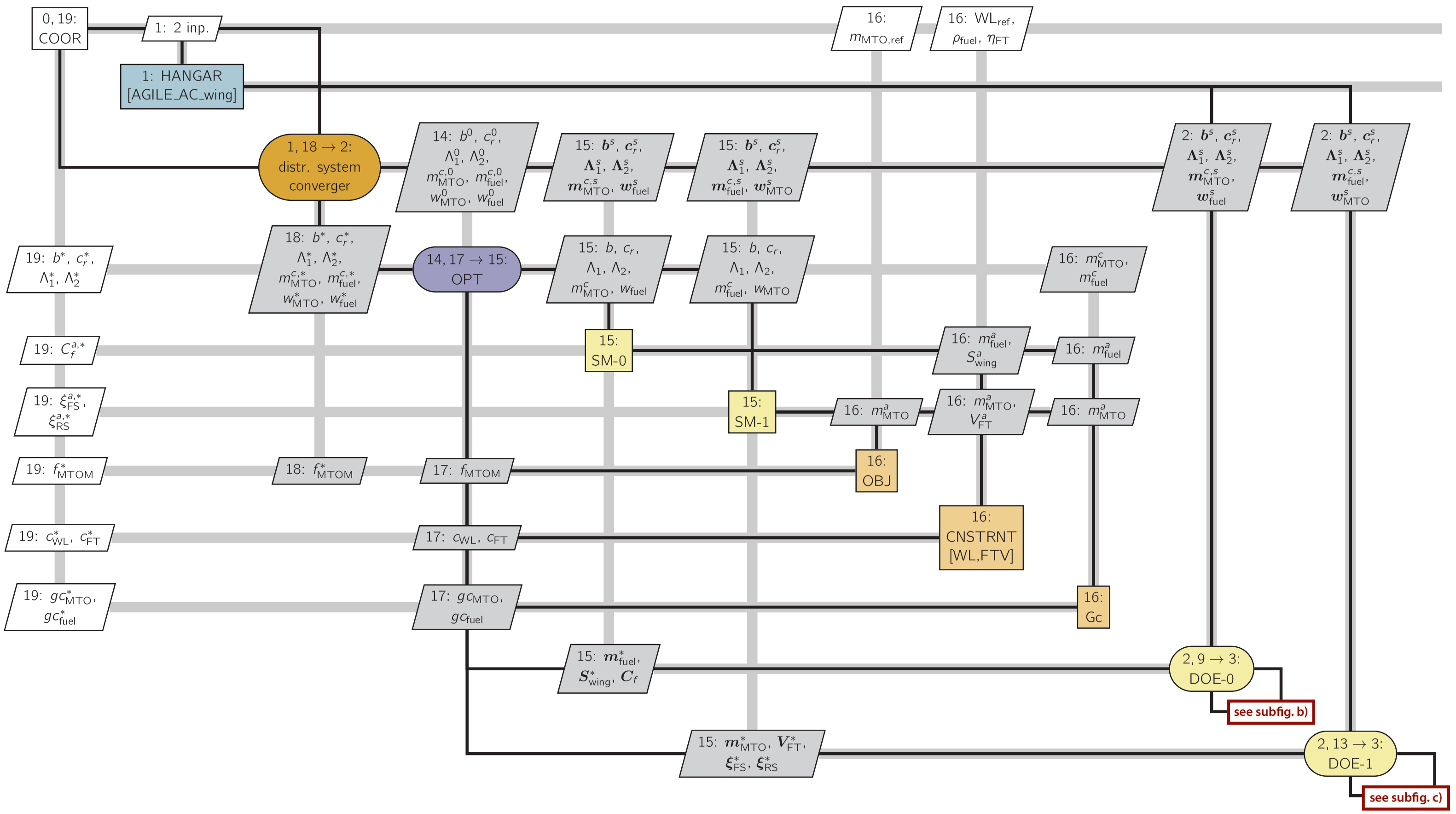
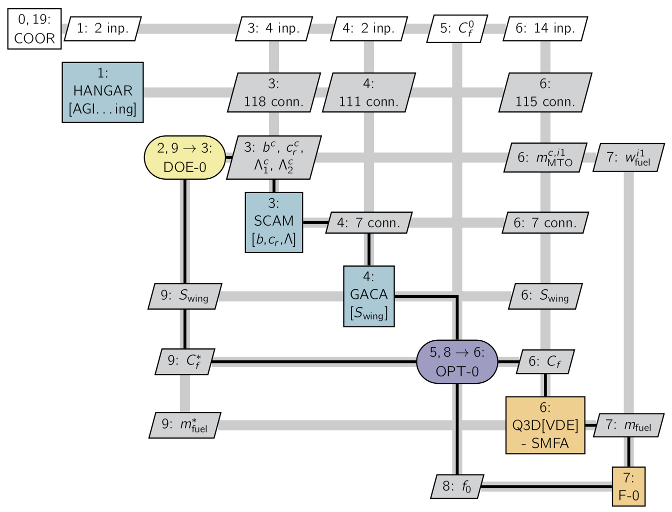


Figure 3.25: XDSM of the combined MDG+MPG for the IDF architecture

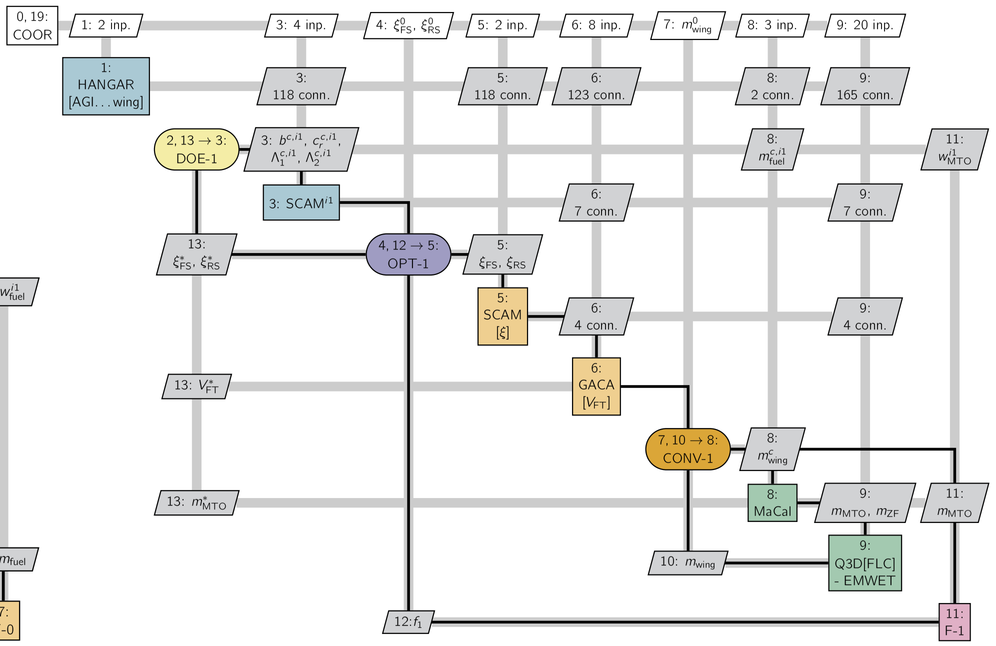
based repository of design tools. The configuration of the system results in the MDAO solution strategy: a blueprint of the workflow to be executed. In the next chapter a format for storing the KADMOS graphs will be presented and in the subsequent chapter this format will be used to automatically instantiate collaborative workflows in multiple PIDO platforms.


0, 19: COOR
1: 2 inp.
1: HANGAR [AGILE_AC_wing]
16: $m_{\mathrm{MTO,ref}}$
16: $\mathrm{WL}_{\mathrm{ref}}$, ρ_{fuel}, η_{FT}
1, 18 → 2: distr. system converger
14: b^0, c_r^0, Λ_1^0, Λ_2^0, $m_{\mathrm{MTO}}^{c,0}$, $m_{\mathrm{fuel}}^{c,0}$, w_{MTO}^0, w_{fuel}^0
15: $\boldsymbol{b}^s$, $\boldsymbol{c}_r^s$, $\boldsymbol{\Lambda}_1^s$, $\boldsymbol{\Lambda}_2^s$, $\boldsymbol{m}_{\mathrm{MTO}}^{c,s}$, $\boldsymbol{w}_{\mathrm{fuel}}^s$
15: $\boldsymbol{b}^s$, $\boldsymbol{c}_r^s$, $\boldsymbol{\Lambda}_1^s$, $\boldsymbol{\Lambda}_2^s$, $\boldsymbol{m}_{\mathrm{fuel}}^{c,s}$, $\boldsymbol{w}_{\mathrm{MTO}}^s$
2: $\boldsymbol{b}^s$, $\boldsymbol{c}_r^s$, $\boldsymbol{\Lambda}_1^s$, $\boldsymbol{\Lambda}_2^s$, $\boldsymbol{m}_{\mathrm{MTO}}^{c,s}$, $\boldsymbol{w}_{\mathrm{fuel}}^s$
2: $\boldsymbol{b}^s$, $\boldsymbol{c}_r^s$, $\boldsymbol{\Lambda}_1^s$, $\boldsymbol{\Lambda}_2^s$, $\boldsymbol{m}_{\mathrm{fuel}}^{c,s}$, $\boldsymbol{w}_{\mathrm{MTO}}^s$
19: b^*, c_r^*, Λ_1^*, Λ_2^*
18: b^*, c_r^*, Λ_1^*, Λ_2^*, $m_{\mathrm{MTO}}^{c,*}$, $m_{\mathrm{fuel}}^{c,*}$, w_{MTO}^*, w_{fuel}^*
14, 17 → 15: OPT
15: b, c_r, Λ_1, Λ_2, m_{MTO}^c, w_{fuel}
15: b, c_r, Λ_1, Λ_2, m_{fuel}^c, w_{MTO}
16: m_{MTO}^c, m_{fuel}^c
19: $C_f^{a,*}$
15: SM-0
16: m_{fuel}^a, S_{wing}^a
16: m_{fuel}^a
19: $\xi_{\mathrm{FS}}^{a,*}$, $\xi_{\mathrm{RS}}^{a,*}$
15: SM-1
16: m_{MTO}^a
16: m_{MTO}^a, V_{FT}^a
16: m_{MTO}^a
19: f_{MTOM}^*
18: f_{MTOM}^*
17: f_{MTOM}
16: OBJ
19: c_{WL}^*, c_{FT}^*
17: c_{WL}, c_{FT}
16: CNSTRNT [WL,FTV]
19: gc_{MTO}^*, gc_{fuel}^*
17: gc_{MTO}, gc_{fuel}
16: Gc
15: $\boldsymbol{m}_{\mathrm{fuel}}^*$, $\boldsymbol{S}_{\mathrm{wing}}^*$, $\boldsymbol{C}_f$
2, 9 → 3: DOE-0
see subfig. b)
15: $\boldsymbol{m}_{\mathrm{MTO}}^*$, $\boldsymbol{V}_{\mathrm{FT}}^*$, $\boldsymbol{\xi}_{\mathrm{FS}}^*$, $\boldsymbol{\xi}_{\mathrm{RS}}^*$
2, 13 → 3: DOE-1
see subfig. c)

a) System-level BLISS-2000 approach



b) Sub-level 0 BLISS-2000



c) Sub-level 1 BLISS-2000

Figure 3.26: Automatically generated XDSM visualization in three parts of the KADMOS graphs for the BLISS-2000 solution strategy imposed on the FPG shown in Fig. 3.24a

3.8. Discussion

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

3.8.1. Impact on collaborative MDAO development process

The step-by-step formulation approach implemented in KADMOS, dramatically increases the agility of the design team in the application of collaborative MDAO as it connects and automates the first three steps of the MDAO development process depicted in Fig. 3.1. After the execution of a multidisciplinary convergence study, for example, designers can easily set up a DOE study and evaluate the sensitivity of the results to certain parameters. This information can then be used to formulate an optimization problem that includes as design variables only the parameters resulting as most effective in the DOE. After that, also the burden to embed the problem into one of the various and diverse MDO architectures is totally eliminated, because KADMOS can manipulate the same FPG into any MDO architecture in just minutes. If preliminary results of the optimization suggest the use of a more convenient architecture, changes will be effortless. Also, if constraints and/or design variables need to be added or removed, or different objectives selected, or different disciplinary tools be involved (as far as compliant to the common data schema), KADMOS provides the necessary agility to easily adjust any of the aforementioned, thus supporting the typical iterative nature of the design process.

The benefit of KADMOS is not limited to the design agility augmentation. The next chapter describes how the KADMOS graphs are stored using a standardized format so other applications of the MDAO framework can also use the formulated systems. For example, the VISTOMS package (co-developed with KADMOS) can be used at any stage of the formalization process to generate the necessary visualizations to report the status of the MDAO system under development, to ease debugging and, most of all, to guarantee discipline experts and MDAO architects the required oversight to manage distributed computational systems of any size.

3.8.2. Originality

Although the graph-based methodological approach proposed in this paper takes inspiration from the work of Pate et al. (discussed in §2.2 and shown in Fig. 2.6), there are notable differences in the graph syntax, as well as in the scope and implementation of the entire MDAO support system:

- Pate's graph formulation is focused on the transition between the first and second stage of the MDAO development process, KADMOS has a broader scope including the transitions to stages three and four. The inclusion of these additional stages required a more sophisticated definition of the syntax and the graph-theoretic conditions the graphs have to satisfy.
- KADMOS is based on the CDS approach for system composition, which further differentiates the syntax and conditions.
- The KADMOS syntax covers both monolithic and distributed MDO architectures, whereas the latter were not covered by Pate's approach.
- In the syntax, the circularity index is one of the fundamental KADMOS extensions to Pate's syntax. Based on the circularity index, crucial new subcategories are introduced to handle circular variable nodes.
- The second key syntax extension is the node instance. Instances enable the splitting of nodes to bring together the graph-based and CDS approaches. The splitting of nodes is a crucial operation, since not every collided or circular node in the data graph can be solved by just removing edges. As was demonstrated in the demonstration case study in §3.7, collisions and looped pairs are commonly present in realistic design cases and need to be handled properly.

With respect to Hoogreef's InFoRMA system: while several KADMOS capabilities match those offered by this system, the theoretical basis drastically differs from Hoogreef's use of semantic webs to represent MDAO systems. Furthermore, InFoRMA focuses on advising the user in selecting the right MDO architecture, which is an aspect that is not included in KADMOS. InFoRMA on the other hand lacks support for schema-based system composition, and the automated sequencing and decompositioning algorithms discussed in §§3.5.3 and §§3.5.4.

The impact of the graph-based methodology in the broader MDAO framework under development in the AGILE project will be presented in Chapter 6. Chapter 7 will demonstrate that the methodology is generic enough to also be applied outside the AGILE context in which it was developed.

4 Proposed standard to store and exchange MDAO system formulations

The previous chapter discussed how the graph-based methodological approach, which was implemented as KADMOS, enables a design team to formulate MDAO systems for all three stages of the formulation phase in the MDAO development process. Within a broader third-generation MDAO framework, the KADMOS package is merely one of the many tools at the team's disposal, as will be elaborated in detail in Chapter 6. Next to the various disciplinary tools, other applications are present in the framework to support other tasks, such as managing the designers and the design process, creating visualizations, or instantiating executable workflows. Many of these applications would benefit from having access to the MDAO system formulations created by KADMOS. However, this would require all these applications to be Python-based like KADMOS, or would require the development of multiple APIs in KADMOS. Both options require developers to become familiar with the KADMOS package, which can be a significant time investment. Therefore, a more straightforward way to access the information on the KADMOS-generated formulations was devised, based on the definition of a dedicated storage standard for MDAO systems. This chapter describes in detail this storage format called CMDOWS (pronounced as: commandos): Common MDO Workflow Schema. The format was developed not only so that other developers and applications can exploit at best KADMOS's services, but also to fill the existing gap in the standardization of MDAO system representations.

First, the development of the new standard is introduced in §4.1, followed by its functional requirements in §4.2. §4.3 forms the core of this chapter, in which the schema is presented. CMDOWS is then illustrated with a small example in §4.4.

The contents of this chapter have been adapted from [96].

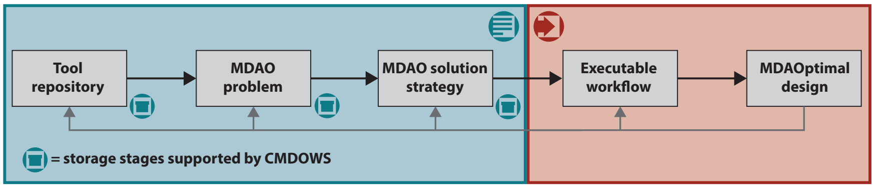


Figure 4.1: The coverage of Common MDO Workflow Schema (CMDOWS) in the MDAO development process

4.1. Introduction

The development of the novel data format for MDAO systems is introduced here in four subsections. First, the different types of framework applications included in a third-generation framework are listed in §§4.1.1, followed by an explanation of the motivation to develop a new standard format in §§4.1.2. The state of the art regarding standardization of MDAO systems will be covered in §§4.1.3 and the section closes with a brief discussion on the consequences of using a standard format for framework development in §§4.1.4.

4.1.1. Framework applications

The need for a novel MDAO framework generation to support collaborative MDAO in the industrial context was discussed in §2.7. In order to establish such a novel environment, the third-generation MDAO framework needs to synthesize a range of applications to support collaborative multidisciplinary design. The following pivotal application categories were identified based on past frameworks and the needs identified in the AGILE project:

Tool repositories A tool repository is a database that contains the definition of a collection of design and analysis tools that can be made available to the design team for execution. The repository does not necessarily contain the tools themselves, as the sharing of the tool might be prohibited by intellectual property restrictions. In that case, a repository contains the specification of the (interlinked) inputs and outputs of the tools and the way in which each tool can be (remotely) executed. Different approaches for composing repositories have been discussed in §2.1.

MDAO system formulation applications This type of application is used in the formulation phases of the development process (see Fig. 4.1). These platforms support the team in defining the MDAO problem and its solution strategy in a PIDO-platform-agnostic manner. In addition, by automating and supporting the formulation task, the MDAO system can be reconfigured more easily to incorporate progressive insights of the team. Examples of these applications are the InFoRMA and GEMS platforms discussed in §2.3, and the KADMOS platform that was presented in Chapter 3. Earlier work in IDEaliSM [26, 97] has shown that the use of an MDAO system formulation platform (i.e. InFoRMA), can result in a significant set-up time reduction, even larger than 90%.

Visualization packages The visualization of large MDAO systems can be challenging, but is crucial to share and discuss the project developments within the heterogeneous team of experts. A visualization package to inspect and communicate the system formulations at the different stages of the development process, which are produced by the aforementioned MDAO system formulation applications, can also contribute to decreasing the set-up time. In addition, such visualizations increase the trust of the design team in the large, complex automated analysis chain that is being built. Different forms of visualizations suitable to MDAO have been discussed in §2.2.

Collaborative workflow platforms The workflows are the executable instances of the MDAO solution strategies produced during the MDAO system formulation phase (Fig. 4.1). The term collaborative is used to express the fact that these workflows combine different disciplinary subworkflows from the tool repository, which are owned by different disciplinary experts (or teams), into one optimization workflow. The combination of such subworkflows can be very challenging, especially when the disciplinary teams are distributed either geographically, digitally (i.e. subworkflows running on different server domains), or both. PIDO platforms for executing these workflows have been discussed in §2.5.

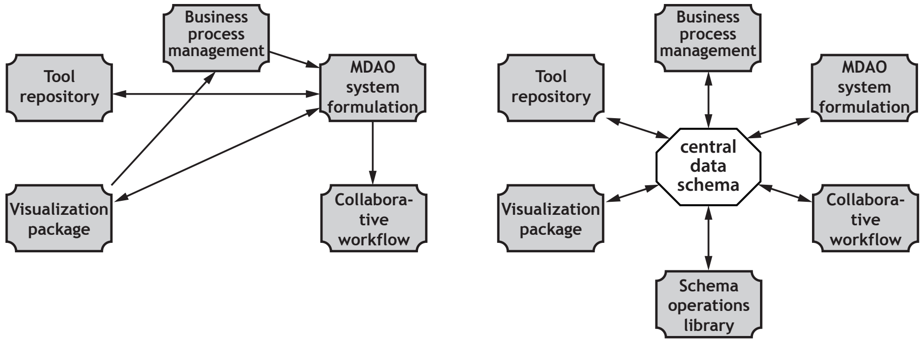
Business process management tool The management of a design team is streamlined using this type of tool. The management tool enables the collaboration between various actors by providing a common platform for management tasks, such as communication, progress tracking, database management, and task planning and execution. Project management tools come in many shapes and sizes. A modern-day application would be web-based and allows all users to access and edit tasks relevant to their project contribution. The team lead uses the platform to keep track of the planning. In MDAO projects, the business process should be tightly integrated with the multidisciplinary system that is being built and executed, if the management tool wants to positively impact project progress.

Schema operations library If standardized files are used to exchange information about the MDAO system (as will be discussed in this chapter), then this application category contains the collection of useful methods to inspect, check, or analyze the schema-based files. Typically, these applications contain functions to check files for their validity (e.g. with respect to the schema definition), to determine key values (e.g. number of tools, number of parameters), and to edit instances of the schema file (e.g. by removing or adding tools and parameters). These libraries can be seen as a key enabler to adapt the schema within a framework. This is equivalent to the ecosystem of libraries for CPACS, which was discussed in §2.1.

Many other application categories could be added to this list, as the capabilities of a third-generation framework extend. Each task of the engineering team could be supported by an application and each application could be integrated in the framework. This could be tasks such as requirements management, workflow data postprocessing, and time planning. In this work, the scope is limited to the application categories listed above, as these are generally at the core of any collaborative framework.

4.1.2. Motivation

In the AGILE and IDEaliSM projects (discussed in §§2.6.2 and §§2.7.2 respectively), various applications used to support the development of an MDAO system were initially coupled in the way illustrated in Fig. 4.2a. Most of the applications had to communicate directly with each other, with obvious problems of flexibility and maintainability of the overall framework, due to the many ad-hoc interfaces. In addition, the benefits (in terms of overall set-up time reduction) of the vendor-neutral graph-based representations of the MDAO system produced by KADMOS (and InFoRMA in IDEaliSM) were found to be limited without the possibility to automatically generate the collaborative workflows using a PIDO platform of choice. Furthermore, investing in the development of visualization packages (such as VISTOMS [S11][39] discussed in §§3.5.1) for one specific MDAO system formulation tool would not have been worth the effort, while the ad-hoc development of such advanced visualization capabilities both inside KADMOS and InFoRMA was also not feasible within the project time frame.



a) no Central Data Schema b) with Central Data Schema

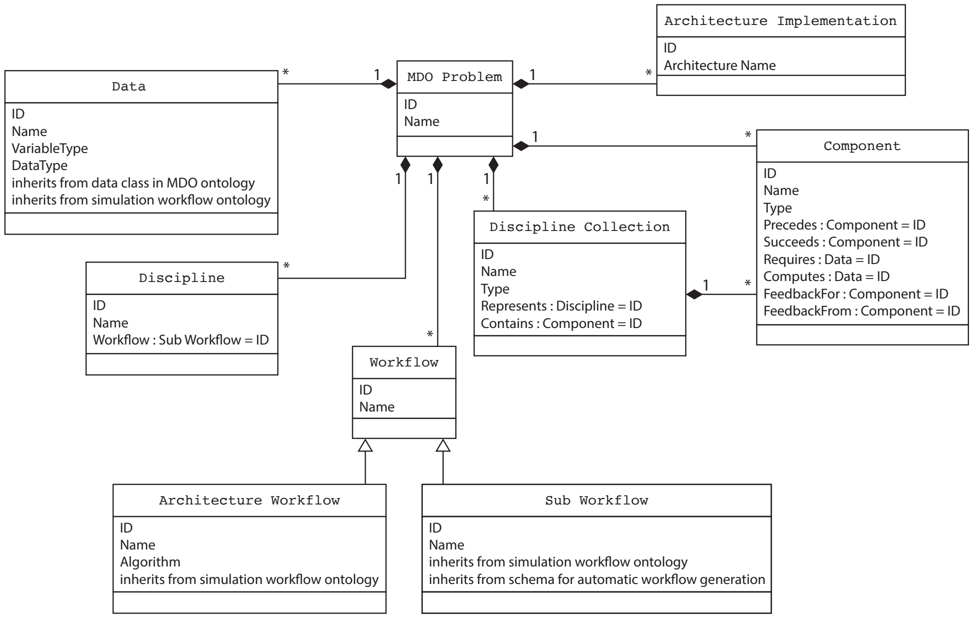
Figure 4.2: Schematic overview of framework integration with and without a central data schema approach.

For all these reasons, and also on the basis of the evident benefit provided by the central-data-schema-based repository creation using CPACS (discussed in §§2.1.2), a dedicated standard format to define, store and exchange MDAO systems is proposed here as a key enabler for the automation of both the formulation and execution phase in any large collaborative MDAO project. The presence of a schema reconfigures the application links to the set-up shown in Fig. 4.2b. The position of such a storage standard in the MDAO development process is indicated in Fig. 4.1.

4.1.3. State of the art

The need for standardization in engineering design is not something new. However, a standard for storing MDAO system definitions in a neutral format to support the integration of collaborative frameworks is not a widely addressed topic. This type of format was first advocated within IDEaliSM to facilitate the translation of the system formalization generated by InFoRMA into executable workflows in Optimus. To this purpose, a prototype neutral format was defined by Hoogreef [26], which is depicted in Fig. 4.3, but this prototype was never developed further or tested within the project.

Other standardized format definitions can be found in earlier work and are usually motivated by the same kind of integration as in the IDEaliSM project. For example, Gondhalekar et al. [98] proposed a neutral format in the context of the "behavioural digital aircraft" project CRESCENDO[99] to exchange computational workflows, and demonstrated that through such a format the same workflow can be built in two different workflow platforms. Similarly, another format that is gaining momentum is the Functional Mock-up Interface (FMI)[100]. FMI is a platform-independent standard that is aimed at supporting both model exchange and co-simulation of dynamic models. Both these examples are concerned with the sharing of tools within the collaborative workflow, but do not consider other types of applications used to support the development of MDAO systems, as was discussed in the previous section. Other related formats store the neu-



Data
ID
Name
VariableType
DataType
inherits from data class in MDO ontology
inherits from simulation workflow ontology
MDO Problem
ID
Name
Architecture Implementation
ID
Architecture Name
Component
ID
Name
Type
Precedes : Component = ID
Succeeds : Component = ID
Requires : Data = ID
Computes : Data = ID
FeedbackFor : Component = ID
FeedbackFrom : Component = ID
Discipline Collection
ID
Name
Type
Represents : Discipline = ID
Contains : Component = ID
Discipline
ID
Name
Workflow : Sub Workflow = ID
Workflow
ID
Name
Architecture Workflow
ID
Name
Algorithm
inherits from simulation workflow ontology
Sub Workflow
ID
Name
inherits from simulation workflow ontology
inherits from schema for automatic workflow generation

Figure 4.3: Unified Modeling Language (UML) class diagram with the outline of the schema for a neutral format's data structure to exchange MDAO architecture and simulation workflow information proposed by Hoogreef [26]

tral description and visualization of processes, as is done with BPMN (Business Process Model and Notation)[101] that is maintained by the Object Management Group. In a way, the XDSM visualization standard by Lambe and Martins [37] (discussed earlier in §2.2), can also be seen as a neutral format to store both process and data flow. However, the file type used for this (either tex or PDF files) is not very suitable to exchange data.

In summary, standardizing MDAO system definitions for information storage and sharing has not been a widely addressed topic. Standards for workflow or (dynamic) model exchange between execution platforms have been developed in the past, but none are broad enough to cover the range of MDAO framework applications generally used to support a design team. The full development of a neutral format for storage and exchange of MDAO systems took place in the AGILE project. The result is the CMDOWS (Common MDO Workflow Schema) format described in this chapter.

4.1.4. Impact on framework set-up

Thanks to the definition of CMDOWS, the application links within the AGILE framework changed from the set-up shown in Fig. 4.2a to the new centralized structure in Fig. 4.2b. In this approach, the graph-based representations generated by KADMOS are stored as CMDOWS files and then used by other framework applications. For example, a MDAO solution strategy stored in a CMDOWS file can be translated into an executable workflow with any PIDO platform able to interpret the format. This aspect will be elaborately discussed in Chapter 5.

As a matter of fact, CMDOWS files are not only usable to store MDAO systems at the final stage of the development process (MDAO solution strategy) depicted in Fig. 4.1, but also for earlier stages (tool repository, MDAO problem). This has two major advantages:

- The visualization package, rather than accessing the different internal data structures of KADMOS (or alternatively InFoRMA), can read and visualize produced CMDOWS files and help inspecting and monitoring the state of the MDAO system during all the stages of the formulation phase.
- The usability of KADMOS is also improved. For example, a tool repository definition can be provided as input to KADMOS in the form of a CMDOWS file. This repository definition can now be provided by another dedicated framework application that is focused on tool repository management. KADMOS can be used to enrich that repository CMDOWS file by adding the problem definition data, or the complete MDAO solution strategy.

The six support application categories discussed in §§4.1.1 can have bidirectional links to the workflow schema, although, for all of them a primary and a secondary link direction can be identified[*], as is illustrated in Fig. 4.4. For example, the visualization package has the primary link of being able to open any workflow schema file and depict the visualizations. A secondary link would be in place, if the visualization package would offer users also the possibility to manually edit the visualized CMDOWS file. Generally, the primary link is the one that is most directly useful and, most of the time, also the easiest to develop for the category at hand.

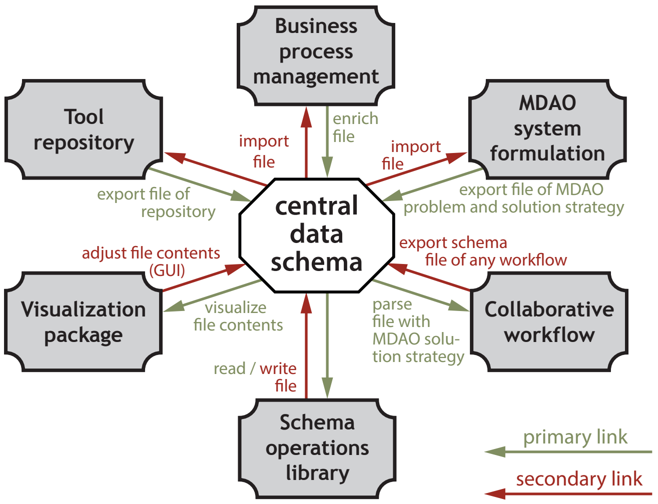

Business process management
Tool repository
MDAO system formulation
import file
enrich file
import file
central data schema
export file of repository
export file of MDAO problem and solution strategy
adjust file contents (GUI)
export schema file of any workflow
Visualization package
Collaborative workflow
visualize file contents
parse file with MDAO solution strategy
read / write file
Schema operations library
primary link
secondary link

Figure 4.4: Primary and secondary links between the workflow schema and the framework application categories

4.2. Functional requirements

The proposed schema is based on the following nine main functional requirements:

I **Machine-interpretable** The format in which the MDAO system is stored should be machine-interpretable up to the finest level of detail.

[*]N.B. This primary/secondary link definition can be considered subjective. Here the ordering is based on the perspective of the system integrator.

II **Human-readable** The schema should allow any designer to inspect at least the top-level correctness of the content, while users and developers with a background in design engineering or computer science should be able to find and understand all the fine details. This human-readability aspect is important to enable the use of the schema by a wider community and ease the connection of new MDAO framework applications.

III **Neutral** The schema should not contain elements that are specific to any project, MDAO framework application, or developed product. However, the schema should accommodate the storage of any such additional information at specific locations to address practical issues of certain projects, applications, or products, thereby allowing project-specific additions to the schema file at the dedicated file locations.

IV **Validation** File instances that are based on CMDOWS, should be easily validated against the schema definition.

V **Adaptable** From one version release to another, the schema should always be flexible enough to provide room for extensions and enrichment, while at the same time its basic structure should not change too drastically to keep any existing framework application links easily (with a small developing effort) compliant with the release of each new version.

VI **Balance of redundant information** Data representation in the schema should aim at minimum redundancy, however, in special cases this redundancy guideline can be violated for convenience (i.e. to facilitate the link with certain applications that lack the capability to automatically derive the required input based on the information stored in the format). Such redundancies bring the risk of generating inconsistencies in file instances and therefore a balance should be found between information that can be implicitly and explicitly stored in the file.

VII **Support all MDAO system stages** The schema should support storage of the MDAO system during the three different stages of the formulation phase indicated in Fig. 4.1.

VIII **Support all MDAO framework categories** The schema should accommodate all information that is required to enable the links with the six different MDAO framework application categories depicted in Fig. 4.2b.

IX **Support disciplinary tool heterogeneity** A broad range of analysis tools from the tool repository, including their execution methods, should be stored in the schema, such as simple mathematical expressions, remotely executed 'black boxes', and surrogate models.

With the requirements listed above in mind the CMDOWS version 0.9 has been completed. The parallelism between CMDOWS and CPACS (the latter was discussed in §§2.1.2) is evident from the requirement list. The one prominent difference is in the neutrality requirement. While CPACS is project and disciplinary tool neutral, it is specific for the developed product: the aircraft. CMDOWS is fully neutral, thus applicable to any possible domain where MDAO is of interest, such as automative, infrastructure, and wind energy.

4.3. Definition of the storage standard

The eXtensible Markup Language (XML) [S23] was selected as the syntax to store the schema definition. With XML, a schema is defined through the XML Schema Definition (XSD) format. This XSD definition of CMDOWS can then be used to validate any

CMDOWS XML instance, thereby meeting requirement IV (req-IV) on validation in §4.2. XML also meets req-I and req-II, as it is both human-readable and supports machine-interpretability. In addition, the XML format is independent of the programming language used. Many programming languages actually include utilities to work with the XML format, thereby supporting the human-readability requirement in the sense that many users can easily familiarize themselves with CMDOWS within their preferred programming environment. Another argument for the use of XML is the adoption of the XML-based CPACS in the aircraft MDAO community, e.g. the FrAECs, IDEaliSM, and AGILE project all relied on CPACS for tool data, meaning that many, both in industry and academia, are already familiar with the use of XML as a data storage format.

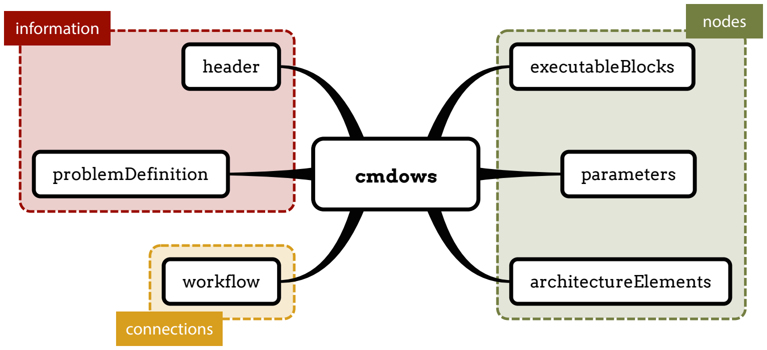


Figure 4.5: Top-level elements of CMDOWS and the three main element categories

The CMDOWS definition (see Fig. 4.5) is structured in six top-level elements, grouped in three basic categories:

- information
- nodes
- connections

This categorization is based on the assumption that any MDAO system can be modeled as a graph, as discussed in Chapter 3, thus build up of nodes and their connections (also referred to as edges). The information category is used to store generic information about the system and the MDAO problem definition. All these categories will be discussed in a separate subsection of this section. For brevity, the description is limited to the top-level elements of the schema up to level 4 (when the root element is considered to be at level 0), whereas the full schema, which has elements up to level 7, can be inspected at the open-source repository [S24].

An important concept, used at different locations in the schema, is the separation between parameters and executable blocks. Any node element describing a tool repository, MDAO problem, or MDAO solution strategy will fall under one of these two groups. The parameters group refers to all the elements inside an MDAO system that are assigned a certain value. Parameters are the I/O of the executable blocks, such as the actual optimization parameters (whose values remain constant during optimization) and the design variables (including both actual design variables and the copy or surrogate variables introduced by different MDAO architectures). The executable blocks are defined as elements that take certain inputs, perform an operation, and finally produce certain outputs. Since the distinction between parameters and executable blocks is a key aspect, the element names `parameters` and `executableBlocks` appear at different levels in the schema. The two terms are analogous to the variable and function categories for KADMOS graph nodes explained in §§3.2.1.

For example, a parameter `x1` is input to an executable block and will be defined in the main `parameters` element of the nodes category. Generic information about the parameter, such as a description, unit, and datatype would be stored directly on the element as metadata. When this parameter has to be indicated as a design variable (including bounds and nominal value) for a certain MDAO system, then this information is stored inside the `problemFormulation/parameters` element. This way, the elements in the nodes category remain independent and valid for any system, while the `problemFormulation` element contains information that is specific to the MDAO problem stage from Fig. 4.1. The two elements are linked together through UIDs as will be shown in more detail in the next sections. The six top-level elements from Fig. 4.5 are discussed in more detail in the upcoming sections.

4.3.1. Elements in the information category

The information elements of CMDOWS are `header` and `problemDefinition`. Lower level elements of the information elements are shown in Fig. 4.6. The `header` element contains metadata relative to the CMDOWS file itself, such as the creator, a description, the schema version used. Additionally, the header contains the contacts database and organigram. The `contacts` database can be used throughout the schema to refer to owners, creators, and other agents required at other locations. The `organigram` allows the team to store the roles of the people involved in the project.

The definition of the MDAO problem to be solved can be stored in the `problemDefinition` element. This element can get a UID assigned as an attribute (indicated with the @ sign in Fig. 4.6) so that it can be referred to in other parts of the schema. The other two main elements of the problem definition are `problemRoles` and `problemFormulation`. In the `problemRoles` branch all the special parameters of the MDAO system get their roles assigned, such as design variable, objective, constraint, or state variable, including certain parameter settings for the problem at hand (i.e. upper and lower bounds, constraint types). All executable blocks also get a problem role, based on their connections with other blocks and their position with respect to the design variables, see `uncoupledDes-VarIndBlocks` and `uncoupledDesVarDepBlocks` in Fig. 4.6. These roles match the KADMOS function node roles of the FPG described in §§3.4.2. The second branch of the problem definition is the `problemFormulation`, where the specification of the MDAO architecture that should be imposed on the problem and the logical order of the executable blocks are stored. This logical order is required to determine (among others) the feedback between different blocks and can also be used to automatically determine problem roles of the executable blocks.

4.3.2. Elements in the nodes category

The node elements all represent either parameters or executable blocks and are separated into three subelements: `executableBlocks`, `parameters`, and `architectureElements`. Their subelements are depicted in Fig. 4.7. The `executableBlocks` element contains the function blocks that are stored in the tool repository. Two main types of executable blocks can be stored inside this element: *mathematical functions* and *design*

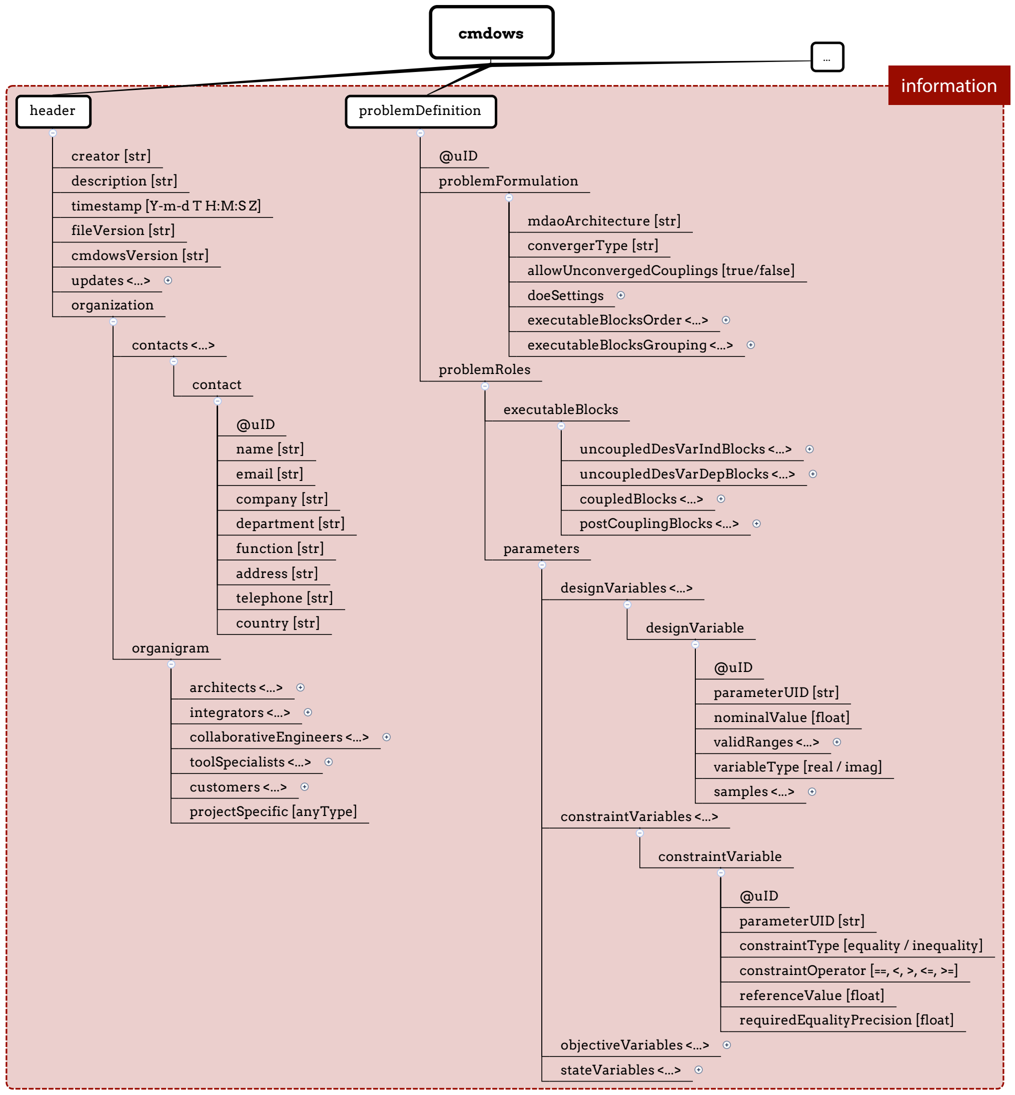


Figure 4.6: Elements in the CMDOWS category Information

competences. The mathematical functions are simple executable blocks that evaluate analytic expressions to determine the value of the outputs. These expressions can be stored directly in the `mathematicalFunction` element, thereby storing the full definition of that executable block. Hence, the actual operation performed by the block is stored for mathematical functions. Contrary to this, design competences represent more complex executable blocks where the operation performed by the block is unknown (or at least cannot be stored as simple mathematical expressions). The `designCompetence` element therefore stores a block that performs an unknown operation (it acts as a so-called 'black box'). Instead of storing the operation itself, the schema of the `designCompetence` element can accommodate a range of specifications for executing the tool. For example, a design competence can be an integrated analysis tool on the local system, a remotely called execution using a special server integration, a surrogate model, or any other form of computational module present in a collaborative workflow. Additional information about the competences, such as ownership and the outcome of eventual verification steps, can be stored in the `metadata` element, see Fig. 4.7.

The second node element is the `parameters` element. This element contains all the inputs and outputs of the executable blocks stored in the main `executableBlocks` element. If the executable blocks are integrated using a Central Data Schema (CDS) approach (i.e. CPACS), then the `parameters` element will contain all the unique elements that are used from that schema as separate parameters. Additional information about the parameters can also be stored, such as a label, description, unit, and data type (real, float, list, etc.).

The last node element is `architectureElements`. This element includes both `parameters` and `executableBlocks`, which differ from those stored in the top-level `parameters` and `executableBlocks` element because they are the additional elements created when an MDAO architecture is imposed on a specified problem. For example, when an architecture has to be imposed, a new executable block has to be introduced to handle the optimization loop: an optimizer. This new element is stored as an `architectureElements/executableBlock`. New parameters also have to be introduced to connect the optimizer to the rest of the system. Initial guesses for all design variables are required as input of the optimizer and the optimal (final) values of the design variables, objective, and constraints have to be connected as outputs. These parameters do not exist before the imposition of the MDAO architecture and are therefore instantiated and stored as `architectureElements/parameters`. Similarly, other architecture elements are introduced including convergers, DOE blocks and other components that are specific to a given architecture. The complete list of possible architecture elements in the schema is shown in Fig. 4.7. Next to newly introduced elements, the original executable blocks are also grouped based on their KADMOS architecture role (discussed in §§3.2.1 and listed in Tab. 3.8) via their UID, see `uncoupledDesVarIndAnalyses` element and below in Fig. 4.7. This categorization can, for example, be used for assigning colors in the visualization of the CMDOWS file or when parsing the file to create an executable workflow.

4.3.3. Elements in the connections category

CMDOWS contains a single element for storing the connections: `workflow`. In this element two different types of graphs can be stored: data graphs and process graphs. As was discussed in §3.4, the combination of these two graphs constitutes the neutral definition

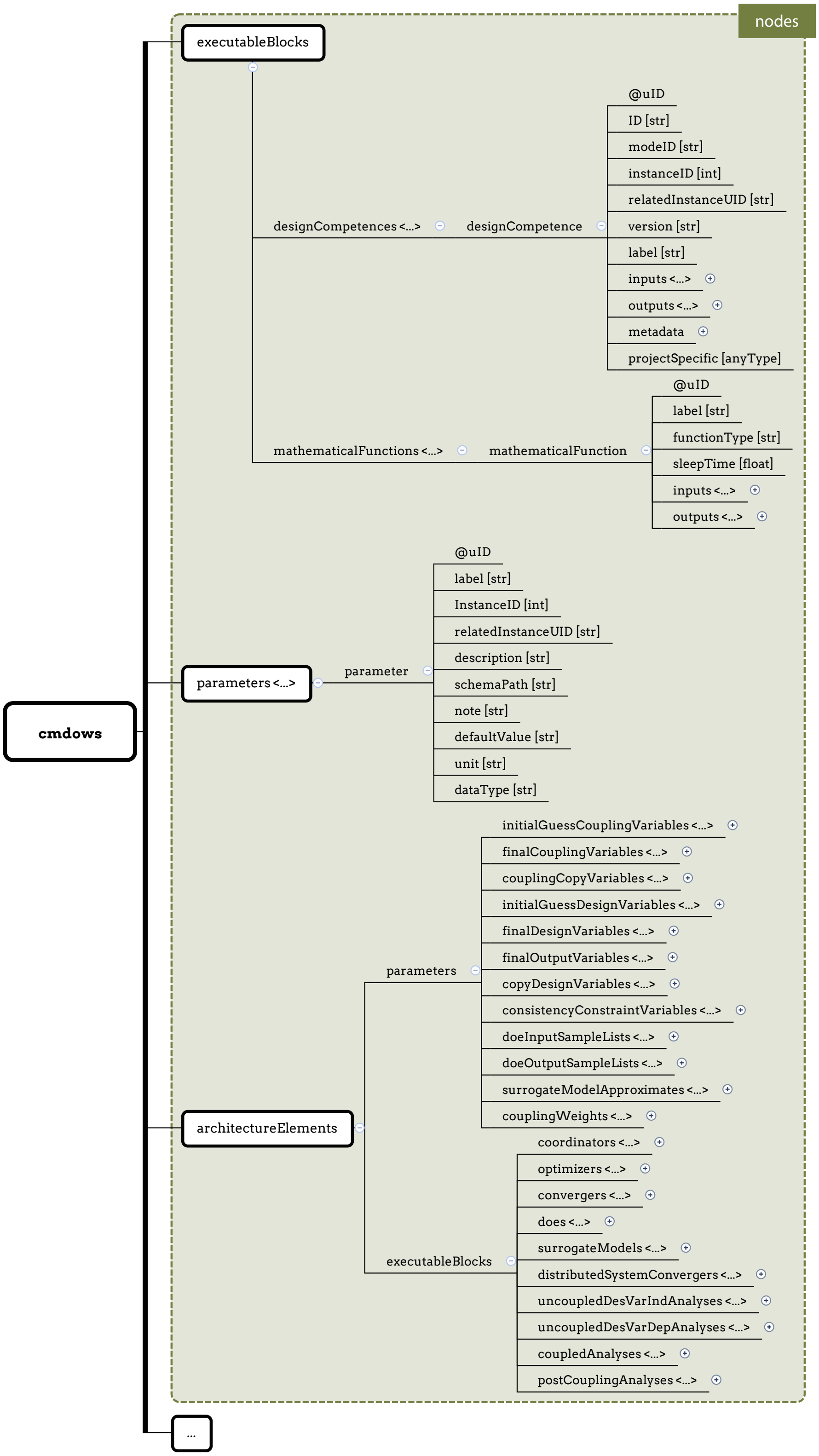


Figure 4.7: Elements in the CMDOWS category Nodes

of a workflow that needs to be executed to solve an MDAO problem: the MDAO solution strategy. The `dataGraph` element contains a data graph storing the connections (or edges) between parameters and executable blocks according to their I/O relations. This data graph can be stored for any formulation stage of the MDAO system in Fig. 4.1, where each stage would be represented by a separate CMDOWS file. The `processGraph` element is only used for the MDAO solution strategy to store the process steps for running the different executable blocks. Metadata about the graphs can also be stored, such as the number of nodes and edges, and the nesting of the process steps for the process graph.

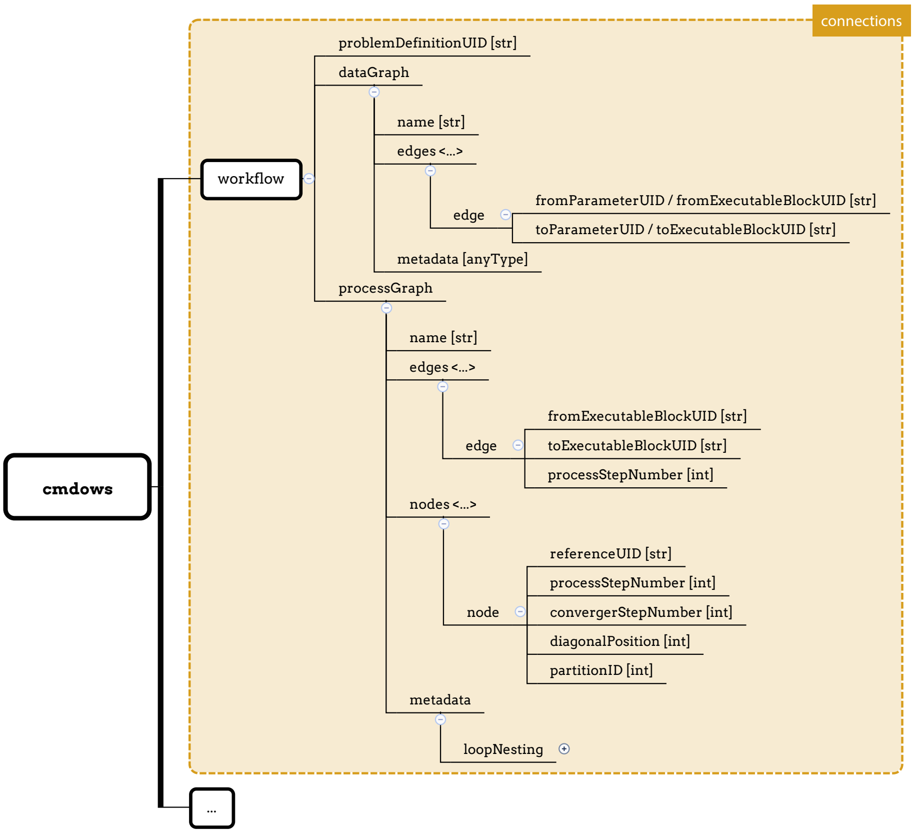


Figure 4.8: Elements in the CMDOWS category Connections

4.4. Illustrative example: Sellar problem

The Sellar problem, introduced in §3.4, is used in this section to demonstrate the use of CMDOWS as a schema to store the three different stages of the MDAO system during the formulation phase. As the MDAO system is transformed between each stage (see Fig. 4.1), a different CMDOWS file is stored for each of them. This section describes how each stage is stored, thereby giving the reader a clearer understanding of the different schema elements. The way this enrichment is performed with the formulation tool KADMOS has been discussed in detail in the previous chapter (§3.4).

4.4.1. Stage I: Tool repository

The tool repository for the Sellar problem was described in §3.4. The original Sellar problem actually contains only five tools, but here, fictitious tools (A, B, C, etc.) were added to demonstrate how an MDAO problem can be based on a subset of tools from the repository. The CMDOWS file shown in Fig. 4.9 matches the RCG described in §§3.4.1. The only elements from the schema needed to store a tool repository are the `executableBlocks`, `parameters`, and `workflow/dataGraph`. As shown in Fig. 4.9, the different executable blocks are integrated differently: the disciplinary analyses D[1] and D[2] are design competences, meaning that the mathematical expressions to be executed are assumed to be unknown (for illustration purposes), while for the remaining functions mathematical expressions are available. For all executable blocks the inputs and outputs of each block are defined by referring to the right elements from the `parameters` list using the relative parameter UID (as is shown for parameters x1 and f in the figure). Finally, the `dataGraph` element contains the full graph, as illustrated on the lower right of the figure, by listing the edges between all executable blocks and parameters. The storage of the edge y1 → F is illustrated in the figure.

4.4.2. Stage II: MDAO problem

One additional element is required to store the MDAO problem in a CMDOWS file: `problemDefinition`, see Fig. 4.10. In this element, the problem roles and problem formulation are indicated. The creation of this FPG matching the CMDOWS file has been discussed in §§3.4.2. As shown in Fig. 4.10, the parameters z1, z2, and x0 get the special role of design variable. Similarly, f is assigned the role of objective for the optimization. The roles of the executable blocks are also specified and only the tools strictly needed to solve the problem have been selected from the tool repository.

4.4.3. Stage III: MDAO solution strategy

When the problem formulation has been set, in this example to an MDF architecture with a Gauss-Seidel type converger, the full schema is used to store the MDAO solution strategy, as depicted in Fig. 4.11. This strategy is automatically imposed on the MDAO problem using KADMOS, see §§3.4.3 and §§3.4.4. Two new elements are added to the file with respect to the problem definition: the `architectureElements` and the `workflow/processGraph`. Actually, it is not just that these elements are added, but all the elements in the CMDOWS file are updated when the architecture is imposed on the problem. For example, compare the data graphs depicted in Fig. 4.10 and Fig. 4.11 to see the large number of adjusted data connections. With the MDF architecture used in this example, the architectural executable blocks `optimizer` and `converger` are added to the file, and a range of architectural parameters are added, such as `initialGuessDesignVariables` and `couplingCopyVariables`.

This concludes a brief illustration of using CMDOWS to store MDAO systems at different stages. The use of the different CMDOWS elements for each stage of the MDAO system is summarized in Tab. 4.1. Naturally, the schema was not created for such small cases, such as the Sellar problem example, but rather to exchange large-scale systems in a collaborative MDAO setting. The application of the schema will be more elaborately discussed

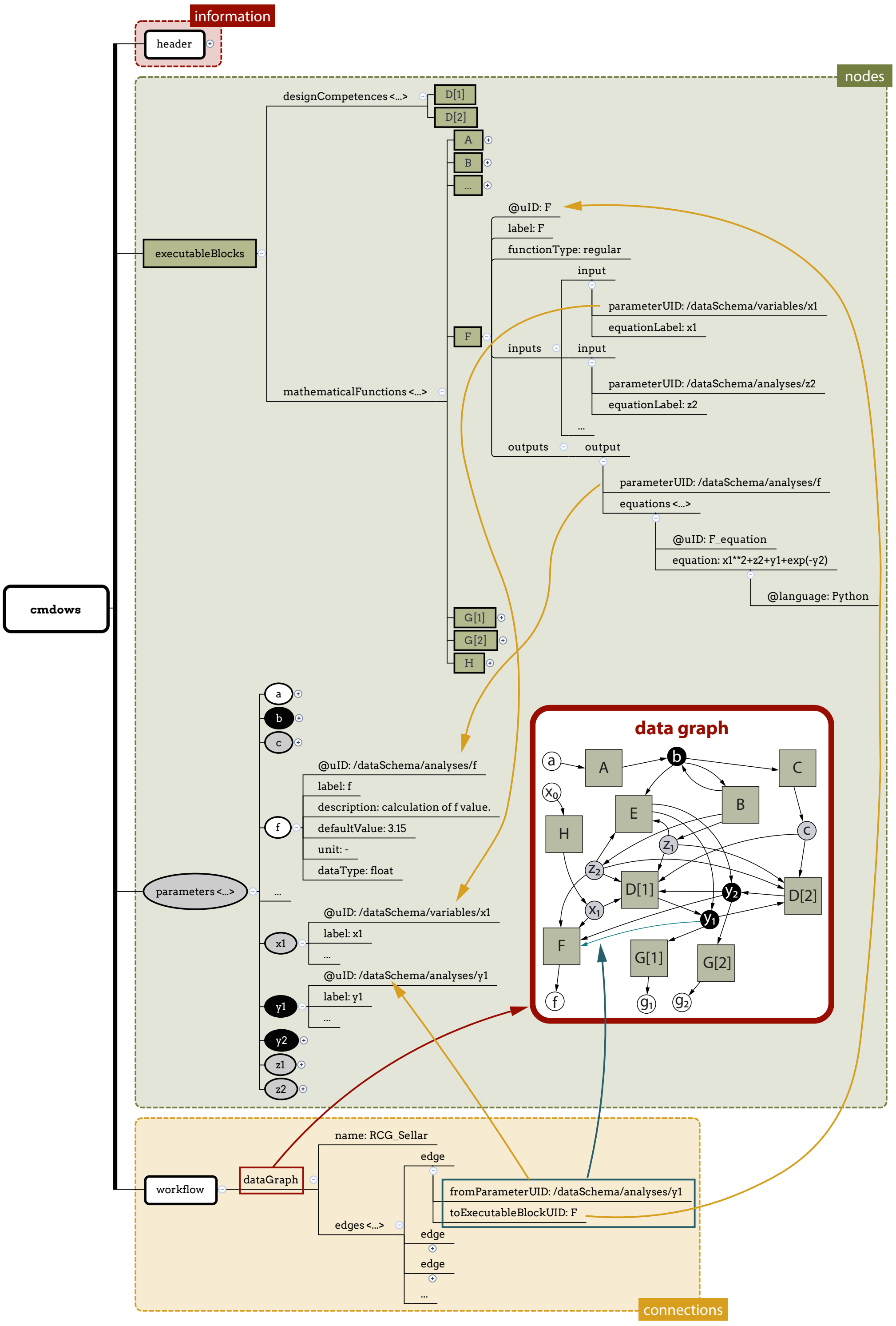


Figure 4.9: Illustration of the storage of the Sellar tool repository in a CMDOWS file. The RCG from Fig. 3.8 is shown in the figure for reference.

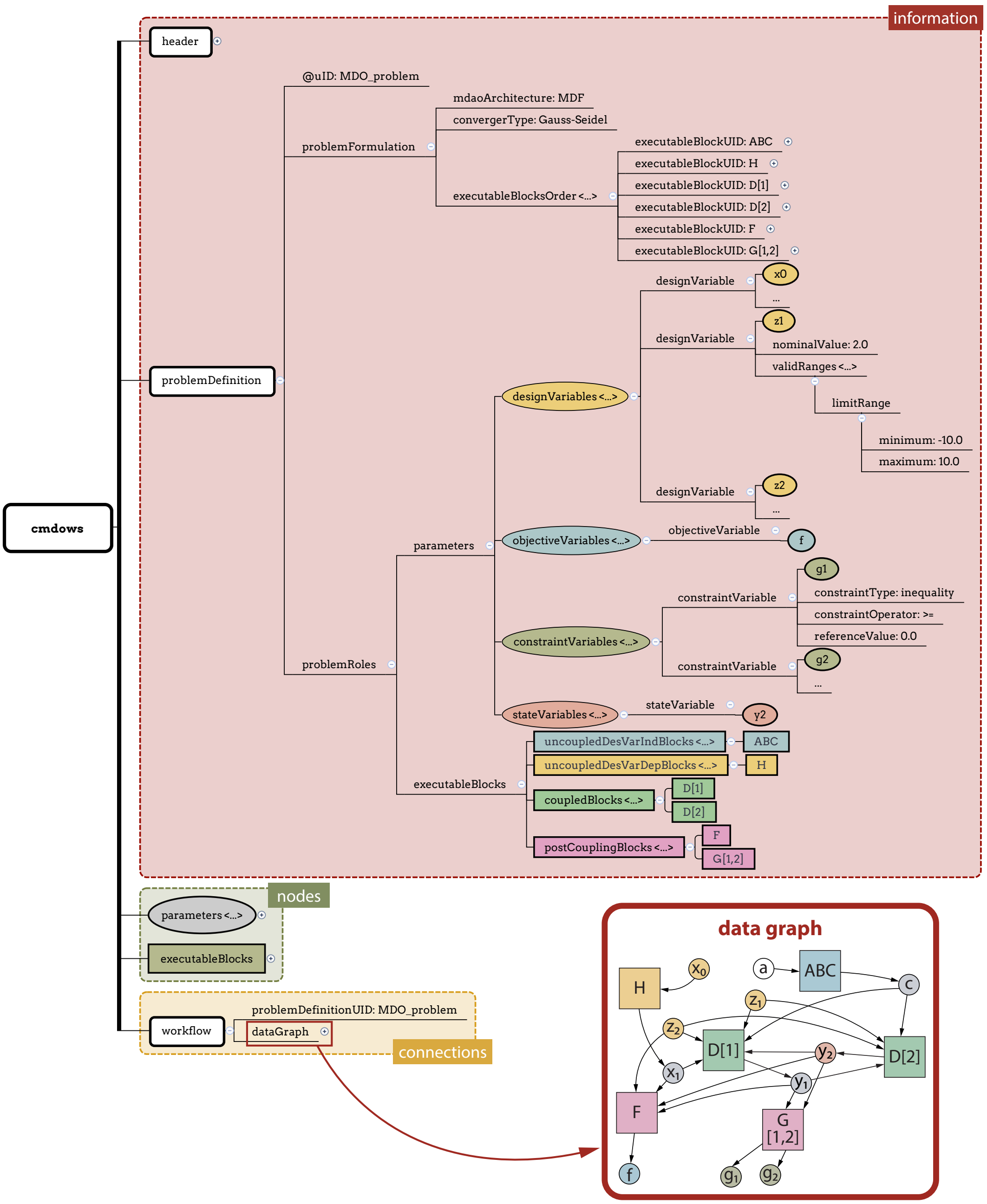


Figure 4.10: Illustration of the storage of the Sellar MDAO problem in a CMDOWS file

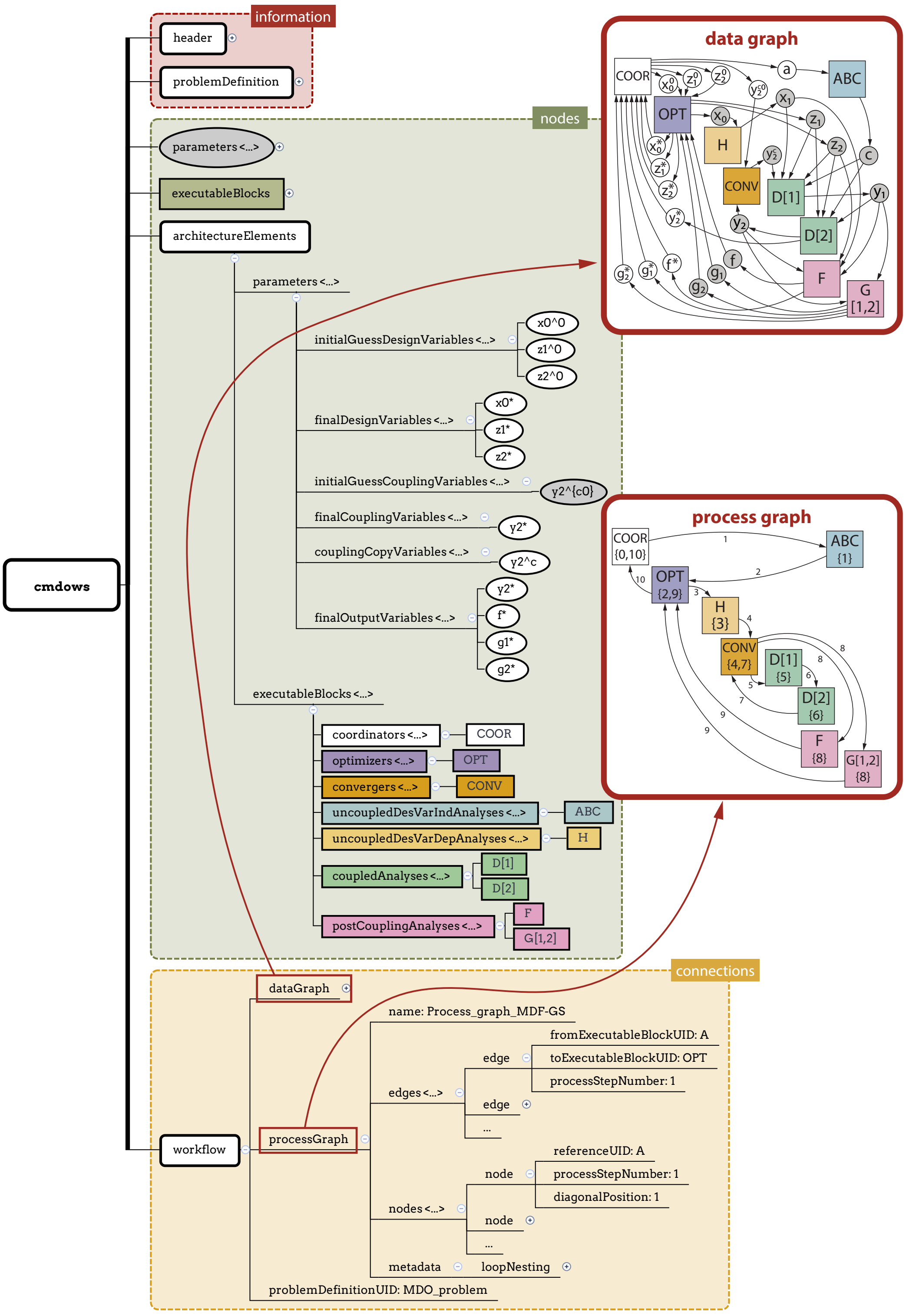


Figure 4.11: Illustration of the storage of the Sellar MDAO solution strategy according to the MDF architecture with a Gauss-Seidel convergence scheme in a CMDOWS file

in the ensuing chapters.

Table 4.1: Tabular top-level overview of the CMDOWS schema for MDAO system representations

schema			**description**	**used to describe MDAO system stage**			**element category**
root	**elements**			**repo**	**problem**	**strategy**	
cmdows	header		Metadata about the file (e.g. creator, creation date, version).	✓	✓	✓	information
	problemDefinition		Definition of the MDAO problem to be solved (design competences used, design variables, constraints, etc.).	×	✓	✓	information
	executableBlocks	designCompetences	Function definition (inputs, outputs, metadata) of executable blocks that are not simple mathematical relations (hence, black boxes).	✓	✓	✓	nodes
		mathematicalFunctions	Function definition (inputs, outputs, equations) of executable blocks that can be described by a set of mathematical relations.	✓	✓	✓	nodes
	parameters		All elements from the product schema that are used as inputs or outputs of the executableBlocks.	✓	✓	✓	nodes
	architectureElements		Additional elements that are required to solve an MDAO problem according to a certain architecture, such as initial guesses, copy variables, or convergers.	×	×	✓	nodes
	workflow	dataGraph	All data input (e.g. parameter x → executableBlock F) and output (e.g. executableBlock F → parameter y) connections of an MDAO system.	✓	✓	✓	connections
		processGraph	Full process description of an MDAO solution strategy using connections and step numbers between exectableBlocks (e.g. F --(step 3)-→ G).	×	×	✓	connections

4.5. Conclusion

The latest version (0.9) of the MDAO system exchange format CMDOWS has been presented in this chapter. CMDOWS supports the storage of an MDAO system of any size at three different stages of the formulation phase: tool repository, MDAO problem, and MDAO solution strategy (see Fig. 4.1). The main goal of CMDOWS is to provide a format that allows different framework applications to exchange the definition of the system.

This chapter has been restricted to a description and illustration of the novel schema to convey the basic underlying ideas and top-level schema definition. The full validation and discussion of the schema will follow later in this dissertation. First, the use of CMDOWS to automatically create executable workflows in multiple PIDO platforms will be covered in the next chapter. Second, the application of CMDOWS at the heart of the third-generation AGILE MDAO framework is demonstrated, analyzed, and discussed in Chapter 6.

5

Bridging the gap between MDAO system formulation and execution

Using KADMOS (Chapter 3), a design team can create the blueprint of an MDAO workflow and afterwards store it as a CMDOWS file (Chapter 4). As helpful as such a blueprint might already be for formulating complex MDAO workflows collaboratively, it cannot be executed yet. Fortunately, many different PIDO platforms are dedicated to this particular task, as was discussed in §2.5. However, these workflows need to be scripted or built manually; a very time-consuming and error-prone task for complex workflows. With the blueprint of the MDAO workflow available as a standalone XML file based on CMDOWS, an opportunity presents itself to automatically build the executable workflows using this information in any PIDO platform. This automated creation of the workflow is called *materialization* and the developed platform capability to achieve it is referred to as a *CMDOWS parser*. After a brief introduction (§5.1), this chapter aims to present and discuss workflow materialization for three PIDO platforms: RCE (§5.2), Optimus (§5.3), and OpenMDAO (§5.4). The chapter ends with §5.5, which contains a comparison on how the different PIDO platforms enable bridging the gap and support collaborative MDAO, followed by recommendations for future developments.

5.1. Introduction

The developments described in this chapter complete the staged MDAO development process that was introduced in Chapter 1, as visualized in Fig. 5.1. Using the development process discussed in this thesis, a graph-based blueprint of the MDAO workflow is available at the end of the formulation phase. This blueprint was formulated using KADMOS and is stored as a CMDOWS file. Typically, the design team would take this blueprint as a reference document and will start building the executable workflow manually using one of the many PIDO platforms available (Tab. 2.1). Manual workflow creation is not a straightforward task, especially if a large amount of data needs to be passed

The contents of §5.3 have been adapted from [102].

around between different disciplines and/or the blueprint describes a complex scheme for solving the design problem, like the distributed architectures CO and BLISS-2000.

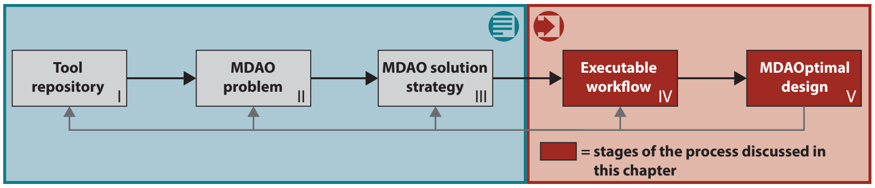

Tool repository I
MDAO problem II
MDAO solution strategy III
Executable workflow IV
MDAOptimal design V
= stages of the process discussed in this chapter

Figure 5.1: The two stages of the MDAO development process that are covered by the workflow materialization discussed in this chapter

Instead of building workflows manually, CMDOWS unlocked the opportunity to automatically generate (hence: materialize) executable workflows based on a single XML file. This idea was actively supported by two prominent PIDO platform developers from the AGILE project, namely Noesis Solutions and the DLR, who saw in the neutral data exchange schema a potential solution to the flexibility demand of their customers, as well as a means to deliver more easily custom solutions to their advanced users. Noesis Solutions and the DLR each own a PIDO platform: Optimus and RCE, which were both discussed in §2.5. In addition to the CMDOWS parsers for these two AGILE platforms, a workflow materialization capability (hence: CMDOWS parser) was also developed for the OpenMDAO platform. Contrary to the other two parsers, which were developed in cooperation with the platform owners, the parser for OpenMDAO was developed in-house for this dissertation. This additional development was motivated by two considerations: 1) a potential performance benefit that was identified based on initial evaluations of this platform, and 2) the need to also materialize blueprints of distributed architectures.

This chapter is compiled from the workflow materialization capabilities developed for three PIDO platforms: RCE, Optimus, and OpenMDAO. These CMDOWS parsers are presented in this chapter for the following purposes:

1. Verify the correctness of the translation by executing materialized workflows and checking the results with theoretical values for different MDAO architectures supported by KADMOS and stored as CMDOWS files:
   - MDA (Gauss-Seidel, Jacobi)
   - DOE
   - monolithic MDO (MDF, IDF)
   - distributed MDO (CO, BLISS-2000).
2. Compare the novel automated development process with a state-of-the-art manual process to:
   - evaluate time gain using the automated process;
   - evaluate how the materialized workflow affects the PIDO platform's performance;
   - evaluate user experience.
3. Validate the automated MDAO development process concerning:
   - independence from PIDO-specific properties and completeness of CMDOWS in describing the solution strategy for the architectures listed above;
   - flexibility to reconfigure the problem definition and/or solution strategy.

The aforementioned targets are not fully covered by all three platforms due to limited time and resources in the AGILE project. Consequently, workflows employing a distributed MDO architecture can only be materialized in OpenMDAO, while the comparison between the automatic and manual approach was only performed with Optimus.

Workflow verification was carried out for each platform using the earlier discussed Sellar problem and SSBJ test cases. The Sellar problem was used as illustrative example in the previous two chapters, see §3.4 and §4.4. The SSBJ case was covered in §3.6 and additional solution strategies are shown in Appendix A. The focus of materialization development was on automating the creation of reliable, executable workflows. Hence, the performance of the workflows is not necessarily optimized, though their execution times will be compared between platforms. The materialized workflows are considered verified if they can be created fully automatically from a given CMDOWS file and their execution results in the expected outcome within a small margin of error. The verification thereby demonstrates the bridging of the formulation-execution gap as well as the value of the platform-agnostic formulation developments from the previous two chapters to achieve a fully automated development process.

5.2. RCE

RCE [S3] is a GUI-based platform, which has been discussed elaborately in §2.5. The platform was used in FrAECs (§§2.6.1), IDEaliSM (§§2.6.2), and the beginning of the AGILE (§§2.7.2) project to manually create collaborative MDAO workflows. RCE does not include any scripting capabilities and therefore the CMDOWS parser had to be build from square one.

5.2.1. Materialization approach

RCE workflows are stored as single files with a '.wf' extension: the workflow file. This file contains the full workflow definition in JSON format. Normally, this workflow file is generated by performing actions in the GUI to add components, connect them, and specify component settings. With the parsing of the CMDOWS file the workflow file is generated automatically. Hence, the XML-based CMDOWS file is translated to the JSON-based RCE format.

The CMDOWS parser has been implemented as a separate plug-in. When a user opens the plug-in an interface appears on the screen. In this interface, the user has to select the CMDOWS file to be parsed. In addition, an XML file that contains initial values for the workflow can also be supplied. A materialized workflow for the SSBJ test case is shown in Fig. 5.2. The matching XDSM of this workflow was shown in Fig. 3.20.

The square components in Fig. 5.2 match with the diagonal components of the XDSM in Fig. 3.20. The round elements are XML readers/writers used to manage XML files and extract I/Os that are required separately by certain components. Following the diagonal from top left to bottom right:

- The coordinator is split into an input and output file block. The input file block is the starting point of the workflow and contains the input file provided in the plug-in's interface. The output file block is the end point of the workflow and collects the optimization results to store them as an XML file.
- The RCE-native optimizer element does not operate on XML files directly, hence

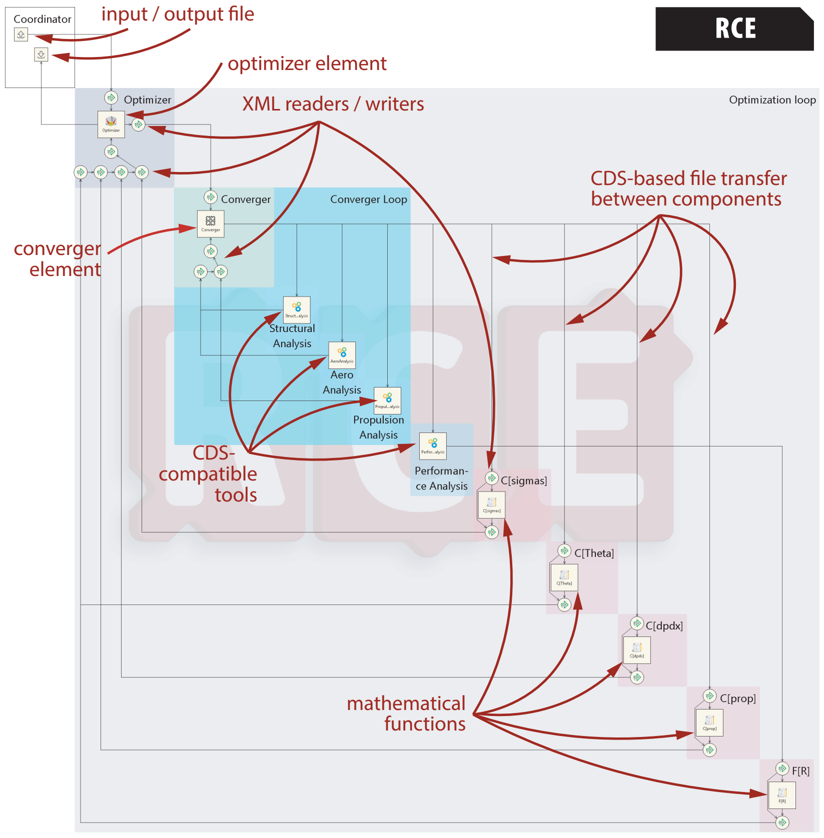


Figure 5.2: Screenshot of the automatically instantiated RCE workflow for the SSBJ validation case for an MDF architecture with a Jacobi convergence scheme. This workflow matches the XDSM shown in Fig. 3.20.

multiple (round) XML elements are added around this block. These blocks read out the optimizer's input from an XML file and write the optimizer's output to an XML file so that the other XML-based tools can directly use these files.
- Similarly, the RCE-native converger element operates on individual values and requires XML elements to operate.
- After the converger, the four CDS-compatible, black-box tools (structures, aerodynamics, propulsion, and performance) are integrated as stand-alone, XML-based components. These tools were stored as design competences in the CMDOWS file. The only prerequisite for a successful materialization, is that locally executed design competences have been integrated as components in the platform's tool library with a matching name and version. The parser then looks for a match between the design competence (name and version) stored in the CMDOWS file and the tools available in the platform library to place the right component on the diagonal. Note that, since the parser assumes that these components are already CDS-based (using a single XML I/O file), no special XML elements are required for their workflow integration.
- The last five square diagonal elements are the mathematical functions from the CMDOWS file, representing the constraints and objective calculations. These components are integrated as Python scripts, using the mathematical relation that is stored in the CMDOWS file. These scripts need individual values (instead of a single XML file) to run, therefore an XML reader is used to read out the inputs before the script element, and a merger is used to write the results after the script and combine them with the input values.
- Finally, the components are connected based on the data connections specified in the CMDOWS file. All connectors in the screenshot are XML file transfers, except the connections between a round and a square element.

Instead of being tools from the local library, the design competences indicated in a CMDOWS file can also be available remotely, on a separate server domain operated by a partner in the design team. In this case, the tool needs to be integrated in the main workflow as a 'Brics component'. Brics, developed by Baalbergen et al. [103] at the Royal Netherlands Aerospace Centre (NLR), is a collection of protocols and middleware that enables data exchange between components residing at two different server domains, whilst respecting the security constraints of both. In RCE, standardized Brics components are available to call a remote tool and to handle such a call on the tool side as well. If remote execution metadata is provided in the CMDOWS file, the design competence will be integrated as a Brics component. In the plug-in the user can provide settings (server address, authentication information, etc.) for distributed workflow execution through Brics. The Brics capability was not used in the verification cases, but will come forward in the next chapter, where RCE workflows are instantiated in collaborative design projects.

5.2.2. Verification of materialized workflows

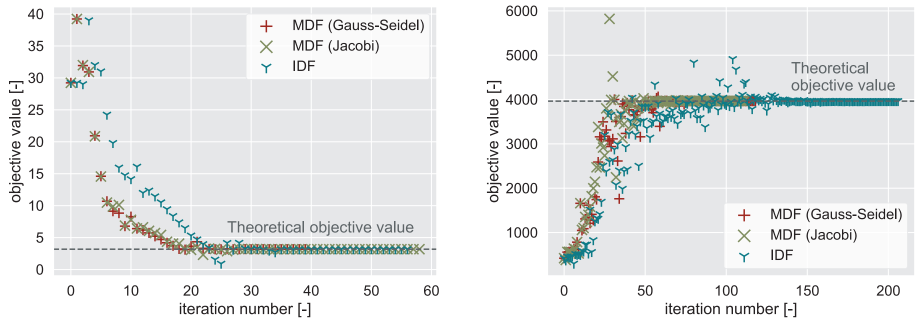
The automatic workflow creation in RCE was validated using two classical MDAO test cases from literature: the Sellar problem and the SSBJ design case. RCE's CMDOWS parser supports all architecture types, except distributed ones. For brevity's sake, the verification presented in this section is limited to three different MDO schemes, as these are the most complex workflows that can be built at the moment. The MDA scheme is implicitly validated, as it is part of the MDF strategy, while the DOE scheme would simply involve the exchange of the optimizer component with a design exploration one. In RCE, the optimizer block provides different algorithms from the Dakota [60] package. For these validations, the COBYLA (Constrained Optimization BY Linear Approximations) algorithm was selected from the package.

The Sellar workflows were instantiated using CMDOWS files containing only mathematical relations. The objective value history per iteration of the COBYLA algorithm is shown for the three architectures in Fig. 5.3a. All three workflows perform as expected and the optimizer is able to find the theoretical objective value and matching design variables. Profiling data and the objective values for the three cases are listed in Tab. 5.1. The Root-Mean-Square Error (RMSE) in the last column is calculated with the following expression for a generic result vector $\boldsymbol{x}$:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\tilde{x}_i - x_i)^2}{N}} \tag{5.1}$$

in which $\tilde{x}$ is the theoretical optimal value, x is the value obtained by the workflow, and N is the length of the vector. The objective value is a single-element vector, but the same expression for the RMSE is used in §5.4 to also verify obtained design variables and couplings.

The execution times for the different cases are comparable, with IDF slightly outperforming MDF in this particular case, though it is worth noting that the performance of IDF depends strongly on the selection of an appropriate starting point; IDF would have more trouble with a starting point where its consistency constraints are not satisfied, while MDF handles coupling consistency with its converger. In general, the optimization times are relatively long for such a small problem in which all components are integrated as native scripts. A direct implementation of the Sellar problem in a Python or MATLAB script would only require a couple of seconds at maximum. This indicates that the materialized workflow with its XML management elements causes quite a lot of overhead, which will be further discussed when the different platforms are compared in §5.5.



a) Sellar b) SSBJ

Figure 5.3: Objective values (denormalized) during execution of materialized RCE workflows for three different MDO architectures

The materialized SSBJ workflows, which contain a mix of mathematical relations and design competences (see Fig. 5.2), also perform as expected, with the objective value history shown in Fig. 5.3b. The IDF scheme requires nearly double the number of iterations, but since it does not enter a convergence loop for each iteration, the execution time is

Table 5.1: Profiling data of the Sellar and SSBJ verification workflows with RCE.

test case	architecture	iters.	execution time (s)	objective	RMSE objective
Sellar	MDF (GS)	40	82	3.1835	9.03E-05
	MDF (J)	59	83	3.1835	1.08E-04
	IDF	57	73	3.1834	1.56E-05
	MDF (GS)	119	1092	3963.9	1.08E-01
SSBJ	MDF (J)	112	1203	3967.0	2.95E+00
	IDF	207	996	3966.2	2.24E+00

comparable to MDF. Performance of the workflows could potentially be improved by fine-tuning the tolerance settings of the optimizer and converger, as the objective history reveals that all workflows perform many iterations close to the theoretical optimum. However, such performance improvements are out of scope of the current verification study, as was described in §5.1.

5.2.3. Development process validation

The verification of RCE's CMDOWS plug-in demonstrates the possibility to build a complete executable workflow in a specific PIDO software, based on the platform-agnostic definition of that workflow in a CMDOWS file. No additional user intervention was required to generate and execute the workflows through the plug-in. Thereby, the verification not only covers the plug-in itself, but also validates the standardized workflow schema from the previous chapter. The RCE CMDOWS plug-in is an important step in the automation of the full MDAO development process, which will be further assessed based on the collaborative MDAO design projects covered in the next chapter. A strong point of the automatically instantiated workflows in RCE is that, after materialization, the workflows are still fully accessible in the GUI for any further adjustments, such as fine-tuning component settings or manually including an additional component.

5.3. Optimus

Optimus [S2] is a commercial PIDO environment developed by Noesis Solutions. As with RCE, Optimus provides a GUI to build workflows. A key difference between the two platforms, also discussed in §2.5, is that Optimus combines simulation workflows built from tool components with separate analysis methods (e.g. optimizer, DOE). Hence, there is no explicit optimization component in the workflow, but the optimizer is configured 'on top' of the workflow itself.

5.3.1. Materialization approach

In addition to the standard GUI-based workflow generation, Optimus also provides a Python-API that enables the programmatic creation of workflows through scripts. This advanced feature is not targeted to single task workflow generation, but is oriented towards more complex applications where multiple instances of a workflow must be created and customized efficiently. This Python-API is used to automatically create the complex MDAO workflows formulated by KADMOS and stored as CMDOWS files.

The materialization is achieved by performing a sequence of operations that translate the CMDOWS formalism into Optimus workflow features. The translation tasks are scheduled according to a fixed sequence to ensure the coherence of the disciplines and their links. This is necessary due to the underlying ordered-graph schema used internally by Optimus. Thereby the automated procedure literally mimics the operations that would have been performed by a specialist through the GUI using an XDSM (or equivalent) as reference.

As a first step, the XML-type dataset is parsed into a Python dictionary. The elements of the workflow and their roles are explicitly stated in the CMDOWS file, but some reordering and clustering are needed. The CMDOWS parser takes the following steps, based on extracted information from the CMDOWS file:

1. Define design variables, along with their name, type (float, integer, string), extension (high-low, stepped, list) and range.
2. Identify parameters; inputs with a constant value.
3. Characterize objective functions and related definition (evaluation method, formula, involved variables).
4. Introduce constraints (limit values, type).
5. Prepare couplings (connection schema of the elements in the workflow).
6. Add components (or disciplines) based on their required inputs and produced outputs.
7. Set execution order.
8. Specify architecture-related components, such as cycles and target variables.
9. Perform equation conversion to ensure correct interpretation and required variable management.

The materialized workflow is structured with two levels, as depicted in Fig. 5.4: architecture and discipline. In case of remote disciplines, an intermediate level is introduced, as discussed later in this section. The single architecture level at the top represents the MDAO solution strategy, and the lower levels are populated by multiple, single tool, independent disciplinary workflows. The top-level workflow is the only one that retains global knowledge of the ongoing analysis, whereas at the remote connection and discipline level ‘separation of concerns’ is implemented.

The data transfer between the architecture-level Optimus workflow and the disciplinary Optimus workflows is ensured by an integrated feature called ‘Optimus-In-Optimus’ [104]. This feature allows a master workflow to invoke a nested subworkflow and transmit the necessary variable values. The two square blocks with a blue ’e’ in Fig. 5.4 are the nested subworkflows. This feature enables clean and modular workflows, positioning discipline-specific elements, such as commercial off-the-shelf software interfaces and file pre- and post-processing, at the appropriate level. The disciplinary workflows are linked to the architecture level only after cross-checking all the variables to ensure that every variable in the MDAO system is mapped to its counterpart in the discipline. This operation saves the user from a time-consuming and error-prone procedure.

As discussed in the previous section, the CMDOWS file supports distributed execution with remote disciplines through Brics. For these type of components, an intermediate remote connection level is introduced to manage data transfer and handle potential synchronization issues. Of the two workflows in the lower left of Fig. 5.4, the top workflow is executed locally and requests the execution of a remote discipline. Thanks to Brics, the lower workflow can reside on another server domain and communicates with the main

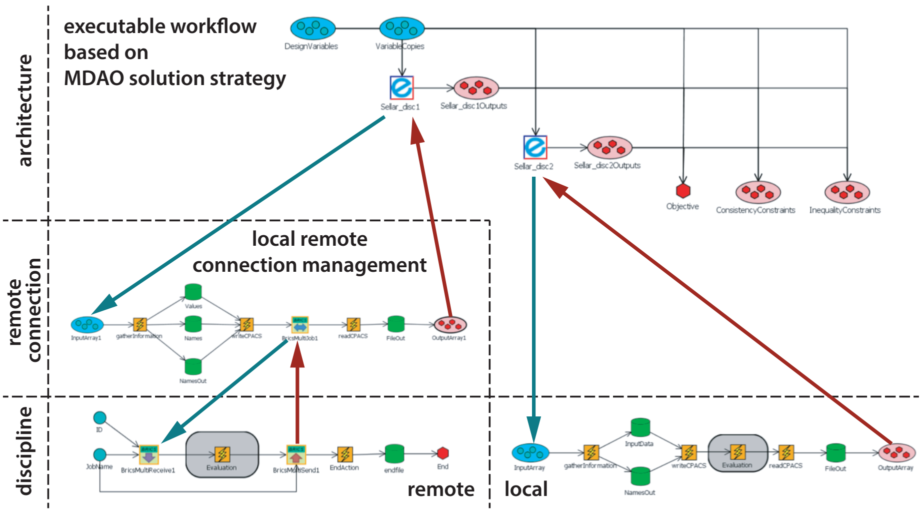


Figure 5.4: The multi-level structure used to automatically build executable workflows from CMDOWS files in Optimus

workflow through the Brics elements in the remote connection workflow. The set-up of the connection interfaces (remote or local) is based on metadata stored in the CMDOWS file. One of the great advantages of the remote connection level is that the remote discipline itself can actually be a workflow preassembled in any other PIDO platform that supports Brics.

After the script-based materialization is complete, the workflow can be inspected in the GUI and if need be, edited. Intrusive modifications, such as re-routing of connections, are possible, but in large MDAO systems this could introduce consistency issues. Therefore, the most convenient approach is to perform structural or architectural changes through KADMOS and re-materialize the workflow. The materialized workflow for the SSBJ optimization case with an MDF architecture is shown in Fig. 5.5.

5.3.2. Verification of materialized workflows

The CMDOWS parser was verified using the same test cases discussed in the previous section. The objective value history per iteration is shown in Fig. 5.6. Note that the Sequential Quadratic Programming (SQP) optimization algorithm has been selected from the Optimus library (instead of the previously used COBYLA algorithm in RCE). This algorithm requires the system's derivatives which are calculated using the finite-difference method. The achieved objective values and profiling data are listed in Tab. 5.2. All design variables and constraints also match their theoretical value, but this is not shown here for conciseness. Hence, the implemented automated approach identified the optimal design points, thereby successfully validating the automatically built workflows.

5.3.3. Comparison manual and automated process

The two verification cases from the previous section have also been performed manually based on a state-of-the-art approach, to be able to assess the complete automated MDAO development process shown in Fig. 5.1 (hence using KADMOS, CMDOWS and

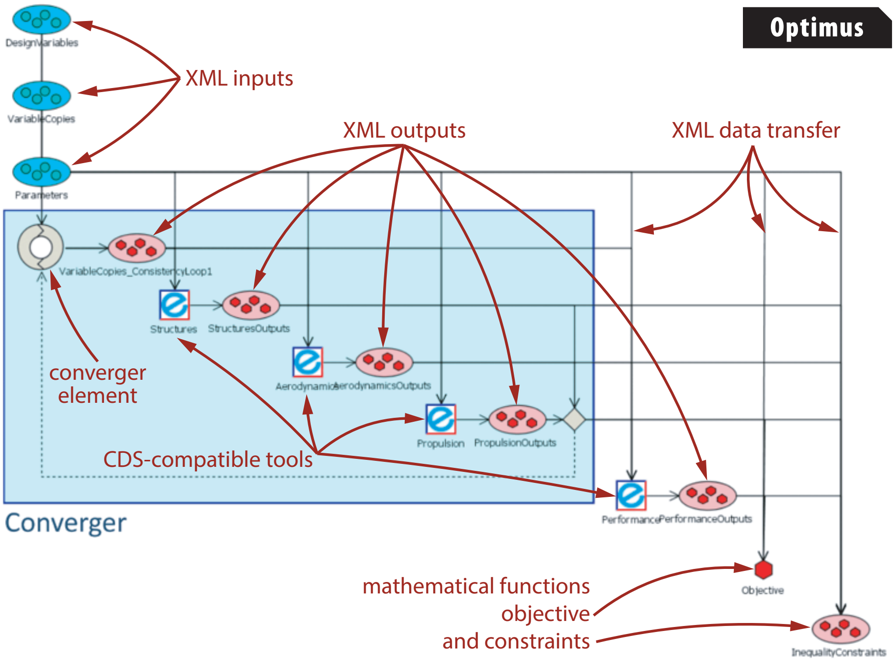


Figure 5.5: Screenshot of materialized Optimus workflow for the SSBJ verification case for an MDF architecture with a Gauss-Seidel convergence scheme. This workflow matches the XDSM shown in Fig. A.1.

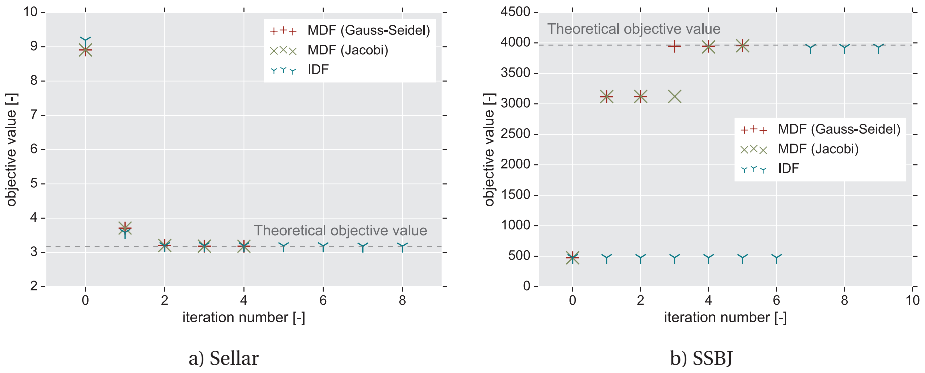


Figure 5.6: Objective values (denormalized) during execution of materialized Optimus workflows for three different MDO architectures

Table 5.2: Profiling data of the Sellar and SSBJ verification workflows with Optimus.

test case	architecture	iters.	execution time (s)	objective	RMSE objective
Sellar	MDF (GS)	5	394	3.1833	5.95E-05
	MDF (J)	5	538	3.1834	2.82E-05
	IDF	9	268	3.1834	5.52E-05
SSBJ	MDF (GS)	6	3047	3954.1	9.87E+00
	MDF (J)	6	5811	3954.0	1.00E+01
	IDF	10	2417	3927.6	3.64E+01

materialization in Optimus). The development process was first followed to configure optimization workflows based on the MDF architecture with a Gauss-Seidel convergence scheme. Subsequently, two reconfigurations were performed to change the architecture: first to MDF with a Jacobi scheme and finally to IDF. The goal of this development process simulation in manual and automated mode was to quantify three aspects for system configuration and two successive reconfigurations:

- time difference between manual and KADMOS-based MDAO system formulation;
- time difference between manual workflow creation and automated materialization, while accounting for the difference in the preceding formulation phase (materialization requires a CMDOWS and hence KADMOS-based formulation);
- execution time difference between manually created and materialized workflows.

In the manual approach, the formulation of the MDAO system was performed without KADMOS by creating hand-written N^2 charts, problem descriptions, and XDSMs on paper. The workflows were also created manually using the GUI of Optimus. An example of a manually created workflow in Optimus is shown in Fig. 5.7. With the manual approach all components are placed in a single workflow, contrary to the multi-level approach visualized in Fig. 5.4. The automated approach exploits the developments presented in this dissertation: KADMOS to formulate the solution strategy based on a tool repository, CMDOWS to store the solution strategy, and the Optimus CMDOWS parser to create the workflow. All activities were timed to compare the two approaches. The recorded timings per activity are listed in Tab. 5.4 and 5.3.

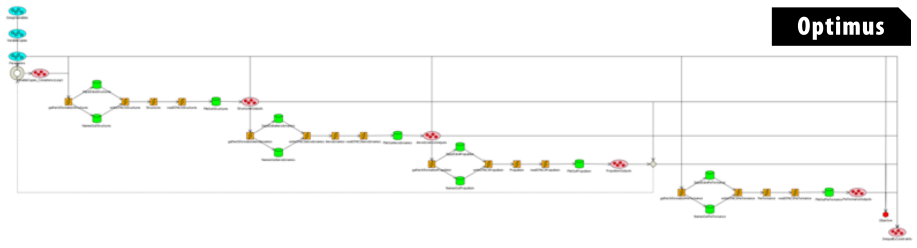


Figure 5.7: Manually created workflow for the SSBJ test case

Table 5.3: Time (minutes) required to perform each step for the different test cases and architectures with hand-written system formulations (hence, no KADMOS script) and manual workflow assembly using the Optimus GUI

stage & activity		Sellar			SSBJ		
		confi-gure for MDF-GS	reconfi-gure to MDF-J	reconfi-gure to IDF	confi-gure for MDF-GS	reconfi-gure to MDF-J	reconfi-gure to IDF
I-II	formulate problem	4	-	-	5	-	-
III	create XDSM	5	3	4	25	8	10
IV	create workflow	10	3	3	30	5	5
V	execution	~0	~0	~0	<1	<1	<1
	total	19	6	7	60	13	25

The comparison in terms of total time spent is listed in Tab. 5.5. Looking at the total timings for the configuration of the Sellar test case, the automated approach clearly does

Table 5.4: Time (minutes) required to perform each step for the different test cases and architectures using the automated approach

stage & activity		Sellar			SSBJ		
		confi-gure for MDF-GS	reconfi-gure to MDF-J	reconfi-gure to IDF	confi-gure for MDF-GS	reconfi-gure to MDF-J	reconfi-gure to IDF
I	load repository	15	-	-	2	-	-
	inspect and debug	2	-	-	3	-	-
II	formulate problem	5	-	-	14	-	-
	inspect and debug	2	-	-	2	-	-
III	create XDSM+CMDOWS	2	2	3	2	1	5
	inspect and debug	1	1	1	1	1	1
IV	create workflow	1	1	1	2	2	2
V	execution	7	9	4	51	97	40
	total	35	13	9	77	101	48

Table 5.5: Overview of the total time (minutes) required to formulate and create the executable workflow for an MDAO systems using the manual and automated approach

			Sellar		SSBJ	
stages	activity		conf.	conf. + 2 reconf.	conf.	conf. + 2 reconf.
I-III	formulation	manual	9	16	30	48
		auto.	27	34	24	32
		diff. [%]	+200%	+113%	-20%	-33%
IV	workflow creation	manual	10	16	30	40
		auto.	1	3	2	6
		diff. [%]	-90%	-81%	-93%	-85%
I-IV	formulation & workflow creation	manual	19	32	60	88
		auto.	28	37	26	38
		diff. [%]	+47%	+16%	-57%	-57%

not outperform the state of the art. This can be attributed to the formulation phase, which has a 200% time increase. That is not surprising, since for a small case like the Sellar problem manual formulation seems to be more appropriate than going through the formal KADMOS process. In addition, this difference can also partly be attributed to the fact that the full system of mathematical relations needs to be defined in KADMOS (since they are needed in the CMDOWS file), which apparently takes more time than through the Optimus GUI. Workflow creation in the automated approach is much faster thanks to the CMDOWS parser, however, the execution time of the workflow is increased significantly by the additional overhead of the multilevel approach (see Fig. 5.4) used for materialization.

The small Sellar case does already offer an indication of the strength of using the automated approach. When the additional time for two system reconfigurations is included (see the second Sellar column in Tab. 5.5), the automated approach caches up: the formulation still takes longer, but the time savings in workflow creation bring the total difference down from 47% to 16%. In conclusion, the small Sellar case would be done quicker manually, regarding both system formulation and workflow execution, though the potential of the automated approach is visible when system reconfigurations are included.

In the SSBJ test case the benefits of the automated approach are apparent: both formulation and workflow creation are quicker with the automated approach, even when system reconfigurations are not included. The automated approach reduces the time spent until the workflow is ready for execution by 57%. As the SSBJ case represents a more realistic collaborative MDAO system, though still relatively small, the achieved time reductions indicate how a design team can benefit from the automated approach. In addition, it is worth noting that time is spent differently with the new methodology: in KADMOS, there is more focus on formulating the right problem and reasoning how the system should be composed, instead of spending valuable minutes on repetitive tasks to manage the system formulation (e.g. updating N^2 charts or XDSMs) or build workflows (e.g. creating all the connections between tools). Another noteworthy aspect is that, for the comparison presented here, the performed system reconfigurations entail a change of architecture, but one can easily imagine how, in the industrial practice, an MDAO system would require many different kinds of reconfigurations, such as adjusting a tool's inputs/outputs, including a new tool, or changing the optimization problem definition. All such reconfigurations are supported by the current automated process.

Looking closer at stage IV, the time required to automatically assemble the workflow is barely influenced by the architecture; only for high-dimensionality problems tackled with the IDF approach the creation of additional variables could have a relevant impact. In addition, the manual creation times in Tab. 5.3 for the different architectures rely heavily on the similarity between the architectures, making it possible to clone one workflow and then edit it; consequently, the implementation of the first architecture is significantly slower than the customization process. However, this approach has clear limitations: IDF and MDF have some commonalities, whereas distributed architectures (not used in this comparative study) are radically different, and their implementation should be performed starting almost from scratch. Furthermore, larger systems would feature a large set of variables and couplings that could be non-trivial to adapt from one architecture to another without creating inconsistencies that would be challenging to identify and correct. This is a limitation that is not relevant for the automated approach, where any architecture would be instantiated almost instantly without relying on clones of earlier workflows.

In defense of the manual approach, the automated approach does require two main investments. The first investment is that the tool repository needs to be created using the CDS approach. In this work the creation (N.B. but not loading and inspecting it in the automated approach) of this tool repository was neglected, as it was assumed that the same repository is used for both approaches. However, compliance with the schema is an additional task at the disciplinary level, but pays off in the achieved agility at the system level, as it enables the use of the automation tools KADMOS, CMDOWS, and workflow materialization in Optimus. The second investment stems from the fact that the design team has to familiarize itself with these automation tools. Though this might seem a trivial task, in larger organizations such cultural changes are generally harder to achieve than expected.

In terms of workflow execution time, the manual approach, which exploits a direct integration of the disciplinary tools as elements in the top level (see the SSBJ workflow depicted in Fig. 5.7), outperforms the multilevel materialization. This significant improvement in execution time is achieved by avoiding the overhead of the Optimus-in-Optimus construct of the parsed workflows as, in the test cases used, the overhead is relatively high for the fast-computing disciplines. However, as the complexity of the implemented disciplines and their run time grow, the Optimus-In-Optimus overhead becomes less relevant (often negligible) and is justified by the improved visibility of the architecture granted by the multilevel approach, as well as the ease to re-use pre-assembled disciplinary sub-workflows inside different architecture workflows, for example when the MDAO architecture is changed.

5.3.4. Development process validation

The verification of CMDOWS-based workflow materialization in Optimus strengthens the validation of the overall MDAO development process further. Optimus is the second platform in which workflows can be created fully automatically based on the same CMDOWS files used with the RCE plug-in. Thereby, it confirms the platform-agnostic nature of the CMDOWS definition. Additionally, the comparative study between the state-of-the-art and automated approach showed that, as system complexity grows, the automated formulation and materialization becomes significantly faster than the manual approach, with an estimated time reduction not lower than 50%.

Future work with respect to the Optimus CMDOWS parser will focus on expanding the list of architectures that can be instantiated. For example, distributed architectures including surrogate models (e.g. BLISS-2000) are expected to provide a performance benefit for distributed systems. Other work will focus on further benchmarking the automated approach with respect to the state of the art, by performing different types of system reconfigurations (e.g. tool removal/addition, problem reformulation).

5.4. OpenMDAO

OpenMDAO [S6] is an open-source, Python- and script-based platform that was addressed in §2.5. The main classes of the OpenMDAO package are shown on the left of Fig. 5.8. Using Python scripts, a user can manually define the executable workflow as an OpenMDAO `Problem` instance. The problem object has two main attributes: model and driver. The model is instantiated using the `System` class and represents the multidisciplinary system of analyses. The driver is a `Driver` instance and contains the optimizer or another design space exploration algorithm. Convergence of cycles in the multidisciplinary system is performed through solvers. These solvers are defined on the system itself and can also be configured per subsystem, as the `System`-`Group` composition in Fig. 5.8 allows a hierarchical system definition. Note that, conceptually, the set-up of the executable workflow is similar to the structure used in Optimus: the system of multidisciplinary analysis is modeled first, convergence is defined in that system, and the analysis algorithm (e.g. optimizer) is defined separately.

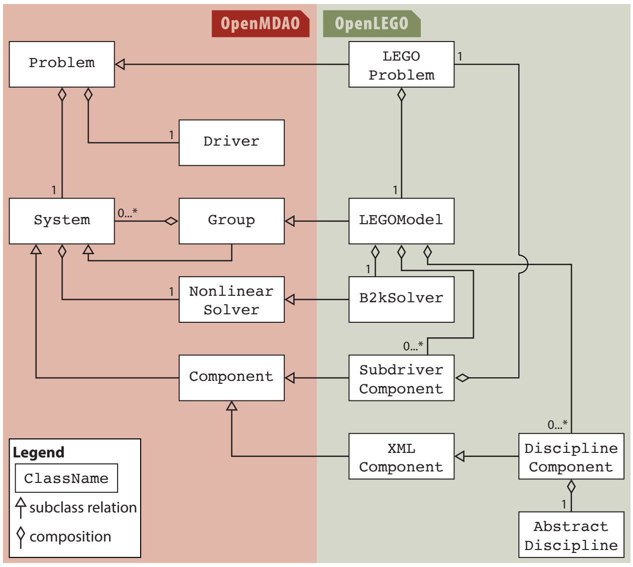


Figure 5.8: UML class diagram of the OpenLEGO package and the relations between the OpenLEGO classes and the OpenMDAO package

Two extensions of the OpenMDAO platform were required for the work presented in this dissertation. First of all, the creation of the `Problem` instance, including its model and driver, should be automatic, based on the workflow blueprint defined in the CMDOWS file. Secondly, distributed architectures should also be supported. The set-up of OpenMDAO does not support the inclusion of multiple drivers in the same `Problem` instance, which is a prerequisite for distributed architectures. For these purposes, a new Python package was developed called OpenLEGO (Open-source Link between AGILE and OpenMDAO) [S25]. Hence, OpenLEGO is the CMDOWS parser developed for the OpenMDAO platform.

5.4.1. Materialization approach: OpenLEGO

The UML class diagram of OpenLEGO including the relation of the OpenLEGO classes to OpenMDAO's classes is shown in Fig. 5.8. The development of OpenLEGO was initialized by De Vries [105]. In his work, De Vries focused on the first extension for OpenMDAO: the automatic instantiation of the model from a CMDOWS file. To achieve this, four classes were created in OpenLEGO: `LEGOModel`, `XMLComponent`, `DisciplineComponent`, and `AbstractDiscipline`. The latter three will be discussed first.

One of the key capabilities of OpenLEGO is the ability to integrate disciplinary tools from the repository as a component in the OpenMDAO model. In the RCE and Optimus parsers, this was achieved by requiring the user to either integrate these tools as local components in the platform or by establishing a remote execution through Brics. In OpenMDAO a similar requirement could be defined, however, the package goes one step further and provides a more flexible and automatic tool integration. To achieve this flexibility, three separate classes are used for tool integration/execution, namely `AbstractDiscipline`, `XMLComponent`, and `DisciplineComponent`. The creation of these classes is motivated by requiring a strict separation between a tool's definition in the formulation phase and the tool's integration in an executable workflow in the execution phase. Hence, we want the tool definition to be fully independent of the platform selected in the execution phase. The implemented approach achieving this is visualized in Fig. 5.9.

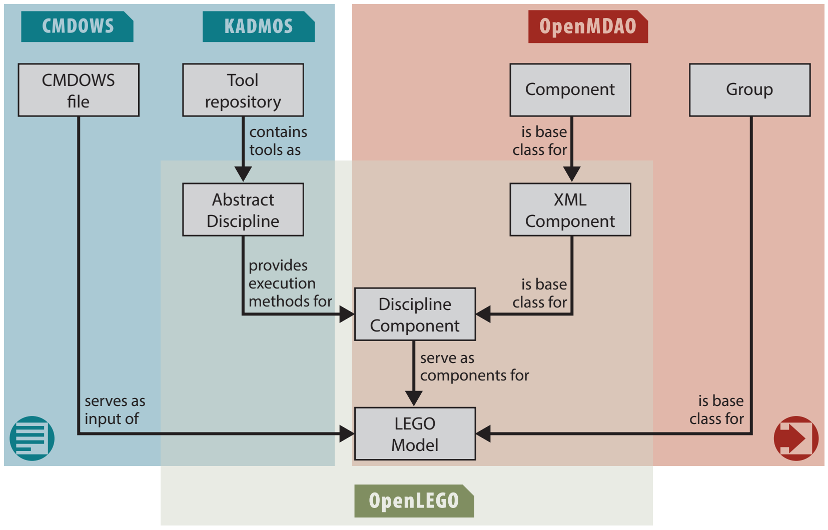


Figure 5.9: The implemented strategy in OpenLEGO to bridge the gap between the formulation phase (with KADMOS and CMDOWS) and the execution phase (with OpenMDAO). (adapted from [105])

In the formulation phase the disciplinary tool needs to be integrated as an `Abstract-Discipline` instance. This class is independent of OpenMDAO and provides a template to integrate a disciplinary tool as a Python class. The integration is based on the CDS approach, meaning that tools are assumed to have a single XML file as I/O. To enable the execution of XML-based components in OpenMDAO, the `Component` class was extended with `XMLComponent`. This class only depends on OpenMDAO and could also be used in OpenMDAO as such. Finally, the `DisciplineComponent` merges the tool definition from the tool repository and the `XMLComponent` class. The relation between the three classes is also shown in Fig. 5.8, where `Disciplinecomponent` is a subclass of `XMLComponent` and automatically incorporates the relevant methods from the `AbstractDiscipline` to execute the tool.[*] In addition to XML-based tool execution, the `AbstractDiscipline` also supports the definition of analytic derivatives for the components. If the computation of these partial derivatives is supported by a tool, Open-

[*]In the previous sections, the disciplinary tools used were actually also Python subclasses of OpenLEGO's `AbstractDiscipline`. Hence, this construct streamlined the execution of the same tool in three different PIDO platforms.

LEGO will notice and automatically configure the tool accordingly in OpenMDAO. For more details on this aspect of OpenLEGO, see [105].

With the integration of disciplinary tools as OpenMDAO components covered, the next step is to automatically instantiate the full OpenMDAO model based on the CMDOWS file. This automated process is enabled by the `LEGOModel` class. As shown in Fig. 5.8, `LEGOModel` is a subclass of OpenMDAO's `Group` class. A `LEGOModel` instance requires two inputs: the location of the CMDOWS file and the location of the tool repository containing the Python files of the design competences. The OpenMDAO model is then automatically constructed based on the information stored in the CMDOWS file. This construction includes the following steps in the `setup` method of the class:

1. Add coordinator (`IndepVarComp` in OpenMDAO) for system-level inputs.
2. Add superdriver components (covered later in this section) .
3. Configure system-level converger for BLISS-2000 (covered later in this section)
4. Add all analysis and surrogate model components depending on KADMOS categorization (see §3.4) and type:
   (a) add design competences for uncoupled and post-coupling components;
   (b) add mathematical functions for uncoupled and post-coupling components;
   (c) add surrogate model components;
   (d) add coupled components (design competences and mathematical functions) and configure the solver for this group (e.g. Gauss-Seidel, Jacobi).
5. Add subdriver components (covered later in this section).
6. Set the order of the different components.
7. Define the design variables.
8. Define the constraints.
9. Define the objective.

After the set-up of the model is completed, the `LEGOModel` object still acts as a regular model from OpenMDAO. This means that all OpenMDAO methods can also be used for the OpenLEGO models.

Apart from the definition of the model, the CMDOWS file also stores information on the algorithms to be used in the MDAO solution strategy. These algorithms should be set as a `Driver` on the `Problem` instance, as depicted in Fig. 5.8. The materialization of the `Problem` instance is performed by the `LEGOProblem` class. In this class, the model is instantiated using the previously described `LEGOModel` and the driver is configured automatically based on the information stored in the CMDOWS file.

This first extension of OpenMDAO with OpenLEGO enables the materialization of executable workflows for all monolithic architectures. Fig. 5.10 visualizes the classes involved in the materialization of such a monolithic solution strategy. The XDSM in the background of this figure matches the XDSM that was defined by KADMOS in §§3.4.3 and §§3.4.4, see Fig. 3.10c. Note how the analysis tools in the system (e.g. ABC, H, D[1]) are either implemented as instances of the `ExecComp` or the `DisciplineComponent` classes, respectively for mathematical functions and design competences listed in the CMDOWS file.

The second extension, support for the execution of distributed architectures, required the addition of two more classes in OpenLEGO: `B2kSolver` and `SubdriverComponent`. The `B2kSolver` (B2k is a shorthanded notation for the BLISS-2000 abbreviation) implements the distributed system converger component, see the third compo-

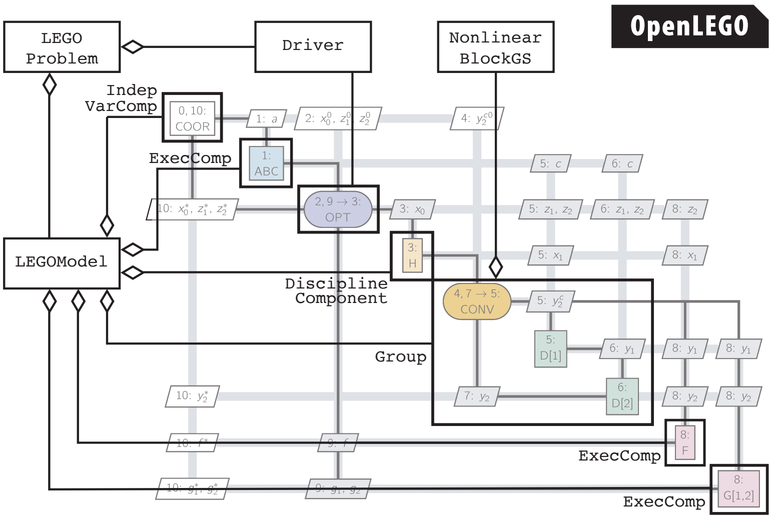


Figure 5.10: Overview of the OpenMDAO and OpenLEGO classes involved in the materialization of a monolithic solution strategy (MDF with Gauss-Seidel convergence) for the Sellar problem solution strategy that was depicted in Fig. 3.10c

nent on the diagonal of the XDSM in Fig. 3.26a. This solver contains the convergence logic of the BLISS-2000 architecture (addressed in §2.4) and performs two main tasks:

- adjust design variable bounds after each system-level optimization to reduce the design space;
- assess system convergence based on the objective value returned by the system-level optimization.

The solver has been written as a subclass of the `NonlinearBlockGS` solver from OpenMDAO, so that the sequential execution of model components is replicated from that solver in combination with a completely different convergence check and the adjustment of the design variable bounds for all optimizers and DOEs present in the model. The solver code was ported to Python based on the BLISS-2000 implementation developed in MATLAB by Agte [106].

Finally, the `SubdriverComponent` class represents the last key development to support distributed architectures with OpenLEGO. When placed in an OpenMDAO model, this component acts as an explicit function; hence, it simply provides output values based on provided inputs. However, inside the component a `LEGOProblem` including `LEGOModel` are instantiated and when the component runs, it will execute a full model and driver. Thanks to this construct, a system with multiple, nested, optimizers and DOEs can be established and executed, thereby supporting distributed architectures.

The different classes of the OpenLEGO package are also aware of this nested structure for distributed architectures and correctly connect the different models to pass data around. When nested models require input values, these always have to be passed down from the highest level. For this purpose, a superdriver component is instantiated in the `LEGOModel`, see step 2 in the construction steps of the model.

The system hierarchy for the OpenLEGO model is determined based on the process graph that is stored in the CMDOWS file. If, apart from the main model driver, another driver is encountered in this process hierarchy, then this driver is instantiated as a `SubdriverComponent` and the components within that subdriver's process are instantiated in a separate `LEGOModel` instance. This `LEGOModel` is constructed based on the same CMDOWS file, but only includes components that are part of the process hierarchy beneath that particular subdriver's UID. Thus, the set-up method recursively builds subdriver components until all drivers from the CMDOWS process hierarchy have been modeled.

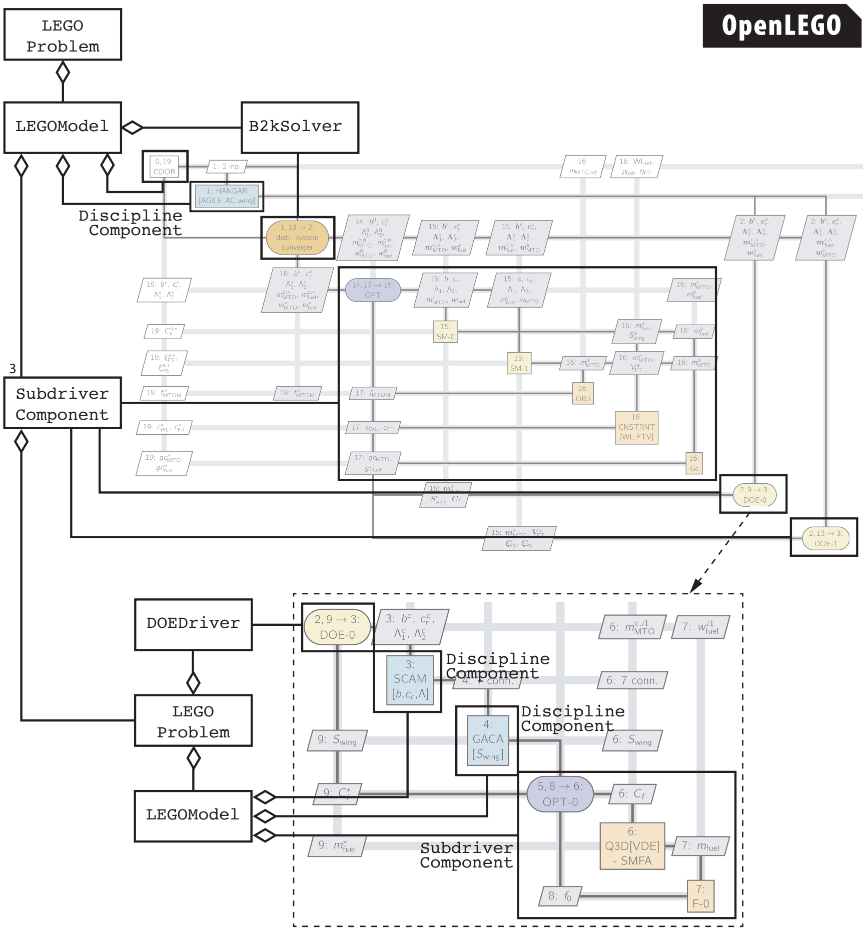


Figure 5.11: Overview of the OpenMDAO and OpenLEGO classes involved in the materialization of a distributed MDO architecture (BLISS-2000) for the wing design problem that was depicted in Fig. 3.26

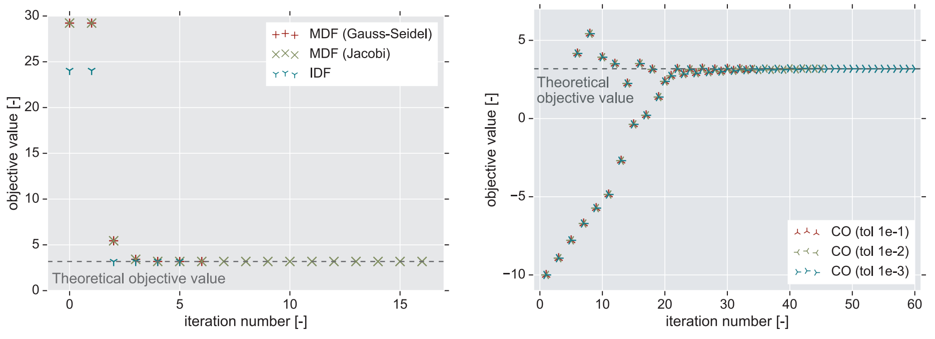
Fig. 5.11 shows how OpenLEGO would construct the hierarchical process of the BLISS-2000 solution strategy that was formulated in the wing design case study in §3.7. The XDSM in the top of the figure is the main level in the process hierarchy. At this level the model contains three subdrivers, one for the system-level optimization and two for the disciplinary DOEs. This model also contains an instance of the `B2kSolver`. At the bottom of the figure an expanded XDSM for one of the disciplinary DOEs is shown. Here, the subdriver contains its own model and driver to construct a nested `LEGOProblem` instance. Each experiment from the DOE block then also needs to be optimized: hence, the DOE subdriver nests another `SubdriverComponent` instance to perform the disciplinary optimization. In total, the full system in this example contains six different models and five drivers. By virtue of the second extension of OpenMDAO by OpenLEGO, a CMDOWS file describing such a large and complex system definition can be automatically created. In the upcoming section these workflows are verified based on two test cases.

5.4.2. Verification of materialized workflows

As for RCE and Optimus, the Sellar problem and SSBJ test cases are used to verify the OpenMDAO workflows. The full range of workflows that can be formulated with KADMOS, from single tool execution in a test run to distributed architectures, was tested and verified. The tested architectures are listed in the first column of Tab. 5.6 (Sellar) and Tab. 5.7 (SSBJ). With the Sellar problem, the tools were integrated both as a system of mathematical functions and a system of design competences including analytic derivatives. This enabled a comparison between different tool integration methods for the same test case.

Monolithic architectures

The profiling data for the Sellar case is shown in Tab. 5.6. The first five architectures in the table do not involve any optimization. Results of these workflows are in line with the expectations. Looking at the execution times, it is clear that the implementation of tools as mathematical functions has a positive impact on the run time. This is not surprising, as design competences require the intermediate reading and writing of XML files to pass data around, while the mathematical functions pass their data directly within the OpenMDAO model. The number of function calls for each tool is the same for the first five architectures, since these architectures do not require any derivative values to run the workflow. Larger differences between the tool integration approaches become apparent for the monolithic optimization strategies. The history of the objective for these architectures is shown in Fig. 5.12a. The optimization results are in line with the theoretical values, which is also clear from the RMSE (Root-Mean-Square Error) columns (see Eq. (5.1)) in the table. As derivative values are required for the optimizer (the SQP algorithm is used in all the test cases), the design-competence-based systems now have less function calls, since the derivative values are provided through analytic expression.



a) monolithic architectures b) collaborative optimization

Figure 5.12: Objective value history (denormalized) for the Sellar problem (with design competences) for different optimization architectures

Table 5.6: Profiling data of the Sellar verification workflows with OpenLEGO. Data with separated with a /-symbol indicates differences between workflows using mathematical relations / design competences.

architecture	iters.	execution time (s)	function calls			RMSE		
			D[1]	D[2]	F	des. vars	couplings	objective
Test run	-	0.12 / 0.11	1	1	1	-	-	-
MDA (GS)	6*	0.10 / 0.16	6	6	1	-	4.20E-08	-
MDA (J)	19*	0.12 / 0.36	19	19	1	-	1.67E-06	-
DOE (GS)	-	0.49 / 0.98	26	26	4	-	-	-
DOE (J)	-	0.15 / 0.90	84	84	4	-	-	-
MDF (GS)	7	0.32 / 0.69	57 / 32	52 / 32	27 / 7	2.76E-09	6.28E-09	4.76E-09
MDF (J)	17	0.52 / 1.56	130 / 105	125 / 105	37 / 17	2.67E-07	6.01E-07	8.72E-07
IDF	7 /	0.34 /	32 /	27 /	27 /	2.00E-09 /	8.84E-15 /	8.36E-09 /
	6	0.49	6	6	6	1.22E-07	1.50E-07	1.32E-08
CO (tol 1e-1)	43 /	17.58 /	6061 /	6872 /	153 /	2.73E-01 /	7.06E-02 /	6.03E-01 /
	35	37.93	883	1406	35	9.01E-03	5.34E-02	9.20E-02
CO (tol 1e-2)	71 /	29.86 /	9814 /	12423 /	241 /	1.32E-03 /	4.97E-03 /	8.72E-03 /
	46	53.65	1475	1840	46	5.51E-02	8.06E-03	1.10E-02
CO (tol 1e-3)	92 /	37.49 /	12163 /	15272 /	297 /	1.47E-03 /	2.78E-03 /	3.35E-03 /
	61	84.42	2370	2825	61	1.93E-03	1.34E-03	7.12E-04

* These are the number of iterations of the solver, other entries in this column indicate the number of iterations of the system optimizer

The profiling data for the same monolithic architectures is shown in Tab. 5.7 for the SSBJ case. This case is a mix of mathematical functions and design competences. Again, the workflow results are fully in line with the expected values, thereby verifying the workflow materialization by OpenLEGO. The history of the objective for the monolithic architectures is shown in Fig. 5.13a.

DISTRIBUTED ARCHITECTURES

The second extension of OpenLEGO can only be validated using distributed architectures. The CO and BLISS-2000 architectures are used for this purpose. The CO architecture is discussed in more detail for the Sellar problem. The BLISS-2000 architecture can only be implemented on the SSBJ case, since the Sellar system does not have the right characteristics to use this architecture: it lacks local design variables for each discipline.

Imposing the CO architecture on the Sellar system results in the solution strategy shown in Fig. 3.11. The objective value history for the system-level optimizer is shown in Fig. 5.12b. A known issue of the architecture is the slow convergence of the optimizer [14]. The tolerance setting of the optimizer can be relaxed to reduce the required iterations, however, this has a negative impact on the precision of the solution. This trade-off between precision and execution time is clearly visible in the last three rows of Tab. 5.6, showing the CO workflow results for three different tolerance settings. As the tolerance becomes stricter, the found optimum matches more closely with the theoretical value, but the wall clock time and number of function calls increases dramatically.

The XDSM for CO with the SSBJ test case is shown in Fig. A.2. A tolerance of $1 \cdot 10^{-2}$ was selected to run this strategy, as it seems to be a good trade-off between precision and convergence in the Sellar test case. The objective history is plotted in Fig. 5.13b. The verification of the CO strategy is focused on obtaining the correct results with materialized workflows, regardless of their performance. Hence, the implementation could be further improved, but this was considered out of the current scope. For example, the system-level optimizer could obtain the derivatives of each disciplinary optimization `SubdriverComponent` using the Lagrangian of the sub-level optimizer at the found optimum. Alternatively, variations on the CO architecture from literature could also be used to improve the strategy.

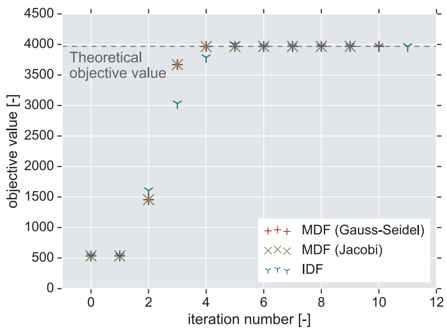
BLISS-2000 is the final and most complex distributed architecture that is validated in this chapter. The BLISS-2000 solution strategy for the SSBJ case is shown in Fig. A.3. The objective history for three workflow executions is shown in Fig. 5.13c. Three runs are shown, as the architecture contains an element of randomness. This randomness originates from the range of experiments performed by the DOE block to run disciplinary design optimizations. The DOE driver uses a latin hypercube sampling method that generates a randomly distributed collection of design points each time each time a surrogate model has to be constructed. In addition, the `B2kSolver` has many options on how the design space should be adjusted based on the found optimum by the system-level optimizer. For reference, Fig. 5.14 shows how the bounds of six design variables are adjusted each iteration for the first run plotted in Fig. 5.13c.

Of the three runs shown in Fig. 5.13c, only the first run reaches the same optimum as the monolithic architectures. In this run, the system-level optimizer is able to find a point close to the theoretical optimum in the second iteration. This shrinks the design space (shown in Fig. 5.14) in the correct direction, such that the global optimum is found at

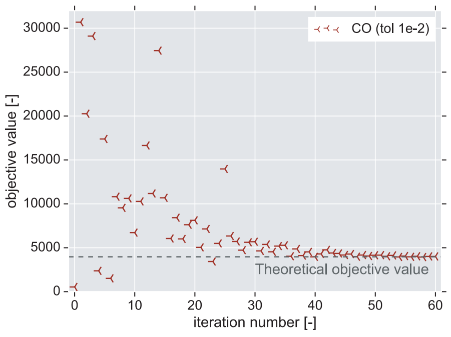
Table 5.7: Profiling data of the SSBJ verification workflows with OpenLEGO. (Struc. = Structural analysis, Aero. = Aerodynamic analysis, Prop. = Propulsion analysis, Perf = Performance analysis).

architecture	iters.	exec. time (s)	function calls				RMSE	
			Struc.	Aero.	Prop.	Perf.	des. vars	objective
Test run	-	0.29	1	1	1	1	-	-
MDA (GS)	6*	0.36	6	6	6	1	-	-
MDA (J)	21*	0.75	21	21	21	1	-	-
DOE (GS)	-	2.78	99	99	99	10	-	-
DOE (J)	-	9.24	370	370	370	10	-	-
MDF (GS)	11	4.14	78	78	78	11	1.08E-09	8.70E-09
MDF (J)	10	8.36	244	244	244	10	1.50E-08	8.85E-09
IDF	12	2.14	12	12	12	12	1.11E-09	8.74E-09
CO (tol 1e-2)	61	631.31	9362	9843	6035	466	2.82E+01	8.11E-02
BLISS-2000 (#1)	13	802.12	9926	8932	17380	4418	1.68E-01	4.08E+01
BLISS-2000 (#2)	22	1318.37	15077	17915	24004	1833	3.91E+01	3.38E+02
BLISS-2000 (#3)	22	1469.04	11735	18315	35969	8693	4.60E+00	4.40E+02

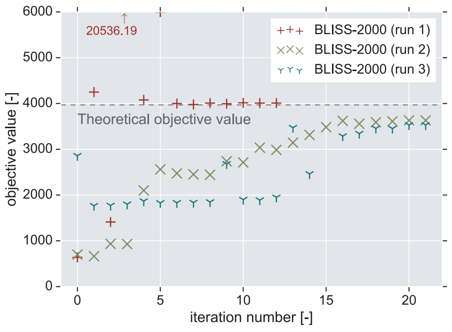
* These are the number of iterations of the solver, other entries in this column indicate the number of iterations of the system optimizer



a) monolithic architectures



b) collaborative optimization

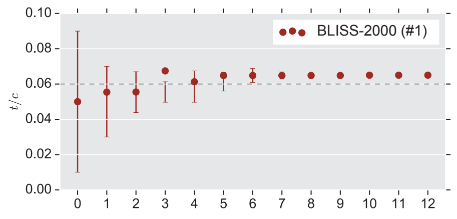


c) BLISS-2000

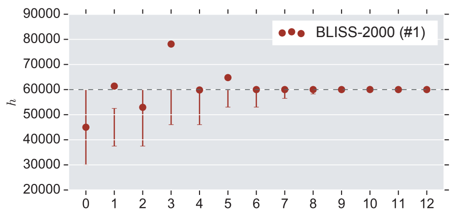
Figure 5.13: Objective value history (denormalized) for the SSBJ case for different optimization architectures with design competences

the end. In the other two runs, the optimizer gets close to the global optimum, however, the design space for the design variables is already reduced so much by the fifth iteration, that it is difficult to reach the global optimum. Compared to the monolithic architectures, the BLISS-2000 runs have a significant RMSE for the design variables and objectives, see the last two columns of Tab. 5.7.

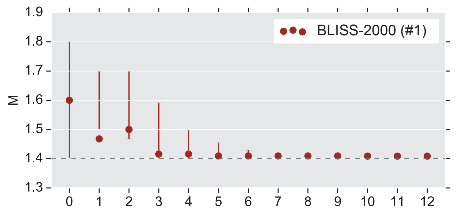
As for the CO architecture, further research into the implementation could be performed to improve its robustness. Based on the current results, such an improvement effort should initially be focused on the design space adjustment method in the `B2kSolver` and the building of better surrogate models for the system-level optimization cycle (at the moment, a response surface model is used for all disciplines). However, the aim in this section was not to present the most optimal distributed architectures, but to demonstrate and verify the materialization of a very complex solution strategy by OpenLEGO, based on a platform-agnostic strategy formulation (KADMOS) that is stored in a neutral data format (CMDOWS).



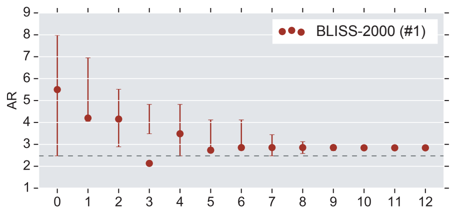
a) thickness-to-chord ratio



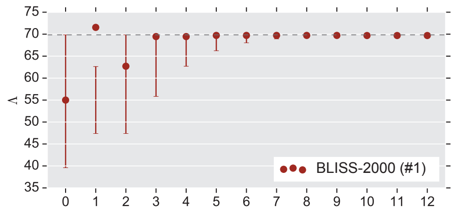
b) flight altitude



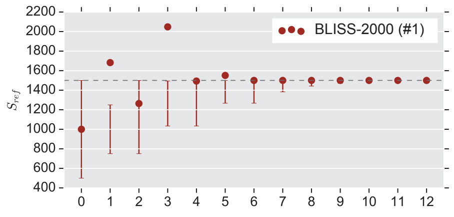
c) Mach number



d) aspect ratio



e) sweep angle



f) wing area

Figure 5.14: Design variable values and their bounds for each system optimization iteration for the BLISS-2000 (#1) architecture

5.4.3. Development process validation

This section has demonstrated that executable workflows for a wide range of MDAO solution strategies can be built in OpenLEGO. Workflow materialization was validated using two test cases. With three platforms able to parse CMDOWS files the bridging of the formulation-execution gap is firmly established. In addition, of the three PIDO platforms, OpenLEGO is the only platform that is able to materialize workflows for distributed architectures. This is a significant step regarding the validation of the preceding developments in the process, as it shows that the platform-agnostic, graph-based formulation of KADMOS can be successfully implemented as an executable workflow using the CMDOWS format, even for the complex scheme employed by a distributed architecture like BLISS-2000.

5.5. Platform comparison

The automatic creation of executable workflows based on a single XML file has been described and validated for three different PIDO platforms in this section. Looking at the profiling data in the different tables (5.1-5.7), the workflows in OpenMDAO clearly outperform the other two platforms. This can partly be attributed to the use of analytical derivatives for the design competences, but another strong point of OpenMDAO is its efficient implementation of all model components through the `DisciplineComponent` construct, including a very efficient class to pass data between XML files with the `XMLComponent` class. In RCE and Optimus on the other hand, tools have to be integrated in the platform separately, and XML data is passed around using additional components. However, this separated tool integration can also be considered an advantage of these platforms, as it enables the integration of a heterogeneous tool repository (where each tool is executed within their own programming environment or a tool can be a subworkflow integrated in a different PIDO platform) using dedicated tool integration modules, whereas the OpenMDAO workflow requires each tool to be defined through the Python-based `AbstractDiscipline` construct.

Despite its performance handicap, the workflows in RCE and Optimus do come with two other advantages. First, the workflows are presented in the GUI of the platform and can still be adjusted or fine-tuned easily based on the user's preferences. This is more difficult with a materialized OpenMDAO workflow, as OpenLEGO provides the Python object to be executed. This object can be visualized with different inspection methods from OpenMDAO (e.g. to create an N^2 chart of the model), but can only be fine-tuned through commands in a script. Hence, the RCE and Optimus workflows are more user-friendly after materialization. Second, RCE and Optimus support the collaborative execution protocol Brics from the AGILE project, which enables the execution of remote components. Getting this capability in OpenMDAO would be a matter of developing a `BricsComponent` class in OpenLEGO. This component does not exist, as the OpenLEGO package was developed at a later stage and was never planned to be used by the design teams in the AGILE project that require Brics to execute remote tools.

The end of this chapter marks the end of the second part of this dissertation, in which the main methodological and software-related developments have been discussed, verified, and validated. The verification and validation has shown that the formulation phase is successfully supported by the KADMOS-CMDOWS combination and enables the bridging of the formulation-execution gap in three different PIDO platforms. This last functionality alone is already sufficient to demonstrate the key role KADMOS and CMDOWS play in formulating, manipulating, storing and exchanging MDAO systems to reduce the set-up time of such systems. The next part will go a step further and discuss the application of the developments in two distinct MDAO frameworks to assess their impact on the development process in realistic, collaborative MDAO cases.

III

Framework integrations

6 INTEGRATION OF DEVELOPED METHODOLOGY IN THIRD-GENERATION MDAO FRAMEWORK

THE developments discussed so far in this dissertation were carried out in the context of the AGILE project. In AGILE, a consortium of nineteen partners has worked together to develop a third-generation MDAO framework, as was discussed in §§2.7.2. The project developments of over three years have led to the establishment of the AGILE paradigm: a model to perform MDAO collaboratively accompanied by a new framework that merges a variety of solutions (also referred to as 'technologies') to support design automation. KADMOS (Chapter 3), CMDOWS (Chapter 4), and the CMDOWS parsers for different PIDO platforms (Chapter 5) are three such solutions. With a variety of MDAO support technologies available, a key challenge is to bring them all together in a single framework to provide a comprehensive environment for a heterogeneous design team to perform MDAO collaboratively. This chapter aims to show how (a subset of) the AGILE technologies have been coupled together to form a coherent unity — called the *knowledge architecture* — and to assess this integration in multiple collaborative MDAO campaigns with a focus on the developments presented in the previous part of this dissertation.

The knowledge architecture will be discussed in §6.1: first as a generic architecture for MDAO, and then followed by the mapping and implementation of the AGILE technologies based on this generic architecture. The use of the AGILE framework in a collaborative design task is presented through a demonstrator in §6.2, in which the wing-engine integration of a conventional airliner is designed using the framework. Four other collaborative designs, concerning the MDAO of unconventional aircraft configurations, are

The contents of this chapter have been adapted from [107] and [108]. As these publications cover multiple AGILE technologies from different partners, the discussion in this chapter has been limited to the developments presented in Part II of this dissertation.

summarized in §6.3. These design cases are used to assess the framework and its technologies in §6.4 based on a questionnaire that was completed at the end of the MDAO campaigns on the four novel aircraft concepts.

6.1. Knowledge architecture

A third-generation MDAO framework was developed in the AGILE project (introduced in §§2.7.2), of which this chapter presents key contributions. AGILE's goal was to extend the current set of applications, strategies and data schemas to enable distributing all tasks involved in the formulation and operation of an MDAO system, thereby offering a truly collaborative environment for disciplinary experts, system architects and end users. The two cornerstones of the AGILE paradigm are the Knowledge Architecture (KA) and the collaborative architecture, as depicted in Fig. 2.17. The KA is the main focus of this chapter, as it contains elements that were presented in the previous part of this dissertation. In this section, a description of the KA as a domain-agnostic methodology to architect collaborative and distributed MDAO systems is discussed first. Then, its specific implementation as a framework to support the AGILE aircraft design teams is presented. The coverage of the MDAO development process by the KA (and hence the topics presented in this chapter) is shown in Fig. 6.1. The KA elements discussed in this chapter cover the process up to and including the materialization of the executable workflows, but do not consider their actual execution and results, as this is part of AGILE's other cornerstone (the collaborative architecture) that is out of scope of this dissertation's developments. Note that the MDAO solution strategy and executable workflow both fall under the generic term 'automated design process', which will be used repeatedly throughout this chapter.

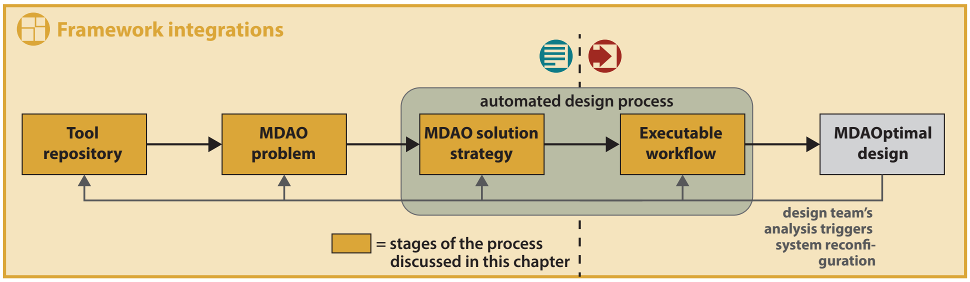


Figure 6.1: The five different stages an MDAO system can have within an MDAO framework and the stages of the AGILE framework covered in this chapter

6.1.1. Methodology

Fig. 6.2 provides a schematic of the knowledge architecture. The KA features a hierarchical four-layer structure. In the top layer *development process* all the tasks required to define, monitor and manage an MDAO system are compiled into one *business process*. Hence, this development process layer serves as 'the cockpit' of the KA from which lower layers are controlled and all other applications are used 'under the hood'. The intermediate layer *automated design* provides the means to formalize the computational architecture of the automated design system. The bottom layer *design competences* hosts the actual synthesis and analysis tools contributed by the various discipline experts involved in the collaborative design process. A fourth transverse layer *data schemas* provides the other three with various schemas to support data exchange between the different components at each level. Interfaces guarantee cohesion between the layers, as clarified in the forthcoming subsections.

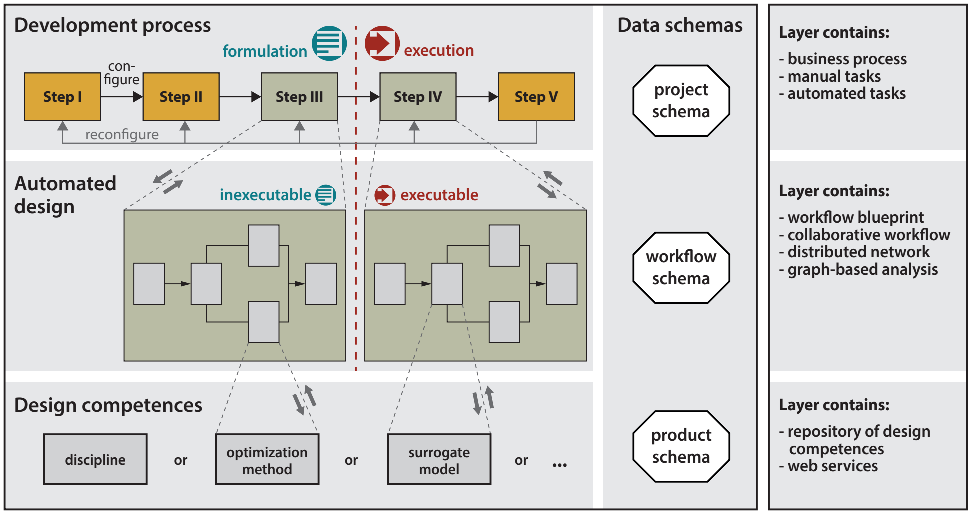


Figure 6.2: The AGILE Knowledge Architecture (KA) to support automated design in large, heterogeneous teams of experts

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

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

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

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

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

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

The specific roles of the various agents in the KA layers are discussed in more detail in the forthcoming sections and examples are provided with the demonstrator in §6.2. The four layers of the KA with their relative interfaces will be discussed in the next subsections.

Development process layer

The development process layer can be seen as the cockpit or decision room where the setup and execution of a new multidisciplinary design problem take place. A business-type process is defined here to allow all the aforementioned agents to collaborate and interact during the two main phases of the MDAO system development: formulation and execution. The business process is named the AGILE development process and is organized in five main steps, which are illustrated in Fig. 6.3 and listed below:

step I: **Define design case and requirements.** Information and requirements are collected concerning the product to design/analyze/optimize, the MDAO system to be developed and the available design competences (tools and experts) from the design competences layer.

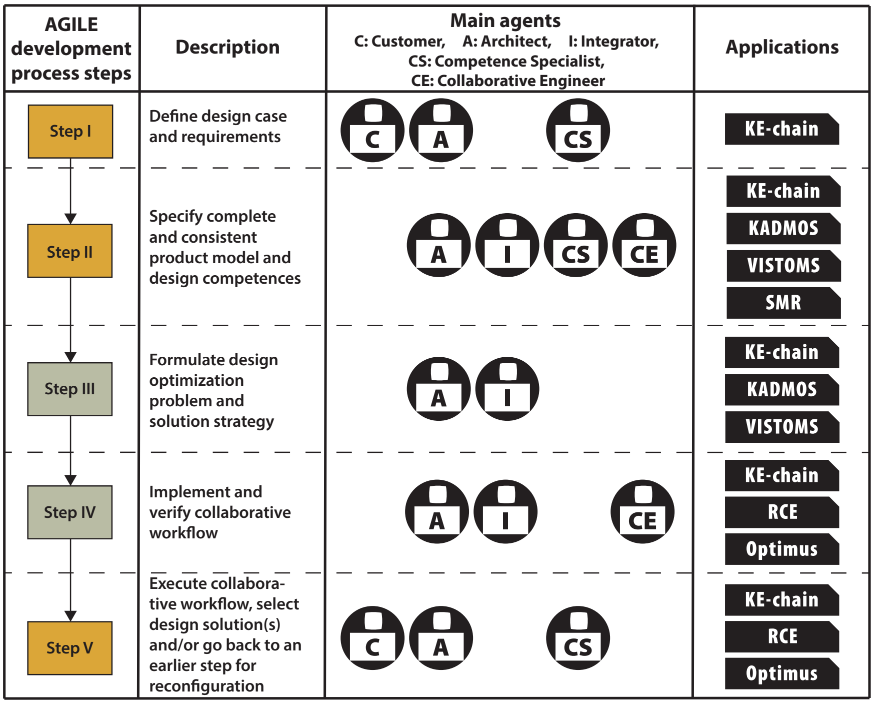
step II: **Specify complete and consistent product model and design competences.** The consistency of the collected information is verified and validated. The design competences are linked to the common product model from the data schemas layer and the connections are verified.

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

step IV: **Implement and verify collaborative workflow.** The common workflow schema from the data schemas layer is used to enable the translation of the formulated (in-executable) automated design process into an executable workflow for the PIDO platform of choice, see Chapter 5.

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

Note that the five-step AGILE development process is not the same as the MDAO development process that was introduced in Fig. 1.3. The AGILE process represents a broader collaborative business process that contains the MDAO system development as a sub-process to (re)configure the MDAO system. The relation between these two processes will be clarified further later in this chapter, as is shown in Fig. 6.6. Throughout this



AGILE development process steps	Description	Main agents C: Customer, A: Architect, I: Integrator, CS: Competence Specialist, CE: Collaborative Engineer	Applications
Step I	Define design case and requirements	C A CS	KE-chain
Step II	Specify complete and consistent product model and design competences	A I CS CE	KE-chain, KADMOS, VISTOMS, SMR
Step III	Formulate design optimization problem and solution strategy	A I	KE-chain, KADMOS, VISTOMS
Step IV	Implement and verify collaborative workflow	A I CE	KE-chain, RCE, Optimus
Step V	Execute collaborative workflow, select design solution(s) and/or go back to an earlier step for reconfiguration	C A CS	KE-chain, RCE, Optimus

Figure 6.3: Five-step AGILE development process, the involved agents in each step, and the applications supporting that step in the AGILE development framework implementation

chapter, the blocks of the MDAO development process are consistently referred to as 'stages', while the blocks of the business process are called 'steps'.

Automated design layer

As illustrated in Fig. 6.2, two instances of the automated design process reside in this layer: the inexecutable formulation of the automated design process defined in step III of the development process and the executable collaborative workflow that is materialized in step IV. This is in line with the formulation-execution distinction used throughout this dissertation. The non-executable formulation (Fig. 6.2, left) represents the blueprint of the automated design process and can be generated by KADMOS (Chapter 3). In the MDAO community, a widely used visualization standard for this formulation is the XDSM, which was introduced in §2.2, see also Fig. 2.5c. Based on the automated design process blueprint, the executable counterpart (Fig. 6.2, right) is automatically generated using CMDOWS (Chapter 4) and a PIDO platform, such as Optimus, RCE, or OpenMDAO (Chapter 5). This automated link between formulation and execution is a key feature of the KA and enabled by the work presented in this dissertation. Without this link the formulation performed in steps I-III would be disconnected from the execution in step V, meaning that any reconfiguration of the automated design process would require manual adjustments of the executable workflows.

Design competences layer

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

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

At the same time, competence specialist are granted the following rights:

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

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

Data schemas layer

Data schemas are used in all layers of the KA, see Fig. 6.2. The project schema used in the development process layer is outside the scope. The other two schemas have already been discussed in detail in previous chapters. The use of such schemas is based on the 'common model integration' approach which enables the efficient integration of different components with a minimal number of interface links, as illustrated for the product schema in Fig. 2.1 and for the workflow schema in Fig. 4.2. The product schema is used by design competences to read and write I/O files. The fixed CDS approach discussed in §§2.1.2 is adopted in the KA, where CPACS is the schema of choice in the aircraft design domain. The workflow schema is used in the automated design layer to store the definition of the collaborative system, such that all applications can use and produce information regarding that system. A schema to store MDAO systems (CMDOWS) has been developed in this research and was presented in Chapter 4.

While the use of data schemas at the different layers of the KA matches well with the heterogeneity of teams, products, and workflows, it can also be seen (and experienced) as a constraint that is put on the other KA layers. This is the *paradox of standardization*: while the data schemas help the integrator tremendously in his task, individual competence specialists might feel they are loosing some freedom in defining their own competence, since any data that needs to be exchanged between members of the heterogeneous teams will have to meet schema definitions. However, schema compliance was found to be crucial for enabling the definition and execution of collaborative workflows in large teams and therefore in the AGILE paradigm its benefits are assumed to outweigh the burden. To lighten the burden of schema compliance, every data schema that is used is accompanied by an ecosystem of tools, such as a schema operations library, as was discussed in §§2.1.2 for CPACS.

LAYER INTERFACES

The interfaces between the three horizontal layers in Fig. 6.2 are discussed here to clarify their relation.

Development process / automated design interface The interface between the development process and automated design layer is indicated in Fig. 6.2 using dashed lines. This interface needs to be bidirectional. In downward direction (development process → automated design) the development process tasks can control the automated design process by changing settings (e.g. a change in design requirements or a tool replacement). In upward direction (automated design → development process) the specification of the automated design process needs to be brought to the development process layer, to give the agents operating in that layer the opportunity to inspect, validate and discuss the automatically generated MDAO solution strategies.

Automated design / design competences interface The bidirectional interface between the automated design and design competences layers is also shown in Fig. 6.2 using dashed lines. In downward direction (automated design → design competences) the automated design process should be able to call the different design competences. To respect the competence domain of the discipline specialist, the automated design process should only be allowed to ask for the execution of a design competence, while leaving the actual execution and feedback up to the competence specialist. This interface will prevent the automated design process to demand the execution of a design competence in ways that are outside the 'comfort zone' of the design competence team.

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

Within the AGILE paradigm it is of key importance to make this interface robust in order to 'get things right', since the automated design process that is created will be large and complex, and it is difficult to find any mistakes or inconsistencies that it might contain. To handle the size and complexity of the automated design process a system needs to be available that can represent the repository of coupled design competences, such that this representation can be used to inspect the repository, and can also be manipulated to formulate the MDAO problem and solution strategy stages of the system. The framework applications developed in AGILE to support these tasks will be discussed in §§6.1.2.

6.1.2. IMPLEMENTATION: AGILE DEVELOPMENT FRAMEWORK

The Knowledge Architecture described in the previous section represents one of the fundamental concepts of the AGILE paradigm. This paradigm provides the abstract formalization of a generic methodology to develop MDAO systems, independently of the application field. Within the AGILE project, such methodology has been specifically implemented to develop a dedicated MDAO framework for aircraft design, the so-called *AGILE Development Framework (ADF)*, which is the topic of this section. The ADF, whose architecture is visualized in Fig. 6.4, is built on top of the following technology enablers and data schemas, all developed or extended during the course of the AGILE project (the institute/company that developed the item is indicated between parentheses):

CPACS product schema [S7] (DLR): open-source, de-facto standard data schema for conceptual and preliminary aircraft design, covered in §§2.1.2.

CMDOWS workflow schema [S24] (DUT): a proposed new, open-source standard schema to store and exchange MDAO systems, discussed in Chapter 4.

KE-chain [S26] (KE-works): a commercial, web-based platform providing the development process environment. KE-chain will be briefly introduced in this section, as it is tightly coupled in the ADF with this disseration's developments.

KADMOS [S19] (DUT): an open-source, graph-based package used to enable the configuration, formalization and manipulation of MDAO systems, described in Chapter 3.

VISTOMS [S11] (RWTH Aachen University): a web-based package to enable the visualization and inspection of large MDAO systems, discussed by Aigner et al. [39] and also briefly introduced in this section.

Surrogate Model Repository (SMR) [S27] (NLR): a web-based interface and database to store and execute surrogate models.

RCE [S3] (DLR): a PIDO platform (introduced in §§2.5.1) for which a CMDOWS-parsing capability has been developed, as discussed in §5.2.

Optimus [S2] (Noesis Solutions): a second PIDO platform (introduced in §§2.5.2) for which a CMDOWS-parsing capability has been developed, as discussed in §5.3.

The full description and discussion of the ADF is presented in [107]. Fig. 6.4 provides an overview of the ADF by mapping the AGILE technologies on the generic KA depicted in Fig. 6.2. In this chapter, the scope of the framework's description and assessment is limited to this dissertation's developments (KADMOS, CMDOWS, and CMDOWS parsers) and their relation to the technologies listed above. Note that all those technologies, except for CPACS, are not domain specific, and thus endow the AGILE paradigm with the necessary neutrality to allow its application to different engineering areas.

BRIEF INTRODUCTION TO KE-CHAIN

KE-chain [S26] is a web-based collaborative environment, developed by KE-works to support the integration of collaborative and hybrid processes, thus combining business-type processes that require manual input and user interaction with fully automated engineering simulation workflows. The environment provides access to a single project for multiple end users through a user-based authentication system, offers functional-

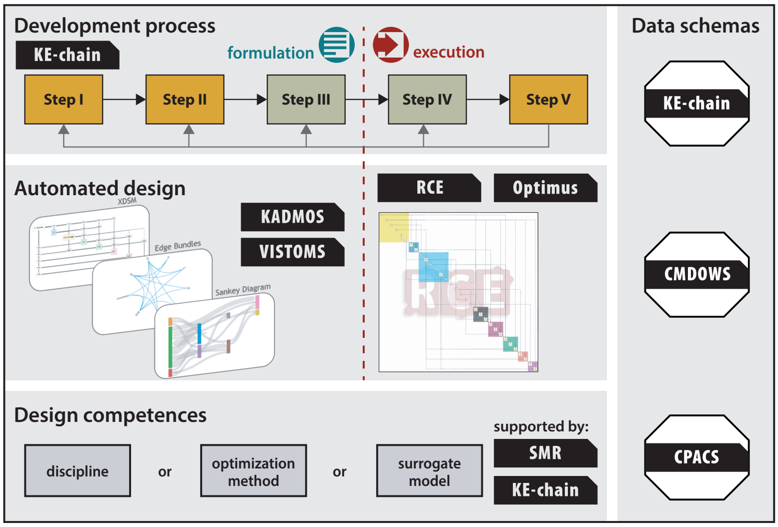


Figure 6.4: Overview of the AGILE Development Framework (ADF)

ity to facilitate the management of project data, the integration of automation solutions within the business process, and monitoring of progress. KE-chain is used to implement the KA's development process layer (Fig. 6.2). Through its web-based GUI (see screenshots in Fig. 6.8 and Fig. 6.10), KE-chain provides control over the setup of the MDAO system by acting as a user-friendly interface for KADMOS, based on the five-step approach shown in Fig. 6.3. The platform relies on CMDOWS to store and exchange the system's definition with the users and between framework applications. In other words, KE-chain provides the ADF cockpit, where the users interactively define all the aspects related to the MDAO system and project and constantly monitor both the development progress and the operation of the system.

Brief introduction to VISTOMS

The number of design competences and, more importantly, the typical amount of exchanged data within a collaborative MDAO system can be so large that is nearly impossible to comprehend all underlying, relevant information at a glance. As a consequence, keeping oversight and control (hence trust) on the overall system and the ability to inspect and debug it, is an extremely challenging job for the integrator. Rendering the relevant information in a human-intelligible way, by means of effective visualizations is of paramount importance. VISTOMS is a web-based visualization package that is able to translate a formulated system by KADMOS, as stored in a CMDOWS file, into dynamic, interactive and human-readable plots and diagrams, in a web-based interface.

The position of VISTOMS in steps II and III of the ADF is depicted in Fig. 6.6. In VISTOMS, visualization techniques such as XDSMs [37, 109], Sankey diagrams [110–112] and hierarchical edge bundles [113, 114] are utilized to provide interactive visualizations of the system. For example, large and complex XDSMs can be expanded or collapsed in order to focus on certain aspects of the system. Couplings between services or variable interdependencies can be analyzed in human-readable diagrams, in which it is possible to dynamically reveal or hide detailed information. This enables insight into information that can either not be visualized at all, or is too confusing to comprehend when visualized all at once. Thus, VISTOMS can be used as an effective debugging environment, in which architects and integrators are able to examine the created solution strategies (stored as CMDOWS instances), review the steps of the product development process and detect possible issues. Several examples of produced visualizations of KADMOS graphs can be found in the next section where the ADF is demonstrated. The package is more elaborately discussed by Aigner et al. [39].

CMDOWS IN THE AGILE DEVELOPMENT FRAMEWORK

CMDOWS has been discussed in detail in Chapter 4. In §§4.1.1 of that chapter, different framework application categories were introduced and described:

- tool repository
- MDAO formulation system
- visualization package
- business process management tool
- schema operations library

Fig. 6.5 provides an overview of how the different ADF application fall within those categories. At least one application is available for each of the six categories in the framework, with some applications fulfilling multiple roles. For example, KE-chain acts as the business process management application, but also supports the creation of a tool repository. KADMOS even falls within three categories: the package primarily acts as the MDAO system formulation application, but it can also provide basic visualizations restricted to PDF files and graphs (as opposed to the advanced, dynamic visualizations provided by VISTOMS), and the package acts as the schema operations library for CMDOWS files by providing a collection of schema-specific methods to easily read and write schema-based files.

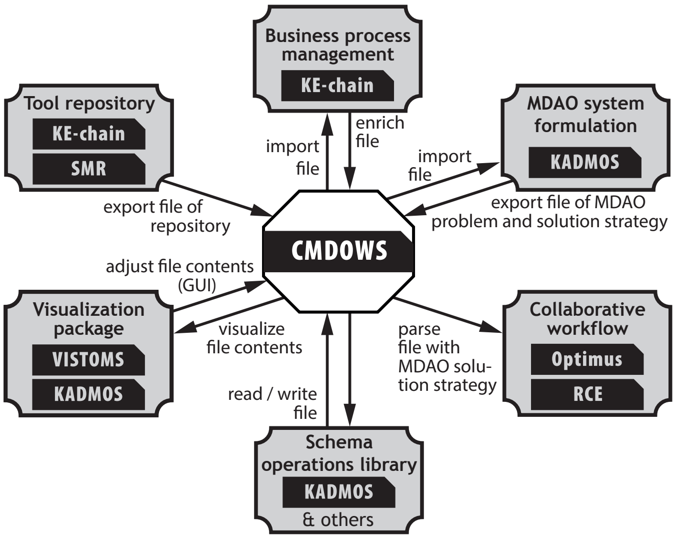


Figure 6.5: The established links between CMDOWS and the AGILE MDAO framework applications

In the ADF, CMDOWS serves as a dynamic storage schema where CMDOWS instances contain the evolving definition of the MDAO system during its development process, as was also illustrated for a small illustrative example in §4.4. In particular, CMDOWS instances are generated from step II to step IV, as illustrated in Fig. 6.6. At each (sub-) step in that figure, the CMDOWS file can be accessed by VISTOMS (§§6.1.2) to generate convenient visualizations of the MDAO system throughout its progressive development. This will offer all ADF stakeholders the possibility to inspect and maintain oversight of systems of any complexity.

KADMOS IN THE AGILE DEVELOPMENT FRAMEWORK

KADMOS was covered in Chapter 3. In the ADF, KADMOS supports the team with three main tasks:

- **Create:** Enabling the specification of large, complex MDAO systems by means of graphs, starting from the definition of the design competence repository, to the formulation of the problem, and concluding with the complete solution strategy, corresponding to the first three stages in Fig. 6.1, respectively.
- **Inspect & debug:** Enabling different experts (i.e. competence specialists, integrators) to inspect sections of the system that are relevant to them, in order to validate it, thereby increasing the level of trust in the system. In addition, KADMOS provides automated validation functions to assist the team in validating the specified system based on strict conditions on the graph construct (e.g. all design competences should have at least one output, all nodes should be connected, etc.).
- **Manipulate:** Automatically manipulating the generated MDAO system specification using graph-based analysis and dedicated algorithms. These computerized manipulations, apart from providing drastic time reductions, also reduce the chance of errors and inconsistencies in the system by eliminating repetitive error-prone human tasks.

The use of KADMOS within the ADF is depicted in Fig. 6.6. The relation between the steps of the AGILE development process and the stages of the MDAO system used within these steps is shown in the top and bottom of the overview. The MDAO development process commences at step II of the AGILE process, where KE-chain and the SMR are used to define the repository of design competences and pass that information to KADMOS through a CMDOWS file. A custom-built KE-chain interface is used to transform the MDAO system from tool repository to MDAO solution strategy. At each transformation step, intermediate stages of the system are stored as CMDOWS file and inspected with VISTOMS.

Hence, KADMOS is not used through Python scripts in the ADF, but is executed through the user interface provided by KE-chain. Thereby, in Fig. 6.2, KADMOS provides the upward design competences/automated design interface for formal workflow specification. In other words, whereas KE-chain provides the ADF cockpit, KADMOS is the engine running under the hood regarding MDAO system formulation. The graphs created and manipulated by KADMOS use or affect all layers of the KA. The creation and manipulation of the graphs by KADMOS are a matter of seconds for the majority of cases, with a maximum evaluation time of up to one minute for very large systems in combination with the most complex manipulations the package can perform. Where there would normally be a gap between the specification of the automated design process to be executed and the actual executable collaborative workflow, KADMOS and CMDOWS enable

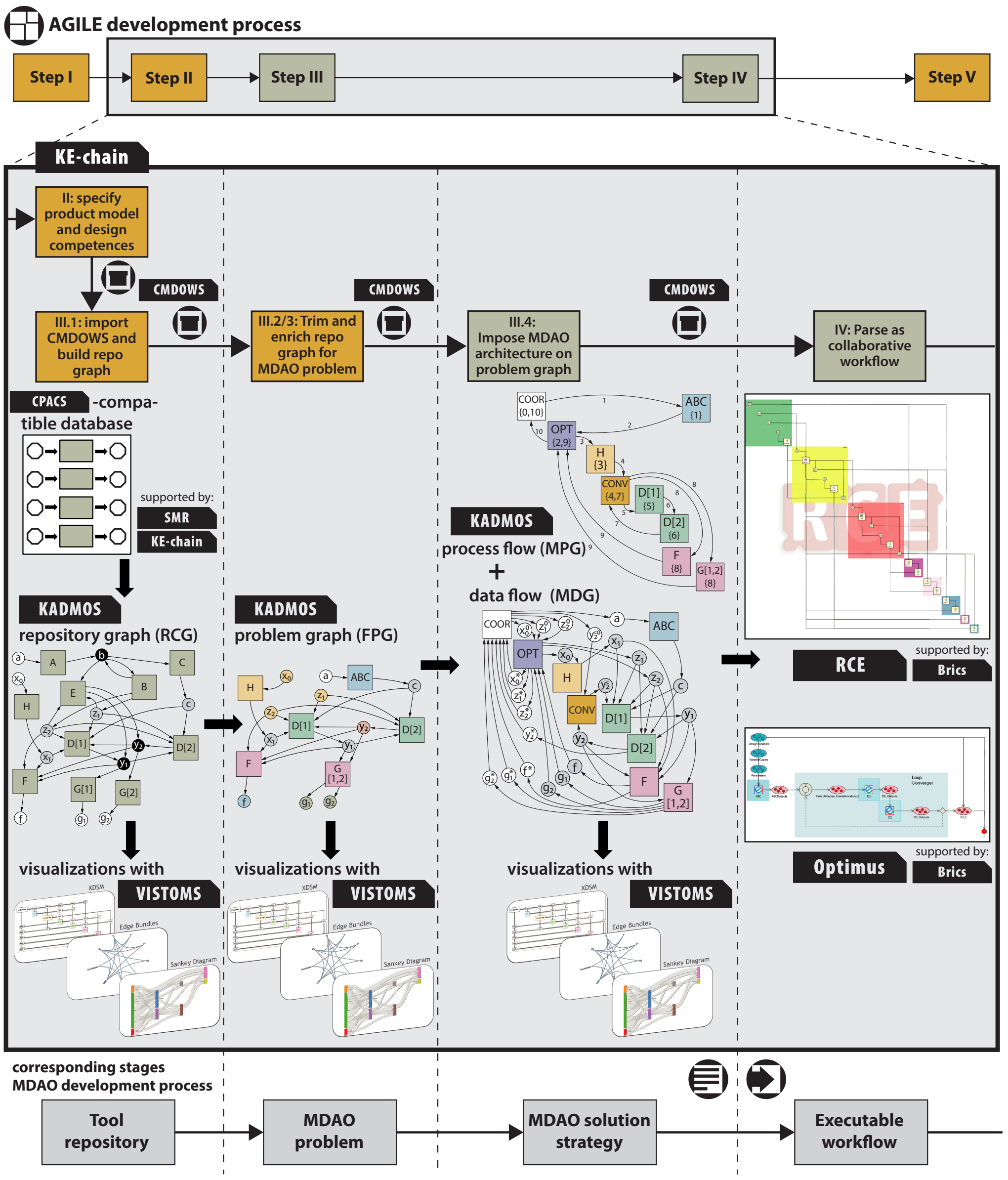


Figure 6.6: Top-level overview of KADMOS and CMDOWS, and their relation to other framework applications and the five-step AGILE development process (all visualizations are based on the Sellar problem [86])

a design team to go, in an agile way, from a repository of design competences to a fully configured workflow in Optimus or RCE, as depicted in the last column of Fig. 6.6.

6.2. Framework demonstrator: airliner

The demonstrator of the ADF is based on the work performed in the second design campaign of the AGILE project, see Fig. 2.18. This campaign targeted the conceptual and preliminary design of a medium-range twin-engine jet airliner, which is depicted in Fig. 6.7. A heterogeneous collection of CPACS-compliant design competences has been used to set up a distributed automated design process. This section focuses on the first three steps of the AGILE development process (Fig. 6.3) to demonstrate how this dissertation's developments come into play in the ADF for MDAO system formulation in a realistic design case. The last two steps concern execution of that system and will only address workflow materialization, remaining execution details can be found in [115]. For each step, the following information will be provided: description of the main function, associated deliverable, deployed framework applications, and main agents involved.

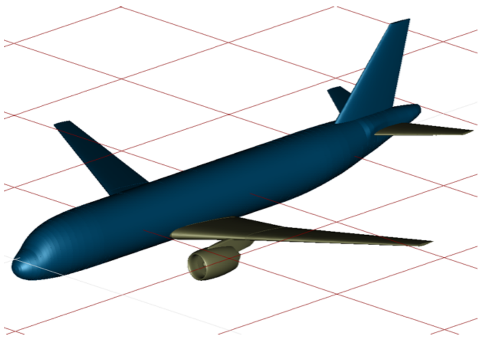


Figure 6.7: Impression of the medium-range twin-engine jet airliner under consideration for the demonstrator

Table 6.1: Top-level aircraft requirements

requirement	value	unit
entry into service	2020	-
range	3500	km
design payload	9180	kg
max. payload	11500	kg
PAX (@ 102kg)	90	-
MLM (%MTOM)	90%	-
long-range cruise Mach	0.78	-
initial climb altitude	11000	m
max. operating altitude	12500	m
residual climb rate	91	m/min
TOFL (ISA, SL, MTOM)	1500	m
Vref (ISA, SL, MLM)	<130	kts
dive Mach number	0.89	-
fuel reserves	5%	-

6.2.1. Step I: Define design case and requirements

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

Agents: customer, architect, competence specialists

KA applications: KE-chain

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

Table 6.2: Design competences used in the demonstrator. All competences have intellectual property restrictions and can therefore not be made available on one server environment that is shared with other partners. (DE=Germany, IT=Italy, RU=Russian Federation)

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

6.2.2. Step II: Specify product model and design competences

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

Agents: architect, integrator, competence specialists, collaborative engineer

KA applications: KE-chain, SMR, VISTOMS, KADMOS

Deliverable: A CMDOWS file containing a complete and consistent repository of design competences which are compliant with CPACS.

Substeps: Four substeps have been identified, namely:

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

step II.2 Import CPACS-ized competences
The competence specialists should develop (or check existing) CPACS-compliant design competences using the base file and provide the integrator with one CPACS input and one CPACS output file for each design competence. In this task, they are supported by the collaborative engineer. Alternatively, CPACS-sized competences can also be imported from the SMR. Based on these files, a CMDOWS file is created to enable other framework applications to access the definition of the tool repository.

step II.3 Inspect repository
The architect can now, through a VISTOMS instance, inspect the design competence repository in consultation with the competence specialists. This step is used to fix inconsistencies in the base file (step II.1) or solve connectivity issues between competences based on the I/O files (step II.2).

step II.4 Append additional information
Finally, the competence specialist specifies the competence execution method and availability (i.e. as a local tool that can be installed and run locally or a remote service that needs to be called through a service-oriented architecture). This final step is meant to prepare all information required for execution and inspection of the MDAO solution strategy in steps IV and V and it is added to the CMDOWS file once it becomes relevant in step IV.

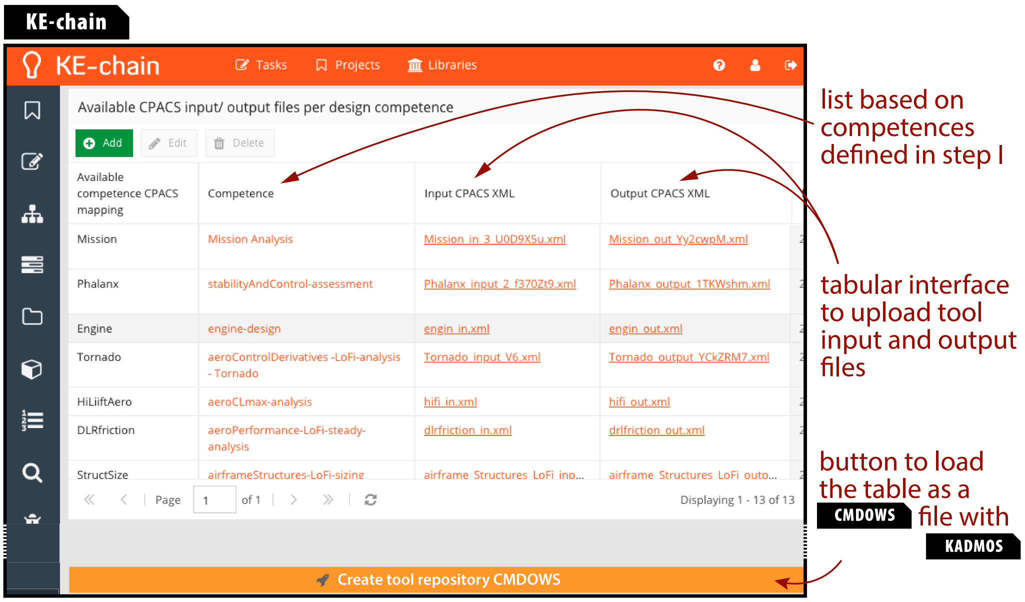
USE OF KADMOS AND CMDOWS BY KE-CHAIN IN STEP II

In step II.2, the integrator is supported by an engineering service that parses the uploaded CPACS files using a KADMOS import function to construct an initial MDAO system stored as a CMDOWS file. The KE-chain interface used for this is depicted in Fig. 6.8. Hence, KE-chain acts as a user-friendly front end to define the RCG, something which otherwise should be performed through a script. KE-chain also provides an interface to the integrator in which versions of the generated CMDOWS files can be managed, and the connectivity of the different design competences can be inspected in more detail in step II.3 through the integration of VISTOMS, described in more detail in the next paragraph.

USE OF KADMOS GRAPH-BASED FORMULATION BY VISTOMS IN STEP II

In this early stage of the problem formulation the MDAO system is represented by the RCG, i.e. the graph containing the design competences, product model elements, and the links between them. This graph contains thousands of nodes and is challenging to visualize as such. The VISTOMS package takes advantage of the structured, graph-based representation of the MDAO system to construct its dynamic visualization. A part of the repository graph for the presented design case is shown in Fig. 6.9 using the XDSM visualization mode of VISTOMS.

Note that the actual RCG for the presented example is significantly larger than the extract shown in the figure. The red overlay boxes (not present in the actual VISTOMS package) have been added in the figure to emphasize some of its visualization capabilities. One of them is the option to detect element collisions, i.e. elements which are written by multiple design competences simultaneously. Overlay frames 1 and 2 in Fig. 6.9 display for instance a so-called collided circular coupling of the main wing. This occurs due to the fact that two competences (AircraftSynthesis and Morphing) both modify the wing


KE-chain
KE-chain
Tasks
Projects
Libraries
Add
Edit
Delete
list based on competences defined in step I
tabular interface to upload tool input and output files
button to load the table as a CMDOWS file with KADMOS
Create tool repository CMDOWS

Available CPACS input/ output files per design competence

Available competence CPACS mapping	Competence	Input CPACS XML	Output CPACS XML
Mission	Mission Analysis	Mission_in_3_U0D9X5u.xml	Mission_out_Yy2cwpM.xml
Phalanx	stabilityAndControl-assessment	Phalanx_input_2_f370Zt9.xml	Phalanx_output_1TKWshm.xml
Engine	engine-design	engin_in.xml	engin_out.xml
Tornado	aeroControlDerivatives -LoFi-analysis - Tornado	Tornado_input_V6.xml	Tornado_output_YCkZRM7.xml
HiLiftAero	aeroCLmax-analysis	hifi_in.xml	hifi_out.xml
DLRfriction	aeroPerformance-LoFi-steady-analysis	dlrfriction_in.xml	drlfriction_out.xml
StructSize	airframeStructures-LoFi-sizing	airframe_Structures_LoFi_inp...	airframe_Structures_LoFi_outp...

Page 1 of 1 Displaying 1 - 13 of 13

Figure 6.8: Screenshot of the KE-chain web-based interface for step II.2

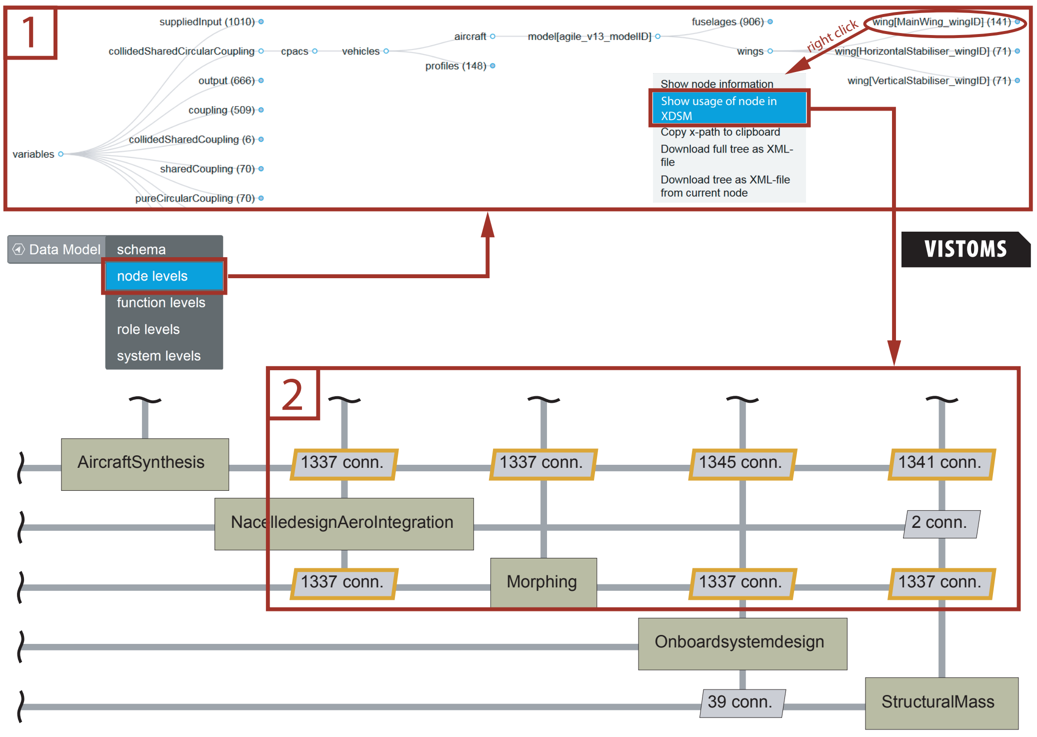


Figure 6.9: Repository connectivity graph of the presented design case as VISTOMS XDSM view including data tree

geometry. These kinds of collisions have to be resolved (or at least noticed) by the integrator in order to keep the underlying product model (CPACS file) consistent, which can be done with KADMOS in the subsequent step of the development process.

The dynamic visualization capabilities combining different visualization elements such as XDSMs for the general layout, hierarchical tree views for the data model (including different parameter categorizations), and the capability to highlight certain points of attention in the MDAO problem, together enable an efficient and effective visualization of all the information stored in an MDAO system that would normally be implicit and almost impossible to access. These visualizations can be created thanks to the graph-based formulation of these systems offered by KADMOS and visualized in a meaningful way to the different members of the design team based on the description of MDAO systems worked out in Chapter 3.

6.2.3. Step III: Formulate design problem and solution strategy

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

Agents: architect, integrator

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

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

Substeps (see also Fig. 6.6): All substeps are performed by the architect unless indicated otherwise:

step III.1 Construct RCG and order tools
First the RCG is constructed in KADMOS using the CMDOWS file from step II. This is simply a translation of the XML-based CMDOWS format to the KADMOS native graph format. In addition, the order of the functions on the diagonal can be determined automatically (using the sequencing method described in §§3.5.3) or set manually.

step III.2 Start FPG by manipulating design competences
The RCG needs to be manipulated in order to arrive at the problem graph. In this first substep, the KE-chain interface focuses on the functions in the graph to allow the user to remove and/or merge them as required.

step III.3 Finish FPG by assigning variable roles
The problem graph specification is finalized by focusing on the variables and assigning certain CPACS elements a special role in the MDAO problem, such as design variables, objective values and constraint values.

step III.4 Impose MDAO architecture
In this demonstrator a DOE architecture is chosen[*], see Fig. 6.11 and Fig. 6.12. The formal mathematical definition of the problem being solved in this demonstrator

[*]N.B. The results of the DOE will provide the design team with an indication of the most influential variables, based on which a surrogate model could be constructed to perform an optimization, however, these additional reconfigurations are discussed more elaborately in other AGILE design case papers and considered out of scope for the work presented here.

is:

$$
\begin{aligned}
\text{determine:}\quad & \text{DOC, recurring cost, ATR and fuel mass}\\
\text{with respect to:}\quad & \text{AR}, \Lambda, S\\
\text{with:}\quad & 9.00 \le \text{AR} \le 11.0\\
& 25.0^\circ \le \Lambda \le 31.0^\circ\\
& 75m \le S \le 110m\\
\text{based on:}\quad & \text{a latin hypercube sampling method}
\end{aligned}
$$

Using the problem graph, KADMOS imposes the DOE architecture and stores the obtained MDG and MPG in a single CMDOWS file. The solution strategy is checked by the integrator.

USE OF KADMOS IN STEP III OF THE DEMONSTRATOR

The CMDOWS file from step II contains eleven design competences (Tab. 6.2) involving around 3700 unique elements from CPACS as in- and outputs. The RCG that is constructed by KADMOS in step III.1 is primarily useful to build visualizations with VISTOMS and to order the design competences in a convenient way, as was discussed in §§3.5.3 on sequencing algorithms. Furthermore, KADMOS categorizes all the CPACS elements (e.g. inputs, outputs, couplings, collisions, etc.) based on the number of connections and the design team can use these categorizations for closer inspection in the VISTOMS visualization of the design competence repository, as was mentioned already in §§6.2.2. For example, in the demonstrator the CPACS data element pointing to the Maximum Take-Off Mass (MTOM) is a collision, as it is both an output of the AircraftSynthesis and the MassBudget design competences. It is then up to the design team to decide which design competence is meant to provide this value. In this case the MassBudget tool was set to provide the MTOM value; however, the less accurate value produced by AircraftSynthesis could be used as initial guess for MTOM, if MassBudget is part of a convergence loop with MTOM as feedback variable.

In step III.2 remaining issues on the data connections are solved. One such issue is visible in Fig. 6.9, where the wing geometry (1337 connections, hence 1337 CPACS elements are used to define the aircraft geometry and coupled as indicated in the figure) is output of both AircraftSynthesis and the Morphing design competences, and many design competences, including the Morphing tool itself, are using the same geometry as input. This is not a case of redundant design competences playing the same role of geometry provider. In this case, a first instance of the wing geometry should be given to the Morphing tool by the AircraftSynthesis tool. Then the Morphing tool should adjust this geometry and provide a second instance of the wing geometry, to be used by all the other tools that need geometry as input. KADMOS configures this data flow by creating instances. A first instance of the wing geometry is created by AircraftSynthesis and provided as input to Morphing. In its turn, Morphing creates a second instance of the wing geometry to be used by the following design competences. These two geometry data instances are also visible in Fig. 6.11 and 6.12 where the wing definition is included in the data connection blocks with 1337 connections.

In step III.3 KADMOS assigns the special roles to selected elements, according to the architecture selected by the architect. The architect is able to assign element roles through the interface in KE-chain as is shown in Fig. 6.10. In case of the demonstrator DOE, wing area, aspect ratio, and sweep have been assigned the role of design variables. These are top-level wing variables that can be set by the Morphing tool (see Fig. 6.11). All the other variables the design team wants to keep track of in the overall process, are assigned the role of 'quantities of interest'. These quantities could later become the objective and constraints of an optimization process. The quantities of interest selected for the demonstrator were: Direct Operating Cost (DOC) and recurring cost from CostAnalysis, Average Temperature Response (ATR) from EmissionAnalysis, and fuel mass from MissionSimulation.

Finally, the DOE architecture is imposed on the problem graph in step III.4. The resulting XDSM is shown in Fig. 6.11 and Fig. 6.12. Two main blocks are added on the diagonal: the DOE regulator and a converger. The DOE regulator provides the design points to be analyzed and collects the quantities of interest of the converged design. Inside the DOE cycle, every experiment provided by the DOE is converged based on the feedback variables MTOM, total lift and drag coefficients ($C_{f,x}$ and $C_{f,z}$) of the aircraft. The specification of this automated design process is the final result in step III that is stored in a CMDOWS file and visualized using VISTOMS, Fig. 6.11 and Fig. 6.12.

INSPECTION OF FORMULATED SOLUTION STRATEGY BY KADMOS IN STEP III

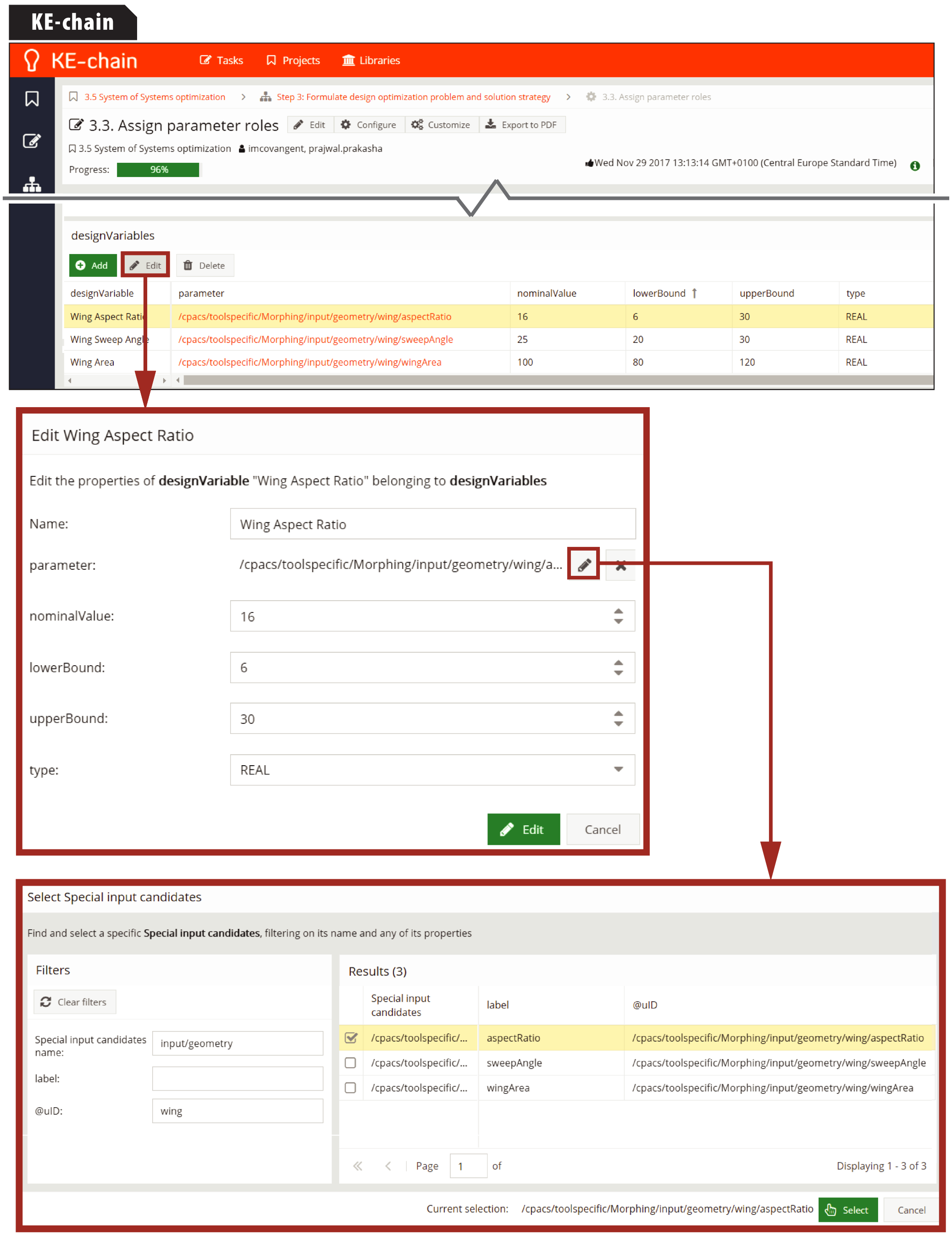
The various CMDOWS files used to store the evolving definition of the MDAO system, from RCG, to FPG and finally the combination of MDG and MPG, can all be visualized in a single VISTOMS web-based instance. In Fig. 6.11 the CMDOWS file containing the DOE solution strategy for the demonstrator system is depicted using a dynamic XDSM.

Fig. 6.11 shows how the competences have been organized by KADMOS in a meaningful order, compared with the unorganized RCG from step II (Fig. 6.9). The wing design variables selected for demonstrator DOE are provided to the workflow by the DOE block (Fig. 6.11, overlay frame 1) and then translated into a full parametric CPACS geometry by the Morphing competence. Subsequently, the wing geometry is processed to the downstream competences such as Aerodynamics and CostAnalysis. Here, it can be observed how the AircraftSynthesis competence only provides an initialized aircraft geometry, while the Morphing competence adjusts this geometry according to the given DOE inputs. The four main quantities of interest of the performed DOE can be examined in overlay frame 3, namely total recurring costs, fuel mass of the specified flight mission, ATR, and DOC.

In Fig. 6.12 a closer insight to the design competence operating inside the convergence loop is given, where the three previously mentioned convergence variables MTOM, $C_{f,x}$ and $C_{f,z}$ are displayed. The MTOM convergence is a typical iteration procedure in aircraft design, whereas the iteration of the aerodynamic coefficients is added to account for the adjustment of these coefficients by the nacelle design and integration design competence (NacelledesignAeroIntegration).

6.2.4. STEPS IV AND V

The CMDOWS file from step III is used here to bridge the gap between formulation and execution by automatically translating the neutrally stored solution strategy into an executable workflow for a PIDO platform of choice, see Fig. 6.13. The implementation details of step IV (workflow materialization) have been covered in Chapter 5 of this dissertation. After execution in the PIDO platform, the architect comes back to the KE-



3.5 System of Systems optimization > Step 3: Formulate design optimization problem and solution strategy > 3.3. Assign parameter roles

3.3. Assign parameter roles

3.5 System of Systems optimization imcovangent, prajwal.prakasha

Progress: 96%

Wed Nov 29 2017 13:13:14 GMT+0100 (Central Europe Standard Time)

designVariables

designVariable	parameter	nominalValue	lowerBound	upperBound	type
Wing Aspect Rati	/cpacs/toolspecific/Morphing/input/geometry/wing/aspectRatio	16	6	30	REAL
Wing Sweep Angl	/cpacs/toolspecific/Morphing/input/geometry/wing/sweepAngle	25	20	30	REAL
Wing Area	/cpacs/toolspecific/Morphing/input/geometry/wing/wingArea	100	80	120	REAL

Edit Wing Aspect Ratio

Edit the properties of **designVariable** "Wing Aspect Ratio" belonging to **designVariables**

Name:	Wing Aspect Ratio
parameter:	/cpacs/toolspecific/Morphing/input/geometry/wing/a...
nominalValue:	16
lowerBound:	6
upperBound:	30
type:	REAL

Select Special input candidates

Find and select a specific **Special input candidates**, filtering on its name and any of its properties

Filters

Special input candidates name:	input/geometry
label:	
@uID:	wing

Results (3)

	Special input candidates	label	@uID
✓	/cpacs/toolspecific/...	aspectRatio	/cpacs/toolspecific/Morphing/input/geometry/wing/aspectRatio
☐	/cpacs/toolspecific/...	sweepAngle	/cpacs/toolspecific/Morphing/input/geometry/wing/sweepAngle
☐	/cpacs/toolspecific/...	wingArea	/cpacs/toolspecific/Morphing/input/geometry/wing/wingArea

Page 1 of

Displaying 1 - 3 of 3

Current selection: /cpacs/toolspecific/Morphing/input/geometry/wing/aspectRatio

Figure 6.10: KE-chain provides an interface for KADMOS to the architect in step III.3 to assign special roles to CPACS elements. This example shows that the architect edits design variable *wing aspect ratio* (middle frame) and selects its corresponding CPACS element through a search dialog (bottom frame) that filters through the entire set of 4371 elements.

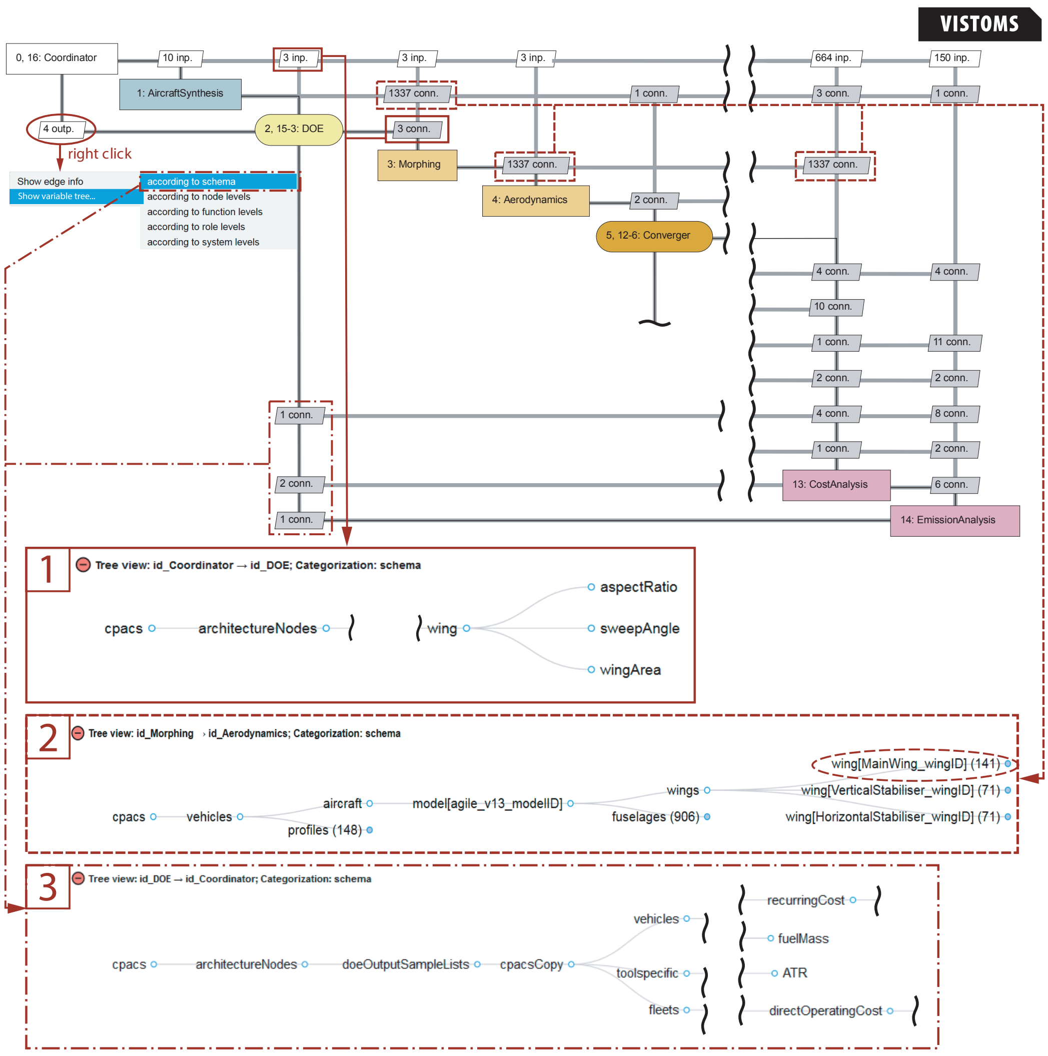


Figure 6.11: Extract of the KADMOS data and process graphs for the presented design case as VISTOMS XDSM view

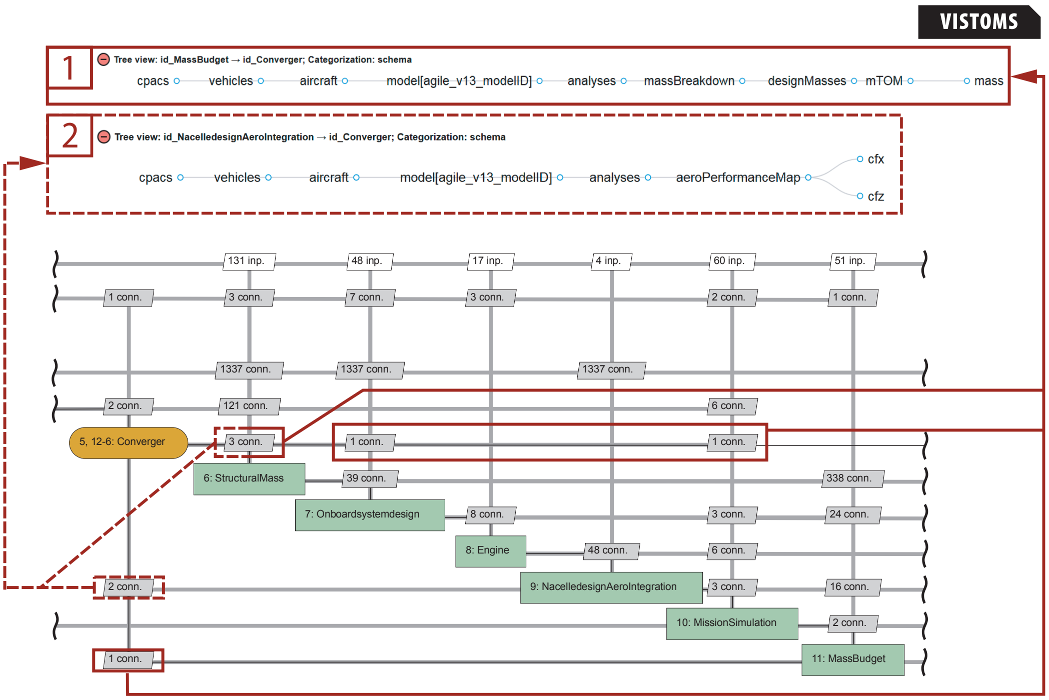


Figure 6.12: VISTOMS generated XDSM showing details of the DOE system convergence loop

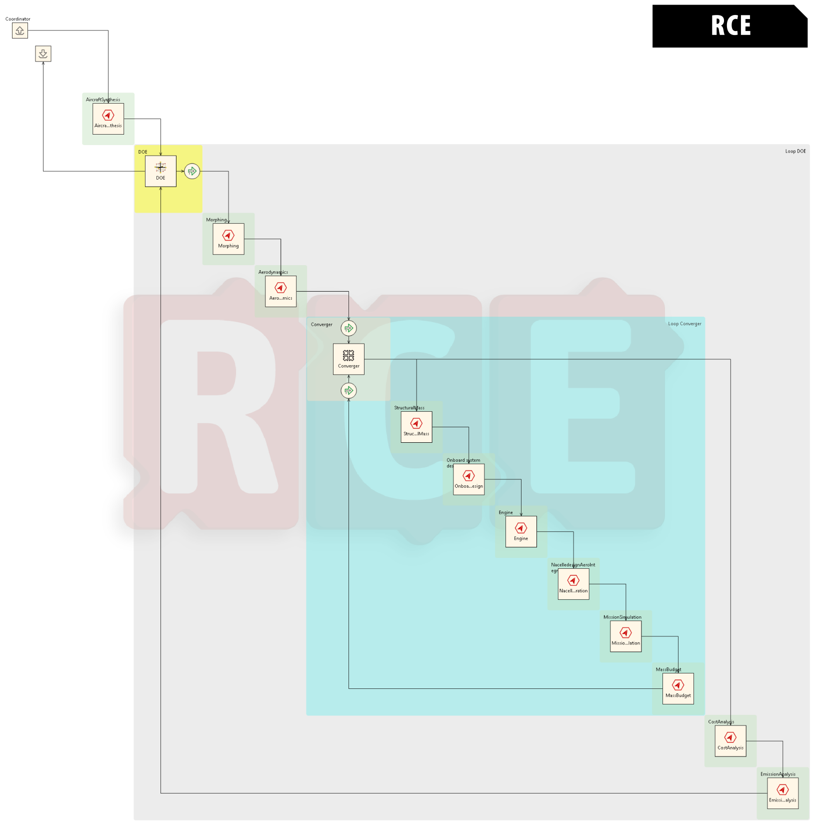
chain platform to upload the results and use the platform's post-processing engineering services to support the result discussion with the design team. One such engineering service is the integration of the ID8 [S28] visualization platform.

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

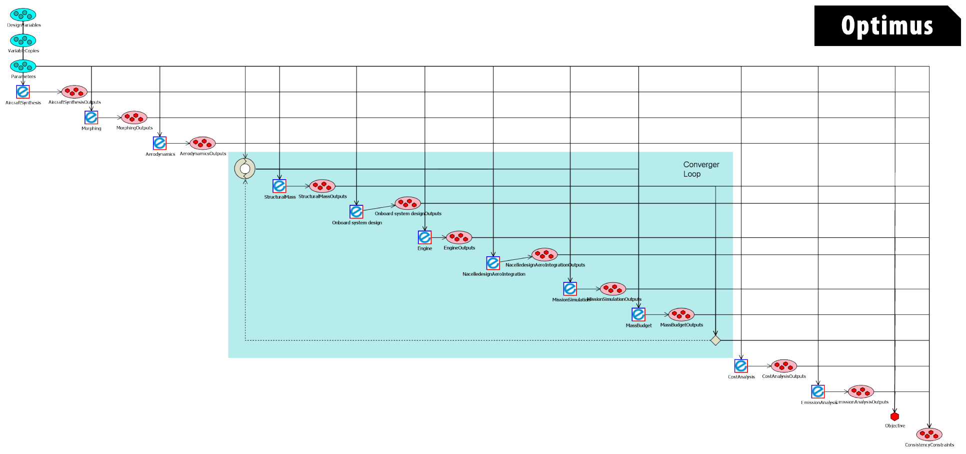
With the ADF fully configured, tested, and demonstrated in the second AGILE design campaign (see Fig. 2.18), the third and final design campaign was used to assess the framework and its individual components in four design cases concerning unconventional aircraft. These cases will be described in the upcoming section, followed by a critical assessment of the developments presented in Part II of this dissertation.

6.3. Framework assessment: aircraft design cases

In the final year of the AGILE project, a series of collaborative aircraft design cases (addressed in the project as *design tasks*) was performed, with the main goal of testing the AGILE Development Framework (ADF) and to evaluate the performance of the novel applications to support the automated development process described in this chapter and



a) RCE



b) Optimus

Figure 6.13: Screenshots of executable workflow instances for two different PIDO platforms. Both workflows are based on the same CMDOWS file that were provided by KADMOS in step III.4 and extended in step IV.

in Part II of this dissertation. In the next section, three key components of the framework are assessed:

- MDAO system storage with CMDOWS;
- MDAO system formulation with KADMOS;
- workflow materialization in RCE and Optimus.

These components* and their role in the ADF are visualized in Fig. 6.6 and were elaborately described for the demonstrator case in §6.2. In this chapter, four AGILE design tasks related to unconventional aircraft configurations are used to test the developed approach.

6.3.1. Design case descriptions

The four collaborative design tasks are briefly introduced here to highlight their collaborative set-up, main objective, and challenges. The results of using the ADF to formulate MDAO solution strategies in these different design cases will be summarized in §§6.3.2.

Strut-braced wing (SBW)

design team DLR (task leader) with eight partners all based at different locations in five different countries

design task objective The emphasis in this task is on the integration of the wing-strut system. The combination of structural and aerodynamic analysis is of key importance. To start, the operational requirements are kept constant while the geometrical parameters of the wing-strut system (e.g. aspect ratio, span wise position of the wing-strut connection and struts' thickness-to-chord ratio) will be set as design variables for a DOE study. Then, also changes on the aircraft operational specifications (e.g. cruise Mach number and altitude) will be investigated. The Strut-Braced Wing (SBW) configuration will be optimized to minimize DOC, cash operating cost, life cycle cost, mission fuel, and/or other composite functions.

(re)configuration challenges A large and heterogeneous set of design and analysis tools (e.g. flight dynamics, on-board systems, aerodynamics, structures, costs) from many different partners must be integrated in different collaborative workflows. Severe reconfigurations are envisioned to move from the DOE studies based on configuration parameters, to the design studies with varying operational requirements, and finally to the setup of an optimization process to minimize one of the cost metrics.

Box-wing aircraft (BWA)

design team ONERA (task leader) with seven partners all based at different locations in six different countries

design task objective The design scope for this configuration includes both mission parameters and aircraft design parameters. The objective is to first identify the best Box-Wing Aircraft (BWA) configurations for a wide range of missions parameters,

*Other components of the framework have also been assessed, but are omitted here as they are out of scope of this dissertation. Details can be found in [108].

then to select the type of mission giving the BWA superiority with respect to conventional aircraft designs. The main objective of the optimization will be minimization of DOC. The broad multidisciplinary focus for this configuration includes aerodynamic characterization including control surface behaviour and high-lift capabilities; the structural design of the wings and fuselage including the aeroelastic effects; more electric on-board system architecture; stability and control, and flying qualities including the definition of a dedicated flight control system.

(re)configuration challenges An extensive heterogeneous set of design and analysis tools from many different partners must be integrated in a design workflow, with a stepwise approach. Different levels of fidelity will be required, ranging from fast semi-empirical methods up to high-fidelity CFD- and FEM-based analysis to assess the aeroelastic behaviour of the airframe and the performance of the novel system of control surfaces distributed over the boxed wing.

Blended-wing body (BWB) novel propulsion/airframe integration

design team CFSE (task leader) with eight partners all based at different locations in five different countries

design task objective The objective for this novel configuration is to minimize the fuel consumption for a maximum range constraint at maximum payload, focusing on the embedded engines integration aspects, based on Boundary Layer Ingestion (BLI). The definition of the number and location of the engines, the design of the air intakes for active BLI control and the aerodynamic characterization of the full Blended-Wing Body (BWB) configuration are the main aspects of this optimization study.

(re)configuration challenges Heterogeneous set of high-fidelity design and analysis tools (e.g. aerodynamics, on-board systems for BLI control, engine integration) from different partners should be integrated in a collaborative workflow. Due to the use of tools that are not fully CPACS-compatible (mainly the high-fidelity aerodynamic tools requiring accurate CAD files as input), the automated integration and execution of the workflow is expected to be particularly challenging.

Turboprop aircraft (TPA)

design team UNINA (task leader) with nine partners all based at different locations in six different countries

design task objective This Turbo-Prop Aircraft (TPA) task aims at comparing a novel turboprop configuration with fuselage-mounted engines with a more conventional wing-podded version. The main focus of the task concerns the minimization of DOC, while including noise constraints in accordance with regulations. Main design variables will be the geometry of the wing for the two configurations: wing area, wing span, wing thickness ratio and wing taper ratio. A key constraint is the wing sweep angle which must be kept minimum in view of exploiting the drag reduction of natural laminar flow wings.

(re)configuration challenges As for the other design tasks, the integration of a heterogeneous set of design and analysis tools from many different partners makes for the main challenge. In this case, the need to execute two optimization studies on two aircraft configurations will put extra constraints on the overall computational performance.

6.3.2. Summarized results

The design teams working on the four design tasks have used the ADF to formulate workflows of increasing complexity, based on their own task objectives. The detailed process of using the ADF has been described for the demonstrator in §6.2. A top-level overview of the results generated by following the AGILE development process is summarized in Fig. 6.14 and Fig. 6.15. Each XDSM in these figures represents a CMDOWS file generated by KADMOS, containing the formulation of a solution strategy that was obtained using the ADF. Finally, the CMDOWS files were parsed in RCE or Optimus to materialize executable workflows. Such automatically generated workflows were shown in Fig. 6.13 for the demonstrator.

First, all design tasks configured a simple MDA to test the different tools and converge a single design point, see second row in Fig. 6.14 and Fig. 6.15. Subsequently, DOEs were configured to explore the design space, using the previously tested MDA, third row in the snapshot figures. In some cases, DOEs have been set up in view of generating surrogate models for use in later optimization runs. Finally, MDO architectures, such as MDF and IDF, were formulated to run actual optimizations, the XDSM snapshots are depicted in the last row of Fig. 6.14 and Fig. 6.15.

6.4. Assessment results

After the formulation and materialization of the different workflows, feedback was gathered from the AGILE consortium partners on the different components of the developed framework through an online survey. In this survey, users were asked to share their experience using the ADF and to indicate the strengths, limitations, opportunities and risks for each component, as well as for the framework as a whole. About thirty AGILE partners have provided their feedback. These partners played different roles within the design tasks, had different levels of experience in using the framework, and worked from different perspectives being based at one of the three aircraft manufacturers, three research institutes, four universities and four software solution and engineering service providers involved in the project. The feedback from the survey on the three framework components original to this dissertation will be elaborately discussed in the next subsections.

6.4.1. MDAO system storage with CMDOWS

The assessment of CMDOWS as a central data schema at the core of a third-generation MDAO framework has been postponed to this moment. The schema's ability to support the materialization of executable workflows has been verified and validated in the previous chapter. However, it is in the ADF that the full schema was put to the test regarding the information exchange it is supposed to enable between a range of framework applications (see Fig. 6.5) to efficiently build and extend a third-generation framework, and perform multiple, collaborative MDAO projects with it. Therefore, the CMDOWS assessment presented here is split in two parts: requirements-based assessment and survey-based assessment.

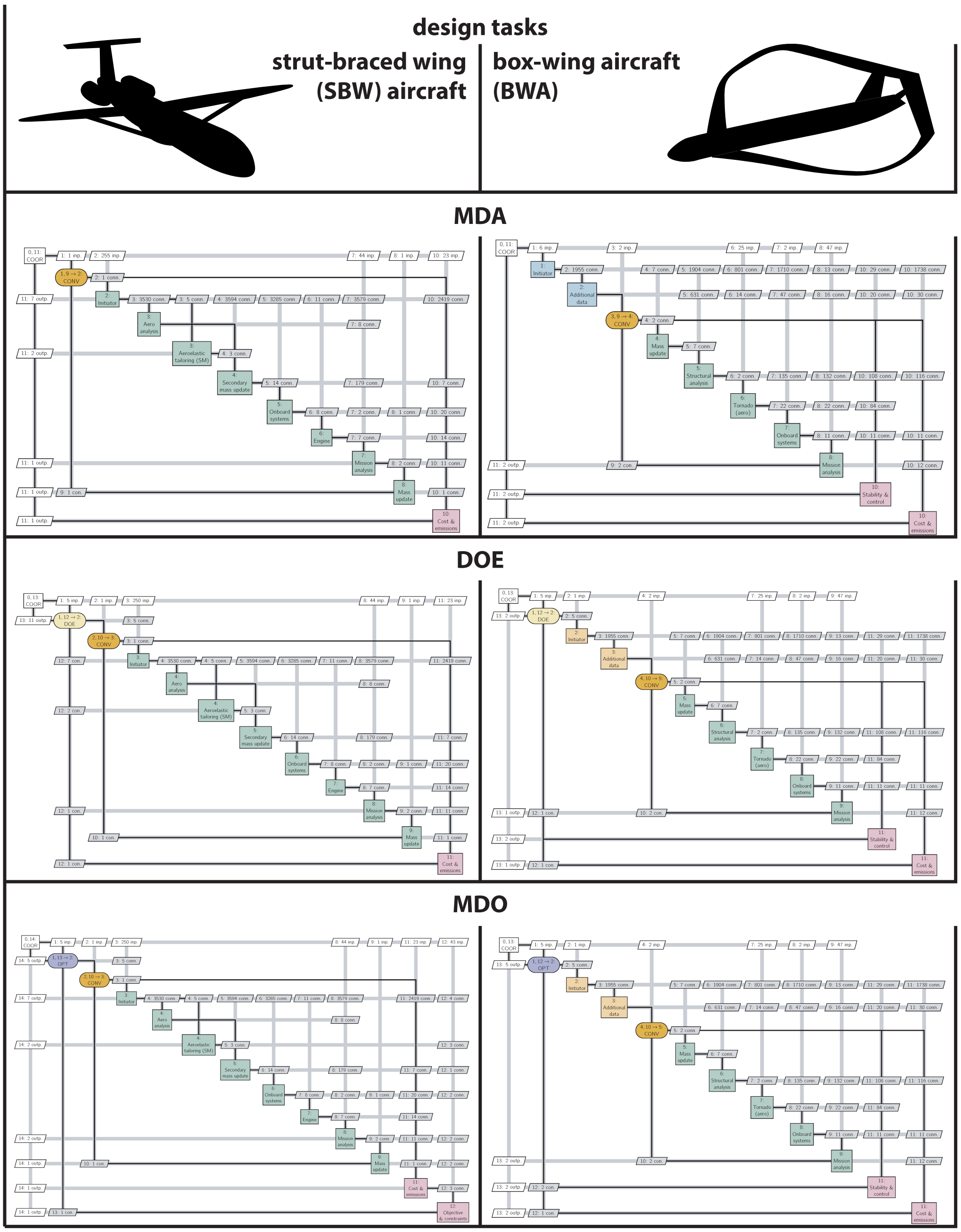


Figure 6.14: Snapshots of the XDSMs of the formulated workflows for the SBW and BWA design tasks. Each XDSM corresponds to a CMDOWS file containing the neutral definition of the solution strategy.

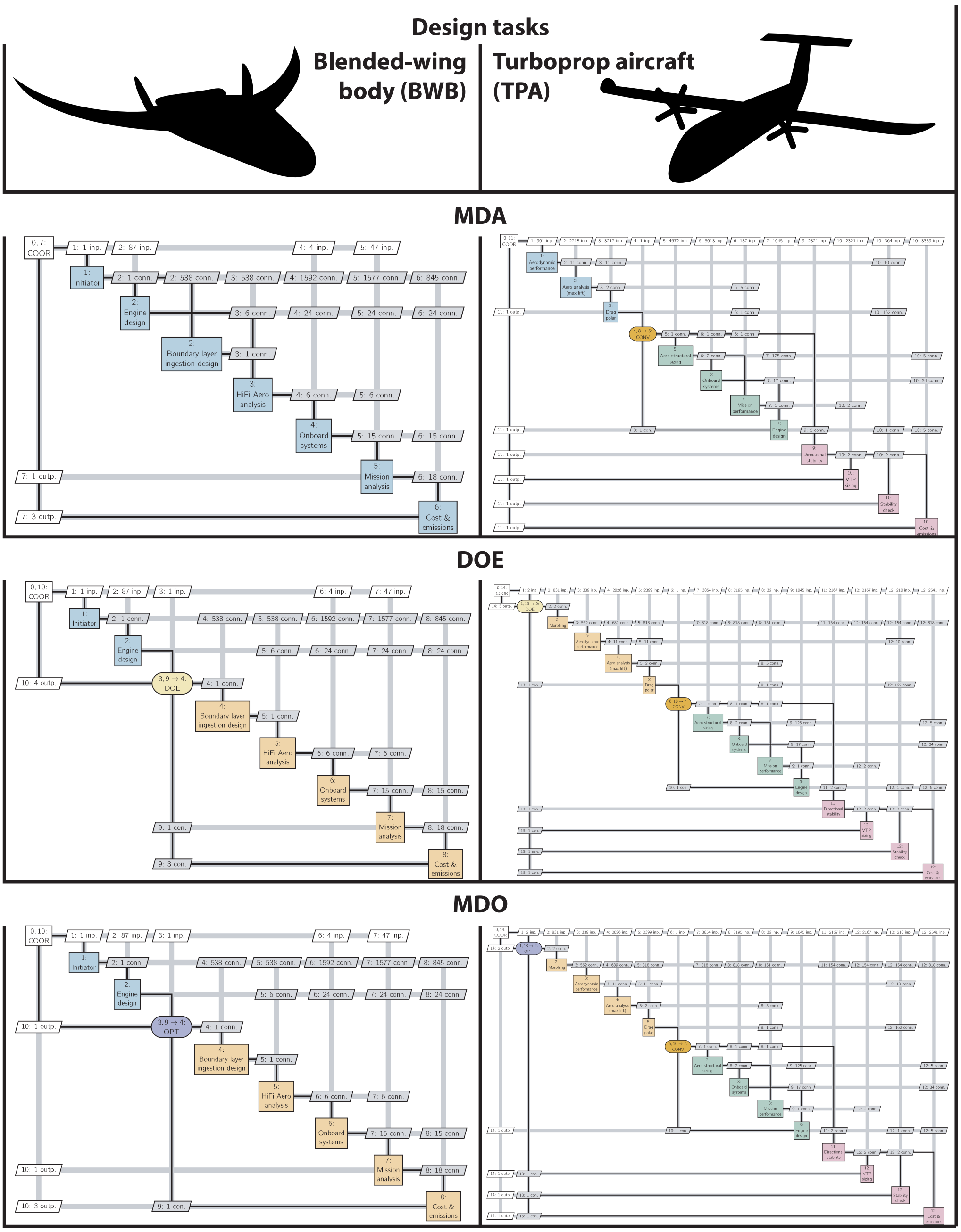


Figure 6.15: Snapshots of the XDSMs of the formulated workflows for the BWB and TPA design tasks. Each XDSM corresponds to a CMDOWS file containing the neutral definition of the solution strategy.

Requirements-based assessment

The functional requirements for CMDOWS, which were listed and described in §4.2, are repeated here:

I Machine-interpretable
II Human-readable
III Neutral
IV Validation
V Adaptable
VI Balance of redundant information
VII Support all MDAO system stages
VIII Support all MDAO framework categories
IX Support disciplinary tool heterogeneity

The solution strategies shown in Fig. 6.14 and Fig. 6.15 clearly shows that the schema satisfies the tool heterogeneity requirement (req-IX). Three types of tools are integrated in one top-level collaborative workflow. The first type is the local tool. Tools of this type have been integrated directly in Optimus and RCE and can be executed from the same system as the MDAO workflow. However, many of the tools in the solution strategies are owned by other partners and need to be executed remotely because of intellectual property restrictions. Hence, this second type of tool cannot be distributed to be executed locally. These tools are stored as remote tools in the CMDOWS files, so that the materialized workflows will contain Brics components (discussed in §§5.2.1) to run the actual tool on another server domain. Finally, simple mathematical functions, usually describing objective and constraint equations are also stored using the schema.

The development of the ADF has shown the compliance level of the current schema. The XML CMDOWS instances support both human-readability (req-II) and machine-interpretability (req-I). This human-readability is also proven by the fact that many developers of AGILE framework applications have been able to connect to CMDOWS in a short time. The neutrality of the schema (req-III) has been maintained, even when adding new elements to support the links with the AGILE framework applications. Hence, there are no traces of application-specific elements like KADMOS, KE-Chain, Optimus, etc. Moreover, the core structure of CMDOWS still allows adjustments (req-V), as the schema was extended step by step to link different applications and more adjustments can be made to meet future demands.

A key future improvement that was found concerns the redundancy of the content of the schema (req-VI). Throughout its development, initial CMDOWS versions were always very lean in the information stored in a CMDOWS file. Some of the application links demanded that certain information is stored explicitly in the schema, even though this information can be interpreted from the information already stored. In future developments, the links with applications should be checked for this type of information and per case it should be decided whether to explicitly add the information in the schema.

The demonstrator (§6.2) and the four unconventional aircraft cases (§6.3) have shown that three MDAO system stages (req-VII) are supported, including the bridge to the fourth stage in execution phase by workflow materialization. At the moment, the links between framework applications and CMDOWS are not always bidirectional (req-VIII), see Fig. 6.5, but all primary links (as explained in Fig. 4.4) have been established for the six application categories. In future work, applications will be extended and the secondary links will also be developed to enhance the capabilities of the ADF.

SURVEY-BASED ASSESSMENT

Survey respondents indicated that CMDOWS has thoroughly demonstrated its capability to act as a data standard to integrate the heterogeneous collection of applications developed and deployed in AGILE, such as KE-chain, KADMOS and VISTOMS. Within the four design tasks, no major issues or limitations were encountered concerning the schema. A minor issue found, was the size of the files, which increase significantly for design tasks involving large systems (>100MB for a single XML file) and a lot of CPACS elements (>4000).

Proof of the increased agility offered by the adoption of CMDOWS was provided when, in the later development stage of AGILE, new applications needed to be integrated in the framework, such as the SMR and OpenMDAO. The integration of the SMR in the framework involved links to KE-chain, KADMOS and VISTOMS. This was achieved in a short time using CMDOWS-based links between the applications, without requiring significant changes or adjustments of any of the involved applications. Similarly, the OpenMDAO platform was conveniently linked to the broader framework through a newly developed Python package OpenLEGO, which was discussed in §5.4.

Another form of increased agility came forward in the way CMDOWS provides a convenient format to store the full MDAO system definition at any moment in the project's development process. The different stages of the system are easily stored and versioned for future reference. This allowed system integrators to quickly try out system (re)configurations and go back to earlier version, if required. In this sense, the format facilitates version control and management of a large and complex system.

Limitations of the schema mainly stem from its relatively low level of maturity. Contrarily to CPACS, CMDOWS was developed from scratch within the AGILE project and has therefore (at the time of the survey) only been tested and applied using AGILE design tasks. Therefore, new design tasks might still bring forth new requirements not considered at the time the schema was conceived. Based on feedback received in the survey, it is suggested to extend the schema in the following directions:

- support a wider range of methods to execute disciplinary tools (e.g. command line execution on different operating systems);
- improve the implementation of surrogate models (e.g. their relation with the original tools used to built them, more advanced definitions to store different surrogate model types);
- include support for a wider variety of workflows: now only MDAO architecture-based workflow types provided by KADMOS are supported;
- add measures to reduce the file size;
- extend the documentation available in the schema repository.

As a side note, although the AGILE framework strongly benefits from the adoption of CMDOWS, a number of criteria should be considered to evaluate its convenience in a generic MDAO study case:

- size of the MDAO system under consideration
- heterogeneity and distribution of the team
- heterogeneity of the MDAO framework

- maturity of the existing MDAO framework with respect to using CMDOWS

When dealing with large (in terms of number of involved tools and coupling parameters) MDAO systems, design teams will benefit greatly from adopting CMDOWS to quickly set up a coherent tool repository and use that repository to formulate the problem and solution strategy. Similarly, a heterogeneous and distributed team (many specialists with different backgrounds working at different locations) will benefit from CMDOWS to serve as a *common language* to streamline the definition and use of systems of any size. The heterogeneity of the framework, as indicated in Fig. 6.5, has been the main motivation for developing CMDOWS and is also the strongest indication for its effective use. There is a general skepticism in the development and adoption of monolithic, holistic solutions that can cover all the aspects of performing MDAO projects collaboratively, e.g. system definition, visualization, problem formulation, reconfiguration and execution. Such solutions, whenever available, become obsolete quickly and are typically inflexible. A simple but comprehensive standard format as CMDOWS, on the other hand, allows the (re-) integration of many different applications, both commercial and in-house developed, thus providing maximum flexibility, scalability and adaptability.

Finally, it is worth noting that not all people involved in a project necessarily need to familiarize with CMDOWS, as this depends on the level of maturity of the MDAO framework. In a mature framework all applications will already use and produce CMDOWS files (see Fig. 4.4) and most people involved simply use these applications to perform their tasks. Only people involved in developing and maintaining the applications will have to invest time for familiarization. Based on experience in the AGILE project, this familiarization time is limited to 2-3 days with some additional time required to understand the basic concepts of MDAO and XML. The AGILE project has proven that CMDOWS interfaces can be easily developed, including parsers for a heterogeneous set of PIDO platforms (RCE, Optimus). Once the interfaces are in place, the presence of CMDOWS is transparent to the user, who does not require any direct manipulation nor familiarization with the format itself.

6.4.2. MDAO SYSTEM FORMULATION WITH KADMOS

As depicted in Fig. 6.14 and Fig. 6.15, KADMOS has successfully played its role as formulation platform for MDAO systems, from tool repository to solution strategy, for a variety of aircraft configurations and heterogeneous design teams. The first step where KADMOS comes into play (step II) is also one of the most appreciated capabilities of the platform by AGILE users: to import the I/Os of a set of CPACS-compatible tools and inspect the couplings between them. In combination with the visualization by VISTOMS, this enables the design team to correctly connect their disciplinary analyses through CPACS, especially when many elements (>4000) from CPACS are involved. Also in the subsequent steps of the problem and solution strategy formulation, KADMOS was found to be a valuable component to the framework, acting as the ‘engine’ behind the cockpit that was provided by KE-chain in the formulation phase of the five-step approach.

The strong link between KADMOS and CMDOWS has also given KADMOS the secondary role as a library of functions to handle CMDOWS files, since KADMOS is able to transform the CMDOWS file to a meaningful graph construct. This role of KADMOS is extensively used within KE-chain and VISTOMS to read and edit CMDOWS files. In conclusion, the integrators of the design tasks experienced that KADMOS makes the task of formulating heterogeneous, distributed systems in a collaborative environment much less cumbersome, with an estimated time reduction of 49%. Three main limitations were identified with having KADMOS as the system formulator in the framework.

No user friendly interface / graph interaction

KADMOS is a Python package and does not have its own GUI. In the ADF a user interface was built in KE-chain, where four different forms are used to progress the system definition from tool repository to solution strategy in step III. Based on the form input provided by the user, different KADMOS system manipulation methods were executed 'behind the scenes'. However, this way of implementing a KADMOS user interface was found to be too inflexible for some of the peculiarities encountered in the design tasks. This was solved temporarily by enabling the inclusion of custom KADMOS scripts on the platform, even though such scripts dramatically lower the user friendliness of the interface. Instead of the form-based manipulation of the graphs currently supported by KE-chain, the user interface should interact more directly with the graph objects and provide a workbench to 'craft' the graphs directly into the desired form.

The envisioned way forward to achieve this type of interface is to further develop VISTOMS as a full-fledged GUI for KADMOS. This development was started after the survey and released as an open-access website called the 'MDAO system interface'*. Another suggestion in the survey was to include a debugging mode in this interface to visualize the results of graph analyses performed by KADMOS. These results can then be rendered in VISTOMS to highlight issues with the MDAO system. Such analyses would include typical problems for MDAO systems, such as multiple tools writing the same CPACS element (collision), or tools without any I/O connections.

Inflexibility of imposing architectures

To enable the high level of automation in the formulation phase, a certain inflexibility was implemented in the way KADMOS imposes architectures on MDAO problems. The current approach permits a strict boundary between the problem and the solution strategy, where different architectures (e.g. MDF, IDF) can be used to get different strategies for solving the same problem. This approach results in a low number of settings to be provided by the user, but at the same time it is difficult to set up a more custom-built approach. This issue will be further discussed in Chapter 8.

Related to additional strategies, another strong wish in the survey was to include additional intelligent analysis of the tool repository. For example, KADMOS could suggest the selection of tools balancing fidelity level and execution time, or the best way to cluster a set of tools to build a surrogate model first, and then execute an optimization using that model. This is another topic that will be covered in Chapter 8.

Expected rigidness of tool I/Os (on project and configuration level)

A final, rather detailed, limitation touches upon the core of the AGILE paradigm: the use of a fixed schema (i.e. CPACS) for tool integration in combination with the tool I/O definition required by KADMOS. This limitation is visually explained in Fig. 6.16 and has to do with the fact that, to be able to support formulation, KADMOS has to be very strict in the description of a tool's I/Os, to make sure they are mapped correctly. Hence,

*available at: `mdo-system-interface.agile-project.eu`, accessed June 11, 2019

KADMOS demands that the I/O definition includes the precise definition and number of certain CPACS elements, while CPACS-compatible tools can usually handle a larger variety in the I/O files and can still be executed. For example, consider that all spars of the main wing are input of a tool. In case two spars are defined in the CPACS file, KADMOS needs to get the following input element definitions, see also the first row of Fig. 6.16:

element 1: /cpacs/vehicles/model/aircraft[@uID="Boeing-747"]/
wings/wing[@uID="main_wing"]/spars/spar[1]
element 2: /cpacs/vehicles/model/aircraft[@uID="Boeing-747"]/
wings/wing[@uID="main_wing"]/spars/spar[2]

Hence, the input definition specifically states that the aircraft is the Boeing-747, the wing has to be the main wing and then the first and second spar elements are used. The input definition has to be very specific, since the CPACS file might contain other aircraft and/or other wings (i.e. horizontal and vertical tails) for the Boeing-747, and their spars might then be coupled illegally to other tools.

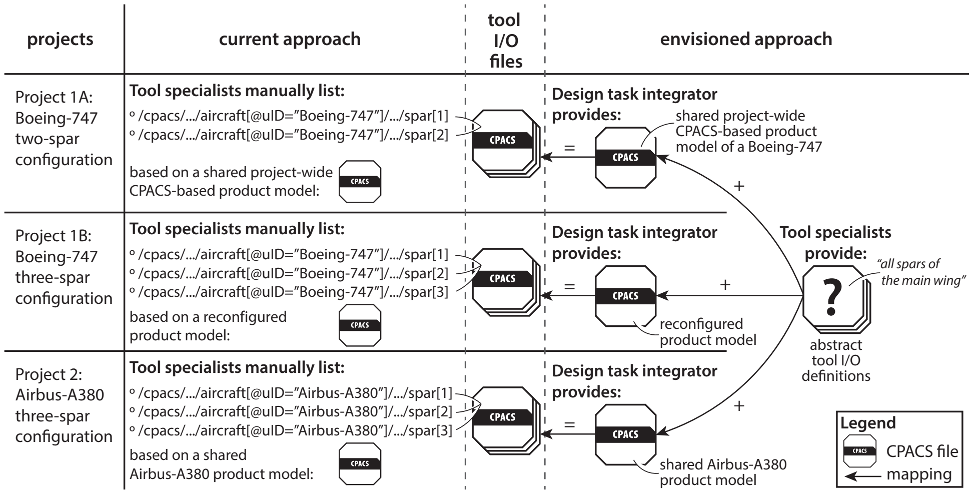

projects
current approach
tool I/O files
envisioned approach
Project 1A: Boeing-747 two-spar configuration
Tool specialists manually list:
° /cpacs/.../aircraft[@uID="Boeing-747"]/.../spar[1]
° /cpacs/.../aircraft[@uID="Boeing-747"]/.../spar[2]
based on a shared project-wide CPACS-based product model:
CPACS
Design task integrator provides:
shared project-wide CPACS-based product model of a Boeing-747
=
+
Project 1B: Boeing-747 three-spar configuration
Tool specialists manually list:
° /cpacs/.../aircraft[@uID="Boeing-747"]/.../spar[1]
° /cpacs/.../aircraft[@uID="Boeing-747"]/.../spar[2]
° /cpacs/.../aircraft[@uID="Boeing-747"]/.../spar[3]
based on a reconfigured product model:
Design task integrator provides:
reconfigured product model
Tool specialists provide:
"all spars of the main wing"
?
abstract tool I/O definitions
Project 2: Airbus-A380 three-spar configuration
Tool specialists manually list:
° /cpacs/.../aircraft[@uID="Airbus-A380"]/.../spar[1]
° /cpacs/.../aircraft[@uID="Airbus-A380"]/.../spar[2]
° /cpacs/.../aircraft[@uID="Airbus-A380"]/.../spar[3]
based on a shared Airbus-A380 product model:
Design task integrator provides:
shared Airbus-A380 product model
Legend
CPACS file
mapping

Figure 6.16: Visual explanation of the current approach for defining tool I/O files (left) and the envisioned approach (right) for a more flexible definition using a baseline file mapping

Unfortunately, because of this required strictness, I/O issues arise at two levels: geometrical reconfiguration and project-to-project sharing. With the strict I/O definition, it is not possible to change the number of spars inside KADMOS without explicitly changing the input file. Hence, if the geometrical configuration (number of spars) changes in the workflow based on some variable, then this might lead to inconsistencies and errors, see second row in Fig. 6.16. In addition, also between projects tool I/Os can only be shared if the exact same aircraft (i.e. Boeing-747) is used in the other project as illustrated in the last row of Fig. 6.16. Of course, the UIDs could be replaced, but the essential point here is that a more abstract I/O definition would be required for a more flexible formulation process, such that tool I/Os are defined strictly enough for KADMOS to establish valid couplings, while at the same time geometrical reconfigurations or project-to-project sharing are handled automatically. Practically, this would mean that the earlier spar input definition would not be explicit CPACS elements, but something along the line of: *all spars of the main wing of the aircraft.* Thus there needs to be a mapping of abstract I/O definitions to specific elements of the CPACS-based product model used in a project. The product model is then a single file that has to be adjusted, instead of having to adjust the full collection of I/O files.

Hence, to improve the use of tool definitions between projects and with geometrical reconfigurations, a *semantic enrichment layer* is required on top of the common schema. This layer would allow a tool I/O definition that is independent of the design task and enable the automatic population of I/O files for a specific task using a mapping approach with the product model. This envisioned approach is visualized in the last column of Fig. 6.16.

6.4.3. Workflow materialization in RCE and Optimus

All the CMDOWS files of the formulations shown in Fig. 6.14 and Fig. 6.15 have been successfully parsed as executable workflows in RCE or Optimus. One example from the demonstrator design task was shown in Fig. 6.13 and annotated workflows were discussed in Chapter 5, see Fig. 5.2 and Fig. 5.5. Being one of the latest AGILE outcomes, CMDOWS parsing is the least mature component of the framework. The two most critical issues, as emerged from the survey, are discussed below.

Large overhead due to data handling

Experienced PIDO platform users noticed long execution times for relatively straightforward workflows. This is because the CPACS-based tool integration in the AGILE paradigm has major implications on the data handling in the workflows, as was discussed in Chapter 5. Not all elements in RCE and Optimus were originally built to handle XML files, and therefore XML readers and writers are required to integrate them in the workflow, see for example the optimizer and converger elements in Fig. 5.2 and Fig. 5.5. Similar data handling has to be done when two tools are executed in parallel and their CPACS output files need to be merged. This continuous data translation adds a large overhead to the execution time of the workflow.

Lazy data handling approach leads to inconsistent CPACS files

Specific for RCE, in order to reduce the overhead discussed above, a 'lazy data handling' approach is implemented, which means that CPACS-compatible tools are assumed to read a CPACS file and then append their results to the same file. This way, in a sequential execution of CPACS-compatible tools, the files can be passed directly without data handling overhead. However, when tools are executed in parallel then their results need to be merged as one CPACS file if the next tool requires results from both tools, as is the case for the two tools in the lower right corner of the RCE workflow depicted in Fig. 5.2. Since data of more upstream tools have been added to the CPACS files of the parallel tools and were forwarded indirectly based on the lazy data handling, the correct merging of the files is not always evident. There is the risk that a value was updated by a tool further upstream of one parallel tool, that might be ignored if the merging of the files takes the old value from the other parallel tool.

These data handling issues could be resolved by implementing a stricter data handling for parsed workflows, though such data handling will also have a negative impact on the already large overhead involved. KADMOS is aware of the exact I/Os of each tool and stores this information in the CMDOWS file. Therefore, the information could be used in the workflows for data handling, among others to assure the consistency of the CPACS files being generated and merged. At the same time, the data handling should be less strict when possible to reduce overhead. Thus, the parsing method should be a trade-off between strict data handling for CPACS consistency and overhead reduction.

An average time reduction of 33% was estimated by the expert users of RCE and Optimus in the survey. Although the CMDOWS standard has already been adopted by a wider variety of framework applications, its original development goal was to enable the creation of executable workflows from a neutral workflow definition. This position of CMDOWS could be further strengthened if more commercial PIDO platforms would adopt this, thus creating a virtuous snowball effect leading to a wider adoption of the schema, and possibly to the upgrade of CMDOWS as official standard. Feedback from the commercial PIDO tool provider in AGILE is very positive and they consider this approach the way forward to improve their capability to support MDAO expert customers with customized deployments of their workflow management system.

6.4.4. Complete AGILE development framework

The complete framework was positively reviewed in the survey and AGILE partners estimated that a 39% time reduction for setting up collaborative MDAO system could be achieved. A trend found in the survey feedback was that technically the framework is performing well, but to grasp, use and appreciate it still requires effort. This confirms the idea that the third-generation framework is the technical implementation of a truly novel paradigm to perform MDAO, as such it takes a mind shift and some time to get acquainted. As the current version of the framework is technically sound (with only minor bugs and issues to be solved for the different components), it would make sense to reduce the effort for embracing the AGILE paradigm by consolidating the different components and making GUIs more intuitive, informative and structured.

Conceptually, the ADF connects two very distinct phases of developing a collaborative system: formulation and execution. A crucial limitation found in AGILE, caused by the use of a fixed central data schema (i.e. CPACS), is that a very strict definition of tool I/Os is necessary for a specific design task in the formulation phase, while tool execution should preferably be based on less strict I/Os so that tools can handle a variety of CPACS files and thereby be used simultaneously in different design tasks. This formulation-execution discrepancy impacts many applications. As it relates back to the root of the AGILE paradigm, future developments should focus on how to tackle this discrepancy starting at the adopted central data schema and its ecosystem.

Another topic, that has not been addressed in this chapter nor in the framework as a whole, concerns the use of derivatives to speed up gradient-based optimization strategies. Most of the tools available within the AGILE consortium were not able to provide derivatives and given the fact tool development was not a goal of the project, it was decided not to include the use of derivatives in the framework. Naturally, including this concept in the framework would impact all components; derivative values need to be stored in the CPACS files, their role in the multidisciplinary system needs to be stored in CMDOWS files, handled properly by KADMOS, and finally, executable workflows need to be parsed that include optimizers using the derivative information. As derivatives offer a huge potential for enabling large-scale MDAO, it is considered to be another crucial topic for future development. With the OpenLEGO implementation discussed in Chapter 5, a first step has been taken in this direction, though the use of derivatives is achieved 'implicitly' there through the tool integration and not considered by other applications (i.e. KADMOS, CMDOWS).

In conclusion, this chapter has presented a cornerstone of the AGILE paradigm: the knowledge architecture. The AGILE implementation of this architecture as the AGILE Development Framework has been used as the environment to apply and assess the developments original to this dissertation in a broader framework with multiple collaborative aircraft design case studies. The AGILE development framework is considered a first step in the right direction to establish a third-generation framework, as well as an initial definition of the paradigmatic shift required to perform collaborative, distributed MDAO accounting for the technical and non-technical hurdles that it may present. This chapter confirmed the crucial role played by this dissertation's developments, as they offer valuable methodological enhancements to the AGILE development process that resulted in significant time reductions for the (re)configuration of large, complex MDAO system where a heterogeneous team of experts is responsible for its formulation and execution.

7 RESTRUCTURING A STATE-OF-THE-ART AIRCRAFT DESIGN TOOLBOX

In Chapters 3 to 5 a fully automated chain was established from MDAO system formulation (in KADMOS) to execution (in multiple PIDO platforms). These developments were implemented and tested in the AGILE framework in Chapter 6, which showed how automating the system development process significantly impacts the collaborative design effort. In this chapter, the same developments are applied in a completely different context to demonstrate how the same technology demonstrated in AGILE can be used to improve the scalability and flexibility of existing frameworks. This chapter will present a novel framework based on the restructuring of the existing aircraft design toolbox 'The Initiator'. Thereby, the framework acts as a proof of concept to show that the developments in this dissertation are also applicable outside the boundaries of the AGILE paradigm and can improve existing multidisciplinary design tools. Each section in this chapter covers a single iteration of the MDAO development process shown in Fig. 7.1. After a brief introduction of the toolbox in §7.1, the framework is configured and validated in §7.2. Then, a first reconfiguration is described in §7.3, followed by a final second reconfiguration in §7.4.

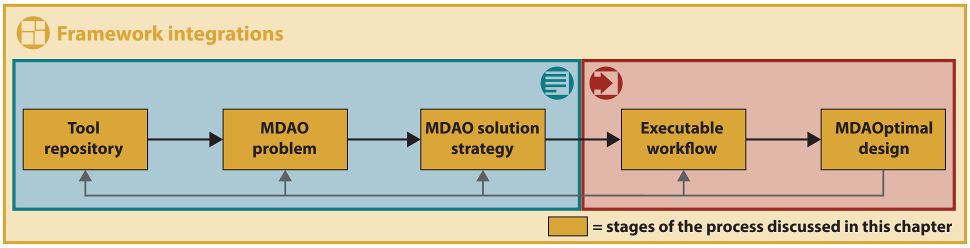


Figure 7.1: Overview of the five MDAO system stages covered in this chapter using the restructured aircraft design toolbox

7.1. Introduction: the Initiator toolbox

The section of Flight Performance and Propulsion at Delft University of Technology carries out research on the design of conventional and unconventional aircraft continuously. As each configuration includes similar steps to synthesize a design to meet top-level requirements, an aircraft design synthesis tool, called the Initiator, was built to enable the fast generation of aircraft baseline designs just starting from a set of Top-level aircraft requirements (TLARs) and the specification of the design's configuration (tube-and-wing, blended-wing body, box-wing, etc.) [91, 116, 117]. The Initiator is a MATLAB-based [S13], object-oriented framework that combines disciplinary modules to perform a step-wise design synthesis. Classical textbook sizing and weight estimation methods [118, 119] are used for the first iteration of the conceptual design loop. After the quasi-analytical conceptual design, specialized sizing modules can be used to further refine the design. For example, the fuselage can be designed in detail using the 'Fuselage Configurator' module, which performs an inside-out approach to determine an initial fuselage geometry. Similar preliminary sizing modules are available for the sizing of wings, landing gears, engines, etc. In addition, the Initiator includes analyses to check certification requirements based on Federal Aviation Regulations (FAR) parts 23 and 25. The process flow for converging a design with the Initiator is shown in Fig. 7.2

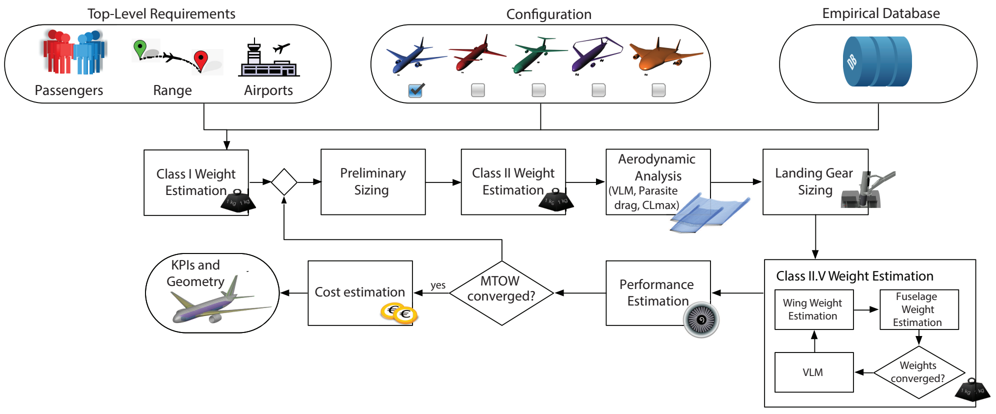


Figure 7.2: Flow diagram of the Initiator toolbox [91]

As its name suggests, the toolbox was developed to be able to quickly synthesize an initial aircraft design. The best design is selected by building a wing-thrust-loading diagram. A typical example of such a diagram is shown in Fig. 7.3. In this diagram, a set of design and certification requirements constrain a design space in which any point could be selected. The most lower right position is selected to obtain a design with a low thrust loading (low fuel consumption) and high wing loading (low drag at high speed).

The structure of the Initiator toolbox is shown as a UML diagram in Fig. 7.4a. All classes in the toolbox inherit from MATLAB's abstract base class `handle`. The design process is executed by an `InitiatorController` object. This object contains the logic to run a design synthesis convergence loop. The controller contains all analysis modules (see `Module`) and an `Aircraft` object, in which the geometry, analysis results, and requirements are stored. The aircraft object contains different `Part` definitions, such as wing, fuselage, engine, and landing gear. Each module gets its input data from the `Aircraft` object and writes its results to that object after execution.

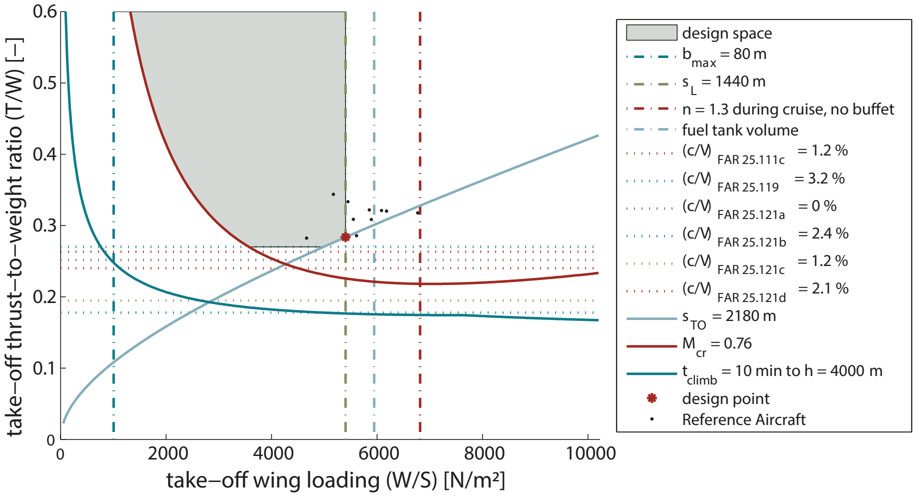


Figure 7.3: A typical take-off wing-thrust-loading diagram from the Initiator with the selected design point in red [91]

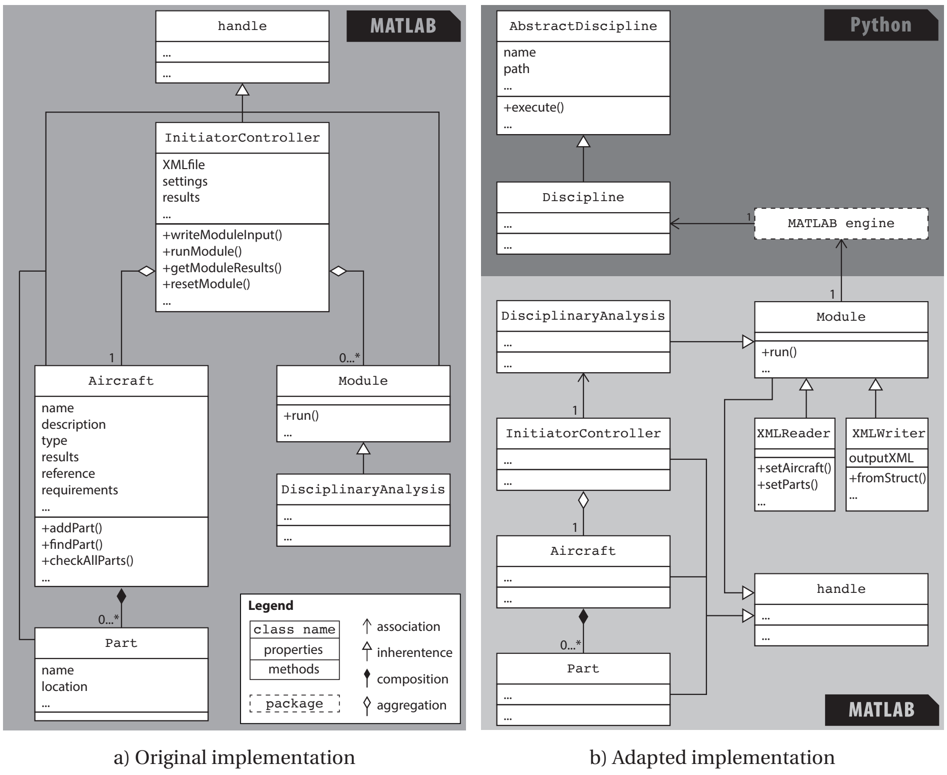


Figure 7.4: UML diagrams of the original Initiator implementation and the adapted implementation developed to establish the new framework.

Over the years, the Initiator has been used to investigate a variety of aircraft [120–122], meanwhile expanding the framework to include new analysis modules and aircraft configurations. Unfortunately, the framework's expansion was not always controlled well, leading to a collection of modules of varying code quality, some abusing the object-oriented set-up. For example, not all modules can be executed independently, as they contain nested dependencies on other analyses that will be called illegally: uncommanded by the controller. Such coding practices throughout the toolbox led to a state of *spaghetti code*, visualized on the left in Fig. 7.5, that is difficult to read, understand, and reconfigure. In addition, the controller object itself is geared towards design convergence and is laborious to reconfigure for other MDAO strategies (i.e. optimization), while the disciplinary modules of the Initiator offer tremendous potential to perform MDAO studies. Hence, a subset of the Initiator modules could be reused to solve another design problem. This situation is very similar to the collaborative MDAO context, since we have a lot of different disciplinary tools with a lot of direct links between them.

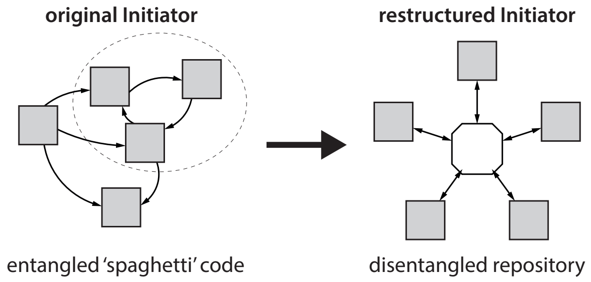


Figure 7.5: Conceptual visualization of the links between disciplinary modules in the current Initiator (left) and the restructured version (right) presented in this chapter

In this section a prototype of an adapted implementation of Initiator modules is presented. This reimplementation was developed in collaboration with Bruggeman [87] and performed to show how an existing multidisciplinary toolbox can benefit from the methodological and software developments proposed in this dissertation.

7.2. Configuration: establishing an agile Initiator

This section presents the first iteration of the development process in Fig. 7.1. In this iteration the agile system is configured and the restructured framework is verified against the original Initiator implementation.

7.2.1. Stage I: tool repository

Conceptually, the creation of the tool repository requires that all modules conform with the CDS (Central Data Schema) approach, discussed in §2.1 and shown in Fig. 2.1. Hence, all modules need to be standalone and are required to read and write their I/O data to a format that corresponds to a single schema. Practically, the following tasks were performed to achieve this:

- define an XML schema to store module I/O data;
- develop an XML reader and writer to wrap the modules around the schema and make them independent;

- select modules from the Initiator and define their I/O files to establish the tool repository.

Each topic will be briefly discussed in a separate section. An overview of the adapted implementation is depicted in the UML diagram in Fig. 7.4b.

XML SCHEMA

The original Initiator toolbox already includes a well-structured data storage implementation with the `InitiatorController` object. For this proof of concept, this controller's MATLAB-based data structure was taken as the basis for a new XML schema: IPACS (Initiator Parametric Aircraft Configuration Schema). The top-level elements of this schema are visualized in Fig. 7.6.

Most elements of IPACS match one-on-one with the controller's data structure. For example, the parts within the aircraft element contain the data for the definition of the `Part` object in Fig. 7.4. Two new elements are introduced that store additional data: the `constraints` element stores the lines in the wing-thrust-loading diagram (see Fig. 7.3) as constraint values for optimization purposes and, similarly, the `scaledVariables` element stores scaled variables that can be used as objective by an optimizer.

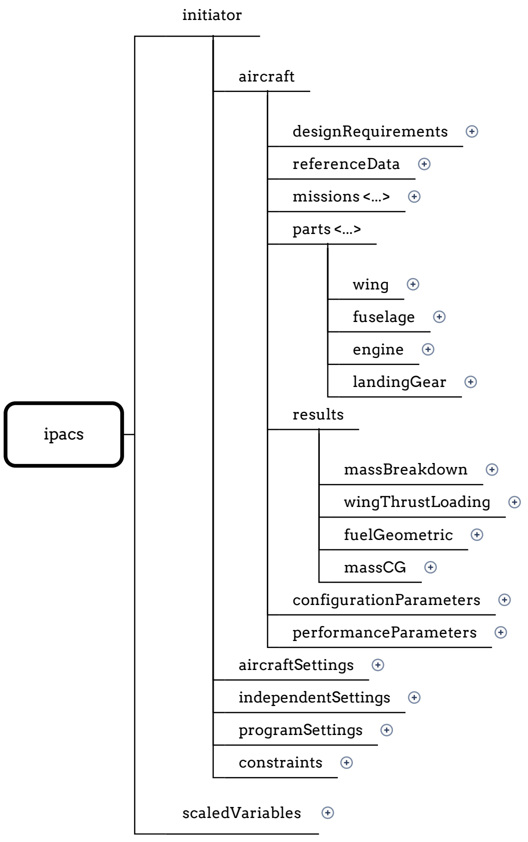


Figure 7.6: The IPACS (Initiator Parametric Aircraft Configuration Schema)

Instead of using the new IPACS, the Initiator modules could also be made compatible with the existing CPACS. This has the major advantage that the Initiator framework can be combined with other CPACS-compatible tools. Unfortunately, IPACS and CPACS were found to be so distinct that a major development effort would be required to make the Initiator's controller and disciplinary analyses compatible. This effort was considered out of scope for the current prototype. However, if all modules are made CPACS-compatible, then each module would be usable as a standalone tool in a CPACS-based framework. At least the complete Initiator toolbox should be made CPACS-compatible, which is the approach implemented in the current Initiator.

XML READER AND WRITER

The original toolbox already contained an XML reader and writer module, though developed for different purposes. These modules were adapted to match IPACS. Hence, the reader is able to load all data from an IPACS file as an `InitiatorController` object so that the `DisciplinaryAnalysis` can be run using that controller. After execution of the disciplinary analysis, the `XMLWriter` is used to export the controller object as an IPACS file. In addition to reconfiguring the modules to include the XML reader and writer, any dependencies on other module executions were replaced as input data to be read from the XML. This is a key feature, as it establishes the repository structure shown on the right of Fig. 7.5.

The reader and writer are the same for all disciplines. Hence, the modules are not optimized for the actual I/O data of a specific disciplinary analysis, but simply read the full IPACS file into the controller object and write the complete controller as an IPACS file at the end. This suboptimal data processing adds overhead to each discipline run, but was considered acceptable for a proof of concept that is not necessarily focused on achieving the shortest execution time.

MODULES AND TOOL REPOSITORY

From the large number of Initiator modules, seven were selected that permit the preliminary design of a tube-and-wing airliner:

Database: collects reference data on aircraft and engines. These data are used by other disciplines to empirically determine initial values for key parameters.

Empirical OEM: estimates the Operational Empty Mass (OEM) based on data from the Database module.

Class I Weight Estimation: estimates the mass breakdown, based on the OEM, using the classical fuel fractions method.

Fuselage Configurator: determines the entire fuselage layout calculating both the inner (cabin layout) and outer geometry.

Wing Thrust Loading: selects a design point based on requirements that constrain the design space, see Fig. 7.3.

Geometry Estimation: provides a geometry for the wing, engine and empennage.

Class 2 Weight Estimation: provides a more refined mass breakdown compared to the class 1 estimation, including component masses and centre of gravity locations.

With the ‘IPACSization’ of the modules completed, the first stage of the MDAO development process can be formulated as a graph using KADMOS: Repository Connectivity Graph (RCG).

The RCG is visualized in Fig. 7.7. The creation of this graph already shows how adopting the approach presented in this work benefits the user of the Initiator. Instead of having to comprehend a large, complex toolbox through MATLAB, the modules are systematically implemented as an MDAO system that can be inspected, debugged, and manipulated to formulate design problems. In addition, the tools on the diagonal are automatically put in a sequence with minimal feedback using the sequencing method described in §§3.5.3. The restructured framework implementation enforces a ‘separation of concerns’; the Initiator modules now act as independent tools and the controller object has been demoted to merely act as a data structure, leaving the formulation, inspection, and execution of workflows up to other specialized applications, such as KADMOS, VISTOMS and OpenMDAO.

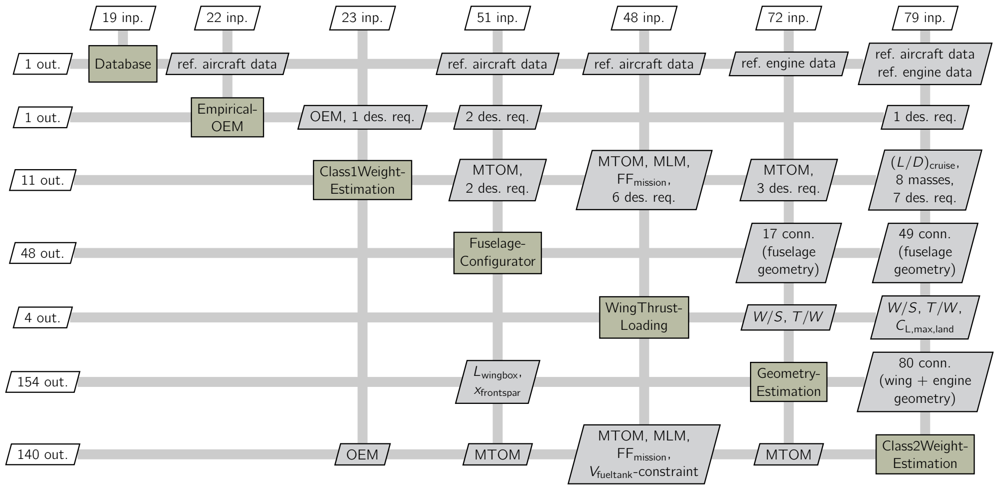


Figure 7.7: Repository overview of the updated Initiator framework showing the sequenced RCG created in KADMOS.

7.2.2. Stage II: MDAO problem

The reimplementation of the Initiator was verified and demonstrated by defining a problem that could also be solved by the original implementation: convergence of an aircraft design based on the TLARs of an Airbus A320-200 and a required range of 4000 km. In KADMOS, the MDAO problem to be solved was formulated such that all tools in the repository (Fig. 7.7) needed to be executed to retrieve the converged MTOM and OEM values. The formulated FPG is shown in Fig. 7.8.

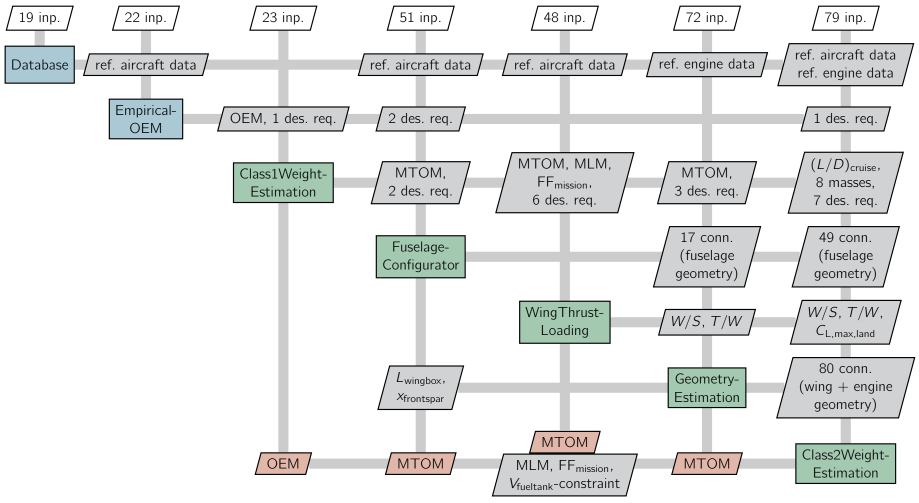


Figure 7.8: Fundamental Problem Graph (FPG) of the convergence problem with the quantities of interest highlighted in red.

7.2.3. Stage III: MDAO solution strategy

Three different convergence strategies were imposed on the FPG shown in Fig. 7.8:

1. Gauss-Seidel
2. Gauss-Seidel partitioned in two subsystems
3. Jacobi

Both Gauss-Seidel strategies are visualized with XDSMs in Fig. 7.9 and Fig. 7.10. The partitioned strategy makes use of the MDK decomposition method in KADMOS, which was discussed in §§3.5.4. The partitioning is used here to validate its use in a realistic design case. The partitioning method divides the coupled tools in two groups and makes a trade-off between balancing execution time and the number of broken couplings between the groups.

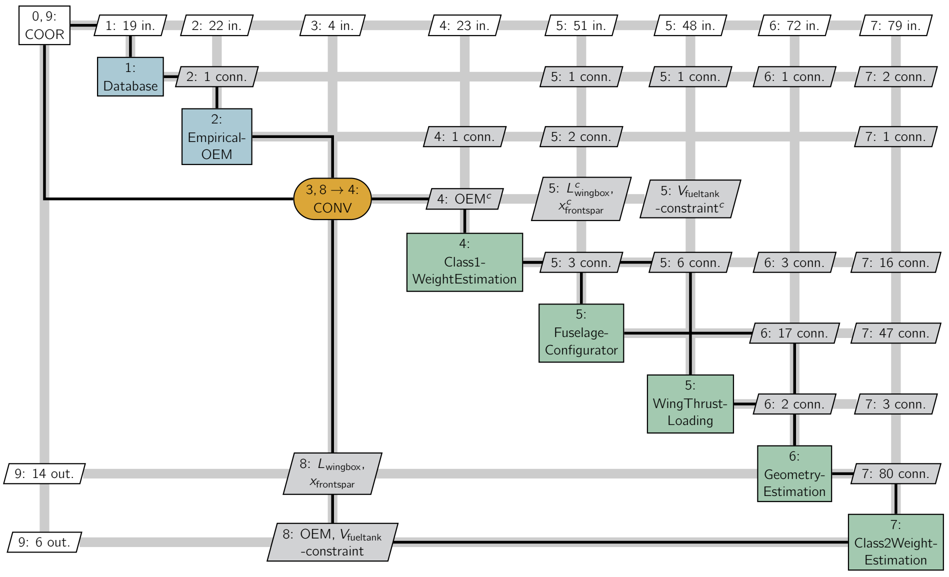


Figure 7.9: XDSM of the solution strategy based on the Gauss-Seidel convergence architecture, which has automatically been formulated by KADMOS.

7.2.4. Stages IV & V: materialization, execution, and results

The convergence strategy was also executed using the original Initiator toolbox in order to verify the reimplemented prototype. With the new framework, the formulated strategies are stored as CMDOWS files and the matching workflows materialized in OpenMDAO with OpenLEGO (discussed in §5.4).

The execution of the framework with OpenMDAO, requires that each disciplinary analysis is integrated as a Python-based `AbstractDiscipline`. Fig. 7.4b visualizes how the Initiator MATLAB modules are wrapped as Python objects. The key component for this wrapper is the `MATLAB engine` package. This package enables the execution of MATLAB files from Python. Each discipline runs on its own engine. The use of these engines introduces overhead that can only be avoided by rewriting the Initiator modules in Python.

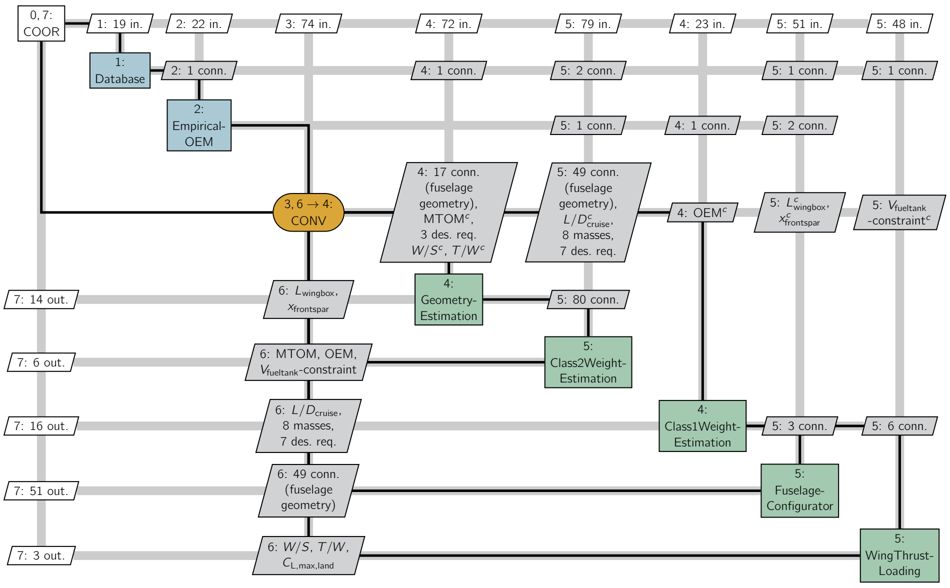

0, 7: COOR
1: 19 in.
2: 22 in.
3: 74 in.
4: 72 in.
5: 79 in.
4: 23 in.
5: 51 in.
5: 48 in.
1: Database
2: 1 conn.
4: 1 conn.
5: 2 conn.
5: 1 conn.
5: 1 conn.
2: Empirical-OEM
5: 1 conn.
4: 1 conn.
5: 2 conn.
4: 17 conn. (fuselage geometry), MTOM^c, 3 des. req. W/S^c, T/W^c
5: 49 conn. (fuselage geometry), L/D^c_{cruise}, 8 masses, 7 des. req.
3, 6 → 4: CONV
4: OEM^c
5: L^c_{wingbox}, $x^c_{\text{frontspar}}$
5: V_{fueltank} -constraint^c
7: 14 out.
6: L_{wingbox}, $x_{\text{frontspar}}$
4: Geometry-Estimation
5: 80 conn.
7: 6 out.
6: MTOM, OEM, V_{fueltank}-constraint
5: Class2Weight-Estimation
7: 16 out.
6: L/D_{cruise}, 8 masses, 7 des. req.
4: Class1Weight-Estimation
5: 3 conn.
5: 6 conn.
7: 51 out.
6: 49 conn. (fuselage geometry)
5: Fuselage-Configurator
7: 3 out.
6: W/S, T/W, $C_{L,\text{max,land}}$
5: WingThrust-Loading

Figure 7.10: XDSM of the solution strategy based on the Gauss-Seidel in two partitions architecture, which has automatically been formulated by KADMOS.

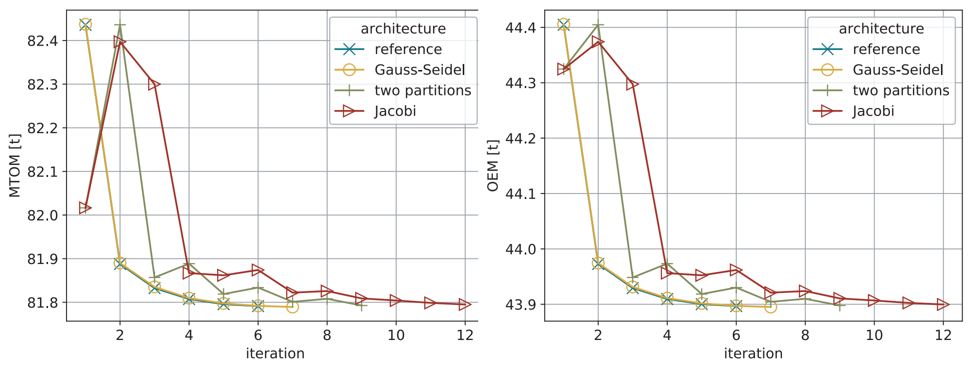
The results for the convergence studies are summarized in Tab. 7.1. The reference strategy on the first row refers to the run performed in the original Initiator. Clearly, all three formulated strategies result in the same values for MTOM and OEM with negligible errors.

Table 7.1: Results of the convergence studies to validate the reimplementation of the Initiator

strategy	iters	MTOM [t] (error ‱)	OEM [t] (error ‱)	execution time [s]	
				actual (1 CPU)	estimated (#CPU)
reference	6	81.791	43.897	105	-
Gauss-Seidel	7	81.789 (-0.24)	43.895 (-0.46)	313	298 (1) / 258 (2)
two partitions	9	81.793 (+0.24)	43.898 (+0.23)	620	214 (2)
Jacobi	12	81.795 (+0.49)	43.900 (+0.68)	735	144 (5)

Fig. 7.11 depicts the convergence history for both variables. Looking at the overlap between the reference and Gauss-Seidel run, it is clear that the original Initiator controller performs convergence based on the same scheme. The number of iterations differ due to a subtle difference in the executions: the original implementation converges the MTOM, while the KADMOS implementation works with the OEM. In addition, the original implementation actually only converges based on the MTOM, while the workflow in OpenMDAO checks convergence for all feedback variables.

In the last columns of Tab. 7.1 execution times are listed. The prototype framework introduces overhead at several locations, as was pointed out earlier (e.g. XML reading/writing, MATLAB engine). Consequently, the execution time with the framework is much higher



a) MTOM

b) OEM

Figure 7.11: Convergence history for the four different strategies list in Tab. 7.1

than the original implementation, see the actual execution time column. However, the agile framework makes it easy to also use alternative convergence schemes that would benefit from the use of a multiprocessing setup. The last column lists the estimated execution times when multiple processing units would be used. Unfortunately, the current prototype does not support multiprocessing in the execution phase due to file handling issues in OpenMDAO; the package does support multicore processing, but was not built to include file reading and writing when multicore processing is switched on. However, the potential of adapting the convergence strategy to a multicore processing environment is clear from the estimated execution times.

In summary, the configuration of the Initiator toolbox into a novel framework separates disciplinary analyses from system formulation and execution. This new setup was verified successfully based on a design convergence comparison. Furthermore, the framework provides the agility to formulate alternative convergence schemes that are not supported by the original Initiator with the potential of taking advantage of multicore processing. With the agile framework properly configured, it can now be used to reconfigure the MDAO system by starting a new iteration of the MDAO development process in Fig. 7.1.

7.3. Reconfiguration: optimization

With the validated design framework in place, it can now be reconfigured to answer new design questions. Design problems can now be solved using strategies that were not available in the original Initiator implementation. For example, instead of performing a convergence study, an optimization can also be formulated and executed. A design question to be answered can be:

> What is the difference between minimizing take-off mass and fuel mass for the same range, design variables, and constraints?

Hence, the following two optimization problems are defined:

minimize:	MTOM @ R=4000 km	(take-off mass at given range)
or:	FM @ R=4000 km	(fuel mass at given range)
with respect to:	$2000.0 \leq W/S \leq 7000.0$	(wing loading in N/m^2)
	$0.0 \leq T/W \leq 0.6$	(thrust loading)
	$6.0 \leq \mathrm{AR} \leq 13.0$	(aspect ratio)
subjected to:	$s_\mathrm{L} \leq 1480$ m	(max. landing distance)
	$s_\mathrm{TO} \leq 1938$ m	(max. take-off distance)
	no buffet @$n_\mathrm{cr} = 1.3$	(load factor in cruise)
	$M_\mathrm{cr} \geq 0.78$	(cruise Mach number)
	$V_\mathrm{FT} \geq V_\mathrm{fuel}$	(fuel tank volume)
	$t_\mathrm{climb} \leq 10$ min	(climb time to 4000 m)
	$(c/V)_\text{FAR-25}$ (6x)	(six climb reqs, see legend Fig. 7.3)

The goal of the optimization is to select the best point in the thrust-wing-loading diagram and find the best aspect ratio for the main wing.

7.3.1. Stage I: tool repository

This second iteration of the development process starts at the first stage again. Instead of using the WingThrustLoading tool to do a prefixed selection of the design point, as was done in the convergence studies (§7.2), the optimization strategy will require a constraint calculation module and will leave the selection of the design point to the optimization algorithm. In total, four tools have to be added to the repository:

- constraint value calculation
- normalized objective value calculation for MTOM
- normalized objective value calculation for fuel mass
- aerodynamics module to estimate induced drag

Here, the power of the current framework in supporting a heterogeneous tool repository comes forward, as these tools are implemented in three different ways. The constraint values are calculated with a new MATLAB module. In this module, the constraining lines from Fig. 7.3 are reformulated as equations where a value larger than zero represents a constraint violation. The normalized objective values are implemented directly in KADMOS as mathematical relations that scale the actual values by a constant scalar. Finally, the aerodynamics module is implemented as a Python tool. This module is included to get a more realistic lift-drag polar estimation that is sensitive to a varying aspect ratio, based on:

$$C_D = C_{D_0} + \frac{C_L^2}{\pi\,\mathrm{AR}\,e}$$

The open-source Python wrapper [S29] for the AVL (Athena Vortex Lattice) [S30] package is used here for the drag estimation. Only induced drag is calculated by AVL, while parasite and wave drag are kept constant at 200 drag counts in the module. This (over)-simplification of keeping the other drag components constant could be avoided by adding additional drag estimation modules to the framework. However, this was considered out of scope for the current prototype, as the addition of the four aforementioned modules already demonstrates the extensibility of the current framework.

After adding the XML I/O files of the new tools to the database, KADMOS can be rerun to update the RCG. The extension of the RCG is visualized in Fig. 7.12.

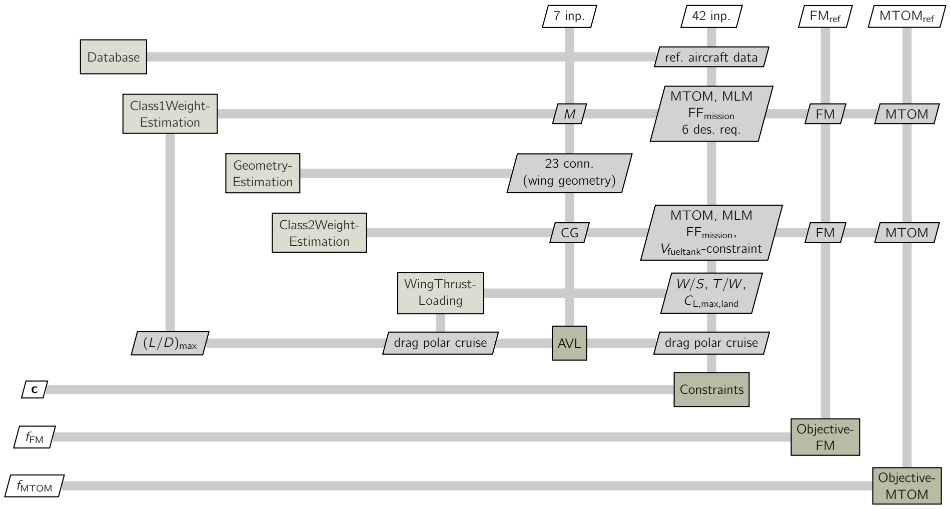


Figure 7.12: Framework repository extension with four new tools and their couplings with existing tools from the repository overview in Fig. 7.7.

7.3.2. Stage II: MDAO problem

The RCG with the full repository is used to formulate the FPG in KADMOS matching the optimization problem provided at the beginning of this section. This FPG is shown in Fig. 7.13.

7.3.3. Stage III: MDAO solution strategy

Using the FPG from Fig. 7.13, KADMOS automatically imposes an optimization strategy on the problem. Fig. 7.14 depicts the MDF architecture with a Gauss-Seidel convergence scheme approach for the MTOM minimization problem. The fuel mass minimization problem would result in a similar XDSM as Fig. 7.14, except for using the Objective-FM mathematical relation from Fig. 7.12 instead of Objective-MTOM.

7.3.4. Stages IV & V: materialization, execution, and results

Once the strategy has been determined, it is stored as a CMDOWS file and materialized in OpenMDAO for execution. The optimization history and results for the MTOM-minimization case are summarized in Fig. 7.15, Fig. 7.16, and Tab. 7.2. The optimizer also selects the lower right corner in the thrust-wing-loading diagram, see Fig. 7.15e. The two active constraints are related to the take-off and landing distance. The final wing planforms for both minimization cases are shown in Fig. 7.16. The minimization

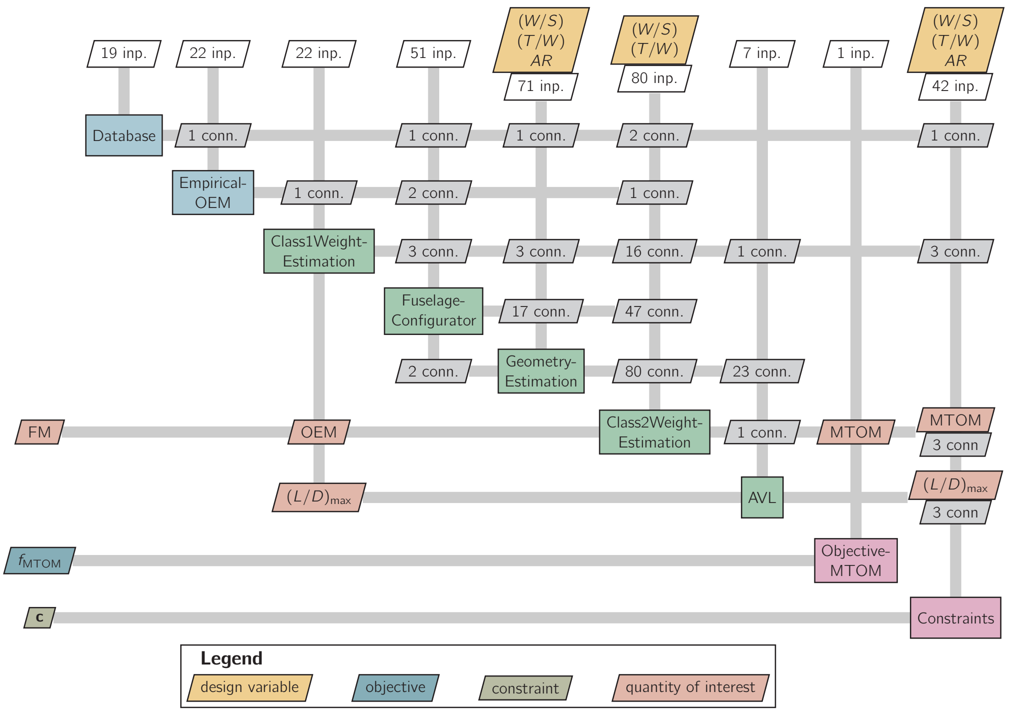


Figure 7.13: FPG of the optimization problem with the quantities of interest highlighted in red.

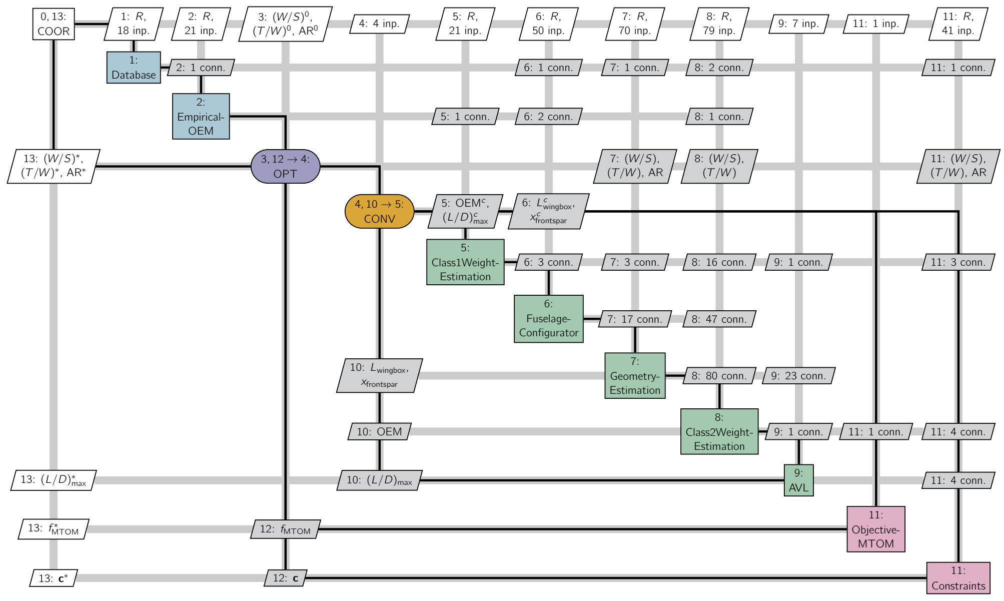


Figure 7.14: Automatically instantiated XDSM from KADMOS of the MDF optimization strategy for the MTOM minimization problem.

for fuel mass leads to a higher aspect ratio wing of 13.0 (instead of 11.9 for MTOM minimization), which is actually at the variable's upper bound.

Table 7.2: Results of the optimizations for minimal MTOM or minimal FM

parameter	unit	min. MTOM	min. FM
AR	[-]	11.89	13.00
(W/S)	[N/m^2]	5130	5131
(T/W)	[-]	0.258	0.25
MTOM	[kg]	81521	81627
FM	[kg]	16368	16015
OEM	[kg]	44617	45076
$(L/D)_{\text{max}}$	[-]	19.20	19.70

7.4. Second reconfiguration: additional constraint

Both wing planforms in Fig. 7.16 would prevent the 'optimized' A320 to use the same airport gates, for which there is a span limitation of 36 meters. Hence, the optimization needs to be reconfigured and an additional constraint is required to limit the wingspan. This is the third and final iteration of the development process in Fig. 7.1 discussed in this chapter. With a previous optimization under the belt, this reconfiguration can be performed very quickly by:

stage I: Tool repository: updating the repository by adding the wingspan constraint to the Constraints module;

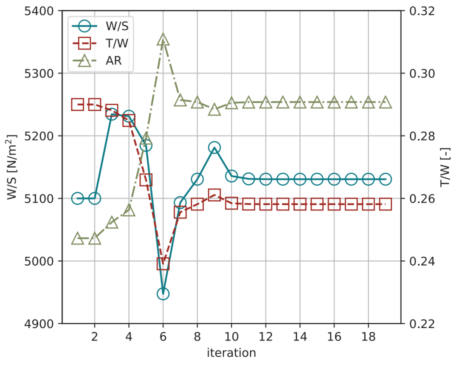
stage II: MDAO problem: updating the KADMOS script to include the additional constraint in the FPG;

stage III: MDAO solution strategy: rerunning the KADMOS script to obtain the CMDOWS file;

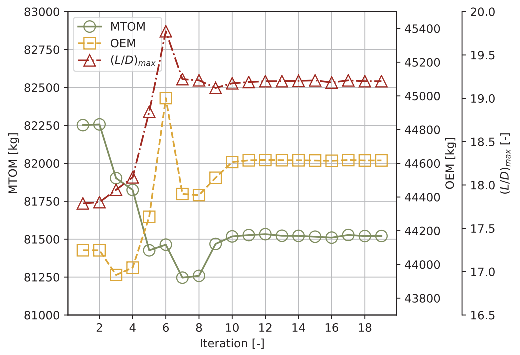
stage IV: materialization and execution: rerunning the OpenLEGO script to materialize a new OpenMDAO workflow and execute it.

These reconfiguration steps can be performed within a coupled of minutes, while with the original Initiator toolbox this type of reconfiguration would be much more cumbersome, requiring the adjustment of multiple MATLAB modules to pass the right data to a new wing span constraint calculation. The results for the MTOM-minimization case results in the new planform geometry and loading diagram shown in Fig. 7.17. The wingspan of the main wing is lowered, see Fig. 7.17a, with the optimal wing having a lower aspect ratio of 8.43 (instead of 11.9). The additional constraint affects the MTOM negatively, with an increase of 1.3 t to 82.8 t. The loading diagram in Fig. 7.17b shows how the additional constraint reduced the design space to a single line, where three constraints meet: maximum wingspan (b_{max}), maximum landing (s_{L}) and maximum take-off distance (s_{TO}). Comparing the loading diagrams in Fig. 7.15e and Fig. 7.17b and considering the indicated reference aircraft (black dots), the wingspan constraint positions the final design closer to existing airliners from the Initiator's Database module, as is to be expected.

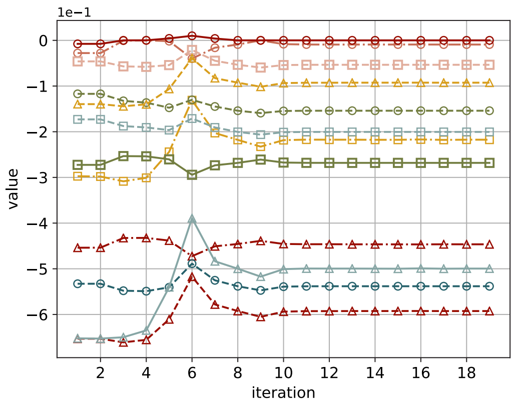
In conclusion, this chapter has presented the restructuring of an existing aircraft design toolkit to establish a more scalable and flexible framework and introduce MDAO capabilities. This chapter has also demonstrated and validated the generic applicability of



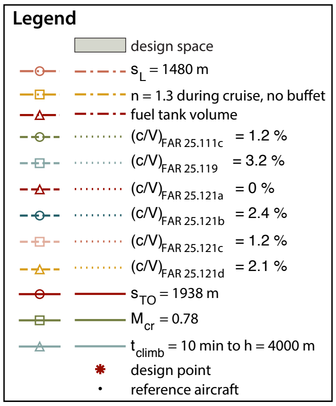
a) design variables (min. MTOM)



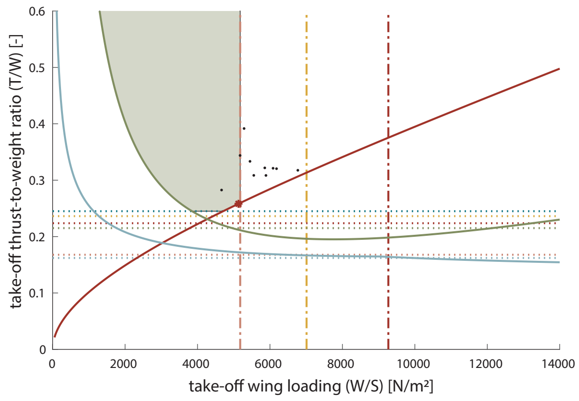
b) objective and other QOIs (min. MTOM)



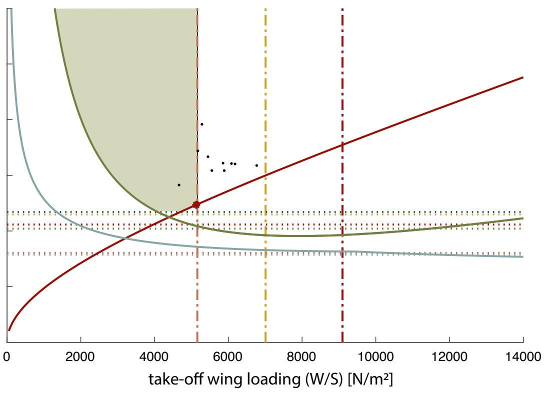
c) constraints (min. MTOM)



d) legend constraints and diagram



e) thrust-wing-loading diagram (min. MTOM)



f) thrust-wing-loading diagram (min. FM)

Figure 7.15: Optimization history and results for the MTOM-minimization and FM-minimization cases

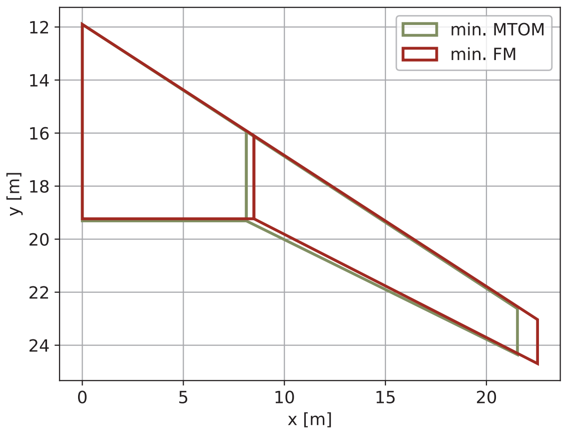
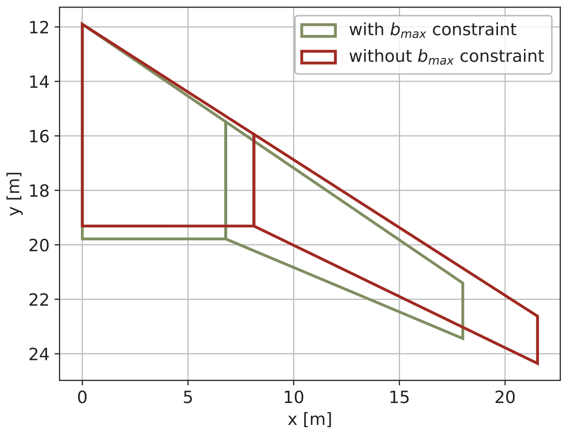
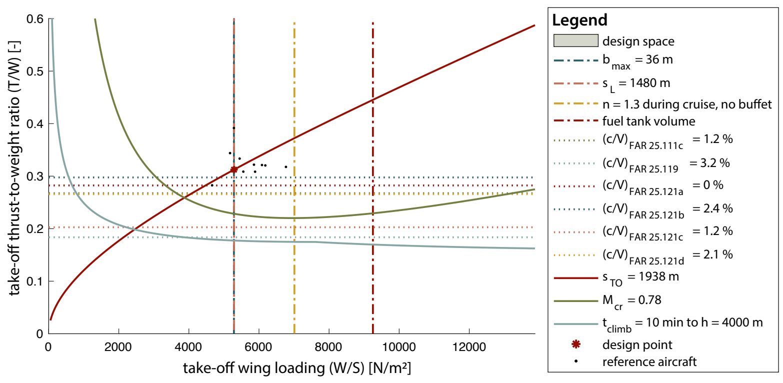


Figure 7.16: wing planform comparison



a) wing planform comparison



b) thrust-wing-loading diagram

Figure 7.17: Results for the MTOM-minimization case including a constraint on the maximum wingspan

the proposed methodological approach, since the Initiator toolkit is completely outside the boundaries of the AGILE project, which originally provided the context for the developments original to this dissertation.

IV

Outlook & conclusion

8 OUTLOOK: COMPOSITE ARCHITECTURES AND MDAO BOTS

IN Parts I to III the major technological content of this dissertation has been presented. This chapter sets the stage for future research by proposing two new concepts: *composite architectures* and *MDAO bots*. The former is discussed in §8.1 with a small illustrative example based on the aircraft design framework covered in Chapter 7. The latter concept, which is presented in §8.2, looks at this dissertation's development from a different angle to provide a conceptual basis for future work.

8.1. COMPOSITE ARCHITECTURES

The optimization runs in the previous chapter demonstrated how a restructured Initiator framework exploits the novel developments presented in this dissertation. The aircraft design toolbox was reimplemented as an MDAO system, so that KADMOS can be used to formulate a variety of strategies to answer design questions, which are subsequently materialized and executed automatically in OpenMDAO. The questions that were answered with the available strategies were:

- What would be a synthesized aircraft for given inputs?
- What would be the best aircraft for a given objective, within certain input bounds (design variables), and meeting certain constraints?

The first question was answered by the convergence studies and the second through the optimization strategies. Another typical question of a design team could be: *How is the optimal design varying if the top-level requirements are changed?* The answer to this question would require a more complex workflow that would vary one (or multiple) of the requirement values within a given interval and optimize the design for each value. Hence, a *composite DOE-optimization workflow* is required to answer this question.

This composite strategy is illustrated here with a small example. Suppose the design team would like to vary the range requirement for the airliner discussed in Chapter 7. The team wants to assess how a change in this requirement would affect the optimal aspect ratio and MTOM of this configuration. Such information could be valuable in the decision-making process on setting the right top-level requirements for a new aircraft development program. Hence, the design question to be answered is:

What is the sensitivity of the tube-and-wing configuration's aspect ratio and MTOM optima to the range requirement?

Translating this question to a problem formulation yields:

minimize:	MTOM	(max. take-off mass)
for:	$R = \{3.0, 3.5, 4.0, 4.5, 5.0\} \cdot 10^3$ km	(range requirement interval)
with respect to:	$2000.0 \leq W/S \leq 7000.0$	(wing loading in N/m^2)
	$0.0 \leq T/W \leq 0.6$	(thrust loading)
	$6.0 \leq \text{AR} \leq 13.0$	(aspect ratio)
subjected to:	$b \leq 36$ m	(max. wingspan)
	$s_\text{L} \leq 1480$ m	(max. landing distance)
	$s_\text{TO} \leq 1938$ m	(max. take-off distance)
	no buffet @$n_\text{cr} = 1.3$	(load factor in cruise)
	$M_\text{cr} \geq 0.78$	(cruise Mach number)
	$V_\text{FT} \geq V_\text{fuel}$	(fuel tank volume)
	$t_\text{climb} \leq 10$ min	(climb time to 4000 m)
	$(c/V)_\text{FAR-25}$ (6x)	(six climb reqs, see legend Fig. 7.3)

The translation of this MDAO problem to a solution strategy has not been automated in KADMOS yet. Such automation would provide the — now manually defined — strategy shown in Fig. 8.1. The strategy contains a DOE block to pass different range values that need to be analyzed, after which the same optimization from Fig. 7.14 is performed for each range value.

This small illustrative example has been built manually using an OpenLEGO script. Fig. 8.2 plots the results of the sensitivity analysis. With increasing range, more fuel needs to be stored in the wing and due to the maximum wingspan constraint, the aspect ratio therefore decreases. As expected, both the Fuel Mass (FM) and the OEM increase with increasing range, but at different slopes. This type of information is useful to set a good starting point for multi-point optimization, where an aircraft is optimized for multiple missions. The example provided here represents a small proof of concept that can easily be built manually as well. However, a requirements sensitivity analysis scales up quickly in a realistic design situation: soon a large set of requirements needs to be varied to assess the configuration's sensitivity. The DOE block would then not provide all possible requirement combinations, but will act as a sampling block to provide a limited set of experiments that covers the requirements space well.

In the everyday design practice, every new design question might require the inclusion of a new strategy to support the automated MDAO development process from Fig. 7.1. This was also one of the feedbacks from the AGILE users of KADMOS (discussed in §§6.4.2): once they recognized the potential of automatically imposing architectures on large, complex MDAO systems, they soon wanted to define and use their own architectures

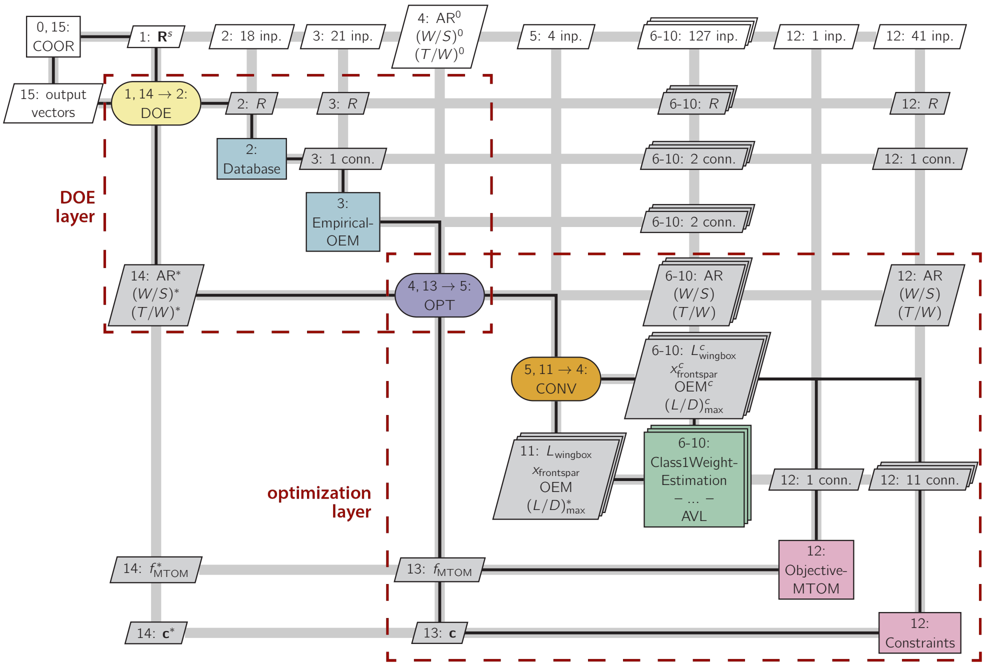


Figure 8.1: Manually made XDSM for a composite strategy combining a DOE and optimization layer to investigate the sensitivity of an optimized design with respect to one of its requirements.

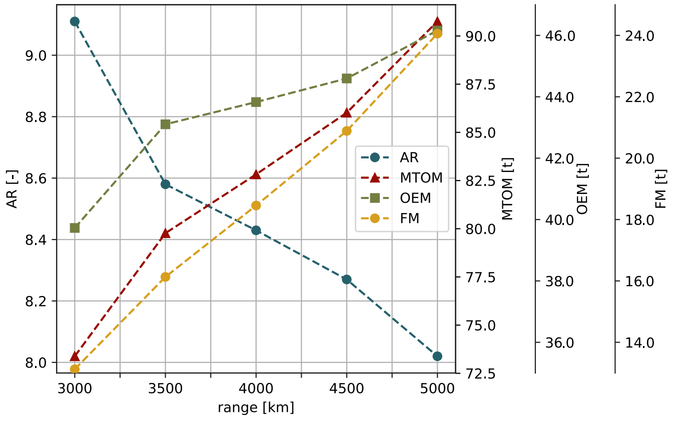


Figure 8.2: Plot of the sensitivity of the aspect ratio, MTOM, OEM, and FM to a change in the required mission range.

that could handle the peculiarities of their specific MDAO problem. In many cases, these architectures required a mix of DOE and optimization blocks, as in the small example just described.

Another possible composite architecture can be defined based on the approach suggested by Piperni et al. [11], which was already discussed in Chapter 1. Piperni et al. describe the step-wise approach adopted at Bombardier to perform preliminary aircraft design using their MDAO framework. This approach contains the following four steps:

1. Perform a DOE to filter out less significant design variables.
2. Create approximation model(s) based on the DOEs of previous step.
3. Perform a surrogate-based optimization using the approximation methods.
4. Perform a full optimization using exact models, starting from the optimum solution of the previous step and only using the most significant variables.

While each of these steps are already automated individually by the developments presented in this dissertation, the possibility to include all four steps in one composite architecture is not that far away. This would entail a complex architecture that contains one DOE layer, two optimization layers, and the capability to build surrogate models.

The development of such composite architectures is a matter of extending the framework applications:

- KADMOS needs to support the formulation of the design question as an MDAO problem in the form of a graph (FPG).
- KADMOS needs to be able to impose the composite solution strategy on the problem to obtain the data and process graphs (MDG and MPG).
- CMDOWS should support the storage of these strategies.
- OpenLEGO should support the instantiation of the OpenMDAO workflow to be executed.

The required effort for these four extensions depends on the complexity of the composite architecture being implemented. For example, the development effort for the strategy in Fig. 8.1 is relatively small, as it resembles the disciplinary DOE subsystem that is part of the BLISS-2000 strategy shown in Fig. 5.11. In KADMOS, one would only have to include a new node attribute to distinguish between design variables for the DOE and the optimization. The imposition algorithm for the solution strategy can be written completely based on existing methods in the KADMOS package. CMDOWS and OpenLEGO already support the storage and materialization of this composite strategy as it resembles the BLISS-2000 strategy, though it differs in some of the details and needs to be checked, debugged, and tested.

The strategy based on the Bombardier approach will entail more cumbersome developments for the framework applications. In KADMOS, multiple new attributes might be required to represent the intended question of the design team as a graph and the complex composite architecture will require new methods to include the creation of surrogate models and the automatic of the most significant variables. These new aspects of the solution strategy also need to be stored properly in CMDOWS, requiring extensions of the schema. Finally, OpenLEGO needs to be extended to also correctly materialize the new elements present in the CMDOWS file.

The power of the developed framework applications in this dissertation is that, once a new strategy has been included to answer a specific design question, then the formulation, materialization, and execution of the workflow is completely automated. Hence, each new composite architecture increases the achieved level of automation of the MDAO development process. Instead of using the small list of classical architectures currently included in KADMOS, future work should focus on extending the graph-based methodology to enable the flexible definition and imposition of composite architectures (and also other existing MDO architecture, such as ECO and analytical target cascading [14]). Ultimately, these extensions will lead to an *architecture factory* environment. This factory would be a graphical, interactive user interface for integrators to create composite architectures by dragging, dropping, and merging graph nodes and methods on a drawing board showing the MDAO system under consideration.

8.2. MDAO BOTS

The semi-automated approach established so far in this disseration is summarized in the top of Fig. 8.3. The main tasks of the involved agents, which were introduced in §§6.1.1 and Fig. 6.3, are also shown in the figure as the used technologies.

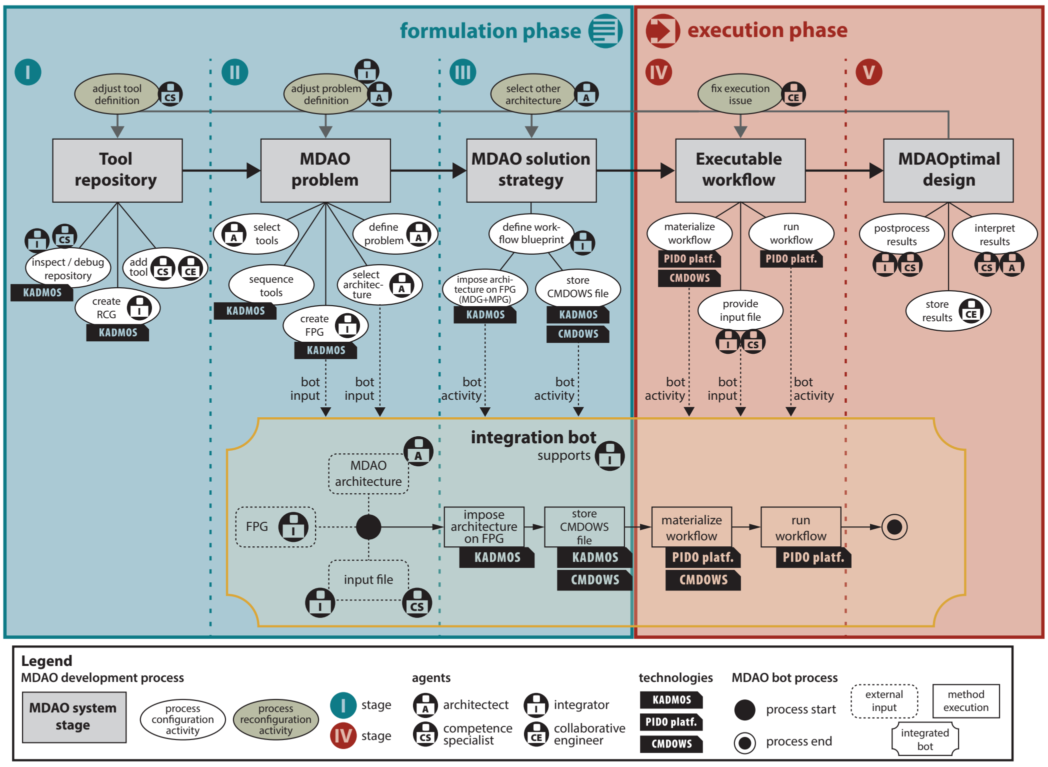


Figure 8.3: Relation of the MDAO bots proposed in this section — including the definition of the integration bot — with the methodology and developments presented in this dissertation.

In the established methodology, system reconfiguration is always assumed to be carried out by one of the agents using their own judgment to adjust the relevant system stage, see the green ellipses in Fig. 8.3. Although the presence of the human in the loop is considered absolutely valuable and should be maintained, also in view of increasing the acceptance of MDAO in industry, the developed methodology does offer further automation opportunities that cannot be ignored. These extra automation opportunities concern rule-based, repetitive and tedious activities that are associated to the core (judging-deciding) tasks of the human expert. The development process could be further robotized by supporting *automated reconfigurations* (thus not automated decision-making).

The automatic reconfiguration of the MDAO system would require the computer to interpret the results of previous workflow executions. This interpretation is driven by the main design question posed by the human agent and performed based on rationale explicitly defined by the expert. If such a process is implemented, the computer seems to be able to answer a complex question autonomously. Such interactive computerized automation is called a bot, which is defined as:

> “an autonomous program on a network (especially the Internet) which can interact with systems or users, especially one designed to behave like a player in some video games.”*

Well-known examples of bots are game bots, chat bots and mail bots. Here, the bot will be interacting with the agents of the design team and the MDAO system. Therefore, this concept is coined the ‘MDAO bot’. Instead of playing a video game, an MDAO bot would play an ‘MDAO game’. In line with above general definition, the MDAO bot is defined as:

> an autonomous program on a network which can interact with an MDAO system and design team agents, to behave like a human agent that answers a design question by (re)configuring and executing the MDAO system.

Note that these bots are not meant to replace the human agents and their irreplaceable judgment, knowledge, and creativity, but rather are meant as ‘robot assistants’ to unburden the agent of repetitive, error-prone, and often boring tasks. The combination of technologies presented in this dissertation (i.e. KADMOS, CMDOWS, workflow materialization in multiple PIDO platforms), already constitutes a first MDAO bot, albeit one that does not autonomously reconfigure the system. This bot covers the process from the imposition of the architecture on the FPG with KADMOS up to and including workflow execution in a PIDO platform of choice. The main agent assisted by this bot is the integrator, as the bot automates tasks that were originally performed manually by him or her. Hence, this bot could be called the ’integration bot’ and is shown in the lower part of Fig. 8.3.

In the next section, several bots of increasing complexity are proposed to illustrate the idea and explicate how the ‘MDAO bot’ concept can serve as a road map for future automation developments.

8.2.1. Conceptual definition of several MDAO bots

In this section the preliminary definition of four MDAO bots is presented.

Diagnostic bot

A design question can be posed in many different ways. For example, a simple question would be:

> *Can we run [a list of tools] based on this [input file]?*

*definition from: `https://en.oxforddictionaries.com/definition/bot`, accessed May 25 2019

This question requires the bot to perform a diagnostic run on a set of tools. Already with such a straightforward task the bot will help the integrator, competence specialists, and collaborative engineer in debugging the set of tools and the input file. The process performed by the bot is illustrated in Fig. 8.4:

1. The bot needs certain information from different agents to answer the question. In this case a list of tools from the repository is provided, an input file based on the adopted data schema (i.e. CPACS) containing parameter values, as well as the design question.
2. Using the list of tools, the bot can formulate the FPG using KADMOS. This FPG puts the tools in a convenient order using KADMOS's sequencing algorithm.
3. The architecture to be imposed is a 'test run': all tools need to be executed in sequence, without involving any iterative components such as optimizers or solvers.
4. With the architecture imposed, the input file should be checked first before moving on to the execution phase. The bot checks the following:
    a. Are all inputs required to run the tools present in the input file? If not, exit process and request integrator to include missing inputs.
    b. If expected type (e.g. float, vector, string) of the inputs is known, then check it against the values found in the input file. If type and value don't match, exit the process and request the competence specialist to update the problematic input file or the expected type by the tool.
    c. Will all values in the input file be used? If not, do not exit the process (or do, this could be a bot setting), but warn the integrator about redundant inputs.
5. If the bot passes the input checks, then the workflow can be materialized in the preferred PIDO platform.

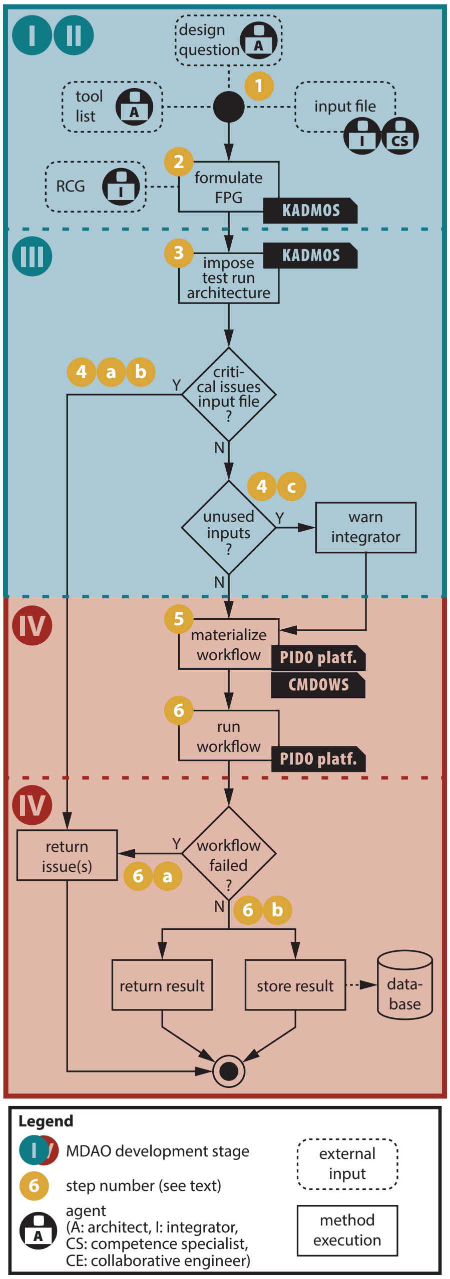


Figure 8.4: Process implementation for the diagnostic bot

6. After materialization the workflow is executed, which can have two possible outcomes:
    a. The execution failed. The bot returns the cause of failure and requests the user to fix it.
    b. The execution is successful. The bot returns the results in an output file and stores diagnostic information (e.g. execution times, derivative values) on tool and workflow execution in a database.

This basic bot illustrates the ability to mimic behavior of the integrator and collaborative engineer by automating a repetitive and error-prone task: checking the execution of a set of tools and also check the available input file. If all is well, then the team can continue, if not, then the bot requests input from the relevant agent (e.g. the integrator for integration issues, the collaborative engineer for technical issues).

This 'diagnostic bot' does not reconfigure the system after execution. However, the bot could easily be extended to perform more complex tasks that do require such reconfigurations. For example, common workflow execution errors might be anticipated and attempts to fix these errors could be included. This means that the bot will first run an automated debugging process before requesting help from the agent. The bot will then have to rematerialize the workflow and rerun it after applying different settings. Hence, workflow issues are solved automatically without interference from the human agent. In addition, the bot serves as a central knowledge base where these common issues and their fixes are stored for future use by other bots and agents.

Instead of adjusting workflow settings (e.g. server settings for web services) to fix errors, the bot could also change settings to check different computational setups. This type of diagnosis can be used to find the best settings for future workflows. For example, if a multicore processor is available, the bot could check workflow execution based on single and multicore processing setups. The multi-core solution strategies are then automatically determined using the partitioning algorithm in KADMOS. The architect's question answered by this bot is then:

> *What would be the best setup if we can choose between [a list of options] to run [a list of tools] based on this [input file]?*

Another extension of the diagnostic bot could be achieved by including a design space specification. If key variables of the system would not just have an single input value defined in the input file, but rather a range or list of values, then the system can be diagnosed throughout the design space. The question answered here is:

> *Can we run [a list of tools] based on this [input file] within the following [design space]?*

This 'design space diagnostic' could find issues with tools, if certain combinations of inputs cause problems. For example, if an aerodynamic tool includes a viscous airfoil analysis, then this tool will run without issues at low angles of attack, but at higher angles it will not converge and cause errors.* The diagnostic bot thereby assists the design team in identifying tool weaknesses within the design space that would cause critical issues in subsequent, more expensive, workflows, such as an optimization run. Diagnosing a full

*N.B. Alternatively, the tool provider could define input limits for all the tool's inputs and then this particular issue could be found before running the tool itself. However, if this input is output from another tool, then it is not know beforehand if this issue will be relevant, so at least the tools preceding this problematic tool have to be executed.

design space will require the inclusion of a sampling algorithm that determines at which points in space the tools are diagnosed. The diagnostic run is then performed for each of these points.

Synthesis bot

A logical next step after a successful diagnosis of the system, would be to synthesize the system, answering the following question:

> *What are the values for these [QOIs] based on this [input file]?*

The synthesis bot will not only run the tools using the diagnostic bot, but will also converge (or synthesize) the system if it contains cyclic dependencies. The automated process executed by the bot is depicted in Fig. 8.5:

1. The bot determines which tools from the tool repository are required to get the values of the QOIs, based on the available values provided in the input file. This task is determined using the graph-based, path-finding algorithm in KADMOS.
2. Based on the tool selection the diagnostic bot is used to perform a safe first run of the tools. If this run unveils issues, then the synthesis bot is stopped prematurely.
3. If the system does not contain any cycles (no convergence is needed at all) or the input file already specifies a converged system, then the synthesis bot is done.
4. If the system still needs to be converged, a convergence scheme is selected (e.g. Gauss-Seidel, Jacobi) and the FPG is formulated with KADMOS. This block is the first decision algorithm, indicated by a hexagonal block in the bot process diagram.
5. Using the integration bot, the solution strategy is obtained by imposing the convergence architecture on the FPG, the workflow is materialized, and executed in the PIDO platform of choice.
6. The execution will have two possible outcomes:
    a. Execution failed: synthesis bot returns issue and requests user to fix it.
    b. Execution successful: synthesis bot returns the results in an output file and stores results and profiling data (e.g. execution times, derivative values) on workflow execution in a database.

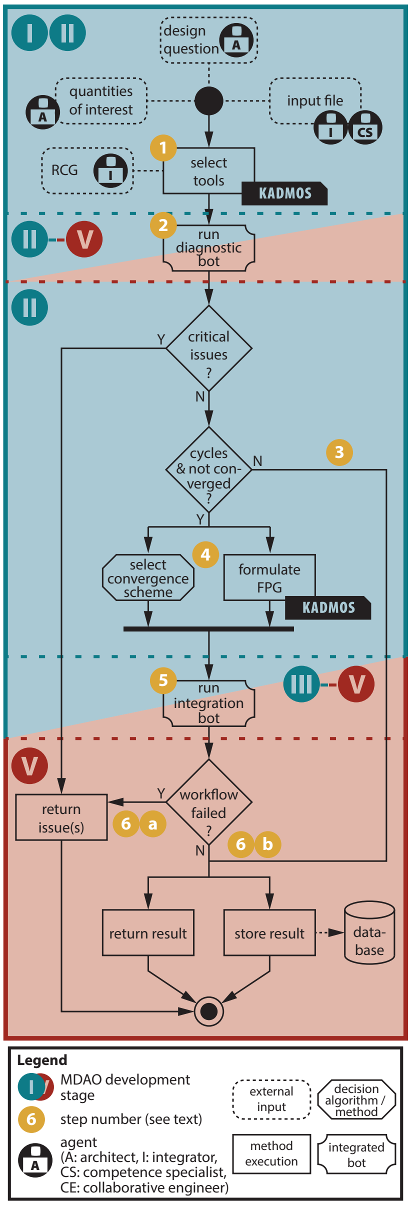


Figure 8.5: Process implementation for the synthesis bot

With this bot, the MDAO system is reconfigured once between the diagnostic bot and the final workflow. As for the diagnostic bot, multiple extension could be included to improve the bot and replace more repetitive human activities: automated error handling, determination of optimal convergence settings, or checking convergence throughout the design space.

Surrogate modelling bot

The previous two bots were focused on running a workflow to diagnose it and obtain the (synthesized) results. A different type of bot can be defined to automatically build a surrogate model. The outcome of such a bot is not an output file, but a new tool for the tool repository. Suppose the architect has the following question:

> *What would be the best surrogate model for this [quantity of interest] within this [design space] when we have [amount of time] available and the following [fitting methods] and [input file] at our disposal?*

This question represents a realistic situation in the industrial context. The architect prefers to create a surrogate model in a limited amount of time to meet practical project demands (i.e. a strict deadline). In addition, creation of different surrogate models for a range of methods is cheap and thus the bot will create multiple surrogate models and select the one with the highest precision. A possible implementation of this surrogate modelling bot is worked out in Fig. 8.6:

1. The bot determines the tools necessary to build the surrogate model for the design space specified. A graph-based, path-finding analysis can obtain the necessary tools between all design variables and the QOI.
2. The synthesis bot (Fig. 8.5) is used to determine which convergence scheme can best be used for the subsystem. Internally, the synthesis bot calls the diagnostic bot (Fig. 8.4), which, among other things, checks the provided input file and provides the execution times of the tools.
3. If the synthesis bot exits successfully, then three tasks are performed in parallel:
    a. The convergence scheme from the synthesis bot is selected.
    b. The FPG is formulated in KADMOS.
    c. A sampling strategy is determined. This strategy is a combination of sampling method (e.g. latin hypercube, Box-Behnken, full factorial) and their method-specific settings.
4. The specified DOE architecture is imposed on the FPG with KADMOS and stored as a CMDOWS file.
5. The bot can now estimate the execution time of the workflow based on the diagnostic information collected earlier. If this time estimation is larger than the available time specified by the integrator, then the sampling method could be adjusted, or the integrator is informed that there is not enough time available to create a reliable surrogate model.
6. If the execution time check is passed, then the CMDOWS file is used to instantiate the DOE workflow and this workflow is executed.
7. The results for each experiment are stored, including diagnostic and profiling data.
8. If too many experiments failed, then the bot reconfigures and executes an updated sampling on the system.

9. Once all workflows are executed, a surrogate model can be established for each fitting method that was provided. This fitting should be performed with a subset of the experiments carried out by the DOE.
10. The subset with unused experiment is subsequently used to determine the precision of the trained surrogate models. Alternatively, the full set of experiments could be used to train the surrogate and the determination of the precision could be skipped or performed based on another DOE.
11. Finally, the surrogate model with the highest precision is stored in the tool repository as a new tool.

The described bot implementation illustrates one possible approach for automating the creation of a surrogate model. Alternate implementations could base tool selection on another criteria than the single QOI used in this example, such as multiple QOIs or by specifying a certain tool to be modeled completely. Alternatively, instead of a time limit, the architect could also specify a minimum required precision of the surrogate model. The sampling method could then start small, fit models, and gradually increase sample sizes until the models are precise enough.

Optimization bot

After the diagnostic and synthesis bot, a logical next step is to also have a bot that supports solving optimization problems. A straightforward optimization bot would be a specialized extension of the integration bot shown in Fig. 8.3: based on the optimization problem formulated with KADMOS, a solution strategy can be obtained by imposing an architecture, instantiated as an executable workflow, and executed to return the optimal results. Hence, such a bot answers the question:

> *What is the best design you can find in this [design space] if the following [objective] needs to be minimized and these [constraints] have to be satisfied, provided we use the [MDO architecture] scheme and have this [input file] at our disposal?*

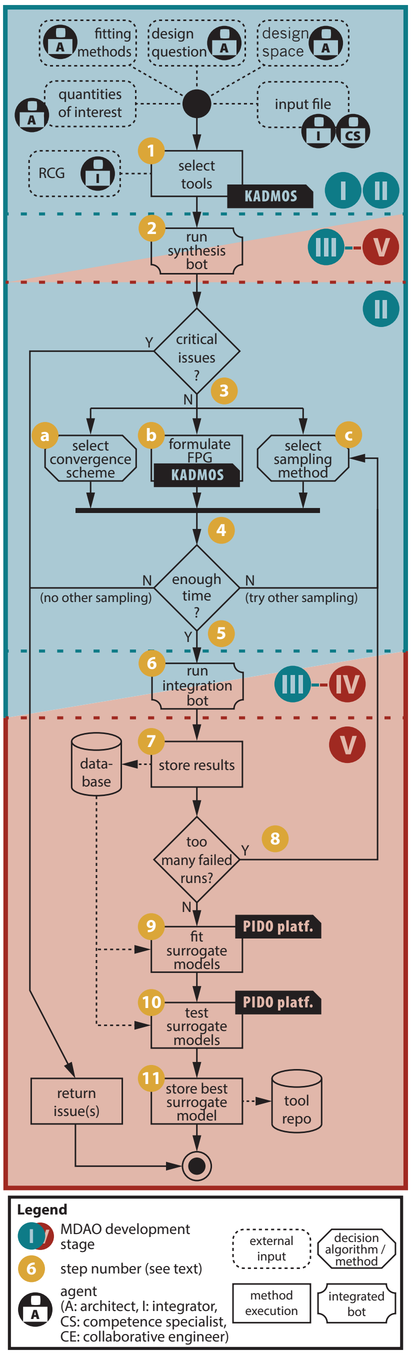
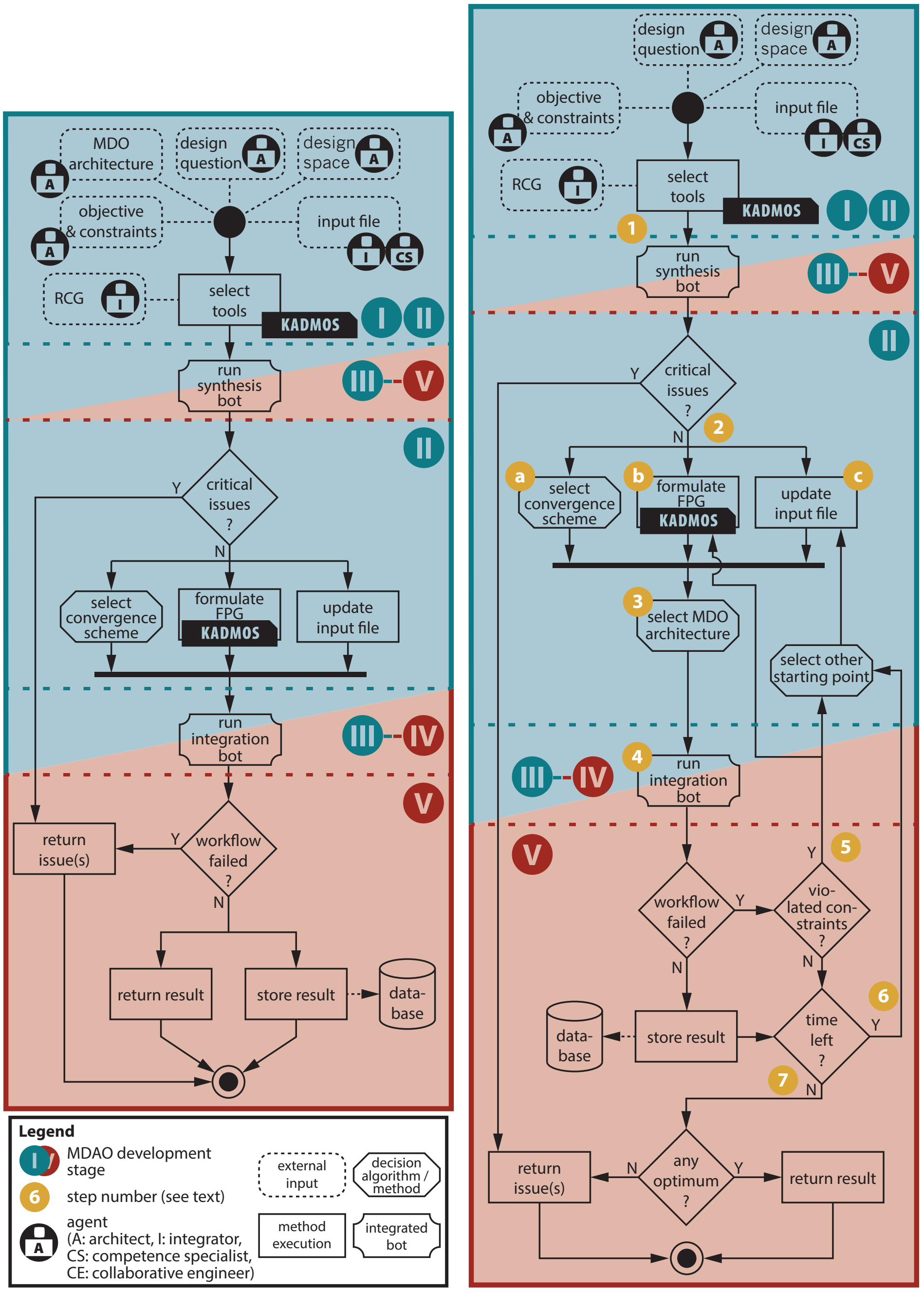


Figure 8.6: Process implementation for the surrogate modelling bot



a) basic optimization bot

b) advanced optimization bot

Figure 8.7: Process implementation for two optimization bots.

The process for the basic optimization bot is shown in Fig. 8.7a. The strength of the basic bot is that it can be extended to automate common integrator tasks by reconfiguring the MDAO system, such as:

- determining the right settings for solution strategy imposition and workflow execution (e.g. MDO architecture, optimization algorithm, tolerances);
- attempting to circumvent common issues encountered with running optimization algorithms (e.g. starting point with violated constraints, presence of local optima in the design space);
- taking advantage of available time (i.e. overnight executions) and/or computational facilities (i.e. multicore processing).

An advanced optimization bot is shown in Fig. 8.7b to illustrate some of these extensions. This bot answers the following question:

> *What is the best design you can find within [amount of time] in this [design space] if the following [objective] needs to be minimized, these [constraints] have to be satisfied, and with this [input file] at our disposal?*

Contrary to the basic bot, this bot automatically sets the MDO architecture. Additionally, the process includes approaches to fix violated constraints, check for local optima, and exploits the full available time. It operates as followed:

1. The tools from the repository required to perform the optimization are determined and the synthesis bot is executed.
2. If the synthesis bot returns a positive result, then three tasks are performed:
    a. The preferred convergence scheme is set based on the synthesis bot.
    b. The FPG is formulated using KADMOS.
    c. The input file is updated to start the optimization with a converged design.
3. Next, the bot determines the most promising architecture for the problem graph that was defined. How this task can be performed will be discussed in §§8.2.2.
4. Using the integration bot, the architecture is imposed on the FPG and the materialized workflow executed.
5. If the optimization fails, then the result is further analyzed: if the optimization algorithm was not able to find a point that satisfies all constraints (e.g. if the starting point includes constraint violations many optimization algorithms can have trouble reaching a valid design point), then the system will first be reconfigured to try and find a valid starting point. This is done by reformulating the FPG to first minimize constraint violations instead of the actual objective and by selecting an alternative point in the design space.
6. If the optimization failed, but not because of violated constraint, or if enough time is still left for another optimization run, then a new point in the design space is selected and the optimization is restarted. These additional runs are used to assess whether another optimum would be found from a different starting position. Hence, the bot checks whether a global or local optimum has been found.
7. If the bot is out of time, then the best design is returned or the architect is informed of the bot's inability to find a valid optimum.

Of course, this bot can be further extended in many directions. Failure of the optimization can have many other causes than constraint violations and the bot could include a range of algorithms to attempt fixes.

Finally, the surrogate modelling bot (Fig. 8.6) can also be included in the optimization bot to support a surrogate-based optimization approach. If this bot is included in the optimization bot, then the process of first building the surrogate model and then setting up and running the optimization using the surrogate instead of the actual tool can be fully automated. The addition of the surrogate bot is depicted in Fig. 8.8. The decision to replace tools for surrogate models could be a setting provided by the architect or a decision of the bot based on the profiling data of each tool and the available optimization time. If the system includes a tool with a long execution time and a small number of outputs, then it is a good candidate to be modelled as a surrogate first.

With the MDAO bot concept, the complex task of performing a surrogate-based optimization can be fully automated by combining different bots and decision-making algorithms in one process. All previously defined bots are used in this complex automated design process. The bots take advantage of a range of automation developments, such as KADMOS, CMDOWS, and OpenLEGO, to perform their tasks. A key aspect of the bot is that it should be able to make certain decisions autonomously. The bot's decision-making represents choices that are usually based on human judgment. The next section will outline how certain judgments can be automated.

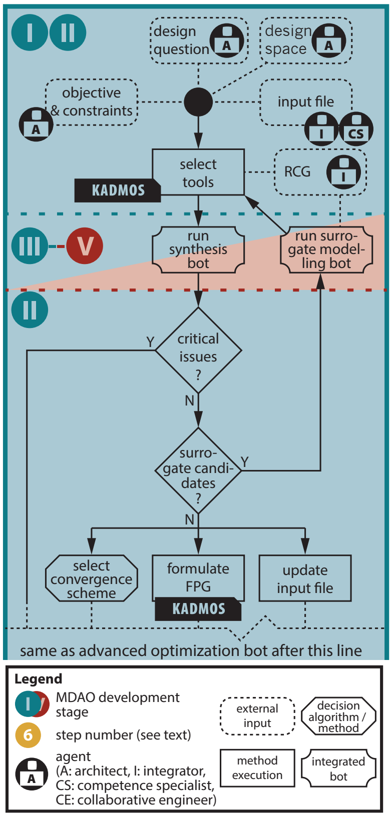


Figure 8.8: Inclusion of the surrogate modelling bot in the optimization bot to automate surrogate-based optimization

8.2.2. Bot automatic selection approaches

The hexagonal blocks in the bot processes represent automatic decision algorithms or methods. Hence, at these blocks the bot has to make a decision autonomously concerning, for example, the MDAO architecture, sampling strategy, or convergence scheme to be selected. Traditionally, these kind of decisions are made based on human judgment (and they can still be if the bot is set up to have these decisions as input), but if the decisions have the potential to be automatic, then two basic approaches are available to make a selection: rule-based or machine learning. The value of automatic decision-making is significant, especially in view of lowering the accessibility level of MDAO in industry.

With a rule-based approach, the decision algorithm contains a combination of if-statements (hence, rules) to make a decision. For example, the selection of a sampling strategy in the surrogate modelling bot could be based on the number of design variables and a rule of thumb on how many samples per design variable are required (usually around ten). If the number of design variables is small, then a full factorial sampling strategy can be selected to fully cover the whole design space. However, the number of experiments with full factorial sampling quickly gets out of hand with an increasing number of variables. Hence, after a certain threshold, the strategy would switch to latin hypercube sampling and the number of variables is simply set to ten times the number of design variables.

Rule-based decision making works well if clear rules can be prescribed based on logic or experience (i.e. rules of thumb). For example, this rule-based approach was implemented by Hoogreef [26] in his InFoRMA system to propose MDO architectures. However, for some decisions it is impossible to define such rules at all. Concerning the selection of MDAO architectures, studies from literature demonstrate there are many characteristics of the MDAO problem to be taken into account, as well as the computational infrastructure that is used, to be able to propose the most convenient MDAO architecture or optimization algorithm to be used for a certain criteria (e.g. speed, accuracy, global optimum). In those cases, a machine learning approach might be more practical. A prerequisite for these algorithms is the availability of a large training dataset. The high level of automation and standardization achieved in the methodology presented in this thesis, lends itself well to create a big, well-structured dataset:

- The tool repository, problem definition and blueprint of the workflow are each stored in a standardized file with CMDOWS.
- Results of the workflow execution are stored in its schema-based format, such as CPACS (§2.1) or IPACS (Fig. 7.6).
- Finally, workflow profiling data (e.g. total execution time, number of function calls) should also be stored in a well-structured format. This format still has to be defined or could be an extension of CMDOWS as it is closely related to the solution strategy definition.

The three data types above are stored in a single database for each workflow execution. If enough data is available, a machine learning algorithm can be trained to relate features of the problem and tool definition (e.g. number of tools, number of feedback variables, availability of derivatives, number of cores available) to a decision option, such as the MDAO architecture, convergence scheme, or optimization algorithm to be selected.

A key issue with the machine learning approach is the so-called 'cold start problem'; if the database does not contain enough data, then the decision algorithm cannot be trained (well). Fortunately, the MDAO bot concept can be used to prevent this issue. This would spark the creation of 'training data generation bots'. Within the design framework of the team, these bots will execute hundreds of MDAO workflows based on predefined scalable MDAO systems [88, 123] to populate the database. Once the database has obtained the critical size, it will be further extended by all subsequent MDAO bot executions, thereby continuously enriching the training database and improving the decision-making algorithm automatically through framework usage.

8.2.3. MDAO BOT FACTORY

The bot processes depicted in Figs. 8.4 to 8.8 describe an automated interaction with an MDAO system that benefits from earlier automation developments (i.e. KADMOS, CMDOWS, workflow materialization) in the formulation and execution phases. The different bots collaborate with other bots within their process and share similar decision-making, analysis, and storage tasks. In addition, small reconfigurations of the processes could be implemented to expand the set of design questions that can be answered automatically. Hence, the MDAO bot concept calls for the creation of a new framework application: the *MDAO bot factory*. This application enables the creation of bot processes by providing a modularized environment to easily integrate task components from different framework applications in one process. A mock-up of this application is shown in Fig. 8.9.

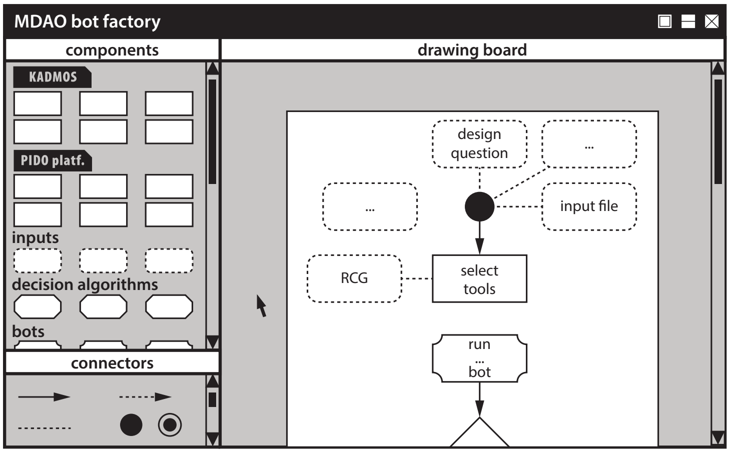


Figure 8.9: Mock-up of the MDAO bot factory application

The MDAO bot factory shares features with the PIDO platforms discussed in §2.5: it is an environment that can be used to formalize a process, albeit at a different level of abstraction and complexity: whereas the PIDO platform supports the execution of multidisciplinary workflows, the MDAO bot factory enables the creation of a higher-level process that includes the PIDO platform workflows and their reconfiguration. Fig. 8.10 shows the position of the MDAO bots in the knowledge architecture from Chapter 6. The bots are built on top of the developments used and presented in this dissertation to support and automate collaborative MDAO. The lower-level developments have automated the foundation of the MDAO process, thereby laying the groundwork for the creation of MDAO bots in future research. Within the AGILE Development Framework (ADF), the bots would become automatic tasks in the top business process layer fully automating the use of the layers below for a specific task. Thus, the bots could be implemented as 'widgets' in KE-chain to be used as automatic tasks in the collaborative platform. Finally, the use of bots might require the creation of a new agent role in the framework, that of the 'bot developer'. This agent will

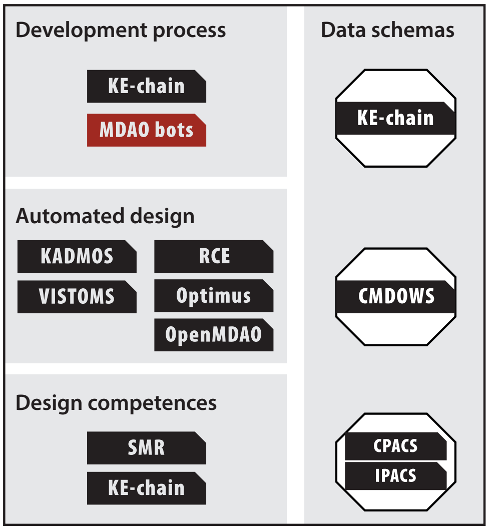


Figure 8.10: The position of the MDAO bots in the knowledge architecture from Fig. 6.4

create and maintain MDAO bots in close collaboration with other agents based on their needs for automation in the industrial practice.

This chapter has presented the outlook on future automation developments for the MDAO development process by introducing two novel concepts: composite architectures and MDAO bots. Both concepts serve as a road map for the continuation of the work presented in this dissertation. Their actual development is left to researchers participating in future collaborative MDAO projects.

9 Conclusions and Recommendations

This dissertation presented a novel methodological approach to realize agile MDAO systems. At the base of this methodology lies the five-stage MDAO development process and the division of that process into a formulation and execution phase, see Fig. 9.1. The developed implementation significantly lowered two hurdles frustrating the application of MDAO-based collaborative design processes in teams of heterogeneous experts:

configuration hurdle the set-up of the first executable workflow takes too long — namely 60-80% of the available project time — in collaborative projects in which a large team of heterogeneous experts needs to operate a large, complex MDAO system;

reconfiguration hurdle once the executable system has been established, it lacks agility and cannot be reconfigured easily to address 'what if scenarios' or answer new questions arising from the insights obtained from executions of previously configured systems.

The step from formulation to execution phase is a particularly challenging step in the MDAO development process. The gap between these two phases is called the 'implementation gap' and highlights the detachment between these two phases in state-of-the-art design processes and the tremendous effort required to create executable workflows.

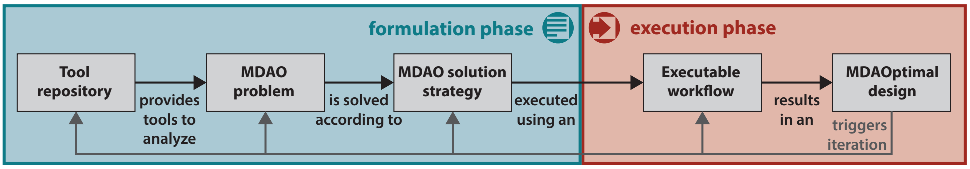


Figure 9.1: The five main stages of an MDAO system within a typical MDAO-based project and their relation

In the collaborative MDAO project AGILE, which provided the context for the developments original to this dissertation, a novel third-generation framework was created to tackle the posed (re)configuration challenges. The following needs were identified for this framework in Chapter 2, based on the examination of the state-of-the-art in MDAO-based design:

- The composition of MDAO systems using a fixed central data schema approach (i.e. CPACS) was selected as the most convenient method for collaborative MDAO. This approach calls for a dedicated platform to provide oversight and the ability to continuously check and adjust the composed system.
- Graph-based approaches to represent MDAO systems provide a solid conceptual basis, but require a rigorous revision and extensions to be useful in a third-generation framework.
- The third-generation framework requires a methodology (and prototype implementation) to handle the MDAO system at all stages of the development process shown in Fig. 9.1 — tool repository, MDAO problem, MDAO solution strategy, executable workflow, and MDAOptimal design — and needs to support all manipulations required to transform it.
- Both monolithic and distributed architecture types need to be supported throughout the development process.
- The formulated MDAO solution strategy needs to be PIDO platform-agnostic and the creation of the executable workflow has to be automated to bridge the implementation gap.

Ultimately, the novel developments in this dissertation needed to enable a novel framework generation that lowers the aforementioned configuration and reconfiguration hurdles. The MDAO systems in this framework, which can be configured quickly and remain reconfigurable easily, are referred to as agile MDAO systems.

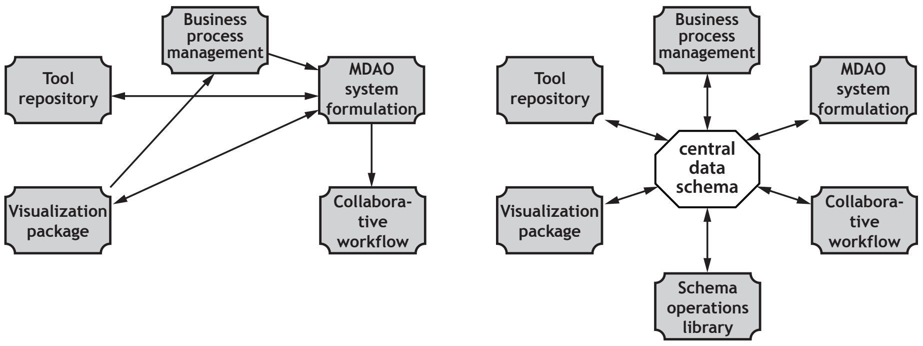
9.1. Developments and achievements

Three major developments were presented in Part II that represent three fundamental contributions to the establishment of the third-generation AGILE framework. In a nutshell, those developments are:

Graph-based methodological approach to formalize MDAO systems A graph-theoretic definition for MDAO systems was presented in Chapter 3. This definition includes all features required to formulate and manipulate the system concerning the three stages in the formulation phase (see Fig. 9.1) and includes two new graph-based concepts, namely instances and circularity index, necessary to handle the peculiarities of large and complex systems. Each stage is defined using graphs that need to meet strict graph-theoretic conditions: RCG (Repository Connectivity Graph), FPG (Fundamental Problem Graph), MDG (MDAO Data Graph), and MPG (MDAO Process Graph). The transformation between the first three stages is supported by automated methods to sequence and partition the problem graph, and finally impose the requested MDAO architecture. This results in the MDAO solution strategy, which serves as the blueprint of the executable workflow. The methodological approach was implemented as an open-source Python package called KADMOS.

Data standard to store and exchange formulated MDAO systems The graph-based formulations of the system need to be stored in a neutral format that is accessible to design team members and a range of framework applications (e.g. PIDO platform, business process management, visualization package). This data format was established in Chapter 4 and is called CMDOWS. CMDOWS is an XML-based schema and is used to store the system, so that its definition can be shared (and enriched) by the various applications involved in a third-generation MDAO framework. The schema is application- and product-agnostic, meaning that it does not contain any information specific to the framework applications themselves nor to the type of product (e.g. aircraft, wind turbine, car) being designed. Therefore, the schema can be used for any future framework. CMDOWS has a massive positive impact on framework structure: direct links between applications are no longer required (or even allowed), as all data passes through the centralized schema, see Fig. 9.2.

Workflow materialization Storing the workflow blueprint as a CMDOWS file constitutes the first half of the brand-new bridge that spans the implementation gap between system formulation and execution. The second half of this bridge was presented in Chapter 5. CMDOWS parsing capabilities were developed for three PIDO platforms: RCE, Optimus, and OpenMDAO. Each platform is able to materialize (i.e. automatically generate) the executable workflow formulated with KADMOS. Hence, this final development closes the crucial implementation gap in the MDAO development process and enables agile MDAO.



a) no Central Data Schema b) with Central Data Schema

Figure 9.2: Schematic overview of framework integration with and without a central data schema approach.

Verification and validation studies accompanying the developments in Part II highlighted the following achievements:

- The case study in Chapter 3 demonstrated that the graph-based methodology can be successfully applied on an MDAO system of industrial complexity that was created to perform aerostructural MDAO of a wing. The study showed that the methodology is able to cover the full formulation phase, handles a system's intricacies at the different stages, and is able to identify and fix typical issues of such systems though the new graph-based concepts (i.e. instances, circularity index).
- Two verification studies (Sellar problem and SSBJ test case) presented for all three PIDO platforms in Chapter 5 demonstrated how CMDOWS succeeds to be a platform-agnostic data standard from which workflows can be materialized and executed. Indirectly, these studies also validated the graph-based formalization from Chapter 3 by confirming its ability to formulate a clear, neutral MDAO solution strategy for a range of architectures (MDA, DOE, and MDO).
- The developed automated approach from Part II was also compared to the state-of-the-art manual approach for the two verification cases. This comparative study was performed using the Optimus platform for execution. The study showed the potential of the developed automation, with a 57% time reduction to configure and reconfigure the system in the SSBJ case.
- Finally, workflow materialization in OpenMDAO (with the OpenLEGO package)

was extended to also enable the generation of workflows based on the distributed architectures CO and BLISS-2000. This proved the support KADMOS and CMDOWS provide to formulate and store these complex architectures.

Although the achievements above already provide clear indications of the benefits of the developed methodology, the cases were performed in a small, non-collaborative test environment. Moreover, the tests focused on the developments themselves and were not placed in the context of the full third-generation MDAO framework.

9.2. Framework integrations

The developments described in the previous section can only be fully validated and appreciated if they are applied in a broader collaborative MDAO framework and tested accordingly. This point was addressed in the two chapters in Part III.

9.2.1. AGILE development framework

Chapter 6 covered the application and assessment of this dissertation's developments in the AGILE Development Framework (ADF). This third-generation framework benefits from the advancements original to this thesis in the following ways:

- The methodology proposed here was expanded in collaboration with the AGILE project partners to establish the 'knowledge architecture': a coherent and comprehensive formalization of the collaborative MDAO process. This architecture was used as a skeleton to develop the AGILE development framework, which is visualized in Fig. 9.3.
- KADMOS is a key application in the ADF that acts as 'engine under the hood' for many capabilities offered to the design team in the framework, such as establishing a coupled tool repository, automatically sequencing the tools in that repository, defining the design problem to be solved, and obtaining the solution strategy.
- The graph-based definition developed in Chapter 3 also impacted other applications, as it offered a well-structured method to describe the MDAO system that is both useful and intuitive for the design team to work with. For example, the graph-based formalization is used in the visualization package VISTOMS as it offers a structure to present the system in a visually meaningful way to the different members of the design team.
- The concept of using a central data standard to store the system's definition, which was worked out in Chapter 4, is one of the most successful and distinctive features of the ADF. CMDOWS gained a key position in the framework by acting as the hub through which a wide range of framework applications were connected, as visualized in Fig. 9.4. In addition, the format makes handling the MDAO system definition so straightforward that it significantly increased the design team's flexibility to try out different versions of the system before making any final decisions.
- The workflow materialization in RCE and Optimus was also implemented in the framework. The possibility to translate the neutral formulation from KADMOS into a fully executable workflow, effectively streamlined the formulation and execution phases of the MDAO development process. Large, complex, distributed executable workflows can now be created with the push of a button, removing the cumbersome and error-prone task of manual workflow creation in a PIDO plat-

form.

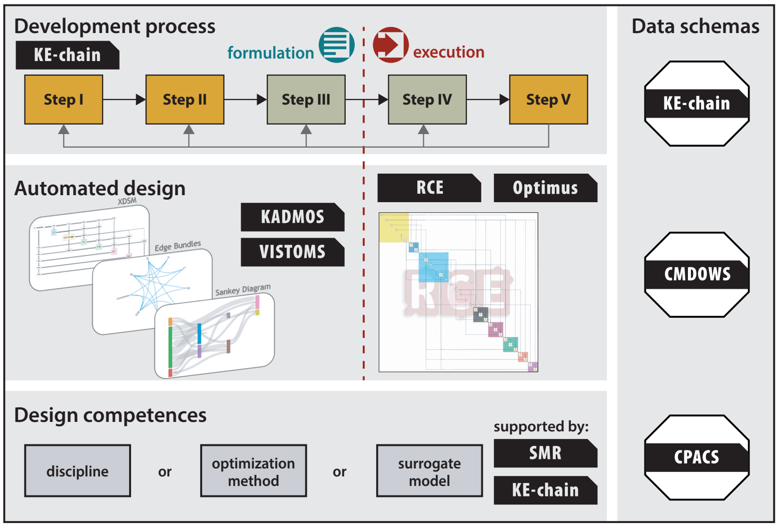


Figure 9.3: Overview of the AGILE Development Framework (ADF)

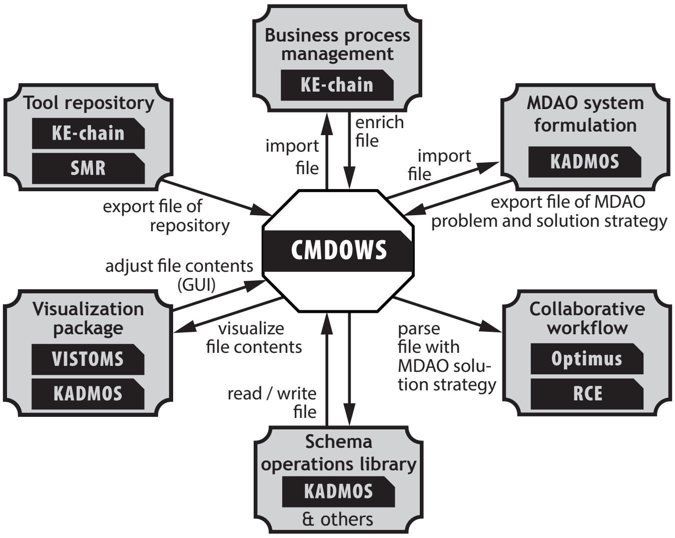


Figure 9.4: The established links between CMDOWS and the AGILE MDAO framework applications

The third-generation AGILE framework was used in different design projects, first as a demonstrator using conventional configurations and later for assessment with four unconventional configurations. Each project required multiple partners to collaborate and share their tools and expertise. After the projects concerning unconventional configurations were finished, a survey was conducted to critically assess the use of the developed framework and obtain estimations on the achieved time reductions. This survey showed that, overall, AGILE users consider the framework to be a success (their recommendations for improvement will be listed in the next section) and realizes the following time savings:

- The use of KADMOS leads to a 49% time reduction in configuring and reconfiguring the MDAO system in the formulation phase.
- The use of CMDOWS and workflow materialization leads to a 33% time reduction in obtaining the executable workflow.
- Using the full ADF, of which the developments original to this dissertation are key components, results in a 39% time reduction to configure and reconfigure MDAO systems.

Hence, in the AGILE context in which this research was performed, the developments presented in this dissertation had a significant positive impact and proved to lower the configuration and reconfiguration hurdles frustrating the use of MDAO in an collaborative setting.

9.2.2. Restructured aircraft design toolbox

This dissertation's methodology was also applied in a different context, in order to assess the general applicability of the work outside its original development boundaries in AGILE. In Chapter 7 the Initiator aircraft design toolbox was restructured to match the methodological approach to multidisciplinary design and establish a new third-generation framework. This restructuring entailed the following:

- the entangled MATLAB-based code was restructured to create modules following the fixed data schema approach;
- workflow capabilities of the original toolbox were disabled in favor of transferring these responsibilities to specialized applications: KADMOS for formulation and the selected PIDO platform OpenMDAO for execution;
- the original collection of MATLAB-based modules from the toolbox was extended with a heterogeneous set of tools (i.e. Python-based and mathematical relations) to demonstrate the extensibility of a third-generation framework and allow the analysis of new design problems.

After this restructuring, the framework was used in the 'agile way', and the following conclusions are drawn:

- The developed methodological approach remains valuable outside the boundaries of the AGILE paradigm.
- The approach disentangles a state-of-the-art design toolbox by imposing a separation of concerns:
  - disciplinary analysis (or framework modules) become standalone components by enforcing the central data schema approach;
  - a dedicated visualization package (VISTOMS) takes care of visualizing the toolbox's structure and dependencies;
  - formulation of design problem and solution strategy are performed by the KADMOS package;
  - execution of workflows is performed by the specialized OpenMDAO package and, thanks to CMDOWS, can actually be carried out by a PIDO platform of choice.
- The approach unveils new opportunities for using the disciplinary modules of the toolbox, to configure also design space exploration and optimization strategies that were originally not available.
- The approach supports reconfigurations of the MDAO system that were nonexis-

tent in the original toolbox, such as multicore processing, and would otherwise require rigorous adjustments of the entangled toolbox that are difficult to implement.

In conclusion, the framework integrations in Part III confirmed that the research goal defined in the introduction has been achieved.

> to develop, implement, and assess a new methodology, specifically addressing the collaborative aspects of performing MDAO, that enables formalized MDAO systems in all stages of the formulation phase, such to lower the configuration and reconfiguration hurdles.

What remains are several recommendations to further improve and extend the methodological approach. Additionally, an outlook for future work was presented in Chapter 8.

9.3. RECOMMENDATIONS AND OUTLOOK

The following major recommendations have been collected throughout this dissertation to further enhance the established MDAO development process presented in this work:

- The graph-based representation of MDAO systems can be further exploited by developing additional analysis and manipulation methods. Suggestions from AGILE users included a method for automatic tool selection from the repository, where the selection is based on given criteria such as tool fidelity levels, runtime, or tool availability (when provided as web services). Another example is the automatic determination of which tools should be 'surrogate modeled' for a specific workflow, including the required graph manipulation to formulate such a surrogate-based optimization approach.
- The fixed central data schema is a powerful concept to couple disciplinary tools in a single system and constitutes a core assumption for system composition in this work. However, by closing the implementation gap a discrepancy between the I/O definitions required in the formulation and execution phases was uncovered, as was explained in Fig. 6.16. Future developments should investigate this discrepancy more closely. The recommended step forward is to add a 'semantic enrichment layer' to the schema definition so that disciplinary experts can use this to define their tool's I/O, as alternative to the current practice of providing an XML input and output files for each individual project.
- The list of architectures that can be imposed on MDAO problems should be extended and 'composite architectures' (see Fig. 8.1 for a DOE-MDO example) should be defined and implemented. Both these aspects call for a flexible way of defining additional architectures. Graph manipulation methods in KADMOS can already be easily reused to create new architecture imposition methods. The creation of such new architectures should be supported by a dedicated 'architecture factory' application that enables a modular approach to defining new (composite) architectures.
- The performance of the materialized workflows in all three PIDO platforms should be improved. Initial developments were focused on generating robust, executable workflows, but cannot compete with their manually created counterparts due to the introduced overhead. This overhead mainly stems from the required XML file handling. There is still a lot of room in the materialization methods to reduce this overhead.

- Solution strategies that take advantage of multicore processing can be defined in KADMOS and stored as a CMDOWS file, however, none of the PIDO platforms were able to run materialized workflows using multicore processing for execution. This is another recommended improvement for the materialization methods.
- Finally, a topic almost completely ignored in this work (a minor exception being workflow materialization in OpenLEGO), is the use of analytical derivatives to improve the performance of gradient-based optimization strategies. Analytical derivatives were considered out of scope, as the two frameworks used in Part III did not contain any modules that could provide them. Nevertheless, there is huge potential in including analytical derivatives in the formulation phase. Simply put, analytical derivatives are just additional outputs of a tool, but (re)configuring MDAO systems including this additional aspect poses another major challenge in the future application of MDAO in industry. KADMOS can be extended to tackle this challenge, by introducing a new module that analyses and interprets 'derivative connectivity' in a new graph, similar to the already defined Repository Connectivity Graph (RCG), to assist the team in correctly configuring a derivative-enabled system.

In addition to these recommendations, Chapter 8 provided an outlook on future research by proposing a new concept to be developed: MDAO bots. This dissertation can be seen as the development of the first MDAO bot, since it supports a semi-automated development process for MDAO systems. Future research can be positioned in the context of establishing increasingly advanced MDAO bots that take over repetitive, error-prone human tasks to further lower hurdles encountered when using MDAO in a collaborative environment involving a large, heterogeneous team of experts. Ultimately, MDAO bots can be developed that automatically (re)configure large, complex MDAO systems in close collaboration with engineers who provide their irreplaceable judgment, knowledge, and creativity.

REFERENCES

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

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

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

[4] Gero, J., and Maher, M., *Modeling creativity and knowledge-based creative design*, Psychology Press, 1993.

[5] Schmit, L. A., "Structural Design by Systematic Synthesis," *2nd Conference on Electronic Computation*, American Society of Civil Engineers, New York, 1960, pp. 105–132.

[6] Schmit, L. A., "Structural synthesis—precursor and catalyst. Recent experiences in multidisciplinary analysis and optimization," Tech. Rep. CP-2337, NASA, 1984.

[7] Haftka, R. T., "Automated procedure for design of wing structures to satisfy strength and flutter requirements," Tech. Rep. TN-D-7264, NASA Langley Research Center, Hampton, VA, 1973.

[8] Haftka, R. T., and Shore, C. P., "Approximation methods for combined thermal/structural design," Tech. Rep. TP-1428, NASA, June 1979.

[9] Bowcutt, K. G., "A perspective on the future of aerospace vehicle design," *12th AIAA International Space Planes and Hypersonic Systems and Technologies, Norfolk, VA, AIAA Paper*, 2003.

[10] Vandenbrande, J. H., Grandine, T. A., and Hogan, T., "The search for the perfect body: Shape control for multidisciplinary design optimization," *44th AIAA Aerospace Science Meeting and Exhibit, Reno, NV*, 2006.

[11] Piperni, P., DeBlois, A., and Henderson, R., "Development of a multilevel multidisciplinary-optimization capability for an industrial environment," *AIAA journal*, Vol. 51, No. 10, 2013, pp. 2335–2352.

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

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

[14] Martins, J. R. R. A., and Lambe, A. B., "Multidisciplinary Design Optimization: A Survey of Architectures," *AIAA Journal*, Vol. 51, No. 9, 2013, pp. 2049–2075. doi: 10.2514/1.J051895, URL `http://dx.doi.org/10.2514/1.J051895`.

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

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

[17] Gallard, F., Vanaret, C., Guénot, D., Gachelin, V., Lafage, R., Pauwels, B., Barjhoux, P.-J., and Gazaix, A., "GEMS: A Python Library for Automation of Multidisciplinary Design Optimization Process Generation," *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2018, p. 0657.

[18] Ciampa, P. D., and Nagel, B., "The AGILE Paradigm: the next generation of collaborative MDO," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017.

[19] Tosserams, S., Hofkamp, A., Etman, L., and Rooda, J., "A specification language for problem partitioning in decomposition-based design optimization," *Structural and Multidisciplinary Optimization*, Vol. 42, No. 5, 2010, pp. 707–723.

[20] Martins, J. R. R. A., and Marriage, C., "An object-oriented framework for multidisciplinary design optimization," *3rd AIAA multidisciplinary design optimization specialist conference. Waikiki, Hawaii, USA*, 2007.

[21] Gray, J., Moore, K. T., and Naylor, B. A., "OpenMDAO: An open source framework for multidisciplinary analysis and optimization," *13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Vol. 5, 2010.

[22] Gray, J., Moore, K. T., Hearn, T. A., and Naylor, B. A., "A standard platform for testing and comparison of MDAO architectures," *8th AIAA multidisciplinary design optimization specialist conference (MDO), Honolulu*, 2012, pp. 1586–1611.

[23] Gray, J. S., Hwang, J. T., Martins, J. R., Moore, K. T., and Naylor, B. A., "OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization," *Structural and Multidisciplinary Optimization*, 2019, pp. 1–30.

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

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

[26] Hoogreef, M. F. M., "Advise, Formalize and Integrate MDO Architectures - A Methodology and Implementation," Ph.D. thesis, Delft University of Technology, 2017.

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

[28] Jepsen, J., *CPACS documentation (version 2.3.1)*, DLR, 2018.

[29] Görtz, S., Ilic, C., Schuster, A., Jepsen, J., Leitner, M., Schulze, M., Scherer, J., Petsch, M., Becker, R., and Zur10, S., "Multi-Level MDO of a Long-Range Transport Aircraft Using a Distributed Analysis Framework," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017, p. 4326.

[30] Moerland, E., Becker, R.-G., and Nagel, B., "Collaborative understanding of disciplinary correlations using a low-fidelity physics-based aerospace toolkit," *CEAS Aeronautical Journal*, Vol. 6, No. 3, 2015, pp. 441–454.

[31] Lefebvre, T., Bartoli, N., Dubreuil, S., Panzeri, M., Lombardi, R., Della Vecchia, P., Nicolosi, F., Ciampa, P. D., Anisimov, K., and Savelyev, A., "Methodological enhancements in MDO process investigated in the AGILE European project," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017.

[32] Moerland, E., Pfeiffer, T., Böhnke, D., Jepsen, J., Freund, S., Liersch, C., Chiozzotto, G. P., Klein, C., Scherer, J., and Hasan, Y. J., "On the Design of a Strut-Braced Wing Configuration in a Collaborative Design Environment," *18th AIAA\ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017.

[33] Dykes, K., Rethore, P., Zahle, F., and Merz, K., "IEA Wind Task 37 Final Proposal Wind Energy Systems Engineering: Integrated RD&D," Tech. rep., Tech. rep. International Energy Agency, 2015.

[34] Lano, R., "The N^2 chart," Tech. rep., TRW Software Series, Redondo Beach, CA, 1977.

[35] Steward, D. V., "The design structure system: a method for managing the design of complex systems," *Engineering Management, IEEE Transactions on*, , No. 3, 1981, pp. 71–74.

[36] Wagner, T. C., and Papalambros, P. Y., "General framework for decomposition analysis in optimal design," *ASME Design Engineering*, Vol. 65-2, 1993, pp. 315–325.

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

[38] Gazaix, A., Gallard, F., Gachelin, V., Druot, T., Grihon, S., Ambert, V., Guénot, D., Lafage, R., Vanaret, C., Pauwels, B., et al., "Towards the Industrialization of New MDO Methodologies and Tools for Aircraft Design," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017, p. 3149.

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

[40] Etman, L., Kokkolaras, M., Hofkamp, A., Papalambros, P. Y., and Rooda, J., “Coordination specification in distributed optimal design of multilevel systems using the χ language,” *Structural and Multidisciplinary Optimization*, Vol. 29, No. 3, 2005, pp. 198–212.

[41] Berners-Lee, T., Hendler, J., Lassila, O., et al., “The semantic web,” *Scientific american*, Vol. 284, No. 5, 2001, pp. 28–37.

[42] Rogers, J., “DEMAID/GAA - Enhanced Design Manager’s Aid for Intelligent Decomposition (Genetic Algorithms),” *6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Sept*, 1996, pp. 4–6.

[43] Guenov, M. D., and Barker, S. G., “Application of axiomatic design and design structure matrix to the decomposition of engineering systems,” *Systems engineering*, Vol. 8, No. 1, 2005, pp. 29–40.

[44] Wilschut, T., Etman, P. L., Rooda, J. J., and Adan, I., “Multi-Level Flow-Based Markov Clustering for Design Structure Matrices,” *Journal of Mechanical Design*, 2017.

[45] Michelena, N. F., and Papalambros, P. Y., “A hypergraph framework for optimal model-based decomposition of design problems,” *Computational optimization and applications*, Vol. 8, No. 2, 1997, pp. 173–196.

[46] Allison, J., Kokkolaras, M., and Papalambros, P., “Optimal partitioning and coordination decisions in decomposition-based design optimization,” *Journal of Mechanical Design*, Vol. 131, No. 8, 2009, p. 081008.

[47] Braun, R. D., “Collaborative optimization: an architecture for large-scale distributed design,” Ph.D. thesis, Stanford University, 1996.

[48] Sobieski, J., Agte, J. S., and Sandusky, R. R., “Bilevel integrated system synthesis,” *AIAA journal*, Vol. 38, No. 1, 2000, pp. 164–172.

[49] Sobieski, J., “Optimization by decomposition: a step from hierarchic to non-hierarchic systems,” Tech. Rep. TR-CP-3031, NASA Langley Research Center, September 1988.

[50] Lu, Z., and Martins, J. R. R. A., “Graph partitioning-based coordination methods for large-scale multidisciplinary design optimization problems,” *14th ISSMO multidisciplinary analysis optimization conference*, 2012, pp. 1–13.

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

[52] Ciampa, P. D., Baalbergen, E. H., and Lombardi, R., “A Collaborative Architecture supporting AGILE Design of Complex Aeronautics Products,” *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017.

[53] Hoogreef, M., and La Rocca, G., “An MDO advisory system supported by knowledge-based technologies,” *16th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Dallas, Texas, 22-26 June 2015; AIAA 2015-2945*, American Institute of Aeronautics and Astronautics (AIAA), 2015.

[54] Sobieski, J., Altus, T. D., Phillips, M., and Sandusky, R., "Bilevel integrated system synthesis for concurrent and distributed processing," *AIAA journal*, Vol. 41, No. 10, 2003, pp. 1996–2003.

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

[56] Zur, S., and Tröltzsch, A., "Optimization of the DLR Space Liner inside the integration environment RCE," *Engineering Optimization*, Vol. 1, 2014, pp. 757–761.

[57] Tröltzsch, A., Siggel, M., Kopp, A., and Schwanekamp, T., "Multidisciplinary Analysis of the DLR SpaceLiner Concept by different Optimization Techniques," *11th World Congress on Computational Mechanics (WCCM XI)*, 2014.

[58] Kroll, N., Abu-Zurayk, M., Dimitrov, D., Franz, T., Führer, T., Gerhold, T., Görtz, S., Heinrich, R., Ilic, C., Jepsen, J., et al., "DLR project Digital-X: towards virtual aircraft design and flight testing based on high-fidelity methods," *CEAS Aeronautical Journal*, Vol. 7, No. 1, 2016, pp. 3–27.

[59] Moerland, E., Deinert, S., Daoud, F., Dornwald, J., and Nagel, B., "Collaborative aircraft design using an integrated and distributed multidisciplinary product development process," *30th Congress of the international council for aeronautical sciences*, 2016.

[60] Adams, B. M., Bauman, L. E., Bohnhoff, W. J., Dalbey, K. R., Ebeida, M. S., Eddy, J. P., Eldred, M. S., Hough, P. D., Hu, K. T., Jakeman, J. D., Stephens, J. A., Swiler, L. P., Vigil, D. M., and Wildey, T. M., "SAND2014-4633: Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 User's Manual," Tech. rep., Sandia National Laboraties, 2015.

[61] Augustinus, R., "Supporting MDO through dynamic workflow (re)generation," Master's thesis, TU Delft, 2016.

[62] Liu, K. J., Zhao, H. L., and Xu, C., "MDO of Anti-creeper on High speed train Using Optimus," *Applied Mechanics and Materials*, Vol. 130, Trans Tech Publ, 2012, pp. 3128–3132.

[63] Koch, P., Wujek, B., and Golovidov, O., "A multi-stage, parallel implementation of probabilistic design optimization in an MDO framework," *8th Symposium on Multidisciplinary Analysis and Optimization*, 2000, p. 4805.

[64] Shen, X., and Hu, W., "Multidisciplinary Design Optimization Research of overall Aero-engine based on Flow Path," *17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2016, p. 3366.

[65] Tfaily, A., and Kokkolaras, M., "Integrating Air Systems in Aircraft Multidisciplinary Design Optimization," *2018 Multidisciplinary Analysis and Optimization Conference*, 2018, p. 3742.

[66] Garcia, J., Brown, J., Kinney, D., Bowles, J., Jiang, X., Dupzyk, I., Huynh, L., and Lau, E., "Co-Optimization of Mid Lift to Drag Vehicle Concepts for Mars Atmospheric Entry," *10th AIAA/ASME Joint Thermophysics and Heat Transfer Conference*, 2010. doi:10.2514/6.2010-5052, URL `http://dx.doi.org/10.2514/6.2010-5052`.

[67] Miano, C., “Using Optimization to Exploit a Composable Satellite Product Line Architecture,” *AIAA SPACE 2015 Conference and Exposition*, 2015. doi:10.2514/6.2015-4618, URL `http://dx.doi.org/10.2514/6.2015-4618`.

[68] Cramer, E., Gablonsky, J., Lurati, L., Sellers, P., and Simonis, J., “Practical Experience with a Multi-Objective Model-Management Framework Optimization Method,” *12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2012. doi:10.2514/6.2012-5603, URL `http://dx.doi.org/10.2514/6.2012-5603`.

[69] Hwang, J. T., and Martins, J. R., “A computational architecture for coupling heterogeneous numerical models and computing coupled derivatives,” *ACM Transactions on Mathematical Software (TOMS)*, Vol. 44, No. 4, 2018, p. 37.

[70] Martins, J., and Hwang, J., “Review and unification of methods for computing derivatives of multidisciplinary computational models,” *AIAA journal*, Vol. 51, No. 11, 2013, pp. 2582–2599.

[71] Hwang, J. T., Lee, D. Y., Cutler, J. W., and Martins, J. R., “Large-scale multidisciplinary optimization of a small satellite’s design and operation,” *Journal of Spacecraft and Rockets*, Vol. 51, No. 5, 2014, pp. 1648–1663.

[72] Chung, H., Hwang, J. T., Gray, J. S., and Kim, H. A., “Implementation of topology optimization using openMDAO,” *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2018, p. 0653.

[73] Gray, J. S., and Martins, J. R. R. A., “Coupled aeropropulsive design optimisation of a boundary-layer ingestion propulsor,” *The Aeronautical Journal*, 2018, pp. 1–17. doi:10.1017/aer.2018.120, URL `http://dx.doi.org/10.1017/aer.2018.120`.

[74] van der Elst, S., Moerland, E., Lugtenburg, J., Hootsman, L. M., Deinert, S., Motzer, M., and Fernandez, C., “D2.3: Industrial service-oriented process methodology,” Tech. rep., IDEaliSM project, 2017.

[75] Moerland, E., d’Ippolito, R., Panzeri, M., Raju Kulkarni, A., Hoogreef, M. F. M., Eheim, M., Bengtsson, K., Haenisch, J., Deinert, S., van den Berg, T., Fernandez, C., van der Elst, S., and Hendrich, T., “D3.2.2: Advanced Integration Framework - update,” Tech. rep., IDEaliSM project, March 2017.

[76] La Rocca, G., “Knowledge based engineering techniques to support aircraft design and optimization,” Ph.D. thesis, Delft University of Technology, 2011.

[77] van Lugtenburg, J., van den Berg, T., Hootsman, L. M., Fernandez, C., Raju Kulkarni, A., Stoeckl, F., Motzer, M., Eheim, M., Tamm, C., and Deinert, S., “D3.2.2: Use-case specification - final,” Tech. rep., IDEaliSM project, March 2017.

[78] Kroo, I., and Manning, V., “Collaborative optimization: status and directions,” *8th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2000.

[79] Dener, A., Meng, P., Hicken, J., Kennedy, G. J., Hwang, J., and Gray, J., “Kona: A Parallel Optimization Library for Engineering-Design Problems,” *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2016, p. 1422.

[80] Bartoli, N., Lefebvre, T., Dubreuil, S., Olivanti, R., Bons, N., Martins, J., Bouhlel, M., and Morlier, J., "An adaptive optimization strategy based on mixture of experts for wing aerodynamic design optimization," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017, p. 4433.

[81] Heydari Beni, E., Lagaisse, B., and Joosen, W., "Adaptive and reflective middleware for the cloudification of simulation & optimization workflows," *ARM 2017 - 16th Workshop on Adaptive and Reflective Middleware*, 2017.

[82] van Gent, I., and La Rocca, G., "Formulation and integration of MDAO systems for collaborative design: A graph-based methodological approach," *Aerospace Science and Technology*, Vol. 90, 2019, pp. 410 – 433. doi:https://doi.org/10.1016/j.ast.2019.04.039, URL `https://doi.org/10.1016/j.ast.2019.04.039`.

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

[84] Smith, D., Eggen, M., and St. Andre, R., *A transition to advanced mathematics*, Nelson Education, 2014.

[85] Hagberg, A. A., Schult, D. A., and Swart, P. J., "Exploring network structure, dynamics, and function using NetworkX," *7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, 2008, pp. 11–15.

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

[87] Bruggeman, A.-L., "Automated Execution Process Formulation using Sequencing and Decomposition Algorithms for Collaborative MDAO," Master's thesis, Delft University of Technology, 2019.

[88] Zhang, D., Song, B., Wang, P., and He, Y., "Performance Evaluation of MDO Architectures within a Variable Complexity Problem," *Mathematical Problems in Engineering*, Vol. 2017, 2017.

[89] Karypis, G., and Kumar, V., "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, Vol. 20, No. 1, 1999, pp. 359–392.

[90] Sobieski, J., Agte, J. S., and Sandusky, R. R., "Bi-Level Integrated System Synthesis," Tech. Rep. TM-1998-208715, NASA Langley Research Center, August 1998.

[91] Elmendorp, R., Vos, R., and La Rocca, G., "A conceptual design and analysis method for conventional and unconventional airplanes," *ICAS 2014: Proceedings of the 29th Congress of the International Council of the Aeronautical Sciences, St. Petersburg, Russia, 7-12 September 2014*, International Council of Aeronautical Sciences, 2014.

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

[93] Elham, A., "Adjoint quasi-three-dimensional aerodynamic solver for multi-fidelity wing aerodynamic shape optimization," *Aerospace Science and Technology*, Vol. 41, 2015, pp. 241–249.

[94] Pfeiffer, T., Nagel, B., Böhnke, D., Rizzi, A., and Voskuijl, M., "Implementation of a heterogeneous, variable-fidelity framework for flight mechanics analysis in preliminary aircraft design," *60. Deutscher Luft- und Raumfahrtkongress*, 2011.

[95] Macquart, T., Werter, N., and De Breuker, R., "Aeroelastic Tailoring of Blended Composite Structures using Lamination Parameters," *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2016, p. 1966.

[96] van Gent, I., La Rocca, G., and Hoogreef, M. F. M., "CMDOWS: A Proposed New Standard to Store and Exchange MDO Systems," *CEAS Aeronautical Journal*, Vol. 9, No. 4, 2018, pp. 607 – 627.

[97] Raju Kulkarni, A., Hoogreef, M. F. M., and La Rocca, G., "Combining semantic web technologies and KBE to solve industrial MDO problems," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017.

[98] Gondhalekar, A. C., Guenov, M. D., Wenzel, H., Balachandran, L. K., and Nunez, M., "Neutral Description and Exchange of Design Computational Workflows," *18th International Conference on Engineering Design*, 2011.

[99] Kesseler, E., and Guenov, M. D., *Advances in collaborative civil aeronautical multidisciplinary design optimization*, American Institute of Aeronautics and Astronautics, 2010.

[100] Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Elmqvist, H., Junghanns, A., Mauß, J., Monteiro, M., Neidhold, T., Neumerkel, D., et al., "The functional mockup interface for tool independent exchange of simulation models," *8th International Modelica Conference*, 2011, pp. 105–114.

[101] Grosskopf, A., Decker, G., and Weske, M., *The process: business process modeling using BPMN*, Meghan Kiffer Press, 2009.

[102] Lombardi, R., van Gent, I., and La Rocca, G., "Reconfigurable Formulation and Implementation of MDAO Systems," *NAFEMS World Congress*, 2019.

[103] Baalbergen, E., Kos, J., Louriou, C., Campguilhem, C., and Barron, J., "Streamlining cross-organisation product design in aeronautics," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, Vol. 231, No. 12, 2017, pp. 2192–2202.

[104] "Noesis Solutions, Optimus Rev 10.19 - User's Manual (available on request)," , 2017.

[105] de Vries, D., "Towards the Industrialization of MDAO," Master's thesis, Delft University of Technology, 2017.

[106] Agte, J. S., *User's Guide for the MATLAB Implementation of BLISS*, DLR, 2005.

[107] van Gent, I., Aigner, B., Beijer, B., Jepsen, J., and La Rocca, G., "Knowledge architecture supporting the next generation of MDO in the AGILE paradigm," *Progress in Aerospace Sciences (accepted for publication)*, 2018.

[108] van Gent, I., Aigner, B., Beijer, B., and La Rocca, G., "A critical look at design automation solutions for collaborative MDO in the AGILE Paradigm," *19th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2018.

[109] Lafage, R., “XDSMjs package - XDSM diagram generator written in javascript,” *https://github.com/OneraHub/XDSMjs*, 2016. URL `https://github.com/OneraHub/XDSMjs`.

[110] Sankey, H. R., “The Thermal Efficiency of Steam Engines,” *Minutes of the Proceedings of the Institution of Civil Engineers*, 1896, pp. 182–212. doi:10.1680/imotp.1896.19564.

[111] Schmidt, M., *Der Einsatz von Sankey-Diagrammen im Stoffstrommanagement*, Beiträge der Hochschule Pforzheim, Hochsch. Pforzheim, 2006. URL `https://books.google.de/books?id=zhBbMgAACAAJ`.

[112] Atkinson, N., “bihisankey.js package - BiDirectional Hierarchical Sankey Diagram,” *https://github.com/Neilos/bihisankey*, 2015. URL `https://github.com/Neilos/bihisankey`.

[113] Holten, D., “Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data,” *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 5, 2006, pp. 741–748.

[114] Bostock, M., “D3.js website,” *https://d3js.org/*, 2015. URL `https://d3js.org/`.

[115] Moerland, E., Ciampa, P. D., Zur, S., Baalbergen, E. H., Noskov, N., D’Ippolito, R., and Lombardi, R., “Collaborative Architecture supporting the next generation of MDO within the AGILE paradigm,” *Progress in Aerospace Sciences (submitted)*, 2019.

[116] La Rocca, G., Langen, T., and Brouwers, Y., “The design and engineering engine. Towards a modular system for collaborative aircraft design,” *28th Congress of the International Council of the Aeronautical Sciences*, 2012.

[117] Dommelen, J. v., and Vos, R., “Conceptual design and analysis of blended-wing-body aircraft,” *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, Vol. 228, No. 13, 2014, pp. 2452–2474.

[118] Torenbeek, E., *Synthesis of Subsonic Airplane Design*, Delft University Press, 1982.

[119] Raymer, D., *Aircraft Design: A Conceptual Approach*, American Institute of Aeronautics and Astronautics, Inc., 2012.

[120] Brown, M., and Vos, R., “Conceptual design and evaluation of blended-wing body aircraft,” *2018 AIAA Aerospace Sciences Meeting*, 2018, p. 0522.

[121] Vos, R., and Hoogreef, M. F., “System-level assessment of tail-mounted propellers for regional aircraft,” *Proceedings of the 31st Congress of the International Council of Aeronautical Sciences*, 2018.

[122] Schouten, T., Hoogreef, M., and Vos, R., “Effect of Propeller Installation on Performance Indicators of Regional Turboprop Aircraft,” *AIAA Scitech 2019 Forum*, 2019, p. 1306.

[123] Vanaret, C., Gallard, F., and Martins, J., “On the Consequences of the" No Free Lunch" Theorem for Optimization on the Choice of an Appropriate MDO Architecture,” *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017, p. 3148.

SOFTWARE REFERENCES

[S1] Phoenix Integration. ModelCenter®. Commercial software, Jan 2019. URL https://www.phoenix-int.com/product/modelcenter-integrate. last checked: 21.01.2019.

[S2] Noesis Solutions N.V. Optimus. Commercial software, Jan 2019. URL https://www.noesissolutions.com/our-products/optimus. last checked: 21.01.2019.

[S3] DLR - German Aerospace Center. Remote Component Environment (RCE). Open-source (Eclipse Public License 1.0) download available on website, Jan 2019. URL http://rcenvironment.de/. last checked: 21.01.2019.

[S4] Microsoft. Excel. Commercial software, Feb 2019. URL https://products.office.com/en/excel. last checked: 08.02.2019.

[S5] Microsoft. Visio. Commercial software, Feb 2019. URL https://products.office.com/en/visio/flowchart-software. last checked: 11.02.2019.

[S6] NASA Glenn Research Center. OpenMDAO. Open-source (Apache License 2.0) GitHub repository, Jan 2019. URL https://github.com/OpenMDAO/OpenMDAO. last checked: 21.01.2019.

[S7] DLR - German Aerospace Center. Common Parametric Aircraft Configuration Schema. Open-source (Apache License 2.0) GitHub repository, Jan 2019. URL https://github.com/DLR-LY/CPACS. last checked: 21.01.2019.

[S8] DLR - German Aerospace Center. Fast and simple XML interface library. Open-source (Apache License 2.0) GitHub repository, Jan 2019. URL https://github.com/DLR-SC/tixi. last checked: 21.01.2019.

[S9] DLR - German Aerospace Center. The TiGL Geometry Library to process aircraft geometries in pre-design. Open-source (Apache License 2.0) GitHub repository, Jan 2019. URL https://github.com/DLR-SC/tigl. last checked: 21.01.2019.

[S10] ONERA - The French Aerospace Lab. XDSM generator written in javascript. Open-source (Apache License 2.0) GitHub repository, Dec 2018. URL https://github.com/OneraHub/XDSMjs. last checked: 21.01.2019.

[S11] Benedikt Aigner. VISualization TOol for MDAO Systems. Online tool, Dec 2018. URL http://mdo-system-interface.agile-project.eu/. last checked: 21.01.2019.

[S12] IRT Saint Exupéry. GEMS. Proprietary software, Feb 2019.

[S13] The MathWorks, Inc. MATLAB. Commercial software, Jan 2019. URL https://nl.mathworks.com. last checked: 23.01.2019.

[S14] Dassault Systèmes. Isight. Commercial software, Jan 2019. URL https://www.3ds.com/products-services/simulia/products/isight-simulia-execution-engine. last checked: 31.01.2019.

[S15] Siemens. HEEDS. Commercial software, May 2019. URL https://mdx.plm.automation.siemens.com/heeds. last checked: 28.05.2019.

[S16] ParaPy B.V. ParaPy. Commercial software, Feb 2019. URL https://www.parapy.nl. last checked: 08.02.2019.

[S17] IILS mbH. Design Compiler 43®. Commercial software, Feb 2019. URL https://www.iils.de/#page-top. last checked: 08.02.2019.

[S18] Dassault Systèmes. CATIA. Commercial software, Feb 2019. URL https://www.3ds.com/products-services/catia. last checked: 08.02.2019.

[S19] Imco van Gent. KADMOS. Open-source (Apache License 2.0) Bitbucket repository, May 2019. URL https://bitbucket.org/imcovangent/kadmos. last checked: 31.05.2019.

[S20] MDO lab. pyXDSM. Open-source (Apache License 2.0) GitHub repository, Feb 2019. URL https://github.com/mdolab/pyXDSM. last checked: 28.02.2019.

[S21] S. Dubreuil and R. Lafage. SSBJ-OpenMDAO. Open-source (Apache License 2.0) GitHub repository, Mar 2019. URL https://github.com/OneraHub/SSBJ-OpenMDAO. last checked: 04.03.2019.

[S22] I. van Gent. SSBJ-KADMOS. Open-source (Apache License 2.0) GitHub repository, Mar 2019. URL https://bitbucket.org/imcovangent/ssbjkadmos. last checked: 04.03.2019.

[S23] World Wide Web Consortium (W3C). eXtensible Markup Language (XML). Open standard, Mar 2019. URL https://www.w3.org/TR/REC-xml. last checked: 01.03.2019.

[S24] I. van Gent. CMDOWS. Open-source (Apache License 2.0) Bitbucket repository, May 2019. URL https://bitbucket.org/imcovangent/cmdows. last checked: 31.05.2019.

[S25] D. de Vries and I. van Gent. OpenLEGO. Open-source (Apache License 2.0) GitHub repository, Apr 2019. URL https://github.com/daniel-de-vries/OpenLEGO. last checked: 04.04.2019.

[S26] KE-works. KE-chain. Commercial software, March 2019. URL https://www.ke-chain.com/features. last checked: 07.03.2019.

[S27] NLR. SMR. Commercial software, May 2018. last checked: 01.05.2018.

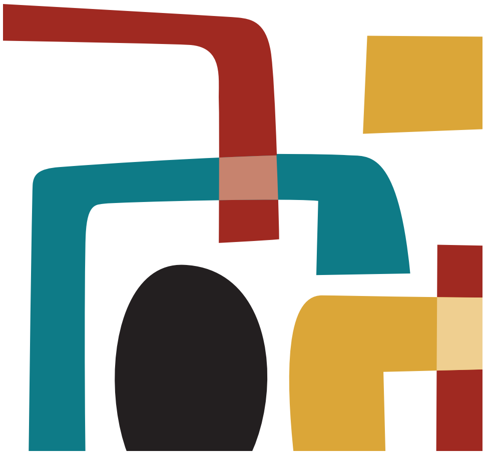
[S28] Noesis Solutions N.V. id8. Commercial software, Apr 2019. URL https://www.noesissolutions.com/our-products/id8. last checked: 17.04.2019.

[S29] R. Elemendorp. AVLwrapper - Python interface for AVL. Open-source (GNU General Public License V3) GitHub repository, May 2019. URL https://github.com/renoelmendorp/AVLWrapper. last checked: 14.05.2019.

[S30] M. Drela and H. Youngren. Athena Vortex Lattice (AVL). Open-access (GNU General Public License V2) download, May 2019. URL `http://web.mit.edu/drela/Public/web/avl/`. last checked: 14.05.2019.

V

Appendices



A

Additional architectures for the SSBJ case study

This appendix shows the XDSMs for different architectures imposed on the SSBJ optimization problem discussed in §3.6. The FPG is shown in Fig. 3.19. The architectures MDF with a Jacobi convergence scheme and IDF were shown in Fig. 3.20 and Fig. 3.21 respectively. Here, the XDSMs for MDF with a Gauss-Seidel convergence scheme (Fig. A.1), Collaborative Optimization (Fig. A.2), and BLISS-2000 (Fig. A.3) are provided for reference.

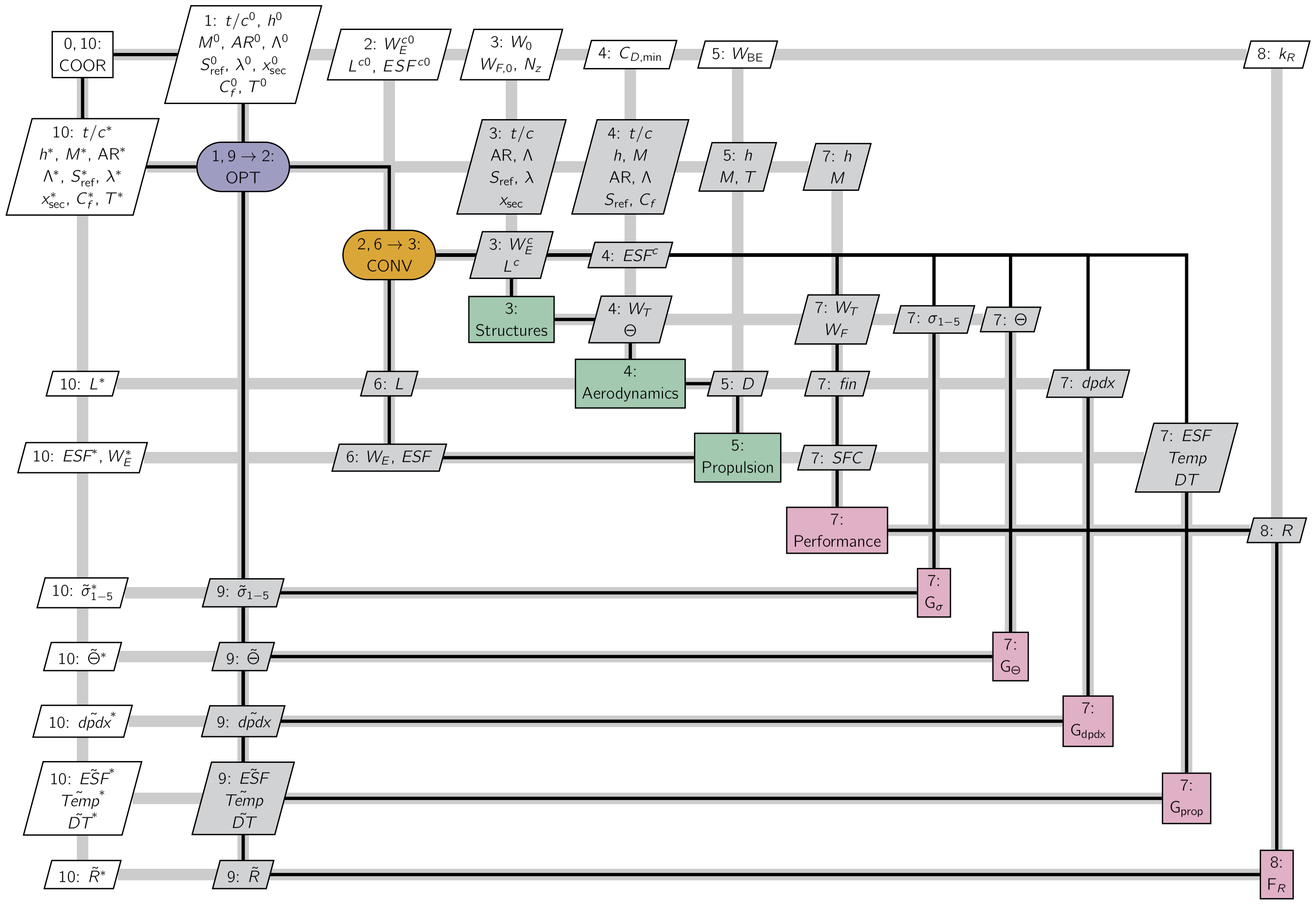

0, 10: COOR
1: t/c^0, h^0 M^0, AR^0, Λ^0 S^0_{ref}, λ^0, x^0_{sec} C^0_f, T^0
2: W_E^{c0} L^{c0}, ESF^{c0}
3: W_0 $W_{F,0}$, N_z
4: $C_{D,\text{min}}$
5: W_{BE}
8: k_R
10: t/c^* h^*, M^*, AR^* Λ^*, S^*_{ref}, λ^* x^*_{sec}, C^*_f, T^*
1, 9 → 2: OPT
3: t/c AR, Λ S_{ref}, λ x_{sec}
4: t/c h, M AR, Λ S_{ref}, C_f
5: h M, T
7: h M
2, 6 → 3: CONV
3: W_E^c L^c
4: ESF^c
3: Structures
4: W_T Θ
7: W_T W_F
7: σ_{1-5}
7: Θ
10: L^*
6: L
4: Aerodynamics
5: D
7: fin
7: $dpdx$
7: ESF $Temp$ DT
10: ESF^*, W_E^*
6: W_E, ESF
5: Propulsion
7: SFC
7: Performance
8: R
10: $\tilde{\sigma}^*_{1-5}$
9: $\tilde{\sigma}_{1-5}$
7: G_σ
10: $\tilde{\Theta}^*$
9: $\tilde{\Theta}$
7: G_Θ
10: $\tilde{dpdx}^*$
9: $\tilde{dpdx}$
7: G_{dpdx}
10: $\tilde{ESF}^*$ $\tilde{Temp}^*$ $\tilde{DT}^*$
9: $\tilde{ESF}$ $\tilde{Temp}$ $\tilde{DT}$
7: G_{prop}
10: $\tilde{R}^*$
9: $\tilde{R}$
8: F_R

Figure A.1: XDSM for the MDF with a Gauss-Seidel convergence scheme architecture

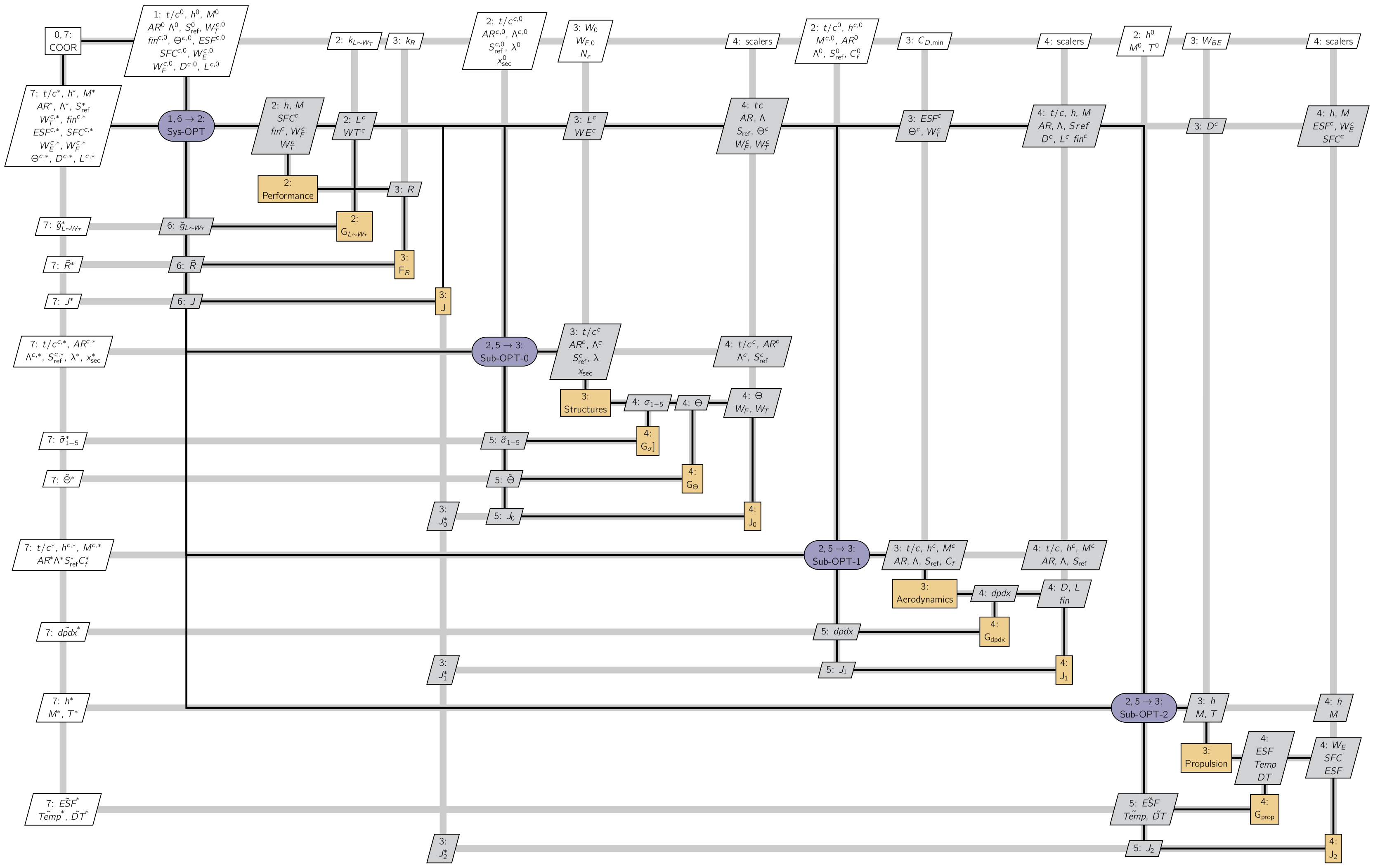
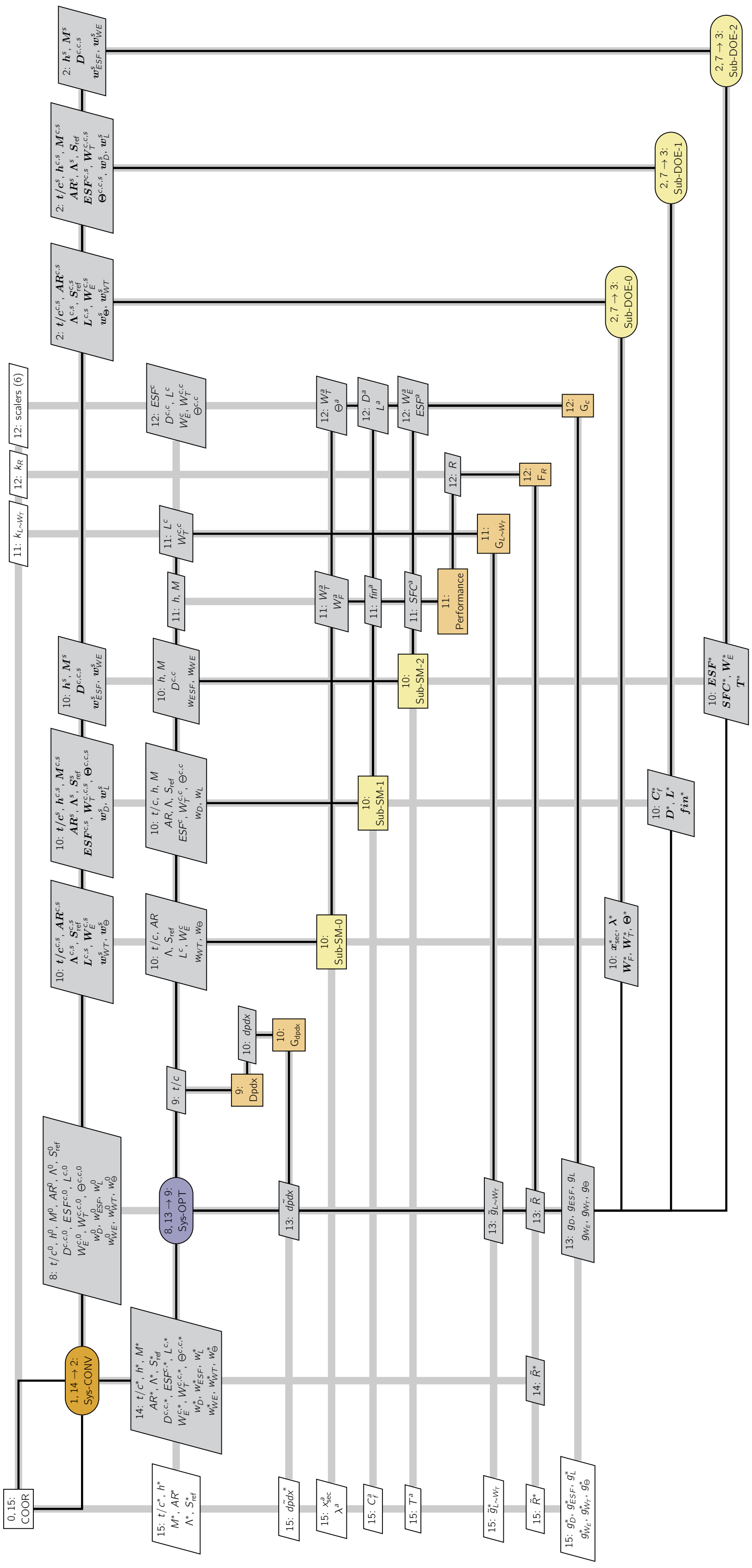
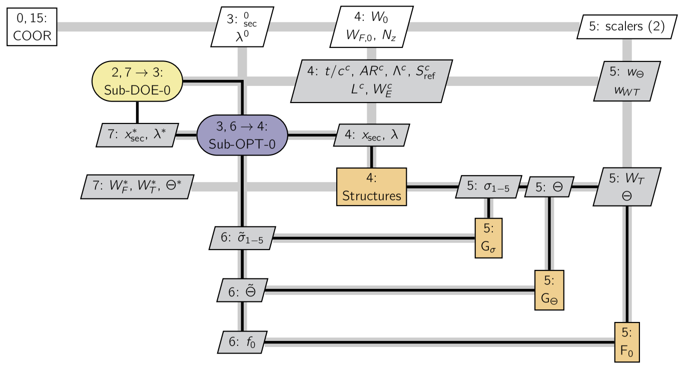

0, 7: COOR
$1{:}\ t/c^0, h^0, M^0, AR^0\ \Lambda^0, S^0_{ref}, W_T^{c,0}, fin^{c,0}, \Theta^{c,0}, ESF^{c,0}, SFC^{c,0}, W_E^{c,0}, W_F^{c,0}, D^{c,0}, L^{c,0}$
$2{:}\ k_{L\sim W_T}$
$3{:}\ k_R$
$2{:}\ t/c^{c,0}, AR^{c,0}, \Lambda^{c,0}, S^{c,0}_{ref}, \lambda^0, x^0_{sec}$
$3{:}\ W_0, W_{F,0}, N_z$
4: scalers
$2{:}\ t/c^0, h^{c,0}, M^{c,0}, AR^0, \Lambda^0, S^0_{ref}, C_f^0$
$3{:}\ C_{D,min}$
4: scalers
$2{:}\ h^0, M^0, T^0$
$3{:}\ W_{BE}$
4: scalers
$7{:}\ t/c^*, h^*, M^*, AR^*, \Lambda^*, S^*_{ref}, W_T^{c,*}, fin^{c,*}, ESF^{c,*}, SFC^{c,*}, W_E^{c,*}, W_F^{c,*}, \Theta^{c,*}, D^{c,*}, L^{c,*}$
$1,6 \rightarrow 2$: Sys-OPT
$2{:}\ h, M, SFC^c, fin^c, W_F^c, W_T^c$
$2{:}\ L^c, WT^c$
$3{:}\ L^c, WE^c$
$4{:}\ tc, AR, \Lambda, S_{ref}, \Theta^c, W_F^c, W_T^c$
$3{:}\ ESF^c, \Theta^c, W_T^c$
$4{:}\ t/c, h, M, AR, \Lambda, Sref, D^c, L^c, fin^c$
$3{:}\ D^c$
$4{:}\ h, M, ESF^c, W_E^c, SFC^c$
2: Performance
$3{:}\ R$
$2{:}\ G_{L\sim W_T}$
$3{:}\ F_R$
$3{:}\ J$
$7{:}\ \tilde{g}^*_{L\sim W_T}$
$6{:}\ \tilde{g}_{L\sim W_T}$
$7{:}\ \tilde{R}^*$
$6{:}\ \tilde{R}$
$7{:}\ J^*$
$6{:}\ J$
$7{:}\ t/c^{c,*}, AR^{c,*}, \Lambda^{c,*}, S^{c,*}_{ref}, \lambda^*, x^*_{sec}$
$2,5 \rightarrow 3$: Sub-OPT-0
$3{:}\ t/c^c, AR^c, \Lambda^c, S^c_{ref}, \lambda, x_{sec}$
$4{:}\ t/c^c, AR^c, \Lambda^c, S^c_{ref}$
3: Structures
$4{:}\ \sigma_{1-5}$
$4{:}\ \Theta$
$4{:}\ \Theta, W_F, W_T$
$7{:}\ \tilde{\sigma}^*_{1-5}$
$5{:}\ \tilde{\sigma}_{1-5}$
$4{:}\ G_{\sigma}]$
$7{:}\ \tilde{\Theta}^*$
$5{:}\ \tilde{\Theta}$
$4{:}\ G_{\Theta}$
$3{:}\ J_0^*$
$5{:}\ J_0$
$4{:}\ J_0$
$7{:}\ t/c^*, h^{c,*}, M^{c,*}, AR^* \Lambda^* S^*_{ref} C_f^*$
$2,5 \rightarrow 3$: Sub-OPT-1
$3{:}\ t/c, h^c, M^c, AR, \Lambda, S_{ref}, C_f$
$4{:}\ t/c, h^c, M^c, AR, \Lambda, S_{ref}$
3: Aerodynamics
$4{:}\ dpdx$
$4{:}\ D, L, fin$
$7{:}\ d\tilde{p}dx^*$
$5{:}\ dpdx$
$4{:}\ G_{dpdx}$
$3{:}\ J_1^*$
$5{:}\ J_1$
$4{:}\ J_1$
$7{:}\ h^*, M^*, T^*$
$2,5 \rightarrow 3$: Sub-OPT-2
$3{:}\ h, M, T$
$4{:}\ h, M$
3: Propulsion
$4{:}\ ESF, Temp, DT$
$4{:}\ W_E, SFC, ESF$
$7{:}\ E\tilde{S}F^*, T\tilde{e}mp^*, \tilde{DT}^*$
$5{:}\ E\tilde{S}F, T\tilde{e}mp, \tilde{DT}$
$4{:}\ G_{prop}$
$3{:}\ J_2^*$
$5{:}\ J_2$
$4{:}\ J_2$

Figure A.2: XDSM for the CO architecture

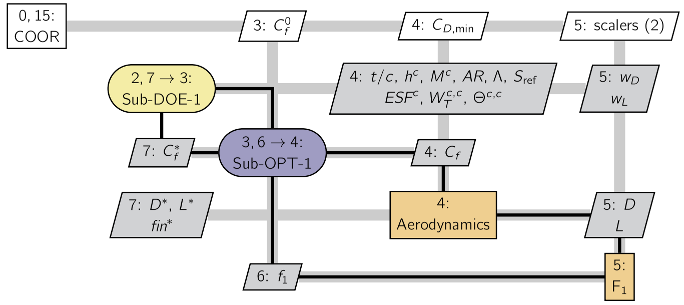


a) System-level BLISS-2000 approach

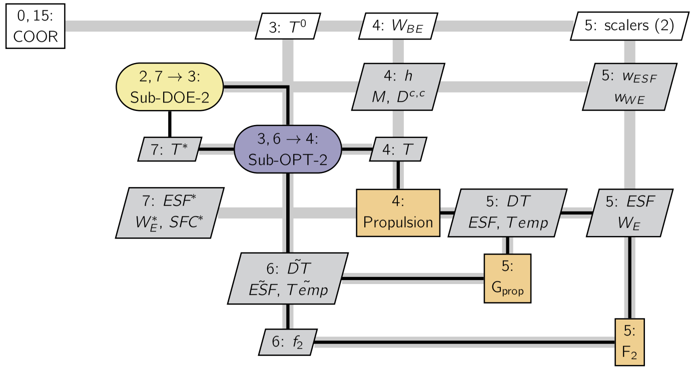


0, 15: COOR
3: x^0_{sec} λ^0
4: W_0 $W_{F,0}$, N_z
5: scalers (2)
2, 7 → 3: Sub-DOE-0
4: t/c^c, AR^c, Λ^c, S^c_{ref} L^c, W^c_E
5: w_Θ w_{WT}
7: x^*_{sec}, λ^*
3, 6 → 4: Sub-OPT-0
4: x_{sec}, λ
7: W^*_F, W^*_T, Θ^*
4: Structures
5: σ_{1-5}
5: Θ
5: W_T Θ
6: $\tilde{\sigma}_{1-5}$
5: G_σ
6: $\tilde{\Theta}$
5: G_Θ
6: f_0
5: F_0

b) Sub-level 0 BLISS-2000



c) Sub-level 1 BLISS-2000



d) Sub-level 2 BLISS-2000

Figure A.3: XDSM for the BLISS-2000 architecture

ACKNOWLEDGMENTS

Although working on a PhD project might seem like a solitary endeavor, I could never have completed it on my own. I'm indebted to a long list of people in many different ways for the past years of having fun researching, reading, writing, traveling, presenting, and so much more.

First of all, I should thank the person that brought the interesting PhD position on MDAO at my alma mater to my attention: Reinier. Thanks for responding to a request from a relative stranger four years ago and for being a mentor throughout the process with helpful advice, brainstorms, and critical remarks over a cup of coffee.

I'm also thankful for my supervisory team: Gianfranco and Leo. First of all for hiring me, an aerospace engineer who wandered off into the offshore engineering world, and bringing me back in the realms of aerospace research and development. And secondly for supporting me in my daily work by staying critical of all my output, suggesting new approaches, and enthusiastically supporting my research on every front. I want to specifically thank Gianfranco for his feedback on all the writing I have done. I believe your tireless reviewing effort took our publications and this dissertation to a higher level. I'd like to thank my committee members for the time and effort they took to read my draft dissertation, provide me with valuable feedback, and initialize fruitful discussions.

My research was financed by the European Union through the AGILE project. Thereby it was supported by any tax-paying citizen of the union, and I'm grateful to anyone that is supportive of the EU and has worked (or even fought) to achieve this ridiculously ambitious "collaborative framework" that is the European Union. Especially in times like these, I feel it is important to stress the positive message of opportunities that arise when we work together and prioritize international collaboration over national interests.

To be a bit more specific, I want to thank all the participants of the AGILE project, which provided me with such an inspiring and valuable environment to do my research. You are too many to name individually, so I have to limit myself to mere highlights. Pier, thank you for being such a great manager of the project and creating an environment to collaborate so successfully with all participants. Bastiaan and Benedikt, it was a pleasure to develop core components of our very own next-generation MDAO framework with you. Thanks for all the brainstorm, debug, and support sessions we've spent together. My work would not have been what it came to be without your contributions to the project and your effort and eagerness to bring it all together. I also want to thank the various participants that so enthusiastically (and maybe naively) took up the challenge to test my developments within their own platforms and/or design projects and provided me with valuable feedback: Riccardo, Luca, Pierluigi, Dominique, Aidan, Francesco, Prajwal, Pier, and Thierry, and so many more. Your enthusiasm and patience for the sake of science has been a humbling and motivating experience for me. While mentioning Luca and Pierluigi, I also have to thank them and Fabrizio for meeting my dietary requirements for the Naples meetings and successfully optimizing the number of Napolitan pizze, espressi, and babas. Grazie mille!

Not only Italy played a key culinary role in my research, France is also worth mentioning. Nathalie, thank you for inviting me for an exchange period at ONERA and your trust in and support of my work. My stay in Toulouse was perfectly timed and allowed me to have a period of complete focus when I needed it most. Also merci beaucoup Thierry, for sharing your office and minion army with me, and teaching me valuable French expressions that one does not encounter in the dictionary. Sylvain and Sebastian, thanks for making me feel welcome in Toulouse and the surrounding skiing areas. Also, thank you Joseph for inviting me back to present my work in your MDAO course and allowing me to once again test the rich multidisciplinary culinary traditions of France.

In my four years as PhD candidate, I've also had the pleasure to collaborate with multiple students and supervise some of them during their graduation projects. Andreas was there when the first lines of code were written for the KADMOS package. Thanks for your contribution and the nice discussions we had on what KADMOS should and shouldn't be. Daniël and Robin have been a huge help during my effort to 'CPACSize' the TU Delft tools. Lukas was a joy to work with and implemented an initial version of CMDOWS like it was his daily business. Maaike was a tour-de-force in documenting and cleaning the code bases, which grew out of hand very quickly. And after Daniël was done CPACSizing, he developed the first version of OpenLEGO and singlehandedly connected a European and American MDAO research framework. Finally, Anne-Liza took up challenging topics that risked becoming out-of-scope of my own work and did a fantastic job tackling them. I'm indebted to all these students and grateful for the rigor and enthusiasm with which they took up the tasks in front of them.

My period as a PhD candidate would not have been so enjoyable without so many fun and interesting colleagues. I will never forget my first proper hike during our Salzburg trip, climbing the Untersberg on espadrilles! Thanks hike masters Tomas, Tom, Nando, Toni, Salvo, and Reynard for not leaving an unprepared hiker behind. Another amazing trip took me all the way to India to attend the wedding of my dear friend and office mate Akshay, a.k.a. "The Diesel Engine". Thanks Akshay and Spoorti for having us and showing us what Indian hospitality and weddings are all about. Back at the office, I'm also indebted to Fabrizio for the occasional much-needed caffeine shot that happily dripped out of his Nespresso machine, accompanied by an exchange of recommendations on where to go in Rotterdam and on Sicily. There are many more activities to be named, such as jazz concerts with Toni, bike rides with Nitish, more group trips abroad, and of course many many beers that would not have been the same without your company. Thanks for that Toni, Akshito, Salvo, Tomas, Tom, Nando, Reynard, Andre, Adam, Tambe-G, Biagio, Sebastian, Marco, Matteo, Maurice, Reno, Sonia, and Lucia.

Despite so much help and support, working on a PhD project still leads to some periods of stress and frustrations. Luckily there was the weekly A.C. Brancaleone football match to get rid of these. Here is not the place to comment on whether these emotions simply evaporated though physical activity, or were merely replaced by the stress and frustrations on our team's performance. Let's just say that it's a good thing us Brancaleone players pursued careers in research rather than in professional sports... Regardless, thanks a million Aquilante, Bastone, Blanco, Guntharius, Scusi, Sinigaglia, d'Arhnem, Bastiaan, Marcel, Tje, Oliviero, Juan, LeBlanc, Buri, Zeno, and many weekly substitutes for being such good sports, before and after the many matches we played.

I'm also lucky to have many long-term friends that have supported me along the road. Vincent, who pursued his own PhD on the other side of the world, was one of the best to

discuss common struggles and I'm very grateful we could meet so many times on either side of the Atlantic to exchange our experiences. Bob has been a constant support at our base lunch station Il Tartufo. Marcin & Haley, thanks for many wonderful evenings and trips, and for letting me obtain one of the most honorable titles imaginable by making me the offer I could never refuse to be the proud godfather of Austin. Sascha & Marius, thanks for your enthusiasm and so many negronis. Ferdi, being your best man was one of the highlights of the past four years for me. Marjo & Aart, having you over in Toulouse was a match made in heaven, even the weather gods strongly agreed on that. Also thanks to the "Rotterdam Art Crew" (Niek, Riri, The Karlenno's, Milou) for many evenings of philosophical considerations, which inspired one of my propositions, and demonstrating my personal limits through pisco sours (never again!) and old fashioneds. Riri, thanks for being such a loyal friend and for sharing so many good movies, food, and cultural trips.

I would not have the person I am today without the support of my family. I would like to thank my parents for raising me to always pursue the best possible education I could find, giving me freedom to develop myself, and supporting me throughout the many years of study I enjoyed before eventually starting my PhD research. I'm also grateful for having such a fantastic brother and sister, who freed my mind from its PhD walls with the occasional cultural or sportive activity. And Melvin, also thanks a lot for your practical advice and help with front-ending the online KADMOS interface. You've become a true programming mastermind.

I also owe a lot to my inlaws, Jos & Lorraine and Hans & Petra. Jos, thanks for the many inspirational wines and the occasional cigar accompanied with much required and invaluable moral support. Also thanks for editing the Dutch translations in this book. Lorraine, thanks for your emotional support and for fueling my research endeavor with your many culinary creations that kept my spirits high (also many thanks for ironing so many shirts for every meeting and conference I had to attend). I also like to thank Hans for his invaluable advice on the lay-out of this book. Your cover design perfectly summarizes my research as an abstract work of art.

Last, but furthest from least, I thank mia bella principessa Nina. If I'd had to fully explain why, I would need to write another book (and I'm not planning on writing any more books for a long time). Nina, you have been my pillar of joie-de-vivre, my infinite moral and emotional support, and I consider myself the luckiest man on the planet to have shared these past four research years with you by my side.

Curriculum Vitae

Imco van Gent

20-8-1988	Born in Rotterdam, the Netherlands

Research and Engineering

2015–2019 **PhD in multidisciplinary design analysis and optimization**
Delft University of Technology, Delft, the Netherlands

thesis	AGILE MDAO Systems - A Graph-based Methodology to Enhance Collaborative Multidisciplinary Design
promotor	prof. dr. ir. L. L. M. Veldhuis
copromotor	dr. ir. G. La Rocca

2013–2015 **Project and Lead Engineer**
Temporary Works Design B.V., Rotterdam, the Netherlands

Education

2015–2019 **Doctoral Education Program**
Delft University of Technology, Delft, the Netherlands

2010–2012 **MSc in Aerospace Engineering**
Delft University of Technology, Delft, the Netherlands

thesis	Cost-Weight Trades for Modular Composite Structures
supervisor	prof. dr. C. Kassapoglou

2009–2010 **Honours Track in Philosophy of Science**
Leiden University, Leiden, the Netherlands

2006–2009 **BSc in Aerospace Engineering**
Delft University of Technology, Delft, the Netherlands

minor	Aerospace System Design and Technology

2000–2006 **Pre-University Education**
Emmauscollege, Rotterdam, the Netherlands

List of Publications

Journal Papers

5. van Gent, I., Aigner, B., Beijer, B., Jepsen, J. and La Rocca, G., “Knowledge Architecture Supporting the Next Generation of MDO in the AGILE Paradigm,” *Progress in Aerospace Sciences*, (accepted for publication).

4. van Gent, I. and La Rocca, G., “Formulation and Integration of MDAO Systems for Collaborative Design: A Graph-based Methodological Approach,” *Aerospace Science and Technology*, Vol. 90, pp. 410–433, 2019. doi: 10.1016/j.ast.2019.04.039

3. van Gent, I., La Rocca, G. and Hoogreef, M.F.M., “CMDOWS: A Proposed New Standard to Store and Exchange MDO systems”, *CEAS Aeronautical Journal*, Vol. 9, Iss. 4, pp. 607-627, 2018. doi: 10.1007/s13272-018-0307-2

2. Aigner, B., van Gent, I., La Rocca, G., Stumpf, E. and Veldhuis, L.L.M., “Graph-based Algorithms and Data-driven Documents for Formulation and Visualization of Large MDO Systems”, *CEAS Aeronautical Journal*, Vol. 9, Iss. 4, pp. 695-709, 2018. doi: 10.1007/s13272-018-0312-5

1. van Gent, I. and Kassapoglou, C., “Cost-weight Trades for Modular Composite Structures,” *Structural and Multidisciplinary Optimization*, Vol. 49, Iss. 6, pp. 931–952, 2014. doi: 10.1007/s00158-013-1019-1

Conference Papers

10. Ciampa, P.D., Prakasha, P.S., Torrigiani, F., Walther, J., Lefebvre, T., Bartoli, N., Timmermans, H., Della Vecchia, P., Rajpal, D., van Gent, I., La Rocca, G., Fioriti, M., Cerino, G., Maierl, R., Charbonnier, D., Jungo, A., Aigner, B., Anisimov, K., Mirzoyan, A. and Voskuijl, M., “Streamlining Cross-organizational Aircraft Development: Results from the AGILE Project.” *AIAA Aviation 2019 Forum*, AIAA Paper 2019-3454, June 2019. doi: 10.2514/6.2019-3454

9. Lombardi, R., van Gent, I. and La Rocca, G., “Reconfigurable Formulation and Implementation of MDAO Systems,” *NAFEMS World Congress 2019*, June 2019.

8. van Gent, I., Aigner, B., Beijer, B. and La Rocca, G., “A Critical Look at Design Automation Solutions for Collaborative MDO in the AGILE Paradigm,” *2018 Multidisciplinary Analysis and Optimization Conference*, AIAA Paper 2018-3251, June 2018. doi: 10.2514/6.2018-3251

7. Lefebvre, T., Bartoli, N., Dubreuil, S., Panzeri, M., Lombardi, R., Lammen, W., Zhang, M., van Gent, I. and Ciampa, P.D., “A Clustered and Surrogate-based MDA Use Case for MDO scenarios in AGILE Project,” *2018 Multidisciplinary Analysis and Optimization Conference*, AIAA Paper 2018-3252, June 2018. doi: 10.2514/6.2018-3252

6. Prakasha, P.S., Ciampa, P.D., Della Vecchia, P., Aigner, B. and van Gent, I., “MDO Framework for University Research Collaboration: AGILE Academy Initiatives & Outcomes,” *2018 Multidisciplinary Analysis and Optimization Conference*, AIAA Paper 2018-3254, June 2018. doi: 10.2514/6.2018-3254

5. van Gent, I., La Rocca, G. and Hoogreef, M.F.M., “CMDOWS: A Proposed New Standard To Store And Exchange MDO Systems,” *6th CEAS Air and Space Conference*, CEAS Paper 969, October 2017.

4. Aigner, B., van Gent, I., La Rocca, G., Stumpf, E. and Veldhuis, L.L.M., “Graph-based Algorithms and Data-driven Documents for Formulation and Visualization of Large MDO Systems,” *6th CEAS Air and Space Conference*, CEAS Paper 173, October 2017.

3. van Gent, I., Lombardi, R., La Rocca, G. and d’Ippolito, R., “A Fully Automated Chain from MDAO Problem Formulation to Workflow Execution,” *EUROGEN 2017*, September 2017.

2. van Gent, I., Ciampa, P.D., Aigner, B., Jepsen, J., La Rocca, G. and Schut, J., “Knowledge Architecture Supporting the Collaborative MDO in the AGILE Paradigm,” *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, AIAA Paper 2017-4139, June 2017. doi: 10.2514/6.2017-4139

1. van Gent, I., La Rocca, G., Veldhuis, L.L.M., “Composing MDAO Symphonies: Graph-based Generation and Manipulation of Large Multidisciplinary Systems,” *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, AIAA 2017-3663, June 2017.