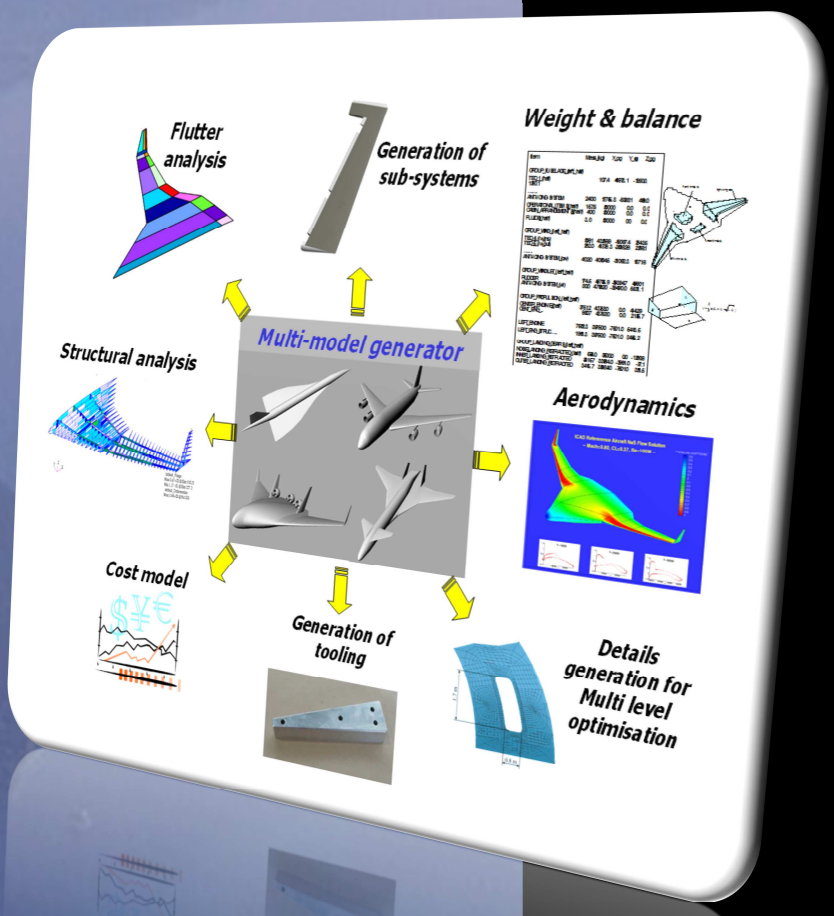
Knowledge Based Engineering Techniques to Support Aircraft Design and Optimization

Gianfranco La Rocca



Knowledge Based Engineering Techniques to Support Aircraft Design and Optimization

Gianfranco La Rocca

Considerate la vostra semenza:
fatti non foste a viver come bruti,
ma per seguir virtute e canoscenza

Dante Alighieri

Ulysses' last journey, Inferno, Canto XXVI

Knowledge Based Engineering Techniques to Support Aircraft Design and Optimization

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.Ch.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op vrijdag 1 april 2011 om 10.00 uur

door

Gianfranco LA ROCCA

Ingegnere Aerospaziale, Universita' di Pisa
geboren te Messina, Italië

Dit proefschrift is goedgekeurd door de promotor:

Prof.dr.ir. M.J.L. van Tooren

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr.ir. M.J.L. van Tooren	Technische Universiteit Delft, promotor
Prof.dr. A.J. Morris	Cranfield University
Prof.dr. A. Frediani	Universita' di Pisa
Prof.ir. J.J. Hopman	Technische Universiteit Delft
Prof.dr. T. Tomiyama	Technische Universiteit Delft
Prof.dr.ir. E. Torenbeek	Technische Universiteit Delft
Prof.dr. R. Curran	Technische Universiteit Delft

ISBN/EAN: 978-90-9026069-3

Cover picture: Isola d'Elba from the sky.

Printed in the Netherlands.

Summary

Knowledge Based Engineering Techniques to Support Aircraft Design and Optimization

Since the 1960s, the demand for air transportation has doubled every 15 years, resilient to every oil crises and international events. However, the current capability of the air transport management system, the demand of increasingly growing levels of quality, comfort, safety and security, and, above all, an environmental sensitivity as high as never before, seem to constrain any further growth. The Advisory Council for Aeronautical Research in Europe (ACARE), similarly to NASA in the United States, has indicated a set of challenging objectives and devised a roadmap to help the aerospace industry stepping into a new age of sustainable growth.

However, major technological advances will not be possible without significant improvements to the current design methodology. In this regard, the present research work aims at the **development of new design methods and tools** that are able to sustain the evolutionary improvement of current aircraft designs, as well as to support the investigation of novel aircraft configurations. To be successful, these design methods and tools must be able to facilitate the aircraft development process as it is currently carried across large and distributed supply chains. Besides, they must account for the increasing scarcity of intellectual resources and the consequent need to increase engineers' productivity and freeing time for innovation.

The **Multidisciplinary Design Optimization** (MDO) approach appears to be the most promising design methodology in the field of aircraft design, both to improve the performance of traditional aircraft configurations and to support the development of novel concepts. However, a number of technical and non-technical barriers have prevented full exploitation of the MDO approach and, so far, limited its industrial application to detail design cases.

To this purpose, the concept of **Design and Engineering Engine (DEE)** has been developed at the Faculty of Aerospace Engineering in Delft, which is a modular, loosely integrated design system able to support distributed multidisciplinary analysis optimization by automating as far as possible the repetitive and non creative activities that hamper the design and analysis process. One of the DEE technology enabler is the Multi Model Generator (MMG), which actually represents the main outcome of this research work. **The Multi-Model Generator (MMG) is a**

Knowledge Based Engineering (KBE) application developed with the twofold intent of 1) providing designers with a parametric modeling environment to define generative models of conventional and novel aircraft configurations and 2) feeding various analysis tools with dedicated aircraft model abstractions, as required for the verification of the generated design. To meet these objectives, two types of functional blocks have been developed, which constitute the main ingredients of the MMG: the High Level Primitives (HLPs) and the Capability Modules (CMs).

Four **High Level Primitives** have been defined, namely Wing-part, Fuselage-part, Engine and Connection-element. These can be figured out as a suite of advanced LEGO blocks that designers can manipulate to assemble the geometry (external surfaces and structural layout) of the aircraft concept they have in mind. Each HLP has been programmed as a class using the **object-oriented programming language** of the employed KBE system. This has allowed capturing the design rules that give the HLPs the capability to automatically adapt their own shape and topology, or to trigger events as a reaction of input changes. By means of the editable MMG input file, designers can assign different values to the attributes of each HLP class and call for multiple HLPs instantiations. In this way, **both conventional and novel aircraft configurations can be automatically generated** and then stretched/morphed into an infinite amount of variants.

During the conceptual design phase, designers "see" the aircraft as an assembly of basic solutions to fulfill functionalities, such as generate lift and accommodate payload, rather than an assembly of points, curves, surfaces and solid features. The capabilities to support the designer' functional thinking and capture knowledge in terms of design rules, have yielded the MMG primitives the "high level" connotation, in contrast with the "low level" primitives of conventional CAD.

Once the model of the given aircraft is available, the preparation for the verification phase starts, which requires the set up of the various discipline abstractions (or views) that must be fed to the analysis tools. In the traditional design process, the preparation of these disciplinary models is acknowledged to be lengthy and repetitive, particularly when high fidelity analysis tools are involved. Up to 80% of the overall design process can be wasted just for these preprocessing activities. However, it has been observed that 1) independently from the aircraft configuration at hand, the same analysis tools and preprocessing methods are generally used by specialists; 2) large part of the preprocessing activities is rule-based and require a large deal of geometry manipulation, which actually represent the strengths of KBE technology. To support this phase of the design process, a set of **Capability Modules** (CM) has been developed to capture the "model preprocessing knowledge" of discipline experts and reuse it to **automate the generation of models** for a

broad range of low and high fidelity analysis tools, both proprietary and commercial off the shelf.

The implemented approach has enabled the use of **high fidelity analysis tools**, such as FEM and CFD, already **in the early stages of the design process**, which not only increases the level of confidence in the designed product, but provides essential means for the study of innovative aircraft configurations, where semi-empirical and statistics based methods fail and first principle analysis is the only way to go.

Due to its ability to be accessed in remote, via web connections, and operated in batch, the MMG also demonstrated to be a valuable asset to **support MDO processes across distributed design frameworks**.

The capability of the MMG has been demonstrated by means of several example applications and two relevant study cases addressed in this work. The first case concerns with the European project MOB, on distributed multidisciplinary design optimization of blended wing body aircraft configurations. The second deals with a MDO system developed in collaboration with Airbus to redesign the vertical tail of an existing passenger aircraft.

A side objective of this work was to improve the dissemination of KBE technology, which is still a relatively young discipline that has not yet found the deserved level of attention and understanding, both in the world of industry and academia. To this scope, an extensive and original **investigation on the Artificial Intelligence roots of KBE** is provided and its object oriented paradigm thoroughly discussed. A **best practice** section to the development of KBE applications is included as well.

Samenvatting

Knowledge Based Engineering technieken ter ondersteuning van vliegtuigontwerp en optimalisatie

De vraag naar luchtvervoer is elke 15 jaar verdubbeld sinds de 60-er jaren, altijd herstellend van elke olie crises en internationale evenementen. Echter, de huidige capaciteit van het luchttransportmanagementsysteem, de vraag van de steeds groeiende kwaliteit, comfort, veiligheid en beveiliging, en vooral een milieubewustheid zo hoog als nooit tevoren, lijkt een beperking te zijn van elke verdere groei. De Advisory Council for Aeronautical Research in Europe (ACARE), vergelijkbaar met NASA in de Verenigde Staten, heeft een aantal uitdagende doelstellingen aangegeven en heeft een routekaart opgezet om de luchtvaartindustrie te helpen met de entree in de nieuwe era van duurzame groei.

Echter, grote technologische vooruitgang zal niet mogelijk zijn zonder aanzienlijke verbeteringen aan de huidige ontwerp-methodologie. In dit verband richt het huidige onderzoek zich op de **ontwikkeling van nieuwe ontwerpmethoden en –tools** die in staat zijn evolutionaire verbetering van de huidige vliegtuigontewerpen kunnen ondersteunen, maar ook ondersteuning kunnen leveren aan het onderzoek naar nieuwe vliegtuigconfiguraties. Om succesvol te zijn, moeten deze ontwerpmethoden en –tools het proces van de vliegtuigontwikkeling faciliteren, aangezien dit nu over grote en verdeelde leveringsketens gaat. Bovendien moeten deze rekening houden met de toenemende schaarste van intellectuele resources en de daaruit voortvloeiende behoefte om de productiviteit van de ingenieurs te verhogen en tijd vrij te maken voor innovatie.

De **Multidisciplinaire Design Optimalisatie** (MDO) benadering lijkt de meest veelbelovende ontwerpmethode te zijn op het gebied van vliegtuigontwerp, zowel voor het verbeteren van de prestaties van traditionele vliegtuigconfiguraties, alsmede voor de ondersteuning van de ontwikkeling van innovatieve concepten. Echter, een aantal technische (en niet-technische) belemmeringen hebben een volledige toepassing van de MDO-aanpak verhinderd en hebben tot nu toe de industriële toepassing gelimiteerd tot gedetailleerde ontwerptoepassingen.

Voor dit doel is het concept van **Design en Engineering Engine (DEE)** ontwikkeld bij de faculteit van luchtvaart techniek in Delft, dat een modulair, losjes geïntegreerd ontwerpsysteem is, dat in staat is om verdeelde multidisciplinaire analyse-

optimalisatie te ondersteunen, door de repetitieve en niet creatieve activiteiten die het proces belemmeren zoveel mogelijk te automatiseren.

Eén van de DEE hoekstenen is de Multi Model Generator (MMG), welke het voornaamste resultaat van dit onderzoekswerk is. **De Multi-Model Generator (MMG) is een Knowledge Based Engineering (KBE)** applicatie ontwikkeld met de tweeledige bedoeling om 1) designers te voorzien van een parametrische modelleeromgeving waarin generatieve modellen van conventionele en innovatieve vliegtuigconfiguraties kunnen worden gedefinieerd en 2) het voeden van verschillende analysetools met specifieke abstracties van vliegtuigmodellen, zoals vereist voor de verificatie van het gegenereerde ontwerp. Om deze doelstellingen te verwezenlijken zijn twee soorten functioneel blokken ontwikkeld, welke de voornaamste ingrediënten van de MMG vormen: de High Level Primitives (HLPs) en de Capability Modules (CMs).

Vier **High Level Primitives** zijn gedefinieerd, namelijk Vleugel-deel, Rompdeel, Motor en Verbinding-element. Deze kunnen worden ingeschat als een reeks geavanceerde LEGO blokken die ontwerpers kunnen manipuleren om de geometrie (externe oppervlakken en structurele lay-out) samen te stellen van het concept van de vliegtuigen die ze in gedachten hebben. Elke HLP is geprogrammeerd als een klasse met de **object-georiënteerde programmeertaal** van het gebruikte KBE systeem. Dit heeft mogelijk gemaakt om de ontwerpregels vast te leggen, die de HLPs het vermogen geven hun eigen vorm en topologie automatisch aan te passen, of om te reageren met veranderingen van invoer. Door middel van het bewerkbare MMG invoerbestand kunnen designers verschillende waarden aan de kenmerken van elke HLP-klasse toewijzen en meerdere HLP instanties genereren. Op deze manier kunnen zowel conventionele als innovatieve vliegtuigconfiguraties automatisch worden gegenereerd en vervolgens worden uitgerekt in een oneindige hoeveelheid varianten.

Tijdens de conceptuele ontwerpfase “zien” de ontwerpers het vliegtuig als een samenstelling van fundamentele oplossingen van te vervullen functies zoals het genereren van lift en het onderbrengen van lading, in plaats van een samenstelling van punten, krommen en oppervlakken. De mogelijkheid om het functionele denken van de designer te ondersteunen en kennis in termen van ontwerpregels te vangen, hebben de MMG primitieven de "hoog niveau" bijbetekenis opgeleverd, in tegenstelling tot de "laag niveau" van conventionele CAD primitieven.

Zodra het model van het gegeven vliegtuig beschikbaar is, start de voorbereiding voor de verificatie fase welke de set-up vereist van de verschillende discipline-abstracties (of weergaven) die aan de analyse-instrumenten moet worden gevoerd. In het traditionele ontwerpproces is de voorbereiding van deze disciplinaire modellen erkend als langdurig en repetitief, in het bijzonder wanneer er high-fidelity analyse

tools betrokken zijn. Alleen al voor deze preprocessing activiteiten kan er tot 80% van het totale ontwerpproces worden verspild. Echter, men heeft opgemerkt dat 1) onafhankelijk van de betreffende vliegtuigconfiguratie, over het algemeen worden dezelfde analysis tools en preprocessing methodieken gebruikt door de specialisten; en 2) een groot deel van de preprocessing activiteiten volgen bepaalde regels en vereisen een grote hoeveelheid geometrie manipulatie, die eigenlijk de sterke punten van KBE technologie vertegenwoordigen. Ter ondersteuning van deze fase van het ontwerpproces is een set van **Capability Modules** (CM) ontwikkeld om de "model preprocessing kennis" te vangen van de discipline-deskundigen en dit te gebruiken voor het **automatiseren van het genereren van modellen** voor een breed scala van low- en high-fidelity analyse-instrumenten, zowel in-house developed als commercial-off-the-shelf.

De geïmplementeerde aanpak heeft het gebruik van **high-fidelity analyse tools**, zoals FEM en CFD, al in de vroege stadia van het ontwerpproces mogelijk gemaakt. Deze verhogen niet alleen het niveau van vertrouwen in het ontworpen product, maar levert een essentieel instrument voor de studie van innovatieve vliegtuigconfiguraties waar semi-empirische en op statistieken gebaseerde methoden mislukken en een first-principle analyse is de enige oplossing.

Vanwege de mogelijkheid om vanaf afstand toegankelijk te zijn, via web verbindingen, en in batch te bedienen, heeft de MMG laten zien een waardevolle aanwinst te zijn voor **ondersteuning van MDO processen over verdeelde design frameworks**.

De bekwaamheid van de MMG is gedemonstreerd door diverse voorbeeld toepassingen en twee relevante studie casussen waarnaar wordt verwezen in dit werk. De eerste betreft met het Europese project MOB, over verdeelde multidisciplinaire design optimalisatie van blended wing body vliegtuigconfiguraties. De tweede gaat over een MDO systeem dat is ontwikkeld in samenwerking met Airbus om het verticale staartvlak van een bestaand passagiersvliegtuig te herontwerpen.

Een nevendoelstelling van dit werk was het verbeteren van de verspreiding van KBE technologie, die nog steeds een relatief jonge discipline is welke de verdiende aandacht en het niveau van begrip nog niet heeft gevonden, noch in de wereld van de industrie, noch in de academisch wereld. Met dit doel wordt een uitgebreid en origineel **onderzoek naar de wortels van de kunstmatige intelligentie van KBE** geleverd en zijn object georiënteerde paradigma wordt uitvoerig besproken. Een **best practice** sectie met betrekking tot de ontwikkeling van KBE toepassingen is ook opgenomen.

Sommario

Tecniche Knowledge Based Engineering per sostenere la progettazione e l'ottimizzazione di velivoli

A partire dagli anni sessanta, la domanda di trasporto aereo ha continuato a raddoppiare ogni 15 anni, reagendo in maniera flessibile alle varie cirisi petrolifere e situazioni di crisi internazionale. Tuttavia, i limiti dell'attuale capacità del sistema di gestione del trasporto aereo, la crescente domanda di livelli di qualita', comfort, sicurezza e protezione sempre piu' alti, e soprattutto, un livello di sensibilita' ambientale mai cosi elevato sembrano vincolarne ogni ulteriore crescita. L'ACARE (Advisory Council for Aeronautical Research in Europe), l'organo di avviso per la ricerca aeronautica in Europa, e similmente la NASA negli Stati Uniti, hanno indicato una serie di obiettivi molto ambiziosi e messo a punto una tabella di marcia per portare l'industria aerospaziale verso una nuova era di crescita sostenibile.

Tuttavia, il raggiungimento di importanti avanzamenti tecnologici rimarra' molto difficile, fino a che le attuali metodologie di progettazione non verranno anch'esse adattate e migliorate in maniera significativa. A questo proposito, il lavoro di ricerca presentato in questo testo mira allo **sviluppo di nuovi metodi e strumenti di progettazione** che siano in grado di sostenere sia il miglioramento degli attuali velivoli da trasporto che lo sviluppo di configurazioni innovative e non convenzionali. L'effettivo successo di ogni nuovo strumento di progettazione dipendera' dalla sua effettiva capacita' di funzionare all'interno dei tipici processi produttivi, che generalmente si sviluppano attraverso una supply chain vasta e geograficamente distribuita. Inoltre, questi nuovi strumenti e metodi di progettazione dovranno tenere conto della la crescente scarsità di risorse intellettuali e, quindi della la necessità di aumentare la produttività dei progettisti e liberare il tempo necessario per l'innovazione.

L'approccio di progettazione e ottimizzazione multidisciplinare (**MDO, Multidisciplinary Design Optimization**) sembra essere la metodologia più promettente nel campo della progettazione degli aeromobili, sia al fine di migliorare le prestazioni delle configurazioni tradizionali, che per sostenere lo sviluppo di nuovi concetti. Tuttavia, una serie di ostacoli tecnologici ha finora impedito il pieno sfruttamento dell'approccio MDO e ne ha limitato l'applicazione in ambito industriale a soli casi di disegno di dettaglio.

A questo scopo, presso la facoltà di ingegneria aerospaziale dell'Universita' di Delft, e' stato sviluppato il concetto di **Design and Engineering Engine** (**DEE,** motore di progettazione e ingegnerizzazione). Il DEE è un sistema computerizzato modulare, adattabile e scomponibile, in grado di facilitare processi distribuiti di progettazione e ottimizzazione multidisciplinare, attraverso l'automatizzazione di tutte quelle attività ripetitive e non creative che generalmente ne ostacolano e rallentano lo svolgimento.

Uno degli moduli chiave del DEE è rappresentato dal Multi Model Generator (generatore di modelli) che di fatto costituisce il principale risultato di questo lavoro di ricerca. Il **Multi Model Generator (MMG) è un'applicazione Knowledge Based Engineering (KBE)** sviluppata con il duplice intento di 1) fornire ai progettisti un ambiente avanzato di modellazione parametrica per la definizione di modelli generativi di velivoli convenzionali e non; 2) fornire ai vari moduli di analisi presenti nel DEE, gli specifici modelli (le varie astrazioni del velivolo) necessari per la fase di verifica del velivolo. A tal scopo, sono stati sviluppati due tipi di blocchi funzionali, che in effetti costituiscono gli ingredienti principali del MMG: le cosiddette High Level Primitives (primitive di alto livello) e i Capability Modules (moduli di capacità).

Le **High Level Primitives** (HLPs) definite finora sono quattro: un elemento d'ala (Wing-part), un elemento di fusoliera (Fuselage-part), un motore (Engine) ed un elemento di connessione (Connection-element). Queste possono essere immaginate come un set di mattoncini LEGO speciali, che il progettista può manipolare e ricombinare al fine di assemblare la geometria del velivolo che ha in mente (sia in termini di superfici esterne che struttura interna). Utilizzando il linguaggio di programmazione a oggetti disponibile all'interno della piattaforma KBE selezionata per questo lavoro, ogni HLP è stata definita come classe. Questo ha permesso di formalizzare all'interno di ogni primitiva le procedure (design rules) che permettono loro di adattare automaticamente la propria forma e struttura, e reagire ad ogni cambiamento degli input. L'utilizzatore del MMG puo' modificare i valori dei vari attributi di ogni (HLP) classe e decidere il numero di instanze necessarie, semplicemente editando l'input file del MMG. In questo modo, e' possibile definire sia configurazioni di velivoli convenzionali che non, per poi modificarle e plasmarle in un numero praticamente infinito di varianti parametriche.

Durante la fase di progettazione concettuale, il progettista "vede" l'aeromobile come una combinazione di soluzioni per soddisfare funzionalità del tipo generare portanza e ospitare il carico utile. In effetti, il velivolo non e' visto come un semplice insieme di punti, curve, superfici e solidi. E' propio per questa loro capacita' di supportare l'approccio di progettazione funzionale che che le primitive sviluppate in questo lavoro sono state definite di "alto livello"; in contrasto con le primitive di "basso livello" (punti, curve, etc..) dei sistemi CAD convenzionali.

Una volta disponibile la configurazione del velivolo, inizia la fase di preparazione (pre-processing) per l'analisi. Questa richiede la generazione dei vari modelli, o "viste", disciplinari, cosi' come richieste dagli strumenti di analisisi presenti nel DEE. Specialmente nel caso di utilizzo di strumenti di analisisi molto accurati (High Fidelity), la preparazione di questi modelli disciplinari puo' essere molto lunga lunga e ripetitiva e consumare fino al all'80% del tempo complessivo di progettazione. Tuttavia, è stato osservato che 1) indipendentemente dalla configurazione del velivolo, gli specialisti adoperano gli stessi metodi e strumenti per la preparazione dei modelli da analizzare; 2) gran parte delle attività di preparazione dei modelli è basato su procedure consolidate che richiedono intensive manipolazioni geometriche; di fatto, i punti di forza della tecnologia KBE.
Al fine di aiutare il progettista in questa specifica fase di progetto, e' stata sviluppata una serie di moduli, detti **Capability Modules** (CM, Moduli di Capacita') in grado di formalizzare e automatizzare la preparazione di modelli per una vasta gamma di strumenti di analisi, sia semplici che sofisticati, sia commerciali che sviluppati privatamente.

In particolare, questo approccio ha permesso l'utilizzo di strumenti di analisi molto accurati, come codici ad elementi finiti e fluidodinamici, già a partire dalle prime fasi del processo di progettazione. Con il risultato che, non solo il livello di confidenza circa i risultati ottenuti ne giova, ma diventa di fatto possibile l'utilizzo di strumenti analitici essenziali per lo studio di velivoli innovativi, per i quali i tradizionali metodi semiempirici basati su dati storici risultano del tutto inadatti.

Inoltre, le possibilita' di accedere e utilizzare il MMG anche in remoto e senza il bisogno di alcuna interfaccia grafica, usando connessioni web standard, fanno del MMG una risorsa preziosa al fine di **abilitare processi MDO distribuiti** (dove i vari moduli computazionali sono istallati su macchine che operano e comunicano da diverse locazioni geografiche).

La capacità del MMG è stata dimostrata attraverso varie applicazioni, tra cui due casi di rilievo descritti in questo lavoro. Il primo riguarda il progetto europeo MOB, sulla progettazione e ottimizzazione multidiscilinare di velivoli tipo blended wing body. Il secondo riguarda lo sviluppo di un sistema MDO, definito in collaborazione con Airbus, al fine di ridisegnare la coda verticale di un aereo passeggeri.

L'obiettivo secondario di questo lavoro consiste nel migliorare la diffusione della tecnologia KBE, che di fatto è una disciplina relativamente giovane, che non ha ancora trovato il meritato livello di attenzione e comprensione, sia nel mondo dell'industria che nell'ambito accademico. A questo scopo, viene qui fornita una trattazione ampia ed originale sui **legami della tecnologia KBE con il mondo dell'intelligenza artificiale** ed il paradigma della modellazione a oggetti. A

benificio degli interessati, questa trattazione include anche una sezione di **best practice** per lo sviluppo di applicazioni KBE.

List of Symbols and Acronyms

ACARE	Advisory Council for Aeronautics Research in Europe
AI	Artificial Intelligence
BWB	Blended Wing Body
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CFD	Computational Fluid Dynamic
CFRP	Carbon Fiber Reinforced Plastic
CL	Common LISP
CM	Capability Module
COTS	Commercial Of The Shelf
DEE	Design and Engineering Engine
ES	Expert System
FBS	Frame Based System
FE, FEM	Finite Elements Method
FEA	Finite Element Analysis
HLP	High Level Primitive
GLARE	GLAss REinforced
GUI	Graphical User Interface
IGES	Initial Graphical Exchange Standard
IT	Information Technology
KA	Knowledge Acquisition
KBE	Knowledge Based Engineering
KB	Knowledge Base
KBS	Knowledge Based System
KE	Knowledge Engineering

KM	Knowledge Management
MDO	Multidisciplinary Design Optimization
MMG	Multi Model Generator
MML	MOKA Modeling Language
MOB	The European research project "A Computational Design Engine Incorporating Multi-Disciplinary Design and Optimisation for Blended Wing Body Configuration"
MOKA	Methodology and Tools Oriented to Knowledge Based Engineering Applications
OO	Object Oriented
OOP	Object Oriented Programming
PCL	PATRAN Command Language
PDM	Product Data Management
PLM	Product Lifecycle Management
RBS	Rule Based System
FBS	Frame Based System
SRA	Strategic Research Agenda
UML	Unified Modeling Language
XML	EXtended Markup Language
XSLT	Extensible Stylesheet Language Transformations

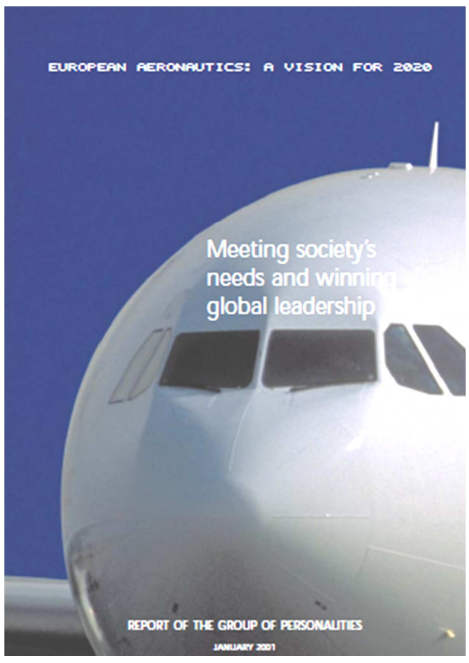
Table of Contents

CHAPTER 1
The Paradigm Shift for the New Era of Aviation

1. A vision for the future of aviation
2. Fast replay: one century of technology in the Air Transport System
3. …the end of the second S-curve
4. The twilight of the Kansas City Aircraft?
5. In preparation for the breakthroughs
6. High level goals and structure of this research work

1.1 A vision for the future of aviation

In 2020, the stressed-out passenger belongs to aviation past. There are no more queues and interminable waiting for a delayed departure or arrival. From start to finish, the entire flying experience is designed to ensure a contented traveler. At all prices, an airline ticket buys choice, convenience and comfort.
There are more routes and more flights to and from most destinations. 99% of all flights arrive and depart within 15 minutes of the published timetable in all weather conditions. Airports are no longer a test of the traveler's stamina and patience. It takes no more than 15 minutes in the airport before departure and after arrival for short haul flights, 30 minutes for long haul. The entire airline system is operating with great efficiency. Aircraft cost less to own, operate and maintain. All these savings are passed on to paying passengers.



In 2020, the skies are safer than ever because safety has remained the top priority of the aircraft builders and operators and of air traffic managers.
Aeronautics has made huge steps towards eliminating accidents altogether by designs and

automatic systems that lighten the burdens on the crew and help them to make correct decisions in any situation.

In 2020, aircraft are cleaner and quieter. Though hydrocarbon-based fuel is still the main source of energy, improved engines allow a reduction of CO2 and NOx emissions by 50% and 80%. With a reduction in perceived noise to one half of current average levels, transport aviation has ceased to be a nuisance to people living close to airports thanks to a concerted effort to develop quieter engines, optimize operational procedures and improve land planning and use around airports. The aeronautics sector's contribution to a sustainable environment is widely understood and appreciated.

In 2020, Europe has managed to create a seamless system of air traffic management that copes with up to three times more aircraft movements than today by using airspace and airports intensively and safely. The development of sophisticated ground and satellite-based communication, navigation and surveillance systems as well as free flight has made this possible.[1]

Fig. 1.1: Examples of advanced non conventional aircraft configurations for the future: the blended wing body (left) and the joint-wing aircraft (right), also knows as Prandtl-Plane (Frediani, 2004).

In 2050, the sky is populated by blended wing bodies, joint-wings (Fig. 1.1), ultra-fast rotorcraft, tilt-rotors, multi-fuselage aircraft developed by European, American, Russian, Brazilian, Chinese, Japanese and Indian manufacturers. They are far quieter, more fuel efficient, safer, cleaner, faster and more comfortable than the 2020 aircraft they replace (Fig. 1.2 right).

[1] "In 2020.." text from *European Aeronautics: a Vision for 2020*, prepared under initiative of the European Commissioner for Research P. Busquin (Group of Personalities, 2001).

Fig. 1.2: artist rendering of the NASA Morphing Airplane (left) [www.dfrc.nasa.gov]. The super efficient and silent Airbus concept plane presented at Farnborough 2010 (right) [www.airbus.com]

In 2050, aircraft use alternative bio-fuels and new types of engines. They are built with multifunctional, self healing bio composite material implementing nano-technologies. They fly without control surfaces, morphing their surfaces as a bird (Fig. 1.2 left) and making use of extensive active flow control. They take off and land on very short runways thanks to their full vectoring thrust capability and extremely light weight structure.

In 2050, self landing and taking off aircraft, one man controlled cockpit are normality. Unmanned, remote controlled freighters transport wares worldwide. Aircraft of any size fly in a worldwide interconnected, uncongested, green and safe airspace system.

In 2050, customers are able to fly from next door facilities, directly to their final destination; they can call an air-taxi or directly book an aircraft at the closest *drive-or-fly* rental and fly it[2]. What a bold vision! If it is true that in the 2020 vision, the whole air transportation system will differ from today's as much as the actual system differs from that of 1930s, the scenario

Fig. 1.3 *The Jetsons*, a popular Hanna-Barbera cartoon series from the 1960s

[2] "In 2050.." text elaborated by the author on the basis of the following reports: *NASA Aeronautic Blueprint: Towards a Bold New Era in Aviation* (NASA, 2002), the ACARE documents *Strategic Research Agenda* (release 1, 2 and addendum) (ACARE, 2002; 2004; 2008) and *Aeronautics and Air Transport: beyond 2020 (Towards 2050)* (ACARE, 2010).

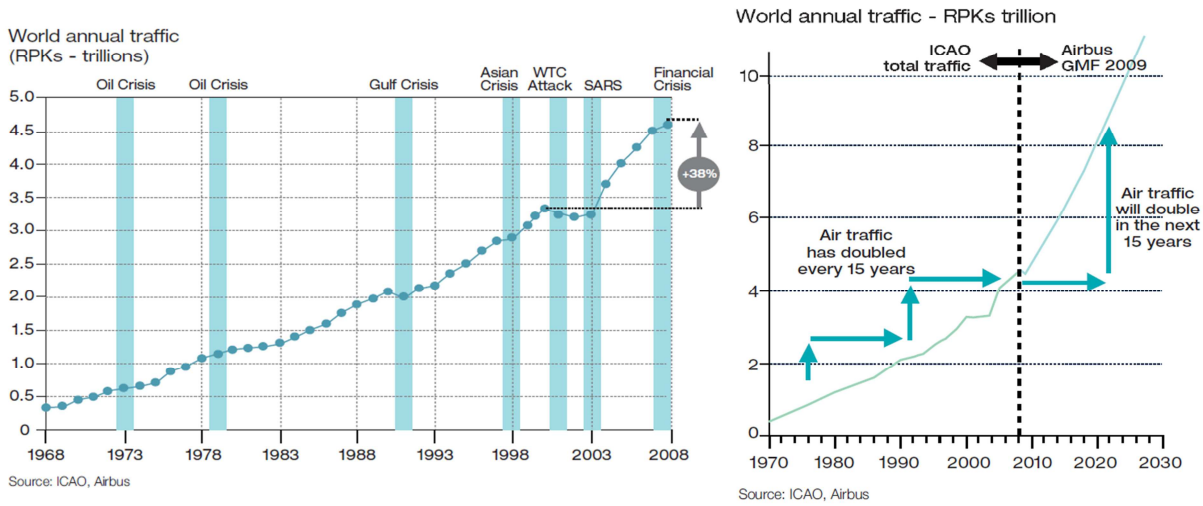


Fig. 1.4: (resilience of the) global air traffic growth and forecast 2009-2028 (Airbus, 2009). (RPKs: Revenue Passenger Kilometers)

envisioned for 2050 by NASA and ACARE (Advisory Council for Aeronautics Research in Europe) seems closer to a *Jetsons* screenplay (Fig. 1.3) than a possible reality. Indeed the aeronautic community will have to make serious advances to cope with a demand that, since the 1960s, has doubled every 15 years, notwithstanding various oil crises and the dark years following September 11 (Fig. 1.4). Furthermore, now as never before the environmental sensitivity was so high. Adding to this the increasing cost of fuel and the current state of the ATM system approaching its limits, it is obvious that changes are just necessary to sustain any further air traffic growth.

1.2 Fast replay: one century of technology in the Air Transport System

Following the publication of the *Vision 2020* document (Group of Personalities, 2001), ACARE was formed and assigned the task to prepare a roadmap to bring the European aeronautical industry to the 2020 targets. This effort has resulted in the Strategic Research Agenda (updated several times since 2002), which opens with the performance analysis of the first century air transportation[3]. The resulting plot (Fig. 1.6) shows two evident S-curves, one covering the so called *Pioneering Age* of aviation, from the Wright brothers' flyer to the 1950s, the other the so called

[3] The source does not specify the metric used for transportation performance. A conventional measure is based on the product of speed, range and payload divided by a measure for the operating costs. However, a correct measure of performance should account for the percentage of customer requirements satisfied. For instance, is the availability of showers on board increasing the transportation performance of the A380?

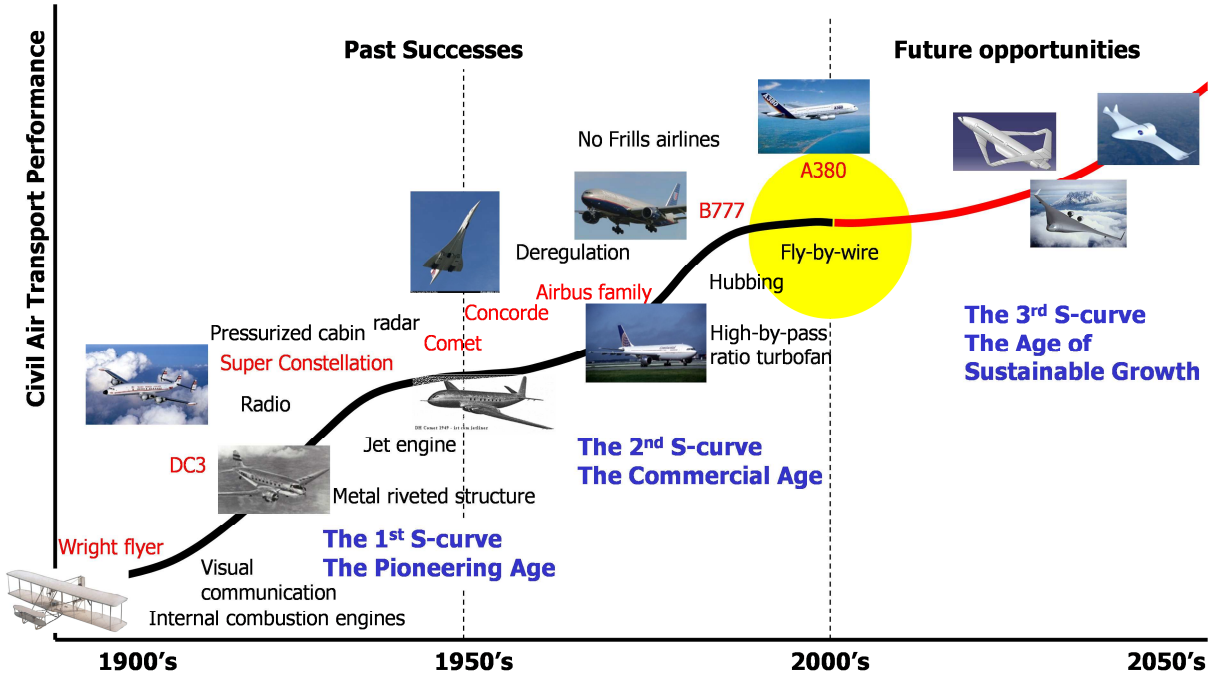


Fig. 1.6: performance of civil air transport since the beginning of last century: the S-curves. Based on (ACARE, 2002).

Commercial Age of aviation, from the debut of the jet engine to date. These curves reveal a progress of the performance based on breakthrough innovations and periods of evolutionary improvements.

Indeed, the Wright's flyer, at the beginning of the first S-curve, had barely any impact on the world of transportation, but further advances in structures and materials (metal riveted structures in place of wood, textiles and wires), aerodynamics (single wing with elliptical planform and retractable gears), propulsion (turbo-charged engines with variable pitch propeller) and navigation (inertial navigation) brought air transport to the extended practical use of the 1930s, when the human perception of distance changed forever. At that point the end of the first S-curve was reached. By the way, the Wright's flyer itself was the result of an evolutionary process that brought the available knowledge of aerodynamics and internal combustion engines to an

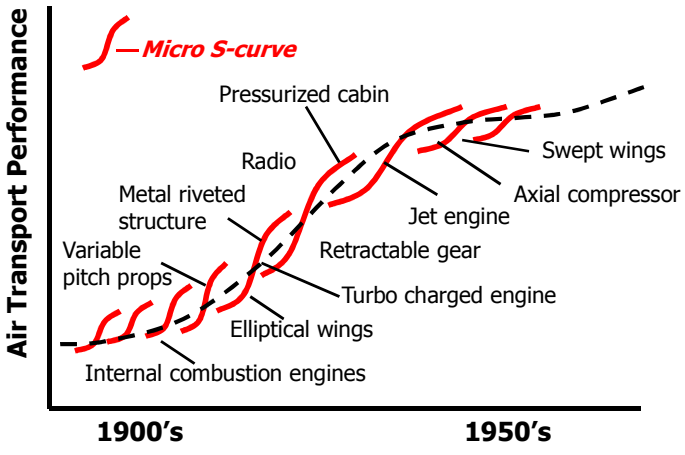


Fig. 1.5: thread of micro S-curves contributing to a macro S-curve.

adequate level of maturity. As a matter of fact, each S-curve appears to be the result of a thread of *micro S-curves*, as shown in the close up of Fig. 1.5. As argued in *Shock of the Old* (Edgerton, 2006), it is the progress and mutations of old stuff that makes the gradual big difference.

Around the end of WW II, the jet engine entered the scene as the new breakthrough. However, it neither affected the outcome of the war, nor had any immediate benefit on commercial aviation: that was just the beginning of the second S-curve. It was thanks to further progress in jet technology (axial compressor) and advance in aerodynamics (swept wings) that in the 1960s commercial aviation entered its rapid conversion to the jet age. Further developments in technology (fly-by-wire, high by-pass ratio turbofan, supercritical airfoils) and new approaches in air traffic control and management (deregulation, hubbing, no-frills airlines) have brought the jet age to level of maturity and consolidation of these days and, apparently, to …

1.3 …the end of the second S-curve

McMaster and Cummings (McMasters and Cummings, 2004) narrate that, at the beginning at the 1980s, the Boeing 757 development team, as part of the explanation to their management on the reason why their new aircraft, in spite of the very large amount of money invested in research and development, the far larger development team and 25 more years of technology and knowledge to leverage, carried no more passengers, any farther and any faster than its predecessor from the 1950s, the Boeing 707[4], came out with a plot, indeed very similar to the one in Fig. 1.6. In a *performance vs. time* plot similar to the one shown in Fig. 1.7, they showed three curves. The first was a horizontal asymptotic line, representing the theoretical upper bound established by the basic law of physics and economics. The second was an oscillating curve, representing what could be accomplished having perfect knowledge of the current technologies and no economic limits. The third line was indicative of the actual achievement: the progress made in years of efforts, striving to reach the asymptotic limits of the current technologies.

And there it was the point of the 757 team: when the gaps between the three curves shrink, the opportunities of further gain in the traditional measures of performance, such as range, speed, payload capacity, become increasingly difficult and expensive to reach.

Even if engineers learn to make a better use of current technology, they must run harder and harder to get smaller and smaller gains, because, on the other side,

[4] Actually, the 757 was developed as successor of the Boeing 727, which was the direct heir of the 707. But this would not change the essence of the story.

customers get more sophisticated and regulation authorities put harder and harder constraints.
Without planning a new breakthrough it will not be possible to sustain the future growth of air transport and, eventually, start a new curve in the plot: the 3rd curve of Sustainable Growth!

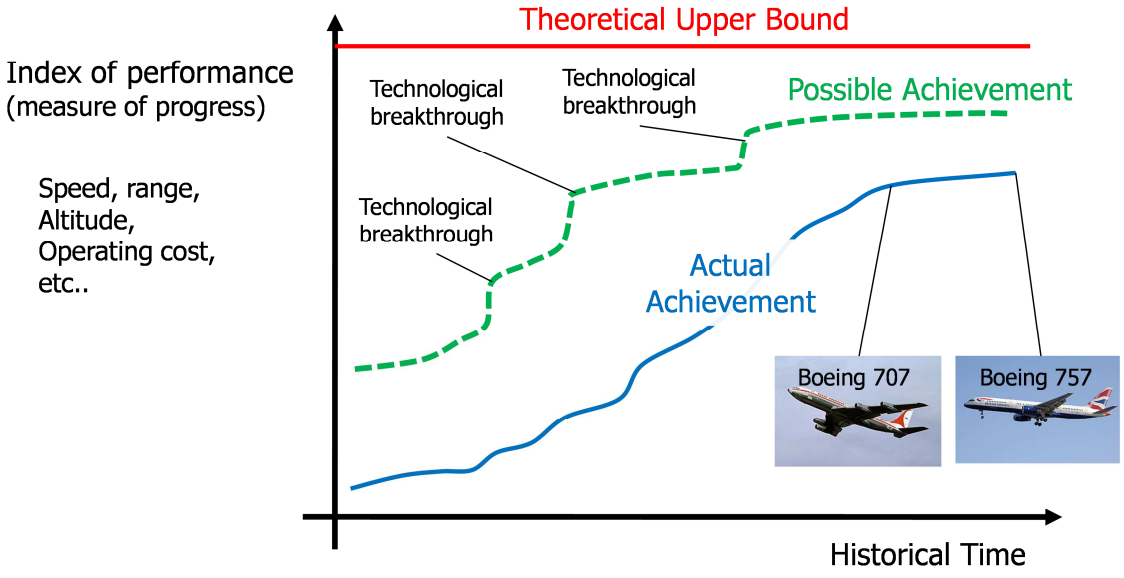


Fig. 1.7: why the Boeing 757 "did not get any better" than the old 707. Based on (McMasters and Cummings, 2004)

1.4 The twilight of the Kansas City Aircraft?

In the musical *Oklahoma!*, cowboy Will Parker, back from an excursion to Kansas City, sings like this: "*Ev'rythin's up to date in Kansas City. They've gone about as fur as they c'n go!".*
Apparently it is here that the nickname originated for the dominant airliner configuration, i.e., the quasi cylindrical fuselage with cantilever (swept) wing and tail empennages, and engines podded either under the wing or at the back of the fuselage. Indeed the Kansas City airplane has gone almost as far as it can go, particularly in terms of improved economic and environmental performance (Green, 2003). That is clearly acknowledged also by ACARE, which states: "*The environmental challenge has clearly identified the limits of current technology, which, while it has more to offer and more that will be achieved over the next decade or so, must be succeeded by completely fresh approaches that require an early start. The* (Vision 2020) *objectives are not achievable without important breakthroughs, both in technology and in concepts of operation* (ACARE, 2002)".

Fig. 1.8: From left: Airbus A380 (first flight: April 2005); Airbus A350 rendering (first flight scheduled in 2012); Boeing 787 (first flight: December 2009).

[www.airbus.com, www.boeing.com]

This is echoed by the Greener by Design Science and Technology Sub-Group, an initiative of the Royal Aeronautical Society, that points at the environmental impact "*as the most serious long-term threat to the continued growth of air travel* (Greener by Design, 2005)"

After more than 60 years of optimization, the Kansas City Aircraft has become 70% more fuel efficient and 20dB quieter than the first generation of jets airliner (Green, 2003), still maintaining the "classical" configuration inherited by the Boeing B-47 Statojet of 1947. Such a configuration, based on the early 1800 Caley's concept of functional separation (i.e., fuselage for payload, wings for lift, tail empennage for control and stability, etc.), has definitely proven effective and dominated the entire passenger aircraft development story to date (van Tooren, 2003). The whole aviation infrastructure (airports, terminals, luggage handling systems, etc.) has evolved around that configuration; manufacturers have refined their design methods and developed their facilities to deal with that configuration. Also for passengers, flying is just about sitting in a cylinder with some small windows at the side...

However, as in any optimization process, once close to the optimum any improvement on the objective function get just smaller. Even more, as in any optimization process, if the objective function and the constraints change, the design space is going to change and the optimum is going to be somewhere else. The effect of the new design objectives and constraints set for a sustainable growth of aviation are likely to bring the search towards areas of the design space where the Kansas City aircraft is not likely to be a winner any more.

The A380, the new 787 and the A350 (Fig. 1.8) appear to be the last outstanding offspring of the Kansas City airplane. As suggested by ACARE, they will keep offering improvements in the new decade or so, but it is now the time to start preparing for a fresh new start.

1.5 In preparation for the breakthroughs

Breakthroughs and inventions need to be scheduled. It means efforts must be invested on the development of the "technology components" (i.e., the micro S-curves discussed in section 1.2), which can, together, lead to the realization of the actual breakthrough.

What actually does not come too apparent from Fig. 1.6 and Fig. 1.5 is that a technology component to the realization of a breakthrough does not have to be necessary a new material, or a new electronic system, or a new engine, but can be also a *new design approach* or a new method *to exploit the available intellectual resources*.

Indeed, it will be very difficult to tackle the challenges indicated by ACARE without first addressing the challenges associated to the very development of any new technological advance.

To this purpose, the following sub-sections will address the organizational issues faced by the big aircraft manufacturers, including the problems related to the availability, productivity and management of the new intellectual resources. Finally, the capability of the current design methodology will be discussed as an introduction to the main focus of this research work.

1.5.1 The challenges of the global organization

The aeronautical industry is characterized by a very large body of knowledge and skills that can only be improved and applied against very large and long term investments. A lot of risks are taken while the margins of revenue are relatively small and sensitive to the world socio economical events (financial crises, fluctuation in the oil price and Euro/dollar exchange rate). Also in the case of the defense industry, after the "wealthy excitement" of the Cold War, the recurrent mantra is *performance at affordable cost* (McMasters and Cummings, 2004).

One of the most visible consequences of the adaptation process the aeronautical industry has undergone in the last decades to survive the troubled seas of the global economy, is possibly its *consolidation*. In just 7 years (from 1990 to 1997), the scenario of the major American aerospace companies has passed through consortia and acquisitions from 15 large companies to just 4 major groups, namely, Lockheed-Martin, Boeing, Raytheon and Northrop-Grumman (Raj, 1998). A similar process has happened in Europe, where the largest aeronautic industrial groups, both in the sector of defense and commercial aviation, have joined in the trans-national corporation EADS.

Consolidation operations actually go far beyond 'one name, one goal, one budget'. A less evident process, but of extreme significance, is the enterprise conversion to a *distributed organization* that employs and coordinates professionals spread worldwide. This disrupts the historical perceived image of the company, once identifiable with its tangible asset, such as its plants and facilities, and now as one virtual brain and many arms that operates 24 hours a day across all the planet time zones.

The other approach more and more commonly adopted by big manufacturers to make new development programs affordable and lower financial risk is to form partnerships with groups of selected risk-sharing suppliers that are willing to invest their own economical and intellectual resources in the development of major systems and components.

This is the case of the Joint Strike Fighter (JSF) project coordinated by Lockheed Martin, and, very recently, of the Boeing 787 development program. Here, for the very first time Boeing decided to outsource (to Japan) also the development and manufacturing of a major system like the wing. Fig. 1.9 gives the feeling of the large number of international contributors to the project.

Also Airbus, started as a trans-national joint venture between English, German,

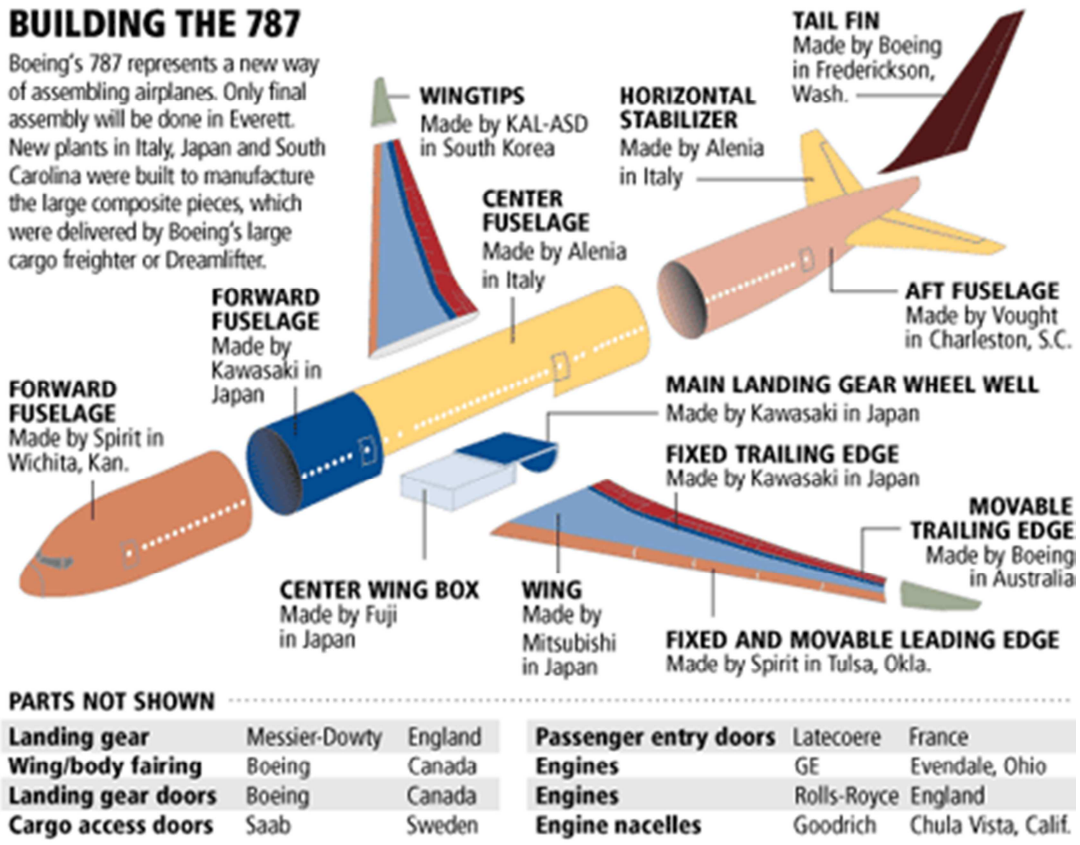


Fig. 1.9: worldwide contributions to the manufacturing of the Boeing 787 [www.boeing.com]

French and Spanish aerospace groups, has since long extended its collaboration outside the Airbus walls to many countries, including Russia, China, Australia, Austria, Belgium, Canada, Finland, Italy, Japan, South Korea, Malaysia, Netherlands, Sweden, Switzerland and the United States.

The flip side of the coin is represented by the new challenges of effectively managing and synchronizing knowledge and resources, which are often rooted in very different socio-cultural backgrounds.

At technical level, the exchange of data and models across scattered repositories and systems, the interfaces between codes, the protection of knowledge both from the side of the integrator and the suppliers, just to mention some, represent continuous headaches.

It is also worth of consideration the effort to organize the logistics to gather parts and components produced around the world into one final assembly location. Often is not only matter of organization but also of development of new infrastructures to move assemblies and parts around, via sky, water and ground (Fig. 1.10).

However, the logistic of wares, whatever the distance, the time schedule and the size, is eventually a less complicated issue than the "logistic of brains". Communicating ideas, harmonizing practices, skills, know-how and aligning different people toward one common objective remain possibly the hardest tasks in any collaboration initiative.

Fig. 1.10: logistics efforts required by the transnational scale of the A380 manufacturing: components transportation via sky, water and ground [www.airbus.com].

1.5.2 The challenges of intellectual resources. Knowledge Management issues

The situation concerning the intellectual resources is rather complex and can be summarized with the following issues:

- They are less
- They are different
- They lack the knowledge strength of the "old type of experts"

Brains drain

From one side the western society is facing an increasing scarcity of new aeronautical engineers (which, by the way, do not always remain in their field), while, on the other, there is an accelerated pace of baby boomer retirements, which is putting out of the companies the minds and the hands that built the story of aeronautics since the 1960s.

In 2002, NASA reported that the average age of those employed in the American aerospace industry was slightly above 47 years, with the engineer and scientist community alone approaching the average of 57 years (NASA, 2002). NASA itself currently loses senior R&D expertise at twice the rate of incoming new researchers. More than 20% of Boeing workforce in this moment is eligible for retirement. Plenty of these data can be found in company magazines and Knowledge Management publications, just searching for "brains drain" (Dunlop, 2010; Patton, 2006; Sopranos, 2005; DeLong, 2008).

> **WANTED:** *Retired Boeing scientists and engineers who want to enjoy retirement to its fullest, while having a flexible, paid part-time career in their disciplines*
>
> [www.**yourencore.com**]

A new breed of workers: the *Knowledge Workers*

The culture of work itself has changed in the last years and new professional figures populate the company positions. Some thirty years ago, Peter Drucker coined the term "knowledge worker" to address the evolved sort of high-profile professionals described in the insert below (Drucker, 1999). New generation experts do not belong anymore to the manual/clerical worker model inherited from the military culture of 100 years ago. They are much more entrepreneurs oriented, they have a strong self-conscience and potentially superior capability to enrich the company knowledge, but at the same time they are more complicated resources to manage. They are

From "Management Challenges of the 21st Century" by P. Drucker

The knowledge worker

"[…] fewer and fewer people are subordinates - even in fairly low-level jobs. Increasingly they are *knowledge workers*. Knowledge workers cannot be managed as subordinates; they are associates […] This difference is more than cosmetic. Once beyond the apprentice stage, knowledge workers must know more about their job than their boss does - or what good are they? The very definition of a knowledge worker is one who knows more about his or her job than anyone else in the organization [...]"

"What motivates workers, especially knowledge workers, is what motivates volunteers. Volunteers, we know, have to get more satisfaction from their work than paid employees precisely because they do not get a pay check. They need, above all, challenge. They need to know the organization's mission and to believe in it. They need continuous training. They need to see results."

continuously in search of professional challenges for which they often change position inside the company, or move elsewhere.

Less and longer projects

Since the 1950s, there has been a continuous reduction in the amount of new aircraft development programs, caused by their growing complexity and required development time.

The evolution to the actual situation, relatively to military programs, is represented in Fig. 1.11, although a similar trend is experienced by civil aviation. As pointed out in (van Tooren, 2003), the consequence is that people who entered the aeronautic business not long ago have matured their professional experience on a relatively low number of aircraft development projects. Newcomers will possibly never get the opportunity to be involved in any single complete project, during their entire career. The recently booming interest in UAV and UCAV development might change this trend, at least for this particular category of aircraft.

The direct consequences

The first obvious consequence of the brains drain is that the aerospace industry, at least in the short/midterm period, will be forced to bear the development of more complex systems, with a lower amount of intellectual resources (*more with less*).

When considering also the other two points mentioned above, then some other knowledge management-related issues arise.

The first is that, while the *risk of losing knowledge* when an experienced employee

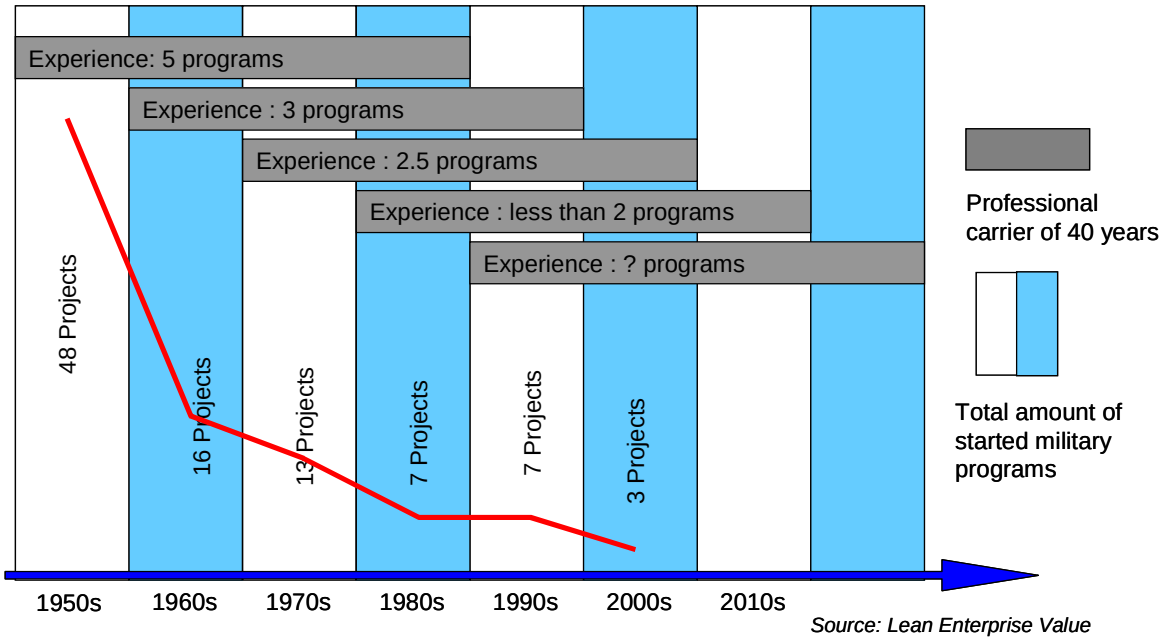


Fig. 1.11: the decreasing amount of professionals' specific experience and the diminishing number of aerospace military program during time (van Tooren, 2003).

leaves the company is rather evident, the risk associated to professionals' mobility (within the same company) are often underestimated, although similar. Indeed, when a company moves one of its professionals to a new business area, often with the intent to secure such intellectual resource, as well as to exploit his/her knowledge in other areas, there is the threat of making inaccessible or unused part of the knowledge that was developed and maintained by the employee during his/her former assignment. In both situations *gaps* are created between the knowledge the organization needs to do business and the knowledge the organization has ready available to deploy (McBriar et al., 2003).

The second concerns with the "type of" knowledge on which the organization will have to account. In this respect, McBrair propose a distinction between knowledge *depth* and *strength* (McBriar et al., 2003). A deep type of knowledge is typical for a well educated professional and allows mastering a specific discipline and performing certain tasks. A strong type of knowledge is what distinguishes a "veteran" from a novice and provides a broader problem solving competence, as well as the capability to understand complex cause-effect systems. It gives the possibility to see beyond the borders of the simple task and solve problems of higher complexity (also in terms of strategy).

On the same line, Quinn proposes a classification of four professional intellect levels (Quinn, 1998) and claims that for an organization to be successful all must be available (see insert below). In this case, the *system understanding* level corresponds to McBriar's knowledge strength concept.

The four levels of a professional intellect. J.B. Quinn

Cognitive knowledge (know-what): the basic mastery of a discipline that professionals achieve through extensive training and certification.

Advanced skills (know-how): translate "book learning" into effective execution. Ability to apply the rules of the discipline to complex real-world problems.

System understanding (know-why): deep knowledge of the web of cause-and-effect relationships underlying a discipline. It permits professionals to move beyond the execution of simple tasks, to solve larger and more complex problems and create extraordinary value.

Self-motivated creativity (care-why): will, motivation and adaptability for success. Highly motivated and creative groups often outperform groups with greater physical or financial resources.

The cost for the company and required actions

The lack of knowledge sharing and retention plans comes always at a cost for the company, generally far higher than the cost of any knowledge management

initiative. In the words of DeLong, the economic consequences can be summarized as follows (DeLong, 2004):

1. Reduced capacity to innovate: fewer "prepared minds" to connect the dots and create something new.
2. Ability to pursue growth strategies threatened: expanding the current business requires people that understand both the business and the technologies.
3. More costly errors: fewer people around who have already made the mistake the new guys are about to make.
4. Less efficiency: loss of the knowledge of how to get things done within the organization.

Indeed, the competitiveness of a company on the market is strongly affected by its capability to deliver the right product at the first time, and the lessons learnt in previous projects can the best guide to avoid reiterating old mistakes. This might sound obvious, but the accessibility to such knowledge in a large, transnational company is far from reality (not only in aerospace companies!). It is a matter of fact that many companies are continuously "re-inventing the wheel". For example, studies carried in the late 1990s at Rolls Royce revealed that 40% of problems tackled within company projects appeared to have been already solved in past assignments (Clarkson, 2000). Today Rolls Royce is a leading company in terms of knowledge management initiatives.

Although it is not the author's intention to further deepen into the pure knowledge management issues of the aeronautic industry, it is clear that knowledge has to be recognized as a business key asset. As such it needs to be managed and engineered, aiming at the maximum return of investment.

1.5.3 The challenge of the design approach

In the early days, creative spirit and practical knowledge of few basic disciplines were enough to allow a very small number of talented individuals to be in control of the whole aircraft development process, from scratch to flight testing. After the 1930s the situation started to change towards the complex. Wind tunnel testing, analysis of thin-walled structures, and the need for controllable and scalable manufacturing processes called for an enlarged team of experts. Such a process of specialization and discipline segregation actually never stopped.

For many years, the subdivision of a complex product design process into disciplinary areas has appeared as the only suitable way to make it controllable, as well as workable within the capability boundaries of available people, design tools and computational systems. Although such an approach has produced satisfactory results for a long period, it does not seem adequate to keep up with the continuously increasing complexity of new development programs, organizations and

technologies. The day the experienced designers/systems engineers will leave without breading a next generation, the system might just turn into loose sand. While segregation started as a solution to a problem, eventually, it turned back into a challenge!

The success of new aircraft development program depends upon quality and timing of decisions throughout the entire design process, which again strongly depends upon timely availability of knowledge. Since, by definition, design is a decision making process within uncertainties, it is of extreme importance that designers can get *reliable and fast* answers to all their *what-if*s.

Unfortunately the reliability of analysis results is generally inversely proportional to the time required for their generation. Time is a scarce and precious resource, which in the current design approach is often wasted in repetitive activities and organizational inefficiency, at expenses of creativity and innovation. Nonaka, one of the Japanese fathers of knowledge management used to say that "companies need *plenty of slack* to remain creative" (Nonaka and Takeuchi, 1998). Indeed, lack of time leads to reapplying the same solution without giving new designs the chance to be fairly traded-off against conventional ones.

Now, more than in the past, there is the opportunity to capitalize on the convergence of a broad front of multidisciplinary advances in technology. In order to make a step change in aviation, a paradigm shift in the *design methodology* will be required. The availability of new *integrated and lean* design methods and tools that are able to harness the available knowledge, investigate with agility the cause-effect network of all the involved disciplines and exploit their optimal integration, will be the key to enable and speed up the transition of new concepts and technologies into operation.

1.6 High level goals and structure of this research work

The challenges discussed above constitute the motivations at the basis of this research work. Indeed, the high level goal of this work consists of the development of new design methods and tools that are able to sustain the evolutionary improvement of current aircraft designs and lower the risk associated to the development of novel aircraft configurations.

Such design methods and tools should facilitate the aircraft development process as currently carried across a large and distributed supply chain. Besides, they should largely increase the productivity of knowledge workers through a better exploitation of their knowledge and the company know-how, thereby reducing the time wasted in the repetitive and non-creative activities of the design process, and freeing the time necessary for innovation.

The description of the work carried in this research work has been structured as follows:

Chapter 2

The need and the opportunities of a transition from the traditional design approach to the one based on multidisciplinary design optimization (MDO) are addressed in this chapter. To this purpose, a review on the current state of MDO is provided, together with a description of the current implementation challenges.
Then, the concept of the Design and Engineering Engine (DEE) is introduced, which is the advanced design system to support MDO currently under development within the chair of Systems Engineering and Aircraft Design of the TU Delft faculty of aerospace engineering. In particular, the role of the Multi Model Generator (MMG) is highlighted, being this a core component of the DEE as well as the main outcome of this doctoral research.
The MMG is a *Knowledge Based Engineering* (KBE) application with the twofold role of providing the designer with a smart parametrical tool to model the geometry of different aircraft configurations, and automatically extract from these models specific abstractions (views) to support multidisciplinary analysis.

Chapter 3

Chapter 3 is fully dedicated to KBE technology. Its main characteristics as well as its roots in the field of Artificial Intelligence are discussed in this chapter, with the intent of shedding light on the similarities and the fundamental differences between a true KBE system, a CAD system and a Knowledge Based Systems. Furthermore, the chapter elaborates on the objected-oriented programming language and the integrated CAD modeling capabilities featured by a KBE system, being the combination of these two the key to effectively capture engineering rules and automate geometry manipulation.

Chapter 4

In this chapter the development of the MMG concept introduced in chapter 2, is fully elaborated. The architecture of the system is described together with the concepts of High Level Primitives (HLP) and Capability Module (CM), which actually constitute the two main functional components of the MMG. Few High Level Primitives can provide the necessary building blocks for generating parametric models of different aircraft configurations and variants, either with conventional or novel architectures. Several CMs provide the mechanisms to automatically preprocess the generated geometry and initiate the MDO process.

Chapter 5

In Chapter 5, the software implementation of the HLP by means of the selected KBE system is described in detail. The goal of this chapter is to explain how the Object Oriented programming language and the geometry manipulation rules addressed in

Chapter 3 can be used to achieve the modeling capabilities illustrated in Chapter 4. In particular the modular architecture and the functionality of three HLPs are described. The parametric modeling mechanisms to generate the aerodynamic surface of wings, fuselages and their internal structure are illustrated and commented.

Chapter 6

In this chapter, the software implementation of some Capability Modules is detailed. In particular the functionalities of those CMs that have enabled the integration of the MMG to external aerodynamic and structure analysis codes are thoroughly described. How expert knowledge can be translated into KBE applications that effectively increase designers' productivity is explained here.

In order to demonstrate the capability of the MMG, two relevant study cases are discussed in this chapter. The first concerns with the European project MOB, on distributed multidisciplinary design optimization of blended wing body aircraft configurations. The second deals with the role of the MMG in an MDO system developed in collaboration with Airbus to redesign the vertical tail of an existing passenger aircraft.

Chapter 7

On the light of the MMG capabilities explained and demonstrated in the previous chapters, chapter 7 provides some considerations and guidelines for an appropriate exploitation of KBE technology. Typical cases are described where KBE has the best chances to make an impact, as well as cases where KBE might not be the best solution. A methodological approach to the development of KBE applications is then provided, based on the state of art in industry. To conclude, this chapter presents an overview on the trends and evolution of KBE technology and a list of recommendations and expectations for KBE systems of the next generation.

Chapter 8

In this chapter the main achievements of this research work are summarized and some conclusions are drawn. The recommendations section includes a glimpse on the current state of development of a new generation MMG system, still based on the HLP and CM concepts introduced in this work, but implemented in a latest generation KBE platform.

CHAPTER 2
From the traditional aircraft design process to the MDO approach. Paradigm of the Design and Engineering Engine

1. Introduction
2. From the traditional aircraft design approach to the promise of MDO
3. Towards innovative aircraft configurations. Role of MDO in design innovation
4. Evolution and current state of MDO technology in aircraft design
5. Towards suitable design systems to support MDO and design space exploration
6. The Design and Engineering Engine solution
7. The keystone role of the model generator and development challenges
8. Development of the DEE Multi Model Generator: beyond the capabilities of conventional CAD
9. A brief discussion on non-technical barriers to MDO

2.1 Introduction

In the design process of a complex product, such as an aircraft, a car, or a generic mechanical component, a diverging and a converging phase can be generally distinguished, as represented in Fig. 2.1 (van Tooren, 2003).
During the first conceptual phase, many potential solutions are synthesized to find best compliance with the list of requirements provided by the market/customer. The broader the amount of proposed solutions, the higher the chance to have enclosed the most appropriate or the closest to the best. On the other hand, the broader the amount of proposed solutions, the larger the design and analysis effort.
In fact, all these solutions, which can be either variants of one product concept, or completely different configurations, must be analyzed (and possibly optimized) in order to perform a fair trade-off. This will initiate the converging phase of the design process, where the best solutions are selected for the next design level. This diverging-converging process is actually strongly iterative and requires a continuous

adaptation and modification of each proposed configuration during the design loops. Large amounts of data and information are continuously generated and exchanged across various discipline experts with their multitude of dedicated design and analysis tools. The generated output from one tool often needs to be re-processed in order to be transformed in usable inputs for others.

As a matter of fact, the typical design process is much more complex than sketched in Fig. 2.1, because many diverging-converging blocks are typically required to address the product at hand in all its major systems, subsystems, and components. Besides, the results of a given diverging/converging block can actually demand the iteration of some previous blocks. For example, when the overall configuration of an aircraft has been selected through the diverging/converging process described above, another diverging/converging phase will start for the design of the wing structure, and again another for the design of the manufacturing tooling, and so on.

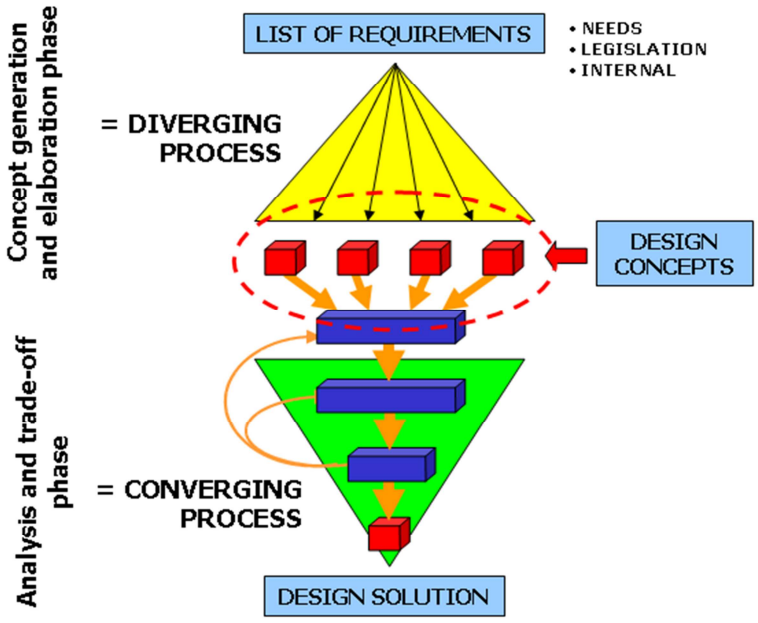


Fig. 2.1: the typical diverging/converging phases of the design process

Although the challenges of the engineering design process are generally well acknowledged, the current/traditional approach still shows inherent limitations in handling such complexity with efficiency and effectiveness. In this chapter the specific issues of the aircraft design process will be addressed.

First the organization of the traditional aircraft design process in industry and its actual effectiveness will be addressed. Having considered the current design process limitations, the advantages, as well as the associated challenges, of the Multidisciplinary Design and Optimization (MDO) approach will be illustrated.

The capabilities of current design systems and tools to implement the MDO approach will be discussed and, based on evidence from literature and working experience, a list of needs for future MDO tools development is compiled.

Then, the *Design and Engineering Engine* (DEE) will be presented, which is an innovative design system concept, currently under development at the Design of Aircraft and Rotorcraft group of TU Delft, to support aircraft MDO. In particular the

role of the *Multi Model Generator* (MMG), a cornerstone module in the DEE infrastructure, and its development challenges will be discussed. The *conceptual development* and *technical implementation* of the MMG, which actually constitute the main achievements of this research work, will be addressed later, in Chapter 4 and 5.

2.2 From the traditional aircraft design approach to the promise of MDO

2.2.1 The traditional aircraft design approach

The current aircraft design process, as it is generally presented in the main reference text books on aircraft design (Torenbeek, 1982; Raymer, 2006) is organized in three main phases, namely the conceptual, the preliminary and the detail design phase (see Fig. 2.2). This phases' distinction is not just an academic argument. Aircraft manufacturers present their product development programs actually organized in this way, as confirmed by Fig. 2.3, which shows the various steps and milestones of an Airbus aircraft development program[1].

Discrimination between the three abovementioned design phases is related to the differences in the different activities that take place, the differences in the tools that

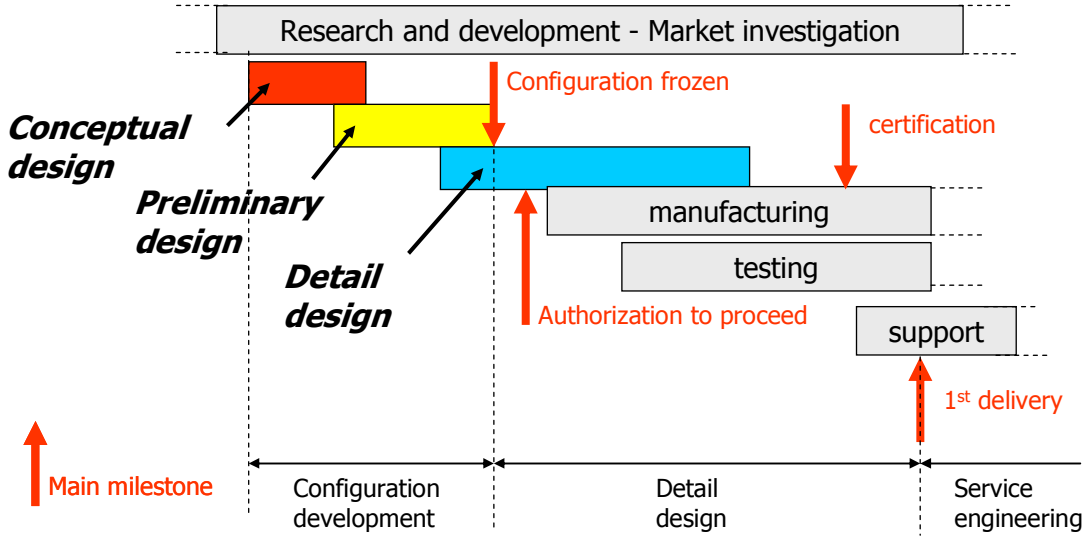


Fig. 2.2: Main phases and milestones in the traditional aircraft development process

[1] In the Airbus process, the conceptual design phase ranges from the second to the fourth technical milestone (M2 - M4). The preliminary phase starts, with some overlap, on M3, in correspondence of the selection of the most appropriate aircraft concept. This second design phase ends at M5, when a detailed and validated aircraft concept is delivered. If the market is favorable and finance for the project is assured, the management can give the Instruction to Proceed (ITP). The ITP triggers the detail design phase, which ends at M7, with the completion of the design of all aircraft components.

are used, the differences in the amounts of people and expertise that take part in the process, the different time scale and, consequently, the different costs involved. See Table 2.1 for a few details.
In the first, the conceptual, design phase, creativity plays an important role. It is here that many different aircraft solutions are proposed and briefly investigated. Here the design is so fluid that everything is allowed to change, including the very topology of the aircraft. However, in order to keep such flexibility affordable, the level of detail is kept very low, as well as the fidelity of the employed analysis tools. As a matter of fact, the aircraft in the conceptual phase is quasi *geometry-less*, in the sense that is mostly described by simple parameters and analyzed/sized (also guesstimated!) by means of simple equations. Simple CAD models are generated to visualize the final results of the conceptual design, rather than to facilitate the conceptual design process.

Activity	Time scale	People involved
Conceptual design: • Definition of the performance goals • Generation of many possible concepts • Evaluation of possible competing concepts • Selection of a baseline design (3 views + data)	Weeks→months	1% of the engineering staff
Preliminary design: • Refined sizing of the baseline design concept • Parametric studies • Global design frozen with the possibility to change only a few details	Months→Months/years	≈9% " "
Detail design: • Detailed design of the whole aircraft down to each single detail • Accurate evaluation of performances • Fine tuning of the design • Release of production drawings	years	≈90% " "

Table 2.1: summary of the main activities, duration and resources for the three main design phases within an aircraft development program

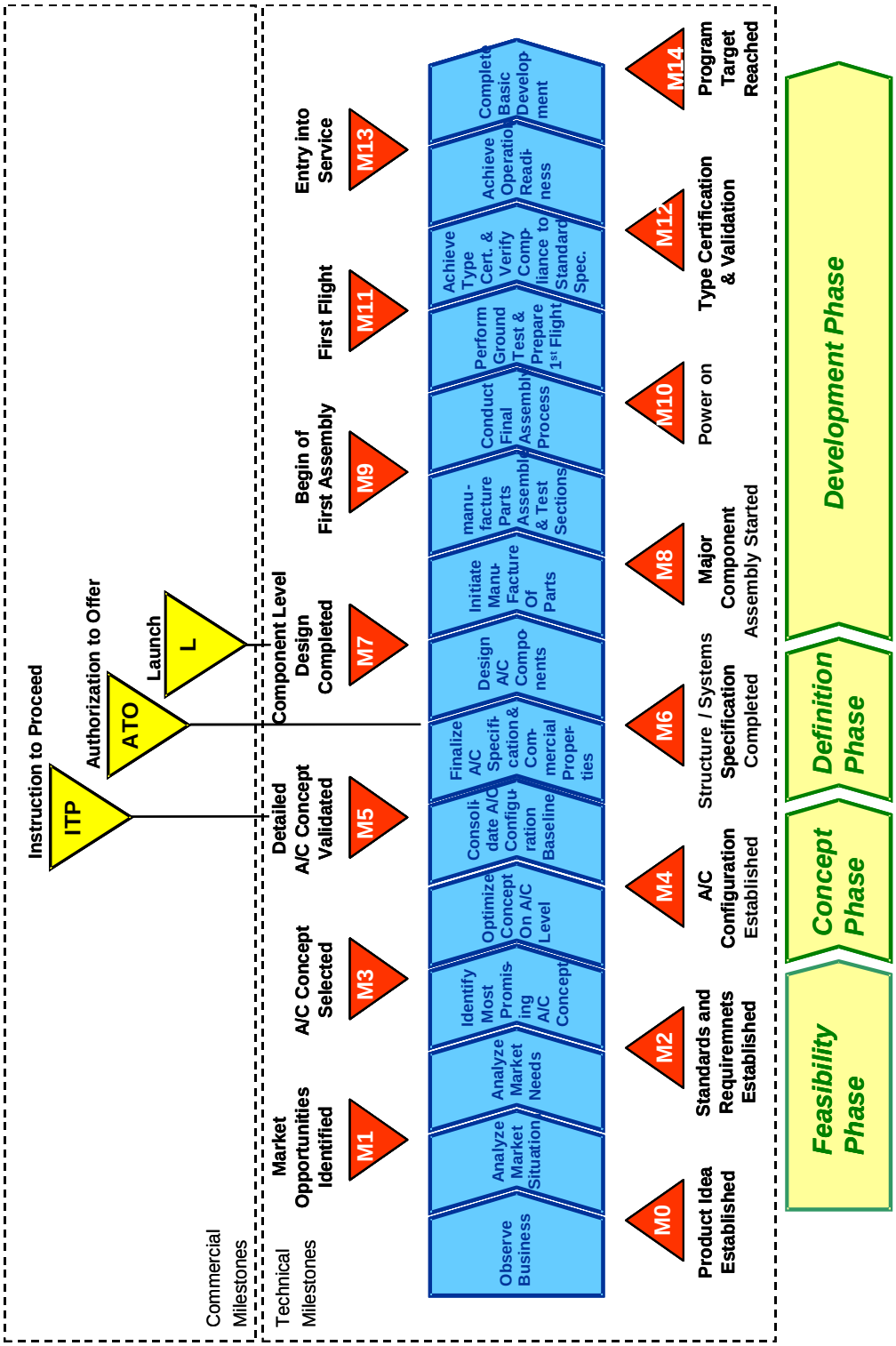


Fig. 2.3: Activities and milestones in the development program of an Airbus aircraft (courtesy of Airbus).

The accuracy and the level of detail produced during the conceptual phase are so low that it is impossible to decide about the quality of the design and make any production commitment. That is why the preliminary design phase is necessary, where the designs synthesized during the conceptual phase are investigated with the highest level of accuracy. Here all the discipline specialists enter the design arena with their sophisticated suite of tools, and testing is initiated (typically most of the wind tunnel tests hours are logged in this phase). However, the amount of time and resources required by this multidisciplinary analysis and testing is so high that it would not be possible to assess a large number of aircraft configurations. That is why from the conceptual design phase only one (or very few) baseline design(s) can be accepted, whose configuration is not going to be varied that much. A canard concept does not turn into a conventional design during preliminary design.

At the end of the preliminary phase so much should be known about the technical and economic performances of the design, that management should have the confidence to give the *Instruction to Proceed* (ITP). As from this moment, the most expensive part of the whole aircraft development process starts: huge resources must be committed to transform the design into a producible product and initiate the manufacturing phase. One of the most evident characteristics of the current design approach is the unbalanced involvement of the various disciplines during the design phases. This is illustrated in Fig. 2.3, from (Schrage et al., 1991).

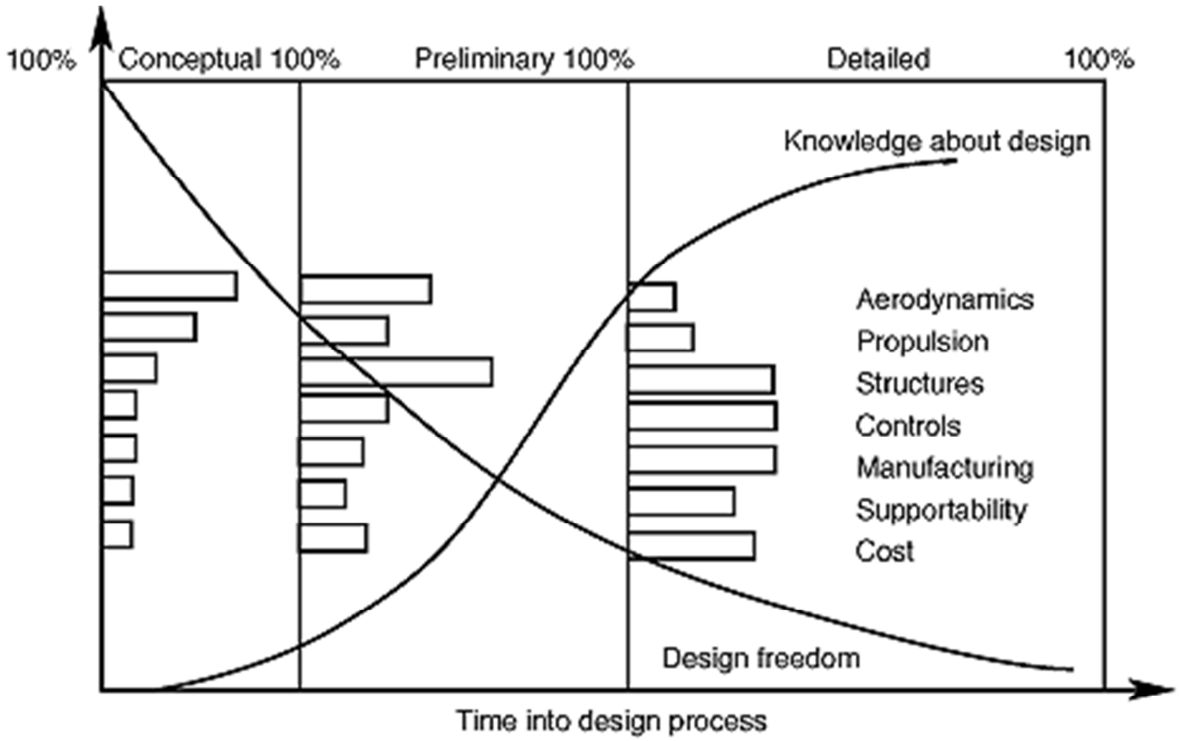


Fig. 2.4: Re-distribution of the disciplines across the main phases of the traditional aircraft design process (Schrage et al., 1991).

In the traditional approach, the synthesis and optimization of the overall aircraft design concept is based on achieving compliance with the customer top level requirements (such as payload, speed, range, etc.) through parametric variation of a few critical design parameters, such as wing loading, thrust-over-weight ratio and aspect ratio. Since aerodynamics and propulsion are generally the two most critical disciplines to achieve the required vehicle performance, they take the largest attention during the conceptual phase. As the baseline configuration is selected and enters the preliminary design phase, structures discipline begins to play a more dominant role. It is only during the detailed design phase that controls discipline gets the same attention as structure. Eventually, also manufacturing plays a primary attention role in this phase, in view of the preparation activity for production.

This design approach has proven satisfactory for many years, however, due to the changes "at the boundary conditions" discussed in Chapter 1, it is showing increasingly evident flaws and limitations, which are preventing any further sustainable growth in the field of aircraft design. Indeed, aerospace vehicles are engineering systems whose performance depends on the not always evident interaction of many parameters. Very large sets of coupled and complex governing equations would be required to model the behavior of such systems. In order to manage complexity, engineers deal with these equations by partitioning them into subsets corresponding to the major disciplines, such as aerodynamics, structures and flight controls. Nevertheless, the couplings among the subsets would be too burdensome to be accounted fully. Hence, during this process of pragmatic partitioning, couplings are retained or neglected judgmentally, on the basis of what is known - or just assumed - about their strength in a particular vehicle category (Schrage et al., 1991).

It is evident that the traditional aircraft design process has developed as *a compromise between design freedom and complexity affordability*, but how effective is this compromise?

Already at the beginning of the 1990s, the aircraft design community has started questioning the validity of the traditional design approach and highlighting the main limitations:

- The conceptual design phase is far too short, especially on the light of the enormous impact that any design decision taken in this phase has on the overall success of the development program in terms of technical performance as well as cost. Fig. 2.5 from (Schrage et al., 1991) shows that at least 70% of the lifecycle costs are already committed during the conceptual design phase, while only 1% of the total costs are incurred (see also (Staubach, 2003)).
- The quality of the baseline concept, which is generated during the conceptual phase and often undergoes just some tweaking during the rest of the design process (Vandenbrande et al., 2006), is mainly based on designers' experience and the results of simple analytical models. However, these often oversimplified

analytical models, typically address only a few disciplinary aspects of the aircraft, as shown in Fig. 2.4. Hence simple conceptual design tools become just inadequate, as soon as requirements are given on aspects like noise emission, manufacturability costs, or some other of the various *–ility* requirements (e.g., maintainability, evolvability, supportability, observability) that are becoming increasingly important both for civil and military applications.

- Considering the fact that new aircraft design programs are fewer and farther apart in time, past experience is less available as the main guide in making design decisions. Considering the increasing complexity of new aircraft and the fact that the more advanced the vehicle, the more complex and relevant the coupling between disciplines, the possibility to judge a priori which couplings can be neglected or simplified, is fading.
- The current design approach is responsible for the so called knowledge paradox: as the designer increases his knowledge about the design, at the same time he loses the freedom to act on that knowledge (see the two curves in Fig. 2.4). It has been demonstrated also mathematically that this approach may actually lead to suboptimal design (Schrage et al., 1991).

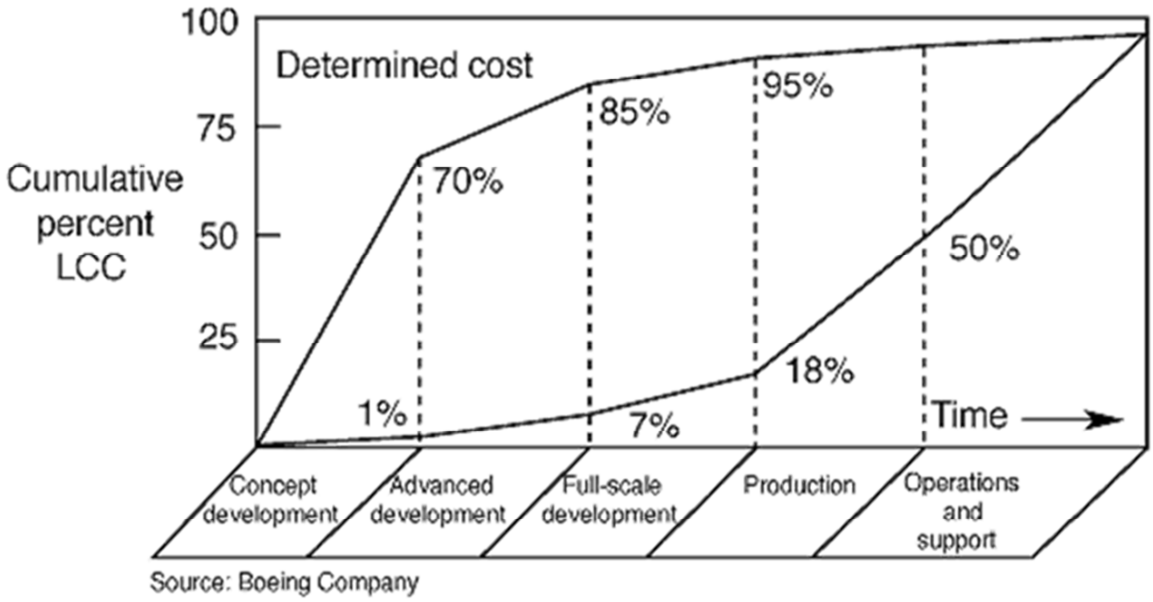


Fig. 2.5: Leverage of design decisions in the developing process: life cycle-cost committed versus incurred by life-cycle phase (Schrage et al., 1991).

2.2.2 The Aircraft multidisciplinary design and optimization approach

The need of a true, systematic, integrated Multidisciplinary Design and Optimization (MDO) approach started becoming evident at the beginning of the 1990s, when the

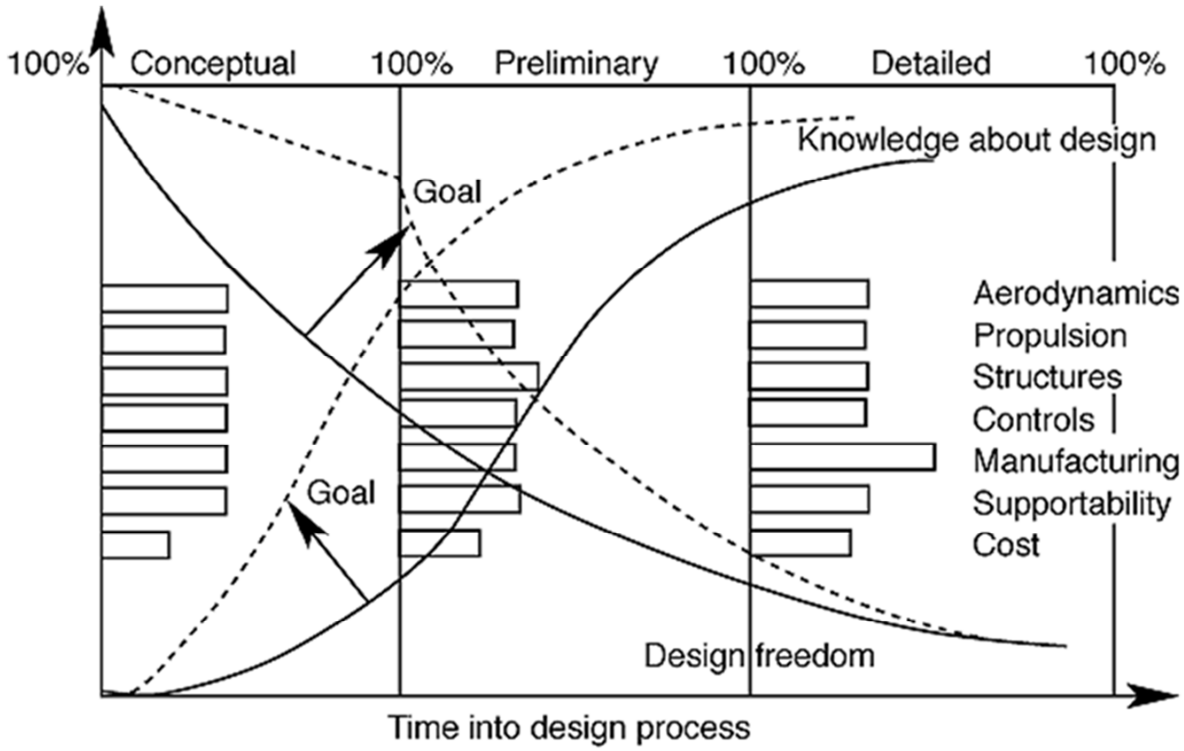


Fig. 2.6: the dashed line projection from the "Knowledge about Design" reflects the requirement that more knowledge will have to be brought forward to the conceptual and preliminary design phases. The dashed line projection from the "Design Freedom" curve reflects the need to retain more design freedom later into the process in order to act on the new knowledge gained by analysis, experimentation, and human reasoning (Schrage et al., 1991).

main design objective started shifting from pure vehicle performance to a balance between performance and life cycle costs. The experience of the 1960s, particularly with military aircraft, had shown that a design driven solely by performance (e.g., speed) becomes usually unattractive in terms of other characteristics, such as, for example, manufacturability or operational costs. The lesson learnt was that the overall system must be optimized, not just performance. The same lesson had been learned earlier by the airlines when meticulous cost accounting had pointed towards high potential cost savings linked to improved reliability and maintainability (Schrage et al., 1991).

At date, the great interest in complex aircraft configurations and technologies such as blended wing body aircraft, morphing wings and aeroelastic tailored composite structures, just increases the urgency of a suitable design approach to integrate *more disciplines earlier in the design process* and to account systematically for their mutual interaction.

Multidisciplinary Design Optimization is then proposed as a design methodology able to solve the abovementioned knowledge paradox and reduce design time, hence

reducing the overall design process duration or allowing in the typical time lap the evaluation and optimization of more design configurations.
As shown in Fig. 2.6, the MDO approach supports a longer conceptual design phase and systematically anticipates the participation to this phase for many of the disciplines that are traditionally confined to the last design stage. In the words of the first AIAA Technical Committee on Multidisciplinary Design and Optimization, "*MDO is seen as a means to achieve the above compression by bringing more information about the entire life cycle and the vehicle performance and cost aspects earlier into the design process. This will enable engineers to make design decisions on a rational basis that gives equal consideration to all the influences disciplines exert on the system, directly, or indirectly through their complex interactions. Doing this early in the process exploits the leverage of the uncommitted design variables. On the other hand, it is equally important to extend the MDO-based approach to the later phases of the design process in order to take advantage of the new information that becomes available during that process through creative thinking, analysis, experimentation, and exploration of alternatives. In order to do that, the design variables that in the conventional design process are decided and set early, need to be retained as free variables much longer into the process* (Schrage et al., 1991)".

2.3 Towards innovative aircraft configurations. Role of MDO in design innovation

In three steps, Fig. 2.7 summarizes 50 years of "evolution" in the configuration layout of passenger transport aircraft. As a matter of fact, the classical "cylindrical fuselage - cantilever wing - aft tail" configuration seems to have no alternative in civil transport aviation, even though proposed in the largest assortment of sizes. This is even truer ever since the retirement of the Concorde. A study on 40 years evolution of the figure of merit M×L/D (i.e., the product between Mach and the maximum lift-to-drag ratio) seems to confirm also a substantial stagnation in the field of aerodynamic design. See Fig. 2.8 from (Liebeck et al., 1998).

In the last decades, several new and "unorthodox" aircraft configurations, like the blended wing body, oblique and joint wings aircraft, have been proposed by visionary designers, not only for military applications. Several internal programs (Boeing/Nasa BWB (Liebeck, 2004)), as well as a number of collaborative research programs (Table 2.2), have been looking into these unconventional configurations. However, the transition from a research study to the industrialization of a new, non-conventional passenger aircraft is not yet at the horizon.
The fact today's aircraft still look like those of 50 years ago, is too often given as argument for the inherent superiority of the conventional design. As far as market and customers' requirements stay the same, this might be true, but considering the changes in requirements already discussed in Section 2.3 of Chapter 2, it is very

unlikely the "Kansas City" configuration still remains the best possible (Morris, 2002; Greener-by-Design-group, 2005). Considered the large amount of specific knowledge and experience matured so far by the major aircraft manufacturers, the actual conservatism is understandable. The financial risk associated to the development of a novel configuration of *unproven advantages* could be unacceptable.
In the end, everything boils down to the need of an adequate design approach to mitigate that risk, by generating at least *digital experience* of some interesting novel configurations and estimate their advantages using the most accurate and reliable analytical model. However, the traditional design approach is too much dependent on the legacy of previous programs and loses most of its validity as soon as the new design starts deviating too much from reference designs.
Furthermore, new designs, as the abovementioned blended wing body, are supposed to achieve a technical and economic quantum leap just by exploiting the synergistic interaction between system components and functions. Conventional techniques

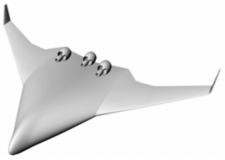
	MOB: A computational design engine for Multidisciplinary design and Optimisation of Blended wing body configuration (Morris, 2002)	EU FP5
	VELA: Very Efficient Large Aircraft (Greener by Design, 2005)	EU FP5
	ROSAS: Research Of Silent Aircraft concepts (Brodersen et al., 2005)	EU FP5
 	NACRE: New Aircraft Concepts Research (Greener by Design, 2005)	EU FP6
	SAI: Silent Aircraft Initiative (Dowling and Hynes, 2006)	UK DTI (UK-US initiative)
	The Prandtl Plane (Frediani, 2004)	Pisa and other Italian universities

Table 2.2: Innovative aircraft configurations investigated in recent European and national research projects.

The European project MOB (Morris, 2002; Morris et al., 2004; Laban et al., 2002) has clearly demonstrated that designing a blended wing body aircraft is an extremely complex task just because of the highly integrated nature of its configuration: aerodynamics, control and stability, structures, propulsion, payload layout, etc. are all strongly coupled.
Small changes to improve one aspect have strong impact on the others. The development of a computational system able to capture and master the complex interaction between the disciplines is a major necessity to address the design of such an integrated configuration.

appear to be inadequate to deal with such integrated configurations.

Therefore, to bring the aerospace community on the 3rd S-curve, a truly MDO approach is necessary, as well as the development of new tools and design systems able to support such an approach (van Tooren, 2003; Morris, 2002; Bowcutt, 2003; Doherty and Dean, 2007). As far as *the only tool in use is a hammer, everything will keep on looking like a nail* (Carty and Davies, 2004).

On the contrary of the previous provoking statement, Fig. 2.8 is actually a proof that MDO can improve the design quality of aircraft design. Indeed during the 30 years of design evolution considered in the plot, the fuel consumption of aircraft has halved, not only because of better engines. Kroo argues the surprisingly constant value of M×L/D is not a reflection of stagnation in aerodynamic design, but rather an indication that the major aircraft companies do a good job of multidisciplinary design

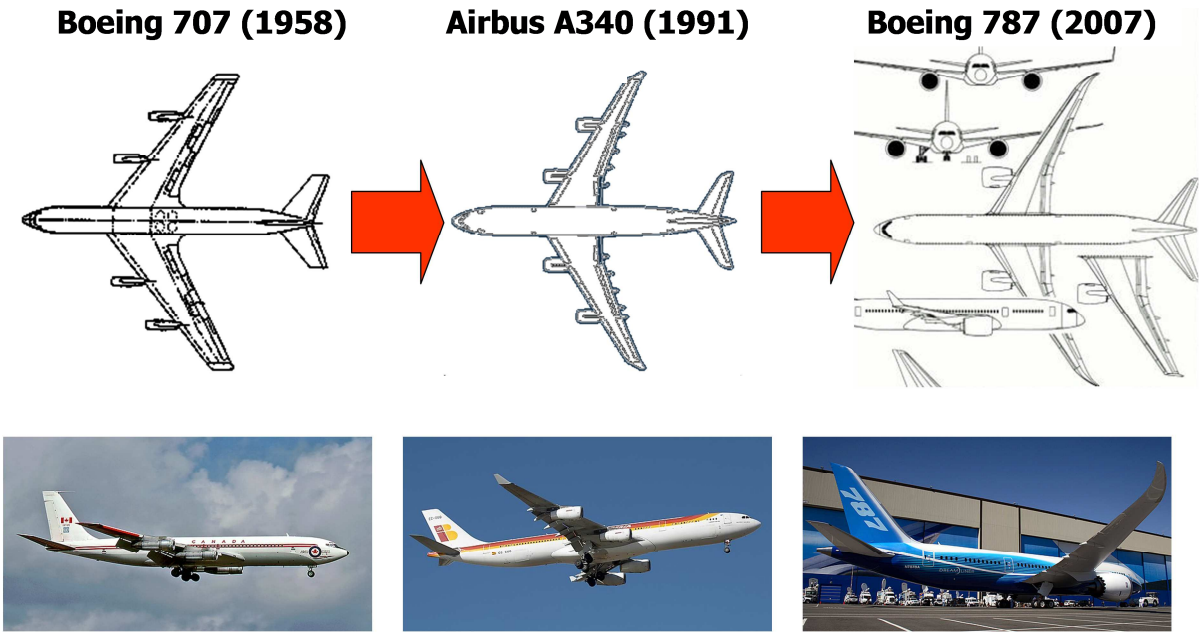


Fig. 2.7: 50 years of evolution in the configuration layout of passenger transport aircraft

(Kroo, 2004). Indeed, improvements in design methodology and understanding of transonic flow phenomena could have produced gains in M×L/D of at least 15% to 20%. Instead, advances in cruise aerodynamics have been largely exploited in parameters associated with other disciplines. For example, supercritical airfoils have not been used to increase cruise Mach number, which is actually not beneficial with the increased engine bypass ratios. In fact, the higher drag divergence Mach values have been exploited to increase wing thickness and lower wing sweep, hence reducing the wing structural weight and requiring simpler hence cheaper high lift devices.

However, while the use of MDO on current mature design can yield improvements in the order of 1-2%, Phantom Works Scientists (Bowcutt, 2003; Vandenbrande et al., 2006) estimate 8-10% gains for new but evolving designs and 40-50% for radically new and undeveloped concepts, like blended wing body and hypersonic vehicles! Eventually, MDO appears to be the most promising design technology to deal with the very large design space associated to many strongly coupled variables, where there is the chance to discover unique and non intuitive design solutions.

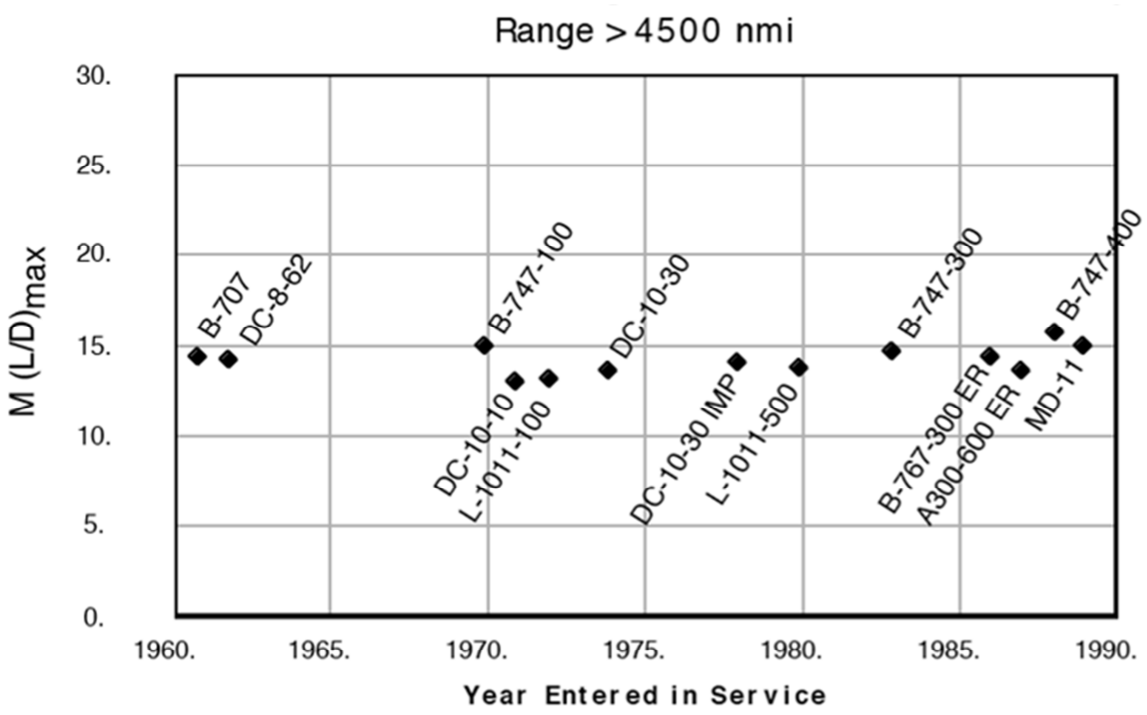


Fig. 2.8: Evolution of M×L/D for long haul commercial transport from 1960 -1990 (Liebeck, Page and Rawdon, 1998) .

2.4 Evolution and current state of MDO technology in aircraft design

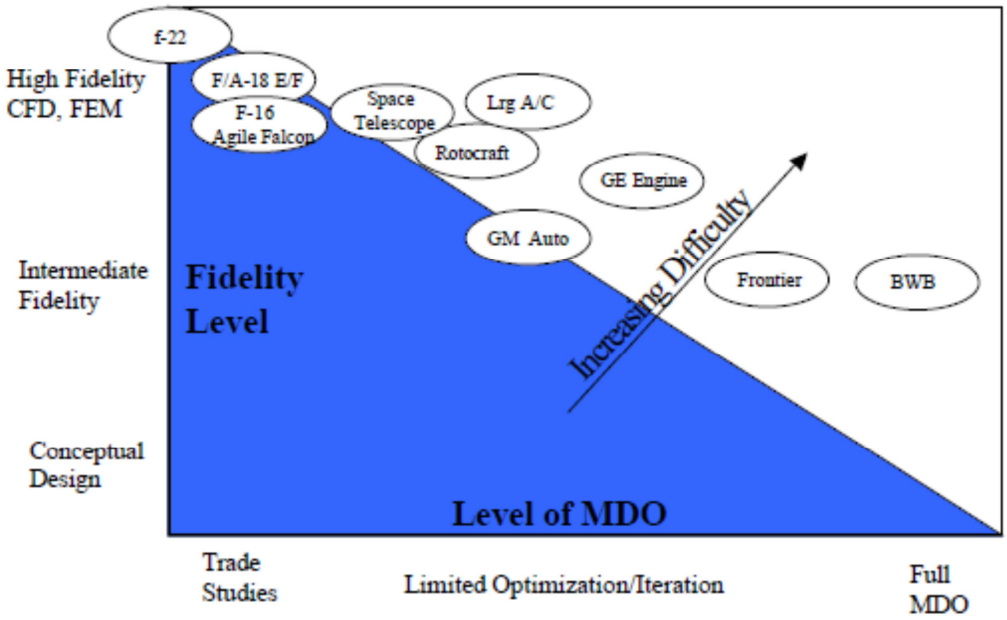
In (Kroo, 1997) there is mention of a relevant AIAA Wright Brothers Lecture from 1982, entitled "*On Making Things the Best—Aeronautical Uses of Optimization*" (Ashley, 1982), where the author surveyed more than 8000 (!!) papers dealing with optimal control, aerodynamic and structural optimization, however, without any single case where the formal MDO procedure discussed in the paper lead to a real application in industry.

Also the outcome of the first 30 years of MDO application (1970–1997) in *conceptual and preliminary* aircraft design was not impressive. The two possible main reasons:

- Too low fidelity level of the analysis methods usable in the MDO framework. Obtained results were not credible, especially those concerning fundamental aerodynamic figures such as drag and C_{Lmax}, which cannot be produced with low fidelity analysis methods.
- Slow computer allowed handling MDO problems with 5-10 variables, hence problems almost solvable by hand (at least one order of magnitude higher was needed to make the formal MDO approach really interesting).

In 1998, less than ten years after the first TC-MDO white paper, a second one was published in (Giesing and Barthelemy, 1998), entitled "*A summary of Industry MDO applications and needs*" This paper provides a picture of the state of the art of MDO utilization in industry and is based on a critical review of ten relevant industrial applications of MDO (not only aerospace related). The results of the review were summarized in Fig. 2.9. Though MDO had started getting the interest of industry, there were strong limitations in setting up a true MDO system based on high level fidelity analysis. Eventually, high fidelity tools could only be used for monodisciplinary optimization or simple tradeoffs. Real multidisciplinary optimization could only be achieved by using low fidelity analysis tools, which is in contrast with the idea of bringing as much knowledge-about-the-design as possible to the early design stage (Fig. 2.6). The paper highlights a broad set of open issues and needs, among which the following two were indicated for outstanding criticality:

- The impossibility to automate the operation of high fidelity tools due to their lack of robustness, as well as for the inherent complexity of generating high fidelity models
- The long computational time required by high fidelity analysis tools



Trade studies: point designs generated and graded relative to each other without formal optimization.

Limited Optimizations/Iterations: disciplinary sub-optimization or optimization with limited disciplinary interaction.

Full MDO: vehicle level optimization with most critical disciplines involved.

Fig. 2.9: Design process fidelity and level of MDO (Giesing and Barthelemy, 1998)

In 2006, during the European-U.S. MDO colloquium, about 70 participants from industry and academia presented and discussed their latest achievements and current state of MDO. Some of the results were collected in (de Weck et al., 2007). The conclusion was that the actual application of MDO methods and techniques in industry had definitely started, though yet hampered by many of the issues previously highlighted in the TC-MDO white papers. Indeed it was observed that the use of genuine MDO methods within industry at large is still rather limited and, for the most part, started at the detail design stage. Also, it was observed that the use of high fidelity models is not yet achievable at preliminary design level, though there is a clear trend of moving upstream. This is qualitatively illustrated in a slide presented by Boeing (see Fig. 2.10), showing the continuous advances across successive aircraft families, towards the target of a full MDO/True physics modeling designed aircraft. It was explained that in the development of the 787, the MDO approach was indeed used with success to improve the design of a few aircraft subsystems. However, major advances are still required to apply MDO based on high fidelity analysis tools to a complete aircraft configuration.

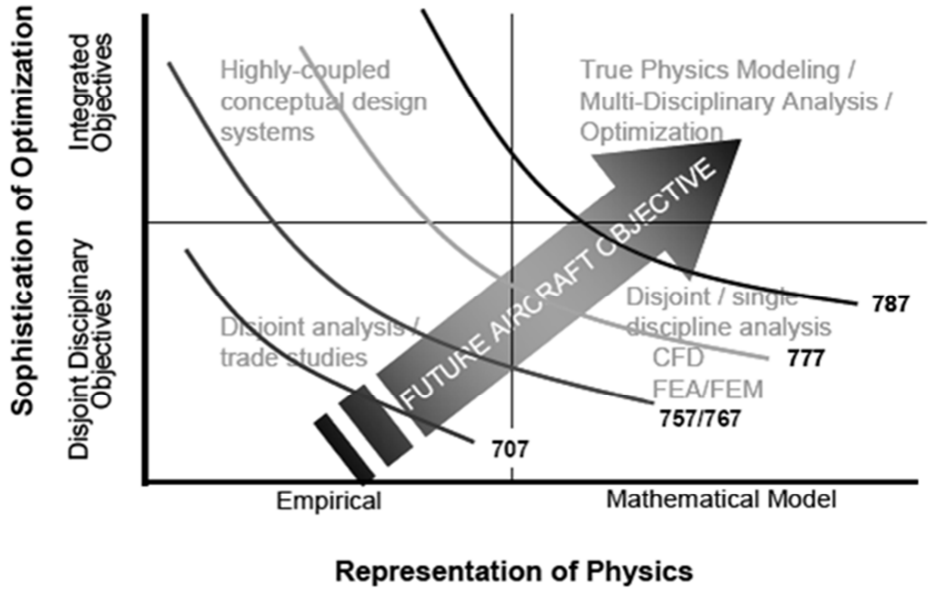


Fig. 2.10 Towards full aircraft MDO across successive Boeing aircraft families (de Weck et al., 2007)

At the end of 2008, presentations from TC-MDO members (Alonso, 2008; Bathia, 2008; Gaudin, 2008) at the 12th AIAA MA&O Conference still address the need to move towards the application of MDO in preliminary design founded on high fidelity analysis tools. This, again, calls for higher tools robustness and an enhanced level of design automation, by seamless integration of tools and the use of advanced parameterized/associative models.

In conclusion, advances in research have not yet lead to a large scale exploitation of MDO in the aerospace industry. Though a positive trend is manifest, a number of technical and non technical barriers are still constraining the transition of MDO from a high potential innovative design methodology to a consolidated practice.

2.5 Towards suitable design systems to support MDO and design space exploration

2.5.1 A collection of needs

On the base of the above study of the state of art of MDO in aerospace and the previous considerations on the traditional design approach (Chapter 5 Section 5.2), a list of needs has been compiled by the author, for a design system able to support the MDO design approach.

The following collection of requirements has been structured in four main groups to address the fundamental aspects of any MDO design system, namely overall architecture, analysis capability, geometry modeling and optimization capability.

Overall system architecture

- The system should have a *loosely coupled* modular structure to adapt, i.e., allow reconfiguration and scalability, to different design cases and to the specific needs of the various design process phases
- The system should be able to support *closely coupled* analysis when needed to fulfill high computation speed requirements
- The system should be able to integrate *both of-the shelf and in house developed* design, analysis and optimization tools, as well as data sharing and communication systems
- The system should guarantee the synchronization of the data/models used by the various disciplinary analysis tools to guarantee a consistent design and optimization process
- The data exchange among the various MDO system components should be based on standard data representation formats
- The system should support *automation* of all the repetitive activities related to the iterative nature of the MDO approach
  - This should include pre-processing of data and models as required to feed different design and analysis tools
  - This should include post-processing of the data generated by the various design and analysis tools
  - This should include the transfer and storage of data between the various design and analysis tools
- The system should make use of dedicated software frameworks for the integration of the various analysis and design tools involved, i.e., to support process coordination and communication among the various design, analysis and optimization tools.
  - The system's framework should be able to control process execution across the *distributed networks* of software tools
  - The system's framework should be robust and easy to set up
  - The system should provide visibility of the overall, complex design process workflow

Analysis capability

- The system should not have any limit on the number of disciplines that can be integrated

- The system should allow the use of analysis tools with different levels of fidelity, with the possibility to switch level (possibly automatically, based on the results of some accuracy sensitivity analysis)
- The system should support the use of the highest fidelity analysis tools
  - The system should account for the lack of robustness of current high fidelity tools
  - The system should account for the difficulty of automating the operation of current high fidelity analysis tools
  - The system should account for the large computation time normally required by high fidelity analysis tools (e.g., support parallel, grid computing)

Geometry modeling

- The system should provide a sharable common vehicle description to facilitate communication among all disciplines (and among companies, sites and design team involved)
- The geometry model should support the use of both low and high fidelity analysis tools
- The geometry modeling system should not constrain the user to conventional aircraft configuration
- The geometry model should have parametric/associative characteristics to maintain accuracy and consistency as design variables are changed
- The geometry model should support the level of automation and robustness required for the use in a MDO framework
- The parameterized geometry description should be compatible with current CAD systems and transferrable through standard data exchange formats

Optimization

- The system should be able to deal with the lack of robustness of many current optimization packages
- Hybrid optimization schemes should be supported, able to deal with continuous and discrete design variables
- The system should be able to support multilevel design decomposition and optimization
- It should be possible to provide designers with visualizations of the design space and not only with single optimum points, to facilitate them judging the robustness and the sensitivity of the reached design point

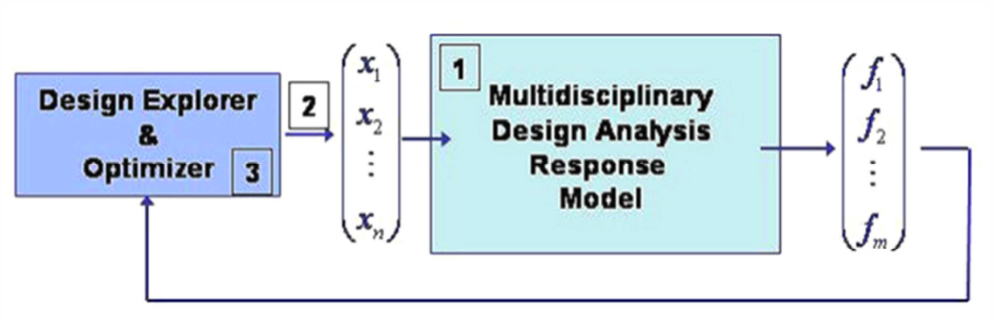


Fig. 2.11: generic architecture of an MDO system (Vandenbrande, Grandine and Hogan, 2006)

2.5.2 The different paradigms of current MDO systems: strengths and limitations

Without entering in the details of any specific implementation, a generic MDO system for aircraft design is built up of the following three functional components (Vandenbrande et al., 2006) illustrated in Fig. 2.11:

1. A modeling and analysis component able to compute the multidisciplinary behavior of multiple aircraft designs (indicated by the responses *f1, f2, .., fn*)
2. A design points generator to sample conveniently the design space and define the aircraft variants - indicated by the variables vectors (*x1, x2, .., xn*) - to be modeled and analyzed by the component above
3. An optimizer to spot the most promising area in the design space, based on the feedback responses. Optimizers often perform both this function and the one above

When compared with the traditional aircraft design approach where the designer is in charge of judging which and to what extent the design variables should be changed in order to improve the aircraft performance, an MDO system is based on a *systematic* search approach, enabled by the modeling and analysis system.

Today, different implementations exist of the generic MDO system of Fig. 2.11. In particular we can distinguish the following three kinds of implementation:

- The *geometry-less* implementation, typically used for conceptual aircraft design
- The *grid-perturbation* implementation, which makes use of a detailed-but-discipline specific geometry representation of the aircraft, particularly suitable for detail design.
- The *geometry-in-the-loop* implementation, which offers the possibility to feed the various disciplinary analysis with geometry models that are updated by the optimizer during each cycle

In Ref. (Vandenbrande et al., 2006), Boeing scientists provide a description of the abovementioned systems, as well as reference to some examples in literature.

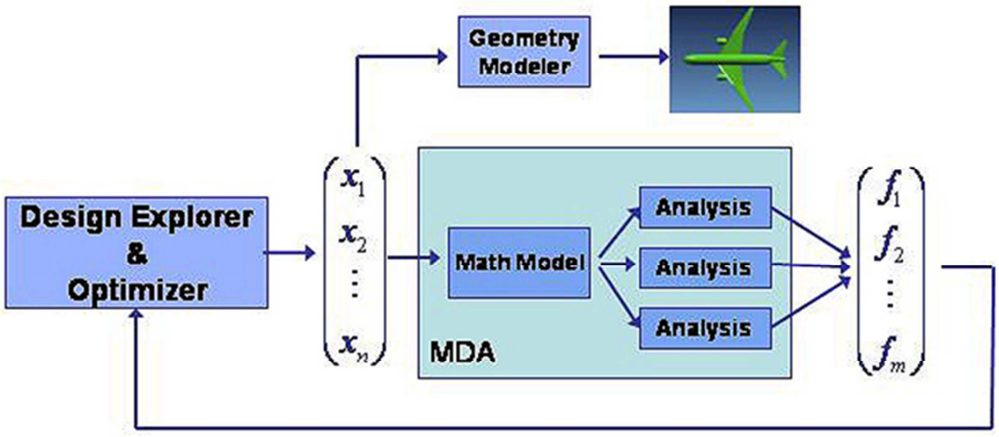


Fig. 2.12: implementation of geometry-less MDO system. Geometry generation, if present, is outside the loop (Vandenbrande et al., 2006)

Geometry-less implementation

In the geometry-less implementation, the aircraft is described by sets of coupled equations including parameters/variables like weight fractions, lift coefficients, wing loading values, etc., which are varied by the optimizer in order to minimize/maximize a given objective function. The analytical tools used in this sort of implementation can compute optimal values of geometric variables, such as wing span and sweep angles, eventually used to generate simple drawings for the sole purpose of visual inspection (see Fig. 2.12).

The geometry-less aircraft design approach is only possible and useful when the designer has availability of semi-empirical and statistical models that are based and validated for aircraft configurations similar to the one at hand.

However, the lack of any reference and statistical data makes the applicability of these models extremely limited - if not useless - to *novel aircraft configurations*. In this case, it is necessary to go back to "first principles" and make use of high fidelity analysis tools that need appropriate geometric representations of the aircraft configuration at hand.

Grid-perturbation implementation

The grid-perturbation implementation (see sketch Fig. 2.13) uses a detailed geometry model of the baseline aircraft configuration for the generation of a computational grid. During the optimization (part of) the grid is perturbed to investigate the effect of shape modification on objective functions and constraints. The optimizer perturbs either single grid points or groups of points, when special grid parameterizations techniques are used. Only small perturbations are generally allowed to prevent grid quality deterioration. Complex techniques are required to

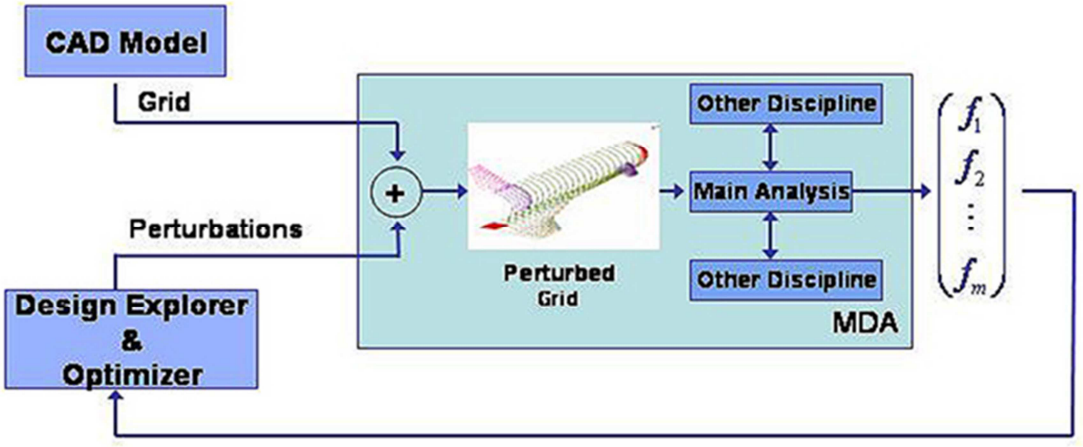


Fig. 2.13: Implementation of an MDO system based on grid perturbation. One CAD geometry is generated for gridding. The grid is used by the discipline analyses during optimization (Vandenbrande et al., 2006)

perturb the grid without creating undesired discontinuities that invalidate computation loops. This MDO system implementation has proven its value especially during detail design, when "sand-paper work" is required for improving a mature aircraft configuration, rather than drastic shape changes or even topology variations (Samareh, 2004; Zang and Samareh, 2001). Another limitation of this system is that one grid is generated (and modified during the optimization process), which is generally tailored to the need of the main analysis code in the system. Therefore, the other analysis tools might have to deal with representations non optimal for their specific needs.

Integrated Geometry Generation implementation

The third method brings the geometry right inside the loop, hence it does not present the limitations of the two implementations discussed above: disciplinary analysis can be performed using actual geometry representations, rather than analytical approximations or extrapolations from statistics or previous designs. These geometry representations can be updated during each optimization loop (if required) and tailored to the needs of the various analysis tools. While in theory this kind of MDO system has the right credentials to fulfill many of the needs listed in Section 2.5.1, it actually brings a huge burden on the geometry generator and turns it into the key element of the whole approach.

Examples have been found in recent literature on advanced aircraft design systems that belong to this third category of MDO implementation. In (Vandenbrande et al., 2006), Vandenbrande, Grandine and Hogan discuss the paradigm of the Boeing design system (Fig. 2.14) and illustrate the functionality of the *General Geometry Generator*, the geometry generator tool developed on

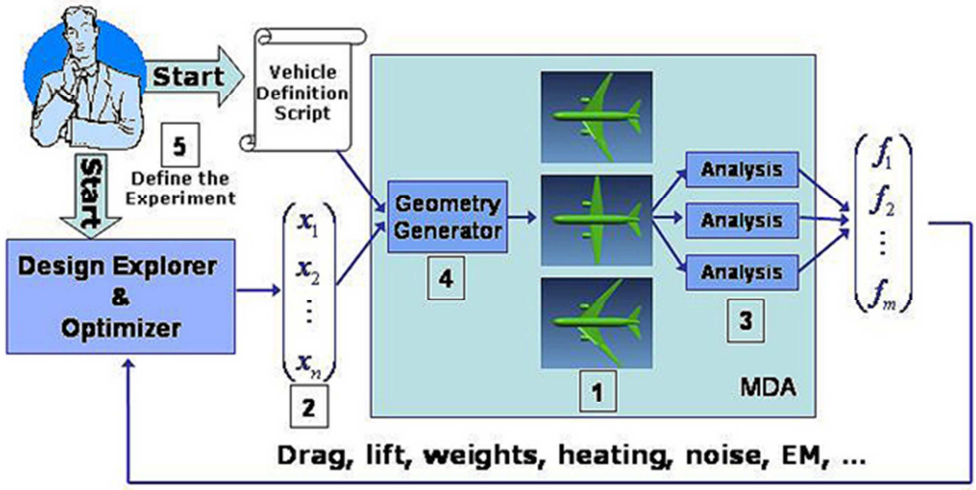


Fig. 2.14: MDO design system with integrated geometry generation capability (Vandenbrande et al., 2006)

purpose. In (Carty and Davies, 2004) , Carty illustrates the advantages of *Rapid Conceptual Design*, the integrated multidisciplinary design system developed at Lockheed Martin. In (Werner-Westphal, Heinze and Horst, 2007; 2008) the *Preliminary Aircraft Design and Optimization* (PrADO) system developed during the last 20 years at the Technical University of Braunschweig is addressed, where an integrated geometry generator communicates with in house developed aerodynamic and FE codes.

Since 2002, the Design of Aircraft and Rotorcraft group of the University of Technology in Delft is also developing a loosely integrated design system to support multidisciplinary design, analysis and optimization of aircraft, called *Design and Engineering Engine* (DEE). The first development activities started within the framework of the European Project MOB and they have kept evolving through other national research projects and collaboration with industry.
The paradigm and functionalities of the DEE will be discussed in the next section. The conceptual development and technical implementation of the DEE Multi Model Generator (MMG) will cover most of this work, being the main contribution of this doctoral research to the DEE development.

2.6 The Design and Engineering Engine solution

The DEE consists of a multidisciplinary collection of design and analysis tools, able to automatically interface and exchange data and information, with the purpose of supporting and accelerating MDO of complex products, through the automation of the non-creative and repetitive design activities. In this thesis, focus will be on an aircraft DEE.

The paradigm of the Design and Engineering Engine is illustrated Fig. 2.15. Note that is a simplified representation of the system; no details are shown of the components' internal architecture and only the main communication lines are drawn. The DEE is an integrated design system built up of loosely coupled modules, which can vary in number and type according to the design case at hand. For example, different analysis modules can be included or left out of a specific DEE implementation as considered opportune. Indeed, what is shown in Fig. 2.15 should be considered the generic template of a design system that can be customized to the user needs. For example, the DEE can be used for mono and multidisciplinary what-if studies or for mono and multidisciplinary optimization studies.

2.6.1 Architecture of the DEE

The main components of the DEE architecture and the way they interact during the MDO process are described in the following subsections (refer to Fig. 2.15)

The Multi-Model Generator (MMG) is the software tool developed in this doctoral research with the twofold intent of providing designers with a *parametric modeling environment* to generate models of conventional and novel aircraft configurations and *automate the generation of input data and specific disciplinary models for various disciplinary analysis tools*. The MMG will be discussed later in full detail.

The Initiator module actually consists of a collection of sizing tools, able to provide an initial set of values for the MMG parameters, i.e., the first of the variables vectors (*x1, x2, .., xn*) mentioned in Section 2.5.2. In fact, the MMG offers the possibility to instantiate an aircraft model based on a given set of parameters values, but does not have any knowledge to select/calculate those values autonomously. Prototypes of various parameters initiating tools have been developed, e.g., to define the fuselage layout of a conventional aircraft given the payload requirements (Alagna, 2005), or to compute a complete aircraft baseline configuration, starting from a limited set of customer and regulations requirements and using simple conceptual design handbook methods supported by optimization techniques (Langen, 2011). Also a structure specific initiator tool has been developed to provide a rough estimation of mass and stiffness distribution in lifting surfaces, based on a preliminary estimation of the aerodynamic loads (Cerulli et al., 2006; Schut and van Tooren, 2007). Eventually, the scope of the Initiator is to provide the MMG with *a feasible* initial solution prior to start with the multidisciplinary analysis and optimization process (see the "*Feasilization"* concept in (Schut and van Tooren, 2007)). Though not shown in Fig. 2.15, the Initiator might also use a MMG (at least part of its capabilities) to extract, for example, some geometry information required to perform the feasilization process. Furthermore, the Initiator can also make use of optimization

techniques and employ a kind of Converger/Evaluator tool (see below). In other words the Initiator can be a DEE by itself.

The ***disciplinary analysis tools*** can be both low and high fidelity analysis computational systems (such as panel codes or CFD), either in-house developed or Commercial of the Shelf (COTS) tools. As anticipated, the DEE does not "contain" a fixed suite of analysis tools, but different tools can (have been) used according to various project needs. Examples of COTS that have been integrated in various DEE implementations are Fluent (Lisandrin, 2007), VSAero (Brouwers, 2007; Dircken, 2008), NASTRAN (Nawijn et al., 2006; Koopmans, 2004; van der Laan and van Tooren, 2005) and Abaqus (Krakers, 2009) (more details in Chapter 6). Though Fig. 2.15 shows the disciplinary analysis tools as separate silos, direct data exchange (i.e., not throughput via the MMG) can occur among them. For example, a structural analysis tool can use the loads computed by an aerodynamic analysis module. In case of highly coupled analysis tools a discipline silo should be regarded as a multidisciplinary tool.

The ***Converger&Evaluator*** functionalities are generally provided by a single off-the-shelf optimizer, whose tasks are checking if the various analysis tools have reached convergence and if the performance/characteristics of the evaluated design have met the objectives set by the designer. This module receives the results generated by the various disciplines (typically post-processed in terms of significant figures of merit, such as aerodynamic efficiency or weight) and generates a new variables vector (*x1, x2... xn*) to feed the MMG, according to the implemented optimization algorithm. The MMG will modify consequently the aircraft model and produce updated data to support a new analysis loop. The generation of the variables vectors (*x1, x2... xn*) can be performed either using a search algorithm (e.g., a gradient based approach) or following a selected strategy to sample the design space (e.g., Latin hypercube).
In case of the impossibility to fulfill the initial requirements, the Evaluator/Converger will quit the interaction with the MMG and call again the Initiator, which will have to synthesize a different aircraft configuration, feed it to the MMG, etc…

The **communication framework**, represented in Fig. 2.15 by the set of connectors linking the various DEE components, takes care of the data and information flow between the various design and analysis tools and manages the overall design process sequence. An innovative agent-based architecture has been implemented to provide the DEE with a very high level of flexibility (Berends and van Tooren, 2007) (see next insert). The agents system allows fast set up and reconfiguration of the overall MDO process, according to the specific design case at hand and the type and location of the available design and analysis tools. The agents use the web to link

sets of heterogeneous design and computational tools, which can be installed on different servers and computers, possibly running different operating systems and belonging to separate networks.

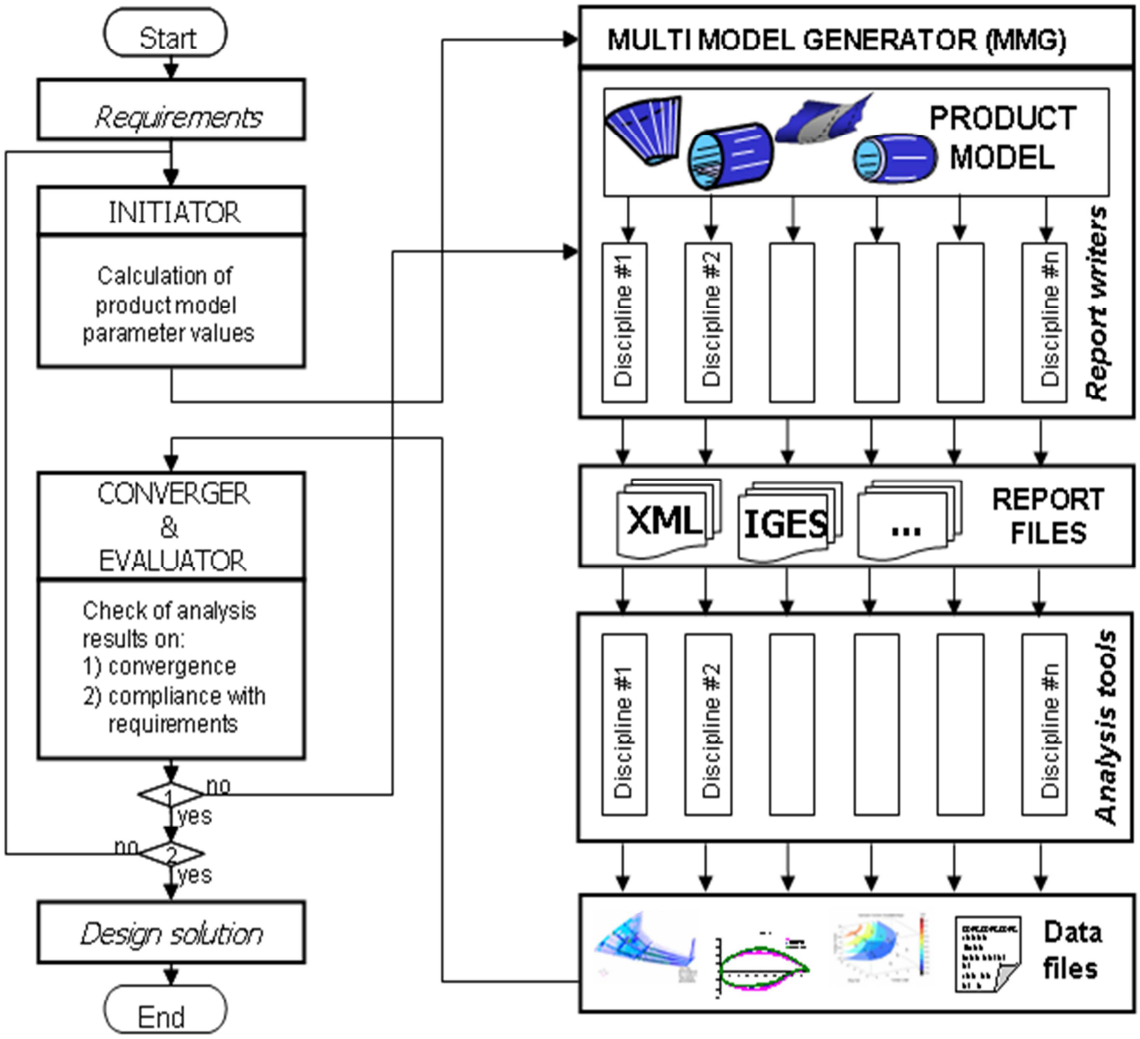


Fig. 2.15: Paradigm of the Design and Engineering Engine (arrows show execution order).

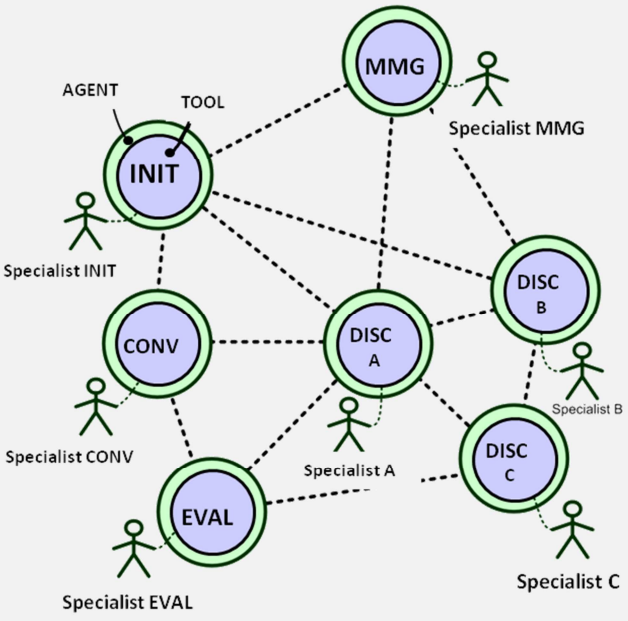
Conceptual and technical development of the DEE agent based framework

The *agent-based* communication framework developed by Berends (Berends, van Tooren and Schut, 2008; Berends and van Tooren, 2007) aims at mimicking the organizational structure of a *design build team* (DBT), where the various human actors work and interact in a flexible *service oriented* approach rather than in a rigidly predefined procedural (throw-over-the-wall) scheme. The figure below shows how the previously illustrated DEE paradigm (Fig. 2.15) is actually a formal and simplified representation of a DBT. In this software implementation of the DTB structure, all DEE specialist's tool need to be wrapped by a software *agent,* which enables interfacing with other tools. Each specialist's tool can be replaced by a different one (or an upgraded version of the same tool) if the interface is kept consistent, hence the agents facilitate a plug-and-play set up of the DEE.

The framework uses a kind of *peer-to-peer/server-client* architecture, where one of the DEE agent-wrapped tools is appointed *server*, while all the others are the *clients*. Each agent-tool combination, in order to join the DEE, must register to the server, from which it receives the updated list of all the other available registered clients (with relative contact data such as host name and IP number and status of activity). The role of server is played by the "most expert" agent (i.e., the longest active agent in the DEE), but in case of necessity each agent in the DEE can equally function as server. If the formerly appointed server agent disconnects from the network, the second eldest agents takes over so that the integrity of the DEE structure and the continuity of the design process is guaranteed (as in the human DBT model).

The agents based framework does not need a predefined description of the process flow, as most currently available integration systems, but uses a *demand driven* (or service oriented) approach to self-configure the process flow on the fly.

Each agent is able to initiate the design process when the output of its wrapped tool is requested, either by another agent, or by the designer. Since each agent is aware of the input needed to start its internal tool, it broadcasts the request for necessary input to all the agents presently alive in the DEE. Each agent is also aware of the output that its wrapped tool is able to generate. In case an agent is able to satisfy the broadcasted request, it will start a *peer-to-peer* connection with the requesting agent. The required input data will be generated and their storage location will be communicated to the requesting client for retrieval.



2.6.2 Modular structure to support flexibility and distributed design

In order to participate in the DEE structure, each of the software components addressed above must be able to operate (also) in *batch mode* and be accessible in remote. That is to say, they must allow hands-off operation (without human interaction) and be accessible and controllable from other computer systems than the one where they are physically installed. This might be a stringent requirement for tools which have been originally developed to operate via graphical user interfaces and expect the user intervention (e.g., by means of selection menu) during operation. However, it is a killer requirement for the automation of any process that involves multiple computational modules or is based on an iterative approach.

The modular architecture is an enabling factor for the DEE flexibility. Modularity comes with the price of interfaces development and, possibly, with a penalty of overall process speed. However, unless extreme computing speed is required, e.g., for real time calculations, a modular architecture is the key to system scalability, adaptability and maintainability.

> *Large, monolithic MDO systems can be difficult to understand, manage and extend. Many grandiose plans for completely integrated aircraft design systems have fallen by the wayside because they quickly became unmanageable.* I. Kroo – Stanford University (Kroo, 1997)
>
> *Requirements instability hampers design, but it is here to stay so it is necessary that design systems and methods are developed accounting for that and support agile reconfigurations.* W. Tam – Aerojet (Tam, 2004)

A modular architecture becomes a necessity for supporting *collaborative* and *distributed design*, where different discipline specialists must be enabled to participate to the design process with *their own trusted* tools (Morris et al., 2004; Bartholomew, 1998). The wish to use trusted tools is typical for collaborative projects that involve different companies, or different departments of the same company, that want to collaborate and contribute with their best practice analysis tools.
Nevertheless, there are continuous attempts by industry and academia to develop complex *integrated* design tools to cover the whole aircraft design cycle, from drafting to high fidelity multidisciplinary analysis and optimization (Butler et al., 1998). The exploitation of these systems within distributed, collaborative design programs appears always very problematic: the eventual substitution of one of the integrated analysis functionalities with an external analysis tool becomes easily an overwhelming problem. Similarly, when only some of the "super integrated tool" functionalities are needed, it might result impossible to disintegrate its monolithic structure into separately usable bits. For this reason, soon after the beginning of the MOB project (Morris, 2002), it was decided to pursue the development of the MMG (thoroughly described in Chapter 4-6), rather than proceed with the Prado system

(Werner-Westphal et al., 2008) as planned. Although mature and sophisticated, Prado did not have the required flexibility to integrate and support the exploitation of the design and analysis tools provided by the consortium partners.

The presence of the initiator tool inside the DEE has the purpose to blend into one system the systematic design space exploration capabilities offered by the MDO approach with the exploitation of the best design knowledge available. The DEE allows the possibility (via the Initiator) to use handbook methods to synthesize a baseline configuration to feed to the "MDO machine"; or also use first principles, but for a simplified problem (i.e., simplified design options, requirements and methods). However, the DEE offers the designer also the possibility to use directly the MMG[1] (as a standalone tool) to build the model of the aircraft configuration he previously sketched "on the back of an envelope" and then proceed with the multidisciplinary analysis and optimization process. In this sense, the DEE offers a possible solution to what Lockheed Martin's specialists indicate as the need to successfully leverage the best design knowledge available, but push beyond results predestined by heritage databases and empirical correlations (Carty and Davies, 2004).
It should be noted how this implementation of the MDO approach blurs some of the distinctions between the conceptual and preliminary design (and even part of the detail design) discussed before. Indeed, the DEE is an attempt to merge the large design freedom of the conceptual design with the systematic multidisciplinary trades, typical of the traditional preliminary design phase.

As a human designer can substitute the functionality of the Initiator, the same is possible for other DEE software components. For example, the designer can also decide which and in what extend to change the design variables, without having an optimizer system in place. Also the expert's judgment or *guesstimation* can be used in place of a discipline analysis tool. Future developments of the DEE implementation are envisioned, where the agent-based framework will directly contact and request the knowledgeable services of a human expert in between the execution of an MDO process.

2.7 The keystone role of the model generator and development challenges

As discussed in Section 6.5, the development of the geometry model generator is of paramount importance to the implementation of a design system like the DEE.

[1] As it will be discussed later, the technology used to develop the MMG allows capturing and reusing design knowledge directly within the parametric geometry modeler. Hence the initiator is not the only occasion to seed some knowledge in the aircraft model, before starting the MDO process.

Advocates of the geometry-in-the-loop approach indicate the geometry generation as the keystone to succeed and often the greatest impediment to integrated design (Bowcutt, 2003). The geometry modeler represents a key technological enabler, as well as one of the most difficult and complex tasks (Samareh, 2004; Vandenbrande et al., 2006). This is generally true for the design of most aeromechanical systems. Other sort of systems, such as wire harnesses, still need a MMG, but pose less demands in terms of geometry complexity (van der Elst and van Tooren, 2008).
A list of relevant needs concerning the geometry modeling aspects of an MDO system was previously provided in Section 2.7. In the next subsections, the associated challenges will be elaborated in more detail, to justify the MMG development approach described in the next chapters.

2.7.1 Modeling flexibility and robustness

A suitable geometry modeler to support design exploration and MDO should not have any representation related restriction on allowable geometry changes. That is to say it should allow investigating any aircraft configuration, conventional and not, without constraining the design process to the available descriptions of the product. This goes beyond conventional parametric modeling and shape deformations and requires the capability to deal with topology changes and product re-configurations.

The level of flexibility discussed above should be combined with adequate *modeling robustness* to survive the harsh perturbations dictated by an optimizer. This requires the capability to maintain *spatial integration* whatever the combination of parameters values. In other words "any feasible combination of parameter and variable values should deliver a healthy geometry model". Ensuring spatial integration can be a rather complicate task, though partly achievable using a smart definition of the model parameters (also addressed in literature as *hypercube parameterization* (Bowcutt, 2003). More in Chapter 4) and by embedding in the modeler some knowledge to deal with the limitations of the employed CAD engine (e.g., to trap errors generated by inaccuracy and missed intersections. More in Section 4.10)

The modeler should offer the capability to embed also some engineering knowledge, such as, for example, mathematical rules to generate particularly engineered shapes (or to compute the number/position of certain geometry features. The possibility to embed knowledge in the geometry modeler (e.g. by combining parameters with rules) would also relieve the optimizer from the burden of too many constraints and would simplify the set up of the optimization problem.

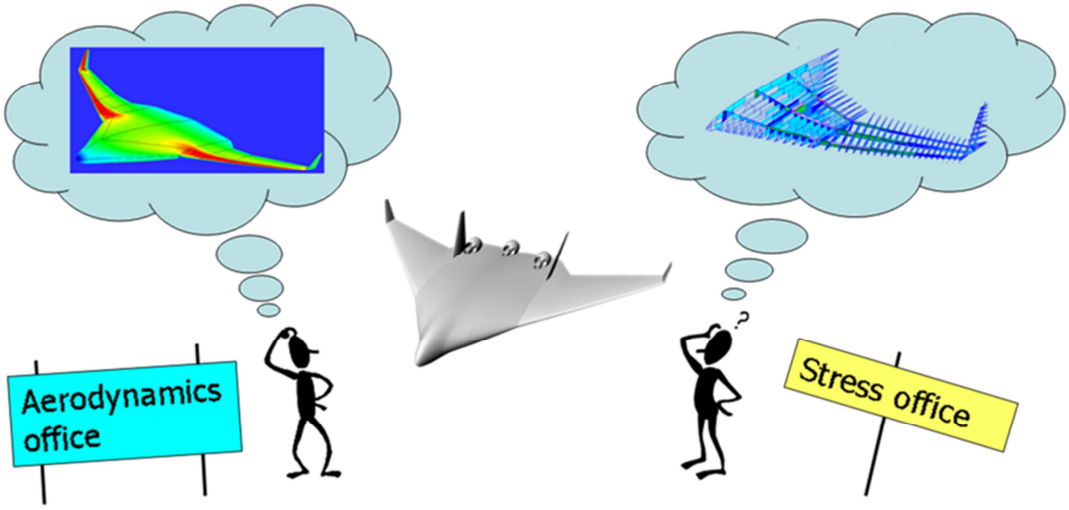


Fig. 2.16 the different views of discipline specialists on the same product

2.7.2 Models consistency and synchronization

To suit the needs of the DEE, the geometry generator must be able to provide the different types of geometric representations required by the various disciplinary analysis tools involved in the MDO process, both low and high fidelity, in-house developed and COTS.

To avoid any inconsistency, the model generator should allow the definition of *one shareable common vehicle description* (the master model), but at the same time should be able to generate a number of *model abstractions* to suit the needs of the various analysis tools. These abstractions reflect somehow the different *views* that various discipline specialists have on the same aircraft Fig. 2.16. Different model abstractions often include different vehicle components and non coincident surfaces. Also they might feature a different dimensional representation of the same component (e.g., the wing skin as a 2D surface or as a solid 3D plate with a given thickness). The generation of these models would typically require some massaging and adaptation of the original geometry, such as the suppression of some features, modifications and additions of elements (e.g., control volume contours for CFD), splitting or grouping of surfaces, etc.

The capability of the model generator could go as far as delivering completely preprocessed models ready for the solver(s) (Fig. 2.17, case b), or directly analytical results (Fig. 2.17, case d). In the latter case, the risk of too tight integration of modeling and analysis functionalities is evident. The model generator should at least take care of the geometry preparation for discretization and then submit a suitable model abstraction to a grid generator, before running the analysis (Fig. 2.17, case a).

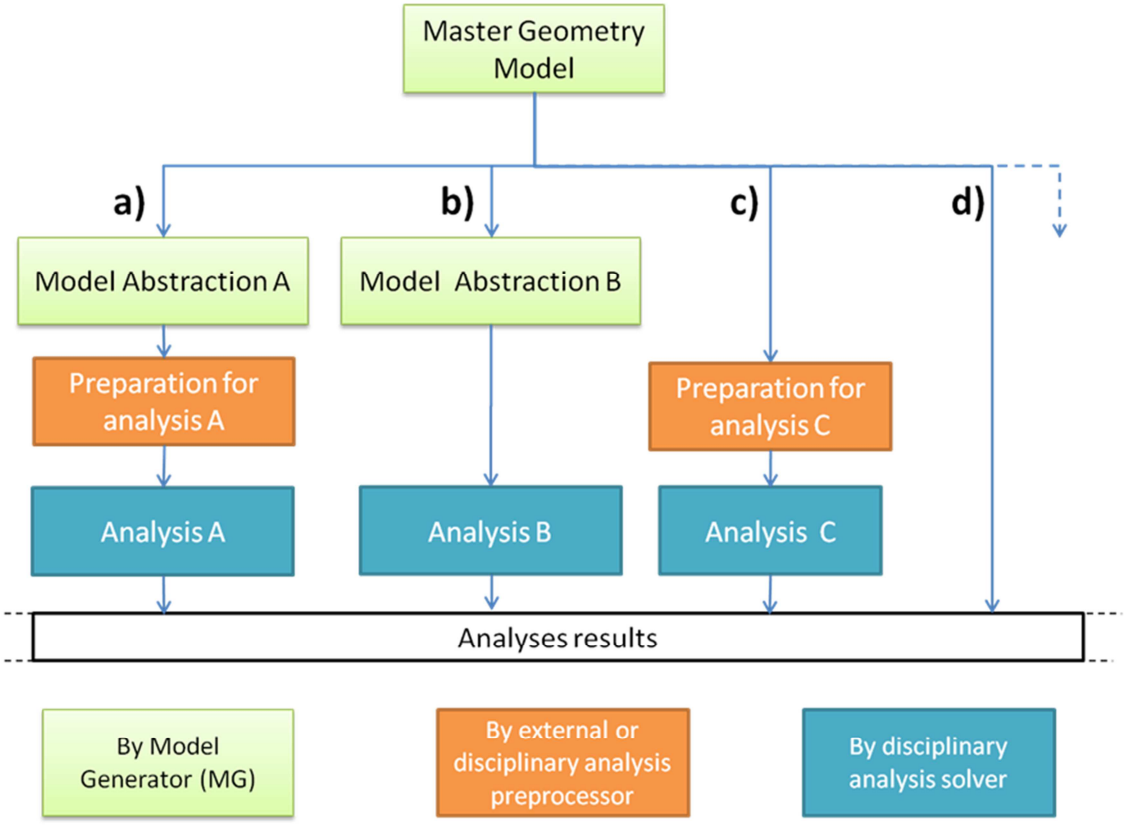


Fig. 2.17: Generation of model abstractions to support multidisciplinary design and optimization:

Case a: MG produces a model abstraction that needs further processing before analysis

Case b: MG produces a model abstraction which is ready for analysis

Case c: MG does not produce any abstraction. The transformation of the master model geometry into a suitable model for analysis has to be performed by the analysis preprocessor (and the specialists)

Case d: MG uses integrated analysis capabilities to generate results directly

In the traditional design approach followed by many organizations, drafting and analysis are carried out by different specialists. Typically, geometric models delivered by draftsmen need a lot of preprocessing to become suitable for analysis (Fig. 2.17, case c), just because draftsmen are not always aware of the diverse and specific needs and preferences of their customers, neither of the problems that may rise during the export of their models to a different platform[2]. A single geometry

[2] As matter of fact, most high fidelity analysis tools provide their own preprocessors to fix and massage the model geometry from the CAD, prior to apply the mesh. For example, Fluent uses Gambit, Nastran uses Patran.

generator can overcome this problem, but needs to be developed in collaboration with the discipline specialists to make sure their requirements on the quality of the output model abstractions are fulfilled.
A robust integration of the tools used for analysis with those used to develop the design (under a geometric standpoint) is indicated by Lockheed Martin specialists as a need to exploit MDO such that is possible to affect real world aircraft. With regard to that, they comment on the peculiar fact that the worlds of Computer Aided Analysis and CAD have developed independently[3], whilst multidisciplinary analysis is not possible without a practical design to start with, and CAD without analysis does not have any scientific foundation (Carty and Davies, 2004).

The generation of input data and/or geometry abstractions for in-house developed tools can present some challenges for the geometry generator. In-house developed tools typically lack flexible and standard interfaces and require a very rigid and specific input data format. This demands embedding some knowledge in the model generator in order to enable the generation of tailored data files.

Whatever the level of fidelity and standardization of the analysis tools, the geometry generator should be able to deal with the *iterative nature* of the MDO approach. Any change in the aircraft shape or configuration (either required by the optimizer, or due to changes in the top level requirements) must lead to the *automatic re-generation of updated data and geometry abstractions* for all the involved disciplines. It is the responsibility of the model generator to *guarantee the synchronization* and harmonization of the analysis process, avoiding disciplines working on obsolete or inconsistent definitions of the aircraft configuration.

2.7.3 Supported usage of High Fidelity analysis

The challenges discussed above just get amplified when the model generator must support the use of high fidelity analysis tools. However the exploitation of High fidelity analysis tools is of paramount importance to the development of novel aircraft configurations and to the credibility of the MDO approach.
In general, the preparation of model for high fidelity analysis such as FE or CFD can be more time consuming than just performing the analysis (since calculation time can often be attacked with computational brute force). It has been estimated that 80% of a FEM analysis cost is spent just preparing the mesh (Chapman and Pinfold, 2001), provided the geometry of the model is free from native corruptions and irresolvable inaccuracies.

---

[3] Interesting developments: In 2005, Dassault Systemes, the company of the CAD system CATIA, acquired Abaqus, the developer of the homonymous popular FE package. In 2003, UGS (since 2007 Siemens PLM Software), the developer of NX (one of the direct competitor of CATIA), purchased a royalty-free license for the FEA software product MSC Nastran, since then evolved as NX Nastran.

NASA specialists claim that grid generation for aerospace vehicles is a number one issue and they challenge the grid generation community to develop tools suitable for automated MDO (Zang and Samareh, 2001). However, though the capabilities of commercial grid generators are increasing, especially with the developments in the field of *associative and unstructured grids*, (as well as of meshless methods) a lot of geometry manipulations are still required, just to have the aircraft model ready to be meshed.
In order to have a model generator able to perform the preprocessing work and/or the generation of grids, it is evident that knowledge will have to be embedded to capture and automate some of the specialists' best practice.
Besides, the level of robustness and flexibility of current high fidelity tools needs to increase. Developers should consider offering more possibilities to interact with the grid generator via a *programmable interface*, rather than the usual "click and select" approach.

2.7.4 Process automation

Considered the strong unbalance (in the order of 20:80) between creative/skillful work and routine/repetitive work (e.g. for preparing model for analysis) typical of the current design approach, improving the *level of automation* should be a primary goal; especially to support a design methodology such as MDO, which is so strongly based on iterations.
Therefore, the model generator (as well as all the other components of the DEE) should be able to work in batch mode. Hence it should be accessible in remote, possibly using web connections, and should be able to operate completely hands off, whereas the use of GUIs should be limited to set up the system or for "off line" interactive work. It appears that the only way to achieve such capabilities is *by writing code*, such that the required knowledge can be recorded directly inside the model generator.
The automatic (re-)generation of the various model abstractions should be robust enough to avoid checking the quality of all output during the iteration of an MDO process (see spatial integration issue addressed above).
In general any kind of manual rework should be avoided. The model generator should give designer the confidence that everything can be easily changed; each design choice can be evaluated and eventually withdrawn. Designers should feel the confidence that suitable models for analysis (especially those for high fidelity analysis) can always be available at demand[4].

[4] The capability enabled by automation to perform more analyses in a shorter time can actually result in a shift of the current MDO systems' bottleneck from generating data to make proper use of these data. J. Staubach, a Pratt & Whitney MDO specialist, speculates on a close future, where automated

2.8 Development of the DEE Multi Model Generator: beyond the capabilities of conventional CAD

In view of developing a Multi Model Generator tool able to function within the DEE framework, *as well as a standalone system*, the designers' needs and the foreseen development challenges have been thoroughly discussed.
Looking at the available technologies, as soon as generation and manipulation of geometry are involved, CAD systems appear to be the prime choice. Indeed, they are excellent tools for detail design and offer a wealth of options for interactive modeling.
However, the MMG must enable designers to model all the aircraft configurations of interest in a fast and effective way, in order to proceed as swiftly as possible with the analysis of their performance. During the conceptual and preliminary phases, configurations can change so fast and so radically, that building models by means of the low level primitives of a conventional CAD systems (points, lines, curves, solids etc.) can be slow and cumbersome.
Besides, designers think in terms of *functions*, not of points, splines, etc. Especially during the conceptual phase, designers are busy considering and rearranging possible solutions to fulfill a number of basic functionalities, such as storing payload, generating lift, provide control, etc. For this purpose, the availability of some kind of *high level design objects* (rather than the low level CAD primitives) would accelerate the transition of concepts and insights from the designer's head to a (geometrical) model to start the verification process. Ideally, such *high level design objects* should be "smart enough" to guarantee spatial integration, while the designer is experimenting with large parameters variations.

> *CAD systems relate to the physical description of a concept; they are not suited for transforming customer requirements to abstract functional descriptions and then to a physical description. The process of concept creation that occurs in the earliest conceptual design stage is quite often vague, and not well understood. How an engineer generates good design concepts remains a mystery that researchers from engineering, computer and cognitive sciences are working together to unravel.*
>
> P. Raj – Lockheed Martin Aeronautical Systems (Raj, 1998)

---

analysis and optimization capabilities, supported by the enormous available computational power, will enable the concept of *zero design time*. Machines will keep computing continuously different solutions in the *whole* design space, while engineers will just need to update the parametric CAD models and the constraints limit of the physics models to account for eventual technological improvements (e.g. availability of new materials). When the customer's need and the company's financial situation align, the design will be ready and waiting to go (Staubach, 2003).

To enable the automatic transformation of the aircraft model in the various disciplinary abstractions, the Multi Model Generator would need some embedded knowledge, as discussed in the previous section. Ideally, the abovementioned *higher level design objects* should "know" how to transform themselves to facilitate the multidisciplinary analysis and optimization process. Again, the low level primitives of a conventional CAD system could not help because of their inadequate knowledge recording capabilities.

In the light of these considerations, a new aircraft modeling methodology has been developed during this research work, based on the object oriented modeling approach. Then, Knowledge Based Engineering (KBE) has been selected as the most suitable technology to implement such modeling approach in the development of the Multi Model Generator.

In order to introduce the background technology and the modeling approach implemented in the Multi Model Generator, Chapter 3 will cover the fundamental characteristics of Knowledge Based Engineering and the basic elements of the Object Oriented paradigm. Finally, Chapters 4-6 will cover the conceptual development of the MMG and its implementation into software.

2.9 A brief discussion on non-technical barriers to MDO

The increasing amount of dedicated publications and international conferences, the explicit interest on MDO-oriented proposals by the European Commission, the growing trend within universities to incorporate MDO courses in their curricula, they are all evidence of the great interest of the international community on this design methodology and the efforts to create awareness and momentum.

Nevertheless it should be acknowledged that what is hampering the explosion of the MDO approach are not only technology limitations, but also *non-technical* barriers. These are rising from the culture and the working attitude of professionals, both at technical and management level (Blouin, Summers and Fadel, 2004; Malone, 2002), as well as from organizational structures intrinsically inadequate to the MDO needs. Belie claims non-technical barriers must be addressed with the same systematic approach, as used for the technical ones (Belie, 2002).

One of the problems is associated to the *black box perception* that engineers might get of an MDO system, hence the fear to be not in control, or even obsoleted by a software system. Ironically, current more manually oriented processes have many of the same attributes as the big MDO black box, but in this case familiarity has bred unfounded confidence (Belie, 2002). MDO will not replace the design team, on the contrary, the team interaction with the process is absolutely necessary to learn about the design, assess the ground rules, add/replace constraints, furnish guidance in area not modeled and keep the optimization on track (Wakayama and Kroo, 1998).

In addition, as Belie correctly points out, many disciplines have been constrained by the tedious tasks associated with their expertise for so long that the automation of "no brainers" (like meshing) seems somehow undesirable (Belie, 2002).

> *Most organizations can't afford to run a program inefficiently; however, they can't find the time to introduce efficiency techniques.*
>
> B. Malone - Phoenix Integration (Malone, 2002)

In the actual contingency of budget cuts and cost avoidance, management may look at a MDO framework just as an irrelevant and large expense, underestimating or completely ignoring the advantages in the mid/long term. This reluctance needs to be addressed by providing management with an adequate metric and tools to estimate and measure MDO costs and benefits. Also a change from a product to process oriented culture seems necessary.

Important changes must happen at the organization level of the aircraft company, which has evolved through the years towards an aggregation of discipline focused groups. Typically, new approaches and tools like those for computational advance are assimilated best if they automate *traditional* tasks and do not cross organizational boundaries. "Unfortunately", MDO is, by its very nature, cross disciplinary and non-traditional (Giesing and Barthelemy, 1998).
Ensuring buy-in of the disciplinary experts to the MDO approach may be difficult (Bennett et al., 1998), but it is necessary because the whole approach is based on the collaboration and integration of the various discipline, which requires experts' commitment to share and translate their knowledge in procedures and explicit rules to be implemented in the MDO framework and modules. Eventually, setting up a design framework as the DEE discussed in the previous sections, would not only enable automation, but would contribute to a better process understanding, control and standardization within the company.

In the current organization, MDO does not have a real home thus no ownership (Hoenlinger, Krammer and Stettner, 1998). However, the set up of a dedicated group would be already a wrong approach to integration. An organizational balance has to be found in order to leave the disciplinary groups in charge of the excellence and the continuous development of their tools, while, the current conceptual design group could assume MDO responsibility and play an integrating function, rather than keep on using and developing simple tools (just to avoid and limit the coordination with the various disciplines)

CHAPTER 3
Knowledge Based Engineering. The AI roots and the OO paradigm

1. Introduction
2. What is Knowledge Based Engineering?
3. The AI roots of Knowledge Based Engineering
4. Knowledge Based Systems + Engineering = Knowledge Based Engineering Systems
5. KBE systems and KBE applications. The programming approach
6. KBE languages: A survey of main characteristics
7. The extra gear of KBE languages: Runtime caching and dependency tracking
8. The rules of Knowledge Based Engineering
9. KBE product models to capture the What, the How…and the Why of design
10. On the convenience of the programming approach
11. Summary 1: How KBE systems differ from conventional KBSs
12. Summary 2: How KBE differs from CAD

3.1 Introduction

In the Chapter 2, the keystone role of the Multi Model Generator to support the MDO approach has been discussed. A number of challenges has been highlighted that traditional design tools, as conventional CAD systems, do not seem able to meet. The need to substantially increase the level of automation in the design process calls for the ability to embed more knowledge in the employed tools. In this case, the use of Knowledge Based Engineering technology appears to be an appropriate choice.

KBE technology stands at the cross point of diverse fundamental disciplines, such as Artificial Intelligence (AI), Computer Aided Design (CAD) and computer programming. Though these single contributing disciplines are widely represented in scientific literature, Knowledge Based Engineering is not at all. At date, not a single book has been written on this topic! As a matter of fact, KBE has been for many years restricted domain of a few and highly competitive industries (aerospace and automotive in particular) and never turned into a subject of academic research. The very limited amount of available information, mainly in form of pamphlets from KBE

vendors, has not stimulated the interest of the scientific community on KBE as a real engineering discipline. On the contrary, it has contributed generating the mixture of misunderstanding and skepticism that has marked the difficult story of KBE at date.

The main purpose of this chapter is to provide a clear understanding of this technology (i.e., *what Knowledge Based Engineering is*) on the base of a few years of hands-on experience, literature study and valuable conversations with KBE-practitioners from the industry.

First a comprehensive definition of KBE is provided. Then, roots and background of KBE are discussed in order to put it in the right context with respect to closely related technologies such as Knowledge Based Systems and CAD, and the object oriented modeling paradigm. Also we try to answer the recurrent question *why "knowledge based"* engineering? Is there some other way on earth of doing engineering that is not based on knowledge!

Then, the most relevant characteristics of a KBE system are identified and elaborated. It is discussed how a KBE system works and how it is structured. In particular, the use and the nature of KBE programming languages will be elaborated on. The actual opportunity of using a programming language to capture design knowledge will be discussed, as well as the required characteristics for this "wish-language". Though KBE technology inherits many features and characteristics both from traditional CAD and Knowledge Based Systems, it clearly stands out as a technology with its own identity. The chapter will conclude summarizing the main differences between KBE and the abovementioned technologies.

In the next chapters, it will be explained how Knowledge Based Engineering and the underlying object oriented modeling approach have been used in this research work for the conceptual development and technical implementation of the Multi Model Generator tool.

3.2 What is Knowledge Based Engineering?

It is rather difficult to fit an exhaustive definition of KBE in one sentence. Various definitions can be found in literature and typically reflect the different views on KBE by different "KBE customers". For example, a company manager sees KBE as a technology to compress product development time and cut engineering costs, whereas the user of a KBE system (a KBE developer) sees it as a particular kind of software (programming) tool. Someone in MDO might see KBE as a solution to support multidisciplinary design, someone into Knowledge Management as a solution to capture company knowledge for effective reuse.

KBE is all of this and lead to our extended definition shown in the insert below. This definition includes the most relevant aspects of KBE and reflects the various constituents of this thesis work. It formalizes the link with the Knowledge Management area addressed in Chapter 1. It emphasizes the capability to automate

the repetitive activities typically encountered in the product development process, hence to enhance engineers' productivity as argued in chapter 2. Besides it addresses the capability of supporting *conceptual and multidisciplinary* design. The latter in particular is our extension, or specialization to the KBE definition of the "old fathers" and it will be extensively discussed in the second part of this thesis work.

Knowledge based engineering (KBE) is a *technology* based on *dedicated software tools* called KBE systems, that are able to *capture and reuse product and process* engineering knowledge. The main objective of KBE is the reduction of time and costs of product development by means of the following:

- *Automation of repetitive, non creative, design tasks*
- Support of *multidisciplinary design optimization* in *all the phases* of the design process

3.3 The AI roots of Knowledge Based Engineering

It would not be fair presenting KBE as a novel revolutionary product from the world of computer science. The strong legacy from the 1970s technology of *knowledge based systems* (KBSs) must be acknowledged. Undeniably, many characteristics of current KBE systems, including much of the related terminology, are rooted in the field of Artificial Intelligence (AI).

AI is the branch of computer science concerned with the use of machines to simulate the intelligent behavior of human beings (i.e. the capability of learning and solving problems) and KBSs are one of its most relevant outcomes.

Just as *Artificial Neural Networks* represent the main result in the simulation of the humans' learning process, knowledge based systems represent the most outstanding AI achievement in the area of problem solving. Both the fundamentals and the added value of KBE systems cannot be discussed without having addressed first the basics of knowledge based systems.

3.3.1 Knowledge Based Systems. Functionality and structure

Knowledge based systems are computer applications that use stored knowledge for solving problems in a *specific domain* (Negnevitsky, 2005; Engelmore and Feigenbaum, 1993).

Similarly to a human expert, a KBS applies some kind of reasoning approach to derive an answer to the posed problem, based on the knowledge existing in its memory. Like a human expert, a KBS is able to justify its decisions and explain *how* it did get to the solution or *why* it needs certain information to carry out the problem solving process. Due to the intent of emulating the behavior of human experts, at

the beginning of the 1970s, KBSs started to be addressed as *Expert Systems* (ESs). Since then, the terms KBS and ES are used synonymously.
The typical structure of a KB system is represented in Fig. 3.1. At glance, five main components can be distinguished, which are addressed below.

The knowledge base (KB) represents the dedicated storage container where the domain expert knowledge to perform certain tasks is recorded. The content of the KB is *independent from the problem at hand*. In other words, the same knowledge can be used by the KBS to solve different specific problems. In a cognitive model of the human expert, the KB would resemble his/her *long term storage* memory.
The knowledge stored in the knowledge base has a *symbolic representation*, which makes it intelligible to humans as well as a computer. For example, knowledge can be stored in the form of *rules*, such as *IF Condition THEN Action*. As will be discussed in the coming two sections, different knowledge representations can be used, which yield to different types of KBSs.

The work space, or ***work area***, (also known as "blackboard") represents the *short term memory* of the KBS. Here the problem to be solved is stored, together with the collected facts and data. These can either be the case specific, intermediate results produced by the rules stored in the knowledgebase, or facts and data provided directly by the user during the various steps of the problem solving process (typically on request of the inference engine).

The inference (or reasoning) engine consists of one or more programmed *reasoning mechanisms*, which allow the KBS to reason upon the stored knowledge and solve a posed problem with the competence of a domain expert (Engelmore and Feigenbaum, 1993). What the inference engine actually does is trying to match all the rules stored in the knowledge base with the facts contained and continuously added to the workspace, until a solution is found. Thereby, it uses the knowledge base to alter the contents of the work space (Milton, 2008). The systematic approach used for selecting and matching the various rules (i.e., the reasoning mechanism) can follow different schemes, like the *forward-chaining* and/or the *backward-chaining* mechanism discussed in next section.
The inference mechanism may not be able to derive a solution in case of missing data or lacking knowledge. However, when a solution is found, it is always consistent with the rules of KB and, as such, it can always be explained.

The Explanation subsystem is the KBS subsystem dedicated to provide the user with the explanation of the found solution (or proposed advice). The explanation subsystem traces all the rules fired during the problem solving session and provides the user with the firing sequence, together with the facts used/obtained step by step during the process. Hence a KBS can explain *how* a certain solution was derived and, in case, *why* additional data were needed to arrive at a solution.

The Users' interfaces. As shown in Fig. 3.1, it is possible to distinguish three different actors interacting with the KBS, namely the domain expert, the software engineer and the end user. In certain cases, the same person can play different roles. However, in general, three different interfaces are required to facilitate the three actors carrying out their specific competences. The domain expert (or the knowledge engineer) needs to update and maintain the KB. The software engineer is responsible for programming and debugging the inference system. Eventually, the end user employs the KBS as a commodity tool. In this case, a good User Interface UI can make a critical difference in the perceived utility of the overall KB system, regardless of the system's performances (Engelmore and Feigenbaum, 1993).

One of the most relevant characteristics of KBS, which already emerges from the structure described above, is the *crisp separation between the knowledge and its processing*. This characteristic has favored the development and commercialization of a number of of-the-shelf tools that facilitate the set up of a KB system. Some are described in (Negnevitsky, 2005), pp. 391-406. These tools, called *shells* or *skeletons*, provide the basic components of a KB system, such as a repository to store and organize the knowledge and one (or more) inference mechanisms. As

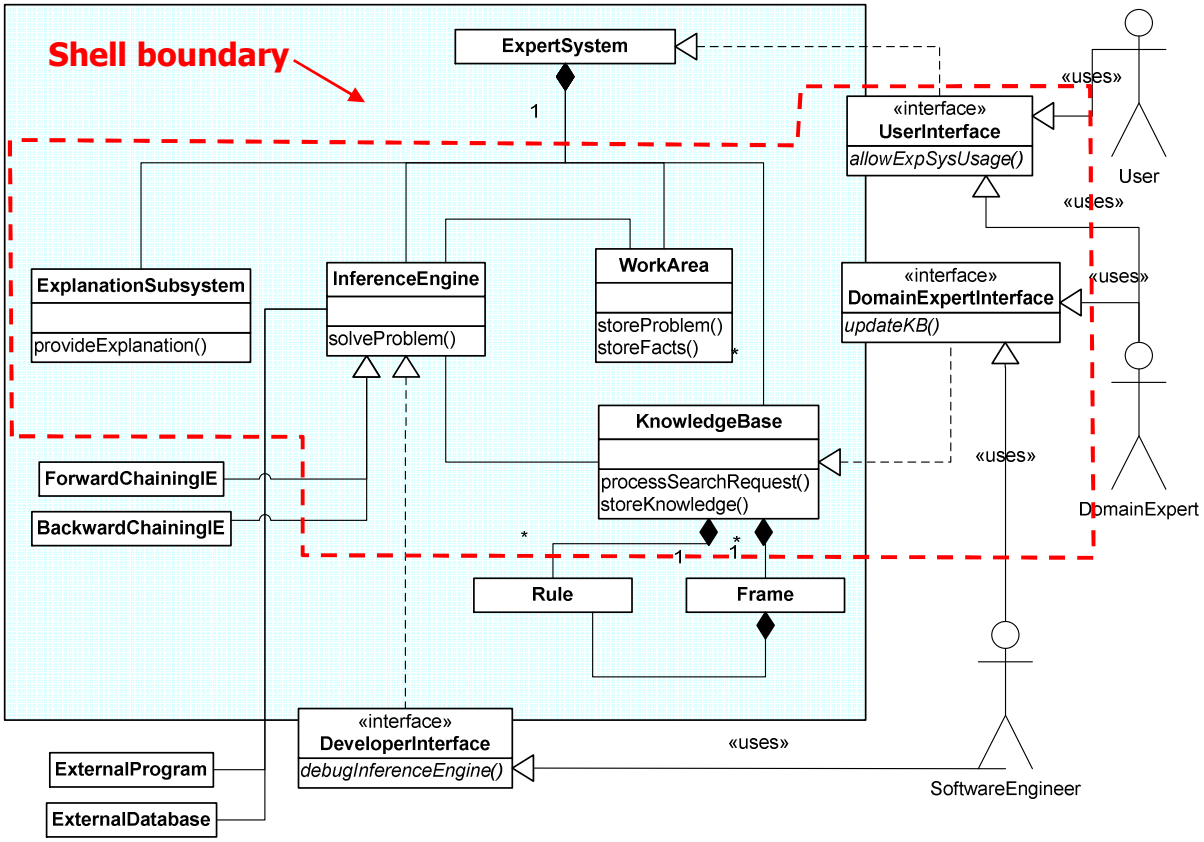


Fig. 3.1: modular structure of a knowledge base system and definition of a shell.

indicated by the dashed contour line in Fig. 3.1, a shell is actually a KBS with an empty knowledge base, but provided with the required interfaces to fill the knowledge base and operate the system. A shell reduces or completely eliminates the need of any programming activities for building a KB system. However, shells do not provide any support concerning the knowledge *acquisition phase*, which remains one of the most critical activity in the whole development process of a KBS (Milton, 2007; Shreiber et al., 2000).

3.3.2 Rule based Expert Systems. Production rules and inference mechanisms

In order to be stored in the KB and be accessible to the inference engine, knowledge has to be structured and formalized by means of some symbolic representation. In the field of knowledge based systems, *rules* and *frames* are the two most common means for representing knowledge, the first being the well known IF-THEN construct, and the second a description of a given entity (or *object*) by a list of *attributes* and associated values.

Expert systems where the whole domain knowledge is codified in the form of IF-THEN rules are called *rule based expert systems*, while those systems using only frames are called *frame based expert systems*. The former represent the most common typology of expert systems and are discussed below; the latter will be addressed in the next section.

Rules are actually statements built up of at least two main components: an if-part called *antecedent* (or *premise*) and a then-part, called *consequent* (or *conclusion*). In general, both the antecedent and consequent can be composed of multiple *facts* (or *conditions*) and *actions* (or *conclusions*), respectively, linked by logical operators such as AND, OR. Each fact and action is built of a *linguistic object* and *value*, linked by an *operator*. See the example in Fig. 3.2.

When the antecedent of the rule is true (when both *Fact A* and *Fact B* are true in our example) the rule fires and *produces* – hence the name *production rule* – the results

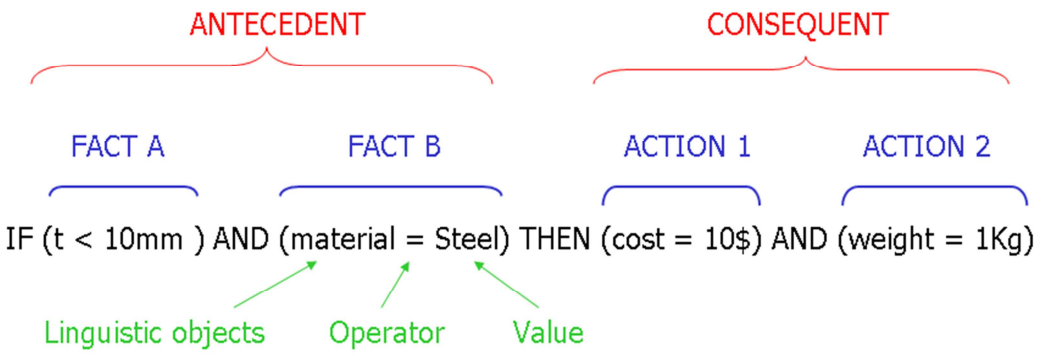


Fig. 3.2: Structure of a production rule

indicated in the consequent part (in the example, *Action 1* and *Action 2)*.
Note that, in order for this rule to fire, the antecedent (i.e., the facts *t < 10 mm* and *material = Steel*) must be available in the KBS work space, either provided by the user or generated by some other rule that fired first. The result of a rule firing will be the generation of new facts that will be stored in the work space. When the rule of our example fires, the two facts *cost = 10$* and *weight = 1 Kg* will be added to the work space.

Once the knowledge base has been populated with rules, the inference engine can put them in use for solving a problem. There are two fundamental ways an inference engine operates, namely by means of the *forward-chaining* or *backward chaining* mechanisms (or a combination of the two). These two approaches, as it will be discussed in the next two subsections, reflect two different approaches typically used by human experts for solving problems.

Forward chaining inference technique

This inference technique is also called *data driven* or *eager* approach. On the basis of the data initially available in the KBS work space, the inference engine will search and fire all the *fireable rules* in the KB (hence all the rules whose antecedent match with the available data in work space). When fired, these rules will either modify or add new facts to the work space. The match and fire process proceeds in cycles, where each cycle ends when all the rules in the knowledge base have been scanned for a match with the current facts in the work space. The whole inference process stops when no further rules can be fired. Two basic conditions apply:

1. During each cycle, the inference mechanism examines the rules *sequentially*, starting from the topmost rule stored in the knowledge base
2. Each rule is fired only once during the whole problem solving process

Fig. 3.3 shows an example of forward-chaining inference, where *A*, *B*, *C*, *D* and E are the facts initially available in the work area. At the end of the inference process, the facts *Z* and *L* are discovered. In order to find *Z*, *X* and *Y* were also computed and added to the work space. By using the forward chaining approach, *all* the fireable rules fire, no matter if relevant to the problem at hand. If, for example, the goal of using the system was the evaluation *L*, then the generation of fact *X* and all the operations occurring in cycle 2 and 3 result in a waste of time and computational resources. If we consider that a real rule based system generally contains thousands of rules, then the use of the forward-chaining mechanism to infer only one particular fact might result very inefficient.

On the other hand, this approach is appropriate for expert systems that must perform analysis and interpretation work, hence infer *all possible facts* based on the available knowledge and a few initial facts. In fact this approach reflects the typical behaviour of an expert who is requested to evaluate a complete scenario depending on a given set of facts.

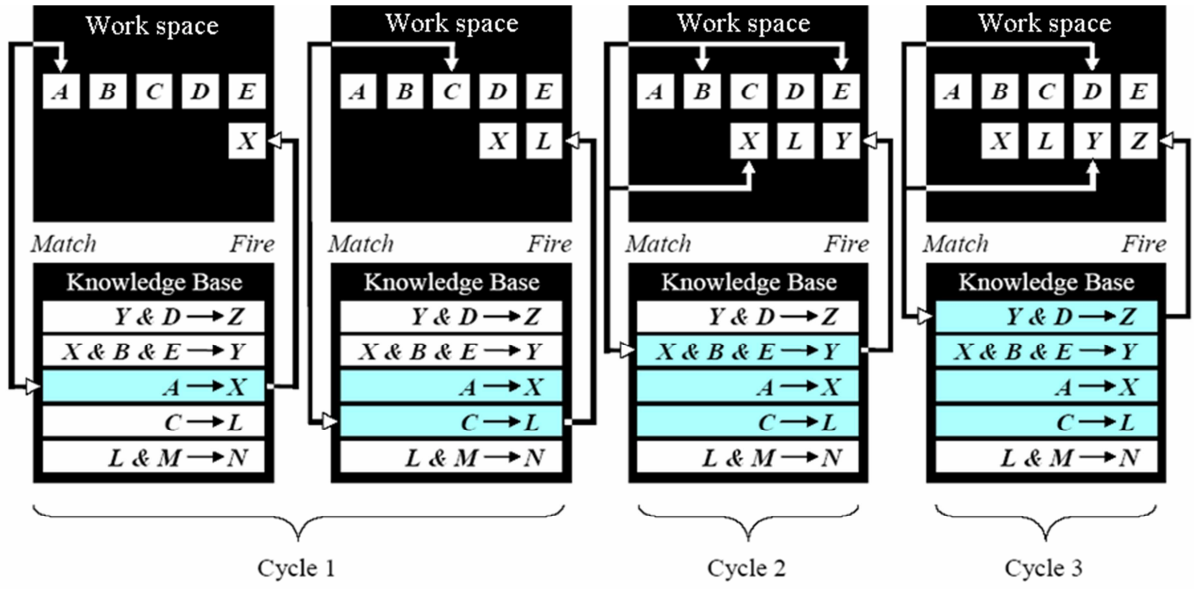


Fig. 3.3: Example of forward chaining (or data driven) inference mechanism (Negnevitsky, 2005).

Backward chaining inference technique

In case only one specific solution must be evaluated, backward chaining is the most appropriate mechanism. This inference technique is also called *goal-driven* (or *lazy*) approach. In fact, the reasoning process starts by assuming a hypothetical solution (the *goal*) and only the rules that are useful to prove the solution are used.

The inference process goes as follow. First, a rule is searched which contains the hypothetical solution in its THEN-part. The data (facts) in the work space are used to check if the IF-part of the selected rule is true (hence if the given rule can fire). In case the data currently available in the work space are not sufficient for evaluating the conditional part of that rule, the rule is *stacked* and the facts contained in its IF-part become the new (sub-)goals. Hence, other rules are searched in the knowledge base that can produce those sub-goals. In case no facts can be generated to prove one of the sub-goals, the inference system *asks the user directly* to provide those facts, which is a substantial difference with respect to the forward chaining approach discussed before. In case also the user is not able to provide those facts, the inference mechanism abandons the initially selected goal (the hypothetical solution) and starts again searching the rule base for another rule with a possible solution for the problem at hand contained in its THEN-part. In other words, a new goal is issued

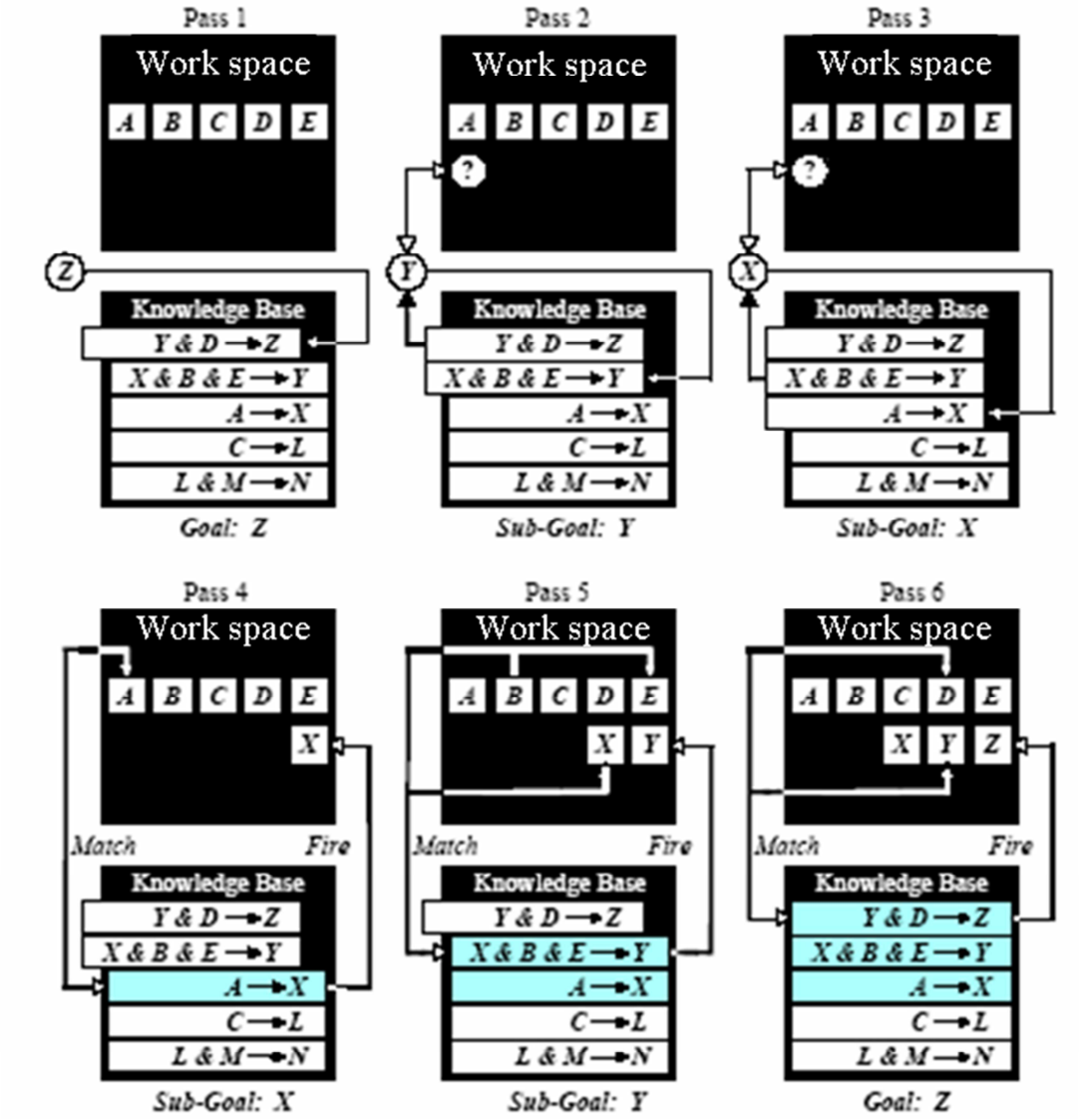


Fig. 3.4: Example of backward chaining (or goal-driven) inference mechanism (Negnevitsky, 2005)

and the process is continued until the system either finds a solution *that can be demonstrated*, or raises the white flag.

Fig. 3.4 shows a backward chaining process example. In this case *Z* is the first hypothetical solution assumed by the inference engine, for which the conditions *Y* and *D* must be true. While *D* is a known fact in the work space, *Y* is not; hence *Y* becomes the new sub-goal, the first rule is stacked and another rule to produce *Y* is searched in the KB. At Pass 4, the third rule in the list fires, allowing also the other stacked rules to fire in the next passes. Eventually, all the facts have been generated to prove *Z*. If fact *A* would have not been available in the work space, the inference engine would have asked it directly to the user. In case of no answer, the initially assumed solution *Z* would have been abandoned and a new goal selected.

Note how both the rule base and the initially available facts in this example are the same as for the forward chaining example of Fig. 3.3. However, only three rules have been used in this case, which were all indispensable to prove the hypothetical solution *Z*. No resources were wasted to evaluate facts (like *L*) not needed to support the direct line of reasoning.

The backward-chaining approach simulates the typical behaviour of a domain expert who first thinks of a possible solution to the problem at hand and then finds data that, either confirm his/her hypothesis, or suggest a different answer. Typically, the backward chaining reasoning approach is implemented in *diagnostic* purpose expert systems, where for example a disease or a system failure can be assessed on the base of symptoms (and by consequence a cure or some other corrective actions can be suggested).

3.3.3 Frame based Expert Systems. The object oriented paradigm applied to knowledge based systems.

While rule-base expert systems require domain knowledge to be *completely expressed* in terms of IF-THEN rules, frame-based expert systems provide a more advanced representation of knowledge by means of *frames*. A frame is a simple but effective solution, proposed in 1970 by Minsky, to store relevant knowledge relative to a certain object, into a single data structure (Minsky, 1975).
In Fig. 3.5 an example of a frame relative to the Fokker 100 aircraft is shown. Each frame has its own name (in this case, Fokker 100) and a list of attributes, also called *slots*, which describe the given object (e.g., number of passengers and range). Each slot can have a value associated to it (in this case, 107 and 2500 Km).

Fokker 100	
MTOW	43000 Kg
No of passengers	107
Wing span	28 m
Total length	35.5 m
Wing area	93.5 m^2
Range	2500 Km

Fig. 3.5: Example of frame, a widely used mean of knowledge representation.

In the intention of Minsky, frames were means for representing generic concepts and stereotyped situations (Negnevitsky, 2005; Auty, 1988; Lassila, 1990). By assigning values to the frame slots, it is possible to create *specific instantiations* of the generic concept in the frame. For example, the Fokker 100 frame of Fig. 3.5 can be considered a specific case for a more generic Aircraft frame. To distinguish between generic and specific frames Minsky introduced the terms of class-frame and instance-frame (in short *class* and *instance*), where the latter is an instantiation (a specification) of the former generated by assigning specific values to the frame slots. Any different set of values yields

a different instance frame of the same class frame.
For those familiar with the Object Oriented programming paradigm, the distinction between the OO concepts of class and objects and those of class and instance-frames proposed by Minsky might result extremely hazy. As a matter of fact, frame-based systems are often seen as the application of Object-Oriented paradigm in the field of expert systems (Negnevitsky, 2005; Auty, 1988; Lassila, 1990). For what concerns the scope of this thesis, we will not make a distinction between the concept of frames and classes[1]. Besides we will take advantage of this introduction on frame based systems to discuss in the next subsections, two of the conceptual pillars of the Object Oriented paradigm, namely *abstraction* and *inheritance*.

Abstraction
According to the object oriented approach to the representation (modeling) of knowledge, every entity (or *object*) is a unique instantiation of a generic class. Besides, every class can be a specialization of an even more generic class.
As exemplified in the UML diagram of Fig. 3.6, the objects Fokker100 and Fiat500 are two possible instances of the Aircraft and Car classes, respectively. The process of instantiation happens in the very moment that a set of values is assigned to all the attributes contained in the classes' slots. Also, the Aircraft and Car classes of the example are both *specializations* of the more generic class called *MeansOfTransportation*. This is said to be a *superclass* (hence a generalization) of the more specific concepts of Car, Train and Aircraft.
When defining objects, classes and superclasses, what we are actually doing is applying a process of *abstraction*, which helps us structuring given domain knowledge in a way that is efficient and suitable to focus on the problem at hand. When abstracting we are actually isolating those aspects that are relevant to the problem under consideration and suppress unimportant aspects (Rumbaugh et al., 1991). If we are looking at the concept of means of transportation, as in the example of Fig. 3.6, we might not be interested in a more detailed definition of the aircraft concept than the one provided there. But if we are interested in the very concept of aircraft, a more refined description of such concept will be useful. Our level of abstraction changes and some details ignored before, become relevant. For example, it might be opportune to generate a hierarchy of Aircraft subclasses. Possibly the Fokker 100 aircraft will become an instantiation of the *SingleAisleAircraft* superclass,

[1] Discussions can be found in literature about differences between FBS and actual OO programming. Most authors agree the main differences exist in the background and the scope of these systems rather than in the technical aspects. FBSs have a cognitive/psychological background and are mainly aimed at building *knowledge representation* systems, while the OO paradigm comes from the IT programming field and it is mainly aimed at *data processing*.

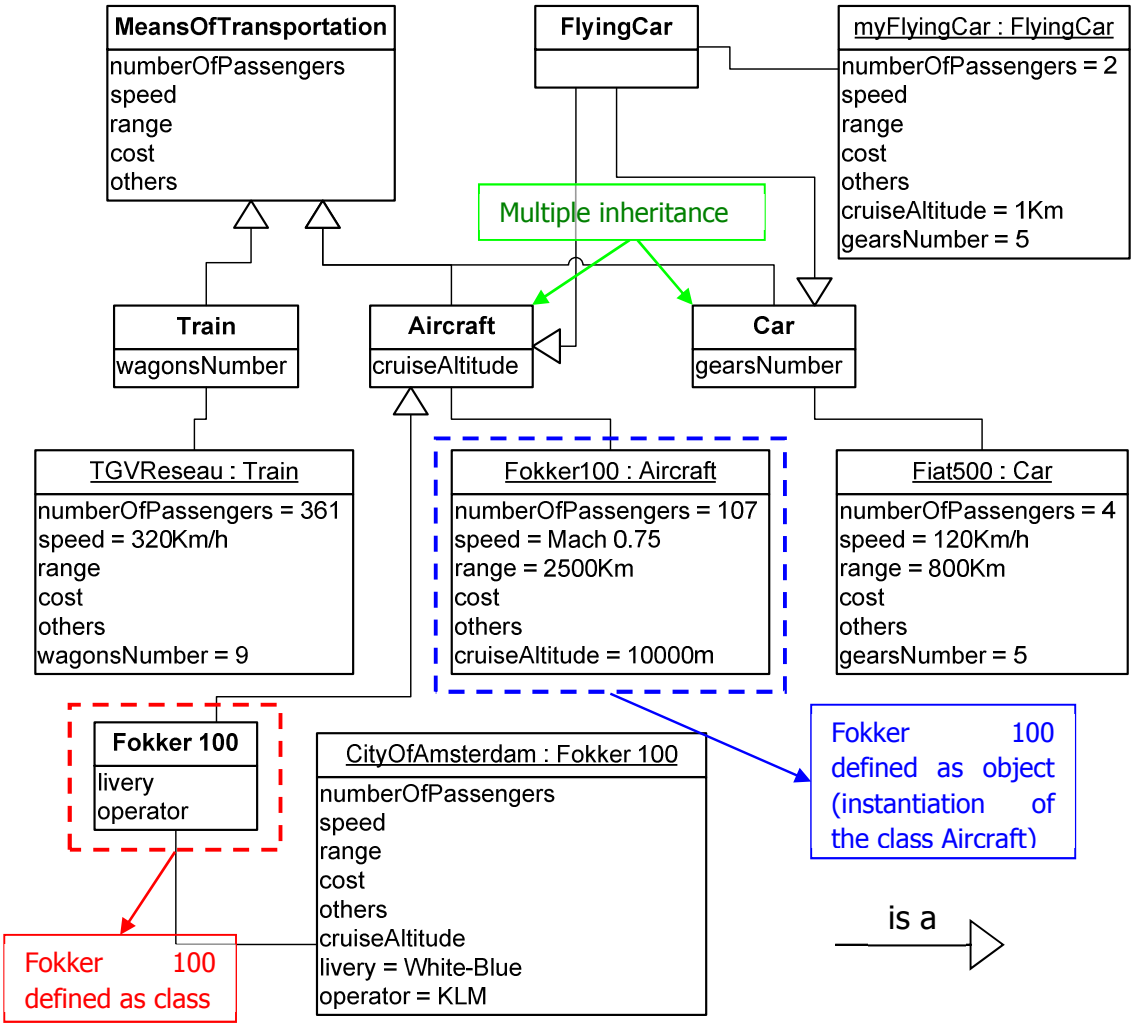


Fig. 3.6: UML diagram representing classes, superclasses and classes instantiation (objects) and relative inheritance relationships.

which in turn will be a subclass of the *JetEngineAircraft* superclass, in turn a specialization of the *ConventionalPassengersAircraft* superclass.
At a different level of abstraction, it might be appropriate to define the Fokker 100 aircraft as a class rather than an instance. For example, if we are looking at the fleets operated by various airlines companies, the hypothetical Fokker 100 named "City of Amsterdam", owned by KLM-Cityhopper, will be a possible instance of the Fokker 100 class.

Inheritance

Classes and objects are aware of their interrelationships. An object "knows" its class as well as a class knows its superclass(es). This information is stored in what we can call an implicit slot. The type of relationship between a superclass and its classes, like superclass *MeansOfTransportation* and the classes Aircraft, Car and Train of the

example in Fig. 3.6, is called either *specialization* or *generalization* according to the direction of the semantic. For example, *Aircraft* is a specialization of *MeansOfTransportation*, which is a generalization of *Aircraft*. Note how in the UML this link is represented by means of empty arrow connectors. The interesting characteristic of this relationship is that each class *inherits* all the slots from the relative superclass, including eventual default values. So all the attributes used to define a superclass automatically cascade down the hierarchy of classes, without the user having to specify them again and again. As showed in the example of Fig. 3.6, since the slot *number of passengers* has been specified in the *MeansOfTransportation* superclass, *Aircraft*, *Train* and *Auto* automatically inherit that slot. So do their eventual subclasses. Typically, new slots are added at each subclass level, as the level of specialization is supposed to increase down the hierarchy. In the example above, the subclass aircraft has added the slot *cruise altitude* and the train subclass the slot *number of wagons*. Furthermore, certain inherited slots can be deleted and some inherited values redefined. An eventual *UAV* subclass of the *Aircraft* superclass, would exclude the *number of passengers* slot.

Most of the frame based systems (and Object oriented programming languages) support also the concept of *multiple inheritance*; hence a class can inherit from more than one superclass. In the example of Fig. 3.6, the class *FlyingCar* inherits from both the *Aircraft* and *Car* superclasses.

3.3.4 Aggregation and association links

Besides the specification/generalization link discussed above, which is the base for specifying taxonomies, i.e., hierarchies of concepts based on the relationship *is-a*, frame based systems allow also the representation of *part-whole* systems, i.e. hierarchies of concepts based on the relationship *has-part*. A frame slot, apart from attribute values, can actually contain pointers to other frames to specify their belonging to an *aggregation*. For example, the class frame *Aircraft* could have a slot called *components*, pointing at some other separately defined classes such as *Wing*, *Fuselage* and *Tail*. This is the so called *aggregation link*, or *has-part* link, which is indicated in the UML by means of diamond connectors (see diagram in Fig. 3.7)[2]. Note that, for simplification, the slot with pointers to other frames is not shown in Fig. 3.7, whereas connectors show the links.

[2] A stronger version of the *aggregation* link exists, which is called *composition* link (indicated in the UML by a filled diamond connector, whereas non-filled diamond are used for aggregation). The components of an aggregation exist also outside the aggregation, while the components of a composition exist only within that composition (e.g., a hole might be a component of a Flange class, but, in general, it does not make sense as a standalone component). Eventually, the difference between the two links depends on the level of abstraction used to represent the given domain knowledge.

Apart from *is-a* and *has-part* links for specialization and aggregation, actually any kind of semantic relationship between classes can be defined by means of slots. For example, in case of the Fokker 100 aircraft, a hypothetical slot *Operator* can point at a number of Airlines frames such as KLM-Cityhopper and Air Berlin, hence representing the *operated-by* relationship. A slot *route* can point to a number of flight route frames (e.g., Amsterdam-London), representing the relation *fly-route*.
Any semantic relationship between classes, which is not of the generalization or aggregation type, falls automatically into the more generic *association* link category. This is indicated in the UML by a simple connector with an explanation tag attached.

In conclusion, a frame based system (and more in general any system based on the object oriented representation) results in a network of *nodes and relations* that provides a structured and concise representation of the given domain knowledge. This represents a major difference with respect to the rule based system described in the previous section, where all the domain knowledge is translated in a flat list of if-then statements.

3.3.5 Inference mechanism in frame based systems. Methods, demons and production rules

The value of FBSs goes beyond the merit of an effective knowledge representation system. FBSs provide also the mechanisms for manipulating and reasoning upon the represented knowledge.
In the previous sections, it is mentioned that a slot can contain a value or a pointer to some other frame. In fact, a slot can also *specify the procedure to compute* its value. This procedure can either consist of a simple production rule (where the slot value is computed according to the evaluation of some preconditions) or even a sequence of commands and operations necessary to compute that value. *Method* is the technical term used in OO parlance to indicate the procedure for computing a slot value.

The capability to include into one structure, both elements of *declarative* and *procedural* knowledge (by means of attributes and methods respectively) is acknowledged to be one of the most relevant features of the object oriented paradigm. It should be noted that methods offer a significant advantage with respect to classic rule based systems, because simple production rules are generally not effective at dealing with procedural knowledge (Negnevitsky, 2005).

Actually, two types of methods are used in FBS: the so called *when-needed* and *when-changed* methods. The former are executed when the value of a given slot is required and not directly available. The latter, often addressed in literature as *demons*, work as dormant processes, which fire in the very moment that certain monitored slot values change. For example, a demon can start a process whenever a certain button is pushed by the user, or can issue a warning when a certain rule is

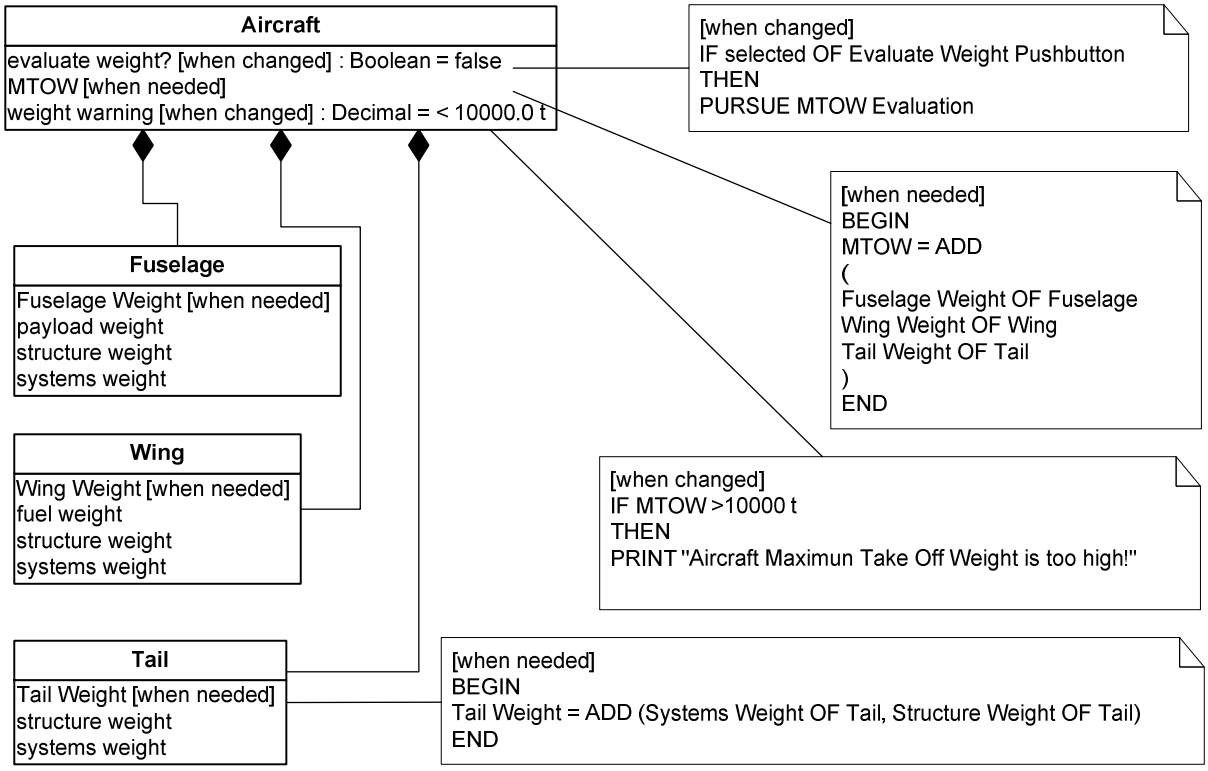


Fig. 3.7: Example of frames with when-needed and when-changed methods. (N.B. the graphical representation of the methods and their syntax are just fabricated for the purpose of exemplification).

violated. Fig. 3.7 shows a simple example of a possible implementation of the two methods. Note that the execution of methods within one frame can also involve the evaluation of attributes from other frames' slots. In this case, the so-called *reference chain* of the attributes must be provided, i.e. the name of the frame where the required slot is contained. In the example, the Aircraft's method MTOW adds the fuselage weight from the frame Fuselage, the wing weight from the frame Wing, etc.

Contrarily to rule based systems, in KBS the inference system is not required to perform exhaustive searches of the workspace and rule base, looking for matching rules. When a slot value is set as a goal, the inference mechanism *directly* executes the method associated to that slot and accesses the slots of other frames in the system, according to a demand driven process. That is to say, the reference chain attached to each attribute informs the inference engine about the specific frame to be accessed for obtaining the needed slot value. The presence of both *when-needed* and *when-changed* methods is such that the inference mechanism can work both with forward and backward chaining.

Even if methods represent the most typical means of knowledge interaction between frames, FBSs do not exclude the possibility of using the typical knowledge representation and problem solving approach of rule base systems (previously

described in section 3.3.2). In fact, lists of production rules can be stored in the knowledge base, next to the collection of frames, and used by the inference system to obtain the value of some frame slots. However, within frame based system, rules always play an auxiliary role while frames remain the main knowledge carriers. The inference mechanism will be informed by a special tag (called *facet*) applied to the given slot when the value has to be computed by using the rule base, rather than querying other frames. Typically, a backward inference mechanism is used here, where the value of the given slot is set as goal for the reasoning process.

To conclude, FBSs add the knowledge structuring power and efficiency of the OO paradigm to the relative simple mechanisms of rule based systems. However, with great power comes great responsibility! Here the user is left with the overhead of deciding both the most suitable way of structuring the knowledge at hand and the mechanism(s) to manipulate that. What are the correct levels of abstractions? How many frames will be required and what is their network of semantic relationships? Are frames sufficient or a separate rule base is also necessary? When to use methods or demons?

In this sense, frame based systems deny the main advantage of rule based systems: a simple and straightforward approach to store and update knowledge. In FBSs is not possible to simply add, modify or delete rules to make the system smarter. The overall hierarchical structure of the knowledge has to be modified. It has to be decided if new classes and attributes have to be added, and if those previously defined need any modification. In case, it should also be considered whether the relationships between classes must be adapted. Eventually, the crisp separation between knowledge and inference engine typical of rule based system, starts to get blurry. Indeed, methods are pieces of programs where knowledge and execution control are closely intertwined.

3.4 Knowledge Based Systems + Engineering = Knowledge Based Engineering Systems

Since the beginning of the 1970s, knowledge based systems started penetrating the market of software applications, addressing problems of various complexity from different knowledge domains. MYCIN (Shortliffe, 1976), DENDRAL (Feighenbaum, Buchanan and Lederberg, 1971) and PROSPECTOR (Duda, Gaschnig and Hart, 1979) are three examples of successful knowledge based systems developed for the diagnosis of blood infections, analysis of chemicals and advice on mineral exploration, respectively. These were outstanding systems in the success story of KBS, but many others just entered the range of daily commodity tools, such as "help on line" and planning/scheduling systems (Engelmore and Feigenbaum, 1993; Milton, 2008).

However, KBSs did not really have an impact on the field of *engineering design*, including here aerospace, automotive and all those areas generally concerned with the development of complex hardware products. Apart from the inherent challenge of translating design knowledge into formal rules, the main reason for KBSs limited success is their inability to deal with two essential activities of the engineering design process, i.e., *geometry manipulation* and *data processing*. As discussed in the previous section, KBSs are tools developed to solve problems by reasoning about facts and not really to perform computations to derive facts or some other complex data processing task (apart from the limited data processing capabilities discussed for FBSs). Besides, KBSs in general do not have any competence in dealing with geometry and any related shape configuration activity.

Most of the engineering work *requires* and *produces* output that involves *geometry manipulation*, deals with the generation and management of *complex products configurations*, *delivers data* to various kinds of *discipline analysis tools* and *depends on* the results of these analyses to advance the design process. An aerodynamicist, for example, will need the results of an aerodynamic computation to decide on the shape of the wing at hand. But, before that, he/she will need an adequate geometric model of the wing to feed the selected aerodynamic analysis tool. Indeed, the generation and manipulation of geometric models take a relevant part of the engineering design process. As a matter of fact, specialized tools for geometry manipulation, data processing and computation in general proliferate in the engineering world. Those are the well known computer aided design (CAD) systems and computer aided analysis (CAA) tools, such as FEA and CFD tools.

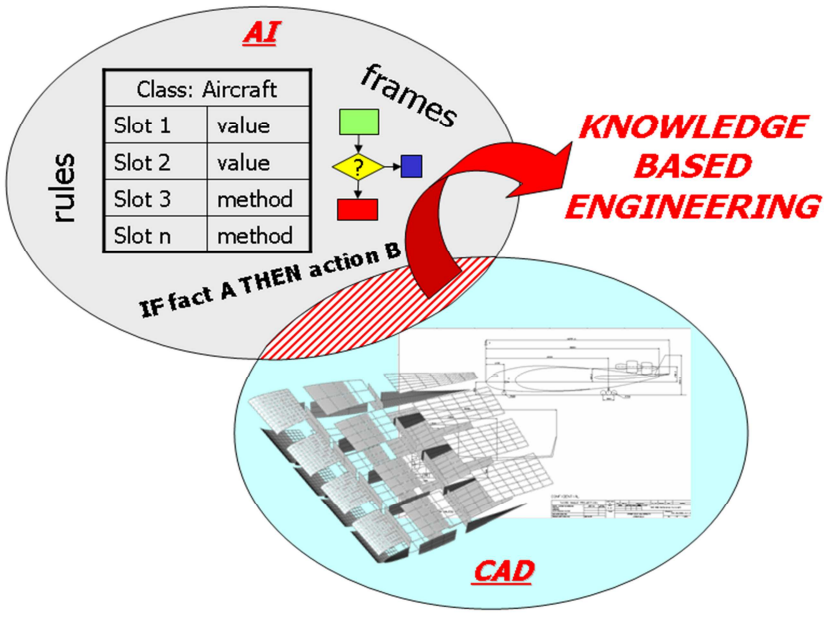


Fig. 3.8: KBE systems: computer programs containing knowledge and reasoning mechanisms plus geometry handling capabilities to provide engineering design solutions.

Therefore, the question rises whether specific systems exist that can merge the capabilities of CAD and CAA systems with the reasoning competence and knowledge capturing and representation ability of KBSs. To a certain

extent the answer is positive: special kinds of KBSs exist, which either have the capabilities of a CAD system built in, or are tightly integrated to an external CAD system (also a combination of the two). These systems allow both implementing analytical procedures (e.g. computational algorithms) and communicating with external CAA tools. They are called Knowledge Based Engineering (KBE) systems!

KBE systems can be defined as an evolution of knowledge based systems towards the specific needs of the engineering domain. Actually, the name Knowledge Based Engineering originated just from the fusion of the two terms *knowledge based systems* and *engineering*. Certainly not in contraposition of doing engineering, not based on the use of knowledge!

In agreement with Lovett, Ingram and Bancroft, we can state that KBE systems are likely to be the best tools at hand whenever a candidate application area involves engineering domain knowledge and demand geometry manipulation and product (re)configuration (Lovett, Ingram and Bancroft, 2000).

As illustrated in Fig. 3.8 and often stated in literature (Chapman and Pinfold, 1999), KBE systems can be considered as the merger of *Artificial Intelligence* and *Computer Aided Design* technology. Not by chance, two of the founding fathers of ICAD Inc., the company that in 1984 developed ICAD, the first KBE tool ever on the market, were coming one from the AI laboratories of MIT and the other from the CAD company Computervision, later Parametrics and nowadays PTC (*Rosenfeld's profile;* Knudson; *ICAD*).

Whilst the entry of ICAD on the market can be considered as the very beginning of KBE, it cannot be considered the start of AI in CAD (or AI and CAD). At the beginning of the 1980s, a significant part of the international scientific community was already involved with the development of experimental systems to bring Knowledge Engineering capabilities into CAD (Tomiyama, 2007). As a matter of fact, the term *intelligent CAD*[3] was already coined in 1983 by Tomiyama (Tomiyama and Yoshikawa, 1983; 1985) just to address this novel concept of CAD systems able to store knowledge and reason on that to support geometry generation. The two epoch-making conferences organized in 1984 and 1987 on *Knowledge Engineering in CAD* and *Expert Systems in CAD* by the IFIP Working Group 5.2[4], were just a proof of the great activity in the field (Gero, 1985; 1987). The scientific discussion on Intelligent CAD, Expert CAD, Knowledge-based CAD, etc. has continued and evolved during the years, however, the ICAD system and the other KBE platforms that have

[3] According to the ICAD developers, the name ICAD was not the acronym of Intelligent CAD.

[4] International Federation for Information Processing [http://www.ifip-wg51.org]. In 2006 Group 5.2 (on Computer Aided Design) has merged into Working Group 5.1 (on Information technology in the Product Realization Process)

followed, possibly represent, still at date, the most successful industrial implementation of the whole "intelligent CAD" concept.

3.5 KBE systems and KBE applications. The programming approach

In order to practice knowledge based engineering, specific software tools, called *KBE systems* (or KBE platforms), are normally required. A *KBE developer* uses a KBE system to build so called *KBE applications*: dedicated *programs* to solve specific problems, generally (but not necessarily) related to the modeling and configuration of hardware products (both in terms of geometry and metadata). ICAD, GDL, Knowledge Fusion and AML are some examples of typical KBE systems. A list of commercial KBE tools, with some related information has been compiled in Appendix A. KBE tools market is very dynamic, refer to (*Knowledge-based engineering*) for up to date information .

A KBE system, similar to an expert system shell, is a general purpose tool, hence does not contain any knowledge about any specific domain (apart from the knowledge required to generate some primitive geometric entities, like points, boxes, cylinders, etc.). While a rule based shell allows the user to fill the knowledge base, by putting in rules via a proper interface, a KBE system requires *a programming language* to generate an adequate formalization of the given domain knowledge. Hence, while in the development of a rule based system no programming is required (if a shell is used), the development of a KBE application is just about writing code! State-of-the-art KBE systems provide the user with an object oriented programming language, which allows modeling the domain knowledge as a dynamic network of classes, in a way similarly to what we discussed concerning frame based systems.
Once a KBE application has been finalized, it can be packaged and deployed as a conventional Computer Aided Engineering (CAE) tool. In this case, designers, engineers and others involved in the design and engineering process can just use it, without being confronted with the syntax of the programming language running under the hood. As a matter of fact, KBE vendors usually commercialize two types of licenses: a *development* license, which enables the generation, modification and debugging of KBE applications and a *runtime* license to allow the use of compiled KBE applications just as normal executables (hence without any access to the source code).

Different from RBSs, but similar to FBSs, a KBE application shows no crisp separation between knowledge and inference mechanism. The domain knowledge and the control structure to access and manipulate this knowledge are strongly intertwined. Again, different from RBSs, but rather similar to FBSs, KBE systems leave the burden of modeling the knowledge domain (i.e. the selection of the adequate levels of abstraction and the definition of the proper networks of classes

and objects) to the developer. Expanding, updating and maintaining a KBE application is not just adding or deleting rules from a list. Even though the OO approach provides a sustainable way of writing spaghetti code (Graham, 2004), it is up to KBE developers to enhance their programming skill to enable code scalability and maintainability. On the other hand, the high level of flexibility and control provided by the OO approach is exactly what is required to build applications fully tailored to the users' need and to the peculiarities of the given products to be developed.

3.6 KBE languages: A survey of main characteristics

As discussed in the previous section, state-of-the-art KBE systems generally put at developers' disposal a programming language that supports the object oriented paradigm. As a matter of fact, KBE languages are very often based on object oriented dialects of the LISP programming language:

- IDL, the ICAD Design Language, is based on Common LISP, which is a dialect of LISP, including the CLOS (Common LISP Object System) object-oriented facility.
- GDL, the Genworks' General-purpose Declarative Language, is based on the ANSI standard version of Common LISP.
- AML, the Adaptive Modeling Language of Technosoft, was originally written in Common LISP, though, subsequently recoded in a proprietary–yet–LISP-similar language.
- Intent!, the KBE proprietary language developed by Heide Corporation and now integrated in the Unigraphics' system Knowledge Fusion, belongs also to the family of LISP-inspired languages.

Coding features	**KBE specific**	**LISP inherited**
Object oriented paradigm	√	√
Declarative coding		√
Dynamic typing		√
Runtime value caching & dependency tracking	√	
Interpreted/compiled mode		√
Automatic Memory management		√
CAD capabilities	√	

Table 3.1: KBE-specific and LISP-inherited characteristics of KBE languages

This non accidental occurrence of LISP in the KBE area is just another strong clue of the AI roots of knowledge based engineering. As a matter of fact, the LISP language, the second oldest programming language after FORTRAN, is still the favored programming language for artificial intelligence research and implementation.
Though many high-level OO languages have been developed during LISP's 50 years lifetime, such as C++, JAVA, Perl, Python and Ruby, LISP is still an extremely powerful and modern language (LISP itself has developed a lot). In the words of Paul Graham, famous LISP hacker and essayist "*If you look at these languages in order, Java, Perl, Python, Ruby, you notice an interesting pattern. [...] Each one is progressively more like LISP. Python copies even features that many LISP hackers consider to be mistakes. And if you'd shown people Ruby in 1975 and described it as a dialect of LISP with syntax, no one would have argued with you. Programming languages have almost caught up with 1958.* (Graham, 2004)".
The name LISP stands from *LISt Processing*, being lists the language major data structure. LISP source code is itself made up of lists. As a result, LISP programs can manipulate source code as a data structure, giving rise to the *macro* systems that allow programmers to create new syntax or even new "little languages" embedded in LISP. From here follows Foderaro's definition of LISP as a *programmable programming language* (Foderaro, 1991). Though LISP is by itself a high-level language, it is possible to use LISP to build even higher-level layers on top of itself. The result is a so called *superset* of LISP, and KBE languages like ICAD IDL and GDL are outstanding examples of supersets. To the user of these KBE languages, it means the full LISP language (and eventual LISP libraries) is always available and, on top of that, also special *macros* are available to provide the user with *higher*-level and user-friendly language constructs. Indeed, the availability of these macros represents the very added value of a KBE language with respect to raw LISP.
Table 3.1 shows a list with the main characteristics of a true KBE system (like ICAD and GDL) and, indicated whether the given characteristic is either KBE specific or just inherited from the LISP language.

3.6.1 KBE macros to define classes and objects hierarchies

The most outstanding example of a macro provided by various KBE systems (though in different form/syntax) is the one used for defining classes and objects hierarchies. Mastering the use of such macro is fundamental for developing any KBE application.
As a representative case, the ICAD-specific macro *defpart* is discussed in this section, since ICAD is the KBE system used for this research. In fact, GDL and Knowledge Fusion (KF) provide their own version of the same construct, though different names

and a slightly different syntax are used[5]. The simple (and very incomplete) mapping Table 3.2 shows the structure of the GDL *define-object* and KF *defclass* macros, which are the specific counterparts of the ICAD macro *defpart*.

ICAD	GDL	Knowledge Fusion (UGS)
defpart	Define-object	defclass
Inputs	Input-slots[1]	Any data type[2] followed by the behavioral flag[3] *parameter* (plus optional default value)
Default-inputs	Input-slot :settable (or :defaulting)	
attributes	computed-slots	Specification of several data types, plus an optional behavioral flag (e.g., *lookup*, *uncached* and *parameter*)
Modifiable-attributes	computed-slot :settable	A data type followed by the behavioral flag *modifiable*
Descendant-attributes	Trickle-down-objects	all attributes descendant by default
Type	Type	Class
Parts	Objects	Child
Pseudo-parts[4]	Hidden-objects[4]	Class name starts with *%*[4]
1: the term slot recalls the terminology of frame based systems 2: differently than ICAD and GDL, KF does not support *dynamic typing* (see section3.6.2). Hence the type of the attribute must always be specified, e.g., number, string and Boolean. 3: a behavioral flag might be used to specify the behaviour of an attribute. The flag *parameter* is used to create a correspondent of the input or input-slot keyword. 4: these objects will not be visualized in the object tree. When *%* is used as the first character of an attribute name, such attribute will not be visible from outside the object definition. That is how *information hiding* is supported in KBE languages.		

Table 3.2: equivalence table for the class definition keywords of ICAD, GDL and UGS Knowledge Fusion.

The *defpart* macro (or the non ICAD equivalent) is the basic means to apply the object oriented paradigm in KBE applications. It allows defining classes, superclasses, objects and relationships of inheritance, aggregation and association, as discussed in section 3.3. The defpart macro is basically structured as follow (see also the code sample of Fig. 3.9):

---

[5] The similarity of the three KBE tools mentioned above is not accidental. GDL is the youngest of the bunch, but like ICAD is based on Common LISP and has been developed as the natural heir of ICAD after the latter, in 2005, was acquired and put out of the market by Dassault Systemes. This similarity is exploited by GDL, which features a dedicated module to convert large chunks of legacy ICAD models directly into GDL, without any formal code translation required by the user.

Intent!, the KBE language at the base of UGS Knowledge Fusion, was licensed to Unigraphics by the Heide Corporation company. Mr Heide was one of the main developers of the ICAD system.

- **Name** of the class
- **Mixin-list**: list of superclasses or other classes from which the class here specified will inherit all the characteristics (attributes and components). The classes specified here can either be formal superclasses (of which the class specified in the defpart is an actual specialization), or other classes with which this class is to share attributes and parts (see next item).
- **Input-attributes**: list of parameters to be assigned in order to generate an instantiation of the given class. This set of parameters represents the so called class *protocol*. Default values can be specified outside the protocol.
- **Attributes**: these attributes are generally expressions which return a value when computed (see also the concept of *method* described for the FBSs). These expressions can either be production rules or any other mathematical, logic or engineering rule (see section 3.8 for a detailed list of possible rules). To evaluate these expressions, values of other attributes can be used or combined, such as the input-attributes or the attributes inherited by the classes specified in the mixin-list. It is also possible to use the attributes of the children (see next bullet) defined in the given defpart, or the attributes of any descendant or ancestor objects in the instantiated object tree: in this case the *reference chain* will have to be specified, as already discussed in 3.3.5.
- **Parts**: this is the list of objects contained in the instance of the defpart. They are also called *children* of the defpart instance. For each part, the following must be specified:
  - the object name
  - the name of the relative class to be instantiated (by using the keyword *type*)
  - values for the input-parameters of the class to be instantiated. This parameter list must sufficiently match the protocol of the class to be instantiated (i.e. at least its required input-attributes)
- **Methods**: these are similar to attributes, but they can accept arguments. Their computed return-values are not *cached* as is done by default with the attributes (see section 3.7 for caching)

Further details can be found in the relative documentation of the various KBE systems.

Any KBE application basically consists of a number of *defpart* (in the case of ICAD) definitions, properly interconnected as required to create structured models of both products and processes. Therefore, the whole network of defparts definitions is typically addressed as *the product model*, whereas the hierarchical structure of objects obtained by instantiating the various classes is called *the objects tree* or *product tree.*

Raw Common LISP already provides the capability to define classes and objects; what is the value of a macro like defpart, then? The defpart macro provides

a much simpler, user friendly and intuitive way of creating complex hierarchies of objects, without requiring engineers (the target users of KBE systems) to possess the hacking capability of a LISP expert. The defpart macro provides a kind of high level interface to the Common LISP object facility and maps any ICAD defined object to an actual Common Lisp object. This explains the double check in the OO paradigm slot of Table 3.1.

On top of that, the defpart macro brings in *caching and dependency tracking capabilities* (addressed in detail in section 3.7), which are normally not available in raw Common LISP. Indeed, behind the concise definition of a macro, there is generally hidden a voluminous and opaque chunk of LISP code, which automatically expands at compile time, (luckily) in a way that is fully transparent to the user. Some *macroexpansion* examples are given in the appendix of (Cooper and La Rocca, 2007).

The ICAD defpart. An example

Fig. 3.9 illustrates a sample of ICAD code, where the defpart macro is used to define the hypothetical class *ConventionalAircraft*. In Fig. 3.10, the UML class diagram and objects tree relative to this KBE application sample are provided as well[6]. Note that next to the specification of the class name, the *mixin* list appears, which is the list of other classes (*Aircraft* and *CostEstimationModule* in our example) from which *ConventionalAircraft* inherits. All the attributes and components of these two classes are readily available to *ConventionalAircraft*.

The two attributes *horizontalTailSpan* and *verticalTailSpan*, used to define the *Tail* part, are neither defined as inputs nor attributes of *ConventionalAircraft*; although not shown in the example, it can be assumed they are inherited from the superclass *Aircraft*.

The *CostEstimationModule* (rather than a real superclass) represents a hypothetical class containing some kind of costs calculation procedure. By including it in the *ConventionalAircraft* mixin list, any instantiation of *ConventionalAircraft* will inherit the capability of computing costs.

ConventionalAircraft is actually an aggregation of the 4 classes (since 4 parts are defined in the defpart) *Fuselage*, *Tail, Wing and AircraftCog.* It means any instantiation of *ConventionalAircraft* will be composed of 4 objects, of which 3

---

[6] Note how the name of classes, objects, attributes, etc. indicated in the diagrams respect the UML standards, i.e., class names are indicated as single words with capital letter; objects and attributes as single words in low case (in case of multi-words names, all the words are connected but opportunely capitalized). On the other hand, the names of the corresponding classes and attributes used in the ICAD code reflect the free style allowed by the programming environment. In the text above, for clarity, we will refer to the various classes, objects, etc.. using the UML style.

represent main aircraft subsystems, whereas the instantiation of *AircraftCog* is a non geometrical object with the ability to compute the position of the aircraft's center of gravity. This means that both geometrical and non geometrical components can be heterogeneously structured in the object tree.
As specified in Fig. 3.9 by means of the command "*Type*" and partly visualized in the UML diagram of Fig. 3.10 (top), the classes *Fuselage*, *Wing* and *AircraftCog* are specializations of the classes *Cylinder*, *WingGenerator* and *CogEstimationModule,* respectively. On the other hand, the superclass of *Tail* is dynamically evaluated by means of an IF-THEN rule. For instance, it can be a "null object" in case the attribute *typeOfTail* is evaluated to "tailless", or some other kind of tail, such as a the conventional configuration assumed in the example of Fig. 3.10 (bottom).
Although not shown in the example, clearly, this KBE application must contain the definitions of the defparts *WingGenerator*, *Cylinder*, *CogEstimationModule* and some other classes to define different types of tail. While the class *Cylinder* is actually one of the geometry classes predefined in ICAD (the so called ICAD geometry primitives), the others will have to be defined by the user using some other defpart.

In order to create an instantiation of the *ConventionalAircraft class*, the ICAD command "*make-part"* will be used. Then, the user will be prompted to provide values to the list of input-parameters (unless the *ConventionalAircraft* class is instantiated as part of some other class, in which case the parameter values will be *passed down* by its parent). Default values for the input-parameters can be assigned, which will be overwritten by fresh values provided by the user, or by the parent when applicable. The attribute values specified for the various parts will be the input values for their relative defparts; hence they will have to sufficiently match those defparts' protocol.

As shown in the UML representation of Fig. 3.10 (bottom), the object *myTail* contains two instantiations of the classes *HorizontalTail* and *VerticalTail* (although not shown in the example, both *HorizontalTail* and *VerticalTail* could be instantiations of two specializations of the class *WingGenerator*, similar to *myWing*). Hence, *myTail* is 1) parent of the two children *myHorizontalTail* and *myVerticalTail*, 2) a child of *myAircraft*, and 3) sibling of *myWing* and *myFuselage.*
The *object tree* shown at the bottom of Fig. 3.10 is the way a KBE system presents the modeled product to the user (actually a simplified version of the UML graph in the picture). Indeed, such *has-part* hierarchies are very familiar to engineers dealing with complex product configurations consisting of assemblies, subassemblies, components, subcomponents, parts and so on. The amount of hierarchical levels in the object tree generated by a state-of-the-art KBE system is actually unlimited.

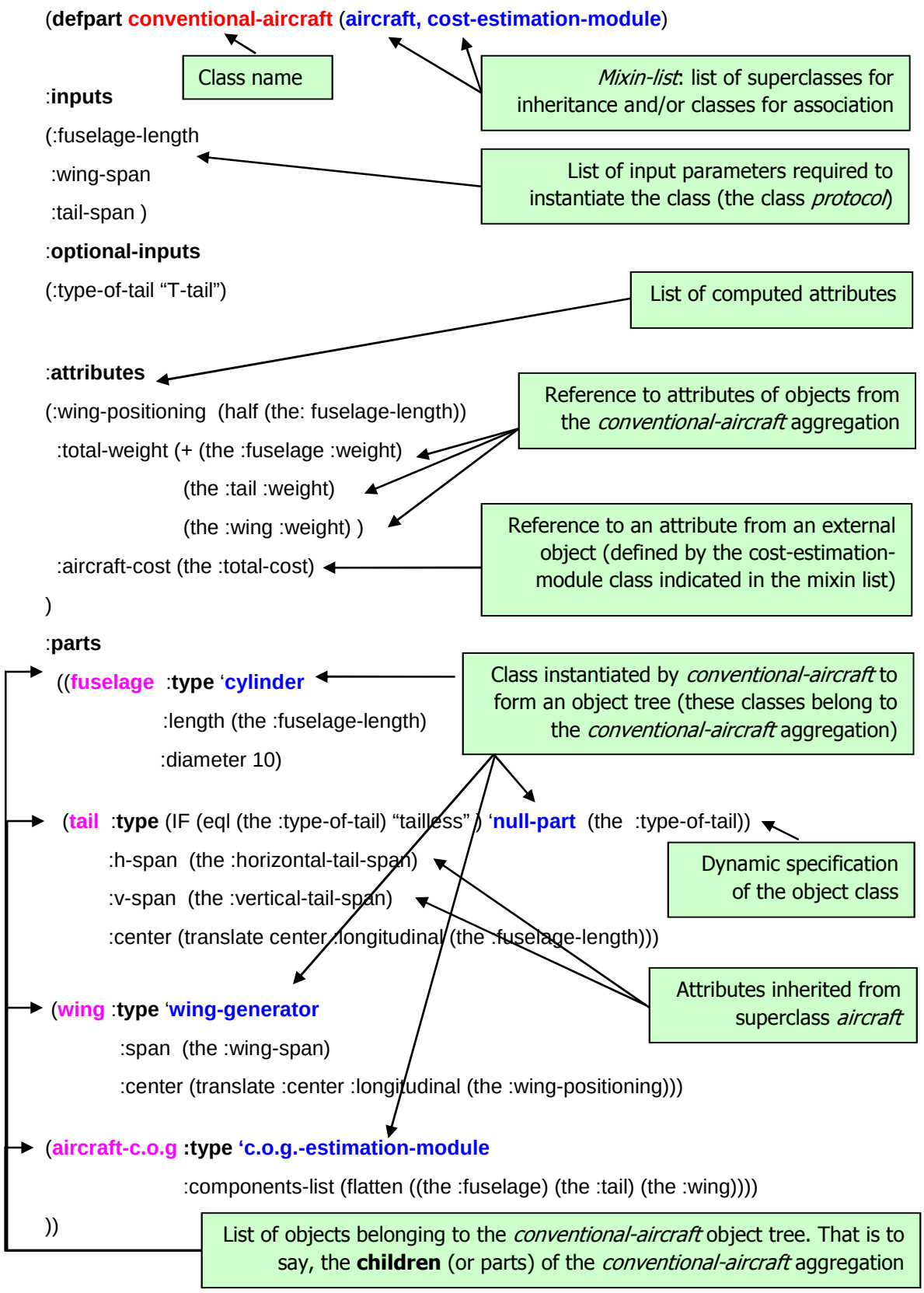


Fig. 3.9: example of ICAD code, showing a class definition by means of the macro *defpart*.

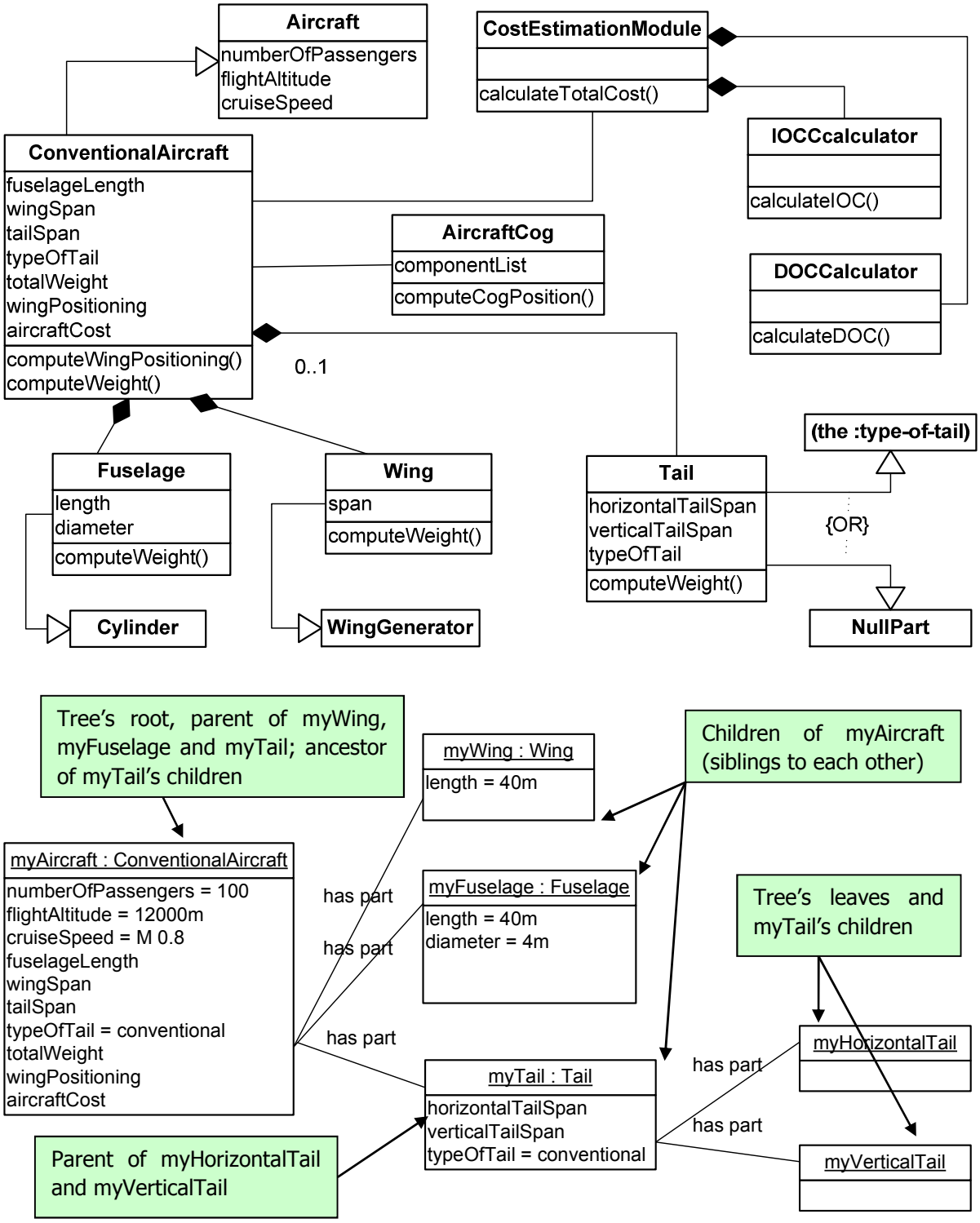


Fig. 3.10: (Top) UML Class diagram for the *ConventionalAircraft* class showing inheritance and composition links. (Bottom) Object tree resulting from the instantiation of the *ConventionalAircraft* class.

3.6.2 Flexibility and control: dynamic typing, dynamic class instantiation and objects quantification

One of the characteristics of KBE languages based on LISP, like ICAD and GDL (not Knowledge fusion as indicated in table 4.1, which is written in a proprietary language) is that *values have types, but attributes not* (at least not necessarily). Hence, contrarily to many general programming languages, like FORTRAN, attributes (variables) do not need to be declared ahead of time to be of a particular type. They can simply be created and modified on the fly, e.g., an attribute can change at runtime from a Boolean value like "NIL" to an integer number. Indeed, defining the type of an attribute value or object directly at runtime offers a high level of flexibility (Graham, 1995; Seibel, 2005). This programming style is known as *dynamic typing*.

Another relevant feature shown in the example of Fig. 3.9, is the possibility to use logic expressions for determining the object type at runtime (see the type definition of the part Tail and further examples later, in Fig. 3.12). Indeed the name of the class to instantiate can be treated as a variable.

In addition, each part can be defined as *a series of objects*, where the number of instances can also change at runtime, depending on the evaluation of specific rules. Defining series of objects in a KBE system is different from just creating "carbon copies" of the same part/feature as typical in many CAD systems. KBE systems allow the instantiation of each single object of the series by using *different parameter values*, as well as *different parameters*, since each object in the series can be the instantiation of a *different class* (see some examples later in Fig. 3.12). It follows that the topology of the product tree, i.e., the number of tree levels as well as the kind of objects in the tree, is not fixed but reconfigurable at runtime. The whole KBE model is dynamic by nature.

3.6.3 Communication between objects: the message passing mechanism

Objects interact by *sending messages* to each other. The input-attributes, attributes and methods of an object are all considered *messages* the given object is able to answer. In the code sample of Fig. 3.9, any instantiation of the *ConventionalAircraft* class (e.g., the object *myAircraft* of Fig. 3.10) sends a request to the *Fuselage* instance asking for its weight, which is needed to compute the *totalWeight* attribute. This is accomplished by including the addendum (the :fuselage :weight) in the specification of the attribute *totalWeight*. In object oriented parlance, any instantiation of *Fuselage* is said to be able to answer to the *weight* message.

To compute the expression associated to the definition of given attribute, an object might need to combine values of other attributes, which can be of its own, inherited from the classes specified in the mixin-list, or attributes of other objects belonging to the same aggregation (Cooper and La Rocca, 2007). In the example of Fig. 3.9, any instantiation of the *ConventionalAircraft* class is able to answer the message

totalWeight. In order to do that, it will send the message *weight* to the instances of its children *fuselage*, *tail* and *wing*.
As could be noted in the examples above, the operation of sending messages is performed by using the *referencing* macro *<the>*. This macro is used both to refer to the value of messages within the current object (e.g. *the :fuselage-length*[7]), or, through *reference-chaining*, to the values of messages in other descendant or ancestor objects from the instantiated object tree (e.g., *the :fuselage :weight* or *the :tail :vertical-tail :span*). The reference chain can be thought of as the *complete address* of the object we are sending a message.

Talking of parent/children relationships and inheritance might easily generate confusion. It should be clear that, in KBE parlance, children do not inherit from their parent, but from the classes they are a type of. In the example of Fig. 3.9, the children *Fuselage* and *Tail* inherit from the classes *Cylinder* and *WingGenerator*, respectively. However, they get the parameter values *length*, *diameter*, *span* and *center* passed down (or cascaded down) by the given *ConventionalAircraft* instance. The process of passing down parameter values from parent to children is the main mechanism to have information flowing down the object tree. On the other hand, the attribute *totalCost* (N.B. not the attribute's value) is inherited by *ConventionalAircraft* from its superclass *Aircraft*. Indeed, classes inherit from other (super)classes

To conclude, by *inheritance*, parameters are transmitted across hierarchies of classes (linked by *is-a* relationship); by *passing down*, parameter values flow down part-whole hierarchies of objects (linked by *has-part* relationship).

3.6.4 Declarative coding style

When writing a piece of code using a KBE language, in general there is no "start" or "end". The order in which attributes are declared and objects defined is not relevant at all. For example we can define attributes for the computation of the total weight of an assembly, before specifying the attributes defining the weight of the components of the assembly. The program interpreter/compiler will figure out at runtime the right order to trigger attributes evaluation and object instantiation. This coding style is opposed to the so called procedural style used, for example, in FORTRAN, where any procedure has to be defined step-by-step using the right temporal order of the events.
Common LISP is very special in this sense (and all the KBE languages based on CL), because it can support both styles. In facts, it is defined a *multi-paradigm language*. While the declarative code is extremely useful for writing dynamic software

[7] Note the correct way of referring to the value of a message within the current object is <the :self :message>, however the variable *self* is implicitly assumed when no other object is indicated in the reference chain.

applications, a local switch to the procedural approach is useful for example in cases like "IF Fact A is True THEN first Do this, then do that and subsequently do the other".

3.6.5 Polymorphism and encapsulation

The UML graphical representation for a class, as shown in Fig. 3.10, provides a slot for attributes and a slot for operations. The way classes are defined in a KBE system, for example by using the defpart macro, can make this distinction rather fictitious. Operations, indeed, are often executed by evaluating attributes. An attribute does not just contain a value but a method to compute it. As a consequence, when sending a message to an object a method is used to answer that message. Different objects can answer the same message but using different methods. In the example discussed so far, *myAircraft* sends a message to all its children to compute their weight. Each child will typically use a different, specific method to answer the message (e.g. the procedure to compute the weight of the fuselage will be different from the one to compute the wing weight). Hence, the operation *computeWeight()* is said to be *polymorphic*.

When sending a message to an object, the recipient object might need to start instantiating other objects and trigger the evaluation of several attributes in order to answer. In the example above, the object *tail* will have to force the instantiation of the two children horizontal-tail and vertical-tail in order to answer the weight message. However all these internal procedures and the associated clutter stay *encapsulated* inside the structure of the various objects and can be accessed/used via the message passing mechanism. As long as the interface of the given objects (i.e. the list of messages these objects are supposed to answer) to the external world stays the same, the internal structure can be changed, modified, updated without affecting the rest of the KBE application. This enabling feature for modular code development is called *encapsulation* and is provided by any OOP language. Indeed *encapsulation* (or *information hiding*), *polymorphism*, *abstraction* and *inheritance* represent the four required characteristics for a language to be considered object oriented (Rumbaugh et al., 1991).

3.7 The extra gear of KBE languages: Runtime caching and dependency tracking

As anticipated in section 3.6.1, the defpart macro (or the equivalent in other KBE languages than IDL) not only provides a high-level interface to the CL objects facility, but brings in *runtime caching and dependency tracking capabilities*. These two features, which are actually complementary to each other, are not present in raw CL and represent one of the most outstanding characteristics of a real KBE language.

Caching refers to the ability of the KBE system to memorize at runtime the results of computed values (e.g., computed attributes and instantiated objects), such that they can be reused when required, without the need to recompute them again and again…unless necessary. And here the dependency tracking mechanism kicks in, keeping track of the current validity of the cached values. As soon as these values are no longer valid (stale), they are set to unbound and recomputed only in the very moment their demanded again.

A dependency tracking mechanism is at the base of *associative modeling*, which is of extreme interest as will be shown later in this work. For instance, the shape of a wing rib can be defined accordingly to the shape of the wing aerodynamic surface. In case the latter is modified, the dependency tracking mechanism will inform the system that a regeneration of the rib shape is required because the previous definition (e.g., the contour of the rib flanges) is no longer valid.

In conventional programming, these activities need to be explicitly coded by the application developer, which is a non-trivial programming task. A KBE language does it automatically and completely transparently to the user. While in the past, people could argue about the large memory consumption due to the caching mechanism, the evolution of commodity computers has actually neglected such issue. Besides, every time a value or an object becomes stale, the LISP garbage collector takes care of claiming back the relative space in memory. This happens completely automatically and transparently to the user, who does not have to be involved at all in any *memory management activity*.

3.7.1 The power of demand driven evaluation

In general a KBE system has two possible ways of operating, namely by *eager* or *lazy* evaluation (or a combination of the two). We already discussed these two approaches when dealing with rule based and frame based inference mechanisms (sections 0.0.0 and 0.0.0). The KBE language compiler/interpreter in this case, either "eagerly" computes all the chains of values when some attribute has changed (e.g., a modifiable attribute gets a new value)[8], or "lazily" computes only those chains of values whose last value is demanded. The latter modus operandi, also called *demand-driven* approach is possibly the most interesting, hence typically set as default mode, for at least three reasons:

- The system computes values *when and only when* they are demanded (Cooper and La Rocca, 2007; Cooper, Fan and Li, 2001), hence there is no waste of computational resources (in terms of computing time and used memory).

[8] This is what actually happens in spreadsheet applications like Excel. As soon as a value in a cell changes, the value of all the linked cells is automatically and immediately updated.

- A typical object tree can be structured in hundreds of branches, unlike the very simple example of Fig. 3.10. The possibility to compute only those branches that are actually demanded allows a very efficient use (as well as test and debugging) of very large models.
- Application prototyping and maintenance is facilitated. The developer can focus on a limited part of his/her KBE application, while the rest of it may be possibly left incomplete or even incorrect. Since this latter part will not be evaluated automatically at run time (unless explicitly demanded), it will not generate any error, which would prevent the developer from testing just the branches of interest.

3.8 The rules of Knowledge Based Engineering

In section 3.3.2 it was discussed how in rule based systems the whole domain knowledge is represented in form of production rules. Frame based systems offer a much more sophisticated way of modeling the knowledge domain. Not only production rules are used, but also some simple data processing capabilities are present, and the entire knowledge domain can be structured according to the object oriented paradigm.
KBE systems, though commonly addressed as systems to perform *rule based design*, are much more similar to FBSs than traditional RBSs. KBE does not force expressing the whole domain knowledge in terms of production rules and, as discussed in section 3.6.1, offers special programming language constructs to define dynamic object hierarchies. In KBE parlance, all the possible expressions used to define attributes, to specify the number and type of objects, to communicate with other tools, etc. are all addressed with the generic term of *rules* (or *engineering rules*).
Within this large and heterogeneous group of rules, indeed we can distinguish a number of rule typologies, whose proposed definition is provided in the following subsections.

3.8.1 Logic rules (or conditional expressions)

Apart from the basic IF-THEN-ELSE rule, KBE languages like ICAD provide some more sophisticated conditional expressions, like *case* and *cond*, which are directly inherited from Common Lisp. From the ICAD manual:

> **case** *expression* (*test consequent*) &optional **otherwise** *otherwise-expression*
> Returns *consequent* if *expression* evaluates to *test;* if *expression* does not evaluate to any *test*, this returns *otherwise-expression* (if supplied) or **nil** (if *otherwise-expression* is not supplied).

cond [(*test consequent*) ...]
Returns *consequent* if *test* evaluates to **t**; if no *test* evaluates to **t**, this returns **nil**.

3.8.2 Math rules

Any kind of mathematical rule is included in this group, including trigonometric functions and operators for matrices and vectors algebra. Basic mathematical operators such as +, -, * are just Common LISP functions; many others are functions and macros provided by the given KBE language.
The mathematical expression:

$L = \frac{1}{2}\rho V^2 \cdot S \cdot C_L$ maps into the attribute definition

```
(:L (* 0.5 (the :rho)(^2(the :V))(the :S)(the :CL)))
```

where "^2" is a macro. Note the use of the *prefix notation* and the absence of the symbol "=", which is a Common LISP function to check whether two numbers are the same. These rules are commonly used for evaluating attribute values in a defpart and compute inputs for the parts to be instantiated. Of course, mathematical rules can be used both in the antecedent and consequent part of any production rules (IF-THEN rule).

3.8.3 Geometry handling rules

In this category we can include both the rules for the *generation and manipulation* of geometric entities and the *parametric rules*.
The first (see examples in Fig. 3.11) are actually KBE language constructs that allow the generation of many different kinds of geometrical entities, ranging from basic primitives (points, curves, cylinders, etc.) to very complex surfaces and solid bodies. Rules exist also to perform operations with the defined geometric entities, e.g., curves projections, surfaces intersections, solids subtractions and many others. In fact, these rules allow performing, via a programming language, many (all of the) operations normally possible in a CAD system by using the mouse and selecting the various menu/options provided by a graphical user interface. The big difference is that a CAD drawing eventually is just the recording of the final result of a human design process, while KBE rules can be applied to record directly the human design process and not just one specific end result.
Normally, geometry handling rules remain outside the range of conventional rule based and frame based systems. Unfortunately, these CAD-like capabilities often lead to the misconception that KBE systems are just CAD systems – and of very inconvenient species – where you are forced to write done in rules with syntax what you could normally do with the fancy GUI of a true CAD system.

Also *parametric rules* belong to the category of geometry handling rules. They allow expressing dimensions, position and orientation of a model element as function of the dimensions, position and orientation of another model element or some other constraint. These rules enable changes that are made to an individual element of the model to be automatically reflected throughout the rest of the model.
For example (not in syntax):

Position_of_point_A = Position_of_point_B + Translation_vector

Diameter _Hole = Diameter _Pin + Clearance_value

Generally, this kind of rules do not generally exist in rule based or frame based systems, because their evaluation requires *the notions of space and relative positioning* of assembly/parts/features. Indeed, these rules are typically available in conventional parametric CAD systems.
However, the possibility in KBE systems to combine these rules with logic rules and configuration selection rules (see next subsection) adds another dimension to the controllability of the model parameters. Spatial integration can be guaranteed also during dynamic variations of the product configuration.

```
(Defpart container (box)
   :attributes
   (:length 10
    :width 20
    :height 30))
```

Definition of a class called *container*, which is a box of dimensions 10X20X30

```
(Defpart wing (lofted-surface)
:attributes
 (:curves (list airfoil1 airfoil2)))
```

Definition of a class called *wing*, as a smooth surface that interpolates (*lofts*) the two previously defined curves *airfoil1* and *airfoil2*

```
 (:parts
  (:my-curve
    :Type surface-intersection-curve
    :surface-1 (the: first-surface)
    :surface-1 (the: second-surface)))
```

Definition of a class' child called *my-curve*, as the intersection curve between two previously defined surfaces called *first-surface* and *second-surface*

```
:attributes
(:distance-object
   (the :my-surface
   (:minimum-distance-to-curve
   (the :my-curve))))
```

Definition of a class' attribute called *distance-object*, as the minimum distance between the previously defined surface *my-surface* and curve *my-curve*.

Fig. 3.11: Examples of rules for the generation and manipulation of geometric entities.

3.8.4 Configuration selection rules (or topology rules)

These rules are actually a combination of mathematical and logic rules. However, they have a different effect than just evaluating a single numerical or Boolean value; hence they deserve a special label. They are used to change and control dynamically the number and type of objects in an object tree. Hence they can affect the topology of any product and process KBE model. Some examples are provided in Fig. 3.12. Note how it is possible to define dynamic series of objects, where each instance in the series can be individually specified in terms of attributes and type as well. As already discussed in section 3.6.2, these rules are generally not available in conventional CAD systems and represent an extremely powerful prerogative of KBE. Without this sort of rules, no real *generative design* exists!

```
parts
 (:my-beam
  :Type 'Beam
  :Quantify : series (round (div (the :total_load) (the :allowable_beam_load)))
  :Beam-length (if (eql (the-child :index) 1) 10 8)))
```

The number of beam instantiations is defined by a mathematical expression. The attribute beam-length of each series' objects is evaluated, based on a logic rule.

```
: parts
 (:my-beam
  :Type 'Beam
  :Quantify : series (IF (> (the :total_load) 100)) THEN 3 ELSE 2))
```

The number of beams is computed based on the evaluation of a production rule. A mathematical expression is computed to evaluate the premises.

```
: parts
  (airfoil
  :Type (IF (> (the :mach-num 0,75) 'supercritical-airfoil  'NACA0014)
```

The class of the object *airfoil* (i.e. the *type*) is selected at runtime based on a logic rule.

Fig. 3.12: Examples of configuration selection rules (topology rules)

3.8.5 Communication rules

In this group all the specific rules that allow a KBE application to communicate and/or interact with other software applications (not necessarily KBE) and data repositories are included. Rules exist that allow accessing databases or various kinds of files to parse and retrieve data and information to be processed within the KBE

application. Other rules exist to create files containing data and information generated by the KBE application. For instance, it is possible for a KBE application to generate as output standard geometry exchange data files like IGES and STEP, or XML files or any sort of free format ASCII files. Rules also exist to start at runtime external applications, wait for results, collect them and return to the main thread.

3.9 KBE product models to capture the *What*, the *How*...and the *Why* of design?

As anticipated in section 3.6.1, a KBE application eventually consists of a structured and dynamic network of classes and objects definitions, where both product and process knowledge, geometry-related and non are modeled using a broad typology of rules. This is the so-called KBE *product model*, and represents the core and essence of any KBE application.

The product model is occasionally addressed in literature as *rule base* (and the KBE approach as rule based design). This can be considered acceptable only if the fundamental differences between the flat and static structure of the rule base in a RBS and the dynamic, object oriented nature of the KBE product model are acknowledged first.

A product model is a generic representation of the product type for which the KBE application has been created. It is not made up of fixed geometric entities, with fixed dimensions, in a fixed configuration. Instead, it can contain the engineering rules that determine the design of the product (Cooper et al., 2001). The product model can function as a knowledge carrier to collect both the information concerning the physical definition of a given product (such as geometry, material and functional constraints), and the process used to design, analyze and manufacture it.

In particular, the focus of KBE is capturing the knowledge about *how* to design a product, rather than producing a static representation of the design process outcome. For example, the detailed drawing of an aircraft wing, including structural elements and systems, does not represent the design process and the knowledge required to generate such a wing design, but it is just the final result obtained from a specific *instantiation* of the knowledge owned by the team of wing design specialists.

A KBE product model is often claimed in literature to be the container of the *What*, the *How* and the *Why* of the design (Cooper et al., 2001). The *What* refers to the capability of capturing the physical definition of a product, with its shape, components configurations and features. The *How* refers to the sequence of steps, actions and transformations required to derive a product configuration, based on input requirements.

The *Why* has a more subtle meaning: it refers to the fact that a KBE system, similarly to rule based systems (see the RBSs' *explanation subsystem* description in section 3.3.1), is able to provide the user with the chain of reasoning/actions which

has led to the final solution. The Why, in the sense of the true intent behind the single rule, or the definition of the single procedure, or the justification of the levels of abstraction used to define classes, actually represents tricky knowledge to capture in the product model. Indeed, such knowledge is not even necessary for the KBE system to operate. To a limited extent, this "type of why" can be captured by means of comments and remarks to be inserted in the code, at discretion of the developer.

3.9.1 The generative capability

The functionality of the product model can be described by the simplified representation of Fig. 3.13 from (Cooper et al., 2001): a set of input values is assigned to the parameters used in the product model, the KBE system applies the rules which process the input values and finally the engineered design is generated, with little or no human intervention. This is typically addressed as *generative design*, and, not by chance, the product model is also known as *generative model*.

The example of Fig. 3.13 assumes the product model to contain the structured formalization of the multitude of corporate and regulatory standards and the handbook principles implemented by some company to deliver an engineered design. As a matter of fact, this example, where a fully engineered product is automatically generated starting from a list of input parameters, represents the use of KBE as envisioned by the first practitioners and promoted by the early (?) KBE vendors. Although success stories of fully integrated KBE design tools are reported in literature, in the author's opinion a different approach is necessary when dealing with very complex products and distributed design. As anticipated in Chapter 2 and discussed later in Chapter 6, a broader, modular design system is proposed, where KBE is used only for the development of one of the system components: namely an advanced parametric model to feed external analysis tools.

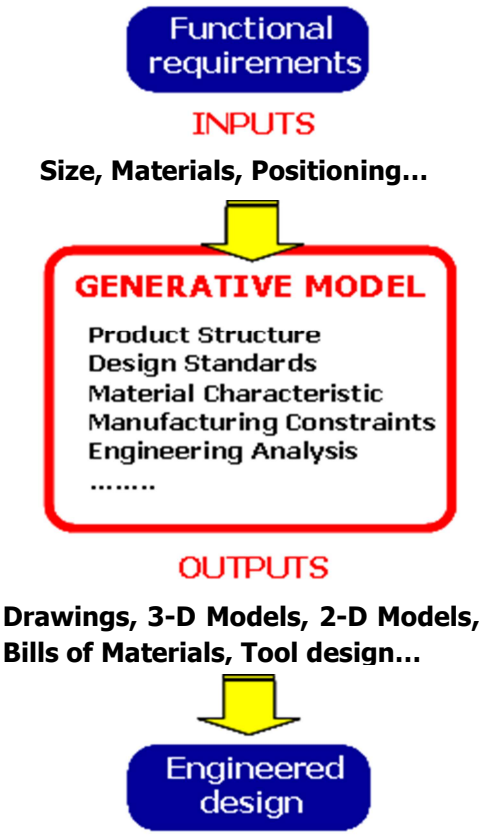


Fig. 3.13: the product (or *generative*) model of a KBE application takes input specifications, applies relevant procedures and generates a product design automatically.

in any case, whatever the level of complexity and "knowledge richness" of a product model, the way it enables generative design does not change substantially. The user has to create an instance of the class which defines the root of the product tree (in the example of Fig. 3.10 the root class was Aircraft). To do that he/she will have to provide a set of values to the input attributes of that class. Then he/she can ask the product model to *forcibly* compute all or specific branches of the product tree. Hence the root object will force the instantiation of its children (passing down the needed attribute values) and each child will do the same.

Otherwise, after having instantiated the root class, the user can ask the product model to deliver some specific output (like the bill of materials of the example of Fig. 3.13). In this case the demand driven mechanism will force the computation of just the branches and attributes of the product tree which are strictly required to deliver the requested output.

The relation between the given set of input data fed into the product model and the output design is univocal and totally unambiguous. Any time the product model is instantiated with the same input values, the same rule will be evaluated, the same objects will be generated, and the same results will be generated. The product model, by its nature guarantees conformity to all the rules implemented and the engineers can rely on the fact that all design derives from a documented and deterministic approach. The definition of the product model provides the clear reason for every dimension, design decision and configuration feature in the generated output, which represents invaluable information, both for designers and for those who review the design.

3.10 On the convenience of the programming approach

Knowledge based engineering is mostly about writing code. The use of a programming language represents the most salient operational characteristic of any true KBE system. The debate over the convenience of using a programming language to support engineering design is open since the first availability of IT systems and tools. The *imposed* use of a programming language, rather than the convenience of it, has possibly been the item number one into the discredit campaign of CAD vendors towards KBE technology. However, in the author's opinion, the advantages of a programming approach to support engineering design are multiple and evident:

- **Capture and communicate the design/model rationale.** When asked to show how "a thing" looks like, a person would probably start sketching something on a paper. When asked to explain how to make "the thing", very likely a person would start telling a story. To capture and communicate a design process, rather than its output, a language is required. To bake a cake, the picture of a baked cake is not sufficient, it is necessary to read the recipe.

- **Flexibility and control.** Designing by a sequential selection of commands displayed via a graphical user interface (GUI), though user-friendly, intuitive and often esthetically appealing, will inevitably limit the freedom of the designer[9]. Any user of a GUI driven tool has faced the problem of missing the button/menu choice to perform some specific operation. The availability of a programming language would enable generating the "whish-button". In general, a programming language can provide the means to capture reasoning schemes and control different series of events, without the need to go through often unnecessary series of menu selections.
- **Support automation and consistency.** Having a programmed generative model (as described in section 3.9.1) is like having a process recorded on some kind of playable medium. Every time the generative model is "played", there is guarantee that the same process can be repeated consistently (the same rules and reasoning mechanisms will be used) for different valid input values, whoever the operator and however large the number of re-plays. There are many cases in engineering design, such as design optimization, where the human interaction in repetitive processes is only an obstacle to automation and a potential source of errors.
- **A step toward standardization.** A request for proposal of a platform-independent model for the exchange of knowledge has been placed by the Object Management Group (Object Management Group, 2005). By using a programming language to capture engineering rules and relations, various KBE vendors have generated specific constructs, which, though syntactically different, are semantically equivalent. As shown by the class definition mapping Table 3.2, there are possibilities to create a standard to facilitate the transfer/exchange of engineering knowledge from one system to another.

[9] In order to provide more flexibility to the designers, without spoiling use simplicity, some advanced CAD systems offer the possibility to write programmable macros or make use of function calls to external routines (written in Fortran, C, C++, Visual Basic, etc.). However, hardcore programming at API level is generally the only way to access and manipulate all the features of the CAD system. In general, the results are not at the level of true KBE, where the programming approach is native. For example, Visual Basic macros are typically orders of magnitude slower than true KBE applications, because they are interpreted and not compiled. On the other hand, powerful languages such as C++ allow writing efficient code. However, the programming skills required to perform such CAD systems hacks are often higher than those required to develop standard KBE applications, which cancels the initial claim of use simplicity.

3.10.1 The characteristics of the ideal KBE-language

How should a programming language look like to be considered suitable for Knowledge Based Engineering? In two words we could say that it must be *engineer oriented*. More explicitly it should be:

- **High-level.** The programming language should be far more close to human language than machine language. It should keep engineers thinking and expressing themselves like engineers and not force engineers to think like machines. All the memory management activities should be taken over by the language, in a way completely transparent to the user
- **Concise**. While some languages need only a few lines of code to do something, others need pages of code. A lean language is required to support engineers working efficiently. A clear semantic and just a few axioms should be available (Graham, 2004), with the possibility to build on top of those.
- **Readable and comprehensible.** The main purpose of a language is to allow communication and knowledge transfer. If this is somehow prevented or limited to the scope of providing instructions to a computer, then the language is not adequate. *Programs must be written for people to read and only incidentally for machines to execute* (Graham, 2004).
- **Suitable to prototyping**. A language should allow the user to "sketch code" without forcing him to write optimally structured code at first hand. A language should allow telling the computer what to do, without entering in the details of how to do it. Dynamic typing and declarative style (see section 3.6.2 and 3.6.4) are two features that totally support code prototyping.
- **Efficient**. In the sense that it should make the work of the programmer efficient, rather than (or at least before) the work of the machine. Languages that take a lot of memory or do not have top of the class speed (with due limits of course), but allow designers to build working applications in a fast way are considered efficient. As Graham provocatively states "*Inefficient software isn't gross, what's gross is a language that makes programmers do needless work. Wasting programmers' time is the real inefficiency, not wasting machine times…especially when computers are just getting faster and faster* (Graham, 2004)".
- **Support reusability.** The concept of a language by itself is already about reusability: a limited amount of words that can be reused to express any kind of concept. However, some languages are more reusability supportive. The object oriented paradigm and the use of macro as discussed in the previous section are good examples. Reusability is at the base of code maintainability and scalability.
- **Multifunctional**. A language to support engineering design must encompass the heterogeneous aspects of the engineering design process. That is to say, it should support calculation and data processing, it should support the problem

solving/reasoning activity, and, last but not least, it should support the manipulation of geometry.

3.11 Summary 1: How KBE systems differ from conventional KBSs

This chapter has discussed the common roots and genes of KBE systems and traditional Knowledge Based Systems, like rule-based and frame-based systems. Knowledge acquisition, knowledge representation, and implementation of reasoning mechanisms are relevant aspects of commonality. However, in this chapter, also the substantial differences have been highlighted, which can be summarized as follow:

- KBE systems have geometry manipulation capability, because of the internal CAD engine or the capability to tightly integrate an external one.
- KBE systems have calculation and data processing capabilities, while conventional KBSs do not, or only to a very limited extent.
- In rule based systems there is a crisp separation between the knowledge base and the inference mechanism, whereas in KBE systems the functionalities of storing rules and using them to solve problems are intertwined.
- In rule based systems the whole knowledge domain has to be translated in terms of IF-THEN-like rules. In KBE systems there are far more possibilities.
- The programming approach used to define a KBE product model is such that there is never the risk of conflicting rules, which on the other hand, is a typical issue in rule based systems[10].

The typicality of KBE systems is such that the author still prefers addressing them as knowledge based engineering systems, rather than just knowledge engineering systems, as often found in literature.

3.12 Summary 2: How KBE differs from CAD

As Cooper (Cooper and Smith, 2005) well indicates, the original KBE systems were indeed created in response to a lack of capability in CAD systems and because of this, their marketing and overall positioning tended to be CAD-oriented[11]. However, as these systems quickly grew and became full general-purpose programming

[10] In order to help the inference mechanism selecting the right rule to fire in case of more (and conflicting) matching rules, extra *metaknowledge* needs to be added to the rule base. For example, a priority score is assigned to each rule, or priority is given to rules which use the most recent data, or priority is given to the rules with the longest list of matching conditions.

[11] In some cases this was evidenced by the name itself, such as *The ICAD System.* Nevertheless the company that developed the ICAD system has always denied it.

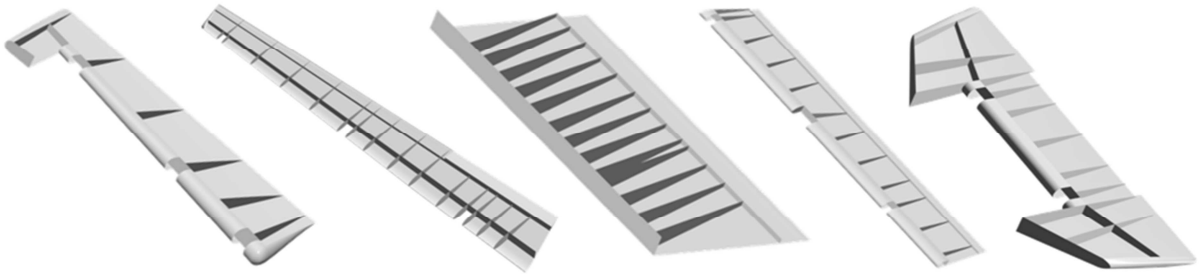


Fig. 3.14: Examples of very different aircraft movable configurations (e.g., rudders, airbrakes, elevators, ailerons), all generated as different instantiations of the same product model (skin removed to show inner structure) (van der Laan and van Tooren, 2005)

environments, clearly they fell into a category different from CAD systems. Anyway, the association in the marketplace with "plain old" CAD systems has persisted for many years and severely limited market penetration of KBE systems. It is only recently, possibly since world PLM/CAD leaders Dassault Systemes and Unigraphics UGS have entered the KBE services arena, that the awareness of KBE technology has reached other industries than giants like Airbus, Boeing, GE, Rolls-Royce and GM. Still, current systems like CATIA V and UGS Knowledge Fusion are CAD centric and have more KBE-ish than true KBE capabilities (Milton and La Rocca, 2008; Cooper and La Rocca, 2007). In the end, despite some points of contact, CAD and KBE represent two different technologies, as discussed in this chapter and summarized below:

- Developing a KBE application is 90% about writing code and 10% interacting with the GUI; in CAD 99% is about interacting with the system GUI.
- CAD systems can only output models, which are *human driven* records of the geometric results of a *human centered* design process. Whereas KBE systems are there to record the design and modeling processes (e.g., procedures, reasoning mechanism, best practices, computation and data processing) that lead to final (geometric) results.
- CAD is the most suitable tool for drafting, sketching and detailing. In those specific areas the interactive drawing capability of traditional CAD systems is more appropriate and efficient than the KBE programming approach.
- While CAD focuses on handling and delivering geometry, KBE focus on *rules and knowledge capturing*, with geometry being just one of the many types of output that can be generated. There are estimates that less than 50% in a KBE application is directly related to geometry (Chapman and Pinfold, 1999).
- The use of conditional and configuration rules (section 3.8) in the definition of a KBE product model allows dynamic alterations of the final product configuration which are far more drastic then the scaling/stretching and the variation of

features' patterns normally allowed by standard CAD systems. See the example in Fig. 3.14 from (van der Laan and van Tooren, 2005).

- CAD systems generally link to other software applications or analysis tools via standard data exchange format, such as IGES and STEP (or via specific interfaces developed by vendors to ease integration with specific commercial packages). KBE systems support standard data exchange format, but allow programming writer/parser modules for any kind of custom data format.

Interaction and automation, heuristic and by-the-rules, geometry related or not, one-off and repetitive are the typical coexistent and interrelated aspects of the design process: CAD and KBE can both contribute to this process and complement each other capabilities in a smart integrated approach.

CHAPTER 4
Conceptual development of the MMG. High Level Primitives and Capability Modules

1. Introduction
2. From designers' mind to concept visualization…
3. …and back! Object oriented modeling and functional thinking
4. High Level Primitives for the MMG modeling approach
5. Geometry modeling capabilities of the MMG
6. HLPs definition: an heuristic approach
7. From the aircraft geometry model to the abstractions for multidisciplinary analysis. Role and definition of the Capability Modules
8. Automatic generation of aircraft model abstractions
9. The MMG architecture: flexibility through modularity
10. Dealing with CAD engine limitations: capturing workarounds for robust modeling
11. Discussion

4.1 Introduction

Based on the discussion in Chapter 2, on the needs and challenges of developing a design tool that is able to support design space exploration and distributed MDO, this chapter will discuss the development of the DEE Multi Model Generator (MMG). It will be discussed how the object oriented modeling approach and the KBE technology presented in Chapter 3 have been exploited to achieve the following two main goals:

1. Provide an intuitive and effective modeling system for aircraft configurations and their variants, including non-conventional concepts.
2. Support and accelerate the (multi)disciplinary analysis of aircraft concepts, by automation of repetitive design activities, especially those required for analysis processing.

Whereas this chapter focuses on the *conceptual development* and the main functionalities of the MMG and its components, the KBE implementation details and

some application cases of the MMG will be presented in the Chapter 5 and 6, respectively.

4.2 From designers' mind to concept visualization…

As anticipated in Chapter 2, designers are, by nature, very good in translating customer requirements into product functional requirements and synthesize an adequate aircraft configuration. The way their creativity, engineering knowledge and past experience are exploited is not really understood and still constitutes an interesting topic for researchers in computer and cognitive science. What is acknowledged is that designers can do this part well and fast and there is hardly any need of computer aid. Actually, designers like so much this part of their job that they would hardly surrender it, even if software would possibly allow them to do it faster and better (Smith, 2007)

However, once the new concept is sparkling in the designer's mind, it is necessary to fix it on an adequate support (the mythical back of the envelope is not always the most ideal one), at least for the following reasons:

- Designers need to have their mental concept visualized to reflect on it
- Designers must be able to communicate the concept (to customers, specialists, etc.)
- Designers need to have suitable models to initiate the analysis and verification phase. Although some of these models for analysis might not relate directly to the physical shape of the aircraft, most of them contain geometry information, because the performance of an aircraft strongly depends on the interaction of its shape with the external world (i.e., passengers, fluids, ground, etc.)

The transformation of the designer's mental concept into a displayable model might definitely benefit from computer aid. Three practical approaches can be identified:

1. The use of a classical CAD system, where *any possible geometry model* can be assembled via a process of selection and manipulation of geometric primitives, such as points, curves, solids, etc. and, possibly, some other predefined CAD features.
2. The use of a modeling system in which a large (infinite?) number of *predefined parametric aircraft configurations* have been stored. The designer could choose the (best) matching `prefab´ aircraft model and adjust it by tuning the parameters values and/or switching on/off some of its features.
3. The use of a modeling system where a *limited number of predefined parametric modules (components)* is available, which the designer can adjust and combine to assemble large number of aircraft configurations and configuration variants.

The first approach is the traditional one; it comes at the cost of efficiency: the process has to be manually repeated any time a different aircraft configuration is suggested. Besides, it provides limited support for the later verification phase within a MDO framework (see Section 2.7 on the role and development challenges of the MMG). Eventually, this approach does not comply with the way engineering designers think, which is generally not in terms of geometry primitives like splines, points, etc.

The second approach is much more suitable for automating the model generation process. However, it is applicable only when the number of configurations to be examined is limited and predictable. If none of the available aircraft configurations fits, a new one has to be generated and added to the catalogue, which typically requires the use of a CAD system, as discussed above. Elsewhere, the risk is that designers are forced to adapt their idea to what is already available in the models catalogue. Also this method is not fully in line with the methodological approach of designers, which (mostly[1]) think in functions and not directly in solutions. The creative generation of a solution follows the need to fulfill a given functionality.

In this modeling approach, solutions are provided, which hopefully can fulfill the needed functionalities. In this case, truly innovative design does not appear to be properly supported.

The efficiency and effectiveness of the third approach, which is actually the one pursued in this research work, depend on the definition of the parametric modules that are provided to the user to play LEGO®... To understand how to define the appropriate parametric modules, it was considered opportune to "go back into the head of the designer" and try to capture his/her way of generating and visualizing solutions.

4.3 ...and back! Object oriented modeling and functional thinking

The object oriented modeling paradigm, introduced in Chapter 2, has a very good reason for its appeal: models built from objects allow a good mimic of the real world (Phillips, 1997; Sully, 1993). At least, the concepts of classification, abstraction and inheritance described in the previous sections seem to be very much in line with the way our mind "perceives" the world.

[1] Indeed, designers make use of their experience and in a certain extent "recycle" ideas that have proven effective in previous design cases (also from different domains). Cognitive science is exploring this problem solving method, known as *case based reasoning*.

One of the appeals of the object oriented paradigm is that it seems to be right in line with human nature. Perhaps we categorize the objects around us because it is easier for our brains to deal with a few categories rather than with many instances. Recent research points to brain areas involved in object categorization. Psychologists Isabel Gauthier and Michael Tarr used novel objects (*greebles*), purposely designed for this research, in conjunction with imaging techniques that show the brain in action.

They found that as people learned to categorize these objects (according to rules defined by the experimenters), the *fusiform gyrus*, a specific area in the cerebral cortex, became increasingly active (Shmuller, 2004b).

A greeble

The concept of objects is richly used in the field of *cognitive psychology*, and the sub-specialization concerned with knowledge representation. It has been demonstrated that people tend to represent knowledge in term of hierarchies, where the lower parts in the hierarchy are specializations of more general classes (sitting at higher hierarchical levels) and may have characteristics that add to or override some of those inherited from the general classes.

E. Rosch, a psychologist working in the area of concept representation, demonstrated that people record memory of objects in terms of a *prototypical schema*, which incorporates all the key representative characteristics of the objects in the form of a generalized abstract schema (Rosch, 1978). This prototypical schema becomes then the root of a hierarchy that possesses specializations. The more an item resembles "something", the more it is categorized in that "something" abstraction; hence it is included within an implicitly defined *range of typicality*. This natural process of memory and knowledge representation provides us with an efficient and economical way of arranging information and gives rise to the concept of *cognitive economy*, i.e. storing of information with the least possible effort (Sully, 1993). An example of how this knowledge representation schema can apply to an aircraft is shown in Fig. 4.1.

Some understanding of the prototypical schema structure and the way the typicality range is set could offer the opportunity to define a more effective modeling system, fine-tuned to the designer's own mental schema.

Whether conventional or out-of-the-box, any aircraft concept a designer could conceive, must fulfill a number of basic functionalities, such as accommodating payload, generating lift, etc. As a matter of fact, fuselage and wing like elements fulfill those functionalities mostly because of their characteristic shape (though some other shape could exist that allow integrating more functionalities). Possibly, the recurrent presence of such geometrical elements could determine the membership of

a given concept to the aircraft typicality range. Some X-configuration that does not feature any of these functional elements (i.e. a fuselage and/or a wing like element) is likely to fall outside the aircraft typicality range, which does not give the certainty the X-configuration cannot be a proper aircraft, but it is certainly a good hint.

The object oriented model of the aircraft

At the top of the hierarchy there is the *prototypical schema* of the aircraft: this aircraft abstraction features all the typical characteristics of a small tourism aircraft plus those of a commercial jetliner plus those of a fighter aircraft. An actual commercial airliner will be just a specialization of the prototypical aircraft abstraction. In turn there will be further specializations of the commercial airliner such as a high-wing turboprop, a T-tail version with fuselage mounted engines, a freight configuration, etc. Each one of these specializations adds to and/or overrides some of the characteristics inherited by the category abstraction it belongs to. For example, the freight aircraft overrides the characteristic "*has passengers accommodations*" of the commercial airliner abstraction, but inherits some other characteristics, such as "*has low wings*", etc.

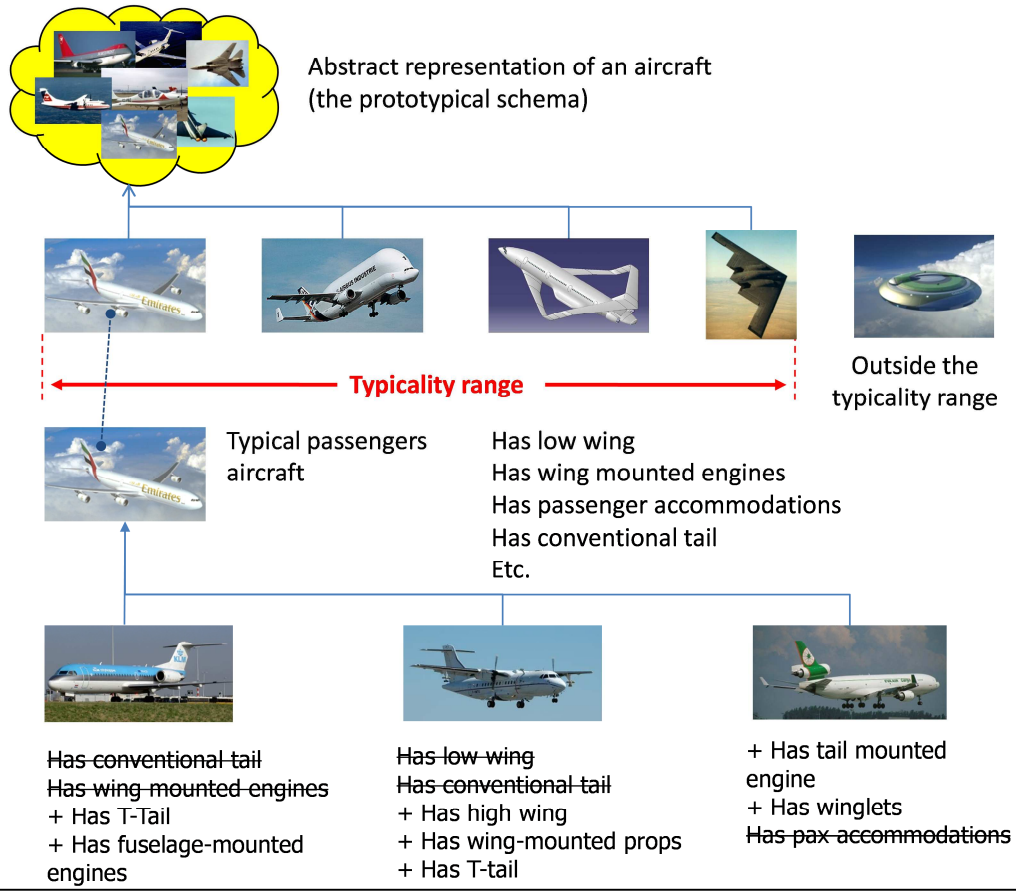


Fig. 4.1: the object oriented model of the aircraft. Prototypical schema and typicality range.

4.4 High Level Primitives for the MMG modeling approach

On the base of the considerations above, it was decided to define a number of functional blocks able to capture *elements of similarity* among very different aircraft configurations and use them as the parametrical modules for the third practical modeling approach proposed in section 4.2. These modules have been given the name of *High Level Primitives* (HLPs), mainly to address the fundamental distinction with the low level primitives of conventional CAD systems. Their implementation is at the base of the modeling capabilities of the Multi Model Generator.

With the definition of just four High Level Primitives, such as *Wing-part*, *Fuselage-part*, *Engine* and *Connection-element* (Fig. 4.2), it is possible to assemble a very large number of aircraft configurations, even with radically different topologies. The functionality of the first three HLPs is obvious; the connection element is needed to join the others into a continuous, watertight surface. Fig. 4.2 and Fig. 4.3 demonstrate the concept, by showing the (re)use of the High Level Primitives to model a traditional airliner and a blended wing body aircraft.

The HLPs allow a much more efficient transition from mental concepts to displayable models, than the low level geometry primitives of a traditional CAD system. Whilst a new CAD model must be manually generated for every different aircraft

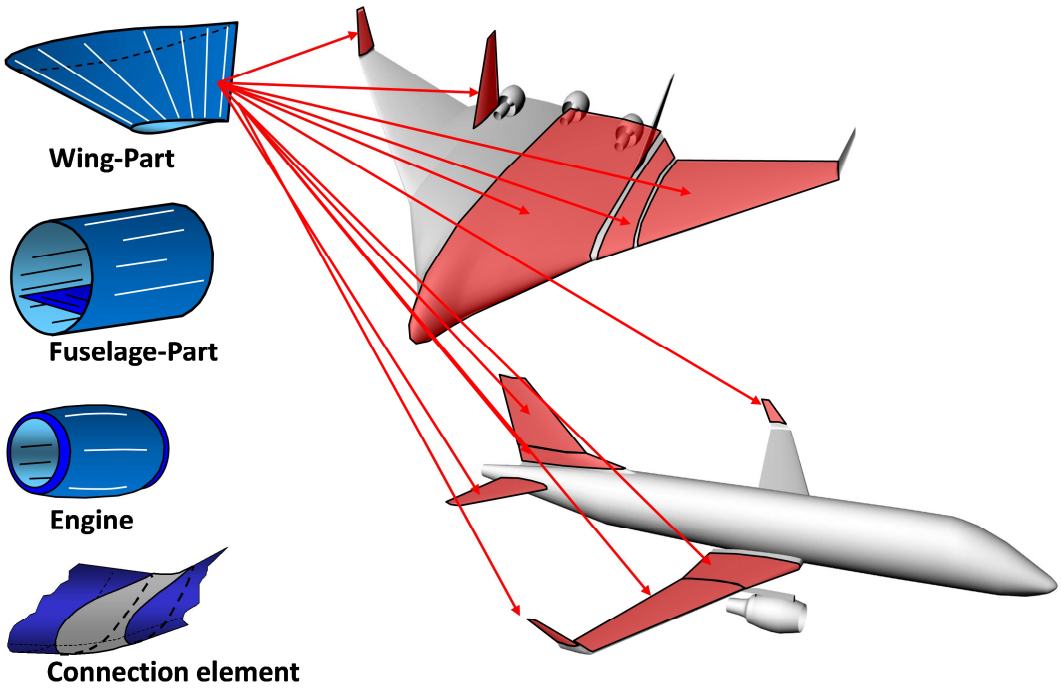


Fig. 4.2: multiple instantiation of the *wing trunk* HLP to model all lifting generating aircraft components

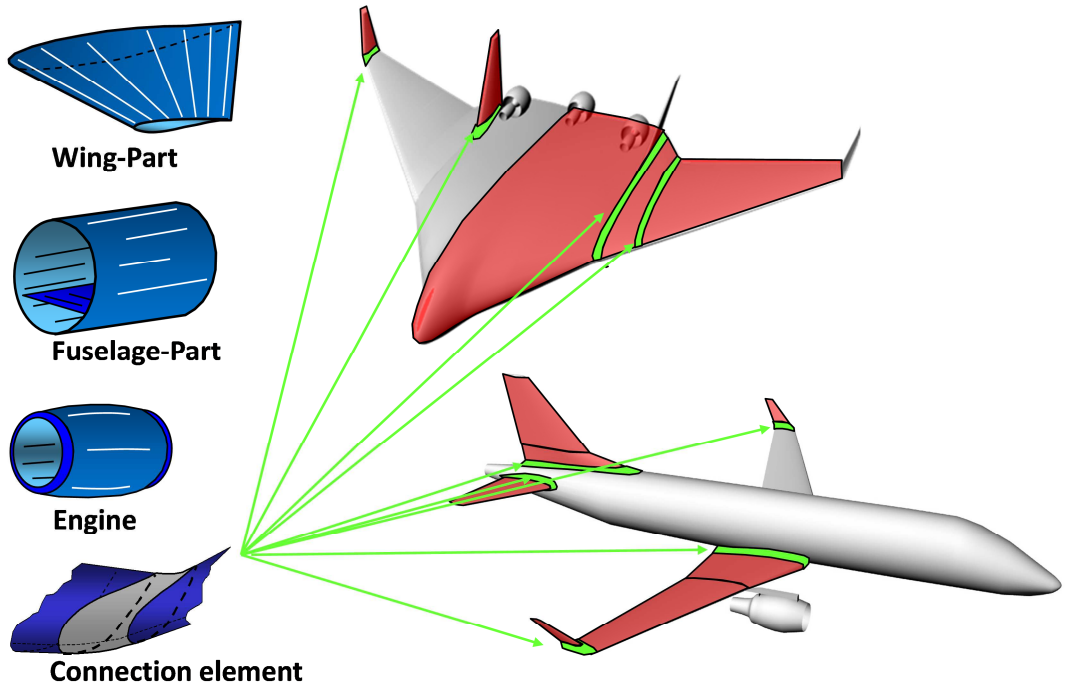


Fig. 4.3: multiple instantiation of the *connection element* HLP to blend the other HLPs instantiations in a continuous water-tight surface

configuration, the HLPs will take care of generating the required curves, surfaces, etc., for the final geometry visualization of the aircraft configuration at hand. What Fig. 4.2 and Fig. 4.3 show is that the HLPs can capture elements of the aircraft prototypical schema, hence not instances but concepts and mental categories, which can be adapted to record different specific models. Indeed, the same wing part primitive, can be used to model wing parts, winglets, canard wings, tail empennages, as well as movable components as rudders, ailerons, etc.
Again the object oriented paradigm suits the case. Indeed, the HLPs can be modeled as *classes* that, once provided with a new set of attribute values, can be instantiated in different *objects*[1] . For example, the class `WingPart` can be instantiated in the winglet of the Boeing 737-800, the fin of the Airbus 340-500, the canard of the Sonic Cruiser, or the outer wing of the MOB blended wing body.

---

[1] To distinguish between the HLPs and the classes defined for the software implementation of the HLPs, the following notation will be used:

Wing-part (the HLP) → `WingPart` (the class used for the software implementation)

Fuselage-part → `FuselagePart`

Connection-element → `ConnectionElement`

Engine → `Engine`

Given its capability to fully embrace the object oriented paradigm and bring along the geometry manipulation ability of parametric CAD, Knowledge Based Engineering appears the right technology to implement the HLPs principles for the development of the Multi Model Generator. Eventually, the selected KBE system employed for this work is ICAD (see Chapter 3), which, at the time the MMG development started, was the top-of-the-class system in the field.
The HLPs have been modeled using the *defpart* macro discussed in Chapter 3 and integrated in the architecture of a more complex product model. In fact, the goal of the MMG is to allow modeling of *complete aircraft configurations*, not just and only single components. The UML diagram of Fig. 4.4 offers an interesting view on the conceptual definition of the MMG, excluding the implementation details, which will be addressed later in this chapter and in Chapter 5.
In this graph the MMG product model (here addressed as `GenericAircraftProductModel`) is represented as an aggregation of the three HLPs[2] classes `FuselagePart`, `Engine` and `ConnectionElement` and a fourth assembly called `LiftingSurface`. The `LiftingSurface` assembly can be used to model a wing, or a tail empennage, or a movable, which, in fact, are all indicated in the graph as *specializations* of `LiftingSurface`. `LiftingSurface` is actually an assembly composed by an unlimited number of `WingPart` and `ConnectionElement` HLP classes. Indeed, it is possible to use more Wing-part primitives to model a lifting surface, as shown in the examples of Fig. 4.2 and Fig. 4.3, where four instantiations of the Wing-part HLP are used for the center section, the inboard wing, the outboard wing and winglet of the blended wing body lifting surface, respectively. The number of instantiations of the Connection-element HLP is not predefined, but it is computed on the fly according to the need of enforcing surfaces continuity at component fuselage-lifting surface intersections (e.g., wing/fuselage) or between the various WingPart instantiations in the same lifting surface.

[2] Note how, though the aircraft product model and the HLPs are actually classes, they have been defined using the special *stereotypes* «MMG» and «HLP». This was done to highlight their role of "special classes" and add more meaning to the graph. Stereotype customization is a possibility offered by the UML (Shmuller, 2004a).

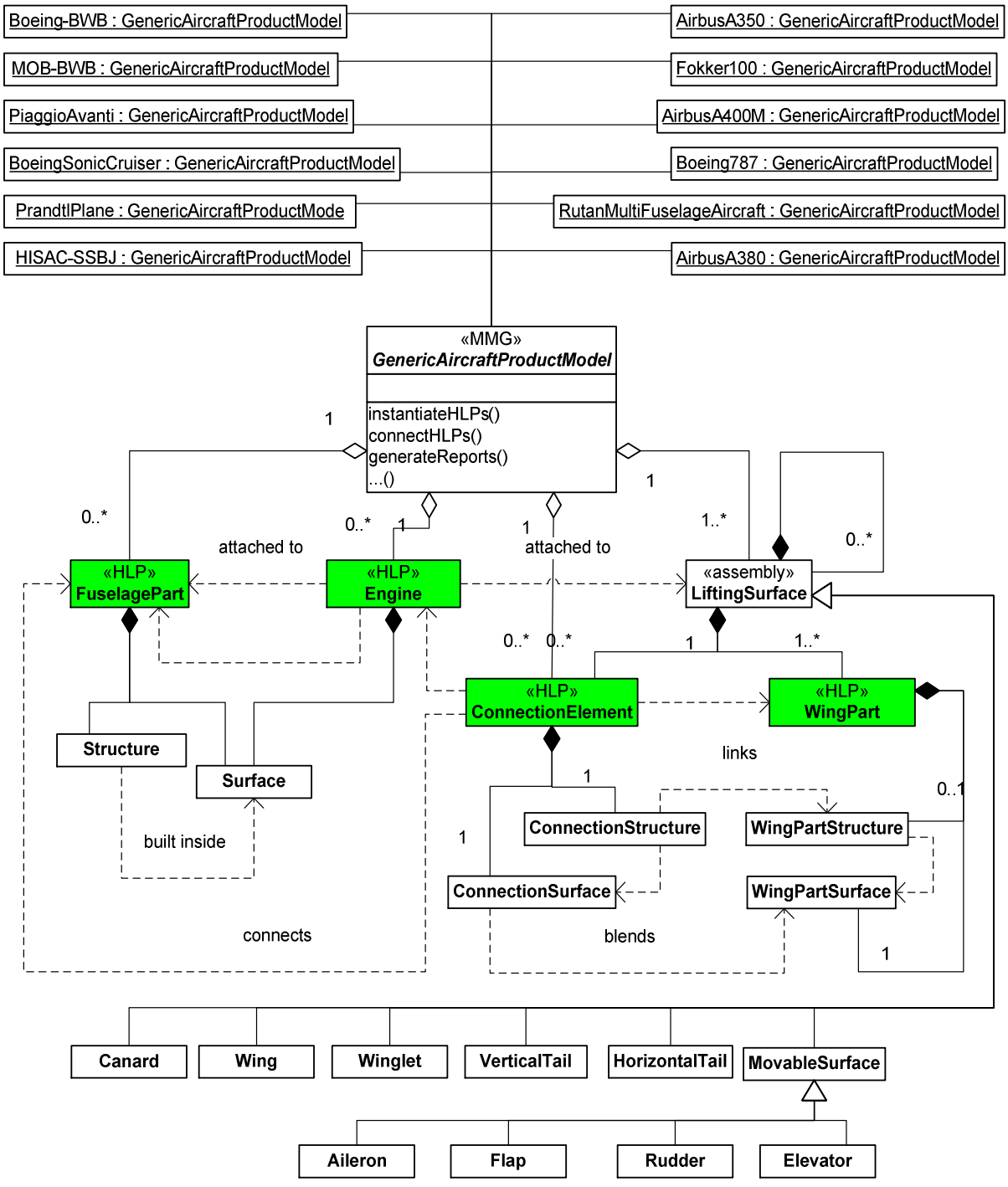


Fig. 4.4: UML class diagram representing the generic aircraft metamodel (the *aircraft of the mind*) and a number of instances (i.e. the *aircraft of the sky*). Representation of the High level primitives' aggregation.

Eventually, the MMG of Fig. 4.4 can be considered an attempt to mimic the functionality of the prototypical schema addressed in Section 4.3. In this case, the HLPs enable the generation of different aircraft configurations, within a certain typicality range. On top of the graph, there is a heterogeneous set of aircraft (the *aircraft of the sky*), which, during this research work, have been generated as instantiations of the aircraft product model (the *aircraft of the mind*). The class diagram of the MOB blended wing body aircraft (La Rocca et al., 2002) is shown in Appendix C, as an example of a specialization of the aircraft product model.

4.5 Geometry modeling capabilities of the MMG

The geometry modeling capabilities of the MMG mostly depend on the definition of the various high level primitives. The HLPs can be considered as a kind of rubber LEGO® blocks, which can be individually morphed due to their parametric definition and assembled to build up a potentially infinite range of different aircraft configurations and variants. Indeed, the parameters used to define the various HLPs represent the actual *degrees of freedom* of the primitives and determine the *typicality-range* of the specific instantiations that can be generated.

The ability of the Wing-part HLP, for example, to capture the resemblance of (a part of) a wing, a canard, a fin, a winglet, etc., depends on the fact that parameters such as chord lengths, span, sweep and twist angle, as well as type and location of the various airfoils can all be defined and controlled by the designer. In case a complex wing configuration has to be modeled, featuring a number of kinks and different values of dihedral and sweep at the various wing sections, the designer can use multiple instantiations of the Wing-part primitive. In this case, a set of parameter values must be provided for each instantiation of Wing-part.

Fig. 4.5 shows some examples of aircraft models generated by the MMG. The family of blended wing bodies has been generated by

Genes and families

According to a sort of *genetic engineering* interpretation of the HLPs concept, the rules defined inside the HLPs classes (e.g., to fit curves, generate surface) constitute the *gene print* of the primitives, hence they represent the *commonality elements* of the HLPs: they encapsulate the knowledge to perform certain tasks and, eventually, determine the typical behavior of all the class instantiations.

On the other hand, the HLPs' parameters (e.g., length, width, sweep angle) allow the *morphologic variation* of the primitives, hence they represent the *individuality elements*: a different set of parameter values yield an HLP's instantiation with a different shape, which makes it unique among all the other instantiations.

This approach based on the definition and combination of commonality and individuality elements to generate individual/specific models, that anyway share common characteristics and behavior, is addressed as *family-thinking*.

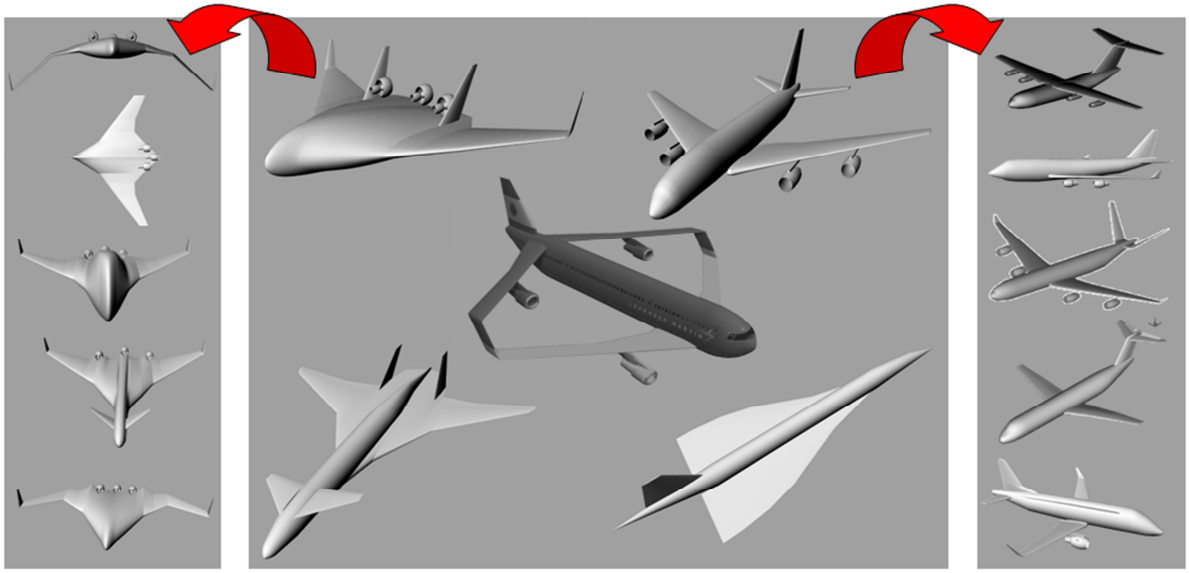


Fig. 4.5: Generation by the ICAD MMG of many aircraft configurations and configurations variants, based on the use of High Level Primitives.

using the same HLPs aggregation, and modifying only the parameter values of some primitive instantiation. The family of more conventional aircraft on the right has required also modifications to the number of HLPs instantiations to change the number of engines and the type of tail configuration.
Eventually, the procedures used by the HLPs to perform the various geometric operations, such as the generation and positioning of curves with respect to opportune reference frames, the interpolation and intersection of various surfaces etc., are all stored in the form of *rules* inside the body of the relative defpart definition. Not only geometry handling rules, but practically all the types of rules described in Chapter 3, Section 3.8 have been used to define the MMG components. For example the connection-element HLP, use topology rules to check whether the generation of connection surfaces is necessary to blend adjacent wing-parts in case of different dihedral angle. Geometry rules, then, enforce the generation of an appropriate connection shape that guarantees overall surface continuity (i.e. based on the local tangency vectors of the adjacent wing-part surfaces to be connected). The technical details on the HLPs implementation can be found in Chapter 5.
Indeed, the KBE approach offers the possibility to *encapsulate knowledge* inside the various HLPs (in this case the knowledge required to perform geometry manipulation), which transforms them in smart and dynamic objects, by far superior to the low level primitives of traditional CAD systems. See this concept illustrated in Fig. 4.6.
Whenever a parameters value is changed and/or the number and type of HLPs used to model a given aircraft configuration, the rules integrated in the aircraft product model definition will enable the model to reconfigure and adjust itself, *automatically*

and without any burden for the designer. This is the power of the KBE generative modeling!

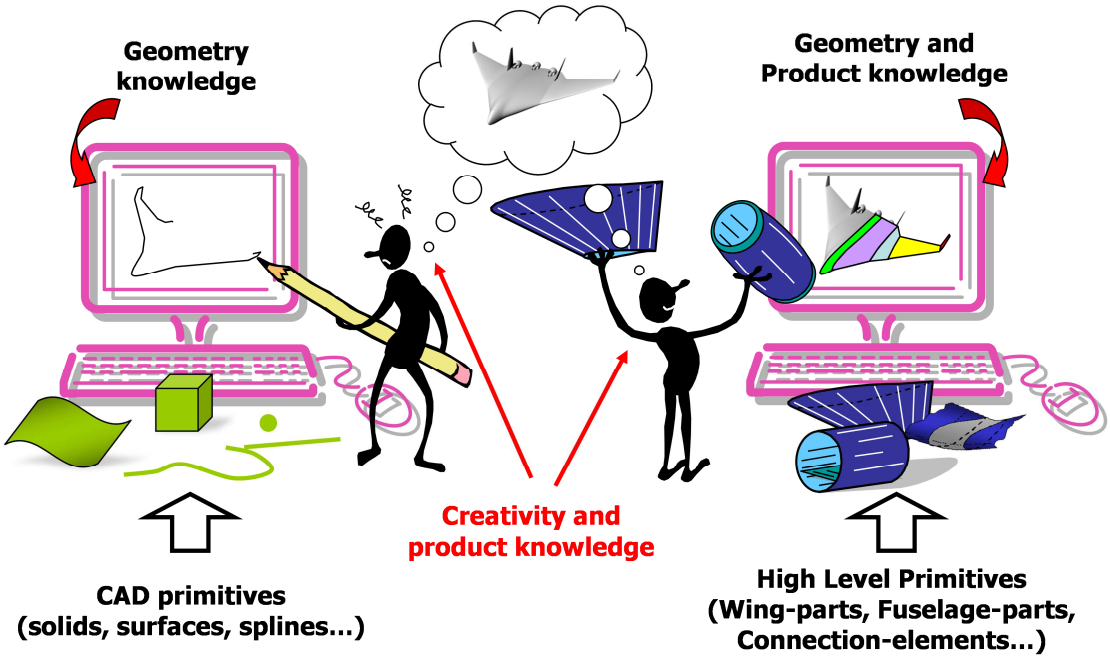


Fig. 4.6: Generation of aircraft models - the KBE High Level Primitives vs. the low level primitives modeling approach of traditional CAD systems.

4.5.1 MMG input file definition for batch modeling

In order to avoid the use of various pop-up menus and other forms of graphical user interface, all the parameters values that are required to instantiate the complete aircraft product model are exposed and assigned via a dedicated input file (see examples of the MMG input file in Appendix D, related to the discussion of Chapter 5). This file is easy to edit either by hand or, automatically, by other software tools (e.g., by an optimizer). Indeed, this represents an important feature, because it allows using the MMG also in batch mode, as required by the operation of the Design and Engineering engine (see previous discussion in chapter 2). The input file is "mixed in" at the root defpart of the aircraft product model, which takes care of passing the parameters values down the product tree's hierarchy levels (sections 3.6.1.1 and 3.6.3 discuss the use of mixins and the attribute values cascading mechanism, respectively).

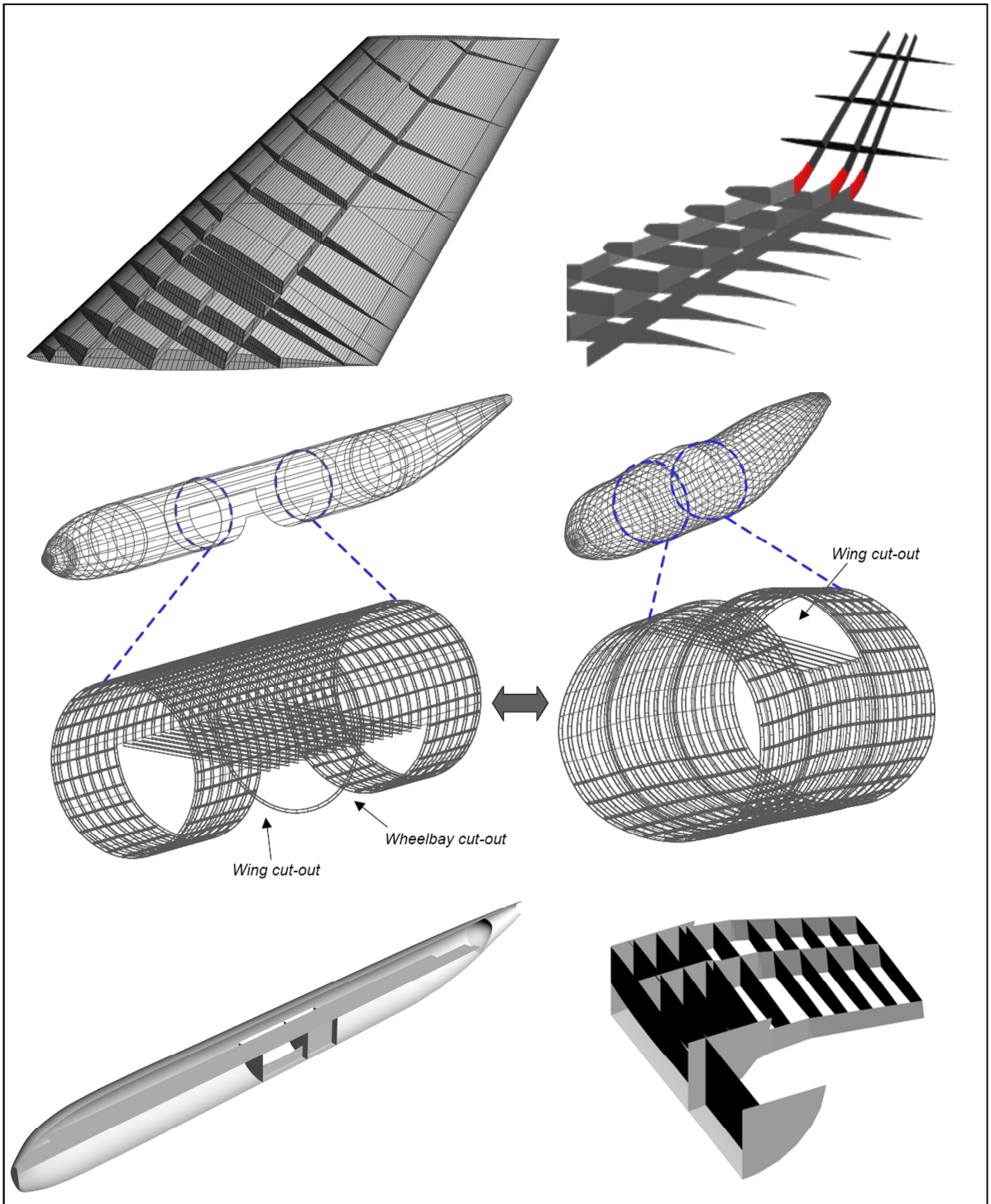


Fig. 4.7: Examples of structural models generated by the MMG. *Top left*: spars, ribs and riblets of a lifting surface. *Top right*: connection elements between wing and winglet spars. *Middle*: floor beams, frames and stringers in the center section of low and high wing transport aircraft. *Bottom left*: floor panels, pressure bulkhead and center section structure in a passengers aircraft. *Bottom right*: detail of center wing structure and keel beam in a low wing aircraft.

4.5.2 Definition of the HLPs' internal structure

The definition of the wing-part, fuselage-part and connection-element HLPs is not limited to the parametric description of the aerodynamic surfaces (La Rocca and van Tooren, 2002b), but includes also the internal structure. So far, the MMG offers the possibility to define the typical spar/ribs and frames/stringers/floors structure solutions for wing and fuselage-like elements, respectively (La Rocca and van Tooren, 2002a).

Van der Laan has extended the range of possible structure solutions to include the use of sandwich, though the implementation is so far limited to a movables dedicated model generator (a kind of MMG KBE tool, for aircraft movables such as rudders, elevators, ailerons etc.), based on the use of the same MMG wing-part HLP (van der Laan and van Tooren, 2005).

By adjusting the values of a dedicated set of input parameters in the MMG input file (examples are provided in Appendix G, I), the designer can modify the position and orientation of each single structural element (e.g., ribs and spars), as well as the overall structure topology (i.e., modify the number of spars, ribs, floors, etc.). The challenge here was to give designers the maximum freedom for positioning the single structural components inside each HLP, while limiting the number of required parameters, as well as the possibility of spatial integration errors. Similarly to the definition of the outer surfaces, also here the possibility to capture rules inside the HLPs definitions was the enabling factor to the generative design. Rules are necessary to determine the amount of connection elements (if required) to

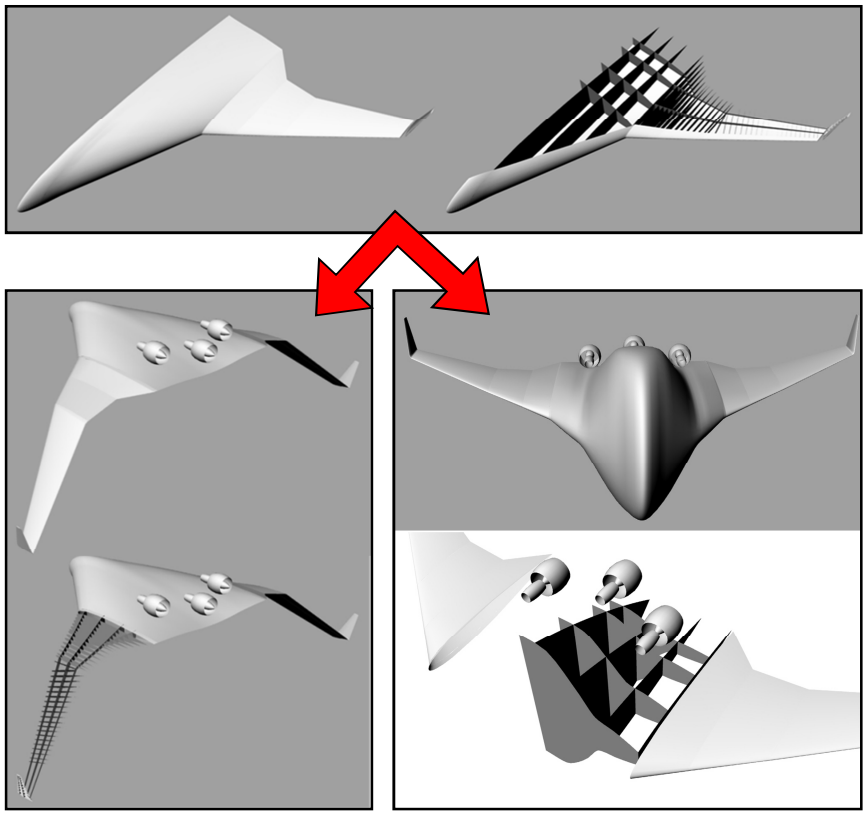


Fig. 4.8 The internal structure definition is associated with the external surface of the aircraft and adapts to its modification.

guarantee spars continuity across adjacent wing-parts. Rules are required for interpreting the values of the input parameters and accordingly use different positioning algorithm for ribs and spars. Fig. 4.7 shows various examples of structural configurations generated with the MMG, and give an idea of the achieved level of modeling flexibility. Note some of the details in the top left wing element: there are both continuous and running out spars. The ribs defined in the wing box and in the leading/ trailing edge sections are all individually oriented, and do not have to run necessarily from front to back spar. It is acknowledged that the level of modeling detail of each structural element is rather low, yet sufficient for conceptual and preliminary study (also by means of Finite Elements analysis) of complete aircraft configurations. Indeed, ribs, spars, skins, etc. are represented as simple surfaces, without any cutout or details such as flanges, caps, etc. As a matter of fact, what we address here as a spar model is actually the model of a spar web! Nevertheless, this modeling approach represents a very good compromise of complexity and fidelity.
The actual definition of the modeling approach for the various structural elements, the use of the relative input parameters and the implementation details in the KBE system are all covered in Chapter 5.

One of the most relevant characteristics of the structure definition is its *associative* link with the HLP outer surface. That is to say the geometry of the various structural elements is defined using the outer HLP surface as boundary, hence when the latter is modified the shape of the structural elements will automatically adapt. Fig. 4.8 shows examples of radical modifications enforced to the design of a blended wing body aircraft: in the bottom-left case, the outer wing is significantly deflected downward, but the shape of the spars geometry adapts without generating discontinuities at the connection elements; in the bottom-right case, the center body has been "inflated" by the implementation of ultra thick airfoils, still the geometry of the internal walls structure adapts automatically.
In the word of Carty, MDO systems specialist at Lockheed Martin, "*the benefits of having associative geometry models are almost something that has to be experienced to be appreciated*" (Carty and Davies, 2004). Indeed, the possibility to investigate even large design modifications, with the possibility to weight systematically any aerodynamic improvement against the consequences on the structure design, is a fundamental enabler of the MDO approach.

4.5.3 Definition of main systems and other non structural masses (the MOB prototype)

During the MOB project framework, the prototype of a modeling system for non structural masses (NSMs) has been implemented in the MMG (La Rocca and van Tooren, 2002a). This model includes the definition of typical aircraft systems such as

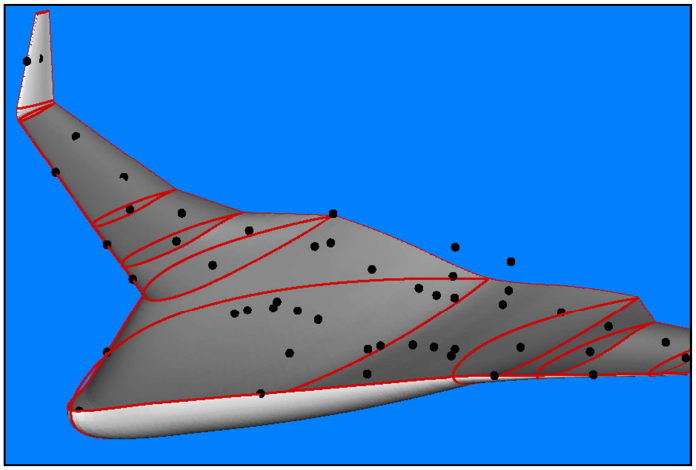


Fig. 4.9: Aircraft systems represented as lumped masses, parametrically positioned with respect to the aircraft structure elements (aircraft upper skin and structure removed) (Laban et al., 2002)

movables' actuators, landing gears, environmental control, electrics and hydraulics, etc. For the purpose of conceptual/preliminary design they have all been represented as lumped mass collocated in the assumed center of gravity of the system itself (see Fig. 4.9).

The actual mass of the various non structural items was determined by an external module to the MMG (which worked as kind of initiator tool), based on handbook methods and engineering judgment. Since all the masses were estimated by the initiator only once for the reference aircraft configuration, rules were implemented in MMG to scale the masses of items such as de-icing systems and movables' actuators with the actual size of the movables and the length of the various lifting surfaces leading edge. In fact, during the aircraft optimization the number and size of the movables, as well as the wing span and sweep angle were varied, hence updated values of the abovementioned masses were required.

The distribution of the various NSMs was required not only by the weight&balance discipline, but also for an accurate structural analysis. Therefore, apart from some simple parametrical rules to *define the position* of the various NSMs with respect to the overall aircraft configuration (or the position of certain aircraft components) (La Rocca and van Tooren, 2002a; La Rocca et al., 2002), other more complex rules have been defined to establish the *NSMs/structure connectivity,* i.e., what NSM item is attached to which aircraft structural components. The former rules allow the MMG to generate a formatted table (see example in Table 4.1) containing mass value and c.g.'s coordinates of each NSM item, whereas the latter allow the generation of the NSMs connectivity information, required for the preparation of FE models. This information is stored inside a dedicated attribute of the various aircraft structural elements. In other words, each structure element "is informed" about the specific NSMs items it has to support. During the project, a system was implemented to harvest this information from the product model and to use it to automate the generation of connection elements (RBE) linking all the NSM with their supporting structure elements, directly inside the FEA environment (La Rocca and van Tooren, 2002c; Pearson, 2001). Fig. 4.10 shows an example of connectivity between the

wing back spar elements and the actuation system of the various trailing edge movables.
Eventually, the positioning and connectivity rules mentioned above guarantee an *associative link* between the NSMs model and the main structural model. Any change, either in the shape of the aircraft or *in the number* and positioning of the structure components, would yield an updated table of masses and c.g.s' positions, as well as a new consistent connectivity definition. Indeed, in the case a spar or a rib is removed, the connectivity rules will enforce other contiguous structural elements to take over the NSMs supporting role.

NSM item	Mass_(kg)	X_cg	Y_cg	Z_cg
GROUP_FUSELAGE_(left_half)				
TED_1_CTRL	153.9	41992.3	-1505.9	1320.2
TED_2_CTRL	248.4	41992.3	-5515.3	1712.2
TED_3_CTRL	225.5	41992.3	-10399.1	2189.8
DE-ICE_(fus)	59.0	19169.6	-9570.2	736.4
FUS_FUEL_SYS	124.4	29900.0	-3263.2	-1551.5
COCKPIT_ITEMS	929.6	1750.0	-553.8	-415.5
ELEC	586.0	27851.6	-5615.9	1171.8
APU	250.0	41992.3	0.0	1173.0
CARGO_HAND	3900.0	26402.8	-5615.9	1171.8
AIRCO	250.0	33600.0	-7501.0	0.0
HYDR_PNEU	387.0	33984.0	-5746.0	146.8
GROUP_WING_(left_half)				
TED_4_CTRL	308.8	41153.7	-15152.8	2445.1
TED_5_CTRL	293.1	41006.5	-20403.2	2592.4
TED_6_CTRL	286.1	44186.5	-27226.0	3143.8
TED_7_CTRL	159.9	48467.8	-34037.2	3901.6
DE-ICE_(iw-ins)	61.6	28040.9	-15277.8	1235.7
DE-ICE_(iw-out)	82.8	32617.9	-20487.7	1774.6
DE-ICE_(ow)	209.3	41907.8	-31062.2	3067.5
WING_FUEL_SYS	420.5	36135.0	-17438.1	2064.3
WING_TRAP_FUEL	200.0	33870.1	-12984.1	1697.4
WING_INST_ELEC_HYDR	322.0	39447.5	-23433.2	2493.1
GROUP_WINGLET_(left_half)				
TED_8_CTRL	208.0	51659.2	-39518.4	7510.3
DE-ICE_(wl)	40.0	50472.0	-39641.2	7488.6
GROUP_PROPULSION_(left_half)				
MID_ENG	4655.7	43758.0	0.0	4142.9
MID_ENG_STR	470.0	43758.0	0.0	2185.7
SIDE_ENG	9311.3	39750.0	-7501.0	5411.0
SIDE_ENG_STR	940.0	39750.0	-7501.0	3453.7
GROUP_LANDING_GEARS_RETRACTED_(left_half)				

Table 4.1: mass values and c.g.'s position of non-structural items of the MOB BWB (La Rocca and van Tooren, 2002a).

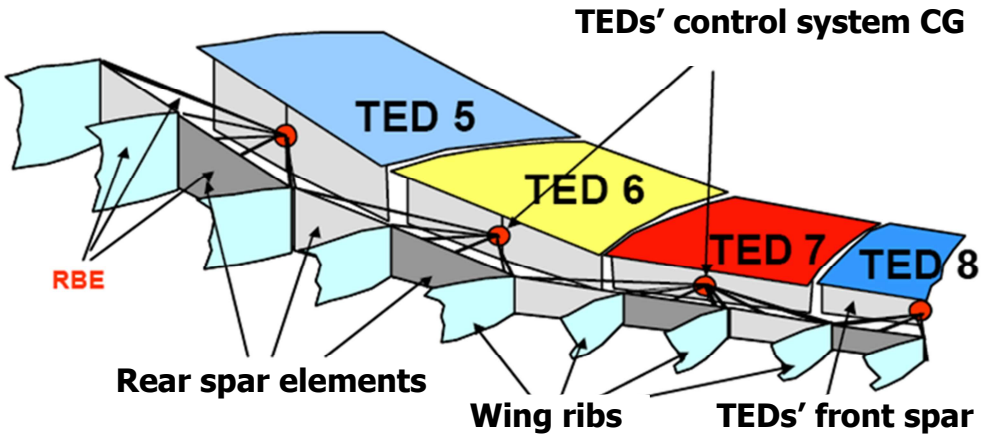


Fig. 4.10: Example of the non structural masses modeling approach. The control systems of the wing trailing edge devices (TEDs) are represented by lumped masses, located in the systems' centers of gravity. Information is generated concerning the connectivity between the lumped masses and relative support structure elements.

4.6 HLPs definition: an heuristic approach

As elaborated in Sections 4.3 and 4.4, the definition of a High Level Primitive is the end result of an abstraction process. Every modeling activity is actually the result of an abstraction process, during which the modeler decides what to include and what to leave out from the model; what are the relevant object's properties and operations to be captured and which are not interesting for the use at hand.
Quite a deal of subjectivity is involved in this process, so that there is not a single, correct way to define a high level primitive such as the wing-part or the fuselage-part addressed before. In the end, the use of different rules and parameterization approaches can yield similar levels of modeling flexibility and capability. As general guideline, the development process of HLPs-like elements and the overall MMG system must be always *user oriented*. A model cannot be judged good in an absolute sense, but it is good if it can fulfill the designers' needs during the design and analysis approach at hand; i.e., if it can fulfill the defined *use case*. Although this might sound obvious, many are the software tools that have been developed upon a wrong use case (or without any use case study at all).
In the next subsections, some other guidelines are given, which should provide the minimum common denominator to any possible parameterization approach.

4.6.1 Parameters and variables

Before discussing about the parameterization approach of the MMG, a clarification is due: whilst in optimization there is a clear distinction between *parameters* and

variables, in the development of a parametrical model such as the MMG, this distinction is not applicable. Each attribute that designers can *explicitly* change to affect shape, configuration or some other information related to the product model is addressed here as parameter. Only during the set up of an optimization process, it will be decided which of these parameters will be selected as variable for the optimizer, and which will be kept as parameter for the traditional parametric studies. It follows that the use of a large number of parameters can improve the flexibility of the model, without necessarily making the model unsuitable for optimization.

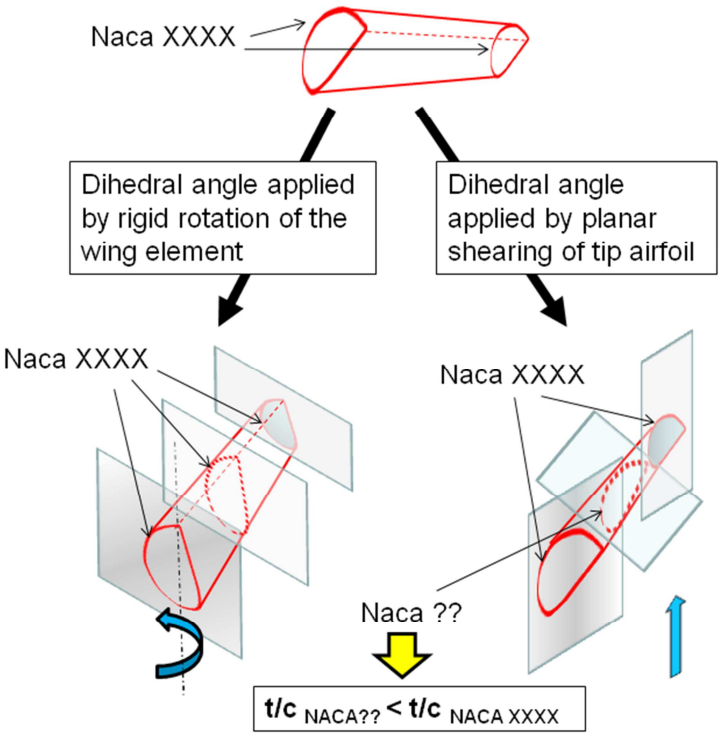


Fig. 4.11: Different approaches to apply the dihedral angle to a wing element. *Bottom-left*) correct approach: wing thickness and dihedral angle uncoupled. *Bottom-right*) incorrect approach: increasing dihedral angle values lower the wing thickness.

4.6.2 Ambiguous parameter definition

All parameters should be defined in a way to avoid any ambiguity, which could lead to over/under constraining of the geometry. For example, if the airfoil thickness (i.e. the wing depth) and the dihedral angle are both selected as parameters to define the wing-part geometry, the variation of the dihedral angle value should not affect the value assigned to the airfoils' thickness. Fig. 4.11 shows the consequences of enforcing the dihedral by shearing the wing sections rather than applying a rigid rotation to the whole wing: the wing depth is affected (decreased), even though the airfoil thickness is not explicitly varied by the user.

4.6.3 Parameter definition for spatial integration and robustness

Parameters should be defined in order to guarantee spatial integration and robustness. A bad choice of parameters, possibly not supplemented by appropriate constraining rules, cannot prevent an optimizer generating unfeasible aircraft geometries (Vandenbrande et al., 2006) such as the wing planform shown in Fig.

4.12-bottom, or configurations with disconnected or self penetrating components, as illustrated in Fig. 4.12-top. This might lead either to parametric models that break and generate errors (hence making the model generator crash) or to a waste of computational resources when unfeasible configurations get analyzed anyhow.

It should be considered that, during optimization studies, it is not always possible (neither convenient in terms of efficiency) to visually inspect the models generated at each cycle. Hence precautions must be taken to guarantee the quality of all the generated models. A sound parameterization can improve the robustness of the model and guarantee spatial integration, as well as simplify the set up of the optimization problem. For example, the longitudinal position of wing and tail empennages can be better indicated as a percentage of the total fuselage length, rather than as an absolute coordinate value. The position of a rib can be better indicated relatively to the length of a reference spar or to the wing/empennage span. For example, a parameter value ranging from 0 to 1 can be used for rib positioning, where the zero value indicate a root placement and 1 a tip placement. Whatever the length of the given wing, a feasible placing of the rib is guaranteed for all the possible values of the rib positioning parameter.

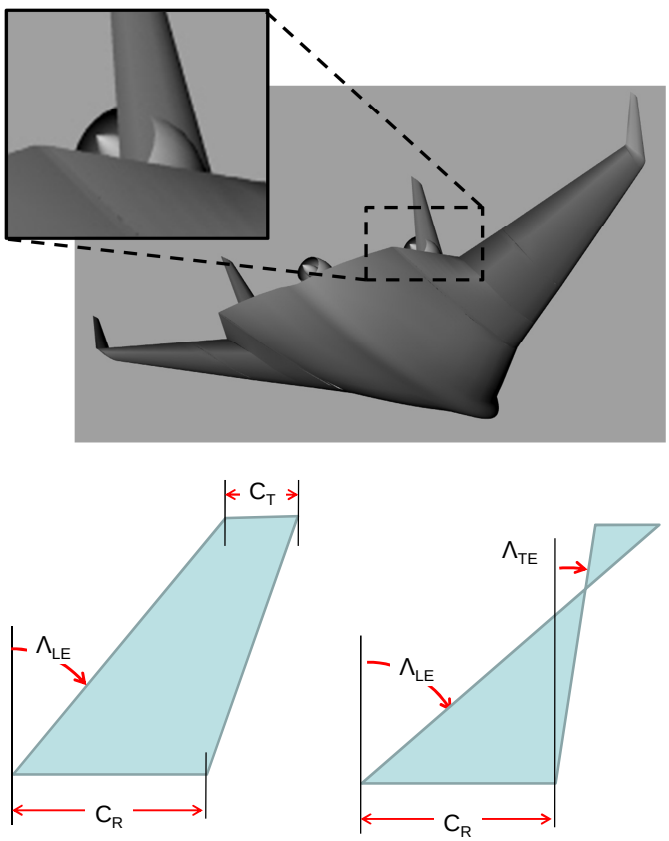


Fig. 4.12: wrong parameterization might cause problem of spatial integration. *Top)* the parameter determining the engine lateral position is not constrained with respect to the fin position, hence fin/nacelle penetration can occur during engine installation study. *Bottom-right)* unfeasible wing planform generation allowed by bad parameters selection (and uncontrolled variation)

In alternative, rules could be added to the HLPs definition, which skip the generation of badly defined components (hence preventing the generation of intersection errors), or overrule some user defined parameter values with valid default values. For instance, some rules could be implemented (Chapter 3, section 3.8.3) to measure the minimum distance between the surfaces of the engine nacelle and

vertical tail surface (Fig. 4.12), and automatically overrule a badly defined engine positioning with a predefined minimum clearance value.
If the robustness of the model is not guaranteed at modeling level, then extra constrains and special bounds must be added to the formulation of the optimization problem, which is generally neither trivial nor efficient.

4.6.4 Modeling aspects decoupling

Unnecessary coupling between different modeling aspects of the primitives should be avoided (unless really necessary). For example, a design rule that defines the winglet *surface* positioning at a certain distance from the wing spars' edges would force the product model to generate first the wing surface, then the wing structure and, once the position of the wing spars is known, then the winglet surface generation could finally take place. This approach triggers a chain of avoidable computations (i.e., those related to the wing structure instantiation) and prevents the possible instantiation of the complete aerodynamic surface model until also the structure model is fully developed. Indeed, it should be possible to evaluate the aerodynamic functionality of a winglet without the need to have predefined any wing structural model. Eventually, this unnecessary coupling has effect on the robustness and efficiency of the parametric model, as well as on the flexibility and maintainability of the overall modeling system.

4.6.5 A sense of familiarity…

Parameters should have a clear meaning to the different users/customers of the model (aerodynamicists, structure designers etc.). The HLPs should result familiar objects to the user, hence they should capture the "natural view" of the user on the product at hand. For instance, aerodynamicists rather manipulate airfoils than clouds of points to interpolate a wing surface; they define the wing sweep angle between a certain wing axis and the lateral aircraft axis, not the longitudinal. Eventually, compliance with designers' conventions has also implications on parameters naming. For example, the designer might not be familiar with a parameter defined as "wing-edge-curves-distance" but he is familiar with the concept of wing span; "edge curves relative rotation" does not sound as familiar as wing twist angle, etc.

4.6.6 Parameters = degrees of freedom of the model

The parameters somehow represent the *degrees of freedom of the model*. On their selection it depends how far and to what detail the user can affect the geometry of the model. In case the designer is simply interested in the manipulation of a wing planform, a parametric model based on very few parameters (span, chord lengths and sweep) will do the job. However, this model will never be able to deliver other

than trapezoidal shapes. It will be inadequate for more advanced aerodynamic studies, where the designer wants to investigate the effect of different airfoils, twist distribution, curved and cranked edges, etc. A different and richer parameterization will be required to improve the model flexibility, i.e., extend its degrees of modeling freedom. The challenge is to allow detailed control of the model shape, without hampering the designer with the definition of too many parameter values. One could think of a parameterization approach (similar to the configuration opportunities offered by many software tools of daily use), where advanced use parameters can be accessed and adjusted only if required, while a limited number of parameters should be sufficient for the most common and less detailed model transformations.
Eventually the width of the typicality range discussed in section 4.3, depends on the number and type of parameters used to define a primitive. Typically, the larger the amount of parameters, the more the possibilities to stretch, morph and adapt a HLP to match also less conventional shapes, hence the less amount of primitives required to model an entire aircraft configuration.
Optimization studies of models defined using many parameters do not require any more computations than simple models defined with very few parameters, as far as the number of parameters used as optimization variables is the same.

4.7 From the aircraft geometry model to the abstractions for multidisciplinary analysis. Role and definition of the Capability Modules

So far the definition and the use of the HLPs to generate very different aircraft configurations and configurations' variants have been described. It has been discussed how KBE technology can capture the generative process of a complex *geometry product* and speed up the transformation process of a conceptual idea into a *geometry model*. As mentioned in Section 4.2, apart from the use of recording and sharing, the availability of such a model is required to initiate the aircraft multidisciplinary analysis and verification phase. The challenges associated to the generation – as far as possible *automated* – of consistent and synchronized models for the discipline specialists' analysis tools (both high and low fidelity; off the shelf and in house developed) have been addressed in Chapter 2.

In this section, the special capabilities that have been implemented in the MMG to address those challenges will be described. Again, as in the case of the HLPs concept, the *engineer has been taken as role model*, and KBE technology has provided the means to capture some of his/her abilities in the rules of the product model. In particular, the conceptual development of methods to automate the generation of different abstractions of the master geometry model, for a range of

disciplines is discussed in this section, while some of the implementation details will be covered more extensively in Chapter 6.

4.7.1 Capability Modules to capture procedural knowledge

The mental process occurring in the head of designers, from the functional thinking to the conception of a product configuration, can only be intuited. Indeed, the development of the HLP concept is just an attempt to map part of that process into a software application. However, the way designers process the general purpose CAD model of an aircraft into a dedicated model that is suitable for Nastran, Fluent or some other analysis tool, is much more explicit and, *relatively*, easier to understand and formalize. By means of interviews, direct observation and other dedicated knowledge acquisition techniques (Milton, 2007; Rhem, 2006) it is generally possible to elicit the specialists' working practice, and the tips and tricks used to prepare models that are suitable for their analysis tools.

Indeed, this preparation work to transform a general purpose CAD model into a set of model abstractions, either ready for the solver of a given analysis tool, or at least for its preprocessor (see Fig. 2.15, chapter 2), can be quite complex and laborious, in particular for high fidelity analysis tools. However, there is hope:

- Quite independently from the aircraft configuration at hand, no matter if the traditional airliner or the blended wing body aircraft of Fig. 4.2, the same analysis tools and preprocessing methods are generally used by specialists.
- A large part of above mentioned preprocessing activities are systematic and repetitive, require geometry manipulation and follow known rules

On the base of these two observations, it looks like there are possibilities to generalize many of these preprocessing activities and KBE appears to be the right technology at hand.

In fact, the ICAD programming language has been used to generate a number of so called *Capability Modules* (CMs), which are special classes with the peculiarity of encapsulating just *procedural knowledge*. Similarly to the HLPs, the CMs have been implemented using the *defpart* macros of ICAD (Chapter 3, section 3.6.1). However, on the contrary of the HLPs, the CMs cannot be instantiated into standalone geometric objects, but use the encapsulated procedural knowledge to operate on the HLPs. In other words, they process the geometry and relevant information of the HLPs' instantiations into model abstractions required for the multidisciplinary analysis process.

Fig. 4.13 shows some examples of the preprocessing activities that can be performed by different CMs on a given wing-part instantiation. From top-left, clockwise:

- The outer surface of the wing-part is translated into a set of Cartesian coordinates points, which can be used as grid nodes for discretization (Qin et

al., 2002), or to support the re-splining of the surface into another proprietary modeling system (Laban et al., 2002).

- The outer skin panels and inner structure components of the wing-part are all intersected with each other and transformed in sets of meshable surfaces for FE structural analysis.
- The wing-part fuel tank volume is computed based on the position of the sealing rib and spars
- A system of axis and properly distributed points is generated to support the transformation of shell FE models of the wing part into a condensed mass models (Cerulli et al., 2006)
- The geometry of the 3D wing-part is transformed into a set of 2D coplanar flat panels, e.g., to support simplified aerodynamic and aeroelastic analysis (Stettner and Voss, 2002).

All the operations that a human specialist typically performs on a geometry model to generate the abstractions of Fig. 4.13, (including the logic rules to choose the specific type and the sequence of these operations), have been translated into software rules and algorithms, which constitute the body of several Capability Modules.

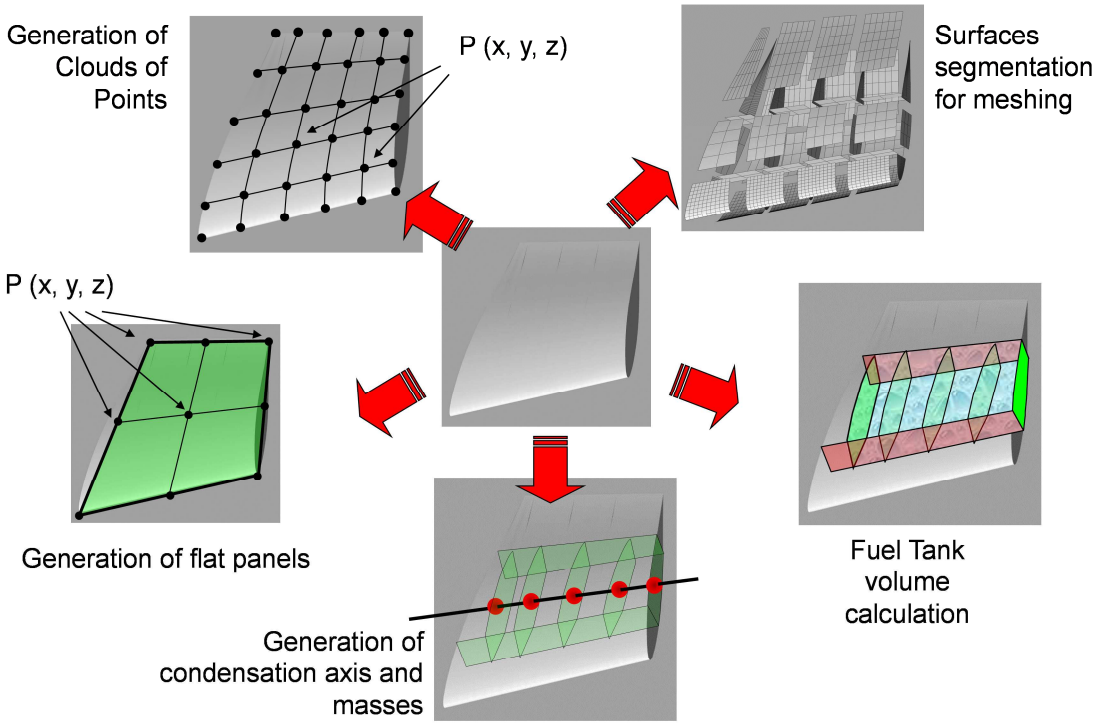


Fig. 4.13: examples of preprocessing of a Wing-part instantiation to support multidisciplinary analysis.

For example, Surface-splitter is the CM responsible to transform the geometry of the HLPs in sets of meshable surfaces for FE analysis; Points-generator is another CM that transforms the outer surface of any HLPs instantiations into sets of points/panels to support the generation of aerodynamic models (see Chapter 6 for the specific details of these two CMs).

When a Capability Module class is *linked* to a HLP class, the procedural knowledge encapsulated in the CM, becomes immediately available to that HLP. In other words the given HLP *acquires the capability* to transform itself - *automatically* - in meshable surfaces, clouds of points, etc. Hence, the name Capability Modules…

4.7.2 Linking HLPs and CMs

A HLP can acquire different capabilities via *the links* to different capability modules. Also, the same capability module can be linked to different HLPs (Fig. 4.14)
The way these links have been realized is by the exploitation of the object oriented paradigm supported by the ICAD KBE system. As a matter of fact, two different HLP/CM link approaches have been used.

Approach 1: CMs included in the *mixin* list of a given HLP defpart

By using this approach, (refer to Fig. 4.15 for the UML class diagram and examples of HLP and CM defpart definitions) all the attributes and operations of the CM are directly inherited by the given HLP class, which means the operations defined inside the CM can be applied directly to the HLP's attributes and children, as they were defined directly inside the HLP. As a consequence, the given HLP will be able to answer new messages, because of the operations defined in the CM. For example, "generate a cloud of points", "compute fuel tank volume", or "compute the total-surface", as in the fictive case of Fig. 4.15. On the other hand, an isolated CM could not work as standalone because it would not have any knowledge about the surface to transform into a cloud of points, or about the fuel tank geometry whose volume must be computed. In the example of Fig. 4.15, `CM-2` does not have any internal knowledge of `Child1` and `Child2`, which are defined as components of the class `HLP1`.

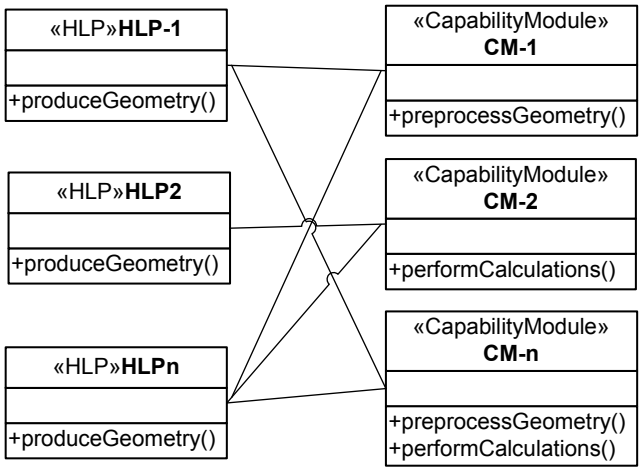


Fig. 4.14 : possible links between HLPs and CMs

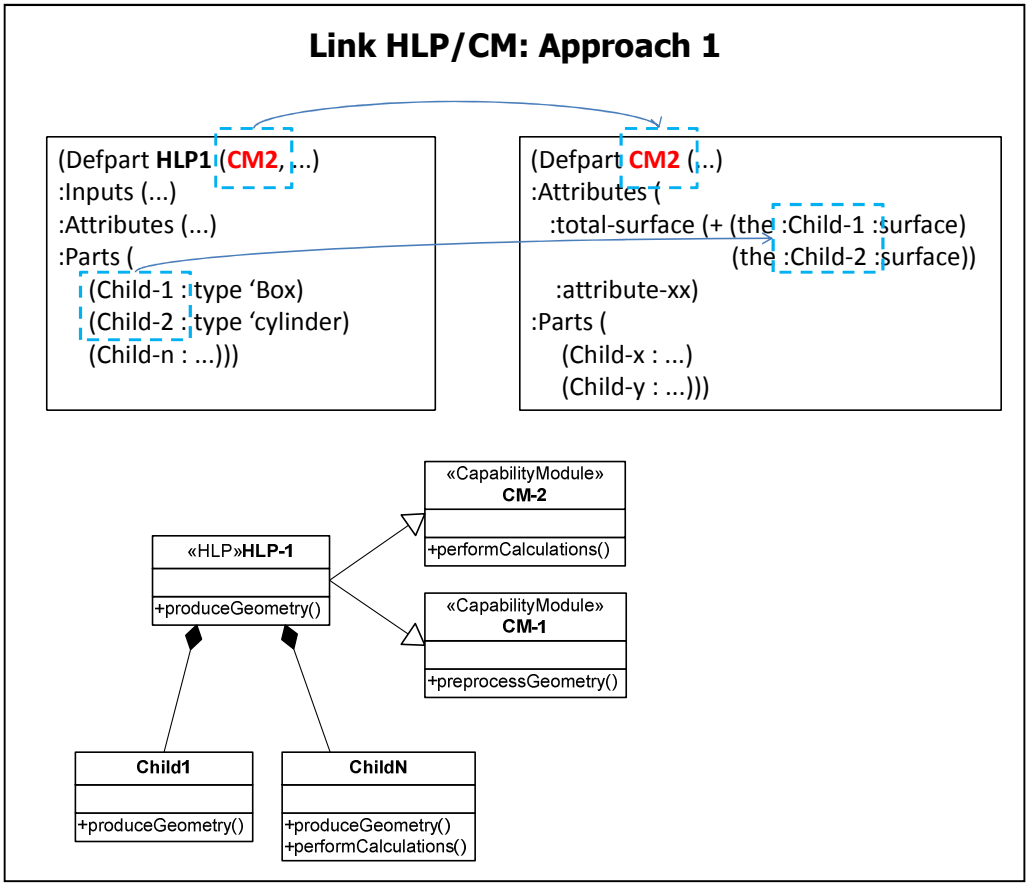


Fig. 4.15: CMs included in the *mixin* list of a given HLP defpart. Examples of HLP and CM defpart definition and UML class diagram

Approach 2: HLP's children defined as specializations of a CM classes

Being a HLP's child a specialization of the CM class (refer to Fig. 4.16 for the UML class diagram and the examples of a HLP and CM defpart definitions), the former will be able to answer all the messages that the CM class can answer. In the example of Fig. 4.16, it will be possible to send a message to any instantiation of the `Child-3` class, whose internal operations are actually defined inside `CM1`. In order to perform those operations, `CM1` will need to receive as input some data generated within `HLP2`. In this example `CM1` receives via `attribute-1`, the surface of `HLP2`'s `Child-1`. Without providing this input values, it would not be possible to instantiate `CM1` and use it as standalone.

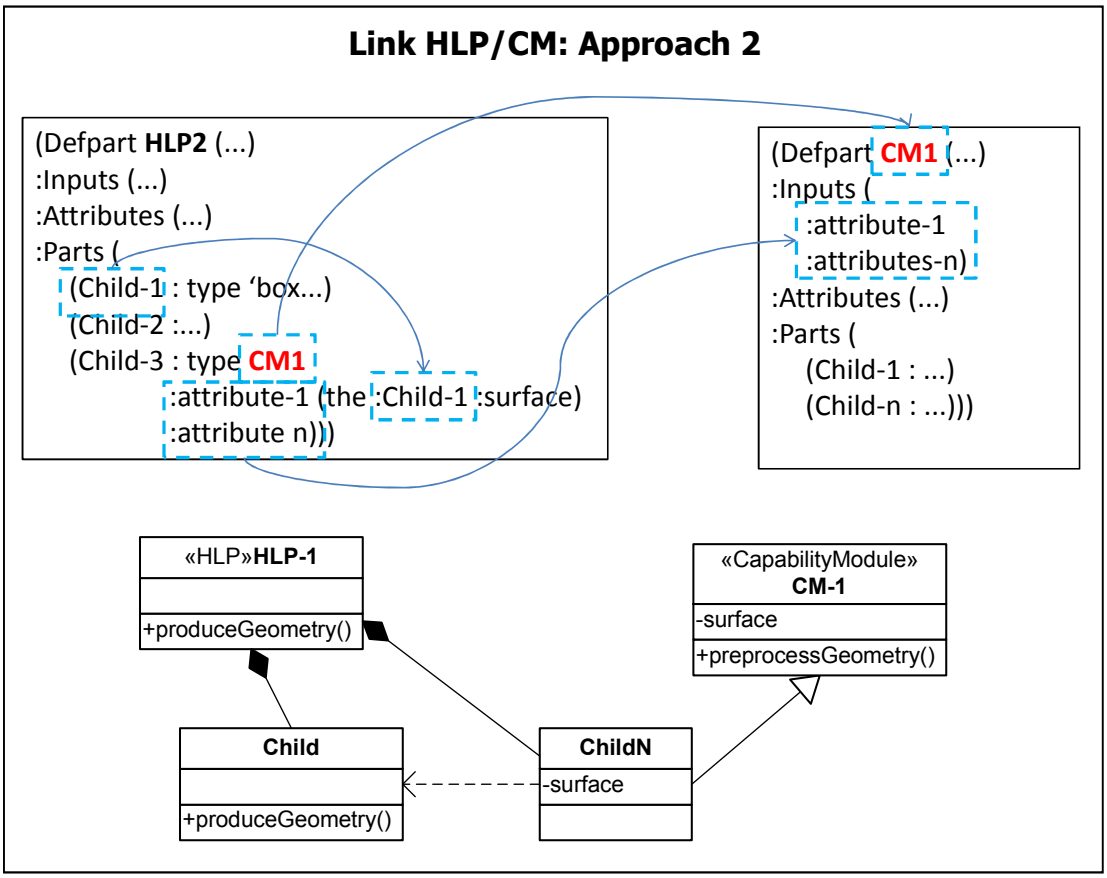


Fig. 4.16: HLP's children defined as specializations of CM classes. Examples of HLP and CM defpart definitions and UML class diagram.

4.8 Automatic generation of aircraft model abstractions

Because of the supported object oriented paradigm, once a HLP has acquired a certain capability (due to the link to some CM), also *all the specializations* of that primitive, automatically, inherit that capability. This has a relevant consequence at overall MMG capability. In fact, the automatic preprocessing capabilities developed at the single HLP level get automatically propagated to the overall aircraft model level.
For example, being the Wing-part and the Connection-element HLPs linked to the Points-generator CM, the whole blended wing body aircraft of Fig. 4.2 (including the center part, all the wing sections, winglet, fins and all the connection fairings) can be automatically transformed into sets of points to support aerodynamic analysis.
At the root level of the BWB product model, messages can be sent to all the various HLPs instantiations asking for the generation of clouds of points. Then all the "partial clouds" can be *collected and merged* into a "global cloud" and, finally, delivered as MMG output files. Special capability modules have been defined just for this purpose, as described in the following subsection.

4.8.1 Scanners and Report Writers

Some dedicated capability modules have been developed in the MMG, whose function is to send messages to the class instances in the aircraft product model, collect the results and output them in a given output file data format. These modules work as a kind of scanners, which parse the whole product tree (or specific branches of) searching for specific *objects* or *objects' attributes*.

For example, *scanner modules* have been defined to collect all the wet-surfaces in the product tree, or all the meshable surfaces generated by the Surface-splitter CM, or the values of the partial fuel tank volume computed for the various wing sections, or the lists of points coordinates generated by Points-Generator CM.

Often a tagging system has been implemented to facilitate the tree scan. That is to say, a dedicated attribute is automatically assigned to the objects that need to be identified in the tree. For example, the attribute "I am a wetted surface" is automatically assigned to all the outer surfaces of the HLPs and the attribute "I am a meshable structure element" is automatically assigned to all the surface segments generated by Surface-splitter. The scanner just searches for all the objects in the product tree that have that identification attribute (i.e., that tag) and collect either the object or some object's attribute, as needed.

The functionality of the various product tree scanners kind of mimic the different engineering views that different discipline specialists have on the same product. The aerodynamic specialists look at the overall aircraft model, but are only interested in the aircraft wetted surfaces, the structure specialists "filter out" the aerodynamic features and focus on the configuration of the structural components, etc. The scanner-CMs, eventually, apply the same model abstraction process that is practiced by human specialists.

Once the required objects and/or data have been collected, they can be organized and exported as MMG's output files. In ICAD parlance, these output files are called *reports*, because generated by special functions called *report writers*. A number of report writers are ready available in ICAD for the generation of IGES, STEP, and other standard format files. Furthermore, other customized report writers have been programmed in the MMG to write reports in diverse customer/tool specific formats (e.g., specifically formatted ASCII tables, or XML files, for which no standard report writer was available in ICAD). Eventually, this is a powerful feature that allows the MMG communicating with a very broad range of external tools, both in-house developed and commercial of-the-shelf. Fig. 4.17 provides a sketch of the scanning and collecting procedure acting on the product tree of the MOB blended wing body aircraft: the surfaces of the various structural components are collected, and exported via sets of dedicated IGES files (La Rocca et al., 2002).

In Chapter 6, some MMG study cases will be discussed, showing the use of the HLP and CM approach discussed so far, to model different aircraft configurations and

automate the preprocessing work of dedicate model abstractions for high and low fidelity analysis tools, both in-house developed and commercial of the shelf.

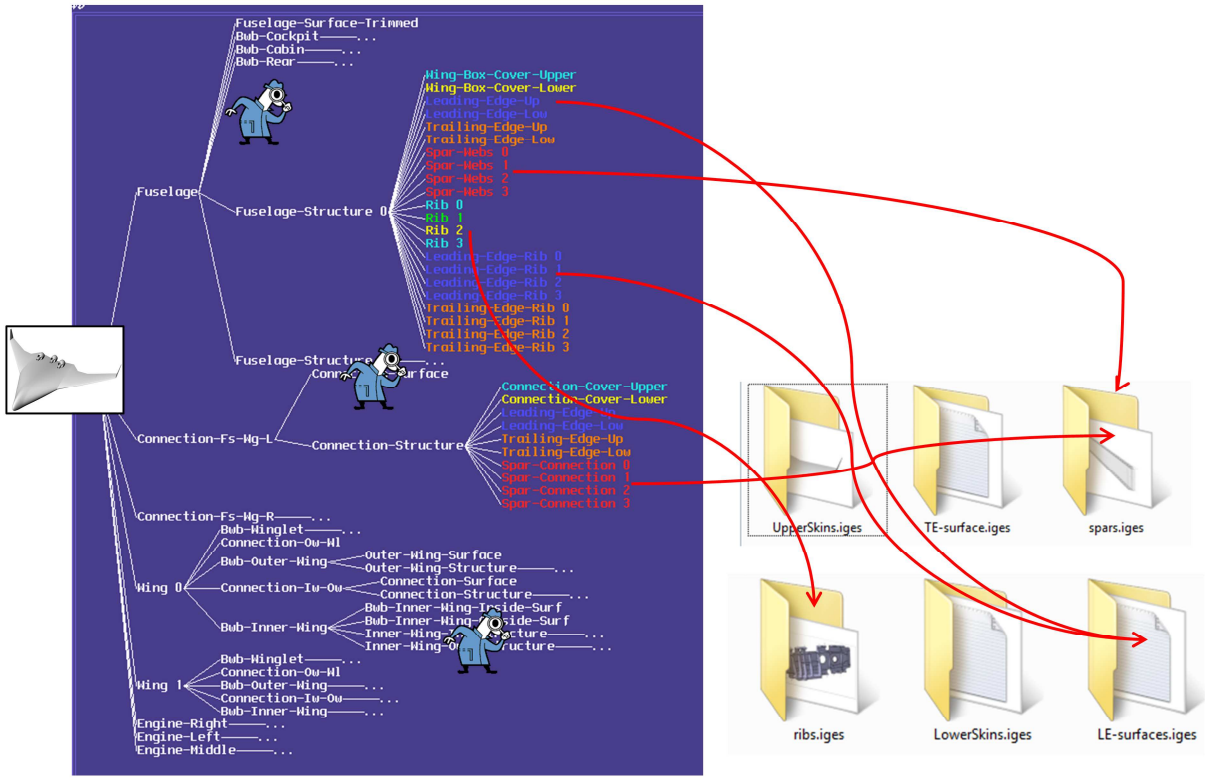


Fig. 4.17: A Scanner capability module parses the aircraft product tree, collects the geometry of the structural components and exports them as sets of IGES files.

4.9 The MMG architecture: flexibility through modularity

The purpose of the large class diagram of Fig. 4.18 is to give evidence of the strong modularity of the MMG architecture (refer to Appendix A for a summary of the UML grammar). The diagram is limited to the structure of the lifting surface aggregation to guarantee readability in this format. The diagram can be actually considered a close up of the one shown in Fig. 4.4. Further detailed views will be provided for discussion in Chapter 5.

The diagram shows that a generic lifting surface can be modeled as an aggregation of Wing-part and Connection-element instances. Also Winglets and movables can be modeled using the Wing-part HLP.

Fig. 4.18 reveals that the HLPs themselves, described so far as kind of monolithic elements, are actually aggregations of separate modules (classes), such as the one responsible for modeling the external aerodynamic surface and the other for the inner structure. The structure-dedicated model itself is defined as an aggregation of more classes. For instance, `Spar`, `Rib` and `Skin` are components of `WingTrunkStructure`.

The diagram shows also the links between some of the CMs introduced in Section 4.7.1 and the (components of) HLPs. For example, `SurfaceSplitter`[3], which is responsible of transforming the wing-part geometry into sets of meshable surfaces for FE analysis, is linked to `Skin`, `Rib` and `Spar`. `PointsGenerator`, which is responsible to transform the aerodynamic surfaces of all wing-parts and connection-elements into clouds of points for aerodynamic analysis, is linked to `WingTrunkSurface` and `ConnectionSurface`.

The modular architecture of the MMG represents a key factor for the flexibility, maintainability and scalability of the overall system. New HLPs and CMs can be added and removed with relative ease, in order to tailor the MMG to the problem at hand. Indeed, the MMG can be used both for the design of a complete aircraft, as well as for the design of a single aircraft (sub)systems, e.g., a fin, a canard, or a complete tail configuration. The designer can decide to instantiate just a branch of the product model, without the need to build a new KBE application. Furthermore, when new disciplinary analysis tools will have to be plugged in the DEE system, new CMs can be developed to generate the new required abstractions of the same HLP-based aircraft model.

Also the single components of the very MMG building blocks, the HLPs, can be improved or rebuilt without affecting the code of the entire primitive. For example, an improved or alternative method to generate the external aerodynamic surface can be defined, without affecting the way the aircraft structure is modeled. Vice versa, new structure models can be developed, based on the same aerodynamic surface to be used as mould line.

[3] Similar to what discussed for HLPs, also for the capability modules, different fonts are used to distinguish a CM (the concept) from its software implementation as a class. (e.g. `SurfaceSplitter` is the class implementing the Surface-splitter CM)

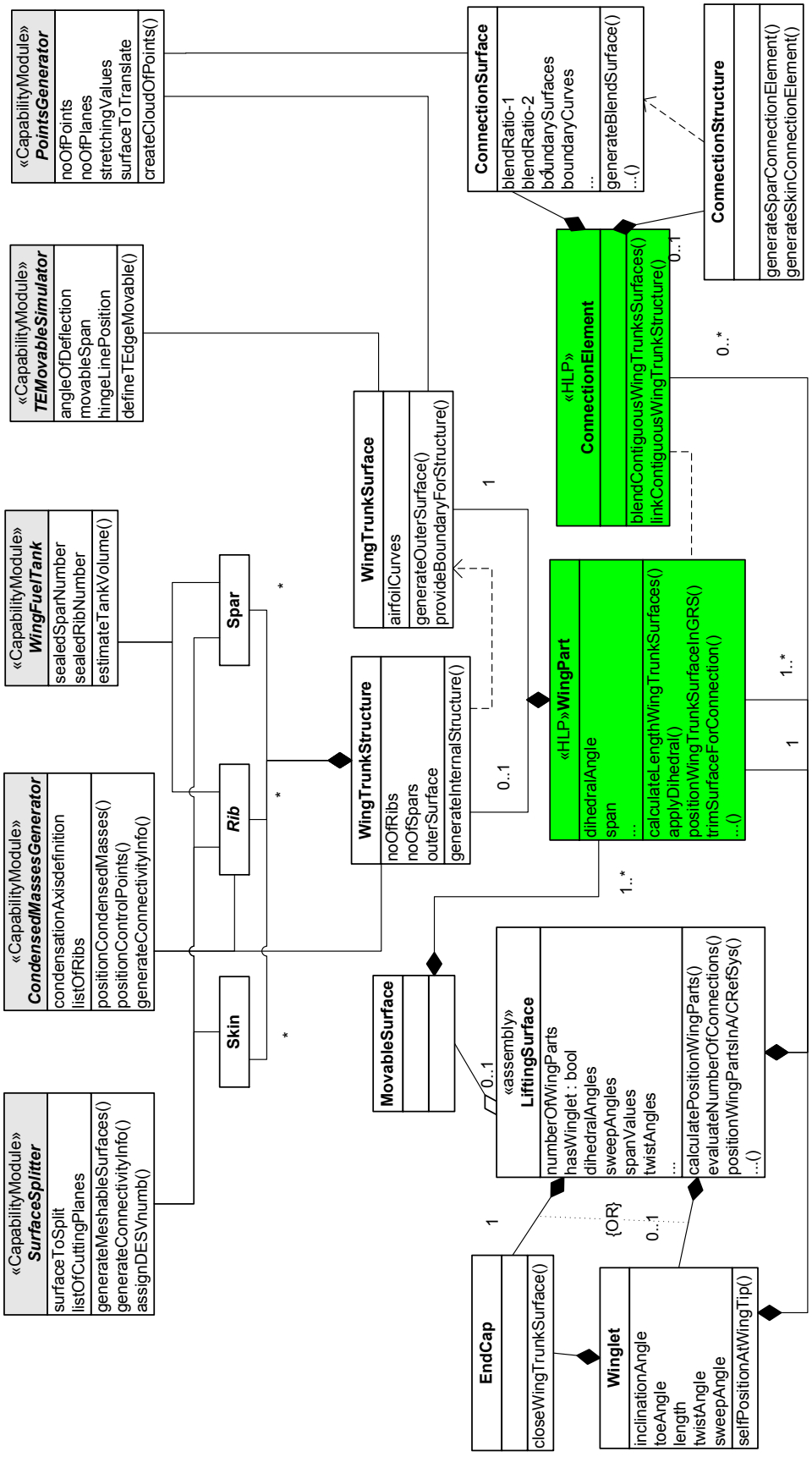
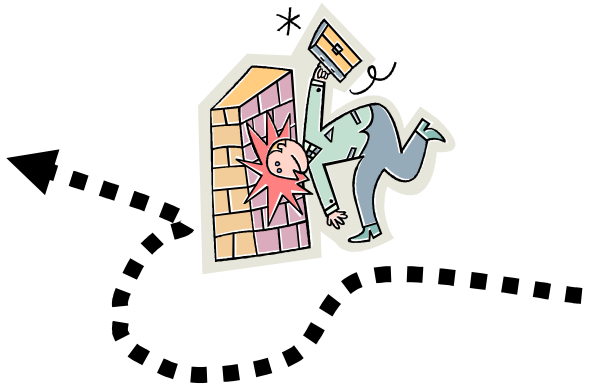


Fig. 4.18: UML class diagram showing the modular structure of the MMG: detail of the Wing-part and Connection-Element HLPs architecture and links with the Capability Modules.

Also in terms of code development management, the modular approach has advantages. In fact, the code responsible for structure modeling can be developed independently from the aerodynamic functionality, which means that different developers can work on the two aspects separately and not necessarily at the same time. If the structural model is still incomplete or has bugs, the use of the aerodynamic modeling features can still be fully exploited. Indeed the user is not even required to fill the MMG input files parts relative to the structure definition, if no structure related class needs to be instantiated.
As far as the links to pass parameters values down the various hierarchies of the product model are properly maintained, new functional blocks can be modified, moved, added and bolted on each other, still maintaining a sustainable level of code "*spaghetti-ness*".

4.10 Dealing with CAD engine limitations: capturing workarounds for robust modeling



It is acknowledged that the development of a robust parametric geometry model for MDO is sometime closer to art than science (Carty and Davies, 2004 ; Bowcutt, 2003; Vandenbrande et al., 2006; Staubach, 2003). Indeed, the model will have to survive all the possible parameter variations imposed by the optimizer, which can possibly address also the topology of the given aircraft configuration (e.g., some extra rib in a wing, one spar less, a different tail configuration, etc.).
As discussed previously in this chapter, a correct parameterization is the necessary starting point to build a robust model.
Unfortunately it is not always sufficient: when dealing with geometry processing, errors can still occur because of missed or inaccurate intersections, failed operations, etc. One single failed operation might just stop the whole generative process or produce a wrong analysis model, which would very difficult to spot during an optimization process.
Considering that preprocessing a complete aircraft geometry model into sets of meshable surfaces for FE analysis requires thousands of intersection operations between surfaces of relative complexity, it is clear that robustness is a major issue.
CAD experts classify parametric assembly failures in three major categories (Staubach, 2003):

- Class I: errors due to over/under constrained geometry models.
- Class II: intersection errors due to invalid parameters range (e.g., when attempting to cut a surface with a non intersecting plane) or missing relationships (e.g., when attempting to use the result of the failed intersection mentioned above to perform some other operation). See section 4.6.3 on invalid parameters range or missing relationships.
- Class III: errors generated by native CAD kernel bugs, or inherent accuracy limitations (e.g., in the mathematical model to compute surface intersections).

Guidelines to avoid Class I and II errors by means of proper parameters definition (possibly supplemented by rules) have been provided in Section 4.6.3. By the way, those guidelines apply to the generation of KBE models as well as to conventional CAD models. However, the greatest obstacle to robustness is represented by Class III failures, which are particularly nasty, because largely out of control of the CAD end-users.

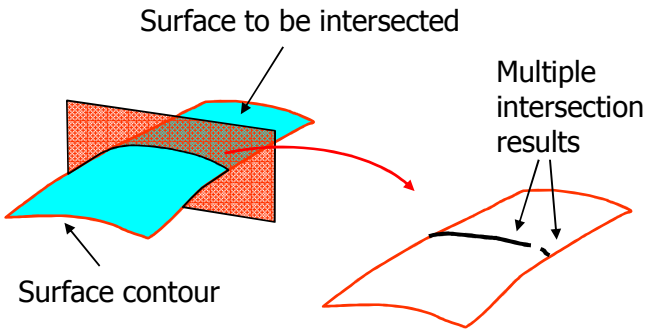
Fortunately, KBE can offer better guarantees in this respect, than conventional CAD. In fact, the possibility to encapsulate rules in the model is the key to robustness. The methods used by CAD specialists to deal with typical model errors, for example, by massaging the geometry and using smart workarounds, indeed, can be largely translated into rules and "taught" to the MMG components.

As a matter of fact, in these years of extensive use of the ICAD system, the author came across a number of limitations of the internal geometry engine, which mainly concern with the surface intersection operations. Once understood the typical occurrence of these errors (see three relevant cases in Fig. 4.19, Fig. 4.20 and Fig. 4.21) the KBE approach has been exploited again: this time to deal with the limitations of the KBE system itself…

Two different approaches have been implemented in the MMG to automatically trigger workaround procedures and significantly enhance its level of robustness:

1. The *proactive approach*: the problem is anticipated and avoided. Knowing a specific limitation, a series of alternative or additional operations is performed, avoiding the use of direct but *known-to-be-unreliable* operations. (See Case I and II in Fig. 4.19 and Fig. 4.20, respectively).
2. The *check&correction* approach: knowing that a given operation might give inaccurate results, a check is performed on the output and, if the result differs from expectations, the operation is repeated using an alternative way. (See Case III in Fig. 4.21)

CASE I: Surface/plane intersection

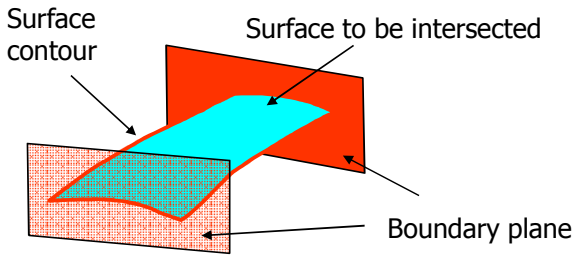


Possible problem:
Intersection operation gives multiple results

Proactive solution:
1. Assume that intersection operations always deliver multiple results
2. Collect results in a list and create a continuous (composed) curve attaching all the collected fragments to each other
3. Use the new composed curve as result from the intersection operation

Fig. 4.19: Use of proactive approach to work around two ICAD Class III errors.

CASE II: Surface/boundary planes intersection

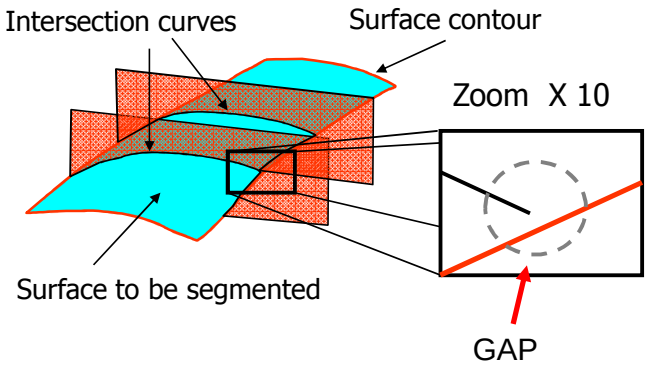


Possible problems:
1. No intersection found (plane misses the surface)
2. Intersection curve is incorrect (one point and …)
3. Intersection operation gives multiple results (See Case I)
4. An error is generated and the program stops immediately

Proactive solution:
1. Detect the boundary planes
2. Decompose the contour of the surface in its curve components
3. Find the closest contour component to the cutting plane
4. Use that contour component as result from the intersection operation (without actually performing any intersection!!)

Fig. 4.20: Use of proactive approach to work around ICAD Class III errors.

CASE III: Surface segmentation with a set of intersection curves



Possible problem:
Not all segments found because of some intersection curves not snapping to surface contour

Check & Correct:
1. Count *N_cutting_curves* and *N_of_surface_segments*
2. If *N_of_surface_segments ≠ (N_cutting_curves + 1)* then EXTEND all cutting curves
3. Repeat surface segmentation with EXTENDED curves

Fig. 4.21: Use of check&correct approach to work around ICAD Class III errors.

4.10.1 Product, process and… implementation knowledge

On the light of the previous discussion on HLPs and CMs and the last one on capturing workarounds, it can be concluded that the KBE approach can be used for the followings:

1. to capture the knowledge required to model the geometry of products,
2. to capture the knowledge to process such geometry and build dedicated abstractions for the analysis tools,
3. to capture the knowledge required to guarantee the correct and robust exploitation of the knowledge captured at point 1 and 2, when using a certain KBE system.

The first two types of knowledge are actually independent from any KBE system; thereby, they are always valid and reusable/transferable in/to any KBE system. On the contrary, the third type of knowledge is strictly dependent on the capabilities and limitations of the KBE system employed for the implementation of the first two types of captured knowledge. We address this one as *implementation knowledge*.

Although key for the success of the KBE generative modeling, implementation knowledge is strictly system dependent and loses value when migrating to a different KBE system. The problem is that implementation knowledge is strictly intertwined with the product and process knowledge; as such, it might block the development of

any translator to transfer KBE applications from one KBE platform to another (similar to the way of transferring geometry files between different CAD systems, using standard exchange data formats). See more about this issue in Chapter 7, Section 7.7.

4.11 Discussion

4.11.1 Separation of declarative and procedural knowledge

In Chapter 3, when discussing the major differences between traditional rule based systems and other object oriented systems like frame based and knowledge based engineering systems, one of the highlighted differences was the crisp separation between rules and inference mechanism typical of the former. On the other hand, the object oriented approach is based on the concept of the class, which is a structure where both *declarative knowledge* (the class' attributes) and *procedural knowledge* (the class' operations) are merged. With this respect, the definition of HLPs and CMs is an attempt to bring back some separation between the knowledge about the product (*mostly* contained in the HLPs) and the knowledge about what to do with the product (*mostly* contained in the CMS). Apart from the advantages of this modular approach in terms of system flexibility, as discussed in section 4.9, conceptual clarity and structure follow as well.
To summarize the extensive description of the HLP and CM characteristics, the follow definitions are given below:

*A **High Level Primitive (HLP)** is a KBE artifact that contains both declarative and procedural knowledge, where the latter consists mostly of the specific operations to generate the geometry of the given HLP instantiations. The encapsulated knowledge is different and specific for each HLP and not shared/reusable by other primitives.*

*A **Capability Module (CM)** is a KBE artifact that contains mainly procedural knowledge, which is not specific to any HLP, but is devised to provide methods to more HLPs. A CM cannot generally function as standalone object, is not able to autonomously generate any geometrical entity and cannot answer any message, unless linked to a HLP.*

4.11.2 Render unto designers the things which are designers'

The HLPs and CMs cannot in any way substitute the designer in his decision making activity. Despite their generative and operative capabilities, these software components do not have any knowledge to judge the quality and the pertinence of the data received as input by the designer (apart from eventually checking their

compliance with the expected data format). HLPs and CMs are means to create a *flexible and robust* modeling environment, which is supposed to be capable of delivering valid output, whatever is the received input. It is the designer's responsibility to decide upon the configuration to be modeled and to judge the quality of the final design through his/her knowledge and with support of analysis tools. Though HLPs and CMs do not have any direct influence in steering the design toward certain directions, they definitely put the designer in a more favorable condition to explore the design space.

4.11.3 Exploitation of KBE technology in the initial design phase

Within large aircraft companies, like Boeing, Lockheed Martin and Airbus, KBE is already a mainstream technology since years. However, so far, its application has taken place mostly in the *detail design phase* of structural components and subsystems, as illustrated in Fig. 4.22 from (Mohaghegh, 2004). On the other hand, this research work proposes a possible use of KBE in the earlier *conceptual and preliminary phases* of the aircraft design process, where the configuration of the vehicle is not yet frozen and can still be influenced by all the disciplines.
Various examples of KBE applications developed to support the design process of complex products (not only aircraft) have been found in literature, mainly addressing the detail design phase, where a full generative approach has been used to deliver completely engineered components (Cooper et al., 2001; Chapman and Pinfold, 2001; Subel, 2002). In other cases (Rondeau et al., 1996; Zweber and Hartong, 1998) demonstrator applications have been developed to support the analysis and optimization process of main aircraft components such as wings. Nothing has been found in literature concerning the use of the KBE approach to support conceptual

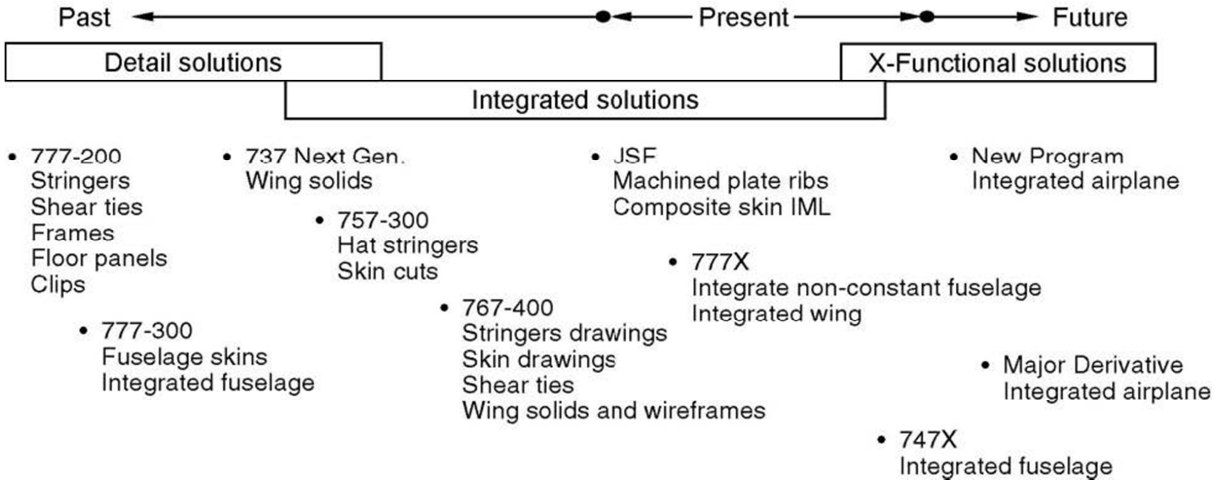


Fig. 4.22: KBE progressive exploitation in Boeing aircraft programs. From structure detailed design to the integrated airplane of the future (Mohaghegh, 2004).

and preliminary design of complete aircraft, *including* non-conventional configurations and the definition of the internal structure, as it is presented in this work.

4.11.4 A different KBE exploitation: parametric modeling vs. integral design approach

In the early days of KBE technology, the development of super integrated design tools, able to cover the whole process from input requirements to the fully engineered product, was proposed (see also fig. 3.13 from (Cooper et al., 2001), in Chapter 3). On the contrary, this research work proposes the use of a KBE with a much more limited scope. Indeed, the MMG is not conceived as an integrated design tool, but as a functional component within a much broader loosely coupled design system (the DEE). In this case, KBE is exploited for the development of a parametric aircraft model able to support the link with *external* multidisciplinary analysis tools. Also the actual aircraft conceptual design knowledge has been kept outside for other dedicated tools, such as the DEE initiator.
The development of a KBE generative model able to produce *a fully engineered aircraft design* would have been not only a tremendously complex task, but even unfeasible and, eventually undesirable in view of supporting a *distributed* MDO approach. As relevant past experiences have demonstrated (Staubach, 2003), the attempt to capture in the product model *all* the rules from *all* the involved disciplines (from aerodynamic to manufacturing), would reduce to zero the chances – of whatever sophisticated KBE application – to successfully deliver an engineered solution.

4.11.5 A different place for AI in the design process

The use of Artificial Intelligence (in particular of rule-based reasoning) to support the aircraft design process is very promising and offers a great potential towards *design automation* as demonstrated by this work. However, the AI approach is worthwhile as far as it addresses capturing and automating the repetitive parts of the design process, without aiming at the replacement of the designer creative role.
In our opinion the development of a computer system that is able to capture and replace the creative and sometimes revolutionary contributions of human designers, would represent not only an impossible objective, but even a bad investment. The return of investment of any whatever smart AI conceptual design tool would result rather limited, because it is not in the creative process of the design that most of the time and resources are drained. Designers are fast and effective in proposing potential solutions to fulfill requirements, but they need help in the verification phase of those ideas and all the related preprocessing work. Automation is needed to

provide quality data to designers as early and fast as possible, so they can make more informed decisions in the early stage of design (Raj, 1998).
Furthermore, AI-supported design tools that are based on the use of performances analogy and search for best matching case (i.e. case-based reasoning (Rentema, Jansen and Torenbeek, 1998)) have the inherent limitation of not supporting the consideration of any novel design configuration. Aircraft configurations generated with case-based reasoning can only result in linear combinations of existing cases: a blended wing body aircraft will never come out as a weighted recombination of all the Airbus and Boeing passengers aircraft developed so far!
In other words: *KBE to support analysis and optimization of good ideas, not to generate good ideas!*

4.11.6 Classes, objects, suckling pigs and other animals that resemble flies at a distance

In sections 4.3 and 4.7 of this Chapter, as well an in Chapter 3, the object oriented modeling paradigm has been presented as a pillar of the conceptual development and technical implementation of the MMG and its components. The appeal of such modeling approach has been claimed to be largely related to the way humans make mental models of the world. Apparently, the concept of objects classification is not universal, as claimed by psychologist Nisbett in his book "The Geography of Thoughts (Nisbett, 2005)" (see insert next page). Apparently people from different cultures get not just different beliefs about the world, but different ways of perceiving it and reasoning about it. Though the author does not fully agree with Nisbett's too crisp categorization of Easterners and Westerner thinkers, neither with the direct legacies Greeks→Westerners, Chinese→Easterners, he concurs with the existence of different cognitive approaches. Eventually, he acknowledges the object-oriented modeling approach to be a *useful* simplification of the world, that offers a reasonably working match between the way some people see the world and the way it can be modeled into a computerized system.

The Geography of Though – How Asian and Westerners think differently and why. Richard E. Nisbett

Jorge Luis Borges, the Argentine Writer, tells us that there is an ancient Chinese encyclopedia entitled *Celestial Emporium of Benevolent Knowledge* in which the following classification of animals appears:"(a) those that belong to the emperor, (b) embalmed ones, (c) those that are trained, (d) suckling pigs, (e) mermaids, (f) fabulous ones, (g) stray dogs, (h) those that are included in this classification, (i) those that tremble as if they were mad, (k) those drawn with a very fine camel's hair brush, (l) others, (m) those that have just broken a vase, (n) those that resemble flies at a distance".

Though Borges may have invented this classification for his own purpose, it is certainly the case that the ancient Chinese did not categorize the world in the same sorts of ways that the ancient Greeks did. For the Greeks things belonged in the same category if they were describable by the same attributes. But [...] for the Chinese, shared attributes did not establish shared class membership. […] They were simply not concerned about the relationship between a member of a class and the class as a whole. […] Finding the features shared by objects and placing objects in a class on that basis would not have seemed a very useful activity. […] The Greeks belief in the importance of that relation was central to their faith in the possibility of accurate inductive inferences: learning that one object belonging to a category has a particular property means that one can assume that other objects belonging to the category also have the property. […]

We might expect, based on the historical evidence for cognitive differences […] that contemporary Westerners would (a) have a greater tendency to categorize objects than Easterners; (b) find it easier to learn new categories by applying rules about properties to particular cases; and (c) make more inductive use of categories, that is, generalize from particular instances of a category to other instances or to the category as a whole.

CHAPTER 5
Implementation of the High Level Primitive concept in the KBE system

1. Introduction
2. Functionality and implementation of the Wing-part High Level Primitive. The surface generation module
3. Wing-part Structure definition
4. Spars definition
5. Definition of Wing box, Leading Edge and Trailing edge areas
6. Ribs definition
7. Implementation of the Connection-Element High Level Primitive
8. Towards a unified connection-element
9. Fuselage High Level Primitive implementation

5.1 Introduction

The technical implementation of the High Level Primitive (HLP) concept in the ICAD KBE system is addressed in this chapter, whereas the implementation of some Capability Modules (CM) will be addressed in chapter 6. In particular, the definition and functionalities of the HLPs Wing-part, Connection-element and Fuselage are described here.

For each HLP, the approach used to define the outer surface is addressed first, followed by the definition of the internal structure. This sequence reflects the associative relation between the HLPs' surface and structure, hence the dependency of the structural model on the aerodynamic model, as it was anticipated in Chapter 4.

The definition of the main parameters used to define the HLPs is provided, to show how the designer can interact with the MMG to control the instantiation of the various aircraft systems (fuselage, wing, etc.). Examples are given to illustrate the level of achieved modeling flexibility, as well as the current limitations.

The Wing-part will be discussed in more detail than the other HLPs. This because of the higher level of maturity reached in the development of this primitive, but also

because of the similarity in methodology applicable to and implemented for the other primitives.

The purpose of this chapter is not to provide a detailed technical report or a user manual for the MMG, but to demonstrate how the concepts discussed in the previous chapters can be implemented in a KBE application that is able to support aircraft design. Reference is provided to documents where more detailed technical information on the KBE application can be found.

5.2 Functionality and implementation of the Wing-part High Level Primitive. The surface generation module

As previously shown in the UML diagram of Fig.418, `WingPart`, the class defined to implement the Wing-part HLP, is actually a composition of two main classes, namely `WingTrunkSurface` and `WingTrunkStructure`. The former, responsible for the generation of the outer surface of any Wing-part instance, is described in this section. The latter, in charge of modeling the internal structure, will be addressed in Section 5.3.

As a matter of fact, the modular definition of the Wing-part primitive goes even further: as illustrated in the class diagram of Fig. 5.1, `WingTrunkSurface` is again an aggregation of several classes (including two capability modules), developed at the scope of providing the following three main functionalities:

1. Generation of Wing-part instances with trapezoidal planform
2. Generation of Wing-part instances with curved leading and trailing edges
3. Generation of Wing-part instances (both with trapezoidal planform and curved LE/TE edges) with a deflected trailing edge movable (e.g., a rudder, an aileron or an elevator)

Functionality 1 was the first one developed during the evolutionary growth of the MMG, and is mainly provided by the `WingTrunkSurface` and `WingSection` classes of Fig. 5.1. The other classes and capability modules have been developed later to extend the basic modeling capability of the MMG. These will be addressed in Section 5.2.11 and 5.2.12.

5.2.1 Generation of Wing-part instances with trapezoidal planform: modeling capabilities and limitations

`WingTrunkSurface` generates the surface of any given Wing-part instance by constructing (*lofting* in ICAD parlance) a smooth surface across a skeleton of wing sections (i.e., airfoils), which are generated by `WingSection`. These wing sections are curves that interpolate through sets of points, whose coordinates are read from external datafiles.

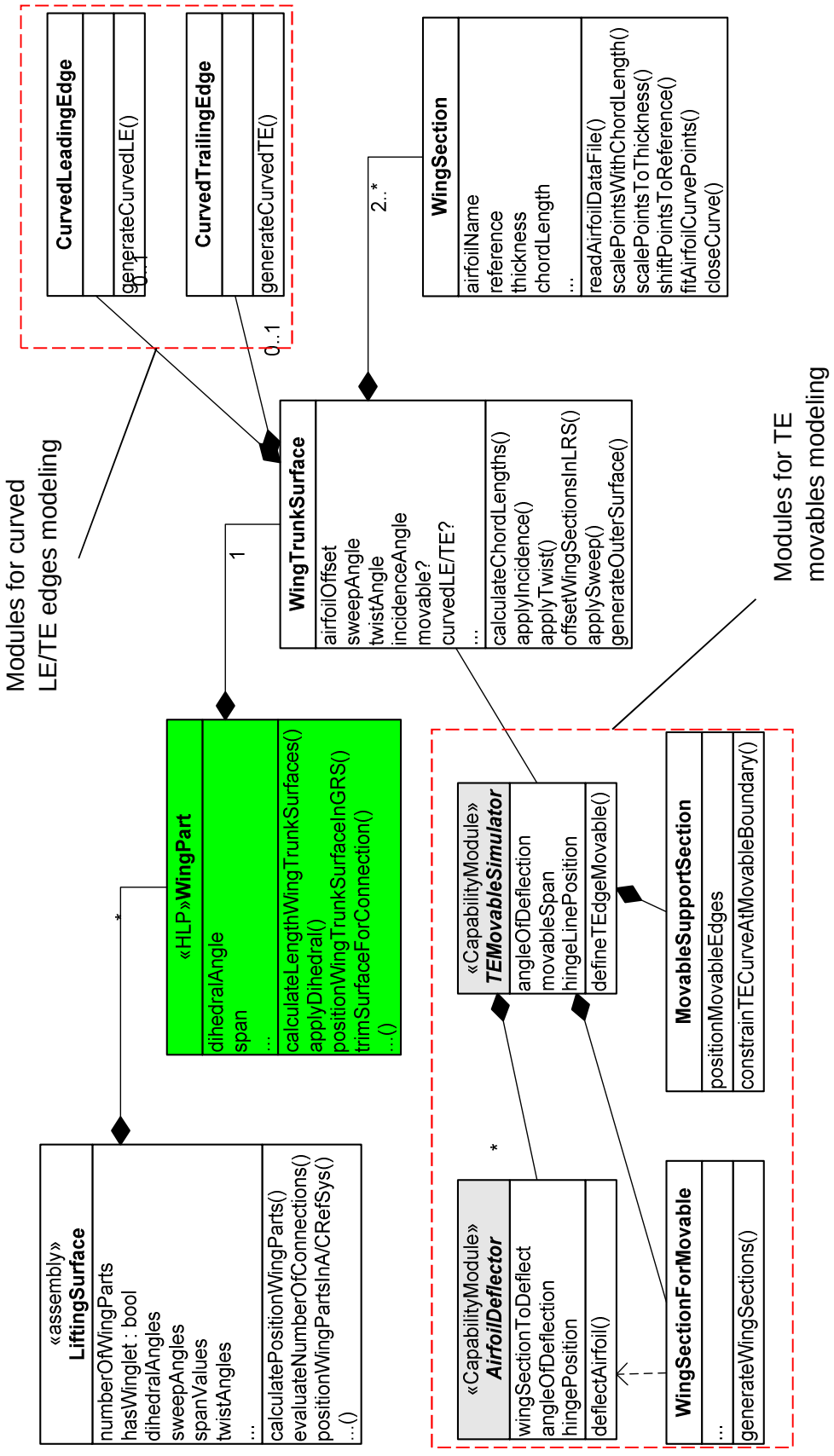


Fig. 5.1: Class diagram of the Wing-Part HLP, with details of the outer surface generation components

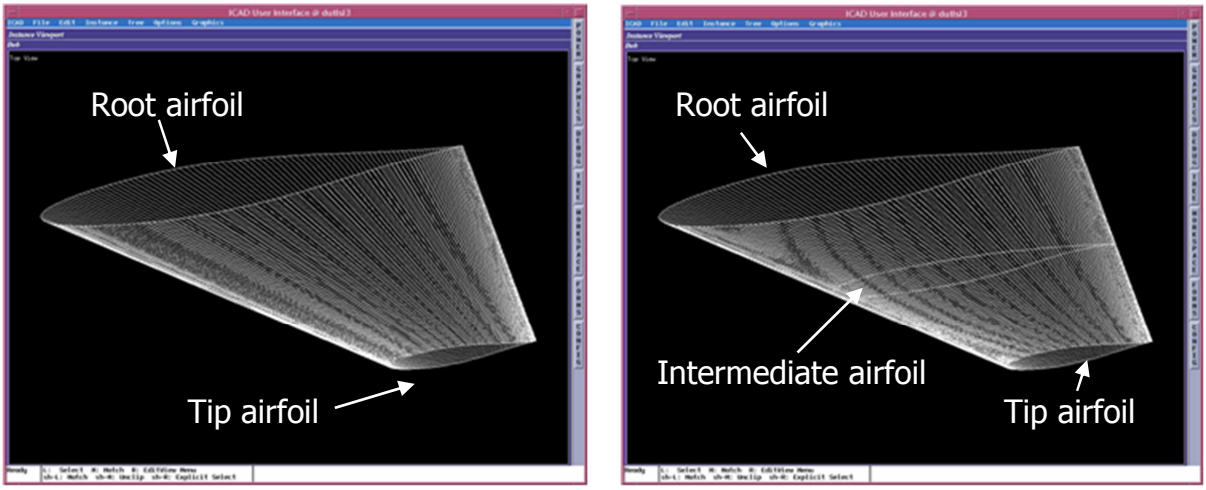


Fig. 5.2: examples of two wing-part surface instantiations built with two (left) and three (right) different airfoils. The surface on the right has double curvature.

`WingTrunkSurface` needs a minimum of two airfoils, one at the root and one at the tip section of the wing-part. However, the amount of different airfoils that can be used is unlimited: see the 2..* multiplicity on the `WingTrunkSurface-WingSection` association (refer to Appendix A for a definition of the UML multiplicity labels). Two instantiations of `WingTrunkSurface` are shown in Fig. 5.2. One is generated using two airfoils, the other using three. In the first case the result of the lofting operation is a *single-curvature* surface, whilst in the second case, is a smooth *double-curvature* surface. Both surfaces are generated using the same ICAD primitive called *lofted-surface* (Knowledge Technologies International, 2001b).
The list of parameters necessary to define a Wing-part surface instantiation (with trapezoidal planform and no movables) is provided in Table 5.1. When more Wing-part instances are required to model, for example, a complex cranked wing, the parameter values indicated in the table will have to be assigned for each Wing-part instance.
On the base on the adopted parameterization approach, the modeling capabilities and limitations of `WingTrunkSurface` can be summarized as follows:

- The type (and number) of airfoil curves, specified by the attribute *airfoil-name-list*, must be names of predefined (.dat) files, containing the airfoils' definition data (see section 5.2.9 for details). All the airfoils datafiles are stored in a folder, which is here addressed as Airfoil-Library.
- Within a given Wing-part instance, the airfoils can be positioned only parallel to each other. Their span wise position (offset) is defined via the parameter *airfoil-offset-list*. The offset values are expressed as percentages of the given Wing-part instance span (the values 0 and 1 are assigned for the root and tip airfoils respectively).

Parameter	Expected value	Example	Description
Cr	Positive real number	200	Root/Tip chord length of the Wing-part instance.
Ct	Positive real number	120	
Span	Positive real number	250	Span of the given Wing-part instance. N.B. In case of a dihedral angle different than zero, the span is different than the distance between the root and tip airfoil planes.
Reference	Real number	0 → reference line = Leading edge 0.25 → reference line = Quarter-chord line	Indicate the reference line used to define the sweep and twist angles of the given Wing-part instance.
Sweep-angle	Real number	(degrees 30)[1]	Angle defined with respect to the Wing-part reference line
Twist-angle	Real number	(degrees 5)[1]	The angle between the root and tip airfoil chords of the given Wing-part instance.
Twist-angle-root	Real number	(degrees 0)[1]	The incidence angle of the given Wing-part instance
Dihedral-angle	Real number	(degree 3)	Rigid rotation of the lofted surface. This parameter affects the length of the given Wing-part instance (which is equal to the ratio of span and the cosines of the dihedral angle)
Airfoil-name-list	List of strings	(list 'MyAirfoil-1 'MyAirfoil-2)	A list of strings, corresponding to names of datafiles[2]
Airfoil-offset-list	List of real in the range [0, 1]	(list 0.0 1.0)	The spanwise position of each expressed as a span fraction of the given Wing-part instance[2]
Airfoil-thickness-list	List of real	(list 1.0 0.5)	A list of multiplication factors to modify the thickness ratio of the airfoils[2]

[1]Without the keyword "degree", the angle is assumed in radians

[2]These lists must contain the same amount of values (i.e., one per wing section)

Table 5.1: list of the main parameters for the definition of a Wing-part instance.

- The parameter *airfoil-thickness-list* contains a list of multiplication factors to modify the thickness ratio of the airfoils indicated in *airfoil-name-list*, without the need to add extra airfoil datafiles to the library (more details in section 5.2.9)
- It is possible to define the length of the root and tip chord only. The chord length of all the eventual intermediate airfoils is automatically determined by *linear interpolation* (trapezoidal planform).
- The twist-angle parameter defines the rotation angle of the tip airfoil with respect to the root airfoil. The rotation of all the eventual intermediate wing sections is evaluated by *linear interpolation*.
- The *sweep-angle* value is constant within a given Wing-part instance and is applied by shifting the various wing sections in their planes (i.e., sweep by wing shearing. See section 5.2.5). The sweep angle is defined with respect to the same reference line (e.g., leading edge line, quarter chord line) as the twist angle.
- The reference line can be selected by means of the *reference* parameter (more details in Section 5.2.3). This is a global parameter. In other words, *all* the Wing-part instances used to model a lifting surface share the same reference line definition.
- The *dihedral-angle* value is constant within a given Wing-part instance and is applied by a rigid rotation of the Wing-part lofted surface (i.e., not by wing shearing). In this way, the dihedral does not affect the thickness of the given Wing-part instance (see discussion in section 4.6.2).
- Since the Wing-part span is a user-defined parameter, the length of a given Wing-part instance (i.e., the distance between the root and tip airfoil planes) is affected by the dihedral angle as discussed in section 5.2.8

5.2.2 Modeling process for trapezoidal planform Wing-part instances

This section describes the implementation of the modeling process, whose capabilities and limitations have been addressed above.
As shown in the diagram of Fig. 5.1, three classes are responsible for the generation of trapezoidal planform Wing-part instances, namely `WingPart`, `WingTrunkSurface` and `WingSection`[1]. The top level class `LiftingSurface` is responsible of assembling the complete lifting surface by appropriate positioning of the various Wing-part instances.

[1] Note how the main operations and parameters manipulated by these classes have been indicated in the related fields of their UML representation.

The sequence of activities implemented by these classes can be split in the following main blocks (details about the various steps will be provided in the subsequent sections):

1. **Generation of the airfoil curves**. All the operations in this block are performed by `WingSection`, as many times as the number of wing sections specified for the given Wing-part instantiation (see section 5.2.9 for details) dictates.
    - Reading the airfoil normalized point coordinates from datafiles
    - Scaling of points according to chord length
    - Scaling of points by application of the thickness factor
    - Positioning of points inside the given Wing-part Local Reference System and generation of fitted curve
2. **Generation of the lofted surface(s)**. All the operations in this block are performed by `WingTrunkSurface`, as many times as the number of Wing-part instances used to model the given lifting body.
    - Spanwise positioning of the airfoil curves in the Wing-part Local Reference System (See section 5.2.4)
    - Application of sweep angle by shifting the airfoil curves along their chord vector (See section 5.2.5)
    - Application of the incidence and twist angles by rotating the shifted airfoil curves around their reference point (See section 5.2.6)
    - Generation of the lofted surface through the set of wing sections (Fig. 5.2)
3. **Positioning and orientation of the lofted surface**(s) in the Lifting-surface Global Reference System. All the operations in this block are performed by `WingPart`, as many times as the previous block of activities requires.
    - Application of the dihedral angle to each Wing-part instance by rigid rotation of the Wing-part Local Reference Systems in the Global Reference system (details on the reference systems in Section 5.2.3)
    - Positioning of each Wing-part instance in the Lifting-surface Global Reference System
    - Trimming of the lofted surface(s) as required for the instantiation(s) of the Connection-element HLP (see section 5.7)
4. **Build up of the complete lifting body surface**. All the operations in this block are performed by `LiftingSurface`, as many time as the number of lifting surfaces present in the given aircraft architecture.
    - Positioning of the Global Reference System in the Aircraft Reference System

- Computation and instantiation of the required number of connection elements, based on the number of dihedral angle discontinuities (more detail in section 5.7).
- Positioning and instantiation of the `Winglet` and/or `EndCap` classes
- Mirroring of the complete Lifting-surface instance, if required (i.e., to model a conventional aircraft configuration, the left wing half and the left horizontal tail empennage are generated first and then mirrored to generate the right half)

It should be noted that, whilst the activities listed above are performed starting from the leaves of the product tree (i.e., first the wing sections are generated, than the lofted surfaces are built and finally the whole lifting surface is assembled), the flow of information required to run the process cascades from the root of the product tree down to the leaves. In fact:

- The location of the surface instances positioned by `WingPart` in the Global Reference System is computed by `LiftingSurface`.
- The length of the lofted surfaces generated by `WingTrunkSurface` is calculated by `WingPart`
- The chord lengths required by `WingSection` to scale the airfoil curves is computed by `WingTrunkSurface`

5.2.3 Definition of Global and Local Reference Systems

As anticipated in the previous section, the implemented modeling approach is based on the use of three reference systems, namely: the Aircraft Reference System, the Lifting-surface Global Reference System (GRS), and the Wing-part Local Reference Systems (LRS) (see Fig. 5.3). The last two are those relevant for the definition of any lifting surface and their relative positioning is shown with details in Fig. 5.4.

Each Wing-part instance is built with respect to a local reference system (LRS), i.e., there are as many LRSs as the number of Wing-part instances.

The *length*, the *twist angle*, the *root-twist angle* (i.e., the incidence angle) and the *sweep angle* of each Wing-part instance are defined in the LRS systems.

On the other hand, the relative position of the various Wing-part instances and the values of their *span* and *dihedral angle* are defined in the Lifting-surface GRS.

As shown in Fig. 5.4, the longitudinal axis of each Wing-part LRS is parallel to the longitudinal axis of the relative Lifting-surface GRS (which is in turn parallel to the longitudinal axis of the Aircraft Reference System).

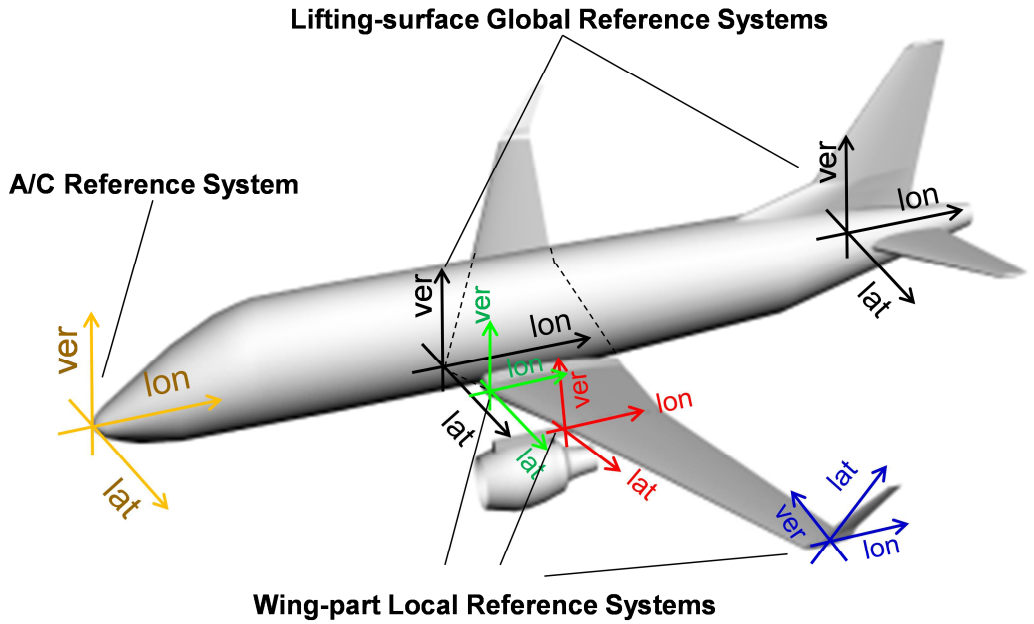


Fig. 5.3: the three types of reference systems used in the implemented modeling approach.

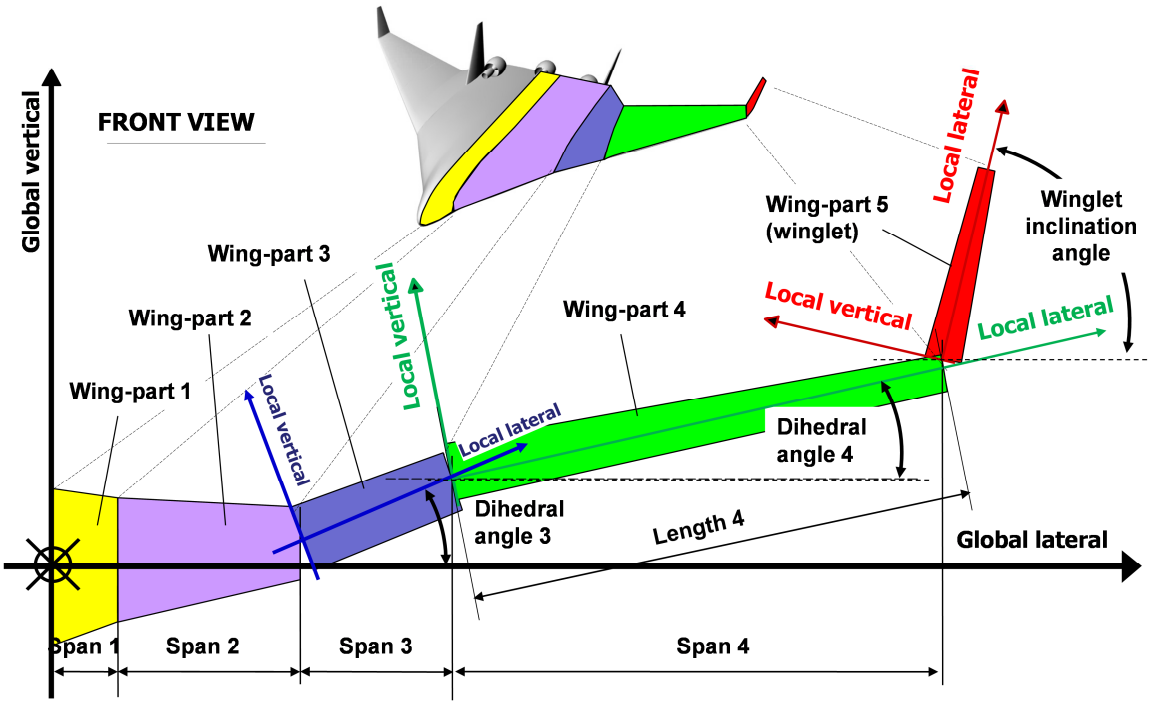


Fig. 5.4: Global and Local Reference Systems to define and position the Wing-part instances in a Lifting-surface assembly. Example of the MOB BWB (La Rocca, Krakers and van Tooren, 2002) (connection elements not shown).

Given a Lifting-surface assembly consisting of *n* Wing-part instances, the LRS-origin of Wing-part instance (*i*) is positioned in the GRS as follow:

$$LateralPosition = \sum_{n=1}^{i-1} span(Wingpart(n))$$

$$LongitudinalPosition = \sum_{n=1}^{i-1} span(Wingpart(n)) * \tan(sweep(Wingpart(n))$$

$$VerticalPosition = \sum_{n=1}^{i-1} span\big(Wingpart(n)\big) * \tan(dihedral(Wingpart(n)))$$

5.2.4 Positioning of wing sections and role of the *reference* parameter

Each airfoil generated by `WingSection` is first positioned in the Wing-trunk LRS, with its chord parallel to the LRS longitudinal axis and at a lateral position equal to the Wing-part length[2] percentage indicated by the input parameter *airfoil-offset-list* (Table 5.1). The longitudinal position of each airfoil is set according to the value of the parameter *reference*. As shown in the examples of Fig. 5.5, when *reference* is set to 0, the airfoil curves are shifted longitudinally such that all the leading edge points get positioned on the LRS lateral axis; when reference is set to 0.25, the airfoil curves are shifted such that the all the quarter chord points (longitudinal shift equals 0.25 times the chord length) are placed on the LRS lateral axis.
Any value is allowed for the reference parameter, although the values 0 and 0.25 are the most frequently used, being the leading edge and the quarter chord line the two most common *reference lines* used to define the sweep and the twist angle of a lifting surface.

5.2.5 Application and definition of the sweep angle

After the preliminary placement of the airfoil curves in the LRS, as described in the previous section, `WingTrunkSurface` applies the user-defined *sweep angle* by shifting the wing sections further along the LRS longitudinal axis, as illustrated in Fig. 5.6. The longitudinal shift of each airfoil curve is computed as follows:

$$longitudinalShift(airfoil(i)) = lateralOffset(airfoil(i)) \cdot \tan(sweepAngle)$$

As a consequence of this double longitudinal shift, the sweep angle results automatically applied with respect to the *reference line* selected by means of the *reference* parameter.

[2] Since the implemented method is able to model only straight wing parts, a certain percentage of the span or the length yields to the same lateral positioning of a given airfoil

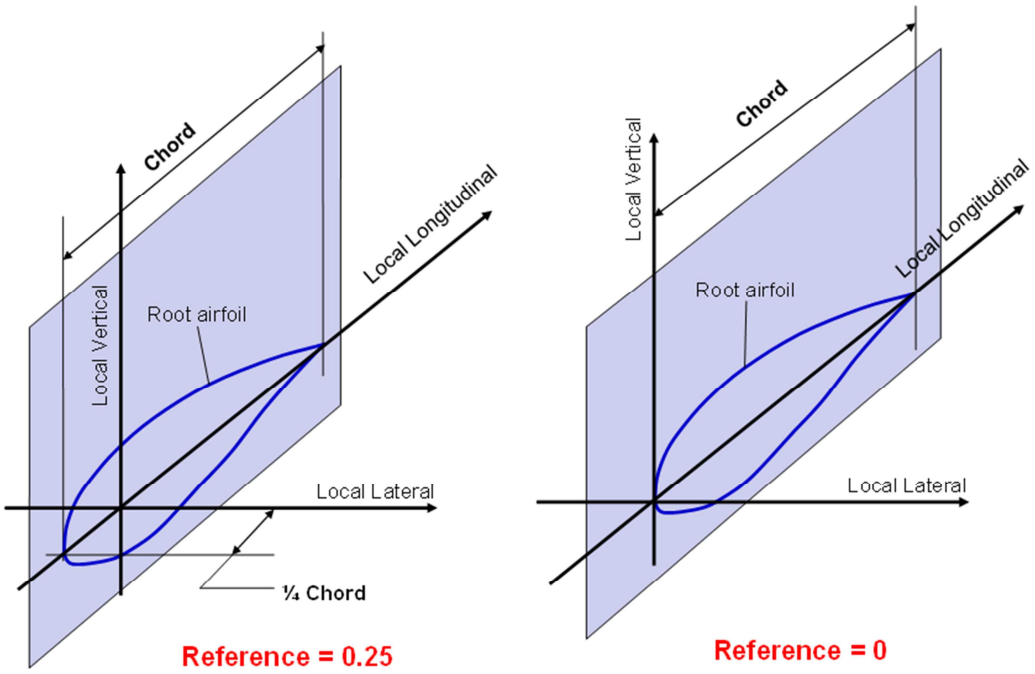


Fig. 5.5: Positioning of the root airfoil curve in the Local Reference System (LRS) of the given Wing-part instance. Effect of the parameter *reference* on the airfoil longitudinal positioning.

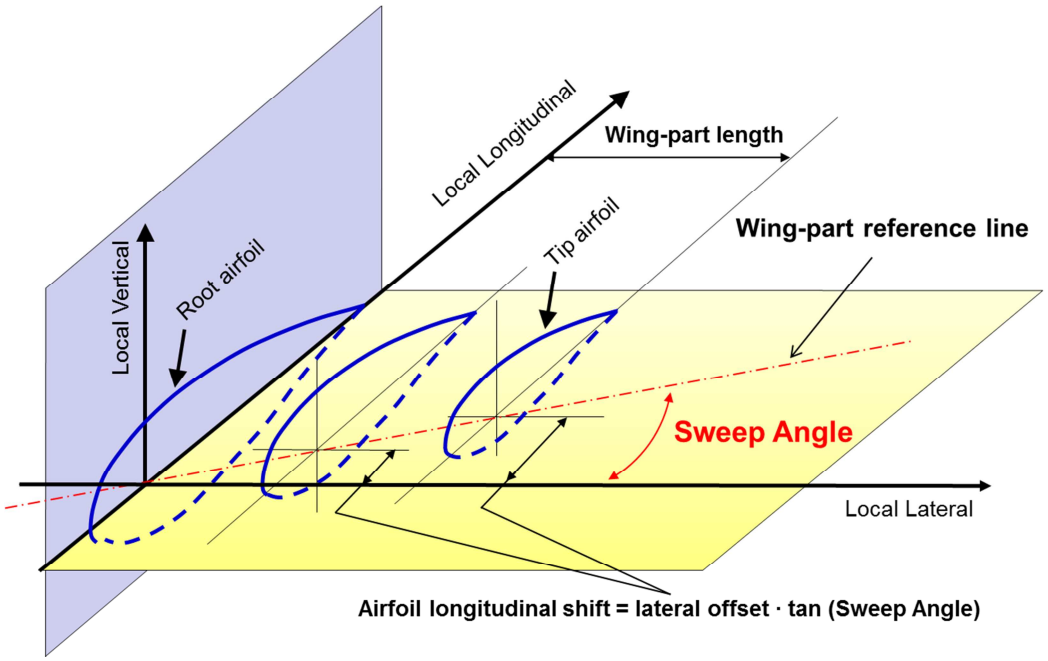
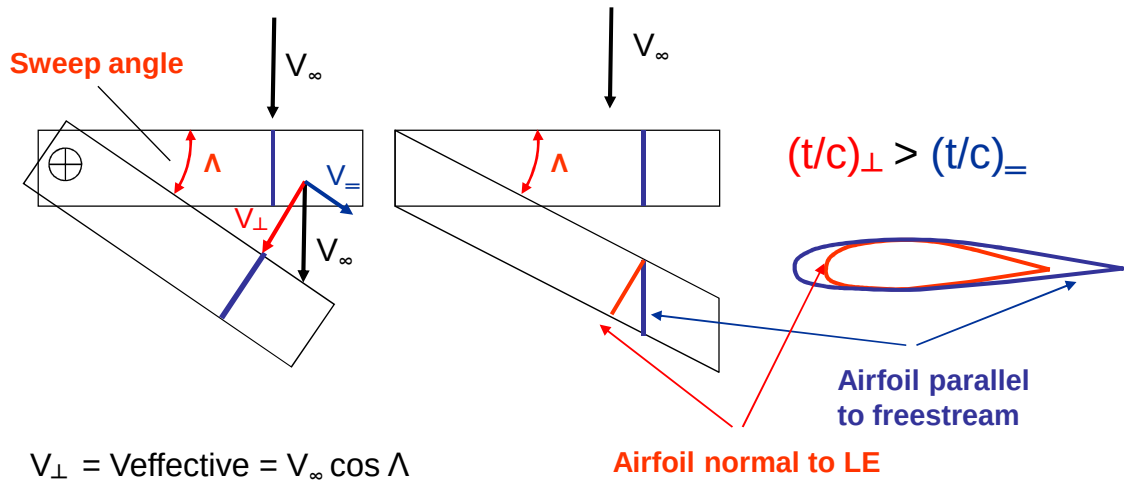


Fig. 5.6: positioning of the wing sections in the Wing-part instance Local Reference System. The length (not the span) of the Wing-part instance determines the lateral position (offset) of the wing sections. The sweep angle is applied by shifting the various wing sections along the LRS longitudinal axis.

Another result from this way of applying the sweep angle (actually not the only one possible, as discussed in the insert below) is that all the airfoil curves remain parallel to each other and normal to the LRS lateral axis, such that *the span of the given Wing-part instance remains unaffected*.

A note on sweep angle definition. Shearing or pivoting?

Two ways of applying sweep to a wing are generally found in literature: the first one is achieved by applying a rigid body rotation to the wing, as shown in the left case of the picture below. In this way the air, through the *effective free stream speed component* $V_{\perp}$, would always "see" the same airfoils. This is what actually happens for pivoting/swing (variable sweep) wings. In the second method – the one implemented in this work - the sweep is applied by shearing the wing, i.e., by shifting the airfoils in planes parallel to the free stream (right case in the picture below). This second method is **industry practice**. However, the designer should be aware that the effective flow component does not see the selected airfoil, but a generally different one with higher thickness ratio, as illustrated below. This reduces the positive effect of sweeping on the critical Mach number.



Being applied and measured in the LRS, the sweep angle is independent from the dihedral angle, which is applied in a later step via a rotation of the Wing-part LRS in the Lifting-surface GRS.

For positive value of the sweep angle, `WingTrunkSurface` shifts the airfoil curves towards the positive direction of the LRS longitudinal axis. Given the orientation of the LRSs in the Aircraft Reference Systems, a positive sweep angle yields a backward swept wing (as by convention).

5.2.6 Definition of the twist and wing setting angle

The twist angle is applied by a rigid rotation of the airfoil curves in planes normal to the LRS-lateral axis, with the rotation point set at the intersection of the given airfoil plane with the reference line (see Fig. 5.8).
In this way, the application of the twist angle remains independent of the sweep angle definition and the sequence in which twist and sweep are applied is irrelevant.
The twist angle is a user-defined Wing-part parameter and is defined as the *angle between the chord of the root airfoil and tip airfoil.* The rotation angle of the intermediate airfoil curves inside a Wing-part is computed automatically by scaling the twist angle *linearly* with the Wing-part instance span:

$$rotation(airfoil(i)) = lateralOffset(airfoil(i)) \cdot \text{twistAngle}$$

The twist-angle is positive when the nose is rotated upward, as by convention.

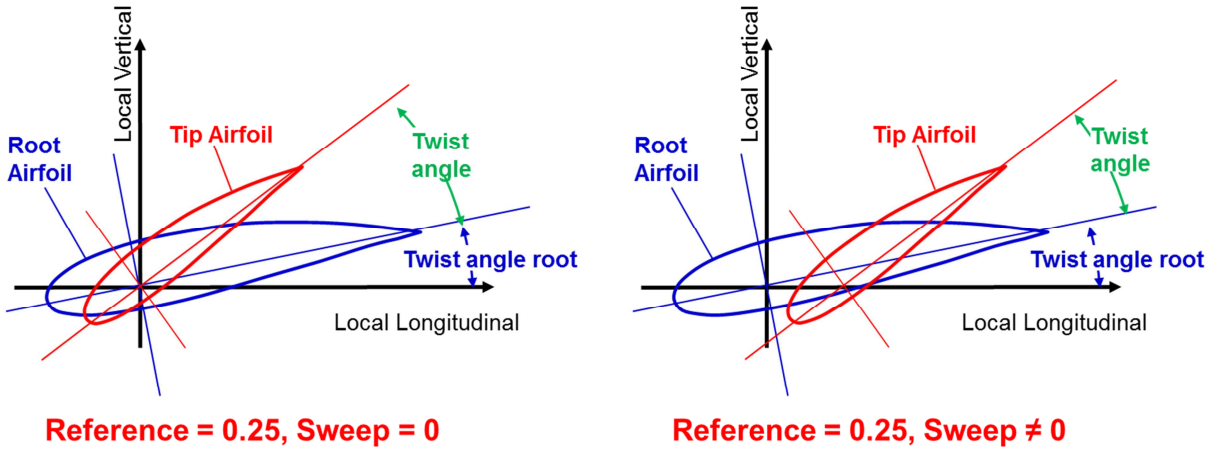


Fig. 5.8: twist angle and twist-angle-root angle (i.e. Wing-part instance incidence angle) definition. The angles definition is not affected by the sweep. (negative angles shown in picture).

The *setting angle* of each Wing-part instance, i.e., the angle between the root chord of the given Wing-part instance and the local longitudinal axis, is determined by the parameter *twist-angle-root*. This angle is simply superimposed to the rotation angle in the formula above.

Whilst the user can define a different twist angle for each Wing-part instance, only one setting angle can be assigned by the user for the whole Lifting-surface instance[1]. This is the *twist-angle-root* of the most inboard Wing-part instance and corresponds to what generally addressed in literature as *wing setting angle*. This is defined as the angle formed by the wing root chord with the fuselage longitudinal

[1]With the exception of the Winglet twist-angle-root (commonly addressed as *toe angle*), whose value is another user-defined parameter in the MMG input file.

axis. For a given alignment of the fuselage with the flight speed vector, the wing setting angle determines the actual angle of incidence (or attack) of the wing and its airfoils.

The parameter *twist-angle-root* of each Wing-part instance (excluded the most inboard one), is automatically computed by the MMG as follow:

$$twistAngleRoot\big(airfoil(i)\big) = twistAngleRoot + \sum_{n=1}^{i-1} twist(Wingpart(n))$$

5.2.7 Definition of dihedral and winglet inclination angles

As shown in Fig. 5.4, the dihedral angles of the various Wing-part instances composing a lifting surface are all defined in the Lifting-surface GRS. The dihedral angles are set by rotating the various Wing-part LRSs (hence, their content) around their local longitudinal axes.

A definition of dihedral (and inclination) angle follows:

*The **dihedral angle** (and winglet/fin **inclination-angle**) is the angle between the lateral axes of the lifting surface global reference system and the wing-part local reference system, measured on the plane orthogonal to the global reference system longitudinal axis.*

The dihedral angle and the winglet-inclination angles are positive when the tip of the wing-part is moved upward with respect to the horizontal position. The winglet inclination angle is *complementary* to the so-called winglet *cant angle* (i.e., they add to 180 degrees).

The dihedral angle is applied as a rigid rotation of the lofted surface generated by WingTrunkSurface, hence *after* the sweep and twist angles have been set.

5.2.8 Definition of Wing-part span and length

The *wing-part span* is defined as the distance between its root and tip airfoil planes – which is the *Wing-part length* – projected on the horizontal plane of the global reference system, and measured along the GRS-lateral axis (see Fig. 5.4).

Being the span and the dihedral angle user-defined parameters for Wing-part, the Wing-part length is first computed by `WingPart` and then fed to `WingTrunkStructure` to start the generation of the lofted surface:

$$length\big(WingPart(i)\big) = span\big(WingPart(i)\big)/\cos\big(dihedral\big(WingPart(i)\big)\big)$$

It follows that the length of a Wing-part instance increases with the dihedral angle, but does not depend on the sweep angle.

In the case of winglets, the length and the span of the Wing-part instances coincide and their value is user-defined, hence independent from the winglet-inclination-angle.

5.2.9 Generation of the airfoil curves

As mentioned above, the MMG user can specify for each Wing-part instance a specific set of airfoils. The name of these airfoils is specified using the *airfoils-name-list* parameter (see Table 5.1); each airfoil name must match the name of a corresponding airfoil data file, which must be available in a predefined "airfoil library" directory. These files are actually plain ASCII files, each containing a list of *point coordinates* (see example in Table 5.2). The user can expand the airfoil library, at any time, by adding files containing new airfoil definitions.
An airfoil file obeys the following conventions (Fig. 5.9):

- The coordinates of the points are specified in a 2D reference system
- The coordinates are normalized with respect to the chord length (i.e., all the airfoils in the library have a chord length equal to one)
- The points are provided in sequence: from the trailing edge (TE) point (1 ; 0), to the leading edge (LE) point (0 ; 0) and back to the trailing edge point.
- The LE point and the TE point(s) coordinates must be included in list.

The `WingSection` class instance opens the given airfoil file, reads all coordinates, scales them according to the required chord length, and generates a curve using the ICAD *fitted-curve* primitive.
In order to obtain correctly lofted surfaces, all the airfoil curves must have the same *direction*. That means the fitted curve must interpolate the points from trailing edge to trailing edge, first generating the upper part of the airfoil and then the lower (see Fig. 5.9). `WingSection` is able to check and correct the airfoil curve direction automatically, so the user is free to provide the points coordinates either in clockwise or counter clockwise sequence.

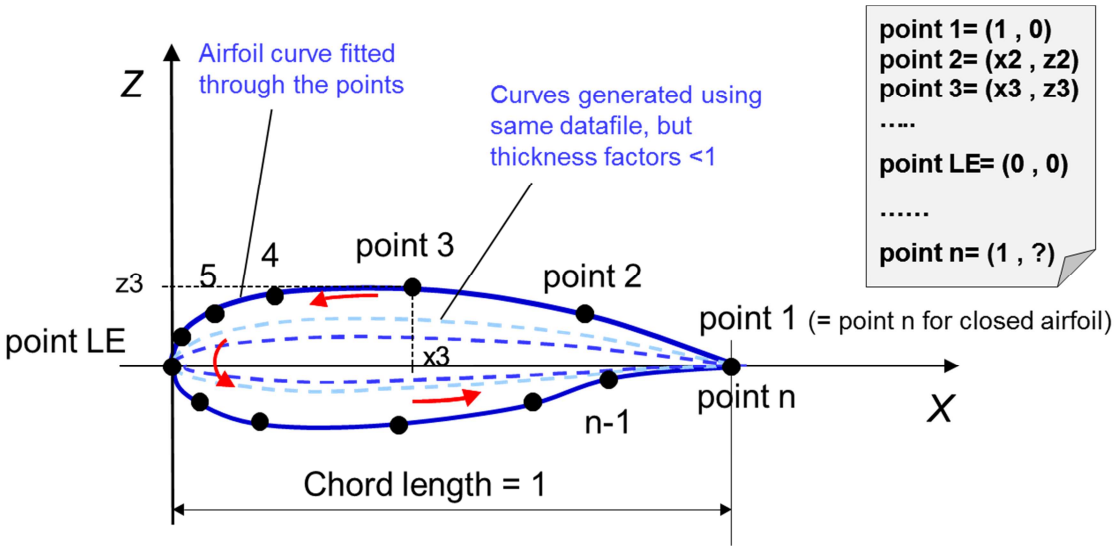


Fig. 5.9: Definition of an airfoil curve by fitting a set of points and scaling effect of thickness factor.

```
1.000000   0.000000
0.999846   0.000009
0.999380   0.000036
0.998603   0.000082
……
0.000899   0.004203
0.000300   0.002566
0.000000   0.000000
0.000300  -0.002563
0.000898  -0.004201
0.001796  -0.005829
……
0.998603  -0.000082
0.999380  -0.000036
0.999846  -0.000009
1.000000   0.000000
```

Table 5.2 : example of an airfoil data file content.

Open airfoils at the TE edge are also allowed (the coordinates of the two trailing edge points in the data file can be different). `WingSection` is able to deal with open airfoils and if required, it can close them automatically (a setting parameter is provided)

The amount and distribution (stretching) of points used to define a given airfoil are at the designer's choice. It is up to the user to supply sufficient and adequately distributed points to have the fitted curve capturing the airfoil curvatures. Different airfoil data files can contain a different number of points with different stretching; Wing-section is able to deal with that (La Rocca and van Tooren, 2002b).

The parameter *airfoil-thickness-list* (see Table 5.1) allows users to modify the thickness of the selected airfoils for the wing-part, to avoid the need to populate the airfoil library with airfoils from the same family but with different thickness ratio (e.g., NACA0010 and NACA0020). The airfoil-thickness parameter works as a stretching factor, which directly applies to the *z-coordinates* of the airfoil points (the *x-coordinates*, are stretched by the chord length). For example, if the coordinates of the NACA0010 are available in the library, a NACA0020 can be generated setting the relative airfoil-thickness value to 2.

5.2.10 Airfoils definition to support optimization. Some considerations

The implemented airfoil definition method is relatively simple and effectively supporting common practice in conceptual and preliminary design. The user can add new airfoil data files to the library without the need to modify any bit of the MMG code. Simple cut-and-paste operations can add any airfoil to the library. The airfoils can be taken from literature (University of Illinois Urbana-Champaign, last visited November 2009; Hepperle; Carmichael) or from one's own design efforts.

However, the point coordinates-based airfoil definition is not very well suited to support in aerodynamic shape optimization. The large number of points generally required to properly define an airfoil would lead to a very high number of design variables. In addition this definition cannot guarantee the generation of smooth curves when perturbing the position of single points, unless coordination mechanisms are put in place to link the perturbation of one point with those of the neighbours (Samareh, for example proposes to parameterize the perturbation rather than the geometry itself (Samareh, 2001a). This would already allow reducing the number of design variables by orders of magnitude (Straathof et al., 2008)). Of course it would

be possible to use the *airfoil name* as discrete variable, while using the airfoil thickness as an associated continuous variable. In this case, however, the size of the design space would be very limited (and discontinuous!), even with a large database of "prefab" airfoils available.

A better way to support optimization while maintaining the airfoil definition approach discussed above, would be that of extracting the *B-spline control points and weights* from the fitted curves generated by ICAD[2] when interpolating through the set of point coordinates. These control points with relative weight could then be used as input data to re-generate the airfoils as B-splines (which are ICAD geometry primitives) (Knowledge Technologies International, 2001b). The advantage for the optimization is twofold: the B-spline control points and weights are much less than the original set of point coordinates; the variation of each control points can affect a large part of the airfoil curve, without creating unwanted wrinkles or waviness. The B-spline approach could be used not only with curves, but also for surfaces, as it has been actually tested for the definition of the Fuselage HLP surface (see Section 1.1).

Other convenient approaches to support the optimization of airfoils and aerodynamic surfaces in general would require an *analytical definition* of airfoils and surfaces, such as the CST method proposed by Kulfan (Kulfan, 2008), based on Bernstein polynomials (and recently extended by Straathof to allow easy manipulation of both local and global shape variations (Straathof et al., 2008)); and the method proposed by Carpentieri (Carpentieri, 2008), based on Chebyshev polynomials. These authors have demonstrated the possibility to cover very large design spaces, using the (few) coefficients of the polynomials as design variables.
Other parameterization methods for optimization can be founded in the surveys by Haftka and Grandhi (Haftka and Grandhi, 1986) and Ding (Ding, 1986) up to 1986, and by Mousavi (Mousavi, Castonguay and Nadarajah, 2007) up to 2007. Also Samareh (Samareh, 2001b) provides an extensive survey of parametric models for combined structural and aerodynamic optimization, up to the year 2001.

Without entering in the specific merit of these methods, we acknowledge that the modular structure of the Wing-part HLP would allow substituting the actual `WingSection` class with an alternative analytical airfoil generator module. This would not require any major modification to the rest of the HLP structure. Also the entire `WingTrunkSurface` class could be replaced by a new class encapsulating analytical surface modeling methods. In fact, for `WingTrunkStructure`, the specific method used to generate the wing-part outer surface is irrelevant, as far as

[2] ICAD allows easy access to low level geometry information, such as control points and weights of generated B-splines. Besides, the B-spline-curve is an ICAD geometry primitive, which needs control points and weight factors as basic input.

an outer surface is eventually generated such that its structure modeling capabilities can be deployed.
The MMG can also be expanded to include both the abovementioned airfoil/surface generation methods and let the user chose the one most suitable to the case at hand. The conceptual design process could be initiated selecting an airfoil off the shelf, and then "translated into a proper mathematical format" to support the optimization process in a later design stage.

5.2.11 Definition of Wing-parts with curved leading and trailing edges

The surface modeling approach described above is only able to support the design of trapezoidal wing parts. There are several cases where designers could benefit from a more "free-form" planform design approach. For example, in order to model other wing tips than simple cut-offs, designers need the possibility to create curved leading and trailing edges, still based on the reference trapezoidal wing shape.

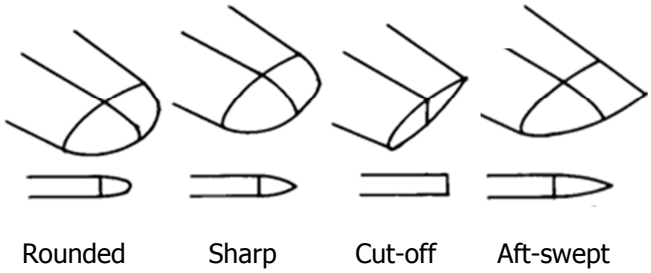


Fig. 5.10: different types of wing tips (Raymer, 2006)

On this purpose, extra functionalities have been added to the basic Wing-part architecture described in the previous sections, which allow, for example, the generation of rounded, sharp and aft-swept wing tips (Fig. 5.10). Modeling of wing fairings or the center section of BWBs can benefit from the possibility to design curved LE/TE edges (Fig. 5.11).
An extra set of dedicated parameters (refer to Table 5.3 for details) has been made available via the MMG input file, which the user can set to generate partially or completely curved leading and/or trailing edges.

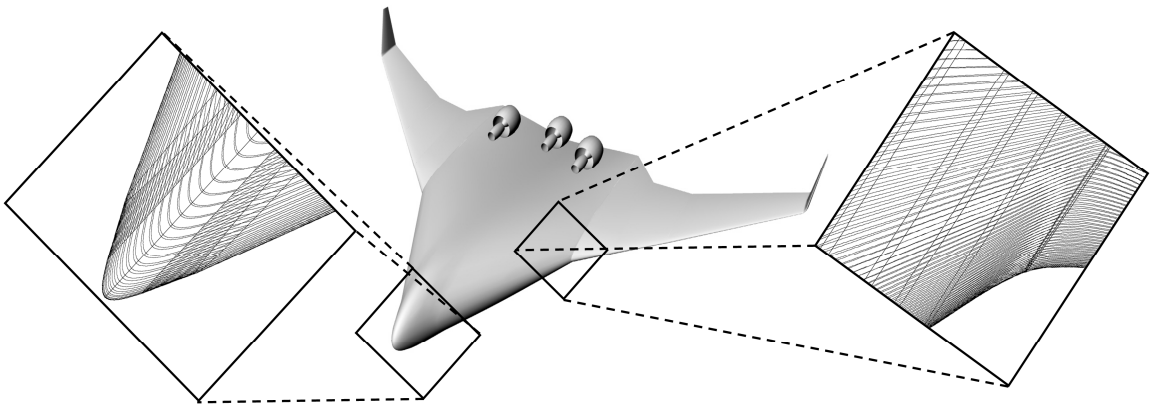


Fig. 5.11: curved leading edges in the BWB center section and at the wing transition.

In order to model Wing-part instances with curved LE/TE edges, the user must still assign the input parameter required for the definition of a trapezoidal planform. In fact, the new introduced parameters work as kind of "correction factors" for the reference trapezoidal planform, as follows (refer to Table 5.3 and Fig. 5.13):

- the *LE(TE)-interval-list* parameter specifies the spanwise portion(s) of the LE and TE edges to be curved and that (those) to be left unmodified.
- the *Delta-Cr-TE/LE* and *Delta-Ct-LE/TE* parameters modify the length of the root and tip chords of the basic trapezoidal planform
- the *Alpha-Cr-TE/LE* and *Alpha-Ct-LE/TE* parameters specify the direction of the tangency vectors of the curved LE and TE, at the root and tip, respectively.

parameter	**example**	**description**
**-interval-list*	(list 0.3 0.6)	Specification of the **curved portion** of the *. The numbers in the list are percentages of the Wing-part span. If this list is empty, no curved parts will be defined on the *. If equal to (0 x) only the part next to the root will be curved. If equal to (x 1) only the part next to the tip will be curved. If equal to (x y) both parts next to the tip and root will be curved. If equal to (0 1) no straight part is defined on *.
*Alpha-Cr-**	(degrees 60)	Local * sweep angle at the root section (values <0 are also allowed). This parameter is ignored in case *-interval-list is empty or equal to (list x 1)
*Alpha-Ct-**	(degrees 90)	Local * sweep angle at the tip section. This parameter is ignored in case *-interval-list is empty or equal to (list 0 x)
*Delta-Cr-**	0.3	Extension (contraction if <0) of the * root chord length. This parameter is ignored in case *-interval-list is empty or equal to (list x 1)
*Delta-Ct-**	-0.35	Extension (contraction if <0) of the * tip chord length. This parameter is ignored in case *-interval-list is empty or equal to (list 0 x)
** stands for either LE or TE* *X is a number in the range [0 1]*		

Table 5.3: parameters for the definition of Wing-part instances with curved LE/TE.

When the *LE(TE)-interval-list* parameter is not null, `WingTrunkStructure` first generates a reference trapezoidal Wing-part instance, then, by means of the abovementioned parameters and the dedicated classes `CurvedLeadingEdge` and `CurvedTrailingEdge` (see Fig. 5.1), generates the new, curved LE/TE edges. From these new curves follows a (not linear) chord length distribution, which is used

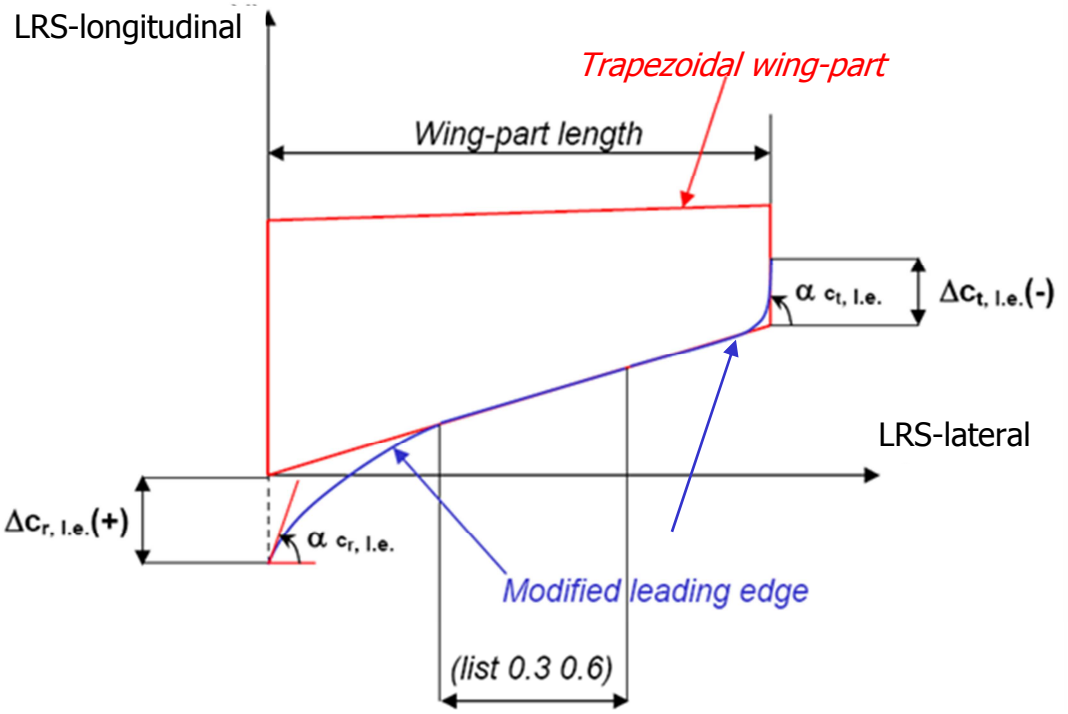


Fig. 5.13: Definition of a curved leading edge Wing-part instance.

to define a new set of wing sections[3]. Finally, a new lofted surface is built through this new skeleton of wing sections. The example of an aft-swept wing tip, with relative parameters is shown in Fig. 5.12.

5.2.12 Modeling of control surfaces

In order to account for the effect of deflected control surfaces in the aerodynamic analysis of the aircraft, additional dedicated MMG modules have been developed. Deflected movable surfaces such as ailerons, rudders and elevators can be defined in any wing-part primitive, with some level of flexibility in their position and planform shape. The input parameters for the movable

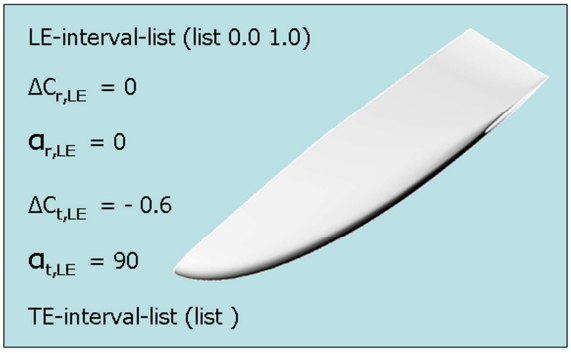


Fig. 5.12: example of an aft-swept wing-tip model with corresponding parameters

[3] The trapezoidal Wing-part surface is actually intersected with at a set of planes orthogonal to the LRS-lateral axis and appropriately distributed along the span. Each resulting intersection curve is then scaled according to the new local chord value and "attached" to the new LE/TE curves in correspondence of its LE/TE points.

definition are described in Table 5.4.
The level of modeling detail is rather limited in the sense that the movables are not generated as entities separate from the wing-part: there are no gaps, laps and slots between movable and wing surface. No composite movables can be modeled (such as elevators with trimming tabs), neither movables that increase the wing-part surface (such as high lift devices with Fowler movement).

parameter	**example**	**description**
Hinge-position-root	0.7	Position of the movable hinge axis, indicated as fractions of the root and tip chord length of the trapezoidal wing-part
Hinge-position-tip	0.7	
Movable-start-direction	(degrees 5), or, fd	Orientation of the movable root and tip edges. 4 options: *fd* → side edge of the movable parallel to flight direction *Hinge* → side edge of the movable normal to hinge line *TE* → side edge of the movable normal to Trailing Edge line *(degree x)* → side edge of the movable rotated x degrees clockwise with respect to the flight direction
Movable-end-direction		
Rotation-angle	(degrees 8)	A positive angle to rotate the movable down. When equal to 0, the TE-deflection procedure is disabled, although the definition of the movable remains.
Start-movable	0.2	Position of the root and tip edges of the movable, indicated as fraction of the wing-part length. If equal to (0 x) the movable starts at the root of the wing-part. *Movable-start-direction* ignored. If equal to (x 1) the movable stops at the tip of the wing-part. *Movable-end-direction* ignored. If equal to (x y) the movable starts at x·(wing-part length) and stops at y· (wing-part length). If equal to (0 1) the movable starts at the root and ends at the tip of the wing-part. *Movable-start/end-direction* ignored
End-movable	0.7	

Table 5.4: input parameters for movable definition.

Nonetheless, the achieved level of modeling accuracy has proven adequate both for the use of simple panel codes, as well as for higher fidelity aerodynamic tools, such as the NLR CFD simulation tool ENFLOW (van Houten et al., 2005; van den Branden, 2004). In fact, when the goal is not computing the flow through the wing/movable gap or slot or around the cut off area of the movable, but obtaining a reasonable estimation of the overall pressure distribution, it is common practice to model lifting surfaces with deflected movables just as continuous surfaces.

The Capability Module `TEMovableSimulator` has been developed to give the Wing-Part HLP the extra capability to model movables (see the UML diagram of Fig. 5.1). The implemented modeling approach is summarized below, with support of Fig. 5.14.

1. The unperturbed Wing-part instance surface is created based on the curves generated by the `WingSection` class (Fig. 5.14-*a*).
2. The position and the orientation of the hinge line and the cut off planes that determine the movable planform shape are evaluated based on the user defined input values (Fig. 5.14-*b*)
3. The `MovableSupportSection` class intersects the unperturbed wing-part surface with planes delimiting the movable side edges (Fig. 5.14-*c*). Although only one cutting plane is shown in the figure, four very close parallel planes are used to intersect the Wing-part surface at each movable side edge: two inboard and two outboard of the given edge. These four intersection curves, are necessary to "support" the operation at point 5.
4. All the unperturbed wing sections generated by `WingSection` plus the new support curves are collected by the `WingSectionForMovable` class, which submits them to the `AirfoilDeflector` Capability Module. `AirfoilDeflector` selects the curves that belong to the movable area and deflects their trailing edge as illustrated in Fig. 5.14-*d*: The selected curves are cut in correspondence of the hinge line and the TE curve segments rotated around the user defined hinge axis by the user defined *rotation angle*. The generated gaps and overlaps are automatically sealed, trimmed and filleted as shown in figure.
5. The final Wing-part surface is generated by lofting across the new set of deflected and unperturbed wing sections. The four support curves generated at each movable edge (of which two are deflected and two not) help the lofting process to produce a smooth and regular surface, even in case of large deflection angles. See the close up in Fig. 5.14-*e*.

The other three possible movable definition cases are shown in Fig. 5.14-*f*.

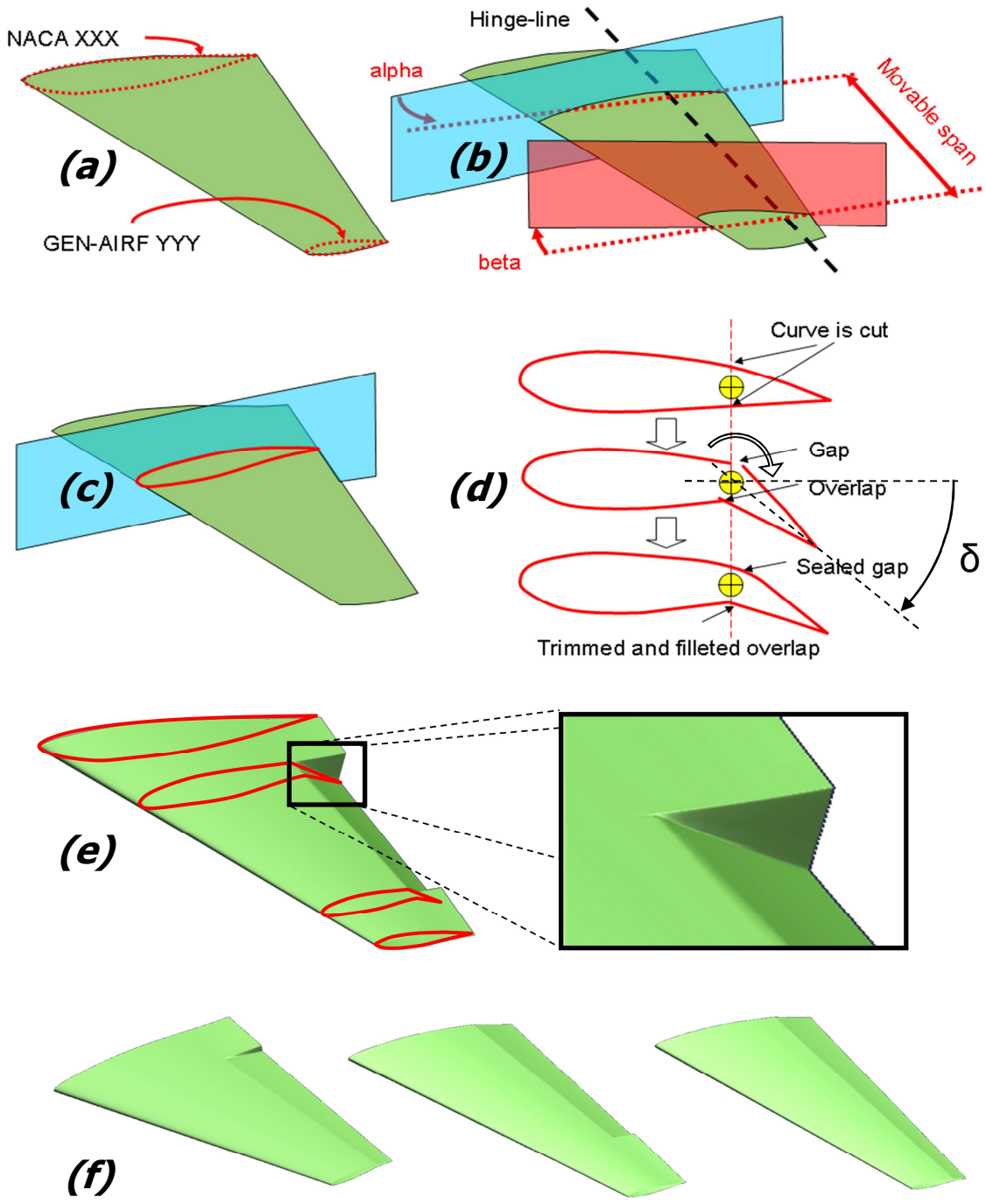


Fig. 5.14: Step-by-step modeling process for wing-part surfaces with TE control movable (a-e), with detail of the output geometry quality. Example of possible wing-parts with movable configurations (f).

Another two movable modeling approaches have been implemented during this research work. The second approach has been specifically developed to link the MMG to the VSAERO panel code. There is no actual wing-part surface deformation required by KBE system. A dedicated Matlab preprocessing module, called COALA, has been developed to deflect the movable panels, directly in the VSAERO preprocessing environment. To do that, COALA needs from the MMG a dedicated XML file containing the discretization of the whole aircraft geometry and information concerning the exact position, planform shape and deflection angle of the movables. Details can be found in (Grotenhuis, 2007; Brouwers, 2007; Dircken, 2008; van Dijk, 2008). The approach developed to enable the MMG-COALA-VSAERO link will be further addressed in Chapter 6.

A third approach has been developed to account for the structural design and manufacturing aspects of control movables in the preliminary design phase. A separated KBE application, called PMM (Parametric Movable Model) has been developed by van der Laan (van der Laan, 2008), which is a kind of MMG[1] dedicated to the design of aircraft movables. The MMG Capability Module `TEMovableSimulator`, on the base of movable definition input, cuts the *undeflected* wing-part surface around the movable boundaries (hinge line and side edges), then exports this trimmed surface, together with some other metadata and geometrical information to the PMM (van Houten et al., 2005). The latter is then responsible to generate a detailed model of the movable, whose shape is fully consistent with the *master geometry* produced and exported by the MMG. For the structural analysis of the movable, the aerodynamic loads can be extracted by the VSAERO model generated by the MMG/COALA system. More information about the MMG-PMM collaboration can be found in Chapter 6.

[1] The PMM KBE application actually shares a number of HLPs and CMs with the MMG.

5.3 Wing-part Structure definition

In this section the generative approach implemented to design the wing-part structure configuration is thoroughly described. The modeling system developed allows for the design of a conventional spars & ribs structural concept. All components are modeled using surfaces, no solids are used. The achieved level of modeling detail is adequate for the conceptual/preliminary design phase. Details as flanges, cleats, doublers, access holes etc. are beyond the scope of the current modeling capabilities. Nevertheless, the designer is provided with a lot of freedom and design flexibility, due to the full parametric definition of all the components.
The UML use case shown in Fig. 5.15 illustrates the main functionalities, and their relations, required from the structural modeling system. Eventually, the main requirements can be summarized as follow:

1. Provide the option to separate each wing-part in three main areas, i.e., the leading edge (LE), the wingbox (WB) and the trailing edge (TE) areas, to account for their different structural design requirements.
2. Provide a single generic wing-part primitive that allows to model the abovementioned three areas as physically separated elements (including separations by gaps) and allows the inclusion of movable components (e.g., ailerons, rudders, etc.).
3. Provide optional generation of spars and ribs in each of the three abovementioned wing-part areas, with the freedom to position and orient each structural element individually.

The UML activity diagram in Fig. 5.16 shows *how* the structure generative process has been implemented.

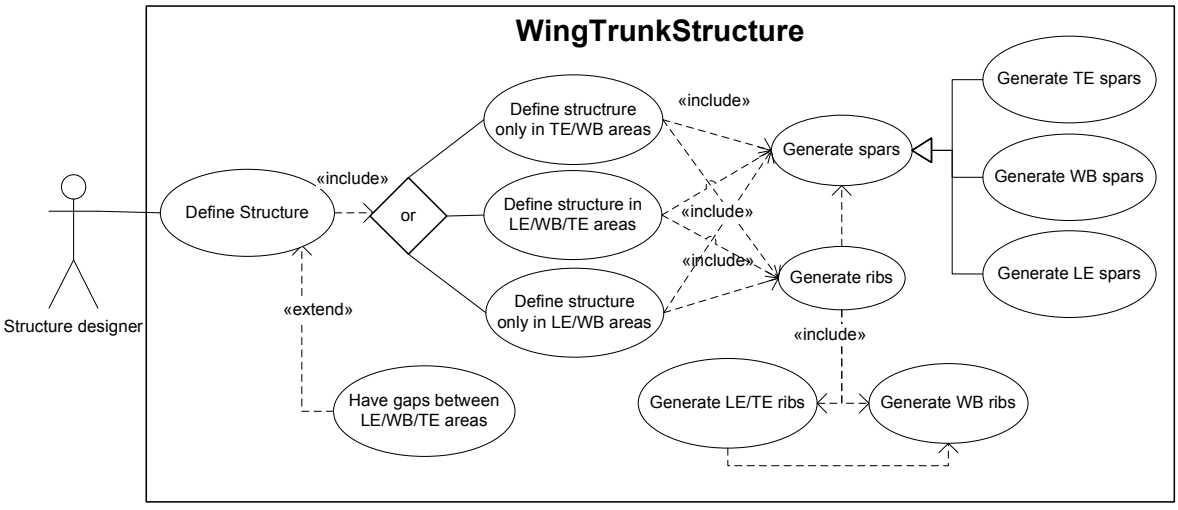


Fig. 5.15: the WingTrunkStructure system basic use case.

The specific MMG module responsible for the structure generation is the `WingTrunkStructure` class: together with the `WingTrunkSurface` class, one of the two main components of the Wing-part HLP.
As shown in Fig. 4.18, `WingTrunkStructure` depends on the wing-part surface generated by `WingTrunkSurface`. Indeed, the first activity in the whole structure generation process (Fig. 5.16) is "*generate outer surface*", whose details have been addressed in the previous section.
The activity diagram in Fig. 5.16 can be used also as a guide to the content of the next sections. As usual, the reader can refer to Appendix A for a summary of the basic UML notation. Appendix F provides the detailed class diagram of the whole `WingTrunkStructure` aggregation.

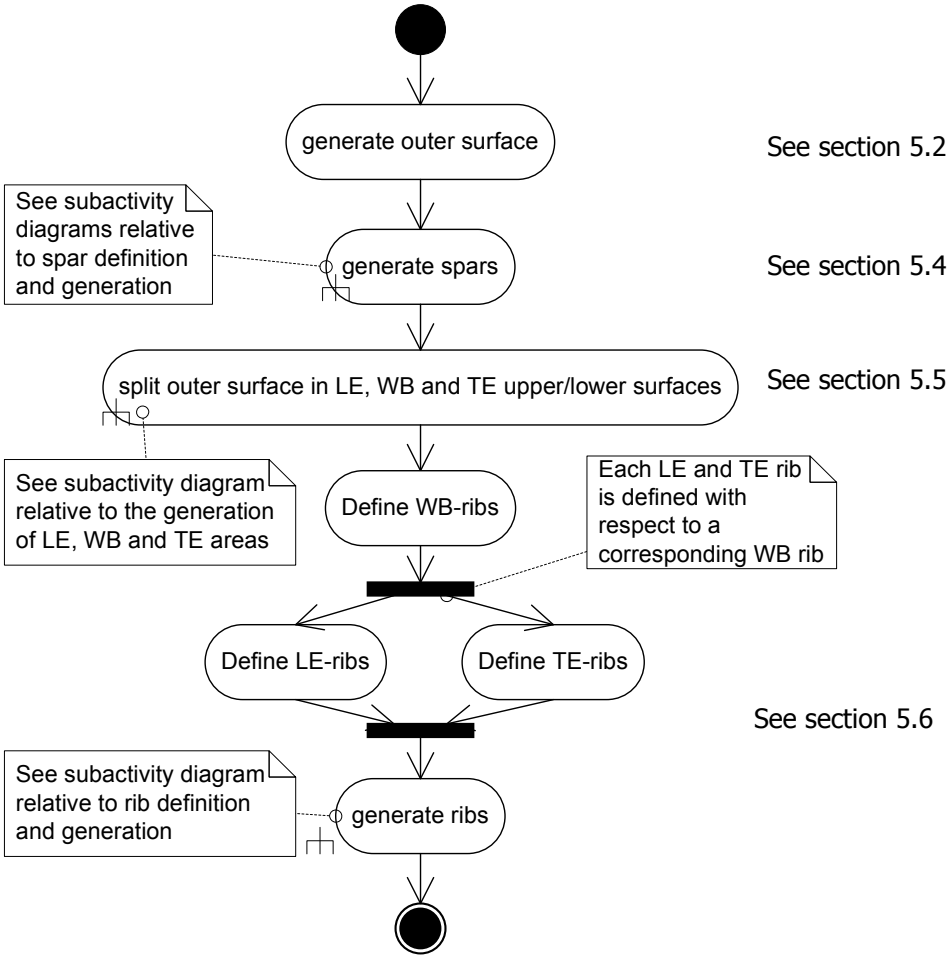


Fig. 5.16: UML diagram showing the main activities involved in the generation of the wing-part structure configuration.

5.4 Spars definition

The spars definition use case of Fig. 5.17 (limited to the spar definition process within a Wing-part) extends and details the "generate spars" use case of Fig. 5.15. Note the two imposed *constraints* to the generation of spars:

1. Spars must be planar (i.e. no warped spar webs allowed)
2. The shape of the spars must be tailored to the outer wing-part surface (hence they should be adaptive to the eventual changes in the given aerodynamic shape)

And the following requirements:

- It should be possible to define spars in all the three LE/WB/TE areas
- It should be possible to define spars using two different positioning methods, i.e. *point-to-point* and *point-and-angle* (see explanation below)
- It should be possible to define special kinds of spars (e.g., virtual spars) and/or assign them special functionalities (e.g., delimit the volume of the wing fuel-tanks)

The definition of the spars is of great importance for the specification of other structural components in the wing-part. The physical division of the wing-part in the three LE/WB/TE areas depends on the actual location of the spars (details in section 5.5) and spars are also used to provide a positioning reference for the ribs in the wing box and LE/TE areas (details in section 5.6).

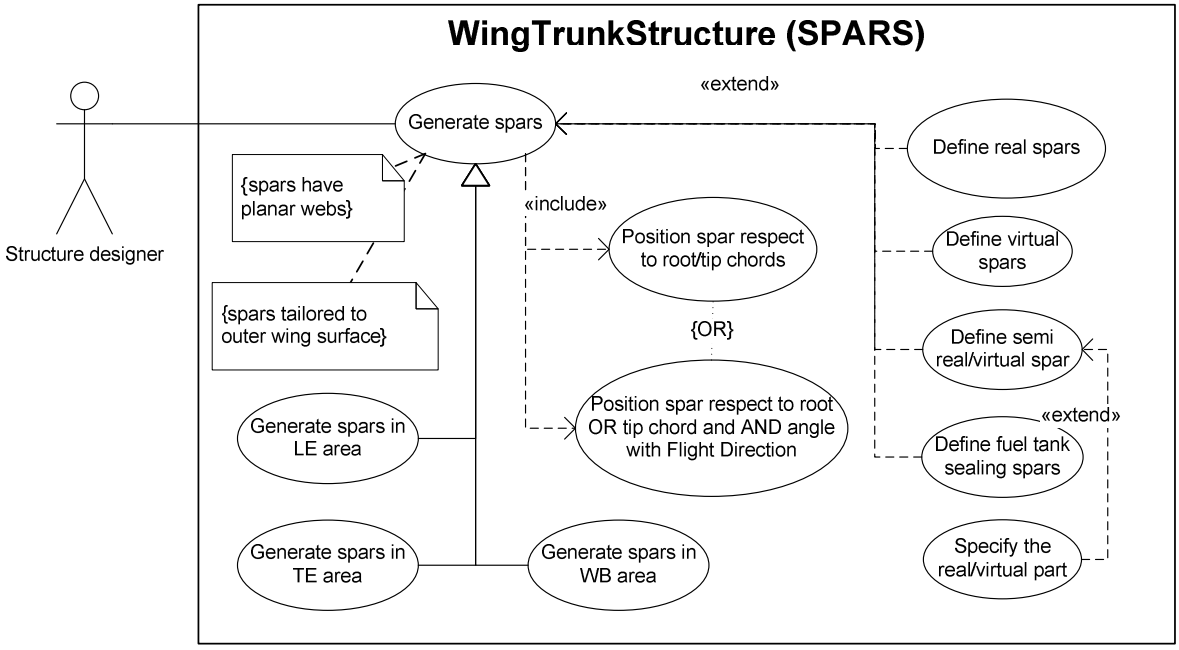


Fig. 5.17: WingTrunkStructure (SPARS) system use cases

5.4.1 Spars generative process

All the LE/WB/TE spars are generated through the following sequence of activities:

- Two points per spar, called *spar-points*, are identified on the root and tip chords of the given wing-part surface, by means of the user defined parameters *spar-offset-list-root* and *spar-offset-list-tip*.
- A straight line, called *spar-line*, is drawn through these points.
- The spar-line is projected along the LRS-vertical axis, onto the upper and lower part of the wing-part surface.
- Finally, the spar surface (actually the spar web surface) is generated by the linear interpolation of the two projection curves (i.e., *upper-spar-curve* and *lower-spar-curve*). The ICAD geometry primitive *ruled-surface* is used for the scope.

The spar generation procedure is illustrated in Fig. 5.18.

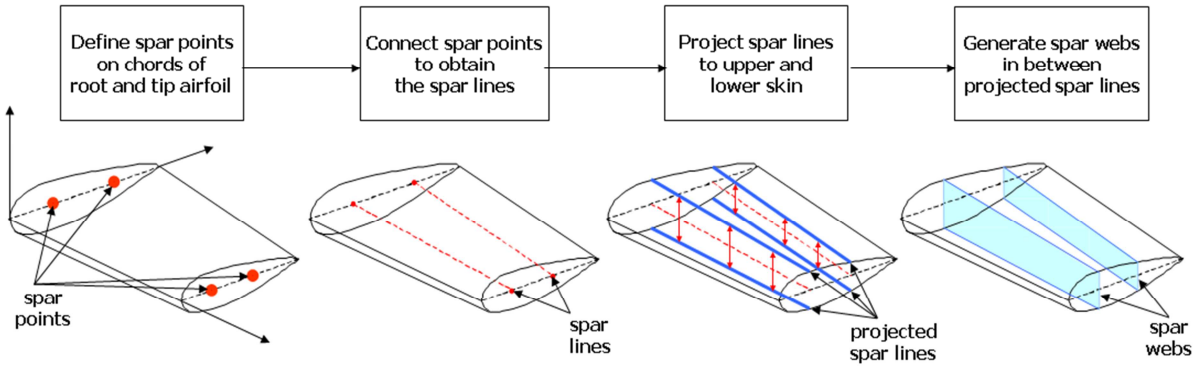


Fig. 5.18: Step-by-step generation process of spars geometry.

In order to define the spar points on the root and tip chords of the wing-part, the user can choose between two methods, so-called *"point-to-point"* and *"point-and-angle"*.

With the ***point-to-point*** method, the user indicates directly the positions of the root and tip spar-points as fractions of the root and tip chord length, respectively.

With the ***point-and-angle*** method, the user indicates the position of one of the two spar-points as fraction of the relative chord length, and the angle that the given spar should form with the flight direction vector. In this case, the MMG will automatically derive the coordinates of the second spar-point and proceed with the same generative process as above.

By using the point-to-point method, the orientation of the spars is always affected by changes in chord lengths and sweep angle. In case the orientation of the spars should not change with wing sweep and taper ratio, then the *point-and-angle* method should be used. See examples in Fig. 5.19 taken from the MOB study (La Rocca and van Tooren, 2002a).

In both cases, chord fractions and/or angle values are assigned using the same input parameters: *spar-offset-list-root* and *spar-offset-list-tip*[1]. Table 5.5 reports the list of the parameters required to define spars in the TE/WB/LE areas of a given wing-part. Examples and notes for the user are included.
A snippet of the MMG input file, with examples of spars definition is provided in Appendix G.
The details (sub-activities) of the "generate spars" activity of Fig. 5.16, can be found in Appendix E.

Parameter	Example	Description
*-type-of-spar-XX	(list 'r … 'v)	• 'r for real spar • 'v for virtual spar • (list 'r A B) for a spar that is real from the fraction A to the fraction B of the spar length • (list 'v A B) for a spar that is virtual from the fraction A to the fraction B of the spar length • 'f for a fuel tank spar.
*-spar-offset-list-root-XX	(list 0.2 … 0.6)	A fraction of the chord length for *point-to-point* spar positioning (i.e., a number in the range [0,1]), or an angle value (expressed in degrees and > 1) for the *point-and-angle* positioning.
*-spar-offset-list-tip-XX	(list 0.2 … 90)	
NOTES: * stands for the given lifting surface configuration, e.g. wing, fin, winglet, etc. XX stands for LE, WB or TE. A, B are numbers in the range [0 1] indicating a fraction of the given spar length		
RULES: The three lists must have same length. The values of *-spar-offset-list-root-XX and *-spar-offset-list-tip-XX defining a given spar cannot be simultaneously > 1 (i.e., both angles).		

Table 5.5: input parameters for spars definition.

5.4.2 The spars generative approach: rules, capability and limitations

- Since the projected spar-lines used to define the spar surface are projection-curves on the wing-part surface, the shape of the spars is always tailored to the surface generated by the `WingTrunkSurface` class.

[1] As shown in Table 5.5, there is no additional switch/parameter to indicate whether the point-to-point or point-and-angle method should be used. When the user defines a spar-offset value larger than one, the system assumes automatically that the point-and-angle method is to be triggered. This "trick" simplifies (and limits the amount of parameters in) the input file, at "the cost" of excluding the possibility to position a spar at a null angle with respect to the LRS-longitudinal axis, hence *parallel* to the wing sections…

- There is no limit to the maximum amount of spars that can be defined in each wing-part and each wing-part area (LE/WB/TE). The designer can add/remove spars in any given wing trunk, just adding/removing the relative values in the MMG input file lists.
- A minimum of two spars is required in the WB area.
- The MMG assigns an index number to each spar, which starts from zero *for the first spar defined in the LE-spars-offset-list* and increases till *n* for the last spar in the *LE-spars-offset-list*. Then the index goes from *n+1* to *m* from the first to the last spar in WB-spars-offset-list. From *m+1* till *k* from the first to the last spar in *TE-spars-offset-list* the same list and so on. See example in Appendix G. This index number is used to address the single spars, when used as reference for the ribs positioning and orientation procedure.
- In the same wing-part, spars can be generated both with the *'point-to-point'* and *'point-and-angle'* methods.
- More spars can start/end in the same points, but are not allowed to cross each other. To define a spar running out onto another spar, the correct approach is defining two contiguous wing-parts and let the two spars just meet at the interface (see Fig. 5.20).
- Spars must extend from the root to the tip section of one wing-part area (see Fig. 5.20).
- No curved spars can be generated
- Spars are not warped even if generated inside twisted wing-parts, because the *spar-lines* are projected along the direction of the vertical axis of the Wing-part local reference system (defined in section 5.2.4).

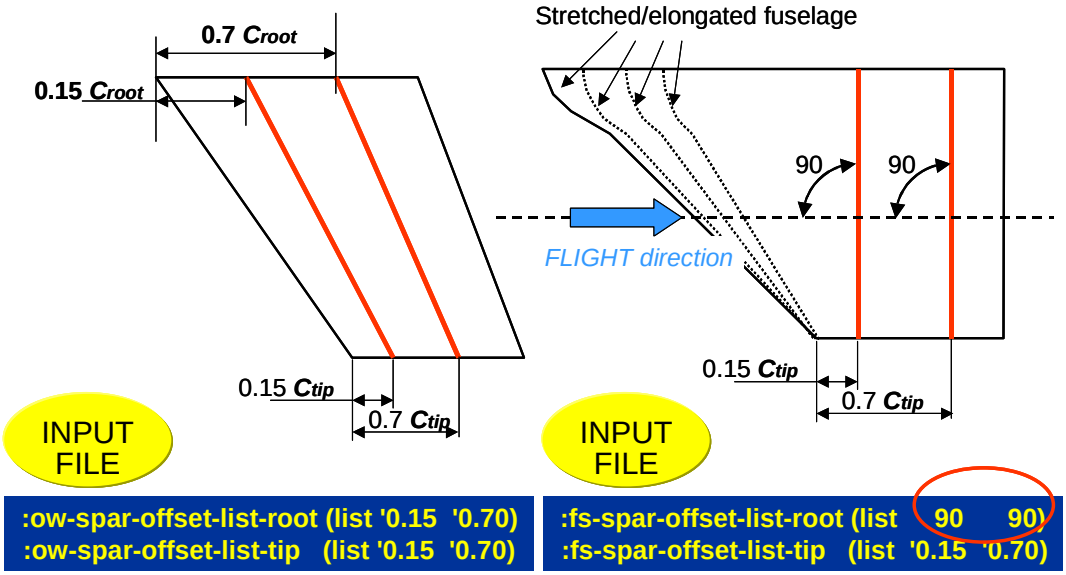


Fig. 5.19: Examples of the two spars positioning procedures: Point-to-point (left) and Point-and-Angle (right).

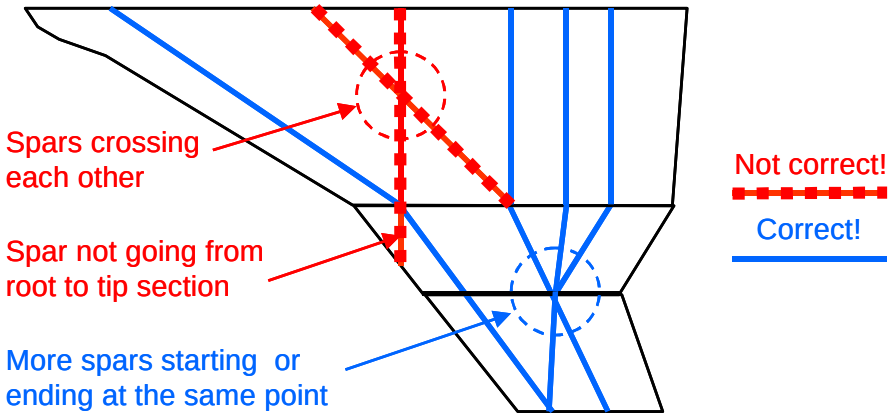


Fig. 5.20: Example of valid and wrong spars definitions.

5.4.3 Definition of real, virtual, semi real/virtual spars

Via the input file parameter *type-of-spar,* users can associate to each spar one of the specific properties/functionalities indicated in Table 5.5.
Of particular interest are the so-called *virtual spars*. They are defined in the same way as the real spars. Like real spars they can be used as positioning reference for ribs and they contribute to the wing-part segmentation process to support FE analysis, addressed in Section 4.7.1 and thoroughly described in Chapter 6. The difference is that they are not accounted as true structural elements; as such, they are excluded from any model generated for structural analysis support.

Semi-virtual or semi-real spars (refer to Table 5.5) can be used to model wing-part configurations where, for example, one spar out of three runs out at a certain span fraction. Without virtual spars the user would be forced to define two adjacent wing trunks, one with three spars and the other with two. However, this solution might create problems later, during the pre-processing phase for FE analysis (see more in Chapter 6). The definition of a virtual spar in the second wing-part, positioned as extension of the one running out, would solve the problem. However, a more convenient solution, which would not even require the overhead of defining two wing-parts, is that of using a *semi-real spar*, hence a spar which is real only for a fraction of its own length. An example of a semi-real spar is shown in figure Fig. 5.24.

5.5 Definition of Wing box, Leading Edge and Trailing edge areas

Once the spars have been defined, `WingTrunkStructure` can start the process of surfaces intersection to segregate the wing-part surface generated by

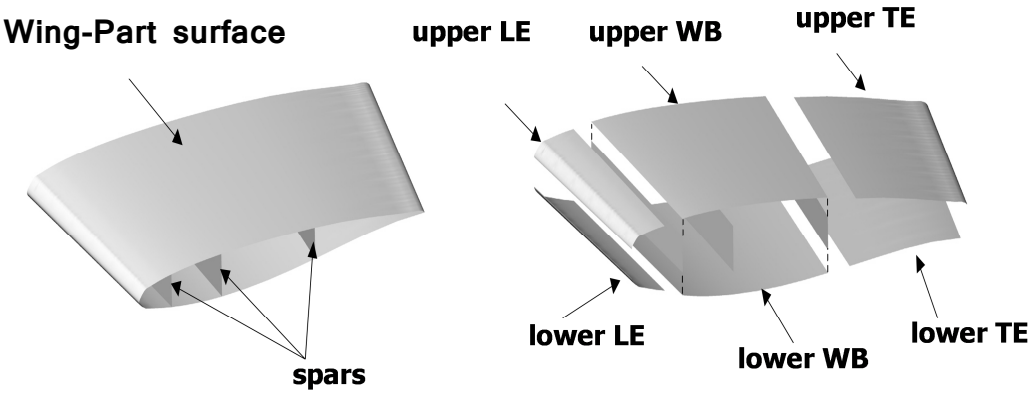


Fig. 5.21: Wing-part surface intersection in LE/TE/WB areas and upper/lower panels

`WingTrunkSurface` according to the three leading edge, trailing edge and wing box areas. For each area an upper and a lower skin panel is obtained (Fig. 5.21).
The procedure can be summarized as follow (refer to Appendix H for the detailed UML activity diagram):

1) `WingTrunkStructure` selects in the list of user-defined WB spars the two spars closest to the LE and TE lines of the wing-part. These two spars, no matter if real or virtual, will determine the wing-box area.
2) Then `WingTrunkStructure` checks the list of LE spars. If the list is empty:
   - No leading edge structure will be generated at all (similar procedure for the TE area).
   - Otherwise, the LE spar farthest from the leading edge curve will determine the extension of the LE area. In case this spar is not coincident with the first spar in the WB area, *a physical gap* between the LE and WB area will be generated (see examples in Fig. 5.22). When the last LE spar and the first WB spar coincide, one of the two can be defined as virtual spar (similar procedure for the TE spars).

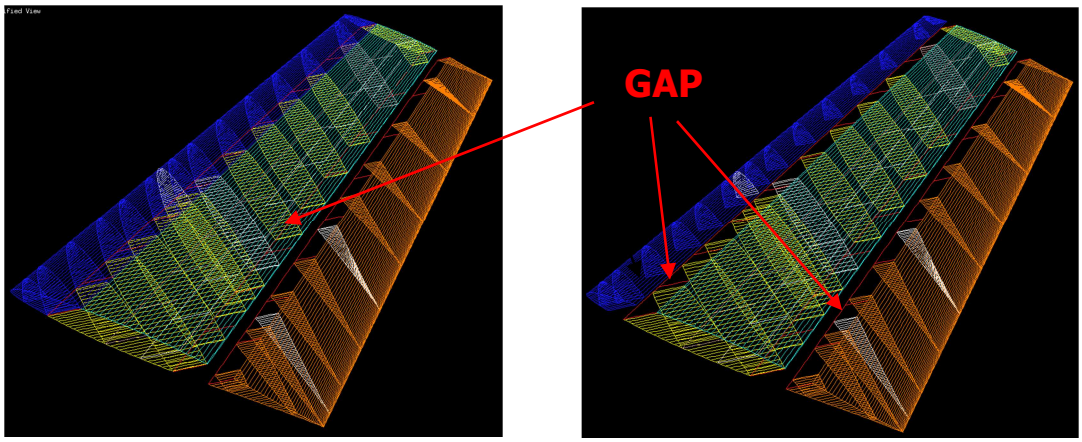


Fig. 5.22: examples of wing-parts with definition of LE/WB/TE areas

This modelling approach allows the generation of centre wing sections (the part of the wing that crosses the fuselage), where no LE and TE areas are present.
It also allows defining wing-part models for structural analysis that include a movable component. The gap allows the physical separation of fixed and movable wing parts, while simplified hinge brackets can be automatically generated in accordance with user-defined *hinge-ribs* (see next section).

5.6 Ribs definition

Similar to the modelling of the spars, the level of detail for the ribs is low but adequate for the evaluation of the *global* structural behaviour of wing-like systems, using FE analysis. All ribs are modelled as plain surfaces, without flanges and any kind of cut-out. Actually, what is modelled is just the web surface of each rib.
The requirement analysis for the rib modeling system has been summarized below. Refer to Appendix J for the complete UML use case.

User defined functionality: the designer should be allowed to perform the following operations:

- Define any number of ribs in any wing-part, with the possibility to position and orient each rib independently,
- Make use of any of the following items as references for the position and orientation of each rib: flight direction, LE and TE curves and any of the spars defined in the given wing-part.
- Define special kinds of ribs (e.g., virtual/semi virtual ribs) and/or assign them special functionalities (e.g., define boundary wall for fuel-tank, support hinge brackets for attachment of movables (control surfaces))

User defined *constraints* for the rib generation:

- The shape of the ribs must be tailored to the outer wing-part surface (hence they should dynamically adapt to any change in the surface generated by the `WingTrunkSurface` class)
- The ribs must lie on planar surfaces
- The planes containing the ribs must be parallel to the vertical axis of the Wing-part local reference system (defined in section 5.2.3)
- The ribs defined in LE/TE areas should have a *corresponding* rib (no matter if real or virtual) defined in the WB area, to be used as positioning reference (this follows from the need of generating valid segmented models for FE analysis. More information in Chapter 6)

5.6.1 Ribs generative process

All the ribs in the wing-box area (the WB-ribs) are generated through the following sequence of activities illustrated in Fig. 5.23 (point and vector definitions in next section):

- A WB-*rib-plane* is defined, which requires:
  - the definition of a point (the WB-*rib-point*)
  - the definition of a vector (the WB-*rib-vector*).
- The upper and lower wing-box skin panels are intersected by the WB-*rib-plane* to generate the WB-*upper-rib-curve* and the WB-*lower-rib-curve*
- The actual WB-rib surface is generated by interpolation between the two intersection curves generated as described above. As in the case of spars, the ICAD geometry primitive *ruled-surface* is used.

In order to generate LE and TE-ribs, the procedure is very similar to the one described above, with the only difference found in the definition of *rib-point*, which, in the case of LE/TE-ribs, is *automatically* calculated as the intersection of the *corresponding* WB-rib-plane, with the *spar-line* of the front WB-spar (back WB-spar for TE-ribs). The *rib-vector* of LE/TE-ribs is still assigned by the user as for WB-ribs.

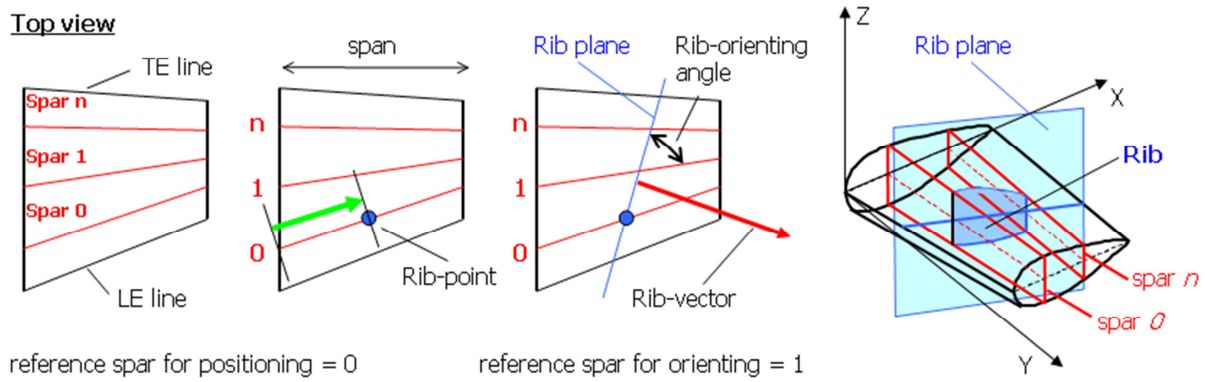


Fig. 5.23: Step-by-step generation process of a WB rib.

The positioning of the *rib-point* (only for WB-ribs) and the orientation of the *rib-vector* can be determined by the user by means of four specific parameters in the MMG input file:

1. *rib-positioning-referred-to-spar* (to be set only for only for WB-ribs)
2. *rib-positioning-offset-list* (to be set only for WB-ribs)
3. *rib-orienting-referred-to-spar*
4. *rib-orienting-angles-list*

Find in Table 5.6 for the detailed list and explanation of the rib definition parameters. A snippet of the MMG input file, with examples of rib definitions inside a multi wing-part vertical tail, is provided in Appendix I.

The way the four abovementioned parameters are used to define the rib-point and rib-plane for any rib can be summarized in the following four steps:

1. (Only for WB-ribs). The spar-line of spar *n* is selected as reference curve to position the *rib-plane-point.* The spar number *n* is assigned via the input parameter *rib-positioning-referred-to-spar*. The LE/TE lines can also be selected in place of a spar.
2. (Only for WB-ribs). The *rib-point* is found on *spar n* (or on LE/TE line), at a distance from the root (measured along the given spar or the TE/LE line) equal to a fraction of the length of spar *n* (or LE/TE), as indicated via the input parameter *rib-positioning-offset-list*
3. Another (or the same) reference *spar m* (or the LE/TE line or the flight direction vector) is selected via the parameter *rib-orienting-referred-to-spar*.
4. The spar line vector (pointing from root to tip point) of *spar m* (or the direction vector of LE/TE line) is then rotated around the vertical axis of the wing-part local reference system, by the angle (in degrees) defined by parameter *rib-orienting-angles-list.* Hence, the *rib-plane-vector* results determined.

Similar to the spar definition approach, a variant to the point-and-vector approach described above, the so-called point-to-point approach, has been developed, still based on the manipulation of the same four input parameters.
Only step 4 of the sequence above is affected as follows:

4. (point-to-point): in case a value < 1 is assigned to the parameter *rib-orienting-angles-list*, this value is not interpreted as an angle, but as a fraction of the length of *spar m* (or LE/TE line). Hence a *second point* is generated on *spar m* (or LE/TE line), at the root offset distance indicated by *rib-orienting-angle-list*. This second point and the previously generated *rib-plane-point* are then used to define the actual *rib-plane-vector*[2].

This (apparently) complex procedure to define and generate ribs has been thoroughly documented with a series of UML activity and sub-activity diagrams, available in Appendix K.

5.6.2 The ribs generative approach: rules, capability and limitations

- The number of ribs that can be defined in a wing-part is unlimited.
- There is no order in which ribs must be defined in the input file (e.g. from root to tip etc.)

---

[2] Similar to the spar case, this approach avoids complicating the input file with the addition of an extra switch/parameter. Similar to the spar case, the cost for the modeling flexibility is null: indeed, ribs oriented at angle smaller than 1 degree with respect to a spar are not realistic.

- Partial ribs can be generated, i.e., ribs that do not start and end in correspondence of a spar (or LE/TE line), but intersect the root and tip edge of the wing-part
- Ribs should not cross each other, however partial ribs can run out on ribs that are positioned at the root and tip section of a wing-part. (i.e., partial ribs).
- Similar to spars, virtual or semi-virtual/real ribs can be defined (refer to section 5.4.3 for the scope of virtual elements).
- A LE/TE-rib always needs a WB-rib in order to be generated (although the given WB-rib can be a virtual one)
- Vice versa, for each WB-rib, a LE/TE rib must always be defined in the input file. However, if a given LE/TE-rib is not required, it can be defined as virtual, or suppressed by setting the corresponding input parameter *type-of-LE/TE rib* = 'x. In this last case, the user must be aware that the rib will not affect the surface segmentation process to support FE analysis.
- Hinge ribs can be defined setting *type-of-LE/TE-rib* = 'h. A hinge rib triggers the generation of simple hinge brackets (La Rocca and van Tooren, 2002a) and affects the position of hinge slots in the movable models generated by the PMM (van der Laan, 2008; van Houten et al., 2005)
- It is possible to define WB-ribs with corresponding, not coplanar LE/TE ribs.

Fig. 5.24 shows some examples of modelled ribs with relative input parameter definition.

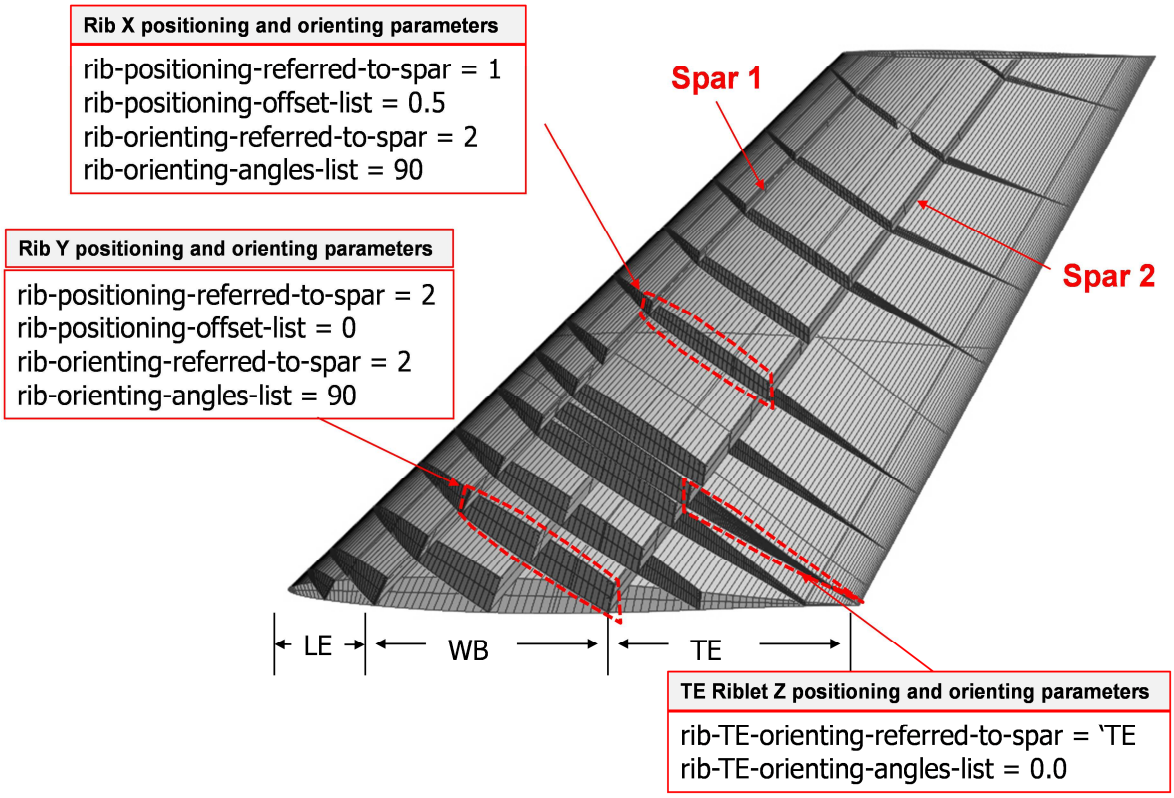


Fig. 5.24: example of the structure configuration of the fin of a commercial transport aircraft (with some ribs definitions).

Parameter	Example	Description
**-type-of-rib-XX*	(list 'r 'x … 'v)	• 'r for real (normal) rib • 'v for virtual rib • (list 'r A B) for a rib real from A·rib-width to B·rib-width • (list 'v A B) for a rib virtual from A·rib-width to B·rib-width • 'f for a fuel tank end rib. • 'h for a hinge rib • 'x (only for LE/TE-ribs) to suppress the given LE/TE-rib
**-rib-positioning-referred-to-spar*	(list 0 … 'LE 'TE)	• A spar identification number • 'LE for the LE line • 'TE to the TE line
**-rib-positioning-offset-list*	(list 0.1 … 0.2)	A fraction of the selected reference item (spar, LE or TE line)
**-rib-XX-orienting-referred-to-spar*	(list 0 'LE 'TE 'FD)	• A spar identification number • 'LE for the LE line • 'TE to the TE line • 'FD for flight direction
**-rib-XX-orienting-angles-list*	(list 0 90 … 0.2)	A number ≥ 1 to indicate an angle (in degrees). (*Point-and-vector* method) **OR** A number in the range [0 1] to indicate a length fraction of the element indicated by **-rib-XX-orienting-referred-to-spar*. (*Point-to-point* method)
NOTES: – * stands for the given lifting surface configuration, e.g. wing, fin, winglet, etc. – XX stands for LE, TE or nothing for WB-ribs. – A, B are numbers in the range [0 1], hence represent fractions of the rib width. – **-rib-positioning-referred-to-spar* and **-rib-positioning-offset-list* are not defined for LE/TE-ribs. – When **-rib-XX-orienting-referred-to-spar* = 'FD, the value of **-rib-positioning-referred-to-spar* is not relevant (but must be assigned anyway!) because all spars and LE/TE lines are straight. – When *-type-of-rib-LE or *-type-of-rib-TE = 'x, the corresponding values of *-rib-XX-orienting-referred-to-spar and *-rib-XX-orienting-angles-list are not relevant (but must be assigned anyway!).		
RULES: – The length of all the lists must be equal. – With the point-to-point rib positioning method, the corresponding values of **-rib-positioning-referred-to-spar* and **-rib-XX-orienting-referred-to-spar* must be different.		

Table 5.6: input parameters for ribs definition

5.7 Implementation of the Connection-Element High Level Primitive

Complex wing configurations, like multi-kinked wings with discontinuities in sweep and/or dihedral angles can be defined using multiple wing parts.
When two adjacent trunks, *k* and *k+1*, are used with the same dihedral angle, and the tip airfoil of wing-part *k* is the same as the root airfoil of wing-part *k+1*, the overall wing surface results "sealed" at the trunks' interface automatically. The resulting surface might be discontinuous (a kink in the wing), but is still watertight[3].
However, when a *dihedral angle* change occurs across two adjacent wing-parts, the relative rotation of their local reference systems (Fig. 5.4) generates *gap* and *overlap* areas in the wing surface, which needs to be resolved to restore the watertight condition. Also the disrupted continuity of the spars needs to be resolved[4].

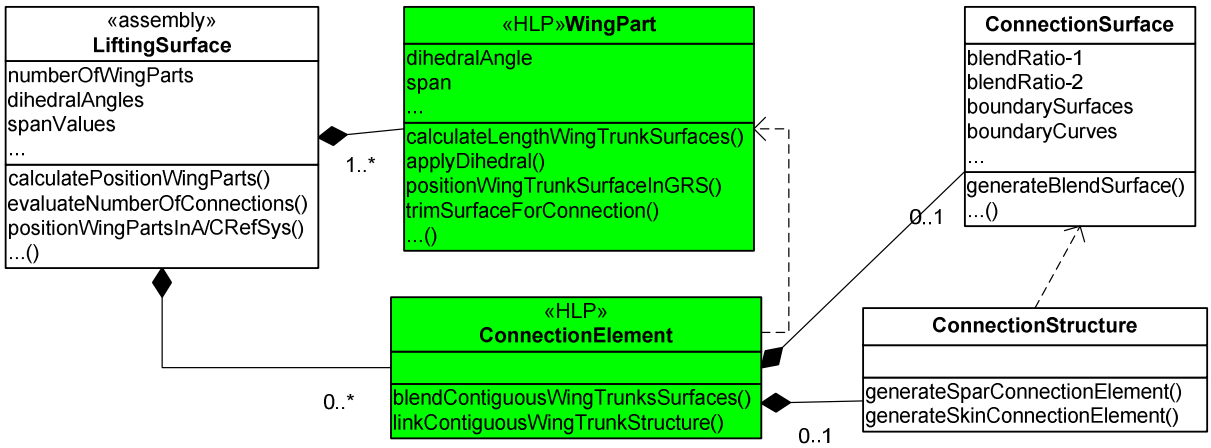


Fig. 5.25: UML class diagram showing relationships and architecture of the Connection-element HLP

As shown in the class diagram of Fig. 5.25, it is the `LiftingSurface` class that checks for variation of dihedral angle between adjacent wing-parts and automatically demands the necessary number of instantiations of the `ConnectionElement` class. Similar to Wing-Part, the Connection-Element HLP is a

[3] In the field of computer graphics, these cases are said to be *g0 continuous*, but *g1 discontinuous*. G0 is the term used to indicate continuity of position, g1 continuity of tangency (and g2 continuity of curvature).

[4] The possibility of restoring the continuity of ribs that cross Wing-part instances with different dihedral has not be considered here. In case of dihedral discontinuity, the most common structural design solution is to place a rib along the discontinuity rather than crossing it. In the case of a rib crossing two adjacent Wing-part instances that are dihedral continuous, the user will have the possibility to define two partial ribs (one per wing part) and take care of their relative positioning.

composition of two modules, namely the `ConnectionSurface` and `ConnectionStructure` classes, whose functionalities will be addressed in the following sections.

5.7.1 Wing-parts connection surface definition

The generative process to build a connection surface between two contiguous wing-parts consists of two main phases:

1. Trimming of the two contiguous wing-parts extremities (in order to cut away the "problem area")
2. Generation of a blend surface as actual instantiation of the `ConnectionSurface` class.

As shown in the UML diagram Fig. 5.25, `LiftingSurface` class has the responsibility to evaluate the required number of `ConnectionElement` instantiations and to demand the subsequent *trimming phase*. The `ConnectionSurface` class is responsible for the final generation of the connection surface, which takes place during the *blending phase* of the process.

Trimming phase (Fig. 5.26-top)
Given the two adjacent wing-parts (addressed in figure as WT-1 and WT-2), positioned with different dihedral angles:

1. Trimming *Curve-1* is determined by intersecting the WT-1 surface with a plane parallel to WT-1 tip-airfoil and tangent to WT-2 root airfoil.
2. Trimming *Curve-2* is determined by intersecting the WT-2 surface with a plane parallel to WT-2 root-airfoil and tangent to WT-1 tip airfoil.
3. WT-1 and WT-2 surfaces are trimmed using *Curve-1* and *Curve-2,* respectively. In this way the surface areas at the tip of WT-1 and at the root of WT-2 are removed, and so are the gap/overlap problems.

Blending phase (Fig. 5.26-low)
Once the gap/overlap area has been cleared, a surface connection element is generated to blend the two trimmed wing-parts and restore the watertight surface condition. `ConnectionSurface` inherits its "blending capability" from the ICAD geometry primitive Edge-Blend-Surface (Knowledge Technologies International, 2001b), for which the following inputs are required:

1. The trimming curves *Curve-1* and *Curve-2* defined above.
2. The *WT-1* and *WT-2* wing-parts surfaces (to provide the tangency condition of the blend)
3. The two parameters *ratio-1* and *ratio-2*, which are required to define the relative influence the WT-1 and WT-2 surfaces have on the blend shape.

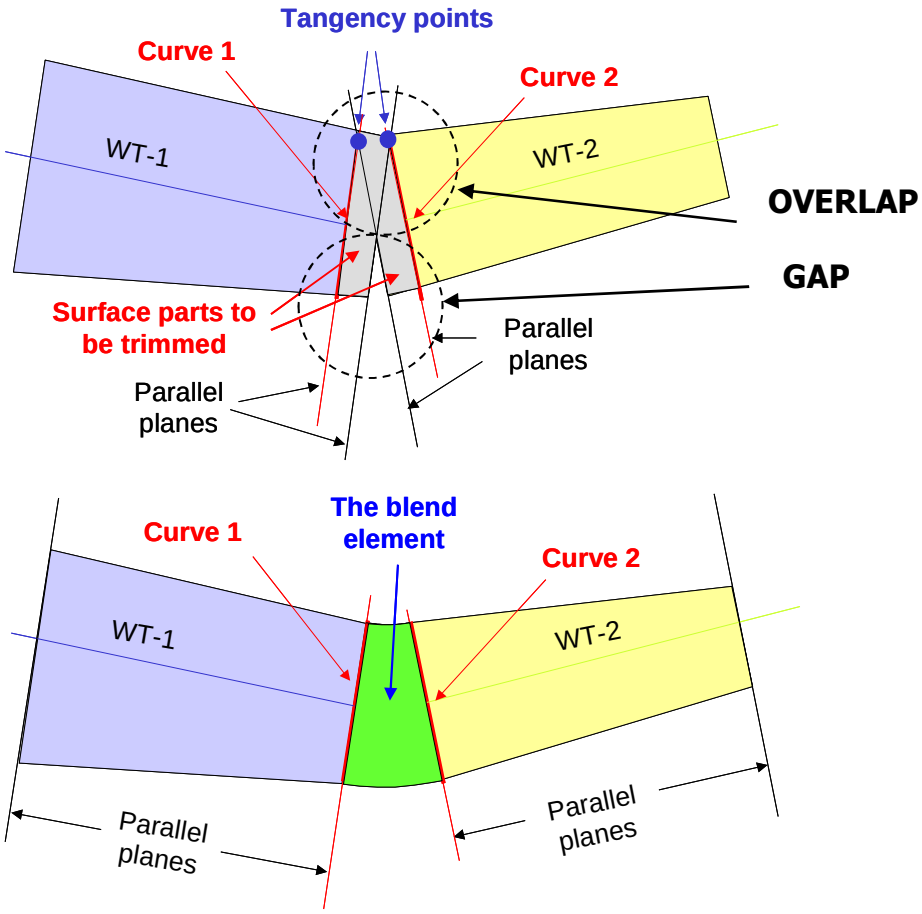


Fig. 5.26: trimming and blending operations to build a connection element

The first two sets of inputs are *automatically* computed by `ConnectionSurface`. Whatever the shape of the two adjacent wing-parts and the change in dihedral angle, the trimming phase will trigger and prepare the surfaces for blending. The definition of the third set of inputs to adjust the fullness of the blend remains the responsibility of the user. The ratio parameters Ratio-1 and Ratio-2 are available via the MMG input file and can take a value between zero and one. Values close to one will enforce the blend surface to maintain as far as possible the shape of the corresponding wing-part surface, which might lead to wavy connection surfaces because of the too high gradients of curvature (Fig. 5.27 top-left, top-right). On the other hand, a null ratio value nullifies the influence of the relative surface; also the tangency condition is then no longer enforced (Fig. 5.27 low-left). In general a value around 0.3 for both ratios, guarantees the best blend (Fig. 5.27 low-right).

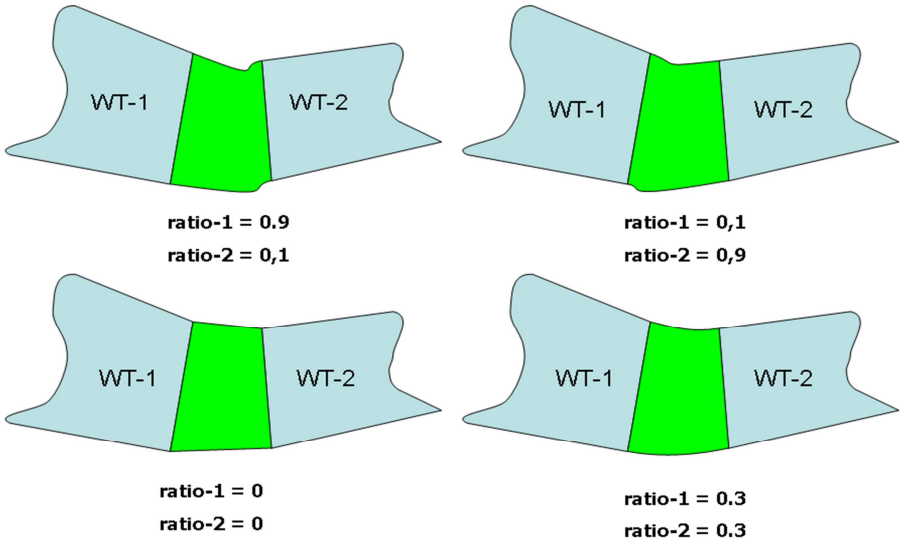


Fig. 5.27: Effect of blending ratios on the shape and tangency conditions of `ConnectionSurface`

5.7.2 Wing-parts connection structure definition

The role of the `ConnectionStructure` is to restore the physical continuity of the structural elements (i.e. skin panels and spar elements) across adjacent wing-parts, when this is lost because of the relative wing-parts rotation. The level of detail of the structural connection is basic, and the scope is to guarantee load path continuity, when performing a global structural analysis of a wing-like system. There is no intent to model and investigate the effect of different technical solution for the connection.

The generative process for the structure connecting elements can be summarized in the following steps (refer to the illustrations in Fig. 5.28 for the terminology used below):

1. The `ConnectionStructure` class gets as inputs the tip *edge-curves* of wing-part *k* and the root edge-curves of wing-part *k+1*. These edges-curves are computed by the `WingTrunkStructure` module of the Wing-part HLP.
2. The edge-curves are used to generate a series of *blend-segments*, with the same approach – and the *same blend ratio values* – as used for the generation of the connection-element surface[5]. The blend-segments recreate the continuity between the skin panels of the adjacent wing parts

[5] In the Wing-part HLP, the *associativity* between outer surface and inner structure is achieved by generating first the outer surface and then using it as "mold line" to define spars and ribs (see the spar-lines projection and rib-planes intersection methods described in 5.4.1 and 5.6.1). In the case of the Connection-element HLP, the inner structure is not built by intersecting/projecting on the outer connection surface. However, the associativity is still guaranteed by generating the blend-segments

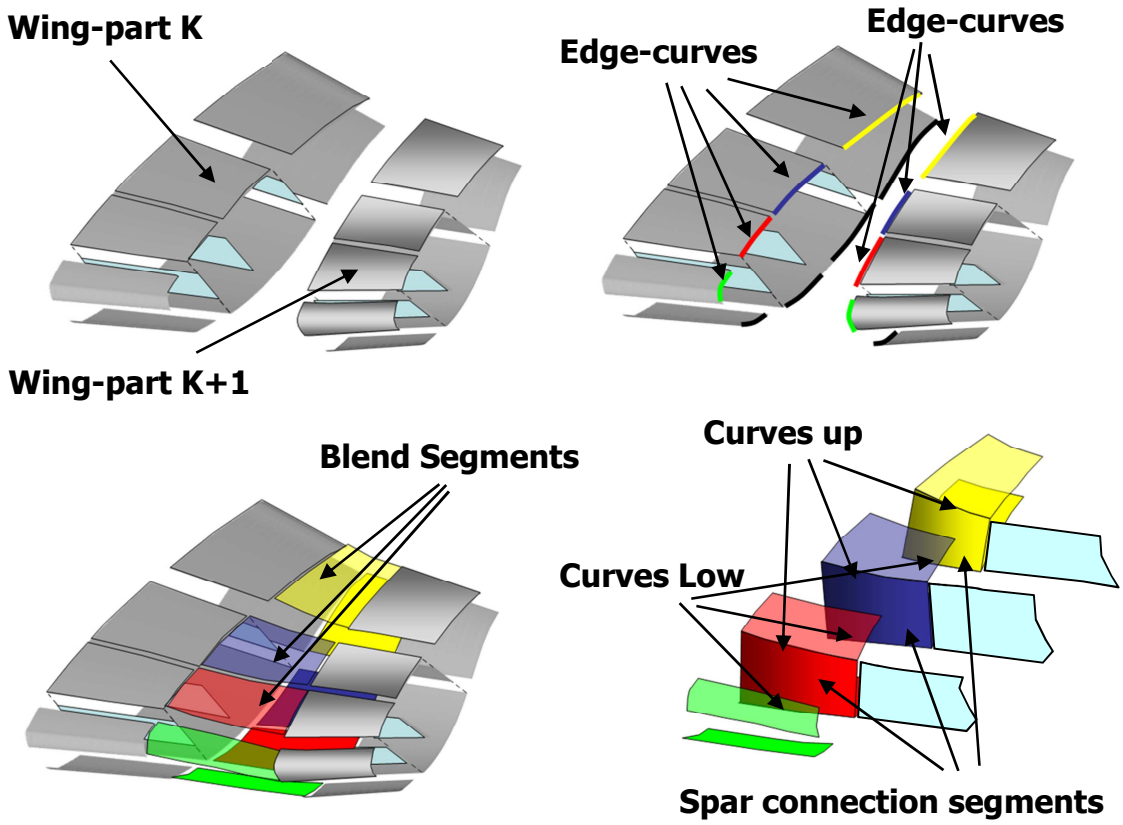


Fig. 5.28: generation of the connection elements between two contiguous wing-parts.

3. The longitudinal boundary curves of the blend-segments (indicated as Curves up/low in figure) are then used to generate, by linear interpolation, the surface of the *spar-connection-segments* (again, the ICAD primitive Ruled-Surface is used). This recreates the continuity between the spars of the adjacent wing parts

In Fig. 4.7 of Chapter 4, an example of restored spar continuity between wing and winglet is shown.

The success and the quality of the process described above depend strongly on the correct definition of the spars in the adjacent wing-parts that have to be connected. Fig. 5.29 shows examples of valid and bad spar positioning for the MOB BWB. The figure shows that it not necessary that adjacent wing-part have the same amount of

---

with the same blend ratios, (part of) the same reference curves and (part of) the same reference surfaces, as used to generate the connection-element surface. In fact, stitching the various blend-segments together would lead exactly to the same edge-blend surface of section 5.7.1.

spars. It is important that the number of upper and lower edge-curves (Fig. 5.28) match properly.
A control system has been implemented inside `ConnectionStructure` to check the validity of the model configuration before triggering any connection generation. If the check signals an incorrect situation, the generation of the structure connection elements is skipped to avoid crashing the MMG session.
Rules have been implemented such that a *virtual* spar-connection-segment is automatically generated if the connecting spars are *virtual*.

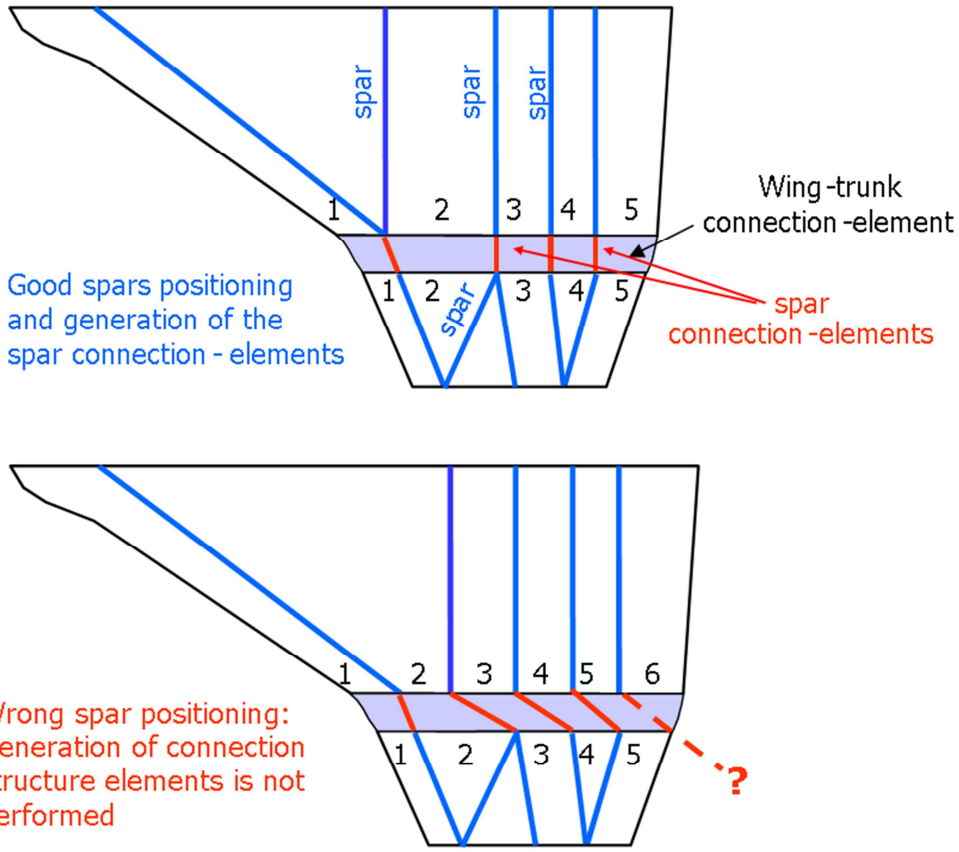


Fig. 5.29: Example of valid and wrong structural modeling for the generation of structure connection elements between adjacent wing-parts.

5.8 Towards a unified connection-element

Other types of connection elements are required to deal with different situations than adjacent wing-parts with dihedral angle discontinuity. Connections to link horizontal and vertical tail planes in cruciform, T-tail and H-tail configurations and connections to link wings, canards and tail empennages with the fuselage require the development of separate classes. Some of these have been developed (besides the

one detailed in 5.7.1) in this research work; mostly to enable the generation of accurate aerodynamic models for a range of different aircraft configurations. While we have progressed towards almost one generalized surface-connection element, the structure connectivity aspects have been typically solved with ad-hoc solutions (Meijer, 2003; Cerulli et al., 2006; Cerulli et al., 2005) and will not be addressed here.

The generation of a connection surface, whatever the type of elements to be connected, always requires the following two main steps:

1. *Preprocessing of the parts to be connected*
2. *Generation of the connecting surfaces*

The trimming and blending phases described in Section 5.7.1 are just a demonstration.

The scope of the first phase is to deliver two proper curves, which can be used in the second phase to build an appropriate surface.

The differences between the various connection-generation cases are found mainly in the preprocessing methods used to derive the two curves (which again depends on the relative position of the parts to be connected), and the sort of connecting surface that has to be generated.

Table 5.7 provides a matrix with the most recurrent connection cases when assembling an aircraft with HLPs. For each case, a list of typical preprocessing activities and possible types of surface generation are indicated based on experience. The preprocessing operations are not listed in a logical or required order; neither are

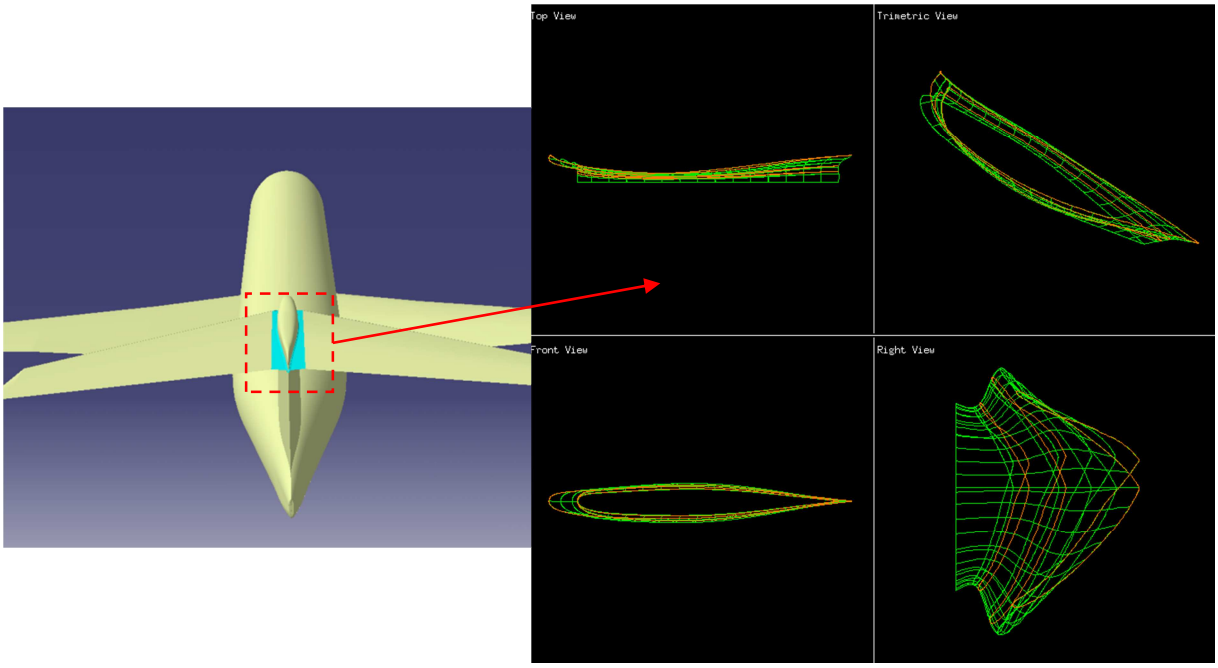


Fig. 5.30: example of Horizontal/Vertical tail connection generated by the MMG.

they all required in each case. Some operations can be seen as alternatives for each other. It can be noted that many operations are common to the three main connection cases, which is a prerequisite for the definition of a unified connection modeling approach.

When an a connection-element instance is required, the system should be able to derive automatically the list of involved surfaces, irrespective of the type of parts to be linked, and select a combination of preprocessing and surface generation methods from the matrix, either a default (best-practice) combination, or the one specifically required by the user (for example via the MMG input file)[6]. More details can be found in reference (van Dijk, 2008).
For example, to connect the horizontal and vertical tailplanes of a cruciform tail configuration, the connection-element class should know that the affected surfaces are the root wing-part surface of the horizontal-tail and any of the fin wing-part surfaces. Then the surface of the horizontal tail could be extended to intersect one or more vertical tail wing-parts. The intersection curve (or the curve composed of the various intersection curve elements across different fin wing-parts) can be adjusted if required (e.g., scaled) and used, together with the root airfoil curve of the horizontal-tail, to build a lofted surface, or a blend surface, etc. Fig. 5.30 shows an example of a horizontal/vertical tail connection, generated by the MMG (van Dijk, 2008).

There are cases of connections where no surfaces/curves preparation is required and the final connection surfaces can be directly generated. This is the case of blended winglets, where a blend surface can be generated directly from the tip section of the wing to the root section of the winglet. In this case, the challenge is to ensure a proper relative positioning of the two parts. Refer to (La Rocca and van Tooren, 2002b; Meijer, 2003; Brouwers, 2007) for details.

There are cases where three parts must connect to each other at the same time, such as H-tail configurations or the Airbus-typical wingtip fences. Reference (Brouwers, 2007) shows how to handle those situations as combinations of the first two cases in Table 5.7.

---

[6] The way connection surfaces are defined can largely affect the pre-processing of the overall aircraft for aerodynamic analysis. In fact, the number and the shape of the surface patches can vary significantly. Certain types of connections can be used to facilitate the preprocessing for analysis, even if they differ from the real aircraft geometry. Of course, the difference should be acceptable to not invalidate the analysis result. Refer to (van Dijk, 2008; van den Branden, 2004) for more info on this issue..

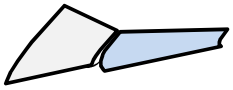
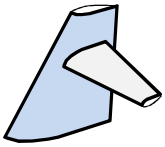
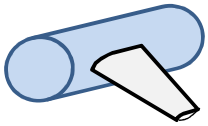
	Adjacent wing-parts 	**Transversal wing-parts** 	**Wing-part/fuselage** 
Preprocessing	• Surface trimming • Insert of extra control curve(s) • (re)positioning of the surfaces	• Surface trimming • Insert of extra control curve(s) • Surface extension • Surfaces Intersection • Consolidation of multi segment intersection/projection curve(s) • Curve Projection • Scaling of intersection/projection curve(s) • Repositioning of intersection/projection curve(s)	• Surface trimming • Insert of extra control curve(s) • Curve projection • Wing-part surface extension • Surfaces Intersection • Scaling of intersection/projection curve(s) • Repositioning of intersection/projection curve(s)
Surface generation	• Lofted-surface with single set of curves • Lofted-surface with two set of intersecting curves • Ruled-surface • Edge-blend-surface	• Lofted-surface with single set of curves • Lofted-surface with two set of intersecting curves • Ruled-surface • Dual-blend-surface	• Lofted-surface with single set of curves • Lofted-surface with two set of intersecting curves • Ruled-surface • Dual-blend-surface • Filleted-surface

Table 5.7: recurrent cases requiring the generation of a connection element. Typical surface preprocessing operations and type of connection surfaces are indicated for each case

5.9 Fuselage High Level Primitive implementation

In the following two sections the basic characteristics and functionalities of the Fuselage primitive are briefly illustrated. Details concerning the technical implementation in the ICAD system are not included here, but be found in the following references (Meijer, 2003; van den Branden, 2004; Koopmans, 2004; Cerulli et al., 2004; van Houten et al., 2005). The intent of this section is to illustrate the implemented parametric modeling approach and the achieved level of flexibility.
Similar to the Wing-part, the Fuselage HLP has a modular architecture, where separate classes have been defined to model the outer surface and the internal structure (see class diagram in Fig. 5.31). Also in this case, the structural components are generated using the outer surface as a support, such that the fuselage structure is always tailored to the outer surface.

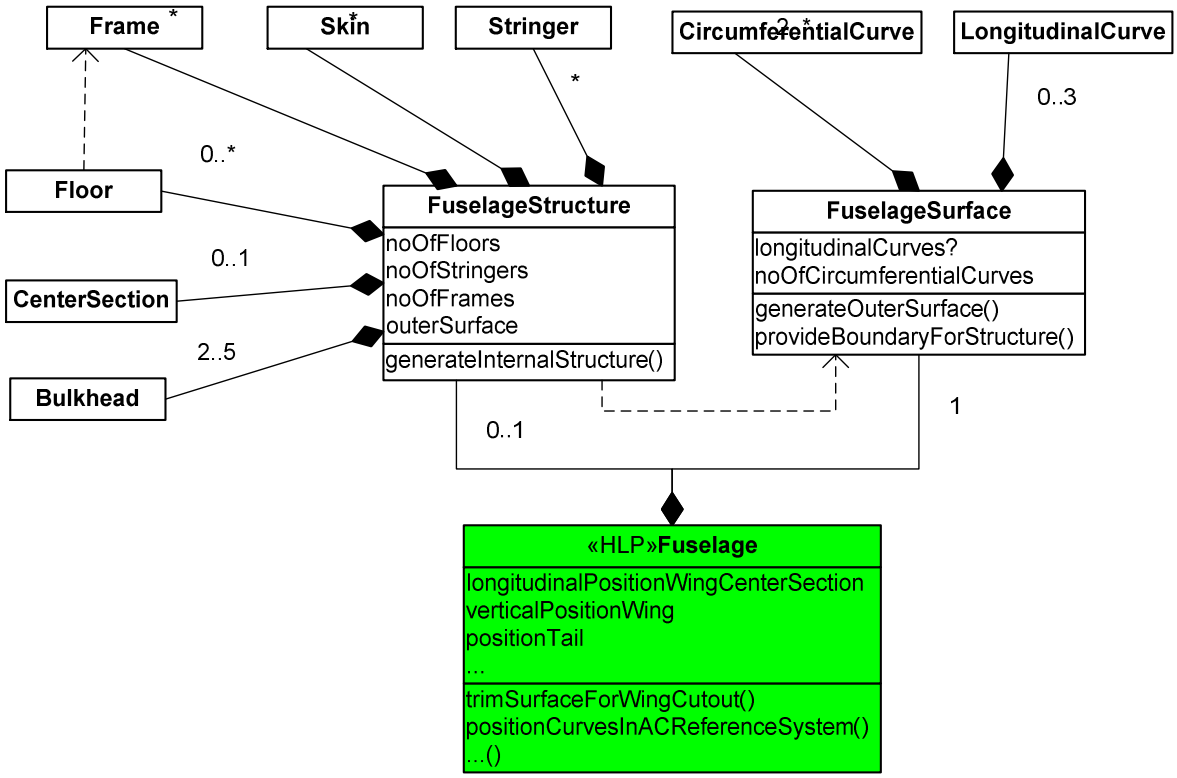


Fig. 5.31: class diagram of the Fuselage HLP.

5.9.1 The fuselage primitive surface generation

The modeling approach implemented to generate the outer surface of the fuselage primitive is rather simple and aims at the generation of one, continuous aerodynamic surface extending from nose to tail.

The procedure consists of the following main steps:

- Definition of a skeleton of support curves
- Interpolation of a B-spline surface[1] on top of this skeleton.
- (optional step) Local modification (*morphing*) of the obtained surface adjusting the B-spline surface weights.

The first step is fundamental to the quality and accuracy of the overall fuselage design. Again, two different possibilities are offered to the designer (a parameter in the MMG input file allows the selection) to define the skeleton of curves. The two options allow for two different levels of control on the final shape:

1. Skeleton definition based on sets of longitudinal and circumferential curves
2. Skeleton definition based on a set of circumferential curves only

Option 1:

This modeling approach is based on the definition of both longitudinal and circumferential curves (see Fig. 5.32).

First, four longitudinal curves must be defined, namely the *crown curve*, the *belly curve* and two *side curves* (actually the definition of the left side curve is sufficient because of the fuselage lateral symmetry). These curves are built by the Fuselage HLP interpolating through sets of user-defined 3D points, stored as .dat files.

Once the longitudinal curves are in place, they can be used to position an arbitrary amount of circumferential curves, of which the user can assign the longitudinal position.

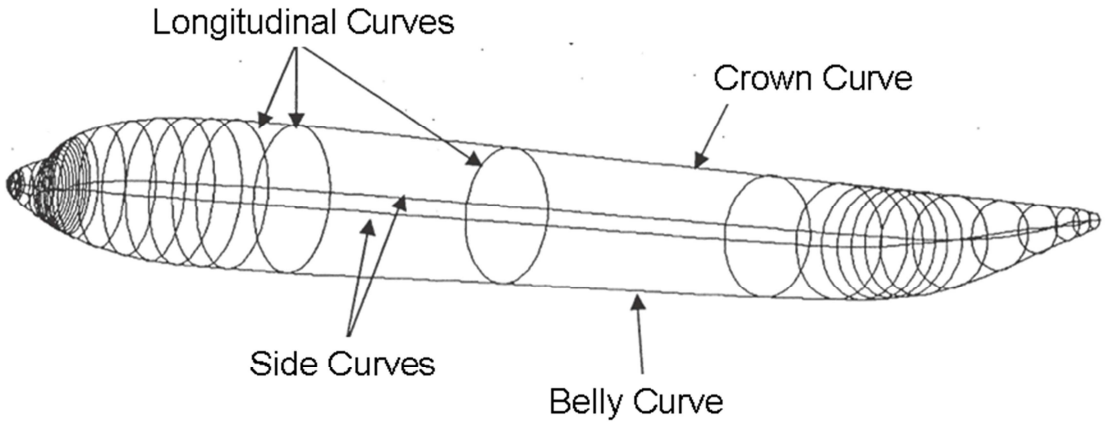


Fig. 5.32: definition of the curve sets used to model the fuselage primitive surface

[1] Details on the definition of B-spline surfaces can be found directly in the *ICAD Surface Designer* (Knowledge Technologies International, 2001b) user manual, or in the Farin textbook (Farin, 1988), which contains ICAD's underlying mathematical representation.

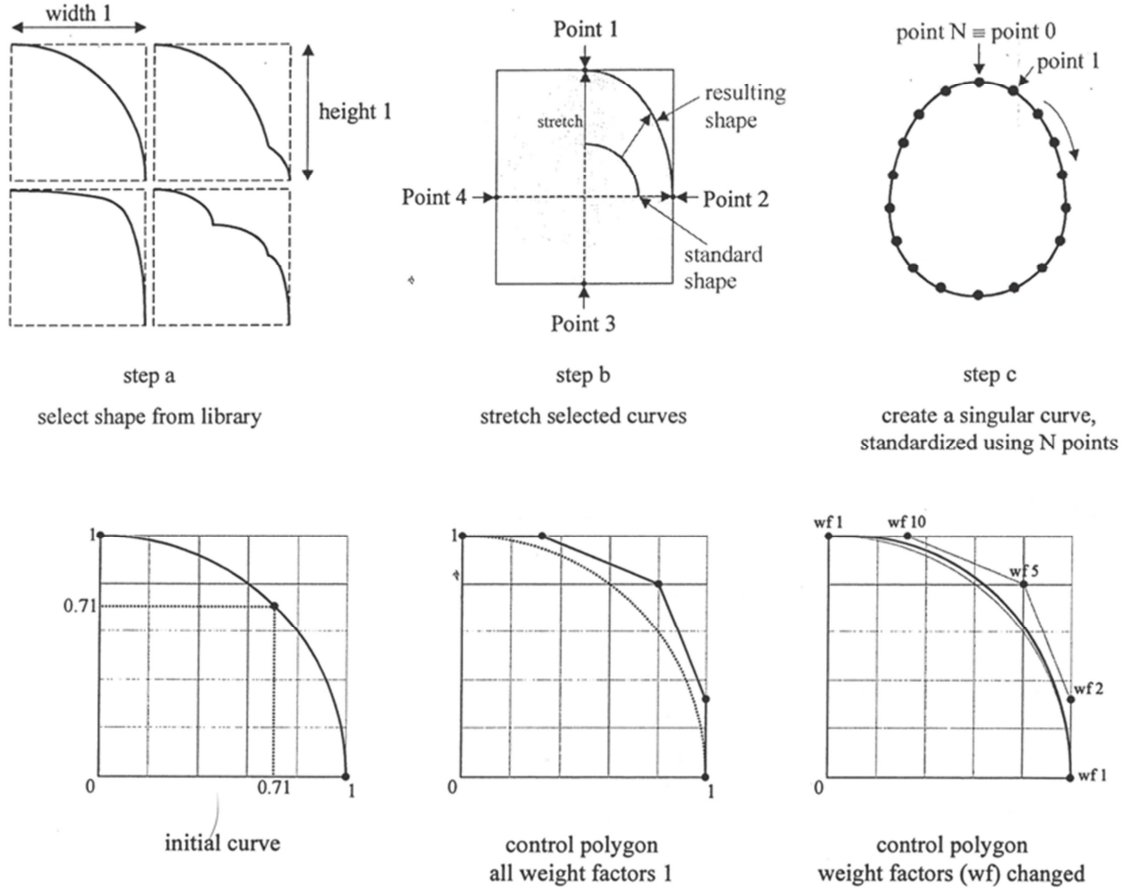


Fig. 5.33: Construction of circumferential curves (top). Only upper-left curve component shown in figure. Modification of B-spline curves using control points weights (bottom)

The circumferential curves are actually defined by merging 4 curve segments: the upper-left curve, the lower-left curve and the two right counterparts. The left curves can be selected by the user among those stored in a pre-generated library (the upper/lower-right curves are automatically generated by symmetry). Similar to the Wing-part HLP airfoils, the circumferential curve components are built interpolating through sets of normalized point coordinates, stored in a dedicated library as .dat files Fig. 5.33-*step a*. Algorithms take care of stretching the four curve components until their extremities match the longitudinal curves (Fig. 5.33- *step b*), and merging them into single circumferential curves (Fig. 5.33- *step c*).
Once the skeleton of longitudinal and circumferential curves is in place, a single B-spline surface is generated on top to obtain the final fuselage surface.

The separate definition of the upper and lower parts of the circumferential curves (as well as the possibility to affect the tangency condition at their merging points) yields quite some modeling flexibility, suitable also for non cylindrical fuselages.

Besides, the designer has the possibility to modify locally the fuselage surface, without regenerating the curves skeleton, but simply adjusting the weight factors of the B-spline surface control points[2] (Fig. 5.33- *bottom*).

Option 2:
This second method requires the user to provide for each circumferential curve the files containing the upper and lower curve segment definition, as well as the 3D coordinates of Point 1 till Point 4 (Fig. 5.33- *step b*). The latter will be used to scale the normalized curve segments appropriately.
This second method has the inconvenience that a lot of circumferential curves need to be provided by the user to model properly areas with large curvature gradient, such as in correspondence of the cockpit area. This can require more efforts from the user, while properly defined longitudinal curves offer the possibility to automatically position more circumferential curves where the gradients are larger.
Similar to Option 1, the obtained fuselage surface can be locally adjusted modifying the weights of the B-spline control points.

In Fig. 5.34 two examples of aircraft models generated using the first modeling option are shown to give an idea of the achievable level of surface definition[3]. At the moment, no functionalities have been implemented to model complex fuselage-wing fairings.

When the surface to be modeled has a simple quasi-cylindrical geometry, the circumferential definition curves can be defined with a very low number of points. On the other hand, if the given shape presents large gradients of curvature, the system does not have problems in handling very densely defined curves (hence provided with many points).

---

[2] As a matter of fact, all the circumferential curves are first generated using the ICAD primitive *interpolated-curve*, which results in a set of B-spline curves. The control points of these curves are subsequently extracted and finally used as input to build the actual fuselage surface (using the ICAD primitive *B-spline-surface*). The weight of this surface control points (initially all set to one) can be modified by the user as required to affect the fuselage shape. The convenience of this method stays in the number of surface control parameter (i.e., the weight coefficient), which is much lower than the amount of point coordinates stored initially in the fuselage curves .dat files.

[3] The surface of the engines nacelles is generated using a very similar approach to the fuselage.

In this latter case the problem is actually how to get such point collections for the library .dat files. Even when native CAD files containing the aircraft geometry are available, the general procedure of extracting curves and points for external use (e.g., to prepare input data for the MMG) can be very labour intensive. On this purpose, a simple "scanning" module has been developed to extract from a CAD model (imported into the ICAD system via IGES format) any number of circumferential curves, sample them with any amount of points and normalize them as required for the method discussed above (van den Branden, 2004). This module has revealed useful to populate the fuselage sections library with several .dat files ready for reuse.

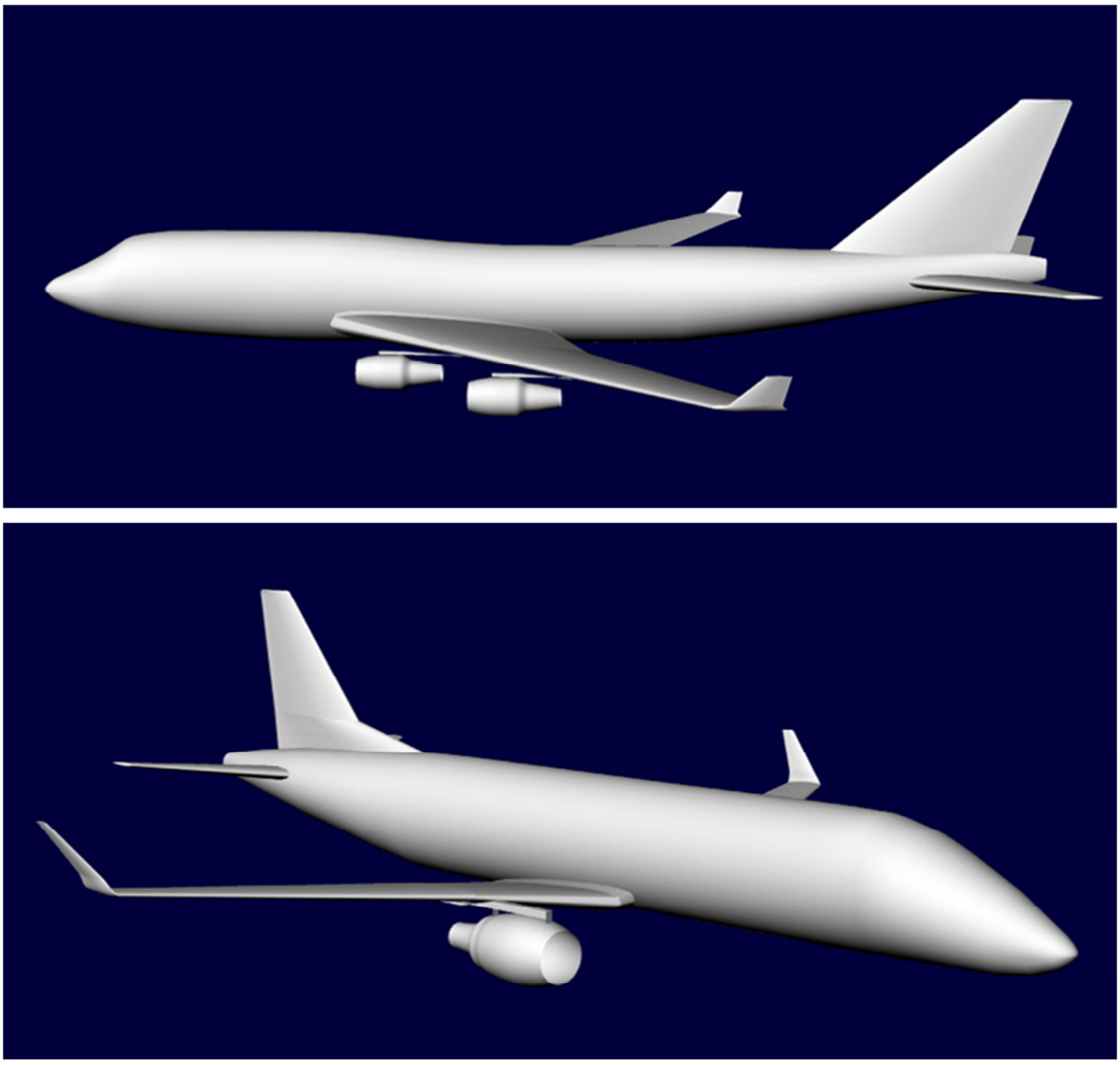


Fig. 5.34: Two examples of aircraft models created with by the MMG.

5.9.2 Fuselage Structure generation process and capabilities

As shown in the class diagram of Fig. 5.31, the structural elements included in the Fuselage HLP structural aggregation are frames, stringers, bulkheads, skins and another aggregation of parts constituting a simple model of the wing center section, which includes the keel beam for low wing aircraft configurations (see sketch in Fig. 5.35)

The modeling process of the structural component starts with the position of the wing with respect to the fuselage. The wing center section is actually not modeled by the Fuselage HP, but by Wing-part and consists of an instance of WingTrunkStructure, where the LE/TE parts have been excluded.

In correspondence of the wing center section front and back spars, two intersection planes are used to cut the fuselage outer surface: the resulting circumferential intersection curves are used to model the wing attachment bulkheads (Fig. 5.35). A similar approach is used to model bulkheads at the tail empennage/fuselage intersections. A third bulkhead is positioned (based on user-defined parameter) to close the landing gear bay.

As shown in the example of fig.4.7 middle, a procedure has been implemented to deal with high-wing configurations too.

Apart from these bulkheads, an arbitrary number of frames can be generated based on a user-defined list of longitudinal positions. Similarly to the bulkheads, all the frames are generated by intersecting the fuselage surface with planes. These intersection curves are then used as guides to sweep a frame profile selected via input file by the MMG user (see sketch in Fig. 5.36). Currently C, Z, U and C section frames can be generated.

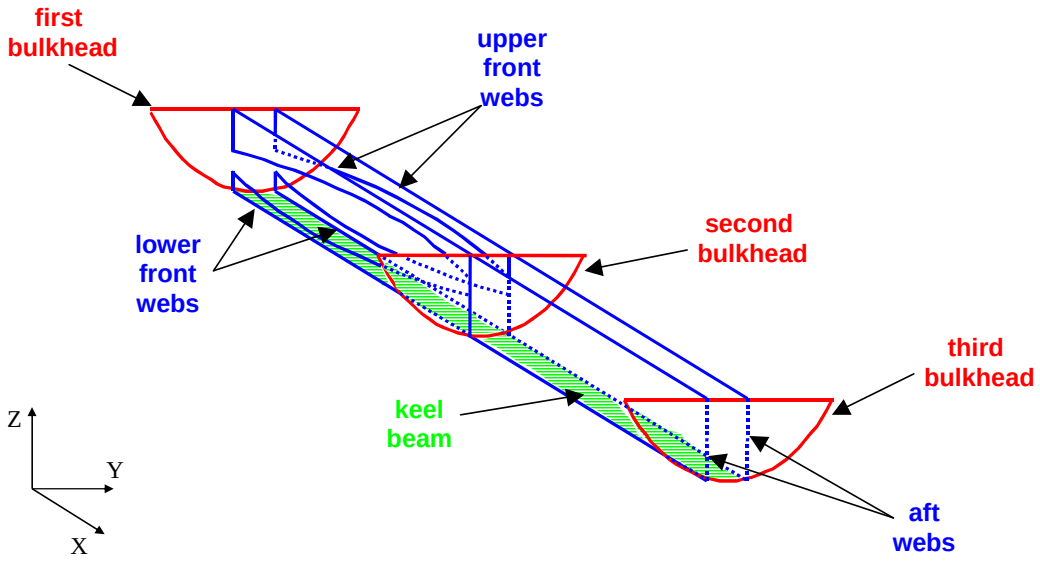


Fig. 5.35: definition of wing-crossing and landing gear bay bulkheads and keel beams as modeled by the Fuselage HLP.

The user has also the possibility possible to specify two frames, which will be used for the definition of the front and back pressure bulkheads.

In order to generate stringers, first a user–defined number of points is generated circumferentially on each frame. All the corresponding frame points are then used to interpolate a set of longitudinal curves, which are then projected on the fuselage surface to form the stringer guidelines (see sketch in Fig. 5.36). Similar to frames, user-defined profiles are then swept along these guidelines to model the actual stringers geometry. Currently it is not possible to define stringers with run-outs.

The user can also define a number of floors. Their vertical position is assigned by means of a user-defined offset with respect to the Aircraft Reference System. Partial floors can be defined that start and end at user-define longitudinal positions. The implemented modeling procedure first computes the intersection between the floor planes and all the fuselage frame lines (Fig. 5.37, top), then uses the intersection points to define floor beams and finally generate a set of floor panels (Fig. 5.37, bottom). The floor panels do not "touch" the fuselage skin but transfer their load directly on the frames.

Although less mature than the Wing-part HLP, the Fuselage primitive allows a reasonable level of modeling flexibility, both concerning the outer surface and the internal structure arrangement. Refer to figure fig.4.7 for examples of fuselage structure models generated by the MMG.
In reference (Meijer, 2003) apart from a detailed description of the Fuselage HLP architecture and implementation, the capability of the MMG to model the geometry of the entire family of Airbus passenger aircraft is demonstrated.

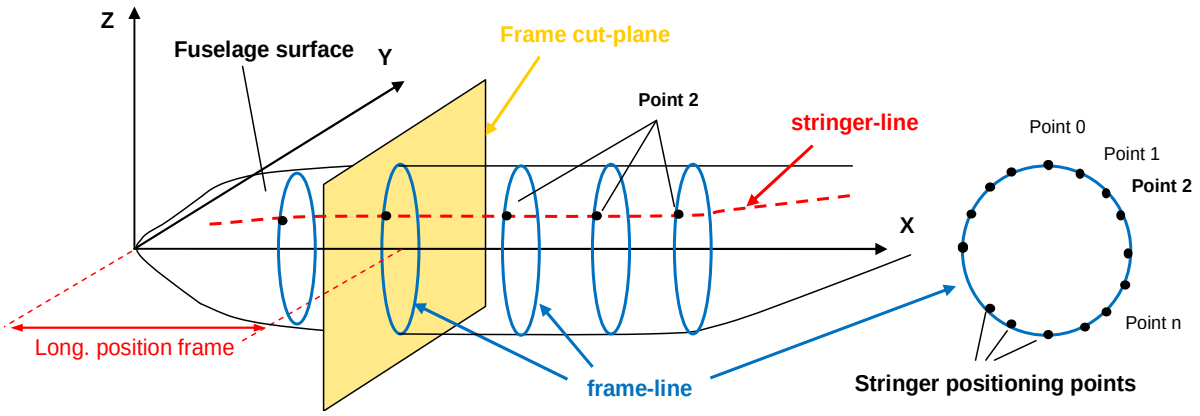


Fig. 5.36: frame and stringer lines definition method

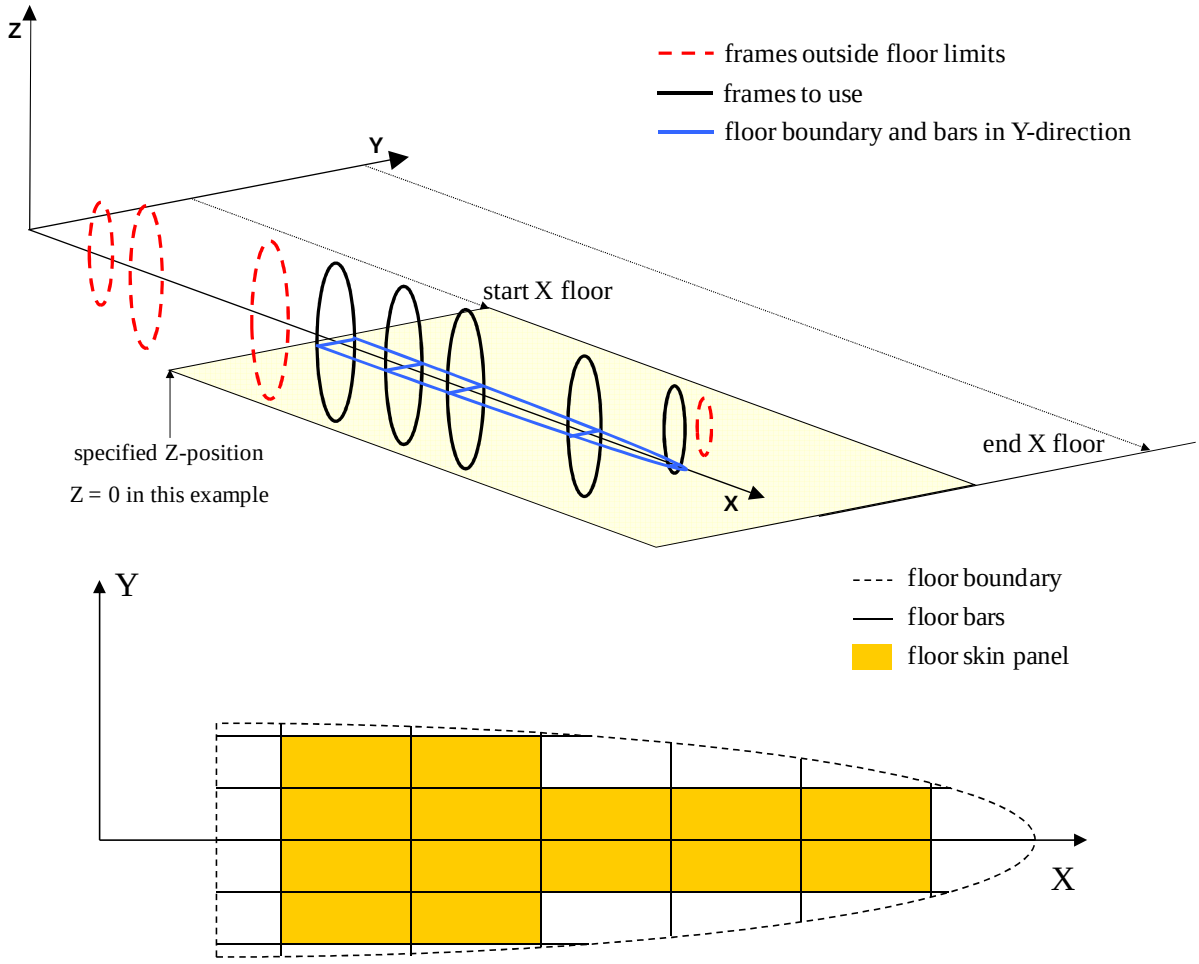


Fig. 5.37: definition of floor boundaries (top) and floor beams and panels (bottom).

CHAPTER 6
Implementation of the Capability Modules and operation of the MMG

1. Introduction
2. Capability Modules for aerodynamic analysis
3. Capability Modules for FE structural analysis
4. MMG – FEA environment integration
5. Operating the MMG
6. Study case 1: The MOB project
7. Study case 2: Vertical tail redesign study
8. Multi-level modelling to manage complexity and support multi-level design

6.1 Introduction

In this chapter, the capability of the MMG to automate the generation of models to support multidisciplinary analysis of aircraft (and aircraft components) and their optimization is discussed and demonstrated.
As any ICAD-developed KBE application, the MMG is natively able to export geometry models using standard exchange formats such as IGES, STEP, STL, as well as several proprietary CAD formats[1] such as CATIA V4, UGII, AutoCAD, Pro-Engineer, etc. (Knowledge Technologies International, 2001a). However, many analysis tools, especially in-house developed ones, do not always support any of those standards, and rely on custom formats, often based on some kind of ASCII table (e.g., containing point coordinates).

---

[1] A separate license is required to operate each one of the various translators. Furthermore, separate licenses are also required to export and import files in the various formats.

The longevity of the standard format file translators, such as IGES and STEP, is of course superior to those of proprietary formats, which must account for the evolution of the related CAD systems. The ICAD system (together with the whole company KTI) was acquired by Dassault Systemes when the translator for the newcomer CATIA V5 was under development.

On the other hand, commercial analysis tools are often provided with proprietary geometry pre-processors and mesh generators, which, however, require plenty of manual operations and are difficult to script in a flexible way.
To the scope of supporting automatic geometry pre-processing and achieve a seamless integration of the MMG with external analysis tools, both commercial of the shelf (COTS) and in-house developed, a number of capability modules (CMs) have been developed. In this chapter, the functionalities and the implementation of some of the CMs introduced in Chapter 4 are elaborated in more detail, in particular of those developed to support automatic generation of structural and aerodynamic analysis models. The achieved integration of the MMG with two commercial analysis codes for aerodynamic and structural analysis is subsequently discussed.

In the second part of the chapter, two study cases will be presented that give evidence of the MMG ability to enable distributed multidisciplinary analysis and optimization of complex products. The first case concerns the conceptual/preliminary design of a blended wing body aircraft, carried within the framework of the European project MOB. The second deals with the redesign of the vertical tail for a large passenger aircraft and has been carried in collaboration with Airbus Germany. In the description of these study cases, the focus is on the role and functionality of the MMG, rather than the goals and finding of the two projects. For those, references are provided.

To conclude, the strategy developed to deal with increasing complex KBE applications for multi-level design is discussed and examples on the current state of development are provided.

6.2 Capability Modules for aerodynamic analysis

The Points-generator CM has been developed to "translate" the surface of any HLP instantiation into a so-called *cloud of points*. Via the MMG input file, the user has the possibility to control the density of the cloud in terms of number of sections, number of points per section and point stretching (i.e., the point distribution on the various sections). The level of control on the cloud density is at the level that the user can demand a different amount of sections per Wing-part instance, or different amount of points and stretching for the wing, tail and fuselage surfaces.
Once the points are generated, their Cartesian coordinates (defined in the Aircraft Reference System) are organized by the MMG as required by the recipient tool and finally transferred, either via plain ASCII file or more structured XML files.
The cloud of point's translation approach has been used successfully to define dedicated models for a heterogeneous range of aerodynamic analysis tools (Fig. 6.1). These models include simple flat panels discretization as those used for aeroelastic analysis (La Rocca et al., 2002; Stettner and Voss, 2002) with ZAERO (Zona Technology, 2009), as well as those for potential codes such as VSAERO (Analytical

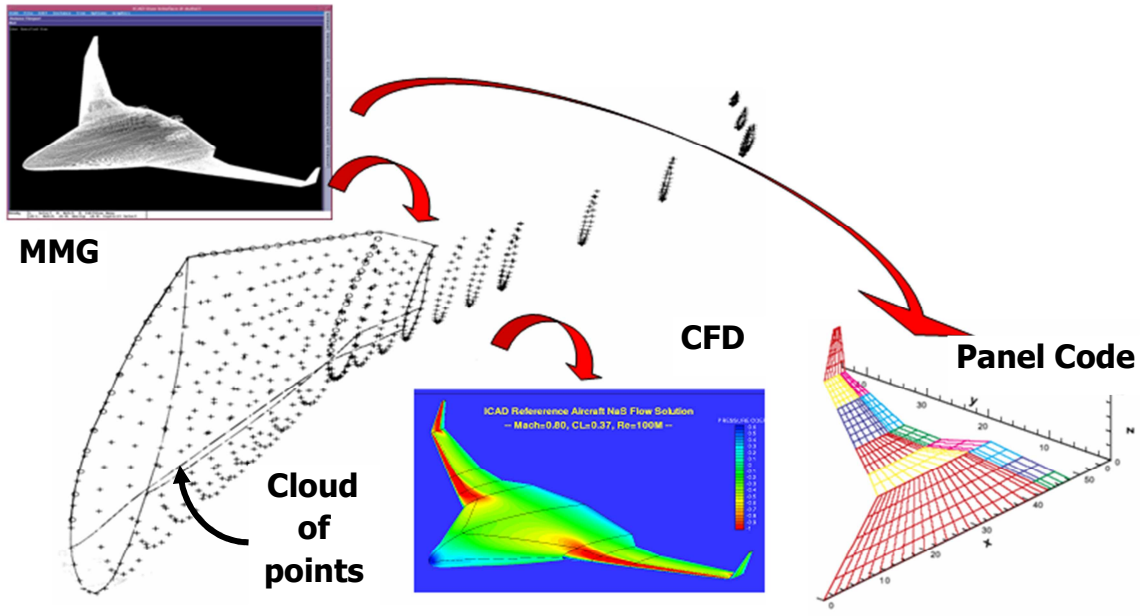


Fig. 6.1: Translation of the aircraft surface to support automatic generation of models for both high and low fidelity analysis. Examples of a refined model for Euler/Navier-Stokes CFD analysis and a simplified one with flat paneling (including TE movables) for aeroelastic analysis.

Methods, 2009) and other in-house developed panel codes (Van Staveren, 2003; van den Branden, 2004).
The cloud of points approach has been used also to support setting up models for high fidelity CFD analysis. In this case the generated points have not been used as grid points, but as *control points* to support the automatic re-splining of the aircraft surface into the pre-processing environments of MERLIN and ENFLOW, respectively, the in-house developed Reynolds-averaged Navier-Stokes tool of Cranfield University and the multipurpose Euler/RANS system in use at NLR (more details in (Qin et al., 2002; Laban et al., 2002)).

In order to operate, the Point-generator CM needs to have available well defined surface patches where to extract points coordinates. Whilst the generation of these patches is relatively straightforward for a configuration like the BWB of Fig. 6.1, which consists only of a series of adjacent Wing-part instances, it is definitely not in presence of intersections between lifting surfaces and fuselage, or between two lifting surfaces (e.g., in case of T-tail, H-tail or cruciform tail configurations). The actual challenge is about generating - automatically and for any type of aircraft configuration - sets of non intersecting patches, with properly matching edges, as shown in Fig. 6.2.

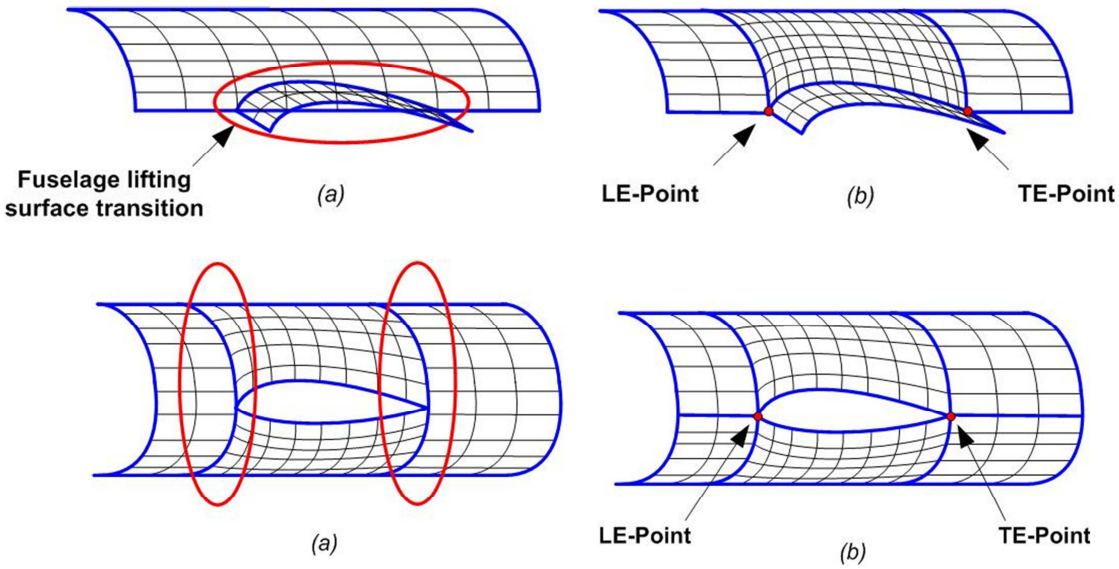


Fig. 6.2: Generation of wrong (a) and correct surface patches (b). Patches must not intersect with each other and must share only one edge with those adjacent (patches indicated with tick blue boundary lines).

Actually, in case of aircraft configurations with an "intersected fuselage", the generation of aerodynamic views is accomplished by means of the abovementioned Point-generator, plus another CM called Surface-patcher. The functionalities of both are described in the following sections.

6.2.1 Surface-patcher, a capability module for automatic patches generation

Given a generic lifting body, such a wing, a canard or a tail empennage, the generation of surface patches is limited to splitting the surface of each Wing-part and Connection-element instance along the leading and trailing edge curves. The resulting upper and lower skin patches can then be directly processed by Point-generator.

On the other hand, Surface-patcher is needed to deal with fuselage surfaces intersected by wings, canards, tail empennages, engine pylons, etc. In this case, Surface-patcher performs the following activities (refer to (van Dijk, 2008) for details):

- Detect the LE and TE points of all the *connection curves* (i.e., the curve resulting from the intersection of the fuselage surface with the "piercing" bodies). Fig. 6.3 (A)
- Generate circumferential curves (not necessarily orthogonal to the fuselage axis) on the fuselage surface that pass through the above detected LE/TE points.

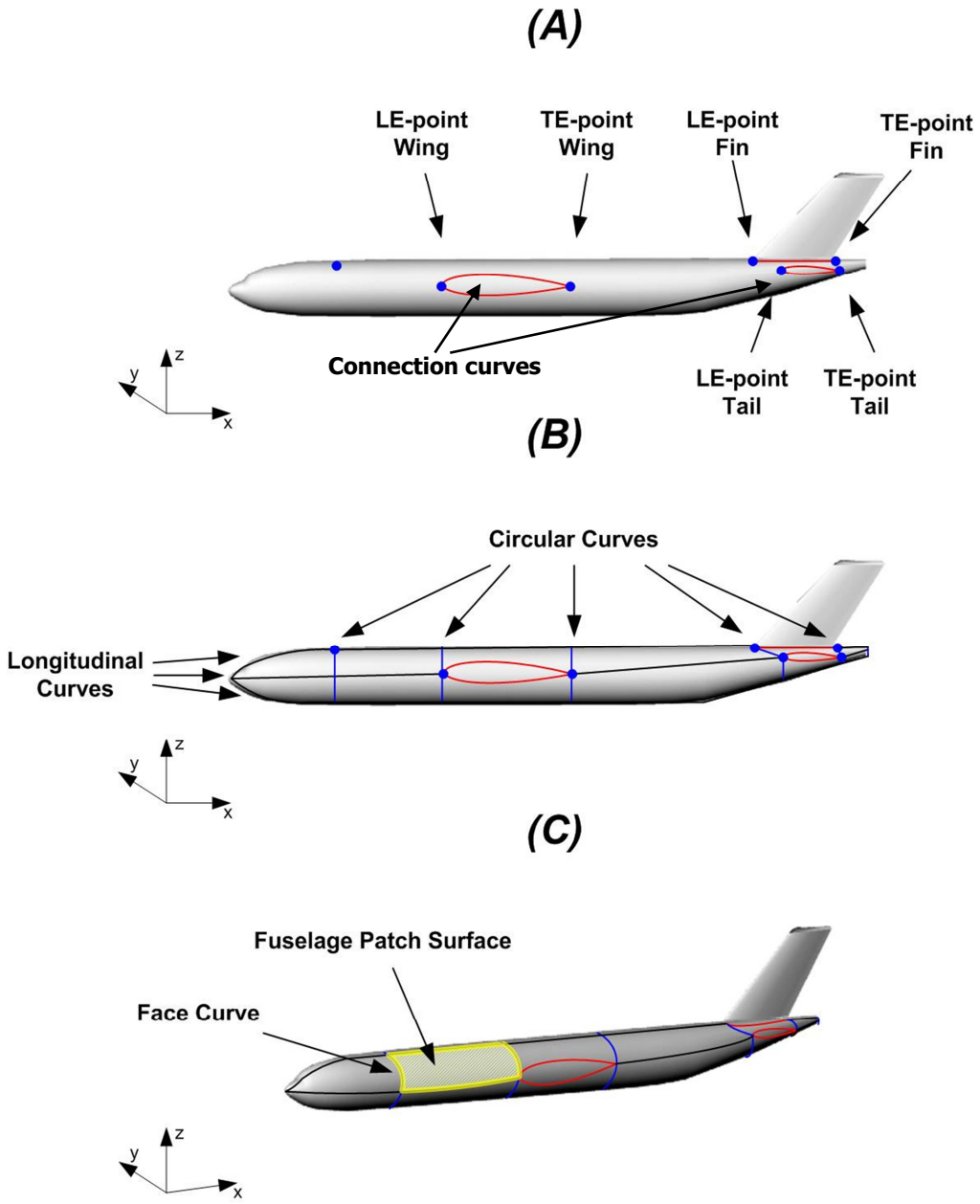


Fig. 6.3: main steps performed by the Surface-patcher CM to split the surface of a fuselage in suitable patches for aerodynamic analysis.

- Eliminate the circumferential curves that intersect a connection curve and combine more close curves in one when possible, to reduce the number of patches. Note the circumferential curves at the fuselage fin intersection in Fig. 6.3 (B).
- Connect the TE point of each connection curve with the LE point of the next one (starting from the fuselage nose going backwards) by means of longitudinal curves defined on the fuselage surface.

- Intersect the longitudinal and circumferential curves with each other and collect the curve segments that delimit each patch. Fig. 6.3 (C).

The patching method described above can be used also in case of more lifting surfaces that intersect each other, such as the horizontal stabilizer and the fin in a T-tail configuration. However, to limit the complexity of the patching problem and the amount of resulting patches (Fig. 6.4, a), it is convenient to define a connection element that extends from the LE to the TE curve of the pierced Wing-part instance (Fig. 6.4, b). In the T-tail example shown in the figure, the chord distribution of the horizontal tail is maintained unaltered, while the span of the connection element has be kept sufficiently small to be ignored by the aerodynamic solver (of course, the designer has still the possibility to modify the shape of the connection element if the intention is to model a real fairing). In this way, the fin surface is not pierced any more, but simply split in more spanwise patches, as shown in the example of Fig. 6.4 (b). Note that using this modelling approach, also the fuselage patching results simplified, because unaffected by the presence of the horizontal tail piercing the fin.

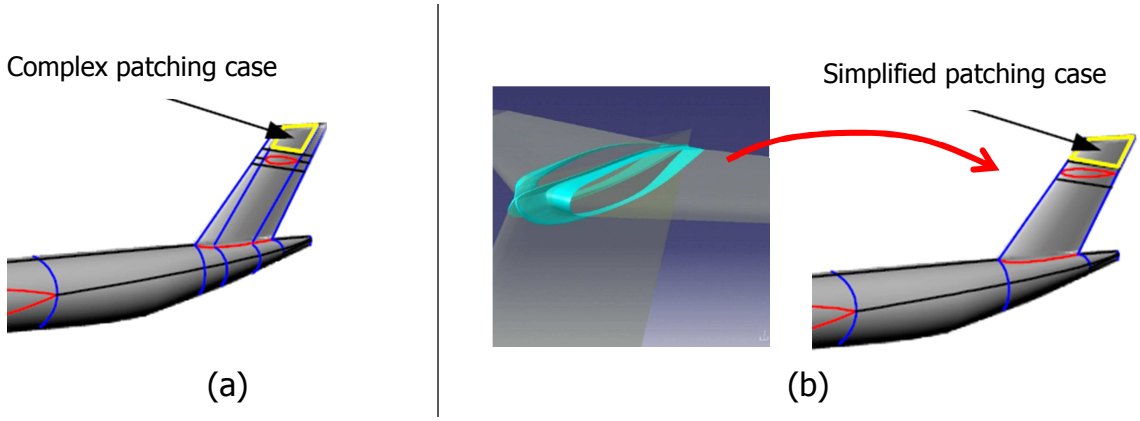


Fig. 6.4: complex patching in case of self intersecting Wing-part instances (a). Simplified case by the implementation of a connection element spanning from the LE to the TE edge of the fin (b).

6.2.2 Point-generator, a capability modules for point generations on surface patches

In order to operate, the Point-generator capability module needs the following input:

- The native surface on which the patch is defined
- The four curves delimiting the patch: Curve-u1, Curve-u2, Curve-v1 and Curve-v2 (Fig. 6.5)
- The number of points to be placed on the u-curves[2] (*no-of-u-points*)

[2] As u-curve is intended any curve in between and "aligned" with Curve-u1 and Curve-u2

- The number of points to be placed on the v-curves (*no-of-v-points*)
- The parameters defining the stretching[3] of the points on the u-curves
- The parameters defining the stretching of the points on the v-curves
- The MMG takes care of sending consistent input values for each patch in the aircraft. The order in which the patch boundary curves are provided to Point-generator, as well as the amount of points to be generated along each u and v-curve must be such to guarantee a proper panelling of the complete aircraft surface. Similar to the patches, also the panels are (generally) allowed to share only one edge with the neighbours, regardless the patch they belong to.

As a matter of fact, two versions of the Point-generator CM have been developed, to be selected depending on the type of patch data provided as input:

Point-generator Version 1: In case of alignment of the patch boundary curves with the iso-lines of the native surface (Fig. 6.5-a), a fast and straightforward point generation process can be implemented:
A number of points equal to *no-of-u-points* are generated along curve-u1, according to the required stretching.
In correspondence *of each point* generated on Curve-u1, the corresponding iso-v line[4] is selected, along which the amount of points indicated for v-curves (*no-of-v-points)* is generated, according to the required stretching (of course the points are generated along the portion of iso-v lines delimited by curve-u1 and curve-u2).
Once points have been generated along all the selected iso-v lines, their Cartesian coordinates are collected and stored together with the point from the other patches.
Indeed, this method exploits the possibility of accessing the iso-lines of the native surface, which are conveniently aligned with the boundary curves of the patch. However, also in case of misalignment (Fig. 6.5-b), there is a possibility to rebuild the patch using a fresh new surface[5], as shown in the example of Fig. 6.5-c. In this way the same version of Point-generator can be used, as far as the surface of the rebuilt patch is given as input, in place of the native surface.
However, the operation to rebuild the patch surface does not always guarantee a good result: the new patch might deviate too much from the underling surface,

[3] A sinusoidal stretching function has been defined to allow tuning the point density at the leading and/or trailing edge of the airfoil curves. Details in ref. (La Rocca and van Tooren, 2002c)

[4] A v-constant iso-curve means that the v parameter is held constant and the u parameter varies (the iso-curve is in the u-going direction). A u-constant iso-curve means that the u parameter is held constant and the v parameter varies (the iso-curve is in the v-going direction).

[5] The ICAD primitive *quad-blend* is used at the scope, which generates a new surface based on four input curves and the surface on which these curves lie. The resulting blend is C1 continuous with the surfaces at the boundary (Knowledge Technologies International, 2001b).

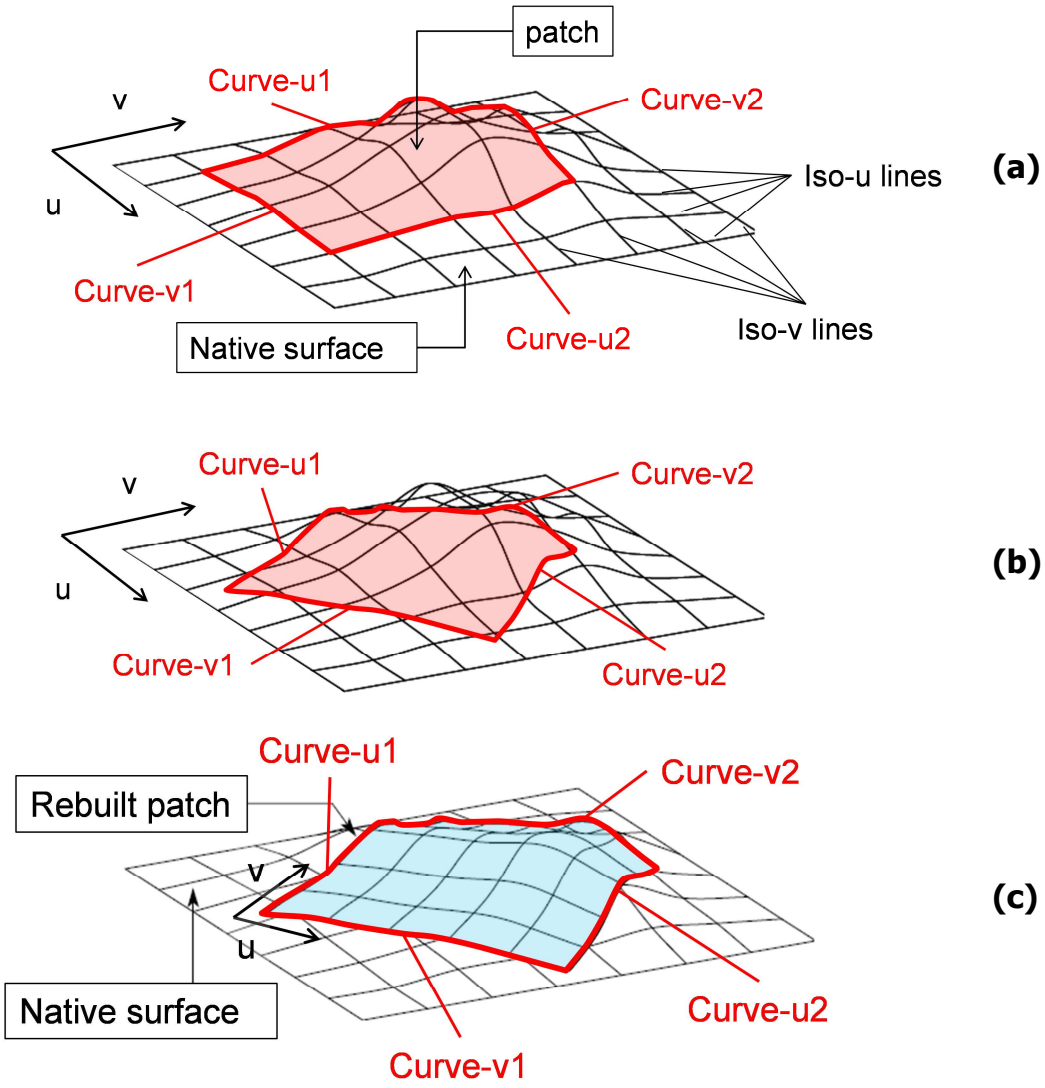


Fig. 6.5: Examples of patches with boundary curves aligned (case a) and crossing (case b) the iso-lines of the underling surface. Case c: a patch surface rebuilt copying the underlying surface.

especially in case of large, double curvature patches, with skewed or irregular boundary curves. For these cases, the second version of the CM must be used.

Point-generator Version 2: This version encompasses a more complex approach based on the generation of a grid of "pseudo iso-lines", on which the required points can be generated. It works as follows (Fig. 6.6):
Points are generated on curve-u1 and curve-u2 (in the amount and according to the stretching indicated via input)

Support curves are generated linking couples of corresponding points on curve-u1 and curve-u2 (although the support curves are enforced to lie on the native surface, some cannot be generated in case of large irregularity of the patch boundary curves)
A number of support points (equal *no-of-v-points*) is generated on each remaining support curve, with the stretching required for the v-curves.
A set of pseudo iso-u curves is generated fitting the support points. All these curves lie nicely on the native surface.
An amount of points equal to *no-of-u-points* is generated on each pseudo iso-u curve, with the stretching required for the u-curves. These points are finally added to those initially generated on curve-u1 and curve-u2.

Point-generator Version 1 is generally used to operate on Wing-part instances, where patches often coincide with the underling native surface, or a part of it trimmed

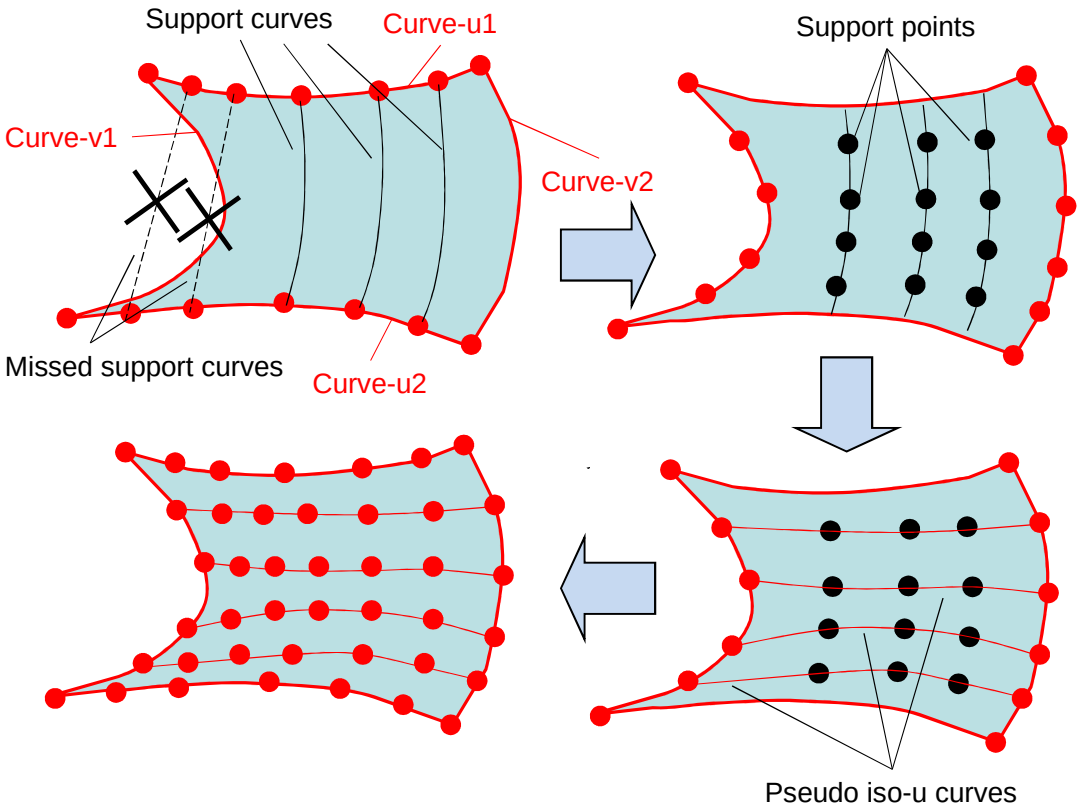


Fig. 6.6: The Point-generator CM (Version 2) in action on a trimmed and curved surface. From top-left, clockwise:

Step 1: Generation of points on curve-u1 and u2 and fitting of support curves
Step 2: Generation of support points along valid support curves
Step 3: Fitting of pseudo iso-u lines through the support points
Step 4: Generation of the user required points along the pseudo iso-u lines

along some iso-curve. Version 2 is generally required to operate on fuselage patches. Although Version 2 alone would be sufficient for all cases, both versions are kept in use, just because of the higher speed of the first.

6.2.3 MMG – VSAERO connection

The pre-processing capabilities described so far (together with COALA (Brouwers, 2007; Grotenhuis, 2007; van Dijk, 2008; Dircken, 2008), the MATLAB application mentioned in Chapter 4) have enabled a seamless integration of the MMG with the commercial panel code VSAERO. The main steps occurring during the preparation and execution of an analysis cycle are summarized in Fig. 6.7:

- The outer surface of the complete aircraft model is generated by instantiation of various HLPs
- All the aircraft surfaces are automatically cut in patches by Surface-patcher
- Each patch is processed by Points-generator, which translates the surface into a set of 3D points, distributed as required by the user.
- The coordinates of all the points are automatically formatted into an XML file. Tags are used to identify the points' membership to the various patches and aircraft surfaces. Information concerning movables surfaces definition and deflection angles are encoded as well.
- COALA reads the XML file and translates the cloud of points into a fully pre-processed VSAERO model.
- A solid angle test is performed to assess the quality of the model (e.g., the presence of undesired gaps/overlaps between panels)
- Analysis is performed. COALA allows the execution in series of multiple user-defined test cases (when test cases concerns the analysis for different movable deflections, COALA takes care of deflecting the relative movables panels, without the need to go back to the MMG).
- Analysis results are automatically post-processed. Aerodynamic and control derivatives and stored by COALA as aero data sets for FMM (Voskuijl, La Rocca and Dircken, 2008), a in-house developed flight mechanics package for the assessment of aircraft performances and handling qualities.

In the current state, engines and nacelles are not included. Configurations with multi element high-lift devices or devices that modify the wing planform area have not been tested because of the current MMG modelling limitations in this regard.

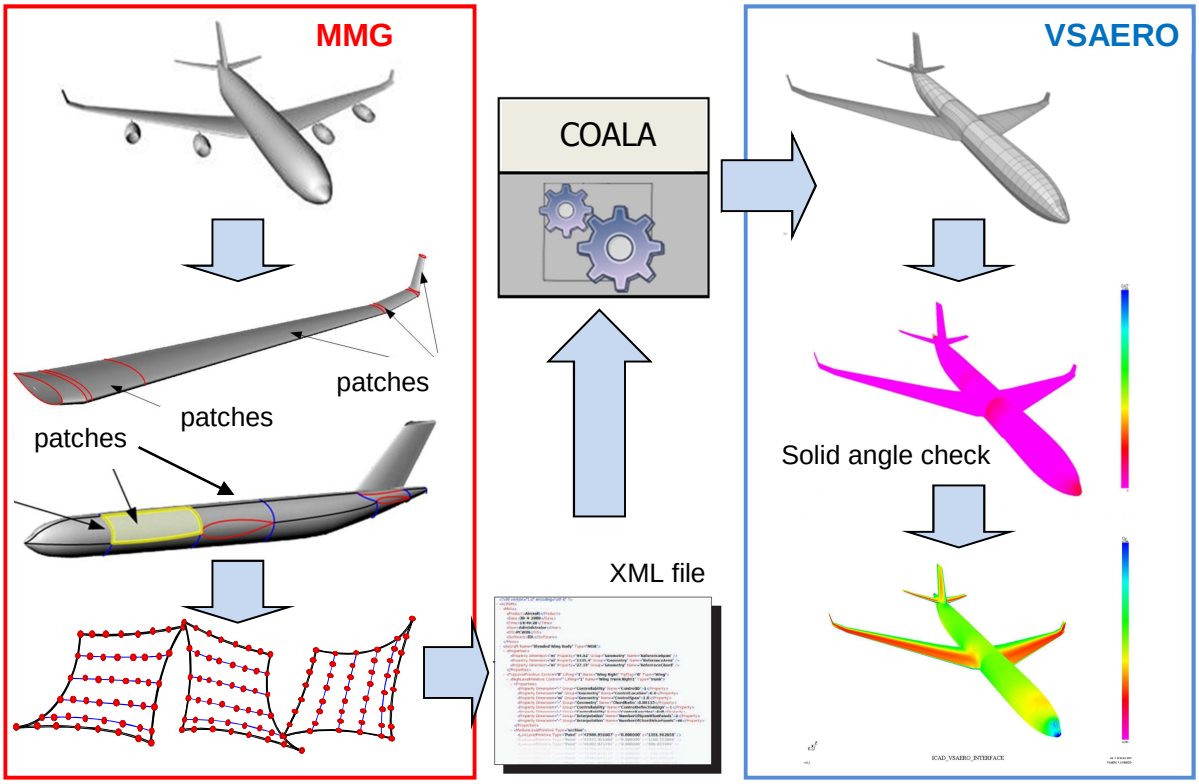


Fig. 6.7: schematic representation of the MMG-VSAERO integration approach.

6.2.4 Integration with other systems for aerodynamic analysis

In the framework of a collaboration project with Airbus Germany, the MMG has been provided also with the capability to generate dedicated models for the Doublet Lattice Method (DLM) in MSC.Nastran (for for linear unsteady aerodynamics) and for an Airbus-proprietary Vortex Lattice Method (VLM), for non-linear steady aerodynamics. This has required transforming the whole aircraft into flat plates, as well as extracting wing curvature information from the 3D geometry to include twist and camber effects in the DLM and VLM analysis. Furthermore, modules were added to the MMG for generating a structural beam model of the aircraft, as required to carry out the unsteady aerodynamic analysis in MSC.Nastran. Report writers have been developed to export all the information directly in the format required by MSC.Nastran (i.e., CAERO cards). Structural and aerodynamic views of the aircraft in Fig. 6.9, technical details in (Koopmans, 2004; Cerulli et al., 2005)

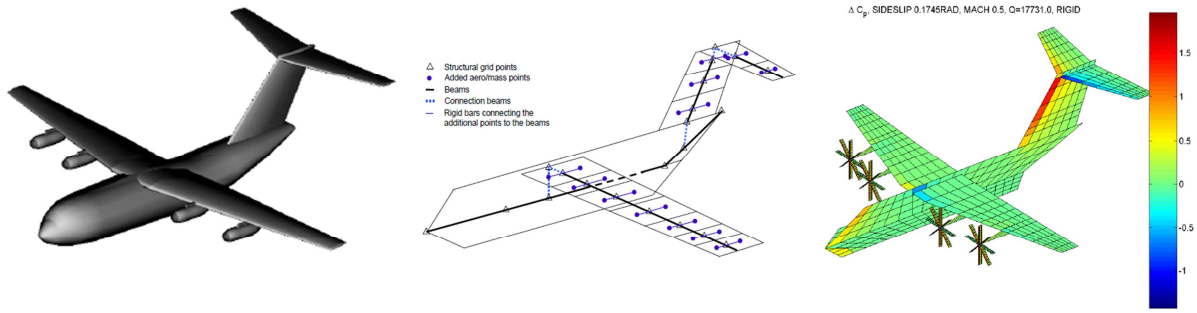


Fig. 6.9: 3D model and relative analysis views generated by the MMG: the structural beam model (middle) used by the DLM model for linear unsteady aerodynamics (right).

Within the national project PARMOD (van Houten et al., 2005), other capability modules have been developed, to extract from the MMG *surface grid models*[6] for the NLR multipurpose Euler/RANS system ENFLOW. A dedicate CM was required to perform the surface patching as required by ENFLOW, which is not compatible with the one used by VSAERO and similar panel codes. The generated aircraft models and its dedicated ENFLOW view are shown in Fig. 6.8 (note the deflected stabilizer surfaces). Technical details in (van den Branden, 2004).

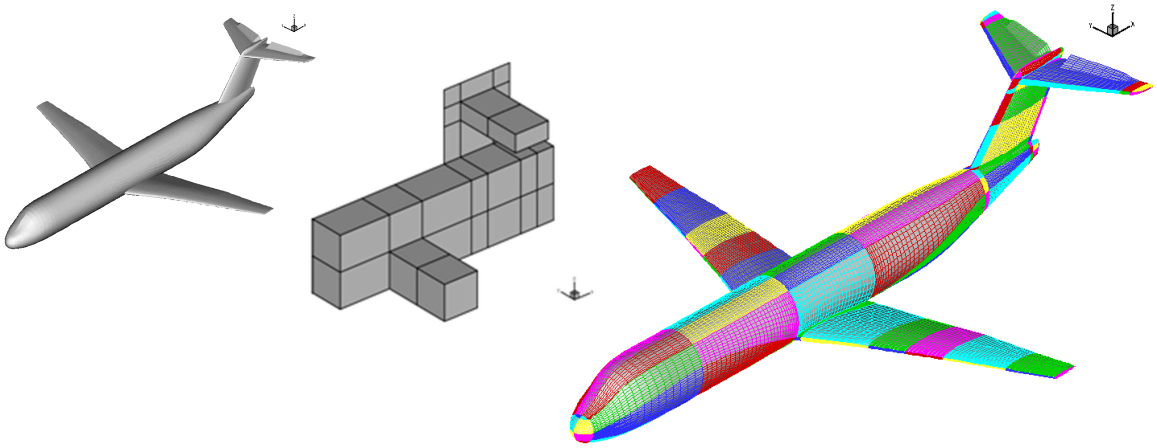


Fig. 6.8: MMG generated Fokker 100 model, and the patching scheme (middle) implemented for the generation of the ENFLOW dedicated aerodynamic view.

6.3 Capability Modules for FE structural analysis

In order to set up a FE model starting from the CAD model produced by the design department, the FE specialist will have to perform a lot of manual work just to

[6] N.B.: complete surface grid models, not just the points for surface re-splining mentioned in section 6.2! However this came at the cost of modeling flexibility, in fact the system functionality was just restricted to aircraft configurations similar to the Fokker 100, the aircraft considered in PARMOD.

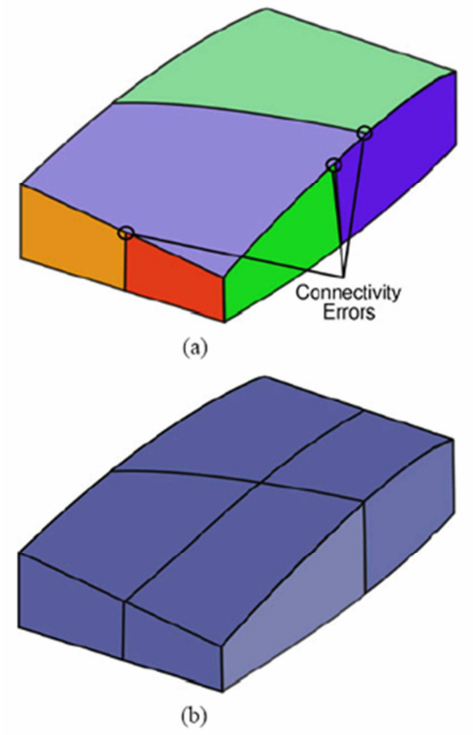


Fig. 6.10: Model with surface segmentation errors (a). Properly segmented model (b).

prepare the model for meshing. The surfaces of all the structural components (e.g., skins, spars, ribs, frames, etc.) must be trimmed along their intersections in order to produce sets of *meshable surfaces*. That is to say, surface segments with no more than four edges, sharing maximum one edge with the neighbouring segments. Fig. 6.10 shows examples of a correct and wrong segmentation of a wing box (Nawijn et al., 2006). The *surface segmentation process* can be very time expensive and often not trivial. Besides, every time a change occurs in the model topology, the segmentation process has to be performed again. Unfortunately, the automatic meshing functionalities provided by most of the FE preprocessors can be used only after all model surfaces have been properly segmented.

The segmentation process is lengthy, repetitive and plenty of rule-based geometry manipulations. As such, it is a good candidate for a KBE application. As anticipated in Chapter 4, a dedicated Capability Module, called Surface-splitter, has been developed to capture the process applied by a FE specialist when manually performing the segmentation process. Given a generic aircraft model built with any number of HLPs instantiations, the Surface-splitter is able to process, one by one, all the various HLP instantiations (Fig. 6.11), and finally deliver a set of surfaces that are suitable to be meshed, *whatever the topology of the generic aircraft and its internal structure.*

Fig. 6.12, shows the Surface-splitter *use case* and includes a number of constraints/indications provided by FE specialists (see text in curly brackets). According to this use-case, Surface-splitter has been developed such that the "*number of meshable surface segments is kept to a minimum*" (as discussed in the next section, the generation of meshable surfaces might require cutting the structural elements into more and smaller segments than those obtained just by trimming them along their reciprocal intersections). When necessary, Surface-splitter generates triangular surfaces, although "*in the least amount possible*" and "*with the least possible sharp angles*" in order to limit the generation of "unhealthy" finite elements with too high aspect ratio. Even if the CM purpose is to automate as much as possible the segmentation process, the designer is allowed to "*insert virtual elements by hand to support the model segmentation*".

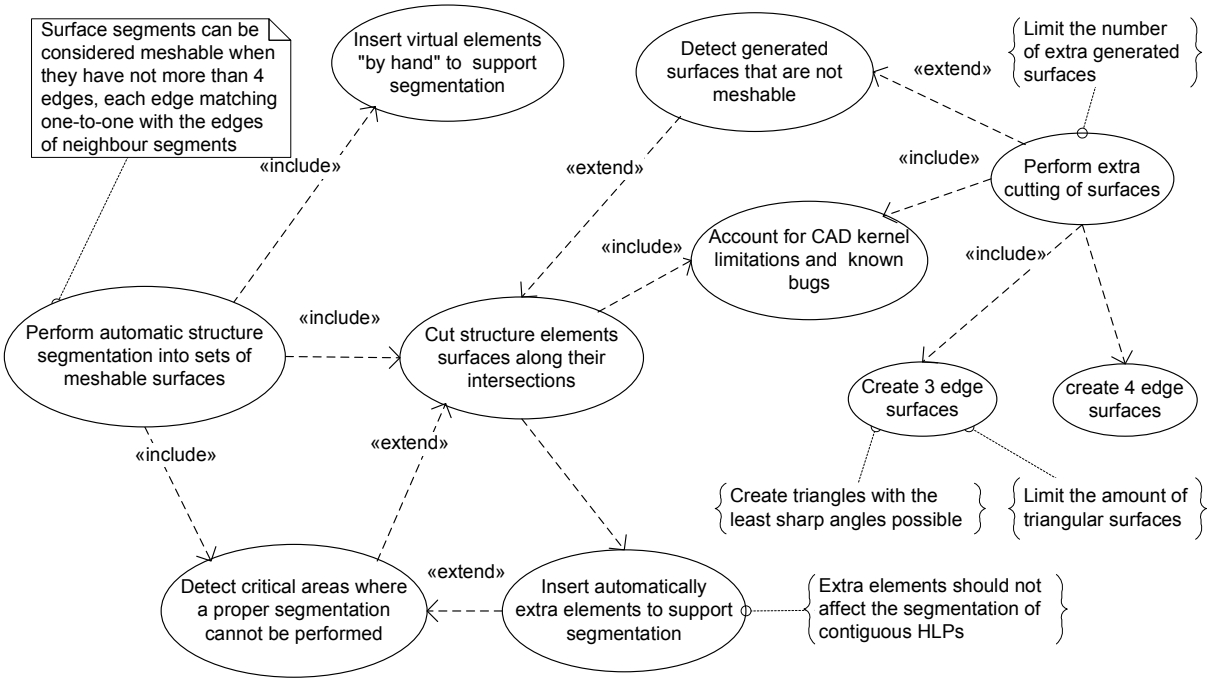


Fig. 6.12: UML use case for the KBE system to perform the automatic surface segmentation of a generic geometric model

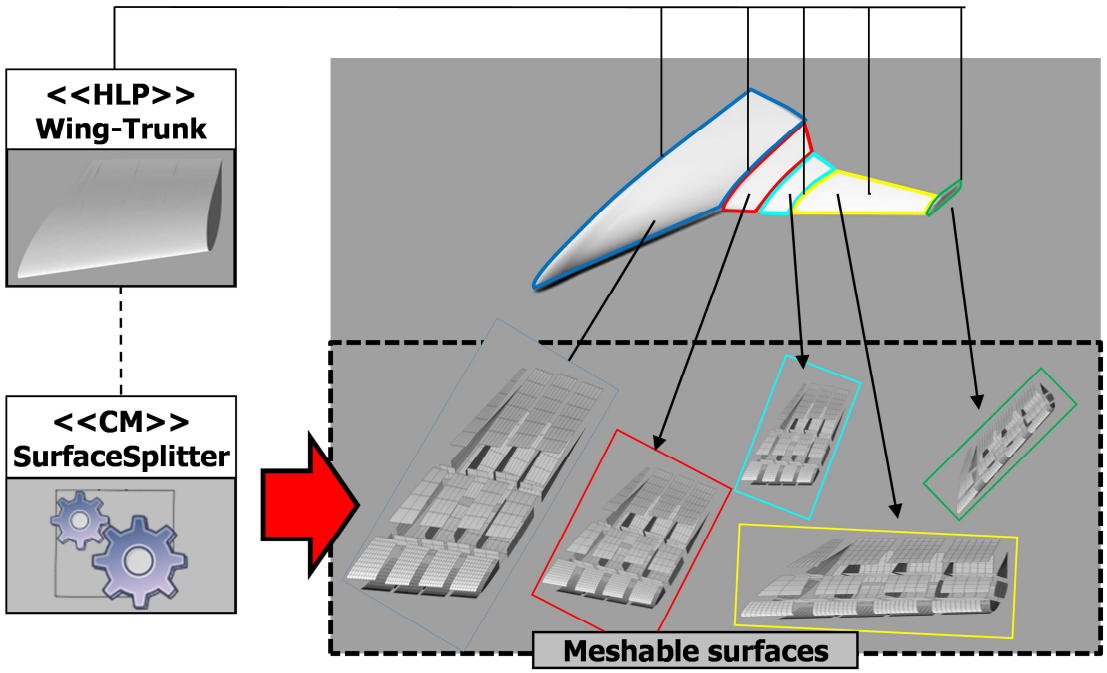


Fig. 6.11: The Surface-splitter CM processes the outer skin and the internal structure of each HLP instance that is used to define an aircraft, into sets of meshable surfaces.

The activity diagrams that document the detailed implementation of the automated segmentation process can be found in Appendix L. The example of a surface segmentation process is described in the next subsection, to show how the Surface-splitter Capability Module actually works.

6.3.1 Surface-splitter, a Capability Module for automatic generation of meshable surfaces.

A generic instantiation of a wing-trunk HLP is shown in Fig. 6.13 (top). Whether such instantiation belongs to a BWB aircraft component, a conventional wing, a tail empennage, or a control surface, is not relevant. The Surface-splitter Capability-module operates on any HLP instance, independently from the instantiation purpose.
The Wing-part instance considered in this example has four spar elements, defining and confined to the wing-box (WB) area, and a number of ribs and riblets crossing the wing-box and/or the leading (LE) and trailing (TE) edge areas. Note that some of these rib elements start and end at either a spar or the LE/TE line, whilst others start or end at the root or tip section of the given wing element (i.e., LE riblet 4, Ribs 1, 3, 4, 5 and TE Riblets 1 and 2). The latter are likely to cause troubles during the segmentation process.

The segmentation process takes place through the following steps:

Step 0
The first operation performed by Surface-splitter is the intersection of the wing-part skins with all the spars and all the ribs. Also, each rib is intersected with all the spars and all other ribs and, finally, all the spars are intersected with all the ribs and all other spars.
Some of these intersection operations might produce no result, but this is handled by the CM without causing any runtime error.
This intersection process delivers sets of spar, rib and skin segments, which are subsequently scanned for non-meshable surfaces. As highlighted in Fig. 6.13 (mid), five non-meshable skin elements (i.e., elements with more than four edges) are detected. As anticipated, they are caused by those ribs and riblets that either start or end at the root or tip section of the wing-part.

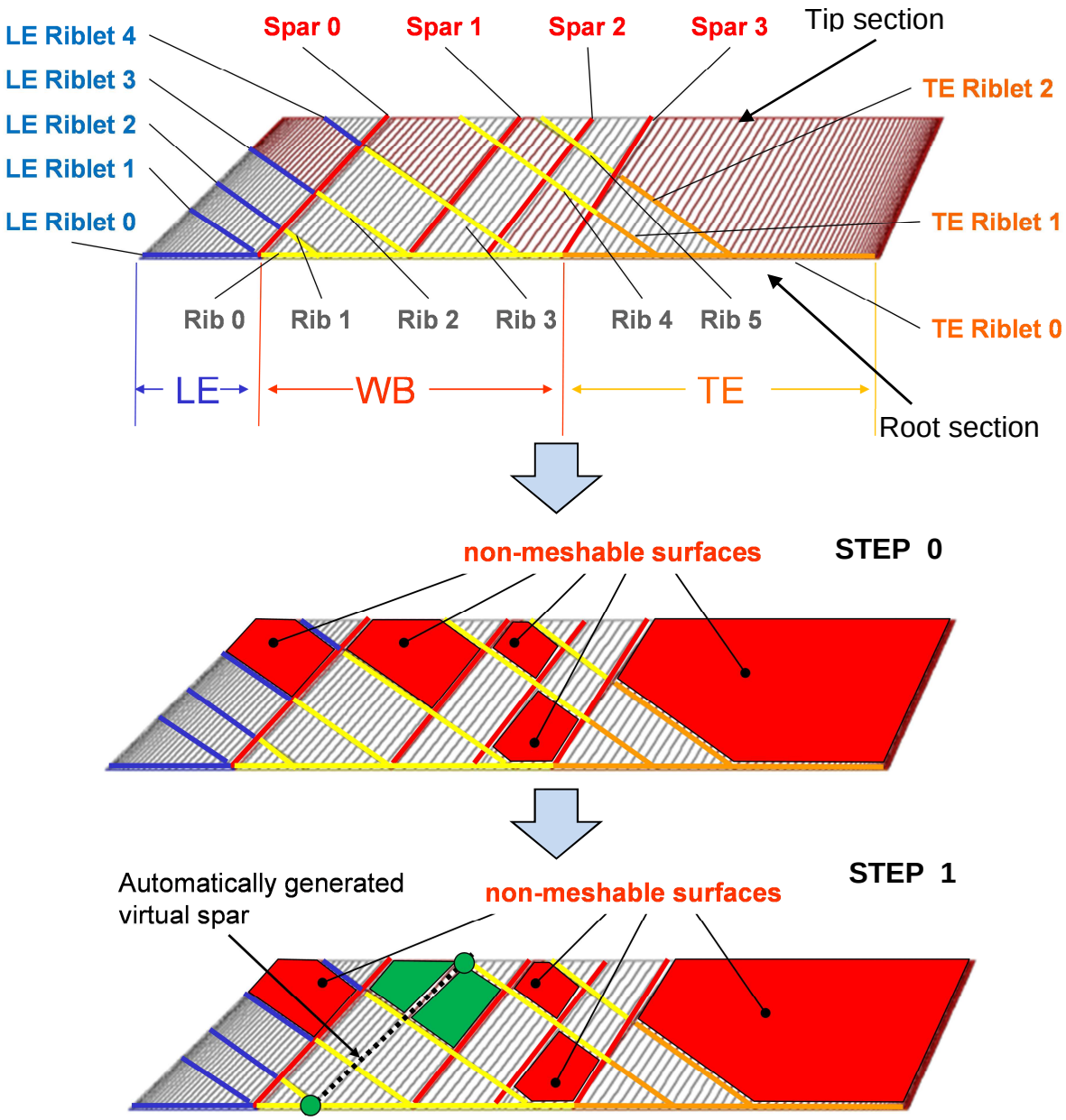


Fig. 6.13: Knowledge based segmentation process of a generic wing like component.

Step 0: skins, spar and ribs are intersected with each other and non-meshable surfaces are detected.

Step 1: a first extra segmentation process resolves some non-meshable surfaces, by automating the generation of some virtual spars.

Step 1

As a FE specialist would, Surface-splitter finds out that, within the skin panel delimited by Spar 0 and Spar 1, it is possible to fix at least one non-meshable surface

by forcing the generation of a virtual spar[7]. As shown in Fig. 6.13 (bottom), the position of the extra virtual spar is automatically defined by the points where Rib 1 and Rib 4 intersect the root and tip section of the wing trunk.
It is evident that the generation of one virtual spar was not sufficient to fix all the non-meshable surfaces, however, in this specific case, only Rib 1 and Rib 4 could be used to define the extra cutting element (i.e. the virtual spar).

Step 2
Indeed, the generation of any other virtual spar to fix the remaining non-meshable surfaces could affect the segmentation of possible Wing-part instances adjacent to the wing-part under consideration. This is not desirable, because it would increase the complexity of the overall segmentation process, as well as the total number of surface segments. Therefore, as a FE specialist would do, Surface-splitter checks the situation at the root and tip border of the given wing-part and allows the generation of extra virtual spars, only if not affecting an adjacent Wing-part instance. The following four cases illustrated in Fig. 6.14 exists (check the left side icons):
Surface-splitter realizes that the wing part to be segmented has a "tip neighbour". Therefore, spar points are automatically generated on the "free" edge (at the root) and used to generate extra virtual spars. As result of this second segmentation step, other two non-meshable surfaces get fixed.
Surface-splitter verifies that it is possible to generate support points for extra virtual spars only at the tip section. Also in this case, other two non-meshable surfaces get fixed.
The presence of adjacent wing trunks, both at the root and tip side, makes this second segmentation step useless.
In the case of an isolated Wing-part instance, this segmentation step is sufficient to obtain all meshable surfaces.

Step 3
In this step, each of the remaining non-meshable surfaces is cut, *individually,* in two segments, using a cutting line passing through two non-contiguous vertices of the given non-meshable surface. As illustrated in Fig. 6.15, all the remaining non-meshable surfaces from the previous steps get finally fixed and without affecting any adjacent wing parts.

---

[7] As introduced in Chapter 5, virtual spars are not real structural elements that access the FE analysis, but, as the real spars, they can be used to position other structural elements and do affect the surface segmentation.

The automatic generation of virtual spars to tackle segmentation issues can be switched off via the MMG input file. In this case, the designer is responsible for the definition of the required virtual spars.

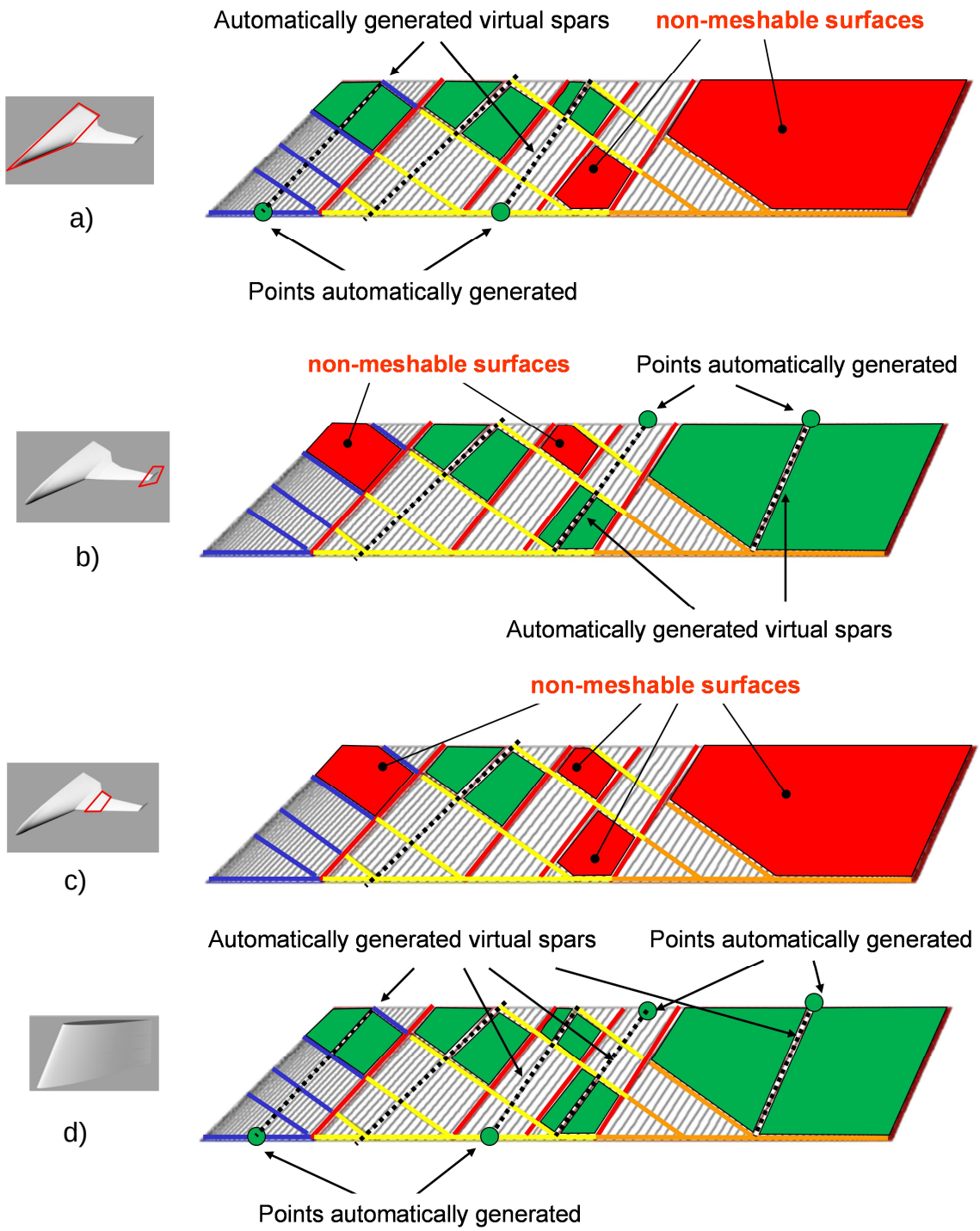


Fig. 6.14: Knowledge based segmentation process of a generic wing like component.

<u>Step 2</u>: Some non-meshable surfaces can be fixed by generating extra virtual spars (only when the extra segmentation is not perturbing eventual adjacent wing-trunks). In case of isolated wing trunks (d), the process is completed successfully. In case of adjacent wing parts both at the root and tip sections (c), Step 2 is ineffective.

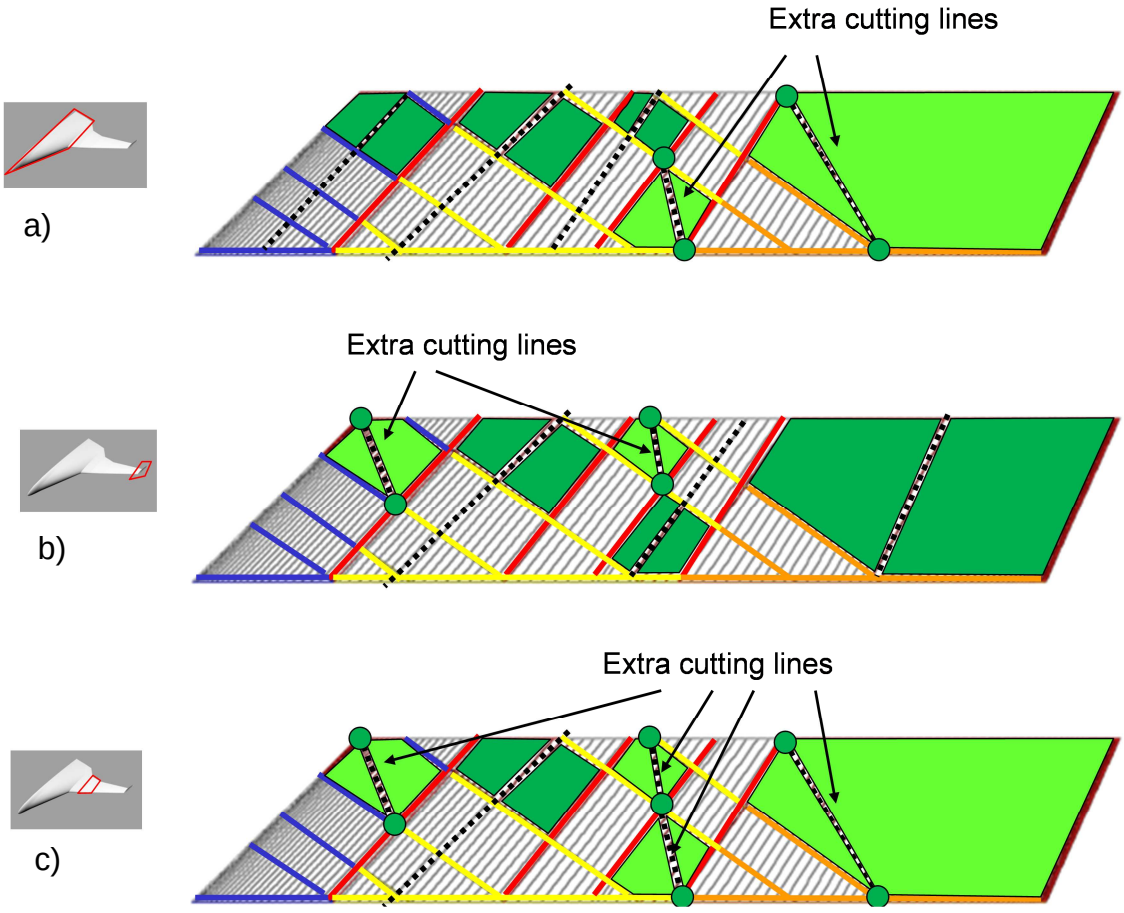


Fig. 6.15: Knowledge based segmentation process of a generic wing like component.

<u>Step 3</u>: All the remaining non-meshable surfaces are fixed by splitting them in two opportune segments, without affecting the segmentation of adjacent wing parts.

As a FE expert would do, Surface-splitter uses this segmentation approach only as last resource, because it is likely to generate triangular surfaces, which are not "as good as" quadrangles for FE analysis.
In facts, there are more ways to split a pentagonal or hexagonal surface (surface with even more edges are not likely to occur) in two meshable surfaces. As demanded in the use case of Fig. 6.12 , Surface-splitter selects the combination of segments with the least sharp internal angles. The detailed process to select the best cutting approach is illustrated in the activity diagram of Fig. 6.16.

Finally, in case a non-meshable surface problem cannot be solved, due, for example, to an untrapped error of the CAD geometry manipulations, the MMG will automatically label the given surface with a special "non-meshable" tag and highlight it in red in the MMG graphical browser.

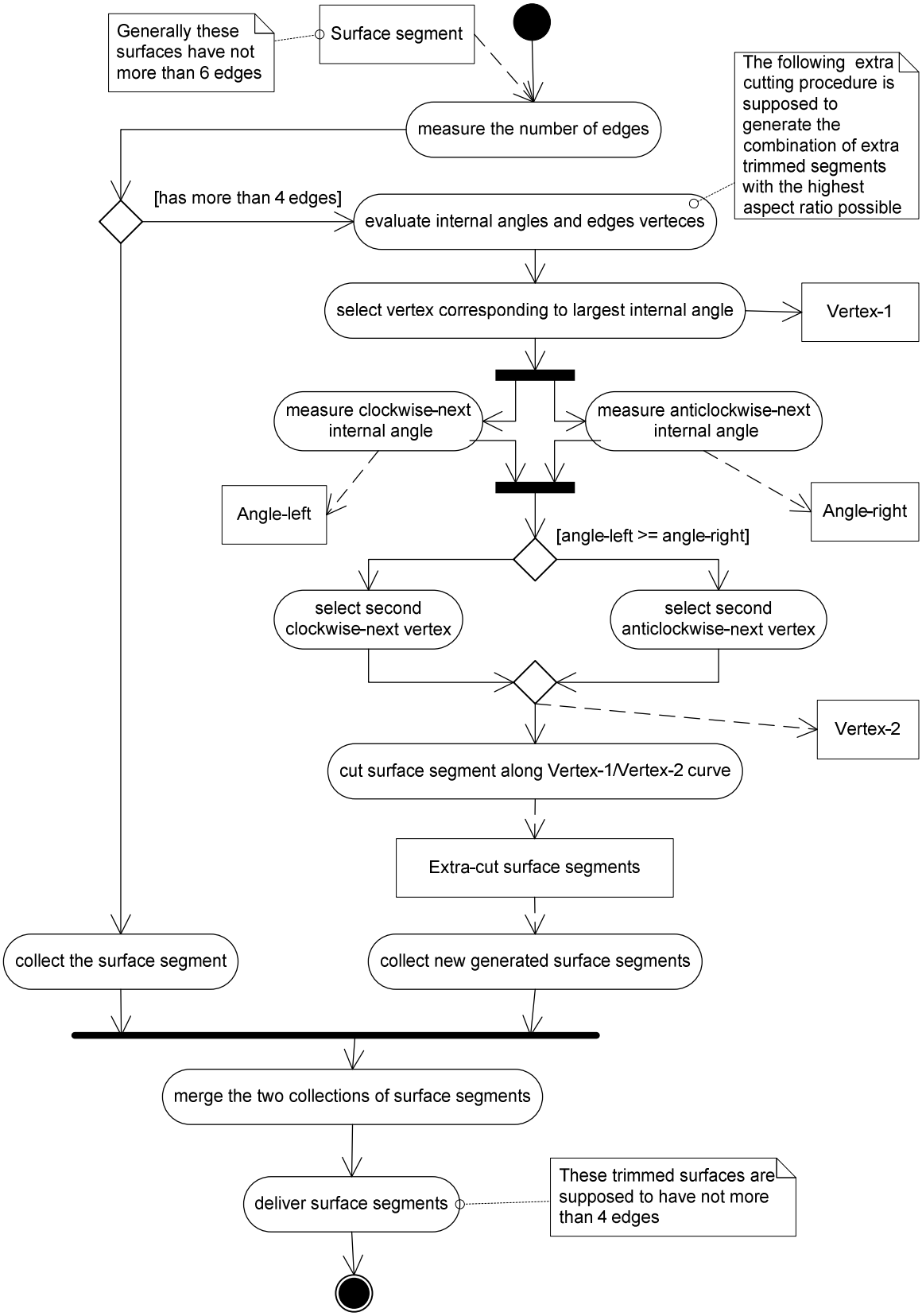


Fig. 6.16: activity diagram detailing the extra-cutting process (Step 3) to deal with surfaces segments that have more than four edges.

6.4 MMG – FEA environment integration

In this section, the approach developed for a seamless link between the MMG and the PATRAN/NASTRAN finite elements environment (FEA) is described.
Indeed, the automatic surface segmentation process described in the previous section is just one of the steps towards the automated generation of FE models.

6.4.1 Extraction of geometry and metadata from the product tree

Every time a surface segment is generated, an *identification tag* is automatically attached to it to record its membership (i.e., the HLP instance and the kind of structural element from which the segments has been derived). Once the segmentation process has been completed, the tags enable a dedicated scanner

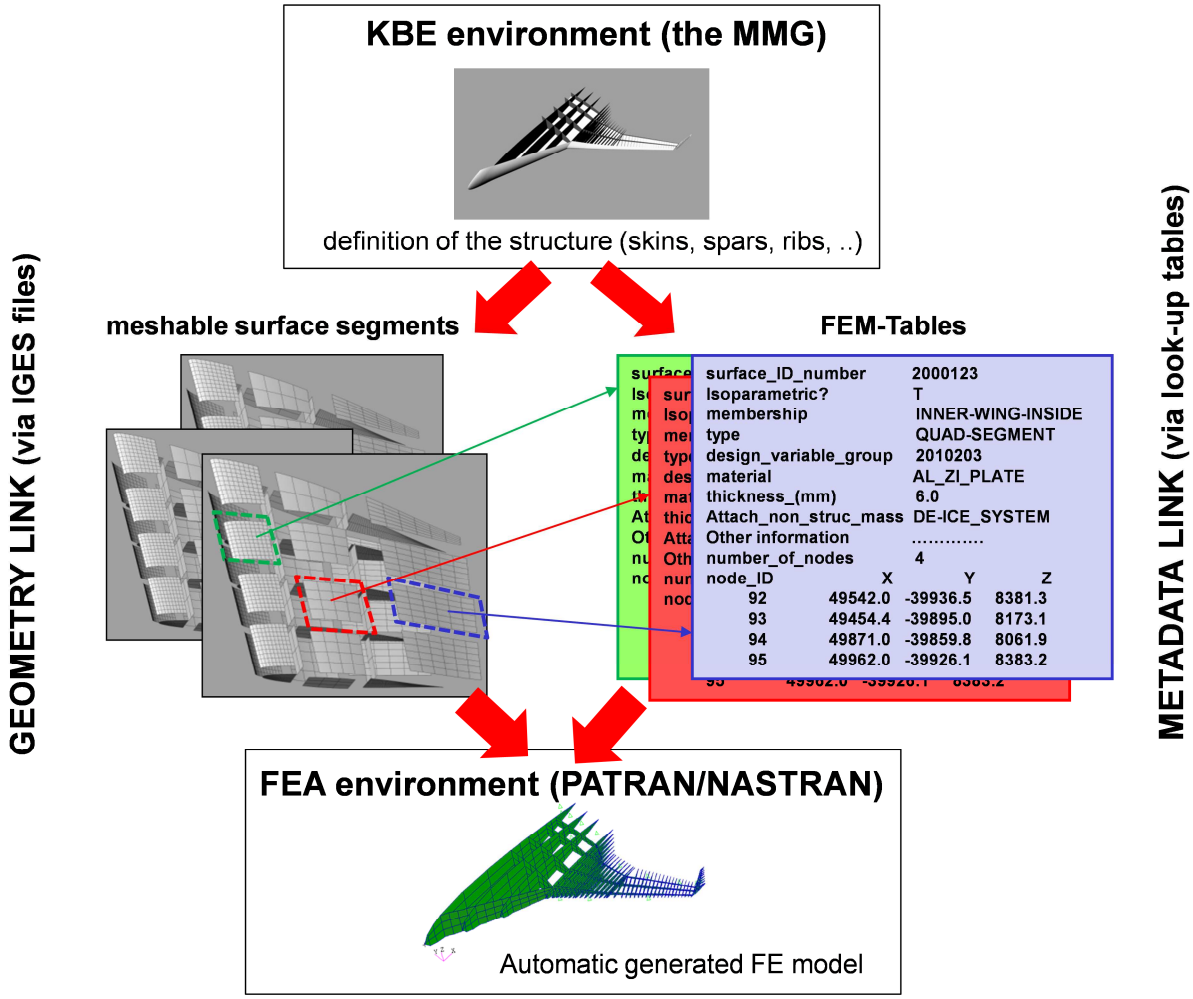


Fig. 6.17: geometry and metadata transfer from the MMG to the FEA environment. IGES files are used to transfer the geometry of the segmented surfaces; look-up tables (FEM-Tables) are used to transfer the information related to each surface segment and required to set up the FE model.

module (see Section 4.8.1) to parse the product tree, collect all the segments and sort them in groups. Finally, these groups of surface segments are exported to the FE environment by means of IGES files (see Fig. 6.17, geometry link).
Since the IGES format can transfer only geometry information, a complementary link is created to export also the relevant non-geometric information required to set up the FE model. For this purpose, the MMG automatically generates one look-up table for each surface segment that is exported via IGES (see Fig. 6.17, metadata link). The look-up tables, addressed in this work as *FEM-tables*, contain information such as thickness, material, membership identification, 'meshability', list of non-structural mass items to be attached, design variable group, Cartesian coordinates of the corner points, and others attributes of all the surface segments generated by Surface-splitter.
Similar to the collection mechanism for the surface segments, the FEM tables are generated by means of a scanner module that parses the whole product tree and extract from each segment the attributes required to compile the related FEM-table.
A report writer has been developed to encode all the FEM tables into one XML file.

6.4.2 Automated FE model generation

The actual KBE-FEA environment interface is enabled by an in-house developed Python application, called PYCOCO (Nawijn et al., 2006). Via a client-server mechanism, PYCOCO generates on the fly instructions for MSC PATRAN[1] and guides it in the process of building up a NASTRAN model.
The main steps are the followings:

- PYCOCO forces PATRAN to open an empty database and import all the surfaces segments delivered via IGES files.
- PYCOCO reads the FEM-tables and compare the Cartesian coordinates of the corner points of each surface segment with those of the surfaces in the PATRAN database. Indeed, the coordinates of the corner points represent the one-to-one link between the geometry entities generated by the MMG and their corresponding representation in PATRAN (see sketch in Fig. 6.18).
- As soon as a match is found, all the relevant information (material, thickness, etc.) stored in the given FEM-table is automatically mapped on the corresponding representation of the surface segment in PATRAN.
- The meshability information contained in the FEM table is used by PYCOCO to instruct PATRAN on the mesher to employ. Quad elements are meshed first, using Isomesh. Triangular elements are meshed later using Paver (for

[1] These instructions are actually given as PCL commands. PCL, which stands for PATRAN Command Language, is the scripting language provided by MSC to operate PATRAN from the command line, or via session files.

unstructured mesh). The size of the mesh elements can be calculated by PYCOCO, based on the size of the structural elements (Pearson, 2001).

- PYCOCO reads the non-structural masses (NSMs) table produced by the MMG (see example in Table 4.3) and forces PATRAN to generate, next to the aircraft model, a set of lumped masses (position and mass value as indicated in the table).
- Based on the list of NSMs indicated in each FEM table, PYCOCO forces PATRAN to build a set of connection elements (RBEs) linking the relative lumped mass(es) to the corner points of the given surface segment (see sketch in Fig. 4.10).
- Loads and boundary conditions are applied.
- In case the FE model must be used also for structural optimization (e.g., by means of the NASTRAN optimization solver Sol 200), each surface segment is collocated in a certain design variable group, according to the dedicated variable group identification code, contained in the FEM table. Refer to Appendix M and (La Rocca and van Tooren, 2002a; La Rocca et al., 2002) for further details.
- Finally, the FE analysis (and/or optimization) is performed using NASTRAN.

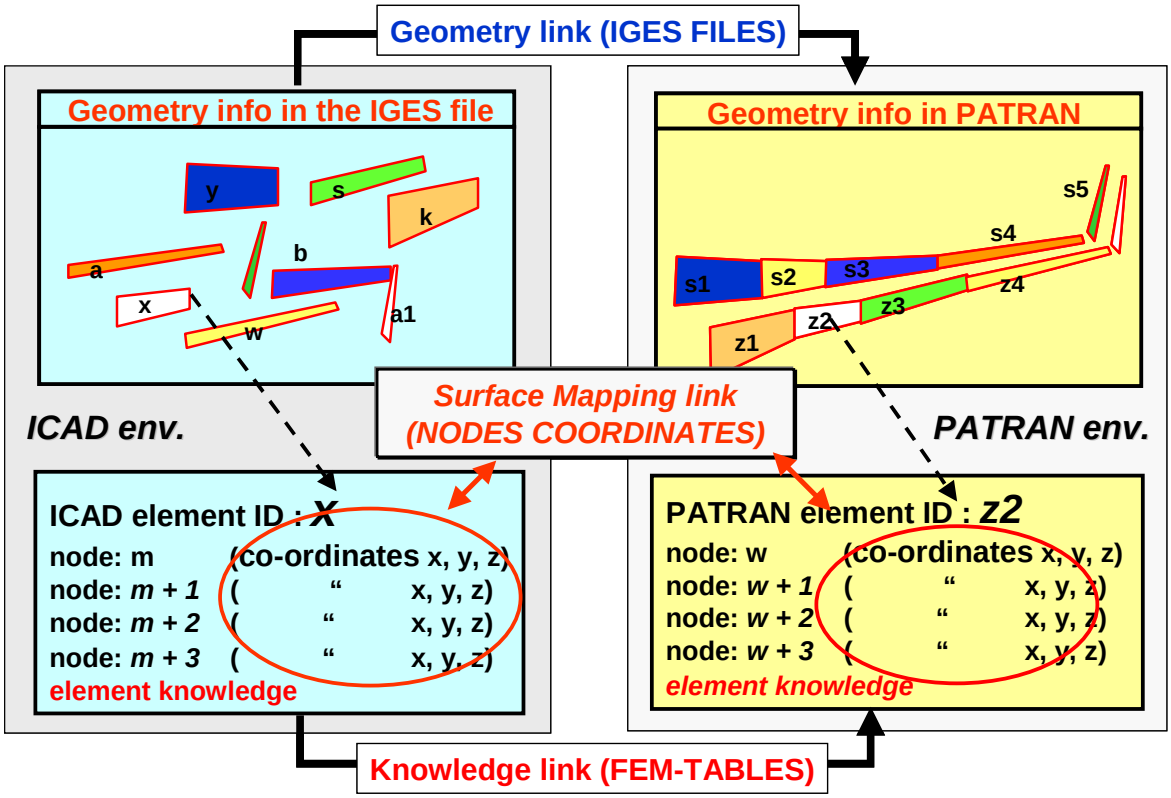


Fig. 6.18: The mapping process of the FEM-Table content is based on the match of the Cartesian coordinates of the corner points of the surfaces stored in the PATRAN database, with the Cartesian coordinates reported in the FEM-tables.

6.4.3 On the convenience of the selected integration approach

The implemented approach to link the MMG with the FEA environment has delivered a powerful and seamless modeling and analysis system. The designer is free to evaluate many different design configurations, without worrying about rebuilding a new FE model each time a variation is enforced in the shape or topology of the aircraft configuration.

Since the whole process works fully hands-off, it enables the set up of multi level optimization studies: an optimizer can vary the top level parameters of the aircraft (shape, structure layout, etc.) via the MMG input file, while at each analysis cycle a FE–based optimization process can be performed, for example, to size the given structure components for minimum weight (section 6.6.

The approach to the automation of the FE model preparation process using PYCOCO, follows the paradigm of Knowledge Based Engineering. In this case, however, the object oriented programming and rule base are not combined with a parametric CAD engine, as in a true KBE system, but with a FE pre-processor and solver. Here, Python provides the object-oriented and rule based design features, while the PATRAN/NASTRAN combination represents the functional engine.

An alternative approach to the one implemented in this work would be the direct generation of the complete NASTRAN model by the MMG. A relevant example of FE model generation using the ICAD system can be found in (Rondeau et al., 1996; Rondeau and Soumilas, 1999). Indeed, this approach would not require the critical process of extracting data and information from the MMG product tree and put it back together in the context of the FEA system. Whilst attributes and relationships of any product tree entity are immediately accessible within the MMG, rebuilding such information, especially the relationships between entities, in another system is a challenging task.

However, once achieved, it comes with relevant advantages:

- The meshing capability of PATRAN is fully exploited, without the need of "re-inventing" a mesh generator in ICAD
- Once PATRAN has completed the model pre-processing, it can generate the input deck for all the supported solvers, "for free". Hence, not only NASTRAN, but also ABAQUS, ANSYS, LS-DYNA and other FE packages become immediately available.
- A different MMG, able to deliver the same output files (i.e., meshable surfaces plus FEM-tables), would be directly endowed with FE analysis capabilities.

- Two teams of experts (KBE developers and FEA specialists) can collaborate on the development of a system like the one discussed in this work, each one working in parallel with its most familiar and trusted software tool[2]
- In collaborative projects involving the participation of different disciplinary teams from various companies, as well as the use of licensed software, this approach can be very convenient.

6.5 Operating the MMG

The architecture of the MMG operational environment is sketched in Fig. 6.19. Indeed, it does not differ from the typical product model architecture shown in Fig. 3.13.

Operating the MMG requires just three steps: preparation (edit) of the input deck, launch of the MMG in batch mode (or interactive operation of the MMG via the ICAD user interface), retrieval of the generated results (reports)

The first step consists of the preparation of 4 input files:

1. The **main input** file. Here, organized in sections, all the parameters required to instantiate the aircraft metamodel are contained (snippets of this file are provided in Appendix D, G, I)
2. The **design variable groups definition** file. Here are the parameters available to the user to sort the aircraft structural segments into separate groups for structural optimization (all the surface segments contained in a group will get the same thickness value from the sizing process). See Appendix M for more detail.
3. The **non-structural masses definition** file. Here the value of each non-structural mass must be provided. The MMG will use these values to produce the NSMs table (after having scaled those values according to the length or the area of some instantiated aircraft components. See section 4.5.3)
4. The **reports list**. This is the list of all the output files the MMG is capable to produce (see section 4.8.1 on report writers). This file is required only when operating the MMG in batch mode (i.e., hands off)). In this case, the user has to select (switch on/off) from the complete list the ones for which it is required to launch the MMG.

On the sole basis of these four files (which can be edited as plain text files, either by a designer or another software tool), the MMG can be instantiated and requested to produce any of the output models described so far.

[2] The same consideration applies for the method described in section 6.2.3, where COALA enables the integration of the MMG with VSAERO

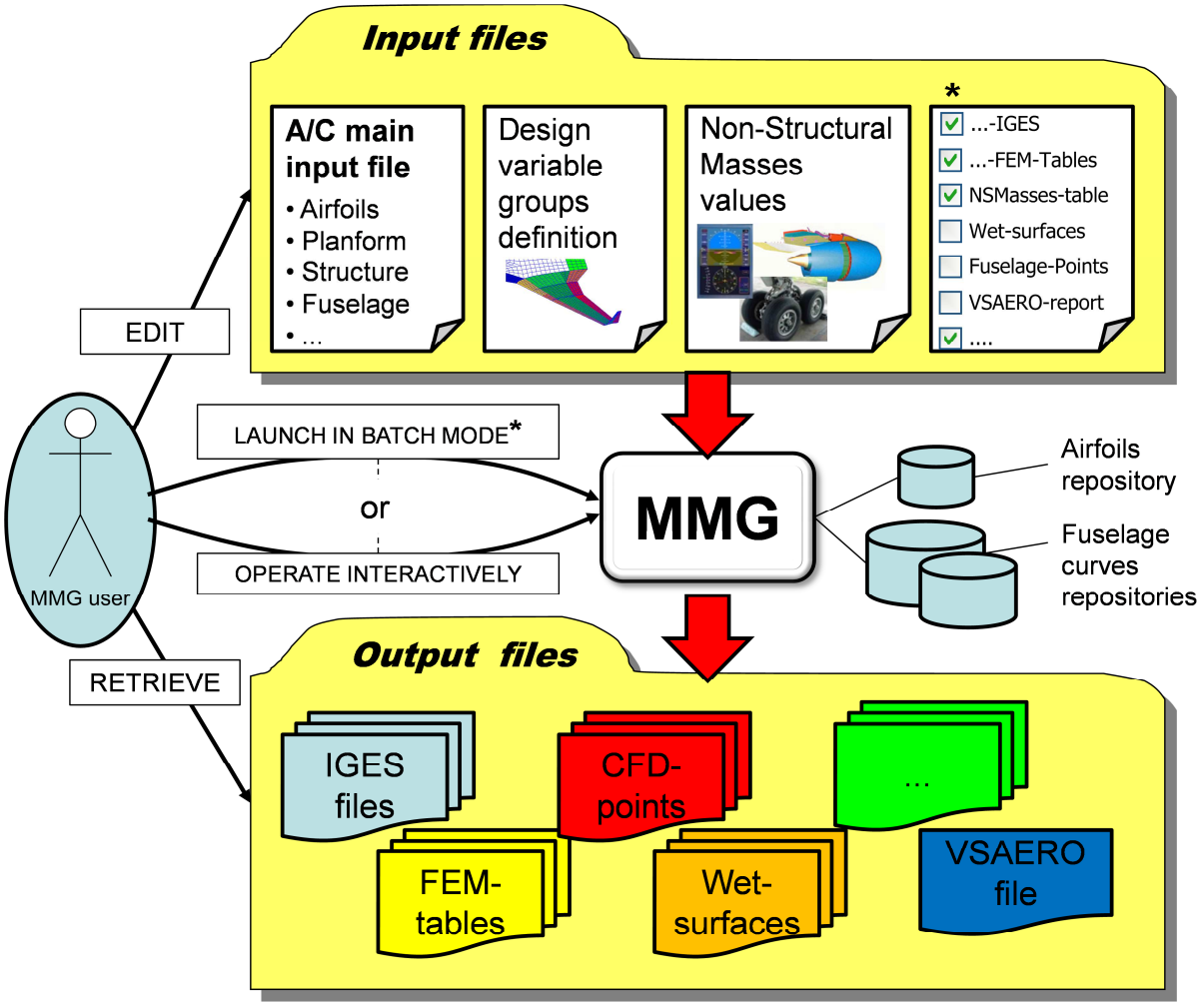


Fig. 6.19: operation of the MMG and its input/output architecture.

This can be done either interactively (sitting behind the system where ICAD and the MMG are installed and using the standard ICAD user interface) or in batch mode, hence running the MMG with a single command and waiting for the demanded results (from the reports list file), without any single user intervention.

Batch operations can be easily performed also in remote, hence accessing the MMG via a web connection from another system, where ICAD does not need to be installed at all.

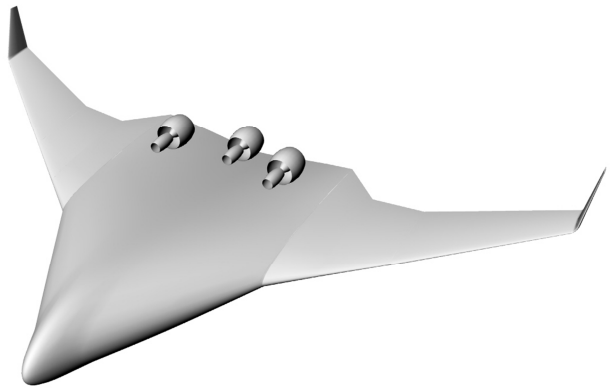
The possibilities to operate the MMG in batch mode as well as in remote represent indeed a great asset (as well as the biggest development challenge). Many non-geographically collocated users can use the same MMG, submit their customized version of the input file and ask for required output reports; in fact using the MMG as Software as a Service (SaaS), according to cloud computing terminology. All the results generated using the same version of the MMG and its input files are guaranteed to be consistent and repeatable.

As such, the MMG stands as a real enabler for distributed multidisciplinary design optimization. As demonstration, two relevant cases are described in the next sections, where the MMG capabilities have been exploited to support the design and optimization of a novel aircraft configuration and a main component of a conventional passenger aircraft.

6.6 Study case 1: The MOB project

A successful validation case of the Multi Model Generator concept presented in this work is provided by the Fifth Framework European project MOB, on multidisciplinary design and optimization of blended wing body (BWB) aircraft configurations (Morris et al., 2004; Morris, 2002). The primary objective of the project was the development of a computational framework (the so called Computational Design Engine, CDE; a prototype of the DEE concept addressed in Chapter 2) for distributed design and optimization. The secondary purpose was to demonstrate the CDE by application to a problem of intrinsic interest, namely a BWB aircraft; a potential competitor to the Airbus superjumbo A380 and with some relevance to military aircraft design. The BWB configuration was ideally suited to function as driving scenario due to its inherent strong couplings between disciplines. Besides, the lack of reference data and experience made the use of design handbook methods impractical and increased the need of data from physics-based analytical models.

The baseline design was provided by Cranfield University and Saab Aerospace (geometry model and main specifications in Fig. 6.20).



- Span: 80m
- Maximum Take Off Weight: 300t
- Payload capacity: 113t (174 LD3 containers over a double deck cargo hold)
- Cruise speed: Mach=0.85, at 35000 ft
- Range at max payload: 11000Km
- Approach speed: 140 knots

Fig. 6.20: Model of the MOB reference BWB and list of main specifications

6.6.1 Set up of the MOB multidisciplinary design optimization system

A multi-level, multi-fidelity, distributed MDO system was put in place to optimize the baseline aircraft for maximum range, while maintaining payload capacity and

maximum take off weight[3], and guarantying inherent stability and controllability. Indeed, the baseline aircraft configuration appeared not controllable longitudinally and directionally unstable.
The main disciplines involved in the optimization exercise were aerodynamics, structures, flight mechanics and aeroelasticity. For the aerodynamic analysis, a range of tools have been employed, including simple panel codes, Euler codes, and, in limited extent, full Navier-Stokes methods to predict the maximum aerodynamic efficiency, the aircraft maximum lift coefficient and the stall angle (Laban et al., 2002; Qin et al., 2002).
Concerning the structure analysis, simple bending beam theory has been used for preliminary weight estimation, whereas full blown FEM-based optimization techniques, including aeroelastic constraints, have been used for a more detailed sizing of the various structural components. For the flight mechanics, basic stability and control analysis methods have been used to assess stability and control. A system of trim tanks with fuel transfer scheduler has been developed on purpose. Handling qualities in closed loop have been addressed as well, including pilot response at the simulator (Stettner and Voss, 2002).

Concerning the optimization strategy, a multi-level optimization process was set up, based on a *global level* loop, where only few design parameters (wing thickness, twist and sweep, fuselage length and camber) were used to affect all the disciplines, and a *local level* loop for the aircraft structural design, where several hundred groups of FE elements thicknesses were used as design variables.
For the global level, a response surfaces strategy was preferred over gradient based optimisation schemes, mainly because of the unavailability of sensitivity information from many of the analysis modules. Anyhow, response surfaces are an excellent means to visualise trade-offs, at least when the number of variables is limited as in this case. For the local optimization level, on the other hand, it was decided to make use of gradient based techniques, considered the availability of sensitivity information as well as the large number of variables (Laban et al., 2002).

6.6.2 Role and operation of the MMG in the MOB computational framework

The role of the MMG was pivotal in the set up of the complex and distributed MOB computational framework. The MMG was able to model a large number of variants of the reference BWB in batch, surviving all the geometry variations imposed by a remotely located optimizer. For each variant it was able to extract, in full automation,

[3] This was a simplifying decision to avoid performing expensive calculations at each structural weight variation. Indeed, the weight of engines, landing gears, etc. varies with the MTOW, hence each structural weight variation would require another structural weight estimation. Here it was decided to "invest" every saved structural weight kilogram into fuel.

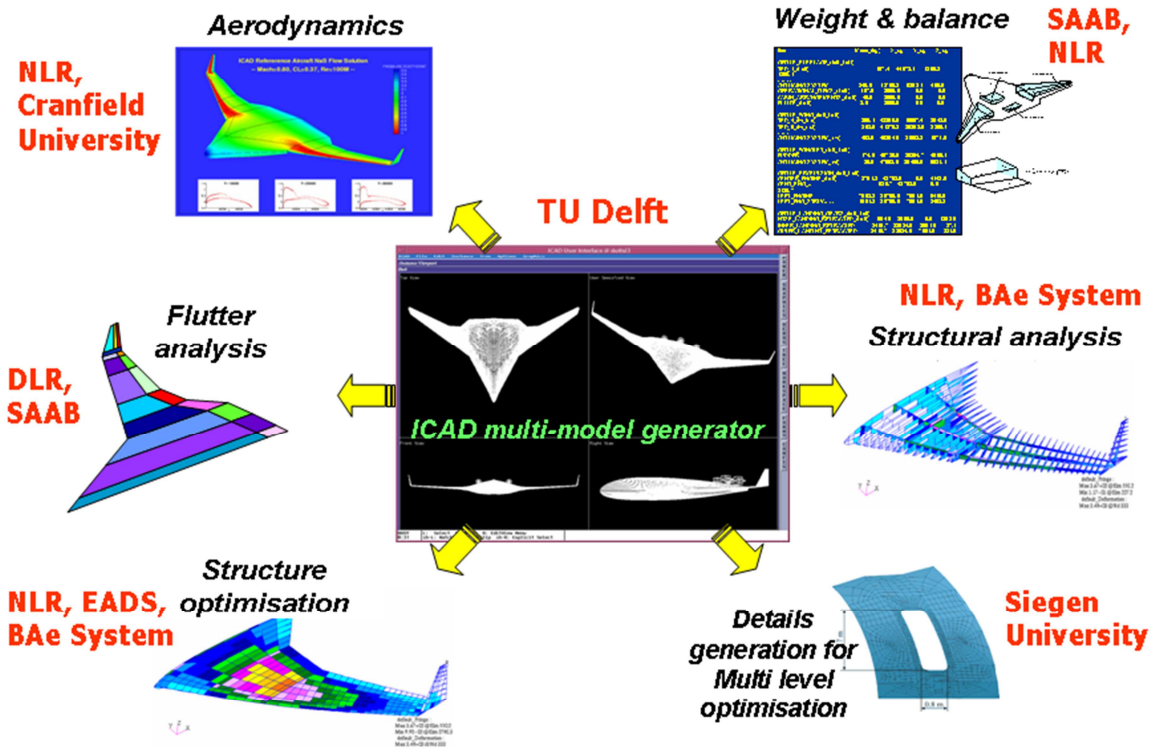


Fig. 6.21. Role of the MMG within the MOB distributed MDO framework. The MMG provides dedicated models to a large set of distributed analysis tools, both low and high fidelity, in-house developed and commercial of the shelf.

sets of different, yet coherent sub-models, tailored to the broad range of analysis tools, both COTS and in-house developed, provided by partners from industry and academia (Fig. 6.21).

The MMG delivered models for both low and high fidelity aerodynamics (PANAIR, MERLIN, ENFLOW), 2-D planform models for aeroelastic analysis (NASTRAN and ZAERO), structural models for FEA (PATRAN/NASTRAN), including the definition of the design variable groups for structural optimization (Fig. 6.22-left and Appendix M), and the c.o.g. distribution of the fuel tanks and the non-structural masses (with weight scaling of de-icing systems, and trailing edge movables actuators). The MMG could also extract the geometry of a door cut-out, to support a further level of detail in the structural optimization loop (more detail in (Engels, Becker and Morris, 2004) and Section 6.8).

A software communication framework was in charge to feed the MMG with the set of edited input files and to extract and distribute the generated models, always via web connections (Vankan and Laban, 2002).

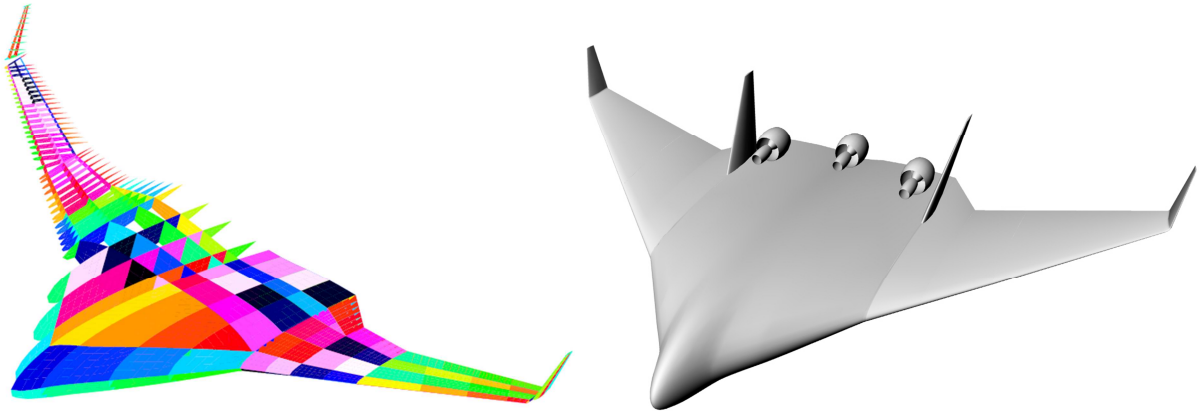


Fig. 6.22: (Left) Visualization of the more than 200 thickness variables defined for the FE-based structural optimization (Laban et al., 2002). (Right) The BWB variant with vertical fins, developed to improve the directional stability of the reference aircraft.

6.6.3 Results

Once the MOB computational framework was in place, more than 50 aircraft variants have been evaluated, by means of both low and high fidelity tools, totally hands off, running on a number of computers distributed across the multi-national consortium. All the computations have been performed in the time frame of just a couple of days, whereas it would have taken months without the use of such a design and optimization framework.

The intermediate results of the top level optimization process were presented to the design team as response surfaces, providing the designers important insights about the effect of certain design parameters on the objective and constraints function, hence guiding the next steps in the optimization. For example, it was observed that all the selected variables (wing thickness, twist and sweep, fuselage length and camber) affected controllability, but no one was individually able to bring it to an acceptable value. Eventually it was found that a combination of more negative wing twist, shorter fuselage length with increased aft-camber was necessary to restore controllability. However, with a negative impact on the range, which was reduced to 9900 Km (Fig. 6.23). Furthermore, it was found that a fuel drain scheduling system plus two trim tanks in the BWB center section were necessary, as well as a full leading edge slat for low speed operation.

Eventually, the aircraft remained directionally unstable, such that, in a later stage of the project, a BWB variant with vertical fins (Fig. 6.22-right) was considered, however, with further consequences on the initially predicted aerodynamic efficiency.

Among other things, the project showed that traditional handbook methods, when applied to a novel aircraft configuration such as a BWB, might not be able to deliver a feasible baseline design, while they tend to predicted too optimistic performances. Besides, the strong disciplines coupling typical of BWB aircraft is such

that non intuitive combinations of parameters are necessary to obtain a feasible design.

More details on the MOB optimization process can be found in ref. (Laban et al., 2002; Morris et al., 2004). A more recent study on the controllability of a blended wing body aircraft, still involving the use of the MMG, is reported in (Voskuijl et al., 2008; Dircken, 2008).

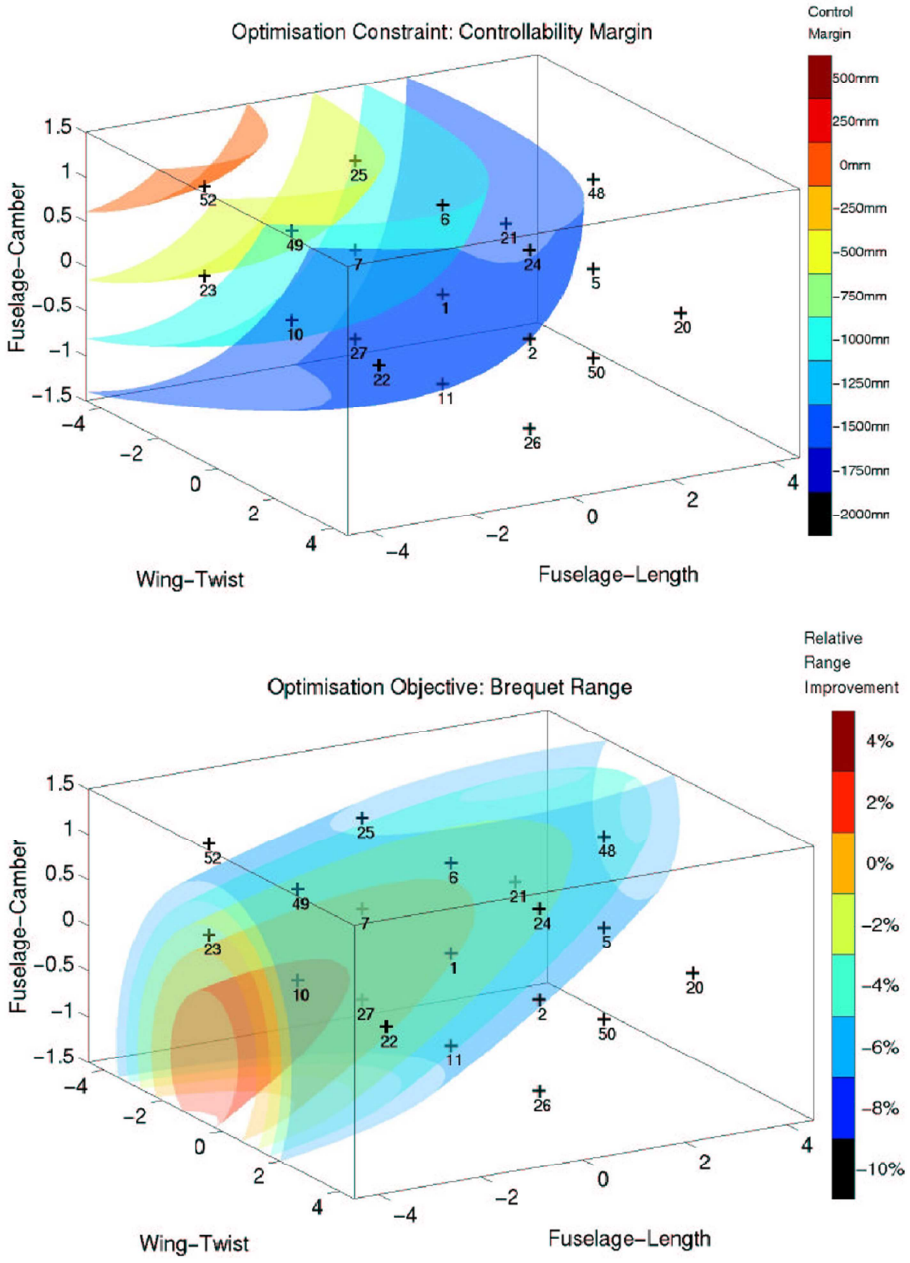


Fig. 6.23. Results of the analysis and optimization process of the MOB BWB, visualized by means of response surfaces. Effect of the most effective combination of design variables on the controllability constraint (top) and on the range objective (bottom).

6.7 Study case 2: Vertical tail redesign study

A Design and Engineering Engine to support the redesign of the vertical tail of a passenger aircraft has been developed in collaboration with the Loads and Aeroelastics group at Airbus Germany. Scope of the project was the development of a computational system to facilitate a fast assessment of different tail design options, such as material and planform shape (Cerulli et al., 2006). The challenge was the rapid generation of aeroelastic and structural models of adequate fidelity, which are usually not available in the parametric way required to perform *what-if* studies in the preliminary design phase, neither cannot be built in a sufficiently short time.

The implemented design process is sketched in Fig. 6.24 and can be summarized as follows. First, the MMG is used to model the configuration of the original vertical tail and to generate automatically the relative sets of segmented surfaces and FEM tables. These are subsequently exported to the PATRAN/NASTRAN environment by means of an extended version of PYCOCO, which request NASTRAN to generate a reduced model of the tail (Cerulli, van Keulen and Rixen, 2007), by condensing the structural and non-structural fin masses on a set of dedicated (condensation) points. For any vertical tail instance, the condensation points are automatically generated by the MMG capability module Condensed-Masses-Generator (already mentioned in section 4.7.1).

At this point, the reduced model of the new tail is connected to the previously generated reduced model of the original aircraft (less the vertical tail). The connection points are also exported by the MMG together with the condensation points.

The obtained reduced model of the complete aircraft model is then fed to VarLoads (Hofstee et al., 2003), a load analysis tool developed at Airbus, where a dynamic yawing maneuver simulation must be performed to predict the load distribution on the fin. On this purpose, VarLoads makes use of proprietary Vortex Lattice Method, which needs a flat panels representation of the tail. Also in this case the model is provided by the MMG, directly in the required NASTRAN CAERO cards format (see section 6.2.4).

The estimated tail loads, are subsequently taken by the DEE initiator/sizing tool (Schut and van Tooren, 2007) and mapped from the condensed mass model to another simplified tail model, which is used to size the tail structural components and produce a weight estimation. The geometry model used by the initiator is a flat plates model, based on the corner points coordinates of the tail surface segments, generated by the MMG and exported via FEM-table.

The condensation and sizing process is then iterated until weight convergence, which is generally achieved within 5-7 loops of 45 minutes each.

The overall process is then repeated for all the tail variants to be studied (e.g., with different span and/or sweep angle value). Hence, a new consistent set of segmented

surfaces, FEM-tables, condensation points and CAERO cards is generated by the MMG, while the DEE framework takes care of the automatic, coordinated execution of the entire process. No action is required by the user, who can, however, monitor the process while it proceeds across the network of Linux and Windows machines, where the various modules are installed.

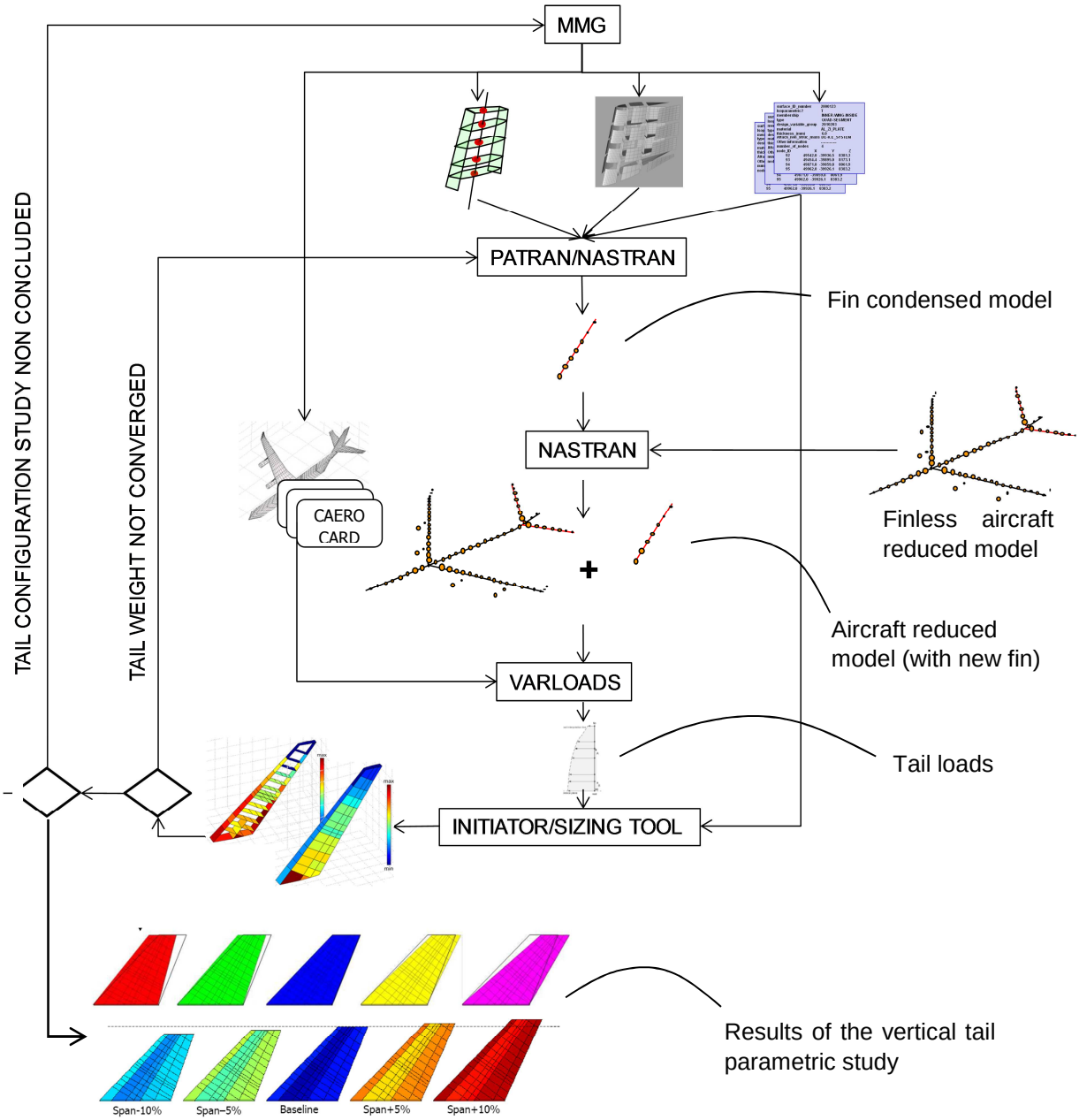


Fig. 6.24: schematic process of a vertical tail parametric study: the MMG feeds the tools for condensation, analysis and sizing with different but coherent models.

6.8 Multi-level modelling to manage complexity and support multi-level design

In the MOB project, the FE-based mass estimations of the BWB structure were corrected by a factor 1.5 (based on literature studies) to account for the lack of details, such as door and window cut-outs, in the FE model.

To increase the reliability of the weight estimation, as well as to promote a multi-level design approach (with preliminary and detail design phases accounted by one design system), a strategy was investigated to account for the effect of a door cut-out in the structural analysis process of the entire aircraft.

The challenge was to include in the computational design framework a module for the design and optimization of cut-outs (frames and doublers), without increasing complexity in the geometry preprocessing phase and in the set up of the (already) multi-level optimization problem.

The following approach was implemented concerning the surface segmentation:

- The Surface-splitter capability module performed the usual surface segmentation (section 6.3.1) on a BWB without door cut-out
- A scanning routine was applied to search and collect all the surface segments "perturbed" by the presence of the door cut out (i.e., those surface segments with a not null intersection with the boundary curves of the cut-out)
- A "disturbed by door-cut out" tag was added in the FEM tables
- The reduced set of perturbed surface segments was further processed, by trimming them as illustrated in Fig. 6.25
- The trimmed surface segments were exported in a separate group via IGES.

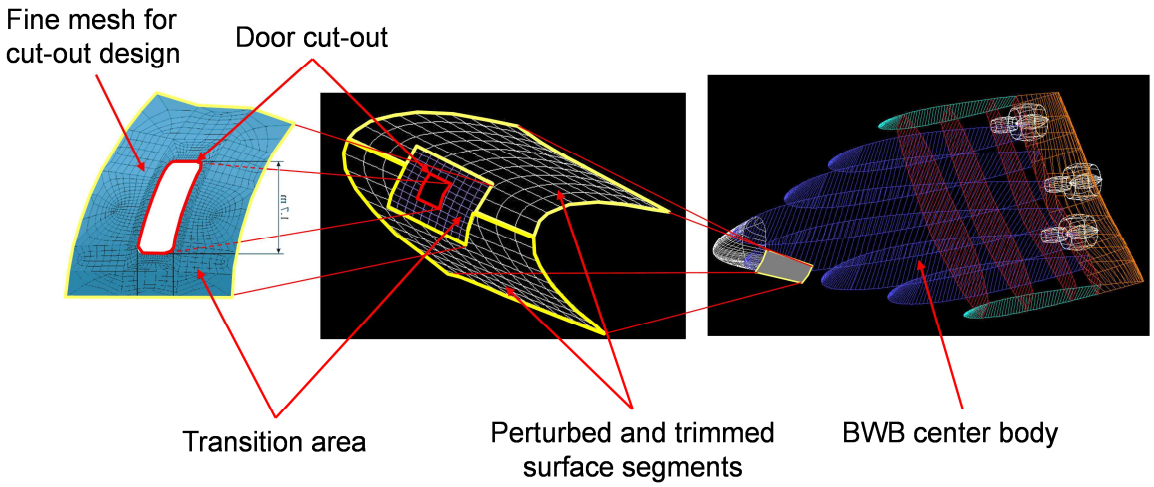


Fig. 6.25: extraction of the surface segments perturbed by the door cut out for detailed design and optimization of door frame and doublers.

The following approach was implemented concerning the FE-analysis loop:

- The whole BWB, without any cut-out, was meshed by PYCOCO using the usual coarse mesh and submitted to a first FE-analysis loop
- PYCOCO read the mesh points and the calculated displacements on the boundary of the surface segments perturbed by the door cut-out (detected via the FEM tables tag)
- The trimmed perturbed surfaces generated by the MMG and the boundary mesh points with relative displacements were fed to a NASTRAN-based application specifically developed to size and optimize the cut-out design (Engels et al., 2004).
- The cut-out design and optimization module produced an optimal cut-out structure and delivered it as a NASTRAN super element to be inserted in the global FE-model, in place of the perturbed surface segments

The results:

- The cutting and trimming process applied to the surfaces disturbed by the given detail did not affect the surface segmentation of the global aircraft model
- The shape of the disturbed elements (used to build the optimal cut-out) was consistent with the shape of the global model.
- The structural characteristics of the optimally designed detail could be fed back in the overall FE-analysis model

The idea of building separated-but-associative models, what we call *here multi-level modelling*, to support multi-level design was further developed in this research work to satisfy the request of using the MMG concept (and KBE technology in general) to address the design of the overall aircraft and its components, simultaneously.
Indeed, the generation of a single "can-do-everything" MMG was considered a too big challenge. Apart from the intrinsic difficulty of developing - and maintaining - a KBE application that is flexible and generic but, at the same time, able to address very specific design solutions, there are actual software limits that impose a maximum size for a KBE application.
The first application of the multi-level modelling approach has been implemented to support the design of wing and tail with movables. A dedicated report writer has been developed in the MMG to extract from the surface model of wings and tail empennages, a patch of the size of the movable (planform shape and location of the movable specified in the MMG input file). This patch is then used as outer mould line (OML) by another "slave" KBE application. In this case, PMM, the Parametric Movable Model developed by van der Laan (van der Laan, 2008) for the generation of detailed

movable models for structure and manufacturing analysis. Any time an updated OML is sent to the PMM, the geometry of the movable model adapts automatically, because of the associative definition with the OML. Together with the OML file, the MMG delivers the location of the hinges, compatible with the position of hinge ribs located in the wing or tail structure. Besides, it provides the aerodynamic models of the complete aircraft configuration (with deflected movables as described in section 5.2.12) to compute the aero loads required for the movable sizing (Nawijn et al., 2006; van Houten et al., 2005).
Van der Laan has further exploited the multi-level modelling approach, by developing also a slave KBE application for detailed modelling of movable ribs, and another one for the automatic generation of production moulds for thermoplastics ribs (van der Laan, 2008).

In the same line, Krakers has developed a KBE application for the detailed design and thermo-acoustic analysis of fuselage barrels (Krakers, 2009). This makes use of specifically developed HLPs and CMs, and requires as modelling basis the fuselage surface instantiated by the MMG.

The development of "master-slave" KBE model generators appears to be an effective strategy for managing modelling complexity. It brings the benefits of KBE technology across various scale levels (aircraft → component → subcomponent), and support a true multi-fidelity analysis approach, both in an affordable way. The associative relationships allow top level decisions to cascade down the model hierarchy and provide designers with early feedback. As such, more informed decisions can be made early in the design process thanks to data and information generally available only at a later stage of the design process.
Last but not least, different KBE developers can be in charge of maintaining and developing different KBE applications in parallel, thereby shortening the development cycle and lowering management risks.

CHAPTER 7
Knowledge Based Engineering. Opportunities and Methodology

1. Introduction
2. Implementation of KBE systems. Identifying the proper application cases
3. Implementation of KBE in the non-integrator company and SMEs
4. Organizational and human issues in the exploitation of KBE
5. Methodological development of KBE applications. The long term view
6. Trends and evolution of KBE technology
7. Recommendations & Expectations

7.1 Introduction

In the previous chapters, mainly the technical aspects of knowledge based engineering have been addressed. Chapter 3 has discussed the origins of this technology and the peculiarities of KBE systems (i.e., the *what* of KBE). Chapters 4-6 have demonstrated *why* KBE is worth the attention of the designers' community. The expected benefits can be summarized as follows:

- Enhancement of the *productivity* level: more product variants can be designed/analyzed in a shorter time because of enhanced level of design automation
- Improvement of products *quality* level: better and more mature design because of the enabled

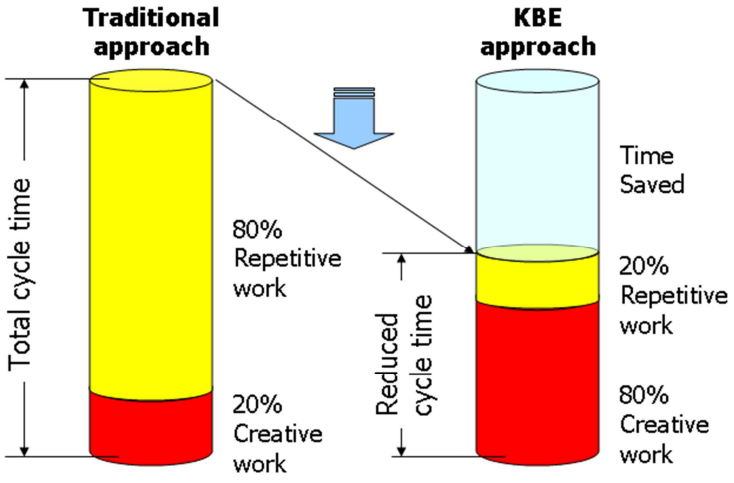


Fig. 7.1: the estimated impact of KBE on the design cycle time. From 80:20 repetitive/creative work time ratio to 20:80, plus net time saving.

multidisciplinary analysis and optimization approach.
- Improved confidence in generated designs, which is beneficial for making proper proposals to customers.
- Improved level of *workers' satisfaction* and more space to *innovation*: less repetitive work, more time for real creative design (Fig. 7.1).
- Capability to *capture*, *retain and re-use* company corporate knowledge

This chapter is meant to complement the discussion by further elaborating on KBE tools development opportunities and methodology, which we can address as the *When*, the *Who* and the *How* of KBE.

When KBE: However, KBE is not the silver bullet for all kind of engineering design cases. Considered the significant economic and human investments required to implement KBE inside a company, the ability to recognize the projects that are most suitable to a KBE approach is extremely important. Also for those companies interested in testing KBE, the selection of an appropriate pilot project is of outmost importance to avoid an unfair assessment of this new technology. Therefore, a list of typical situations is presented in this chapter, where KBE initiatives are likely to have the highest chance of success.
Who KBE: Though KBE had its start as an expensive technology typically oriented towards OEMs, Section 5.4 elaborates on the possible advantages non-integrator companies and SMEs can get from the implementation of KBE.
How KBE: Section 5.6 discusses the importance of a methodological approach to KBE application developments. The need to secure the investment made with the development of KBE application calls for a systematic development approach, which can guarantee the continued functioning and maintenance of the application. The roadmap for the development of a KBE application will be discussed and needs and characteristics of the various phases of the KBE lifecycle are illustrated.
..and then, what's next in KBE: The chapter concludes with a discussion of the evolution and trends of KBE technology and a list of recommendations and expectation for the new generation of KBE systems.

7.2 Implementation of KBE systems. Identifying the proper application cases

The development of a KBE application typically represents quite an important investment that organizations commit to. The costs of software licenses and training of those appointed as the future knowledge engineers and KBE developers in the company, need to be regained. It is of fundamental importance that the proper application is selected in order to make the KBE investment a success. The failure of a pilot project might represent not only a waste of money, but could also stop for

good any further KBE initiative and inhibit future profits from other KBE-appropriate business cases.
The question rises immediately: what kinds of projects are suitable for a winning implementation of knowledge based engineering? What are the indicators that identify the proper case for KBE application?
Although it is not possible to measure *the level of KBE potential* of a given design case, there are indeed a number of typical features, which strongly hint KBE opportunities.

The design case is highly rule-driven
Due to its rule-based nature, KBE fits well those applications that are *highly rule-driven*. On the contrary, applications that require high sketching freedom, style exercises, or deal with very fuzzy attributes and constraints are generally not good cases. The rational and structured approach at the base of any KBE product model development is just inadequate to support emotional design. However, KBE can play a role from the moment that the initial phase of esthetic focused design (where direct interaction through free-form techniques is the best option) is concluded and the design has to be engineered. For example, after the geometries sketched by the style department of an automotive company have been completed and the concept needs to be assessed in terms of aerodynamic performance and manufacturability/formability, then KBE can play a role.

The design process is well understood and consolidated.
In this case the design rules are evident, easy to codify or already codified, and possibly stable. The whole KBE technology relies on the fact that *rules are known or available* somewhere, such that they can be implemented in the product model. As thoroughly discussed in Chapter 3, knowledge is not always available in the *explicit format* of a rule. Sometime, significant efforts can be required to elicit knowledge from the head of experts and give it an explicit structure, such that "it can be written down" using a programming language. The success of a KBE application, or even the kick-off of its development process, strongly depends on the work of knowledge engineers and their knowledge acquisition ability (Shreiber et al., 2000; Stokes, 2001). Indeed, part of the knowledge used in practice is often related to designer's intuition, past experiences and heuristics, and generally difficult to be translated in rigorous rules. However, the KBE approach offers the possibility to make use of *rules-of-thumb,* which represent a very useful means to tolerate a certain extent of fuzziness, yet building effective KBE applications.

A relative *stability of the rules* is also favorable to the development and longevity of a KBE application. As discussed in Chapter 4, changing or updating rules in a KBE application is not the same as in conventional rule-based systems. In case of a KBE application, it might require changes in the main structure of the product model,

hence expensive re-writing of many lines of code. Of course, the Object Oriented paradigm at the base of the KBE language, together with an adequate programming style (i.e., modular, generic) can largely mitigate such risk and offer the opportunity to adapt and reconfigure the structure of the product model.

The design case is multidisciplinary.
As previously discussed in Chapter 3, the generative design capability of KBE can provide an enormous support to multidisciplinary analysis. Different discipline-specific views from the same product can be generated fully automatically and fed to a range of in house and COTS analysis and design tools. The mix of geometry handling and problem solving characteristics required in the preprocessing activity of analysis models are generally a good target for KBE. More evidence will be provided in the next chapters.

The design process is highly *repetitive*.
Design processes where the same rules are continuously re-applied and evaluated are typically very well suitable to KBE solutions. For example the generation of the geometry model of large quantities of parts/products, which are *all different, but in the end just small variations of the same thing* (e.g. ribs in a wing structure (Rondeau et al., 1996)).

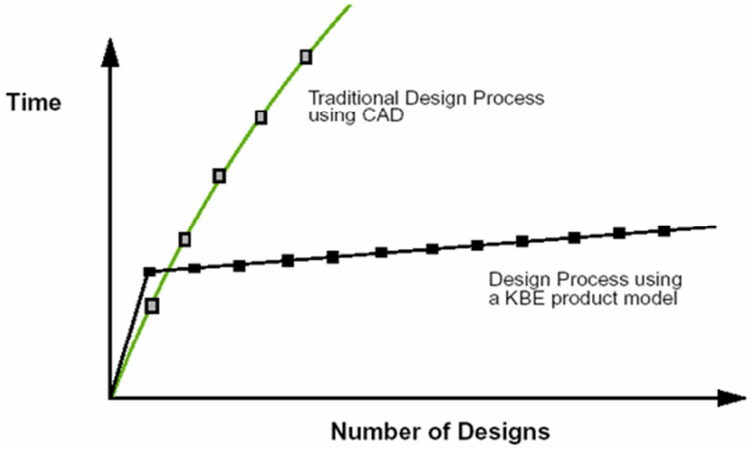


Fig. 7.2: Qualitative benchmark of KBE and CAD, indicating the convenience of KBE for applications requiring the generation of many design variants.

In this case the number of iterations is determining whenever KBE is a better solution than CAD. In fact, the generation of a product model might take more time than required to generate one or a few CAD models. However, once the product model is in place, all the variants are generated automatically "for free", while the CAD operator takes almost the same amount of time for every iteration[50]. Fig. 7.2 from (Knowledge

[50] Indeed the estimation of the break-even point is the most relevant information for the business case. Unfortunately, a methodology to estimate the time required for the development of a KBE application does not exist in literature. KBE-user companies have their own legacy information, which is based on the performance of past KBE development cases.

Technologies International) shows a qualitative benchmark of KBE and CAD technologies.
Though the automatic generation of technical drawings has been one of the first and most successful applications of KBE, *repetitiveness* can be found also somewhere else in the design process, *earlier than in the detail phase*. For example, in the field of optimization, variants of the same product have to be generated at each loop. MDO, where the features of repetitive/iterative design and multidisciplinary design come together, is definitely another good candidate for KBE solutions. It is not by chance that the latest large European projects on MDO have all included, though in different extent, the use of KBE (Allwright, 1996; Morris et al., 2004; de Weck et al., 2007). In the next chapters, large evidence will be provided of a possible use of KBE to exploit repetitiveness as from the conceptual phase of design.

The design process features a mix of problem solving, geometry manipulation and data processing
As far as some of the hints addressed above are present (i.e., rule-driven, repetitive design, etc.), the coincident occurrence of geometry manipulation, data processing and problem solving features is a sign that KBE is probably *the most suitable* technology at hand, not just a good candidate. In this case, the use of KBE can provide a competitive margin with respect to any other design approach based on CAD, conventional knowledge based systems, or the use of some general purpose programming language.

7.2.1 Backfield or first line? A note on the strategic deployment of KBE

The occurrence of one or more of the features described in the previous section gives information about the eligibility of KBE for the design case at hand. However, how can a company decide about the strategic deployment of KBE? Should KBE be used in the backfield to consolidate and record the design knowledge of the company, or should it be used in the first line to address urgent and critical design cases? Should KBE be used as mainstream technology or just deployed for some design emergency? Again there is not one straight answer, as expected, these choices being a matter of company specific strategy.
Considering the risk associated to a new technology, that requires highly educated people, licenses and training expenses and possibly some organizational change (see later in Section 7.5.4), some companies might opt for a KBE deployment in offline activities first. However, rising awareness of such technology across the whole company is very slow and the payback of the investment in training people and buying licenses is lengthened. Problems of justifying the investment might rise as

well. Another philosophy used by the KBE department at Airbus UK[51] is to select the program within the company with the biggest troubles to deliver in time and just go for it. This will give immediate visibility of the benefit of KBE. Cooper in ref (Cooper and Smith, 2005), confirms that, so far, in the industry world, KBE seems to be accepted just and only when it solves difficult technical problems or gets a desperately needed answer fast: KBE as *the ultimate weapon* for the first line. However, for different companies than large integrators like Airbus, a different strategy can be implemented with success, as discussed in section below.

7.3 Implementation of KBE in the non-integrator company and SMEs

There are situations that allow also for a profitable *offline* KBE development; they especially arise in those companies, like suppliers and sub-contractors, whose products present a large degree of commonality. They have the possibility to analyze their *different-but-similar* products, investigate for a unified design knowledge assessment and build up the KBE application that captures and automates it. This concept, imported from the management world, is addressed as *family thinking*.
For instance an aircraft components supplier, with years of experience in designing and manufacturing for different customers parts such as tail sections, fins, rudders and movables surfaces in general, should be in the condition to develop a KBE application that captures in one generic product model all the many peculiarities of the components and parts produced in the previous years, for the different customers. The design results of the past assignments will automatically supply the validation of the KBE tool; at the same time such application would provide the company with a powerful and effective tool to generate and discuss clear-cut proposals for customers, with the confidence of knowing in advance the design factors that will affect weight, price and delivery time. The company's knowledge gained to develop a product for previous customers will be available to propose products to new ones.

It should be considered that such kind of company (the product family supplier) will benefit of the KBE approach maybe more than an integrator company. Suppliers are often put in competition with others by integrator companies to deliver a competitive proposal within a very tight time frame. The amount of data they receive from the integrator to set up the proposal is generally rather limited, and even after the work has been commissioned, the information flow remains slow. The use of some pre-developed KBE application could put suppliers in condition to generate autonomously, in-house the amount of data and information sufficient to prepare a

[51] A conversation with S. Allwright and A. Murton from the KBE group at Airbus UK, Filton.

competitive proposal. Once the work has been commissioned, the KBE application used to generate the data for the proposal, could be tuned with the more detailed information provided by the customer. Possible, design decisions and constraints may be changed and re-assessed by the contractor without affecting the final delivery time, because the dynamic definition of the KBE product and its generative capability can allow large reconfiguration and assessment of the design at any moment.

In reference (Lovett et al., 2000) the issue of implementing KBE technology in small and medium enterprises (SME) is discussed. The volume of work within this kind of companies might seem not big enough to justify the large investments to acquire KBE platform licenses and/or employ or train people to develop KBE applications. Often there is a lack of people with IT experience and the budgets for R&D are in general limited, if available at all, in which case it might be convenient to rely on external consultants to develop and set up KBE applications. It should be considered that, in comparison to a large OEM, a reduced external workforce would be required, in proportion with the smaller size of the application an SME might need. On the other hand, the usefulness of a KBE application to keep and secure SME's knowledge might be even higher than for large companies. In fact, in hard times, the reduction of personnel for SME is more risky than for bigger companies, because the knowledge is concentrated in less people.

7.4 Organizational and human issues in the exploitation of KBE

Being KBE a new technology to many companies and requiring quite a different vision to business, it might have hard times getting acceptance. A technically good developed KBE application will anyhow require high level and on-going support to promote its use. The success of KBE, apart from the selection of the right application, will depend on a number of organizational and human issues that any company will have to take into consideration. Some are just discussed below.

The cost of a continuous investment

The development of a KBE application requires high-educated people, with vision, and analytical and abstraction capability. They need training and technical support to integrate their applications within the IT environment of the company and of course time to get proficient with the KBE platform in use. This of course represents a big investment for the company: training cost, software licenses and dedicated people to develop applications. The training costs represent a continuous investment: since people move within the organization, they will not be always in charge of maintaining and further develop their application. New trained people will be required to keep the activity running with continuity. On the other hand, it should be considered that the effectiveness of KBE applications will generally require less people in a project.

Education for confidence

Companies, which have deployed KBE systems successfully, have recognized the need to have a workforce educated on the topics of knowledge management and KBE and their potential benefits. Without such an education, a negative or suspicious attitude towards such technology can arise: employees may feel that "giving away" their knowledge will make them vulnerable to redundancy, or alternatively they may resent a perceived role of "*being told what to do*" by the KBE system (Sainter et al., 2000b). Some might have a black-art perception of KBE applications (Object Management Group, 2005) and use them with limited trust, not knowing "what is inside that box".

In-house vs. outsourced KBE development

There is often a debate whether to employ people external to the company to develop KBE applications, e.g., consultants from the KBE system vendor. In some cases, this might turn into a short term solution to eliminate the training costs required to form internal developers, and to shorten the time required to get an application operative. However, the ideal KBE developer is the one who is both proficient with KBE technology and familiar with the particular application field: the engineering expert and the KBE developer in one person. The latter situation would be extremely beneficial in terms of giving confidence to the company end-users about the effectiveness and real value of the given KBE application, which is not seen as a "black box" developed by externals ("…*who don't know how things actually work…*") and superimposed by management. Possibly, the combination of internal dedicated personnel and external KBE developers might give the best results. Sounder KBE applications can be generated faster and, at the same time, in-house KBE expertise is increased (Lovett et al., 2000).

Food for knowledge workers

Developing KBE applications generally requires a *multidisciplinary vision on the design problem*. At the same time, it is proven that developing KBE applications just augment such vision in developers. The required ability to synchronize knowledge from different disciplines and to make explicit in rules what in ref. (Whitney et al., 1999) is called the *interaction knowledge* (i.e., how the different involved resources and generated results interconnect and logically link to each other to complete a design assignment) is such that the KBE developer, in the end learns much more than the application end-user. The "down side" is that good KBE developers are often promoted to higher, more managerial positions, hence creating the need to train someone else to take over his/her KBE development tasks.

Mental pigeonholing

Once technical and non-technical barriers to KBE have finally been demolished and a good KBE application has been developed and operative, a side effect of mental

pigeonholing might arise. As Cooper, Fan and LI comment in ref. (Cooper et al., 2001), people have the tendency to think KBE is just good for that very application where the KBE effectiveness was proven. Depending on the various localized KBE experiences, people can claim that "…*KBE is just good for structural optimisation…*", or "…*KBE works for system configuration…*" etc. Time and multiple applications are required to consolidate a wider confidence.

7.5 Methodological development of KBE applications. The long term view

KBE applications should not be written *ad hoc*. Ad hoc solutions generally restrict the success of KBE applications to the very short term, while hazarding the potential long-term benefit (Sainter et al., 2000b). It is very inefficient to use an expensive and complex KBE system to write an application that is only able to address one very specific instance of a problem, and contains rules that applies only in one very specific case. The first consequence of such a faulty approach is the very limited flexibility and durability of the KBE application. As soon as similar but not identical design cases come at hand, that KBE application will result inadequate or very limited in use.

Some other time, KBE applications might disorderly grow under consecutive layers of advancement, brought by separate developers, with different scopes. At a certain point, they might have become so unwieldy and complicated, that new developers will not be able to understand their structure and further develop/maintain them.

In the one case or the other, it will result in the extensive re-coding of the application, or an extra production of patches and fixes, which will make the application even more inflexible, difficult to use and also very inefficient.

Eventually, there is an evident risk to lose knowledge or hamper its reuse, which paradoxically were the two reasons to implement KBE technology! Therefore, the definition of a *methodology* to guide, structure and support the development activities of KBE applications becomes an important factor for a long term success.

7.5.1 The need for a methodology. The MOKA project contribution

A KBE dedicated methodology consists of the definition of guidelines, standard procedures and techniques to help KBE developers generating effective applications, which are also reusable and maintainable.

There is a tendency to think that the time required to (define and) apply a methodology might be better invested to speed up the development of the application at hand (Sainter, Oldham and Larkin, 2000a). However, it has demonstrated that the adoption of a proper methodology is not just beneficial for the mid-long term, but it yields also a development time reduction of 20-25% (Oldham

et al., 1998). Indeed prototyping has an unquestionable value and should not be hampered. However, it is difficult to imagine that a sound KBE application created to last and evolve can start by switching on the editor and beginning with free-flow coding.

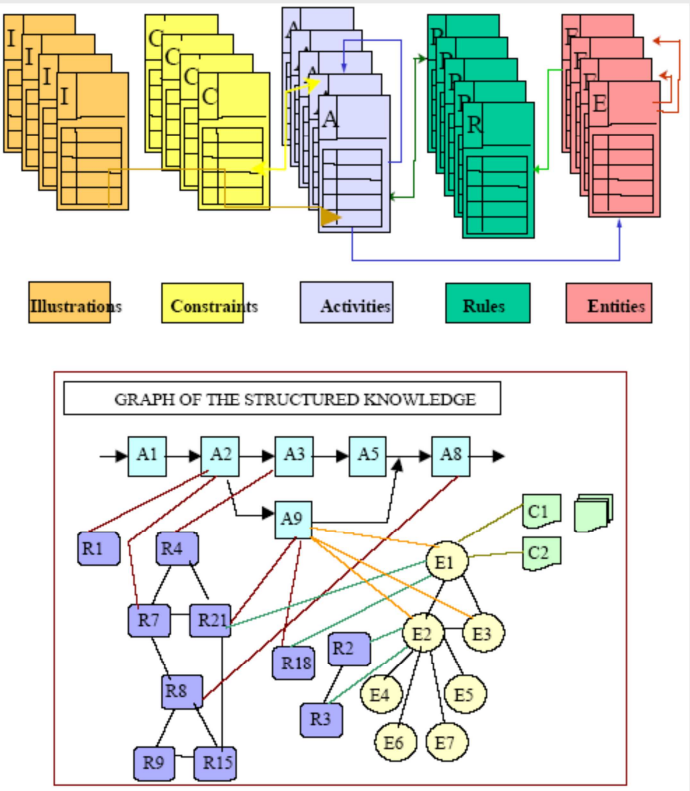
Though some general methodologies, such as CommonKADS (Shreiber et al., 2000), KIF and DEKLARE (Stokes, 2001) have been developed with the scope of supporting the development of knowledge based systems (see Chapter 4), the MOKA methodology (Methodology and tools Oriented to Knowledge-based engineering Applications, 1998-2000) (Stokes, 2001; Brimble and Sellini, 2000) is the only methodology available at date, which specifically addresses the development of knowledge based engineering applications. The main results of the MOKA project can be summarized as follow:

- A consistent way of capturing and representing product and process knowledge (see insert next page), supported by a graphical modeling language called MML (MOKA Modeling Language), based on the UML.
- A software tool to assist in the capture, representation and maintenance of this knowledge.
- Identification of a typical KBE *life cycle*. The main phases in this life cycle are analyzed and related needs specified.
- Preliminary investigation on the possibility to automatically generate KBE code from this software tool.

The first point is possibly the most valuable contribution of MOKA. The relevance of knowledge modeling and of visual modeling techniques in particular is further elaborated in Section 7.5.2 (see also insert next page).
Concerning the development of the software tool to assist in the capture, representation and maintenance of knowledge MOKA produced a prototype system, never made available to the public. However, the PCPACK tool marketed by Epistemics (Epistemics) can be actually used as a kind of MOKA tool[52] (Milton, 2008).
The identification of the KBE life cycle was also a relevant contribution of MOKA and can still be used as a kind of roadmap for the implementation of any KBE initiative in a company. This is elaborated in Section 7.5.3.
Concerning the ability to automate the generation of KBE code, actually, only some prototype concept was elaborated in MOKA. The XML language was used to map the MML product model into the backbone of an ICAD application. The automatic mapping from a knowledge representation tool to a KBE system is still an open issue of high interest. More in Section 7.7.6.

[52] PCPACK is not the MOKA tool, but it can be used to satisfy the requirements for a supporting software tool for the MOKA methodology. It supports the capture, analysis, modeling and publishing of design knowledge using a MOKA framework (ontology)

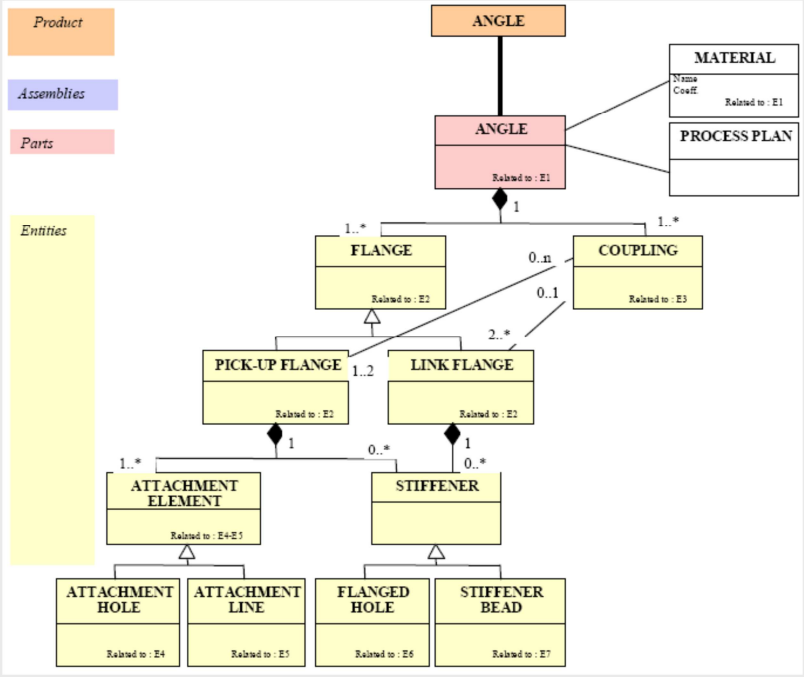
Engineering knowledge representation in MOKA



The MOKA methodology (Stokes, 2001) provides a two-level modeling approach to capture and formalize declarative and procedural engineering knowledge, which consists of the so-called *informal* and the *formal* model.

The informal model makes use of the five ICARE templates (one for each of the 5 knowledge objects: Illustrations, Constraints, Activities, Rules and Entities) to capture the expert's knowledge and give it a first structured representation, in a way that is very familiar to the experts' (to ease validation by the experts themselves). See figure on the left side.

The second level of modeling supports the transition of the raw knowledge from the ICARE forms to a more formal representation, based on the Object-Oriented modeling paradigm. A dedicated visual modeling language called MML (MOKA Modeling Language) has been developed, which is actually an extension (yet extendible) of the industry standard UML (Unified Modelling Language). Knowledge modeled in the MML is well understandable by software developers and largely reduces the gap between the experts' informal language and computer programming languages like Common LISP, for example. See the MML representation of a product model in the figure on the right.



7.5.2 The importance of knowledge representation and visual modeling techniques

Any suitable methodology to support the development of KBE applications has to address the Knowledge Engineering and Knowledge Management aspects related to KBE. Hence it has to provide a valid knowledge representation/modeling system. The use of a standard knowledge representation eases the level of understanding between domain experts and KBE developers, as well as between different developers. It facilitates the validation of the knowledge by the experts, so that KBE developers can proceed coding it in the KBE application. It also reduces the difficulties and the time needed to recall the design rational of a given KBE application, which can be easily lost among thousands lines of code. A reduction in the lead-time is the first consequence, but knowledge re-use in general may also increase, because well-structured knowledge is easier to be managed and shared, both at human and computer systems level.

The use of *visual modeling techniques* represents a main element in the knowledge representation methodology. Visual models, graphs and diagrams are known to be an extremely efficient and effective means of communication. Indeed they represent a high-level language, where the combination of simple shapes can be used to carry extremely large amounts of information.

Visual modeling languages such as the UML (Shmuller, 2004a) and the MML (MOKA Modelling Language Core Definition, 2000), developed in the MOKA project as an extension of the UML, are specific for engineering knowledge representation and their use can be extremely beneficial not only to communicate, but also to preserve knowledge from loss. Several examples of UML diagrams have already been shown in the previous chapters and many more are used in the coming chapters to describe the details of a developed KBE application. Practically, these diagrams can be considered as a kind of back-up, or blueprint of the given KBE application. We might regard them as *neutral representations* of the engineering knowledge, independent from any specific KBE environment. Whenever the commercialization of KBE platform ceases or is no longer available within the organization, the loss of knowledge stays limited.

Commercial software tools, such as the abovementioned PCPACK, support KBE developers in the application of a methodology, providing a smart software environment to build multiple-but-interconnected representations of knowledge, and generate UML(-like) diagrammatic models. See an example of MML diagram in the previous page insert.

7.5.3 The KBE lifecycle

One of the outstanding results of the MOKA project was the definition of the so-called KBE lifecycle, which is the detailed roadmap to the development of a KBE

application, from the identification of an opportunity for KBE development to the final deployment of the application in the company (Oldham et al., 1998). As shown in Fig. 7.3 four main phases can be identified, namely:

- Problem/Opportunity Identification,
- Knowledge capture and formalization
- Packaging
- Deployment.

The feedback and consequences from the deployment of the KBE application might identify new development opportunities or call for the maintenance of the existing KBE system, hence triggering a new cycle.

As illustrated in the more detailed process flow of Fig. 7.4, each of the 4 main lifecycle phases can be further detailed in more specific activities. Different actors, tools and needs characterize the four phases as elaborated in the next subsections.

Problem or Opportunity identification: In this preliminary phase, the opportunities to develop a new KBE application or to modify an existing one are identified. Consequently, all the required resources to pursue the initiative are identified to build the actual project plan. These include: the possible knowledge sources (e.g., domain experts, data-bases and manuals), the appropriate knowledge acquisition tools and techniques, and the suitable analysis tools and techniques for knowledge representation. If the expected benefit and technical feasibility of the new/modified KBE application can justify the development case, then management can give approval to proceed with the next lifecycle phase.

Knowledge capture and formalization: Two main activities can be distinguished in this phase, namely *capture* and *formalize*. In the first activity the relevant knowledge is elicited from the experts and other sources (See (Milton, 2007; Shreiber et al., 2000) for detailed information concerning knowledge acquisition techniques) and then structured through *informal* diagrammatic representation. The informality of this representation is required to allow the domain experts to read back and validate the knowledge that has been captured (i.e. to make sure that no misunderstanding occurred during the elicitation phase and prevent incompleteness). The knowledge *capture* activity includes the removal of the typical vagueness

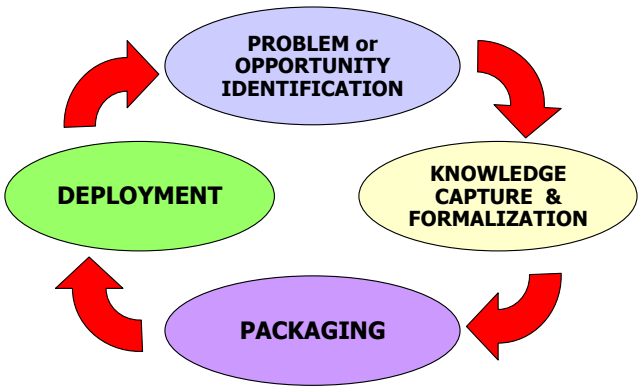


Fig. 7.3: the main phases of the KBE lifecycle.

of the spoken language, to prepare the knowledge to be formalized (i.e. to be translated in rules). The *formalize* activity analyses the informally structured knowledge, as obtained from the *capture* activity, and represents it in a *consistent*, *formal*, *neutral format*, to enable it to be assessed for correctness and suitability for (re-)use. This activity might eventually require other iterations of the *capture* activity until all the inconsistencies are solved. The integration of the *capture and formalize* activity reduces the gap from the raw knowledge, as elicited form the expert, to the formality level typical of a programming language. Still no coding on the KBE platform has actually started in this phase. The contribution of MOKA mainly addresses this lifecycle phase. See in the previous insert the two-level modeling approach to capture and formalize knowledge, consisting of the so-called informal and the formal models. The use of visual modeling tools and languages like PCPACK and the UML (or the MOKA's UML extension MML) find in this phase their main application field.

Packaging: In this phase the knowledge assessed and formally structured in the *previous* phase is finally coded into the selected KBE environment (e.g. ICAD, GDL, etc). This is the real programming phase in the KBE lifecycle. The skills of the KBE developer, as well as the capability of the selected KBE system and its programming language are the most important factors. The coding should proceed under the supervision and approval of the expert/end-user, which should help in testing and validating the efficiency, effectiveness and robustness of the different modules as they are developed. The packaging phase actually sets the important transition *from the neutral*, KBE system-independent knowledge representation, *to the specific* format dictated by the language of the selected KBE system. This knowledge transition is generally carried out by hand, with evident limits in time efficiency and "translation accuracy". Indeed the generation of dedicated *translators* to automate the code generation on the basis of a neutral knowledge representation, represents a very interesting topic. MOKA demonstrated the feasibility of automatic generation of the backbone of an ICAD application directly from the MML formal model (See (KBE coupling illustration, 2000) and Chapter 14 of the MOKA book (Stokes, 2001)). However, the test case was of very limited extent (no process knowledge included) and indeed the "non generic" aspect of the whole methodology stops exactly when the specificity of the selected KBE system comes in.

Deploy: in this phase the application is distributed to all the potential users in the organization who could benefit from it. The KBE tool is installed, documentation is provided and, eventually, end-users are trained. In this phase of the KBE lifecycle, the support of IT is essential for providing a proper infrastructure to store the KBE application and make it available to the users (Sainter et al., 2000a). This might consist of a network of computers connected to a central repository where the KBE

application is stored. An adequate management policy of the different versions and updates of KBE applications is indeed required (including for example the implementation of a files *revision control system*) to allow the development and debugging of the application, while a working version is still available to end-users. These kinds of IT organizational issues, in the mid-long term, always reward the efforts.
After the KBE application has been deployed to support the relevant engineering activity, new opportunities to improve or extend the application might become evident. In this case, a whole new cycle of activities can start.

7.5.4 The KBE development team

The development of KBE systems is first of all about the people involved in the process and their specific roles and relationships. The picture presented in Fig. 7.5 shows the main actors involved in the process and the way they interact with each other. It should be acknowledged that such a schema is based on the simpler version provided by the CommonKADS methodology (Shreiber et al., 2000) to illustrate the team and the organizational aspects related to the development of a generic knowledge based system (not specifically a KBE system). It is clear how, from the organizational point of view, the development of KBE and KB systems does not present substantial difference. This is the KBE development team:

- *The domain expert(s):* he/she owns and provides the discipline knowledge. He/she must validate the knowledge implemented in the KBE system to be sure it is complete, consistent and correct such that it can be exploited by the KBE application end-user.
- *The knowledge engineer(s):* he/she elicits the knowledge from the expert and the requirements from the KBE application end-user. The knowledge engineer also structures the captured knowledge and delivers the proper representation models to the KBE application developer.
- *The KBE developer(s):* he/she is the expert in working with the given KBE platform and writes the code of the KBE application, based on the knowledge models provided by the Knowledge engineer.
- *The project manager:* he/she is responsible for the management of the knowledge engineers and system developers.
- *The software supplier(s):* they deliver the KBE platforms used by the developers and the knowledge management tools used by the knowledge engineers. Though external to the actual KBE team, they are often involved as consultants and (hopefully) collect users' feedback to improve the tools they market.

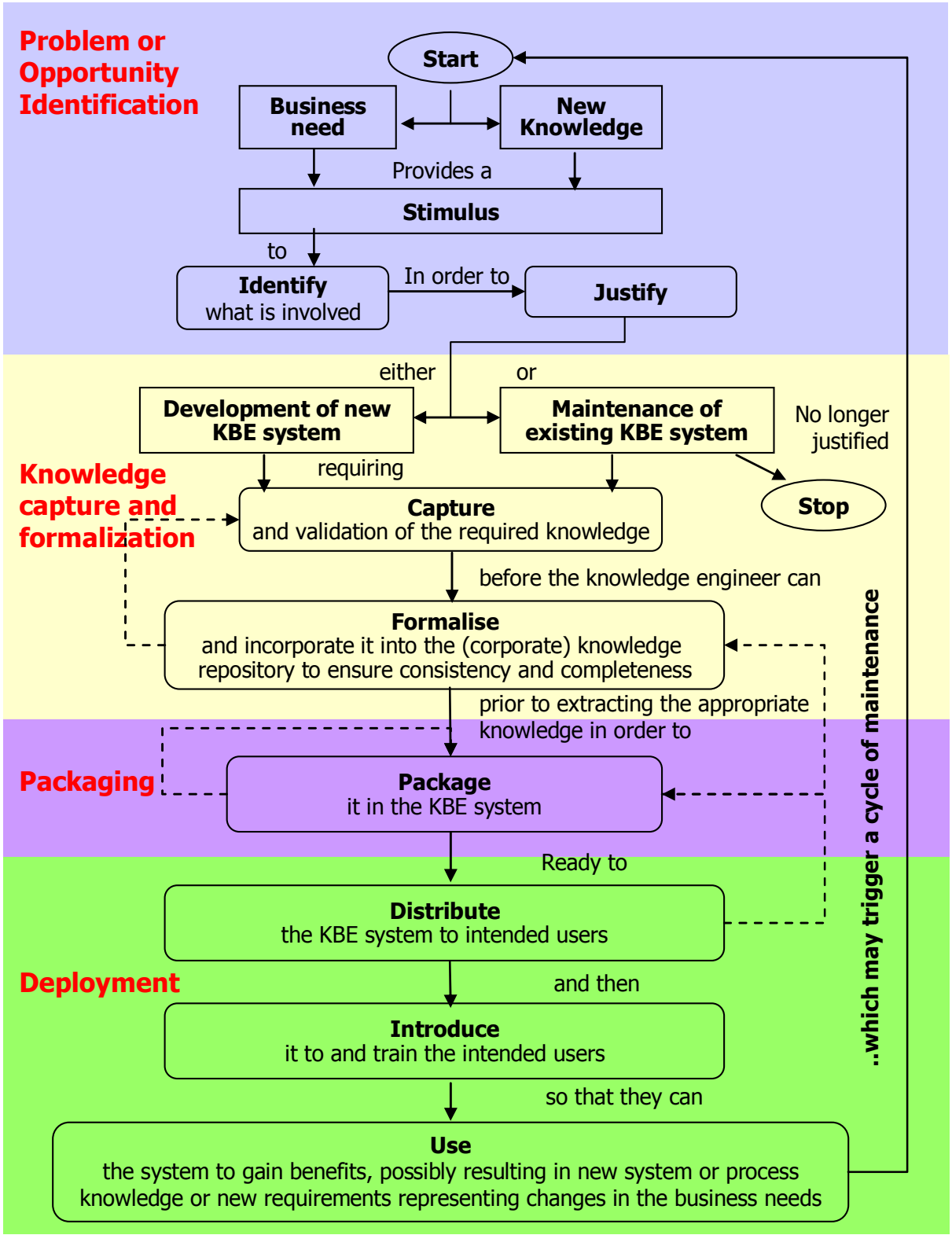


Fig. 7.4: Detailed model of the KBE lifecycle. Based on (Oldham et al., 1998).

Depending on the size of the organization and KBE application to be developed, the level of expertise and capability to write code, the same person can actually "wear more than one cap". It is not uncommon for the discipline expert to code the KBE application him/herself. Eventually he/she can be also the end-user or the person that mainly benefits from the developed KBE application.
The stimulus and the continuous support to the KBE initiative should come from a rooted knowledge management culture in the organization. *The Knowledge manager* is the responsible appointed to define a global knowledge strategy, initiate and develop knowledge related projects and facilitate the knowledge distribution within the organization.

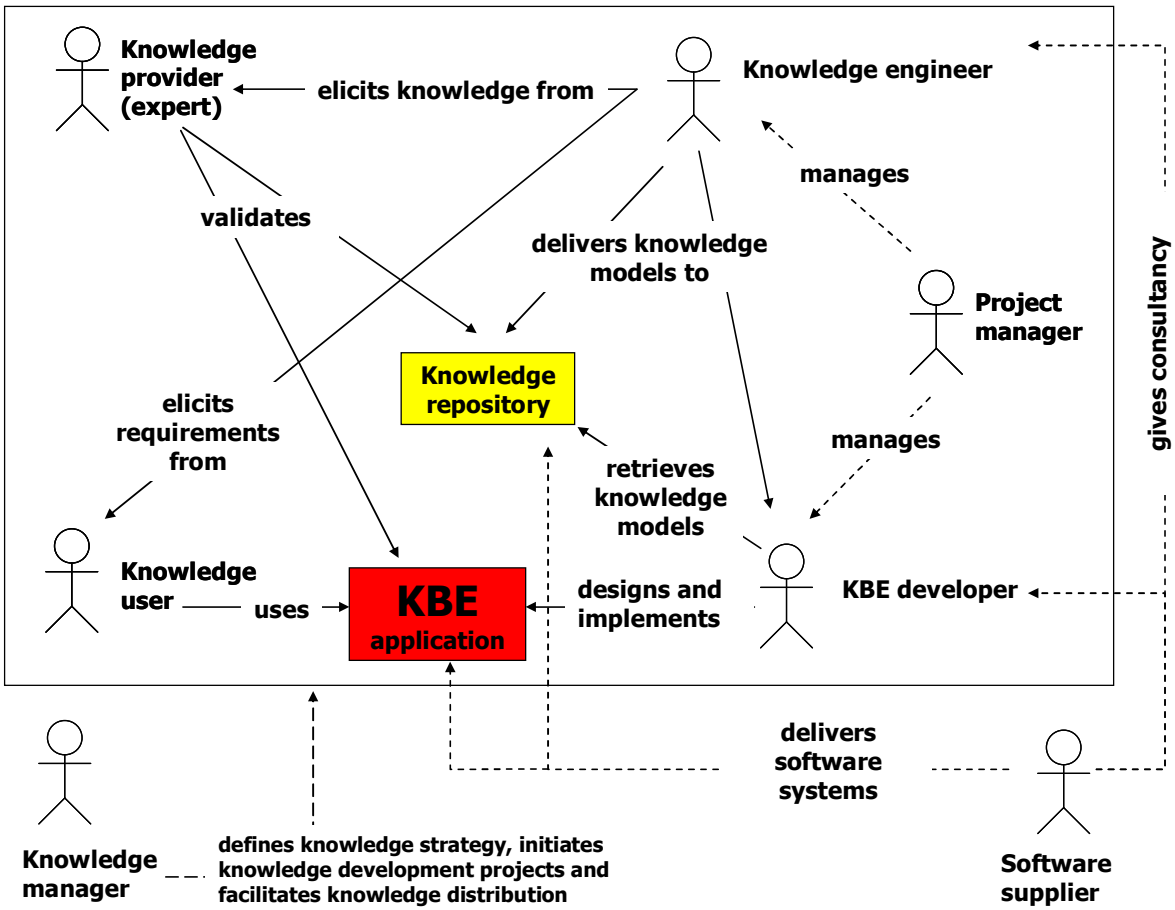


Fig. 7.5: The KBE development team: actors, roles and relationships

7.6 Trends and evolution of KBE technology

The first commercial KBE system arrived on the market in the 1980s, however only the last 10-15 years KBE technology has started to be used seriously. Notwithstanding its huge potential, KBE has not penetrated the market like CAD systems have done in their first 15 years. The reason for this limited success can be attributed to a combination of causes:

- *High costs of software licenses and needed hardware*. It should be considered that the cost of a full development license of one of the first generation KBE systems was around 100000$ per year. The same amount of money was in the beginning required to acquire adequate machines to run such systems (Lisp Machines). It is only since the early/mid 1990s that Unix workstations came on the market for a cost one order of magnitude lower (still not negligible!) Only the costs have been the main reason to limit the range of possible KBE customers to the largest companies from the aerospace and automotive sectors.
- *Arguable marketing approach by KBE vendors.* As Cooper elaborates in (Cooper and Smith, 2005), the business model of the first KBE-vendors was an impediment to the diffusion of just their own technologies. Due to the large commitment required to customers to adopt such complex and expensive technology, the KBE vendors felt the need to play multiple roles. They were at the same time KBE systems suppliers, consultancy providers (to help company to implement and operate effectively the KBE system they just sold them), as well as sellers of KBE end-user applications. As result, a third party company who wanted to use the KBE tool to offer consultancy service to some large KBE user, was becoming at the same time customer and competitor of the KBE system vendor. Moreover, the vendor was in many cases even in competition with its own end-user customer.
- *General lack of literature, study cases and metric*. Indeed, the scarce amount of KBE dedicated literature, the difficulty to find useful information about KBE success stories, as well as the lack of a good metric to estimate and judge the possible advantages of using KBE compared to traditional methods have all played a role in limiting the market penetration of KBE. Since the success or even the launch of a KBE initiative depends on the level of confidence and commitment of management, these issues absolutely cannot be disregarded. Disinformation and little understanding of the subject have often generated large misunderstandings on the actual capability and possible roles of KBE in the development process of products. The long endurance dispute KBE vs CAD is just an example.

If we add to the list also the *issues related to human acceptance* of KBE (already discussed in section 7.4) we might end up looking at the past successful

implementations of KBE just as "lucky" exploits by some elite broadminded organizations.

During the years, a number of technical developments and strategy changes have created the situation for a sustainable growth of KBE in the world of industry and research. First of all the cost of hardware has decreased dramatically: what yesterday required an extremely expensive Lisp Machine and then a workstation, today runs on any laptop, at a speed tenfold higher[53]. Many KBE vendors have actually adapted their system to the most widespread operating systems and computer architecture, i.e., simple desktop computers running Windows. However, only few current KBE systems can work with Linux and Mac.

During the years, the growing complexity of the products to be engineered has nurtured the development of continuously more complex computer aided engineering tools. This trend has somehow played in favor of KBE, because the relative complexity gap with KBE tools has automatically decreased.

Another important change which is going to strongly affect the future of KBE is the fact that leader companies in the development of PLM solutions have finally recognized the value of the KBE approach and finally endowed their top-end CAD products with KBE capabilities:

- In 1999, PTC introduced the Behavioral Modelling toolkit for Pro/ENGINEER 2000i, which allows methods to capture rules to steer the CAD engine
- In 2001, UGS acquired the KBE language Intent! from Heide Corporation and embedded it into Unigraphics to form Knowledge Fusion (In 2007 UGS has been bought by Siemens PLM software)
- In 2002, Dassault Systemes acquired KTI and their product ICAD. DS sinks ICAD and exploits KTI expertise to develop KnowledgeWare, the KBE add-on of CATIA V
- In 2005 Autodesk acquired Engineering Intent Corporation and integrated their KBE system with Autodesk Inventor, to form AutodeskIntent (now Inventor Automation Professional)
- In 2007, Bentley acquires the company Design Power and integrates their KBE system Design++ with Microstation

Without entering into the details of the different approaches pursued by these companies or questioning the real KBE effectiveness of their systems, some consequences of this trend are unquestionably positive:

[53] The development of the MMG, the KBE application described in Chapter 6-8, started on a 8000 Euro SUN workstation and 2 years later continued on a much faster, 800 Euro desktop. Unfortunately the licenses cost did not follow the same trend.

- KBE is entering into the mainstream of the other traditional product development software tools (though the "KBE side" of the abovementioned CAD systems still represent quite an exoteric and unfamiliar aspect for most of the typical CAD users).
- Finally there is the availability of a huge force in terms of technical knowledge, organizational, support and marketing capabilities, which can surely help disseminating the KBE technology to an unexpected large amount of customers of all size, from large integrators to SMEs.
- Inevitably, the entry cost of KBE, including hardware, licenses and training will decrease at a level that any company will have the possibility to evaluate the impact KBE can have on their business.
- Eventually, freelance consultants and independent professionals will have the possibility to own and operate a single license on their laptop.

At the beginning of the 1980s, KBE came out as a reaction to the limitations of current CAD systems. After a period of glory for exclusive customers, followed by few years of impasse, eventually CAD and KBE get along again… or is this a too positive claim?

7.7 Recommendations & Expectations

It is of extreme importance that the development of new KBE tools is based on the lessons learnt and experience gained by the pioneers and experienced practioners of this technology. The actual level of capability and maturity of available KBE systems is such that the concept does not need to be reinvented, but rather consolidated and, possibly, enriched with new capabilities to lower the accessibility level and to help programming better KBE applications in less time.

7.7.1 Consolidating the fundamental technical strengths

KBE platforms providers should keep their focus on two fundamental technical goals:

1. The enhancement of the level of robustness of geometry manipulation and interfacing with CAD kernel(s)
2. The enhancement of the computational capability and system stability

As a matter of fact, the integration of rule-based design, geometry manipulation and computation capability represents the real added value of KBE with respect to conventional CAD and Expert Systems, as well as to other general purpose programming languages. The future of KBE passes here. Compared to the two targets above, any other technical improvement is only a "nice to have".

7.7.2 Lowering the accessibility level

The use of a programming language to "instruct" a KBE system must be preserved and consolidated, being the programming approach the most powerful and comprehensive way to control and access all the functionalities of a KBE system. The almost unlimited flexibility offered by a high level, object oriented programming language offers designers an extremely valuable means for capturing engineering knowledge and making it reusable with consistency. On the other hand, the accessibility level to KBE technology has to be lowered. The role of KBE developer should be easily undertaken by engineers and not only by programming gurus. As discussed in Chapter 3, Section 3.10.1, programming languages can have a fundamental role here: they should resemble the "everyday engineering language" and also relieve the KBE developer from any burden related to memory management and similar low level details.

7.7.3 Supporting interactive geometry manipulation

Furthermore the time required to write KBE applications must be drastically reduced. On one hand the full programming approach gives total control to developers, on the other it might make certain operations very cumbersome, especially concerning geometry manipulation. Though more than 70% of a product's definition is often non-geometric, more than 70% of coding/debugging time is spent in geometry manipulation. Therefore, the capability to have some automatic code generation, while "manually" performing geometry manipulation in a more advanced graphical interface than current KBE systems, would be of great advantage. Besides, the graphical interface of typical KBE systems, like ICAD and GDL, allows just the visualization of the geometry produced by the product model, but does not allow (or only in a very limited extent) the interaction with that geometry, e.g., the mouse-selection of a part to query and modify its attributes.

7.7.4 Supporting web collaborative solutions and open source initiatives

When the first KBE system came into the market, the World Wide Web did not exist yet, neither the concept of open source software. It is natural to expect new KBE systems (on the contrary of the first generation systems) to be web-friendly, hence geared toward their use and deployment via the web (Cooper and La Rocca, 2007). To be noted that this option would free users from any demands on the kind of computer system required using the given KBE application; a web browser would suffice. The possibility to free the way to open source solutions, from one side, would relieve the end-user to return so often (with cash) to the "KBE vendor shop" and buy dedicated plug-ins (e.g., for VRML and X3D graphic generation and visualization, or for the generation of PDF and XML files etc). From the other side it would relieve the

KBE vendor from the in house development of typically short life and buggy applications. Indeed, an open source application is likely to have fewer bugs, since a large number of peers had the possibility to look at it and bring improvements.

7.7.5 The value of dynamic *code → ← documentation* generation

In order to allow developers accessing and understanding applications written by others, code documentation is extremely important; unfortunately producing and keeping updated quality documentation is as much useful as time-consuming. A KBE system able to autonomously produce *descriptive documentation* of its own application (possibly implementing standard visual modeling languages such as the UML[54]), would be more than welcome to any developer who, at least once, has struggled to debug or rework others' code…and often also his/her own!

The automatic link code-documentation should actually work *dynamically in both directions*. Engineers should have the possibility to agilely and interactively generate diagrams representing the structure and the design process of their products, having an interpreter active in the background that is able to generate at least the main structure of the KBE application code. This would allow passing from the formal knowledge representation provided by knowledge engineers (e.g., the MOKA formal model), directly to the programmed application (or at least its main structure). Furthermore, it should be possible to visualize the structure of a given application and modify its code just by modifying its diagrammatic representation[55]. In this case, the barrier of the programming interface would be completely abated, eventually allowing engineers to play the role of engineers rather than software developers…

7.7.6 Automatic KBE code generation. Issues and Opportunities

As discussed in section 7.5.1, MOKA already addressed the issue of dynamic code generation mapping from the MML formal model. Indeed, the fact that tools like PCPACK use an internal XML representation of the knowledge models interactively

---

[54] Indeed, KBE systems like ICAD and GDL offer a so called Documentation Tool, which allows the automatic generation of reference documentation from a set of KBE code files. However useful, it is not the kind of visual representation offered for example by the UML.

[55] It should be acknowledged that in May 2002, during the International ICAD User Group in Boston, KTI announced and presented ICAD Release 9, which was including a so called IDE (Integrated Development Environment) to help engineers developing ICAD applications by means of dragging and dropping predefined and user-defined classes on a canvas, to be connected and adjusted as needed. Release 9 included also a graphical user interface to allow interactive geometry picking. In November 2002, KTI was acquired by Dassault Systemes, the ICAD development and soon after its support stopped, Release 9 never came to the market.

generated by knowledge engineers, offers the possibility to use XSLT[56] to translate those knowledge models into KBE code (see sketch in Fig. 7.6).

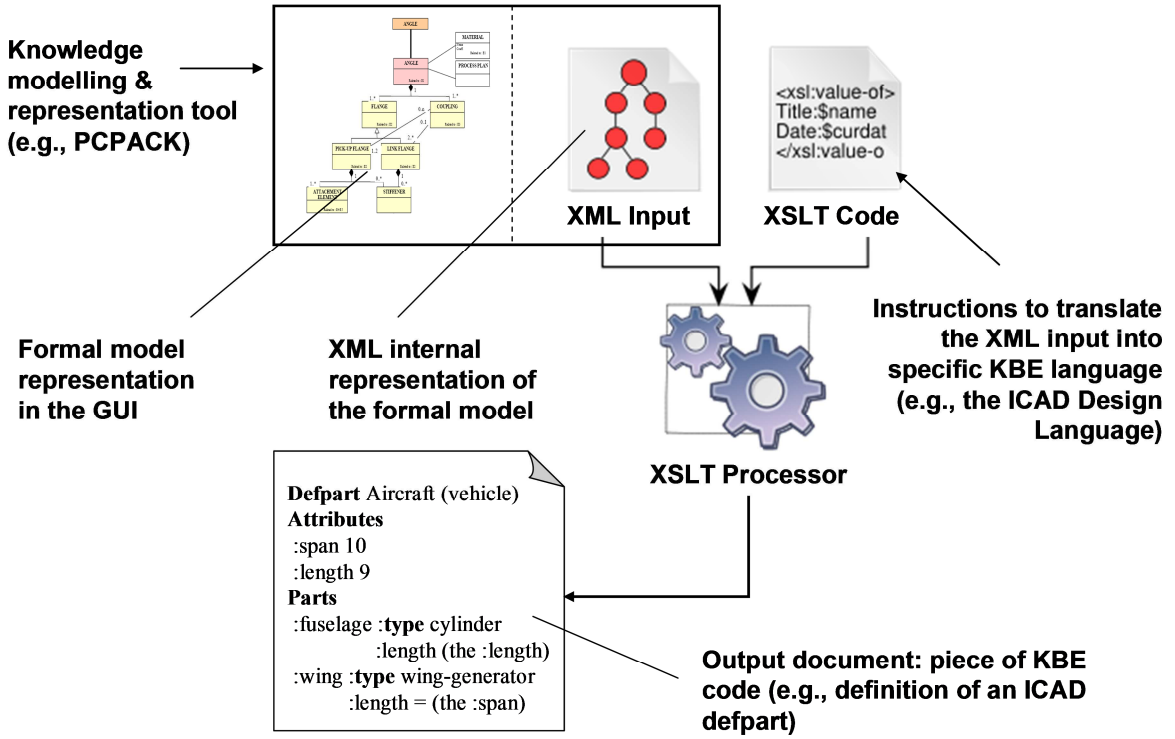


Fig. 7.6: principle of automatic generation of KBE code from a formal knowledge representation, by means of XLST.

Two issues can be foreseen. First different translators should be developed to pass from the possibly different XML knowledge representation provided by different PCPACK-like tools to the different syntaxes of various KBE systems. Second, and this is a more subtle issue, a KBE product model actually consists of a mix of the "pure" domain knowledge (as elicited from the expert and stored in the formal knowledge model), and the so called *implementation-specific knowledge.* The latter consists of the *KBE system-specific method* to implement the domain knowledge. For example, the formal model can specify that a certain surface has to be segmented in parts along the intersection with some other surfaces. However, the way to achieve this segmentation can largely vary from one KBE system to another. The KBE developer will have to deal with the practices of the given KBE system and the possibilities offered by its programming language. Experts report cases when using a certain

[56] Extensible Stylesheet Language Transformations (XSLT) is an XML-based language developed by the World Wide Web Consortium (www.wc3.org), used for the transformation of XML documents into other XML, HTML or some other "human-readable" documents.

language, a given function could be programmed in 2 lines against over 100 required by another language! (Knudson; Graham, 2004)
Eventually, according to the problem at hand, a KBE product model can consists of equal amounts of *domain knowledge* and *specific implementation knowledge*. There cannot be automatic generation of KBE applications, without dealing with the latter.

Recently the OMG has issued a request for proposal (Object Management Group, 2005) soliciting a standardization of KBE services in order to facilitate sharing *the information that generates engineering data*. In other words, a platform independent model for the exchange of knowledge in terms of the currently available constructs in KBE, such as engineering rules and relations.
As discussed in Chapter 3, a way to relieve the problem could be the use of *standard programming* languages, rather than proprietary. However, the competitive advantage of certain KBE systems exists in some of their very non standard and unique functionalities!

CHAPTER 8
Conclusions and Recommendations

1. Conclusions
2. Recommendations
3. A glimpse of the next-generation MMG

8.1 Conclusions

The vision and the crude reality

In 2002, following the publication of the Vision 2020 document, the Advisory Council for Aeronautical Research in Europe defined the roadmap to drive the European aeronautical industry towards a set of very challenging sustainable growth targets. The target concerning the environmental impact is seen as the most difficult to achieve, practically impossible without important breakthroughs, both in technology and in concepts of operation.

Since then, the Airbus A380 has entered into service; the Boeing 787, notwithstanding an already accumulated delay of 3 years, is on its way, while the unconventional design of the Sonic Cruiser seems doomed to dust on the shelves of some Boeing archive. Airbus is expecting to deliver the first A350 in 2013, while the A30X, the replacement for A320 successful single aisle family, cannot be expected earlier than 15 years from now.

None of these aircraft feature the technology to meet the awaited ACARE targets. Since these are the aircraft that will operate in the next 40 years, the bankruptcy of Vision 2020 can be already declared. Radical innovation is indeed a too risky business.

So now what?

The search for possible solutions must begin from a careful observation of the actual context:

- The modern aerospace company is transnational. It operates in the global market across a large, complex and distributed supply chain.

- New aircraft development programs are increasing complex and have a much longer duration. The amount of time spent for the conceptual design of a modern aircraft is more than the time spent in the 1940s for a complete aircraft development, including production.
- The brains drain phenomenon is taking away a large part of the already scarce knowledge resources that are indispensable to deal with the abovementioned challenges. At the same time, the modern knowledge worker needs to be managed differently.
- The Kansas City aircraft design is the result of more than 60 years of evolutionary optimization. However, the recent changes in the top level requirements (i.e., the new and more stringent environmental impact constraints) are modifying the morphology of the design space. Most likely the new optimum can be found somewhere else.

To be able to step into the new age of sustainable growth, it will be necessary to realize that

- Knowledge is a key business asset, strategically more important than other classical more tangible resources. As such, it must be properly managed and engineered.
- Changes are required in the very design approach. New tools and methods are indeed required to
  - Support the design process across large distributed teams
  - Increase the productivity of the scarce intellectual resources
  - Better support the decision making process
  - Free time for innovation and exploitation of engineering skills
  - Lower the risk associated with the development of novel aircraft configurations

The MDO promise and challenges

The MDO approach appears to be the most promising design methodology in the field of aircraft design, both to improve the performance of traditional aircraft configurations, and to support the development of novel concepts. Actually, non conventional configurations such as blended wing bodies and Prandtl planes, due to their intrinsic level of integration, are expected to benefit the most from the multidisciplinary design optimization approach.

The opportunity to bring more knowledge upfront in the design process, while extending the designer's freedom to make late changes; the ability to account and exploit any discipline interaction; the prospected capability to thoroughly explore the design space and find new non trivial solutions are the dreams of any designer.

However, a number of technical (and non technical) barriers have prevented a full exploitation of the MDO approach so far, and limited its industrial application mostly to detail design cases. The lack of adequate generative modeling systems

able to hook up to distributed and heterogeneous sets of analysis tools and the frustration from the lengthy and repetitive preparation work demanded by the iterative multidisciplinary analysis process have been indicated as some of the most critical issues for the implementation of MDO systems, especially in the conceptual design phase. The integration of high fidelity analysis tools, due to their lack of robustness and flexibility, has demonstrated particularly challenging, thereby reducing the impact of the MDO approach as a whole.

The DEE concept and the development of the MMG

To this purpose, the concept of Design and Engineering Engine has been developed, which is a modular, loosely integrated design system able to support the multidisciplinary analysis and optimization process by automating all the repetitive and non creative activities. One of the core components of the DEE is the Multi Model Generator, which is actually the technological enabler of the DEE CAD-centric architecture and represents the main outcome of this research work.

Due to their current limitations in supporting the functional thinking approach of designers and, in particular, to their lack of knowledge recording and reuse capabilities, conventional CAD systems are not able to provide the level of design automation and flexibility necessary to support MDO.

On the contrary, the inherent capability of KBE to integrate object-oriented rule-based design with the geometry manipulation skills of a top-class parametric CAD system, can offer the required generative modeling capability.

For this reason, the Multi Model Generator has been developed using a commercial KBE platform, where the KBE ability to define and manipulate geometry and other engineering knowledge via an object oriented programming language has been exploited to define the two main types of MMG components, namely the High Level Primitives and the Capability Modules.

The High Level primitives

The HLPs, with their modular and parametric structure, can be considered as smart LEGO blocks, which enable designers to build up a large number of aircraft configurations and configurations variants, including novel concepts.

The HLPs can support designers better than the conventional CAD primitives, because of two main reasons:

1. They are functional blocks, hence they can better match the way of thinking of a designers, which derive shapes based on functionalities to be achieved (e.g., generate lift, accommodate payload, etc.).
2. They are able to record knowledge and reuse it to adapt their own shape and topology, or trigger some event, as a reaction of some input change.

Indeed, whenever a HLP parameter value and/or the number and type of HLPs used to model a given aircraft configuration is changed (e.g., by an optimizer), the rules

integrated in the product model enable the aircraft to reconfigure and adjust itself, automatically, without any burden for the designer.

The Capability Modules

On the basis of the following observations:

1. Quite independently from the aircraft configuration at hand, the same analysis tools and preprocessing methods are generally used by specialists
2. A large part of the preprocessing activities required to transform a geometrical model into a dedicated abstraction for disciplinary analysis are systematic, repetitive and follow known rules

A number of so called Capability Modules has been developed to capture discipline expert knowledge and reuse it to automate the generation of dedicated models (also called discipline abstractions or views) for a broad range of analysis tools, including in-house developed and commercial off the shelf, low fidelity and high fidelity tools.
In particular, it has been demonstrated how one of the biggest MDO challenges can be met by means of KBE, i.e., the automatic generation and modification of FE models, both for a complete aircraft model or a component, independently of the vehicle configuration and its internal structure layout.

High fidelity analysis in the conceptual design process

The enabled use of high fidelity analysis tools in the early stages of the design process can largely increase the level of confidence in the performance prediction of the design under consideration. While this is a significant achievement in general, it actually represents a fundamental step towards the development of novel aircraft configurations.
In this respect, the EC sponsored project MOB, has demonstrated the effectiveness of the MMG tool to support the design of a blended wing body configuration, where the inherent strong disciplines couplings and the lack of reference data make the use of traditional design methods less effective and stem for a full MDO approach, based on first principle analysis.
It can be noted that the early use of high fidelity analysis systems enabled by the KBE approach tends to blur the boundaries between conceptual and preliminary design. Indeed the availability of one modeling system able to serve both low and high analysis tools can improve the transition between the two design phases and avoid discontinuities in the models supply process.

The joy and scope of design automation

The application of the MMG (within the DEE) for the design of a complete aircraft or some of its major components has shown that is possible to reduce the length of the analysis process from months to days, from weeks to minutes.

As a result, more cases can be evaluated during the time normally required by a single manual iteration. More what-if scenarios can be investigated. More time can be dedicated to creative design.
Since discipline experts do not have to assemble manually new analysis models when changes occur in the product configuration, their level of frustration decreases, as well as the occurrence of human errors, while design rules and best practices are systematically applied.
Knowledge Based Engineering, with its set of tools from the world of Artificial Intelligence, has proven a valuable technology to bring more automation in design, but, with the very limited scope to repetitive and non creative activities. In other words, KBE can support the analysis and optimization of good ideas, but cannot generate ideas. For that, the designer is always in charge.

Enabling distributed design
The capabilities to be accessed in remote via a web connection and work in batch mode allow the MMG to operate inside a real distributed design and optimization environment. Actually, the MMG itself becomes an enabler for distributed design. A large range of non-geographically collocated tools and experts can be supplied any time with dedicated models that are guaranteed to be consistent because extracted, on the fly, from the same product definition.

Modularity to grow, adapt and survive
The modular structure of the DEE system, as well as of the MMG and its components provides optimal conditions for maintenance, debugging and incremental development.
Furthermore, this modular structure allows a prompt integration of new and different analysis capabilities and facilitates the reconfiguration/adaptation of the system to different design cases, or the reuse of single modules in different DEEs or just as stand alone.
To cite a founding work of the modern age: *"It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change"* (C. Darwin, The Origin of Species).

The thread of micro S-curves for the MDO macro S-curve
As discussed in Chapter 1 (Section 1.2, Fig. 1.5), jet engines would not have brought to a step change in aviation performance without the enabling contributions of metal structure, axial compressors, swept wings, retractable landing gears, etc.
Similarly, the impact of MDO will depend on the various advances in the fields of optimization techniques, fast computing, communication & integration frameworks, high fidelity analysis tools and, last but not least, smart modeling systems. It is right here that this research work is intended to contribute.

There is (KBE) life outside planet Aerospace Engineering

The methodology described in this research work to support aircraft design and optimization finds a lot of application also outside the aerospace field. The design of any complex craft that requires the involvement of many discipline stakeholders and whose optimum performance depends on non intuitive combinations of physical and process parameters can strongly benefit from the technologies discussed in this work. As a matter of fact, dedicated DEEs and MMGs have been developed and demonstrated in other two EC projects.

The first one, called UPWIND (www.upwind.eu), is a 5th framework project investigating the development of new extra large scale wind turbines, where a MMG is required to feed a range of distributed analysis and simulation tools (Chiciudean, La Rocca and Van Tooren, 2008; Cooper and La Rocca, 2007).

The second one, called PEGASUS (www.pegasus-eu.net), is a 6th framework project that aims at improving the development process of plastic car components, by means of a better and more pro active integration of knowledge and tools from OEM and suppliers. In this case a DEE is being developed to optimize the design of injection molding tooling for minimum cost, environmental impact, and manufacturing process time (van Dijk et al., 2011).

8.2 Recommendations

A KBE application as the one illustrated in this research work has the tendency to become one of these systems whose development never stops! Every success in automating a part of the design work, every achieved link with another analysis tool, will bring the happy user (actually the increased number of happy users) back to you with a new wish.

Basic lessons from the IT world

However, the modular incremental development approach alone will not suffice to sustain such a continuous growth, neither will it facilitate code reuse and sharing, if not complemented with some software development best practice. These practices are not always familiar or properly implemented by the common aerospace engineer, until too late...

These include the following guidelines:

- Make use of understandable naming for attributes, functions, classes, etc.
- Never hardcode parameters inside the code, including explicit paths to external files and disks
- Keep program code as clean and structured as possible (in a way, any code file must be a comfortable place to work in)
- Always document the code, both inserting explanations in between the code lines, and by means of external documentation. What seems to be a trivial

function definition to you today, it will not to another developer or to yourself in a few months (this would be an inconvenient degradation of knowledge back into data)
- Organize the various pieces of source code according to some agreed filing structure (ontology). For example different folders/subfolders for generic classes and functions, HLPs, CMs, libraries, input and output areas, etc.
- Make use of a control revision system and a properly backed-up file repository to store and share the code. Do not commit code into the repository that has not been tested, unless you are and you will remain the only user of that code.
- Develop and keep up to date a code testing system. Before committing any new version of the code or just some new module, all tests must be passed (e.g., it should be possible to generate all the possible output files for a number of significant aircraft configurations), to verify that all the previous functionalities are still intact.
- When a stable version of the KBE application is available, it is convenient to compile it into an executable file, which is the most robust and convenient form to distribute the application (no local installation of the KBE system will be required on the machine where the executable has to run)

Lean to be lean

Developing a quality KBE application is not trivial and can be very time intensive. Code (re)structuring, testing and documentation generation can take a significant amount of the overall development time. Cutting time by saving on these activities is not uncommon, especially within a research environment where the proof of concept is generally the goal, rather than a finalized tool ready for professional deployment.
However, within an industrial environment this is generally not an option. Different strategies are required to compress the development time of a KBE application, such to increase its return of investment or justify its development (pressure can be high when the development costs must be covered by a single project).
As KBE is used to achieve a lean approach in design, methods are needed to achieve a lean KBE application development.
New PhD research programs are currently on going at the TU Delft faculty of Aerospace Engineering, with the scope of developing a suite of tools and methods to increase the productivity of KBE developers. Among others the following items need to be investigated (see also Section 7.7):

- Automated code documenting systems
- Automated code generation/manipulation from KBE platform independent knowledge models
- Improved debugging and testing systems
- Ontology development to facilitate structuring and reuse of KBE modules

- Extension and revision of the MOKA ontology to support the development of DEEs (and similar MDO frameworks) and not only of pure KBE applications.

A new look at the MMG

Although, the modeling capabilities of the MMG described in this work have proven capable of generating very different aircraft configurations and variants, it is recommended to consider the following improvements:

- Allow for a non linear distribution of chord lengths, dihedral and twist
- Allow for a free placement and orientation of airfoils all over the span, including the possibility to have canted root and tip wing sections
- Ease the approach to define curve leading and trailing edges (also to reflect the more curvy shapes allowed by manufacturing with composite material)
- Allow the possibility to model complex (cranked) planform by means of a single wing-part primitive to limit the amount of required wing part instances and relative connections
- Increase the flexibility of the structural element definition and placement to avoid, for example, redefining spars for each wing-part instances inside a cranked wing, even when the spars extend continuously from the wing root to the tip
- Allow the definition of multi element wings with movables that are free to be deflected and extracted
- Similar improvements in term of modeling flexibility should be considered also for the Fuselage and the Engine primitives. The integration of new types of engines on new aircraft configurations is going to become a very interesting aspect, where the DEE can offer a lot.
- Although the MMG has been developed to operate in batch, driven by a user defined input file, it would be convenient to develop a graphical user interface to facilitate both the generation/modification of the input file and the interactive operation of the MMG.
- Expand the scope of the MMG (and the overall DEE) to other "hot design areas", such as systems integration and wires/piping routing. The recent problems on the A380 wire harnesses routing and their impact on the overall delivery schedule are sufficient to justify the relevance of this research area. The complexity of these systems has grown so drastically that it is no more convenient neither possible to address them as late as during the detail design phase.

8.3 A glimpse of the next-generation MMG

Whilst recommendations are a nice academic way of stating *"I did my part! Someone else will take care of cleaning my mess and solving the problems I was not able to*

solve", the author had the privilege to initiate the development of a new generation MMG and already take care of some of the recommendations listed above.
After the ICAD exit from the market, GDL, the new KBE platform by Genworks International, has been adopted at our chair to start the development of the next-generation MMG. The activities to recode (an improved version of) the HLPs and CMs are already ongoing and Fig. 8.1, Fig. 8.2 and Fig. 8.3 offer a peek on the most recent developments.

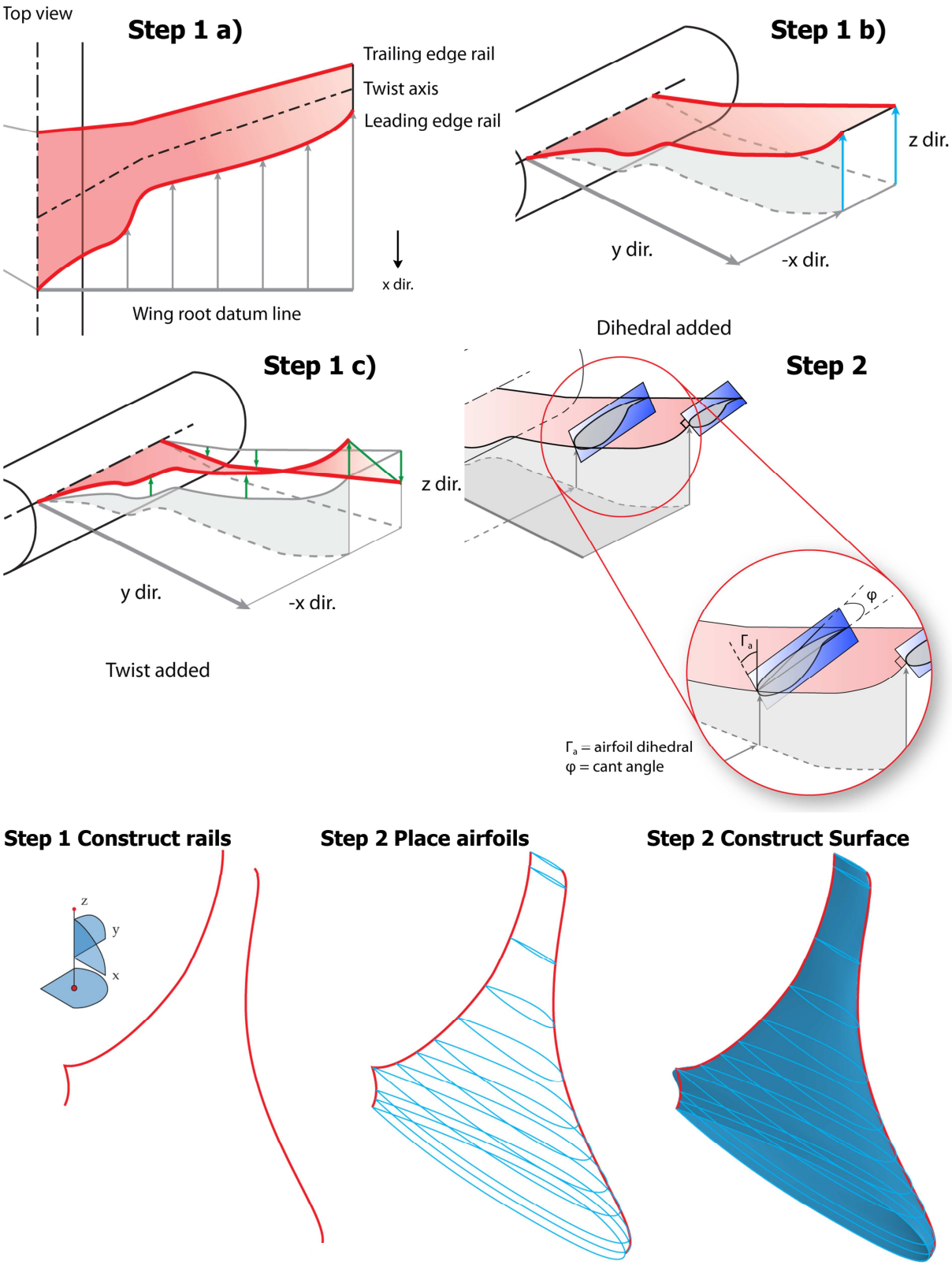


Fig. 8.1: Definition of the new Wing-Part HLP. Two curvilinear and not necessary continuous rails are defined and used to "hinge" airfoils at any angle. Twist, dihedral and sweep angles distributions are not necessarily linear (Koning, 2010).

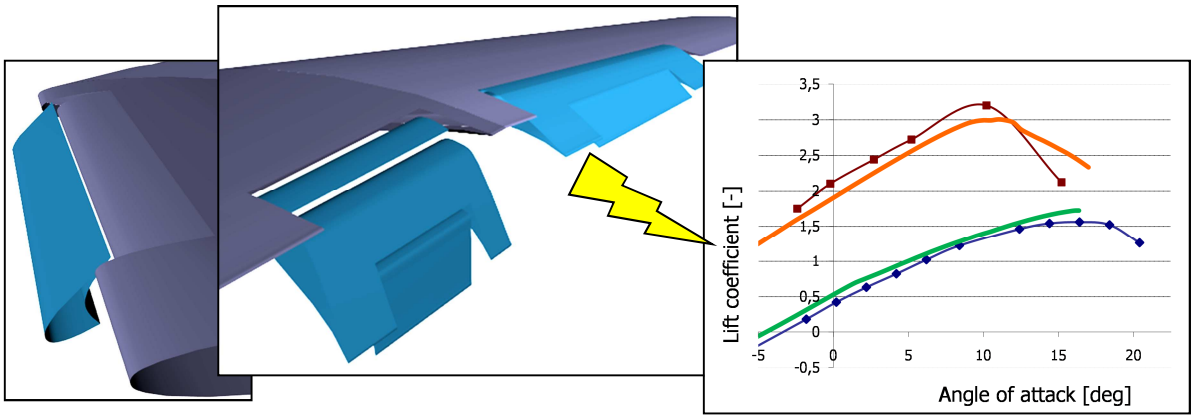


Fig. 8.2: Examples of MMG generated multi-element wings (slat and triple slotted flaps). Direct links both to CFD simulations (MSES) and semiempirical methods (ESDU) for the derivation of the flapped wing lift curves are in place (van den Berg, 2009).

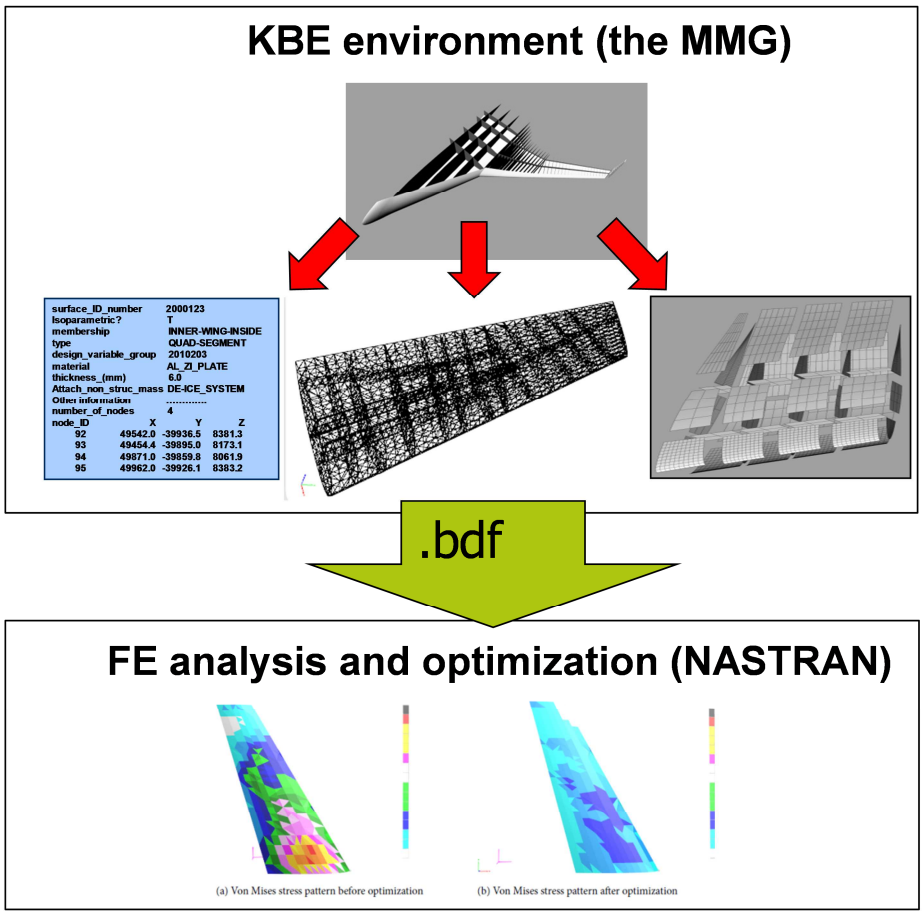


Fig. 8.3: The GDL surface tessellation capability is used to generate suitable grids for FE structural analysis directly within the MMG. A new Capability Module generates the NASTRAN bulk data deck (.bdf) file without the need of Pycoco and PATRAN. This can simplify the MMG-NASTRAN integration approach discussed in Section 6.4 (van Hoek, 2010).

References

ACARE. 2002. Strategic Research Agenda 1 - Executive Summary & Volume 1-2. 1-2. Available: http://www.acare4europe.com.

ACARE. 2004. Strategic Research Agenda 2 - Executive Summary & Volume 1-2. 1-2. Available: http://www.acare4europe.com.

ACARE. 2008. 2008 Addendum to the Strategic Research Agenda. Available: www.acare4europe.com.

ACARE. 2010. Aeronautics and Air Transport: Beyond Vision 2020 (Towards 2050) - *background document*. Available: www.acare4europe.com.

AIRBUS. 2009. Flying Smart, Thinking Big. Global Market Forecast 2009-2028. Available: www.airbus.com.

ALAGNA, A. 2005. *Development of a Cabin Layout and Fuselage Design Tool.* MSc Thesis, TU Delft.

ALLWRIGHT, S. 1996. Technical Data Management for Collaborative Multidiscipline Optimisation. *AIAA Conference on Multidiscipline Optimisation.* Seattle.

ALONSO, J. J. 2008. Requirements for MA&O in the NASA Fundamental Aeronautics Program. *In:* AIAA (ed.) *12th AIAA/ISSMO MA&O Conference.* Victoria, BC, Canada.

ANALYTICAL METHODS, INC., 2009. VSAERO - Integral methods for potential and boundary layer flows. Redmond, WA.

ASHLEY, H. 1982. On Making Things the Best—Aeronautical Uses of Optimization. *Journal of Aircraft.* AIAA.

AUTY, D. 1988. Object Oriented Programming Systems and Frame Representations, an Investigation of Programming Paradigms. *In:* NASA (ed.) *Technical Report 186084.* NASA.

BARTHOLOMEW, P. 1998. The Role of MDO within Aerospace Design and Progress towards an MDO Capability through European Collaboration. *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.* St. Louis, MO: AIAA.

BATHIA, K. G. 2008. A perspective on Optimization. *12th AIAA/ISSMO MA&O Conference.* Victoria, BC, CANADA: AIAA.

BELIE, R. 2002. Non-technical barriers to multidisciplinary optimisation in the aerospace industry. *9th AIAA/ISSMO Symposium of Multidisciplinary Analysis and Optimisation.* Atlanta, Georgia: AIAA.

BENNETT, J., FENYES, P., HAERING, W. & NEAL, M. 1998. Issues in Industrial Multidisciplinary Optimization. *In:* AIAA (ed.) *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.* St. Louis, MO: AIAA.

BERENDS, J., VAN TOOREN, M. J. L. & SCHUT, J. E. 2008. Design and Implementation of a New Generation Multi-Agent Task Environment Framework. *In:* AIAA (ed.) *49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference.* Schaumburg, IL: AIAA.

BERENDS, J. P. T. J. & VAN TOOREN, M. J. L. 2007 Design of a Multi-Agent Task Environment Framework to support Multidisciplinary Design and Optimisation. *45th AIAA Aerospace Sciences Meeting and Exhibit.* Reno, NV: AIAA.

BLOUIN, V. Y., SUMMERS, J. D. & FADEL, G. M. 2004. Intrinsic Analysis of Decomposition and Coordination Strategies for Complex Design Problems. *In:* AIAA (ed.) *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference.* Albany, New York: AIAA.

BOWCUTT, K. 2003. A perspective on the future of aerospace vehicle design. *12th AIAA International Space Planes and Hypersonic Systems and Technologies.* Norfolk, Virginia: AIAA.

BRIMBLE, R. & SELLINI, F. 2000. The MOKA Modelling Language. *Knowledge Engineering and Knowledge Management Methods, Models, and Tools.* Berlin/Heidelberg: Springer

BRODERSEN, O., TAUPIN, K., MAURY, E., SPIEWEG, R., LIESER, J., LABAN, M., GODARD, J. L., VITAGLIANO, P. L. & BIGOT, P. 2005. Aerodynamics Investigations in the European Project

ROSAS (Research on Silent Aircraft Concepts). *35th AIAA Fluid Dynamics Conference and Exhibit.* Toronto.
BROUWERS, J. J. J. 2007. *Parametric Modelling of Aircraft Tail Configurations to Support Aerodynamic Analysis.* MSc, TU Delft.
BUTLER, R., LILLICO, M., HANSSON, E. & VAN DALEN, F. 1998. Comparison of MDO codes for use in conceptual and preliminary aircraft wing design. *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.* St. Louis, MO: AIAA.
CARMICHAEL, R. *Tables from Theory of Airfoil Sections* www.pdas.com [Online]. Santa Cruz, CA [Accessed November 2009].
CARPENTIERI, G. 2008. *An adjoint-based shape-optimization method for aerodynamic design.* PhD Dissertation, Delft University of Technology.
CARTY, A. & DAVIES, C. 2004 Fusion of Aircraft Synthesis and Computer Aided Design *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference.* Albany, New York: AIAA.
CERULLI, C., BERENDS, J. P. T. J., VAN TOOREN, M. J. L. & HOFSTEE, J. W. 2005. Parametric Modeling for Structural Dynamics Investigations in Preliminary Design. *2005 CEAS/AIAA/DGLR International Forum on Aeroelasticity and Structural Dynamics.* Munich, Germany: AIAA.
CERULLI, C., MEIJER, P. B., VAN TOOREN, M. J. L. & HOFSTEE, J. 2004. Parametric Modeling of Aircraft Families for Load Calculation Support *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference.* Palm Springs, California: AIAA.
CERULLI, C., SCHUT, E. J., BERENDS, J. P. T. J. & VAN TOOREN, M. J. L. 2006. Tail Optimization and Redesign in a Multi Agent Task Environment. *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference.* Newport, RI, USA: AIAA.
CERULLI, C., VAN KEULEN, F. & RIXEN, D. J. 2007. Dynamic Reanalysis and Component Mode Synthesis to Improve Aircraft Modeling for Loads Calculation. *48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference.* Honolulu, Hawaii: AIAA.
CHAPMAN, C. B. & PINFOLD, M. 1999. Design Engineering – A Need to Rethink the Solution Using Knowledge Based Engineering *Knowledge-Based Systems,* 12, 257-267.
CHAPMAN, C. B. & PINFOLD, M. 2001. The application of a knowledge based engineering approach to the rapid design and analysis of an automotive structure. *Advances in Engineering Software,* 32 pp. 903-912.
CHICIUDEAN, T., LA ROCCA, G. & VAN TOOREN, M. J. L. 2008. A Knowledge based engineering approach to support automatic design of wind turbine blades. *Design Synthesis, CIRP Design Conference.* University of Twente, NL.
CLARKSON, P. J. 2000. 'Signposting' for risk reduced innovation in aerospace design. *Knowledge Based Organisation conference.* Paris.
COOPER, D. J. & LA ROCCA, G. 2007. Knowledge-based techniques for developing engineering applications in the 21st century. *7th AIAA Aviation Technology, Integration and Operations Conference.* Belfast, Northern Ireland: AIAA.
COOPER, D. J. & SMITH, D. F. 2005. A Timely Knowledge-Based Engineering Platform for Collaborative Engineering and Multidisciplinary Optimization of Robust Affordable Systems. *International Lisp Conference 2005.* Stanford University: Stanford.
COOPER, S., FAN, I. & LI, G. 2001. Achieving Competitive Advantage Through Knowledge Based Engineering - A Best Practice Guide. Department of Trade and Industry.
DE WECK, O., AGTE, J., SOBIESZCZANSKI-SOBIESKI, J., ARENDSEN, P., MORRIS, A. & SPIECK, M. 2007. State-of-the-Art and Future Trends in Multidisciplinary Design Optimization. *48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference.* Honolulu, Hawaii: AIAA.
DELONG, D. 2004. *Lost Knowledge: Confronting the Threat of an Aging Workforce,* New York, Oxford Univeristy Press.
DELONG, D. 2008. Knowledge loss prevention. Five keys to decisions vis-à-vis an ageing workforce. *InsideKnowledge.*
DING, Y. 1986. Shape Optimization of Structures: A Literature Survey. *Computers and Structures,* 24-6, 985-1004.
DIRCKEN, F. W. M. 2008. *Flight Mechanics Modelling within a Design Framework.* MSc, TU Delft.

DOHERTY, J. J. & DEAN, S. R. H. 2007. MDO-based concept optimisation and the impact of technology and systems choices. *In:* AIAA (ed.) *7th AIAA Aviation Technology, Integration and Operations Conference.* Belfast, Northern Ireland: AIAA.

DOWLING, A. P. & HYNES, T. 2006. Towards a silent aircraft. *Aeronautical Journal,* 110-1110, 487-494

DRUCKER, P. F. 1999. *Management Challenges of the 21st Century,* New York, Harper Business.

DUDA, R., GASCHNIG, J. & HART, P. 1979. Model Design in the PROSPECTOR Consultant System for Mineral Exploration. *Expert Systems in the Microelectronic Age*, pp. 153-167.

DUNLOP, M. 2010. Brain Drain Among Boeing's biggest challenges. *Herald Net.* Everett, WA.

EDGERTON, D. 2006. *Shock Of The Old: Technology and Global History since 1900: Technology in Global History Since 1900,* London, Profile Books LTD.

ENGELMORE, R. S. & FEIGENBAUM, E. 1993. Expert Systems and Artificial Intelligence. *Loyola/WTEC Study Report* [Online]. Available: http://www.wtec.org/loyola/kb/c1_s1.htm.

ENGELS, H., BECKER, W. & MORRIS, A. J. 2004. Implementation of a multi-level optimisation methodology within the e-design of a blended wing body. *Aerospace Science and Technology,* 8-2, 145-153.

EPISTEMICS. UK. Available: www.epistemics.co.uk [Accessed October 2009].

FARIN, G. 1988. *Curves and surfaces for computer aided geometric design: a practical guide,* San Diego, CA, USA Academic Press Professional, Inc.

FEIGHENBAUM, E. A., BUCHANAN, B. G. & LEDERBERG, J. 1971. On Generality of Problem Solving: a Case Study Using the DENDRAL Program. *Machine Intelligence,* Vol. 6, pp 165-190.

FODERARO, J. 1991. LISP: Introduction. *Communications of the ACM,* 34-9, 27.

FREDIANI, A. 2004. The PrandtlPlane. *ICCES04.* Madeira, Portugal: Tech Science Press.

GAUDIN, J. 2008. Structure Analysis Contribution to multi-disciplinary optimization. *12th AIAA/ISSMO MA&O Conference.* Victoria, BC, Canada: AIAA.

GERO, J. S. 1985. *Knowledge Engineering in Computer-Aided Design,* Amsterdam.

GERO, J. S. 1987. *Expert Systems in Computer-Aided Design,* Amsterdam.

GIESING, J. P. & BARTHELEMY, J. M. 1998. A Summary of Industry MDO Applications and Needs. *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.* St. Louis, MO: AIAA.

GRAHAM, P. 1995. *ANSI Common Lisp,* Englewood Cliffs, New Jersey, Prentice Hall.

GRAHAM, P. 2004. *Hackers & Painters. Big ideas from the computer age,* Sebastopol, CA, O'Reilly.

GREEN, J. E. 2003. Civil aviation and the environmental challenge. *The Aeronautical Journal,* 107-1072, 281-299.

GREENER-BY-DESIGN-GROUP 2005. Air Travel - Greener by Design. *In:* SOCIETY, R. A. (ed.). London.

GREENER BY DESIGN, SCIENCE AND TECHNOLOGY SUB-GROUP, 2005. Air Travel - Greener by Design. Mitigating the Environmental Impact of Aviation: Opportunities and Priorities. *In:* SOCIETY, R. A. (ed.). London.

GROTENHUIS, J. 2007. *Development of a conceptual controllability analysis tool.* MSc, TU Delft.

GROUP OF PERSONALITIES. 2001. European Aeronautics: a Vision for 2020. Available: www.acare4europe.com.

HAFTKA, R. T. & GRANDHI, R. V. 1986. Structural Shape Optimization - A Survey. *Computer Methods in Applied Mechanics and Engineering,* 57-1, 91-106.

HEPPERLE, M. *Airfoil database and tools* [Online]. Braunschweig, Germany. Available: www.MH-Aerotools.de [Accessed October 2009].

HOENLINGER, H., KRAMMER, I. & STETTNER, M. 1998. MDO Technology Needs in Aeroservoelastic Structural Design. *In:* AIAA (ed.) *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.* St. Louis, MO: AIAA.

HOFSTEE, J., KIER, T., CERULLI, C. & LOOYE, G. 2003. A Variable, Fully Flexible Dynamic Response Tool for Special Investigation (VarLoads). *CEAS/AIAA/NVvL International Forum on Aeroelasticity and Structural Dynamics.* Amsterdam, The Netherlands: AIAA.

ICAD [Online]. Available: http://en.wikipedia.org/wiki/ICAD [Accessed December 2010].

KBE coupling illustration. 2000. *MOKA Technical Report Deliverable D.3.5.*

Knowledge-based engineering [Online]. Available: http://en.wikipedia.org/wiki/Knowledge-Based_Engineering [Accessed December 2010].
KNOWLEDGE TECHNOLOGIES INTERNATIONAL White paper - Knowledge Based Engineering and the ICAD system. Burlinghton, MA.
KNOWLEDGE TECHNOLOGIES INTERNATIONAL 2001a. The ICAD System Output Interface User's Manual (The KBO environment documentation). 2.0 ed.
KNOWLEDGE TECHNOLOGIES INTERNATIONAL 2001b. The ICAD System Surface Designer User's manual (The KBO environment documentation). Release 2.0 ed.
KNUDSON, S. *KBE history* [Online]. Available: http://www.stanleyknutson.com/kbe-history.html [Accessed October 2009].
KONING, J., H. 2010. *Development of a KBE Application to Support Aerodynamic Design and Analysis. Towards a next-generation Multi-Model Generator.* MSc Thesis, TU Delft.
KOOPMANS, W. 2004. *Parametric Modeling of Unsteady Aerodynamics for Loads Estimation in the Pre-design Phase.* MSc Thesis, TU Delft.
KRAKERS, L. A. 2009. *Parametric fuselage design - Integration of mechanics and acoustic & thermal insulation.* PhD Dissertation, TU Delft.
KROO, I. 1997. Multidisciplinary Optimization Application in Preliminary Design – Status and Directions. *38th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit and Adaptive Structures Forum.* Kissimmee, FL: AIAA.
KROO, I. 2004. Innovations in Aeronautics. *In:* AIAA (ed.) *42nd AIAA Aerospace Sciences Meeting and Exhibit.* Reno, Nevada: AIAA.
KULFAN, B. 2008. Universal Parametric Geometry Representation Method *Journal of Aircraft,* 45-1, 142-158.
LA ROCCA, G., KRAKERS, L. & VAN TOOREN, M. J. L. 2002. Development of an ICAD Generative Model for Blended Wing Body Aircraft Design. *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimisation.* Atlanta, GA.
LA ROCCA, G. & VAN TOOREN, M. J. L. 2002a. Description of the ICAD BWB Structure Generator Code (issue 2). *MOB Technical Reports.* Delft: TU Delft.
LA ROCCA, G. & VAN TOOREN, M. J. L. 2002b. Description of the ICAD BWB Surface Generator Code (issue 3). *MOB Technical Reports.* Delft: TU Delft.
LA ROCCA, G. & VAN TOOREN, M. J. L. 2002c. Description of the ICAD Multi-Model Generator High Integration Tools. *MOB Technical Reports.* Delft: TU Delft.
LABAN, M., ARENDSEN, P., ROUWHORST, W. F. J. A. & VANKAN, W. J. 2002. A Computational Design Engine for Multidisciplinary Optimisation with Application to a Blended Wing Body Configuration. *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimisation.* Atlanta, GA: AIAA.
LANGEN, T. H. M. 2011. *Development of a Conceptual Design Tool for Conventional and Boxwing Aircraft.* MSc Thesis, TU Delft.
LASSILA, O. 1990. Frames or Objects, or Both? *In:* AAAI (ed.) *8th AAAI National Conference on Artificial Intelligence. Object-Oriented Programming in AI.* Boston, MA.
LIEBECK, R. H. 2004. Design of the Blended Wing Body Subsonic Transport. *Journal of Aircraft,* 41-1.
LIEBECK, R. H., PAGE, M. A. & RAWDON, B. K. 1998. Blended-Wing-Body Subsonic Commercial Transport. *36th Aerospace Sciences Meeting and Exhibit.* Reno, Nevada: AIAA.
LISANDRIN, P. 2007. *Elements of Automated Aeroelastic Analysis in Aircraft Preliminary Design.* PhD Dissertation, TU Delft.
LOVETT, J., INGRAM, A. & BANCROFT, C. N. 2000. Knowledge Based Engineering for SMEs - a methodology. *Journal of Materials Processing Technology,* 107 384-389.
MALONE, B. 2002. On the financial impact of MDO on the corporation. *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimisation.* Atlanta, GA: AIAA.
MCBRIAR, I., SMITH, C., BAIN, G., UNSWORTH, P., MAGRAW, S. & GORDON, J. L. 2003. Risk, gap and strength: key concepts in knowledge management. *Knowledge-Base Systems,* 16-1, 29-36.
MCMASTERS, J. H. & CUMMINGS, R. M. 2004. From Farther, Faster, Higher to Leaner, Meaner, Greener: Further Directions in Aeronautics. *Journal of Aircraft,* 41-1.

MEIJER, P. B. 2003. *Parametric modelling of an Airbus aircraft family for dynamic response simulations.* MSc, TU Delft.
MILTON, N. 2007. *Knowledge Acquisition in Practice: A Step-by-step Guide,* London, Springer.
MILTON, N. 2008. *Knowledge Technologies,* Monza, IT, Polimetrica.
MILTON, N. & LA ROCCA, G. 2008. KBE Systems. *In:* SICA (ed.) *Knowledge Technologies.* Monza, IT: Polimetrica.
MINSKY, M. 1975. *The Psychology of Computer Vision*, McGraw-Hill.
MOHAGHEGH, M. 2004. Evolution of Structures Design Philosophy and Criteria. *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference.* Palm Springs, California: AIAA.
MOKA Modelling Language Core Definition. 2000. *MOKA Technical Report Deliverable D.1.3.*
MORRIS, A. J. 2002. MOB, A European Distributed Multi-Disciplinary Design and Optimisation Project. *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimisation.* Atlanta, Georgia: AIAA.
MORRIS, A. J., LA ROCCA, G., ARENDSEN, P., LABAN, M., VOSS, R. & HÖNLINGER, H. 2004. MOB - A European Project on Multidisciplinary Design Optimisation. *24th ICAS Congress.* Yokohama, Japan: ICAS.
MOUSAVI, A., CASTONGUAY, P. & NADARAJAH, S. K. 2007. Survey of shape parameterization techniques and its effect on three dimensional aerodynamic shape optimization. *18th AIAA Computational Fluid Dynamics Conference.* Miami, F: AIAA.
NASA 2002. NASA Aeronautic Blueprint: towards a bold new era in aviation. Washington, D.C.
NAWIJN, M., VAN TOOREN, M. J. L., ARENDSEN, P. & BERENDS, J. P. T. J. 2006. Automated Finite Element Analysis in a Knowledge Based Engineering Environment. *44th AIAA Aerospace Sciences Meeting and Exhibit.* Reno, Nevada: AIAA.
NEGNEVITSKY, M. 2005. *Artificial Intelligence - A Guide to Intelligent System,* Harlow, England, Addison-Wesley.
NISBETT, R. E. 2005. *The Geography of Thought. How Asian and Westerners think differently - and why,* London, UK, Nicholas Brealey Publishing.
NONAKA, I. & TAKEUCHI, H. 1998. The Knowledge-Creating Company. *Harvard Business Review on Knowledge Management.* Boston, MA: Harvard Business School Press.
OBJECT MANAGEMENT GROUP. 2005. Request For Proposal: KBE Services for PLM. Available: http://www.omg.org/docs/dtc/05-09-11.pdf.
OBJECT MANAGEMENT GROUP. 2005 KBE services for PLM. Report dtc/2005-09-11. Available: http://www.omg.org/.
OLDHAM, K., KNEEBONE, S., CALLOT, M., MURTON, A. & BRIMBLE, R. Moka - A methodology and tools Oriented to Knowledge-based engineering applications. Conference on Integration in Manufacturing, 1998 Göteborg, Sweden.
PATTON, S. 2006. Beating the Boomer Brain Drain Blues. *CIO Magazine.*
PEARSON, D. 2001. MOB Structural Model Generation by PATRAN Session file. *MOB Technical Reports.* Warton, Preston, UK: BAe System.
PHILLIPS, R. E. 1997. Dynamic Objects for Engineering Automation. *Coomunication of the ACM,* 40-5.
QIN, N., VAVALLE, A., LE MOIGNE, A., LABAN, M., HACKETT, K. & WEINERFELT, P. 2002. Aerodynamic Studies for Blended Wing Body Aircraft. *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimisation.* Atlanta, GA: AIAA.
QUINN, J. B. 1998. Managing professional intellect: making the most of the best. *Harvard Business Review on Knowledge Management.* Boston, MA: Harvard Business School Press.
RAJ, P. 1998. Aircraft design in the 21st century: implications for design methods. *29th AIAA Fluid Dynamics Conference.* Albuquerque: AIAA.
RAYMER, D. P. 2006. *Aircraft Design: A Conceptual Approach,* Washington D.C.
RENTEMA, D. W. E., JANSEN, F. W. & TORENBEEK, E. 1998. The application of AI and geometric modelling techniques in conceptual aircraft design. *In:* AIAA (ed.) *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.* St. Louis, MO: AIAA.

RHEM, A. J. 2006. *UML for Developing Knowledge Management Systems,* Boca Raton, FL, Auerbach Publications.
RONDEAU, D. L., PECK, E. A., WILLIAMS, A. F., ALLWRIGHT, S. E. & SHIELDS, L. D. 1996. Generative design and optimisation of the primary structure for a commercial transport aircraft wing. *6th NASA and ISSMO Symposium on Multidisciplinary Analysis and Optimization.* Bellevue, WA: AIAA.
RONDEAU, D. L. & SOUMILAS, K. The Primary Structure of Commercial Transport Aircraft Wings: Rapid Generation of Finite Element Models Using Knowledge-Based Methods. *In:* MSC, ed. proceedings of the 1999 Aerospace Users' Conference, 1999. MacNeal-Schwendler Corporation.
ROSCH, E. 1978. Principles of Categorization. *In:* EDS, B. (ed.) *Cognition and Categorization.* Hillsdale, New Jersey.
Rosenfeld's profile [Online]. Available: http://www.glengaryllc.com/all-team-profiles/Larry-Rosenfeld.aspx [Accessed February 2011].
RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F. & LORENSEN, W. 1991. *Object-Oriented modeling and design,* Englewood Cliff, New Jersey, Prentice-Hall,.
SAINTER, P., OLDHAM, K. & LARKIN, A. 2000a. Achieving benefits from Knowledge-Based Engineering systems in the longer term as well as in the short term. *International Conference on Concurrent enterprising.* Toulouse.
SAINTER, P., OLDHAM, K., LARKIN, A., MURTON, A. & BRIMBLE, R. 2000b. Product knowledge management within knowledge-based engineering systems. *ASME Design Engineering Technical Conference.* Baltimore.
SAMAREH, J., A. 2004. Aerodynamic shape optimization based on free-form deformation. *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization conference.* Albany, New York: AIAA.
SAMAREH, J. A. 2001a. Novel Multidisciplinary Shape Parameterization Approach. *AIAA Journal,* 38-6.
SAMAREH, J. A. 2001b. Survey of Shape Parameterization Techniques for High-Fidelity Multidisciplinary Shape Optimization. *AIAA Journal,* 39-5.
SCHRAGE, D., BELTRACCHI, T., BERKE, L., DODD, A., NIEDLING, L. & SOBIESKI, J. 1991. AIAA Technical Committee on Multidisciplinary Design Optimization - White Paper on Current State of the Art Available: http://endo.sandia.gov/AIAA_MDOTC/sponsored/aiaa_paper.html#appendixI.
SCHUT, J. E. & VAN TOOREN, M. J. L. 2007. Design "Feasilization" Using Knowledge-Based Engineering and Optimization Techniques. *Journal of Aircraft,* 44-6.
SEIBEL, P. 2005. *Practical Common Lisp*, Apress.
SHMULLER, J. 2004a. *UML,* Indianapolis, Indiana, US, SAMS.
SHMULLER, J. 2004b. Understanding Object Orientation. *UML.* 3rd Edition ed. Indianapolis, Indiana, US: SAMS
SHORTLIFFE, E. H. 1976. *MYCIN: Computer-based Medical Consultations,* New York, Elsevier Press.
SHREIBER, G., AKKERMANS, H., ANJEWIERDEN, A., DE HOOG, R., SHADBOLT, N., VAN DE VELDE, W. & WIELINGA, B. 2000. *Knowledge Engineering and Management: The CommonKADS Methodology,* Cambridge, MA, MIT Press.
SMITH, I. F. C. 2007. Engineering design support challenges. *AI EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 21-1.
SOPRANOS, K. 2005. The age of experience.Boeing is curbing brain drain in a new experiment, recruiting retired engineers for urgent and complex technical projects. *Boeing Frontiers on Line.*
STAUBACH, J. B. 2003. Multidisciplinary Design Optimisation, MDO, the Next Frontier of CAD/CAE in the Design of Aircraft Propulsion Systems. *AIAA/ICAS International Air and Space Symposium and Exposition.* Dayton, OH: AIAA.
STETTNER, M. & VOSS, R. 2002 Aeroelastic, Flight Mechanic, and Handling Qualities of the MOB BWB Configuration. *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.* Atlanta, Georgia: AIAA.

STOKES, M. 2001. *Managing Engineering Knowledge - MOKA: Methodology for Knowledge Based Engineering Applications,* London and Bury St Edmunds, UK, Professional Engineering Publishing.

STRAATHOF, M. H., TOOREN, M. J. L. V., VOSKUIJL, M. & KOREN, B. 2008. Aerodynamic shape parameterisation and optimisation of novel configurations. *The Aerodynamics of Novel Configurations Conference.* London (UK): Royal Aeronautical Society.

SUBEL, C. 2002. A New Modeling Technique for Numerical Analysis and Multidisciplinary Optimization. *In:* AIAA (ed.) *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.* Atlanta, Georgia: AIAA.

SULLY, P. 1993. *Modelling the world with objects,* Englewood Cliff, New Jersey, Prentice-Hall.

TAM, W. F. 2004. Improvement Opportunities for Aerospace Design Process. *In:* AIAA (ed.) *Space 2004 Conference and Exhibit.* San Diego, CA: AIAA.

TOMIYAMA, T. 2007. Intelligent computer-aided design systems: Past 20 years and future 20 years. *AI EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 21-1.

TOMIYAMA, T. & YOSHIKAWA, H. Knowledge Engineering and CAD - An approach to intelligent CAD for machine design. Proceedings of the Graphics and CAD symposium, 1983 Japan. Information Processing Society of Japan, 223-230.

TOMIYAMA, T. & YOSHIKAWA, H. 1985. Knowledge Engineering and CAD. *FGCS Future Generation Computer System,* 1-4.

TORENBEEK, E. 1982. *Synthesis of subsonic airplane design,* Delft, Springer.

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN. last visited November 2009. *UIUC Airfoil Database* [Online]. Urbana, IL. Available: http://www.ae.illinois.edu/m-selig/ads/coord_database.html.

VAN DEN BERG, T. 2009. *Parametric Modeling and Aerodynamic Analysis of Multi-Element Wing Configurations.* MSc thesis, TU Delft.

VAN DEN BRANDEN, M. 2004. *Development of a design and engineering engine for wing preliminary design optimization.* MSc, TU Delft.

VAN DER ELST, S. & VAN TOOREN, M. J. L. 2008. Domain Specific Modeling Languages to Support Model-Driven Engineering of Aircraft Systems *In:* AIAA (ed.) *The 26th Congress of ICAS and 8th AIAA ATIO.* Anchorage, Alaska: AIAA.

VAN DER LAAN, A. 2008. *Knowledge Based Engineering Support for Aircraft Component Design.* PhD Dissertation, TU Delft.

VAN DER LAAN, A. & VAN TOOREN, M. J. L. 2005. Parametric Modeling of Movables for Structural Analysis. *Journal of Aircraft,* 42-6, 1605-1613.

VAN DIJK, R. E. C. 2008. *A Knowledge Based Engineering Approach to Aircraft Movable Design.* MSc, TU Delft.

VAN DIJK, R. E. C., D'IPPOLITO, R., TOSI, G. & LA ROCCA, G. 2011. Multidisciplinary Design and Optimization of a Plastic Injection Mold Using an Integrated Design and Engineering Environment. *NAFEMS World Congress.* Boston.

VAN HOEK, M. 2010. *Structural Design, Analysis and Optimization of a Lifting Surface in a Knowledge Based Engineering Environment.* MSc, TU Delft.

VAN HOUTEN, M. H., LA ROCCA, G., VAN DER LAAN, A., ARENDSEN, P., LABAN, M., VAN TOOREN, M. J. L. & NAWIJN, M. 2005. PARMOD - Parametric Modeling. *In:* NLR (ed.) *Technical Report.* Amsterdam: NLR-TUD.

VAN STAVEREN, W. H. J. J. 2003. *Analyses of Aircraft Responses to Atmospheric Turbulence*, Delft University of Technology - PhD Dissertation.

VAN TOOREN, M. J. L. 2003. Sustainable Knowledge Growth. *TU Delft Inaugural Speech* Faculty of Aerospace Engineering ed. Delft: Faculty of Aerospace Engineering.

VANDENBRANDE, J., H., GRANDINE, T., A. & HOGAN, T. 2006. The search of the perfect body: Shape control for multidisciplinary design optimization. *44th AIAA Aerospace Science Meeting and Exhibit.* Reno, Nevada: AIAA.

VANKAN, W. J. & LABAN, M. 2002. A SPINEWARE Based Computational Design Engine for Integrated Multidisciplinary Aircraft Design. *In:* AIAA (ed.) *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimisation.* Atlanta, Georgia.

VOSKUIJL, M., LA ROCCA, G. & DIRCKEN, F. 2008. Controllability of blended wing body aircraft. *ICAS 2008.* Anchorage, Alaska.

WAKAYAMA, S. & KROO, I. 1998. The Challenge and Promise of Blended-Wing-Body Optimization. *In:* AIAA (ed.) *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.* St. Louis, MO: AIAA.

WERNER-WESTPHAL, C., HEINZE, W. & HORST, P. 2007. Multidisciplinary Integrated Preliminary Design Applied to Future Green Aircraft Configuration. *In:* AIAA (ed.) *45th AIAA Aerospace Sciences Meeting and Exhibit.* Reno, Nevada: AIAA.

WERNER-WESTPHAL, C., HEINZE, W. & HORST, P. 2008. Multidisciplinary Integrated Preliminary Design Applied to Unconventional Aircraft Configurations. *Journal of Aircraft,* 45-2.

WHITNEY, D. E., DONG, Q., JUDSON, J. & MASCOLI, G. 1999. Introducing Knowledge-based engineering into an interconnected product development process. *ASME Design Engineering Technical conference.* Las Vegas, Nevada.

ZANG, T. A. & SAMAREH, J. A. 2001. The Role of geometry in the multidisciplinary design of aerospace vehicles. *In:* SIAM (ed.) *SIAM Conference on Geometric Design.* Sacramento, CA: SIAM.

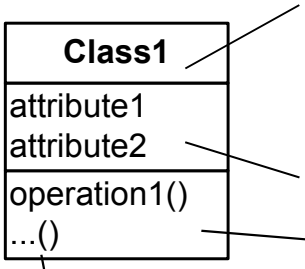
ZONA TECHNOLOGY, I. 2009. ZAERO. Scottsdale, AZ

ZWEBER, J. & HARTONG, A. 1998. Structural and control surface design for wings using the adaptive modeling language *In:* AIAA (ed.) *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.* St. Louis, MO: AIAA.

APPENDICES A-M

Appendix A Summary of basic UML notation

Class and Object diagrams

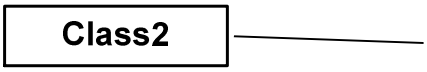


Class name: By convention, a single capitalized word (in case of multiple words names, all the words are joined together and capitalized)

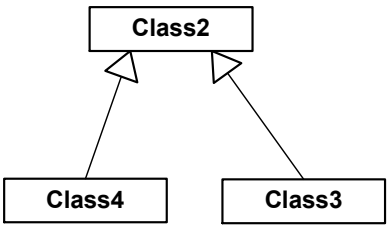
Attributes: By convention, all attributes are indicated with a single low case word (in case of multiple words names, all the words are joined together and capitalized starting from the second word)

Operations: By convention, all operations are indicated with a single low case word followed by "()". In case of multiple words names, all the words are joined together and capitalized starting from the second word)

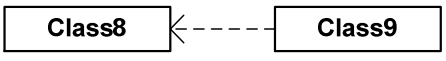
Ellipsis: three dots (…) are used in the attribute and/or operation fields to indicate that only some of the attributes or operations have been indicated.



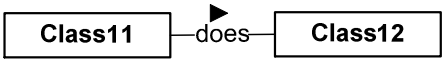
An **elided class** representation is one where one or both the attribute and operation fields are not shown for simplification.



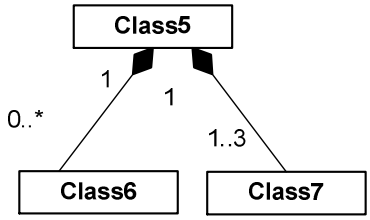
Generalization/Specialization link: Class4 and Class3 are two specializations of Class2 (they inherit attributes and operation from Class2). Class2 is a generalization of both Class4 and Class3.



Dependency link: Class9 depends on Class8. E.g., Class9 needs something from Class8 for the evaluation of its attributes, or to perform some of its operations.

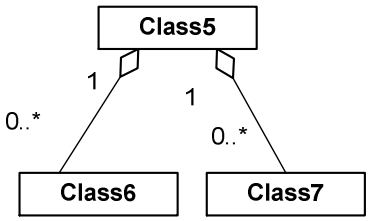


Association link: Class10 and 11 are associated: they are linked by a relationship different than generalization, aggregation, dependency, etc. A tag and an arrow can be used to specify the type and direction of the association



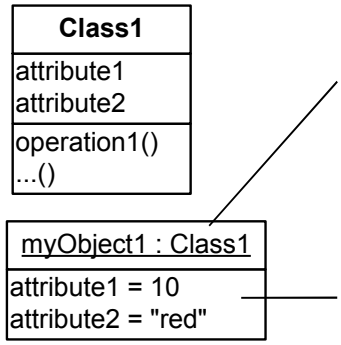
Composition link: Class6 and Class7 are components of Class5.

Multiplicity label: Class 5 is a composition including from 0 to an infinite number [0..*] of Class6 components; and 1 to 3 [1..3] Class7 components. Both Class6 and Class7 belong to only 1 Class5 composition [1].



Aggregation link: Class6 and Class7 are members of the aggregation Class5.

*A **composition** is a stronger relation than **aggregation**: all the members of an aggregation can live autonomously without the need of the aggregation class. On the contrary, if a composition is eliminated, all the members are also eliminated.*

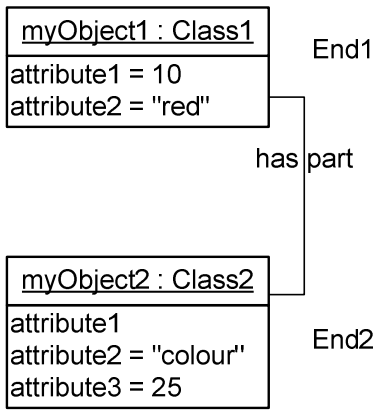


Objects: myObject1 is an instance of Class1.

Generally two fields are used to represent objects. The **class name is always indicated next to the object's name** in the top field. Hence, it is not always necessary to show the class-object link using a connector.

The bottom field is reserved to **list of attributes and their values**

*By convention, **object names** are indicated with a single low case word. In case of multiple words names, all the words are joined together and capitalized starting from the second word.*

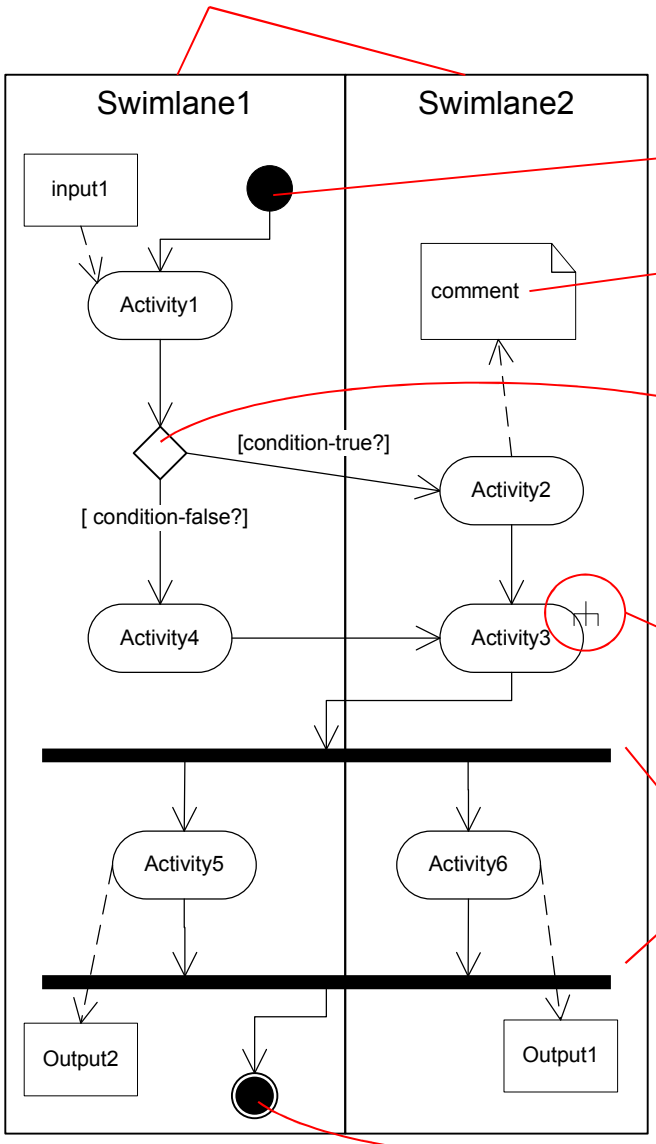


Object links: relationships between objects are indicated by connectors. Connector labels can be used to specify the type of link.

In this example, the link is of type "has part": MyObject2 is part of myObject1

Activity diagrams

Swimlanes separate the activities performed by different actors



Start point of the sequence of activities

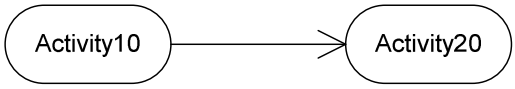
Explanatory note

Decision Point: one branch is selected according to the evaluation of some conditional statement

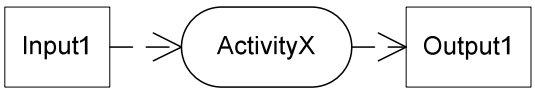
Zoom-in fork: another activity diagram is provided, to show the details of Activity3

Synchronization bars: the following activities cannot start until all the previous activities have been completed

End point of the sequence of activities

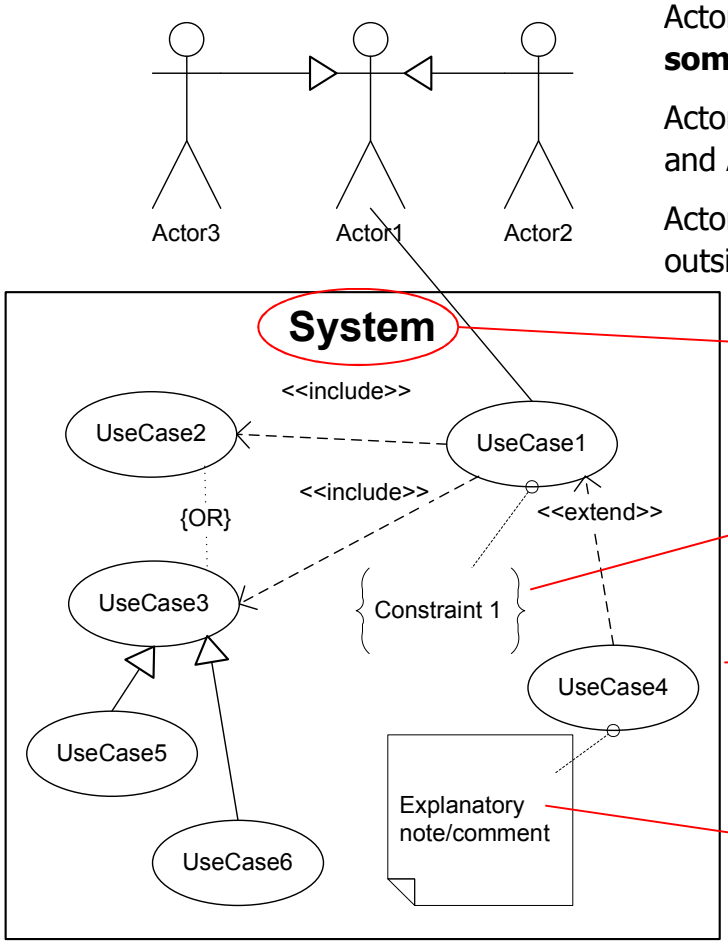


Activity10 is **followed by** Activity20



ActivityX **requires** Input1 and **generates** Output1

Use case diagrams



Actor1 **initiates and/or receives something** from UseCase1.

Actor1 is a generalization of Actor2 and Actor3

Actors are always collocated outside the boundary of a system

System name

Constraint applying to UseCase1

System boundary

Exlanatory note for UseCase4

UseCase5 and UseCase6 are **specializations** of USeCase3 (hence they both inherit from UseCase3)

UseCase1 either **includes** UseCase2 **or** UseCase3. In other words, UseCase1 includes in its definition the steps of either UseCase2 or UseCase3. UseCase4 adds some steps to the existing UseCase1. That is to say, UseCase4 **extends** UseCase1.

Inclusion and extension links are indicated using **dependency links** (i.e., dashed arrows with appropriate tag)

A system can provide several use cases. Different actors can initiate/receive from different use cases.

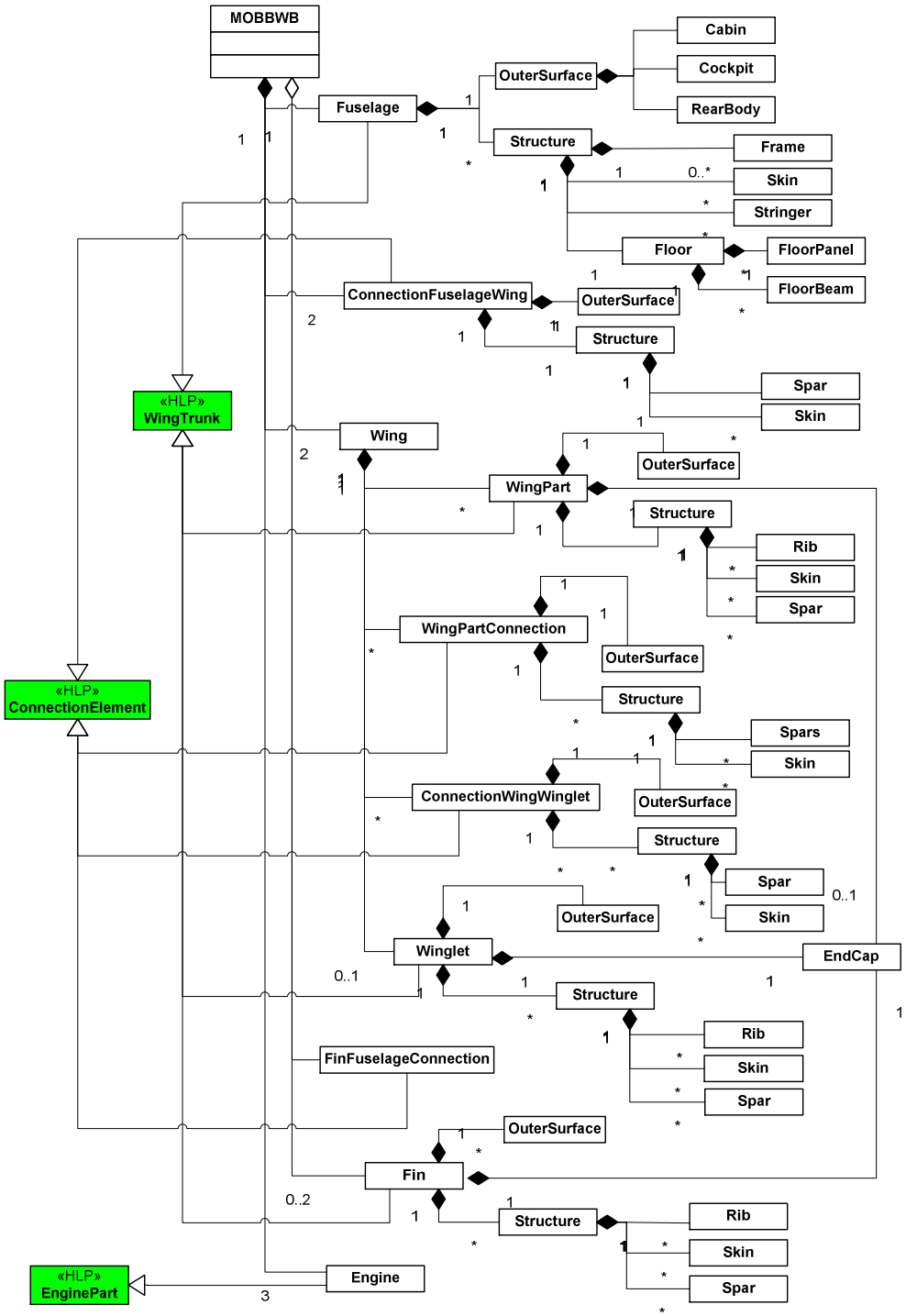
Appendix B Commercial KBE Tools

A non exhaustive list of commercial KBE tools, including some related information and web links, is provided in the table below. Considering the fast dynamic of the KBE tools market, the reader is advised to refer to the KBE page on Wikipedia (http://en.wikipedia.org/wiki/Knowledge-Based_Engineering), which is regularly updated by KBE developers and vendors.

ICAD	By Knowledge Technology International (KTI). Since 2002 part of Dassault Systemes	http://en.wikipedia.org/wiki/ICAD http://www.3ds.com (http://www.ktiworld.com) Former website
ICAD came out in the early 1980's and was the first KBE system on the market. ICAD uses a programming language called IDL (ICAD Design Language), which is an object-oriented, declarative language based on LISP. ICAD provides a proprietary CAD engine for surface modeling, and relies on the Parasolid system to handle solids. After KTI was bought by Dassault Systemes, ICAD has ceased to be supported, though many ICAD applications are still used by companies like Airbus and Boeing		
AML	By Technosoft	http://www.technosoft.com/aml.php
AML is an object-oriented, knowledge-based engineering modeling framework. AML enables multidisciplinary modeling and integration of the entire product and process development cycle. According to Technosoft, no other commercial framework or development environment provides the full range of capabilities that AML includes out of the box.		
INTENT!	By Intent! (since 2005 in Autodesk)	http://www.autodesk.com
INTENT! Used to be a stand alone KBE language, quite similar to ICAD (indeed it was developed by people who worked on the development of ICAD). It used to be LISP based, though it turned out into a proprietary language. At the beginning, INTENT! used AutoCAD as its geometry engine. In 2005 INTENT! has been acquired by Autodesk and it is now integrated in the Inventor Automation Professional package (the design and configuration rules system for Autodesk Inventor)		

GDL	By Genworks International	http://www.genworks.com
GDL is a new generation KBE system that combines the power and flexibility of the older ICAD system with new web technology. It is available for many different platforms such as Windows, Linux and Mac. Its programming language is based on the standard ANSI Common LISP. It allows the manipulation of very simple geometry primitives, and optionally provides full integration to the NURBs Surfaces and Solids modeling kernel from Solid Modeling Solutions Inc.		
CATIA V Knowledgeware	By Dassault Systemes	http://plmus.3ds.com/V5/knowledge.cfm
Knowledgeware is a set of applications available to extend the native functionalities of the CATIA V5 CAD system in terms of design automation and rules capturing. Knowledgeware offers the possibility to define product templates so that automated parametric design is facilitated. Other tools are provided to organize and manipulate parameters, create flexible rules and specification checks. Apart from Knowledgeware, CATIA V5 also offers designers the possibility to write pieces of Visual Basic to further extend the design automation capability.		
Knowledge Fusion	By UGS (since 2007 Siemens PLM software)	http://www.plm.automation.siemens.com/
Knowledge Fusion is an integrated KBE tool that permits knowledge-based extension of the CAD system NX. Knowledge Fusion is the result of a tight integration of the KBE language INTENT! from Intent Engineering Corporation (now part of Autodesk) with the proprietary CAD engine Parasolid. Designers and application developers can work with Knowledge Fusion directly within the NX user environment to create rules that capture design intent. These rules can be used to drive product design, ensuring that engineering and design requirements are fully understood and fully met.		

Appendix C UML class diagram of the MOB Blended Wing Body product model

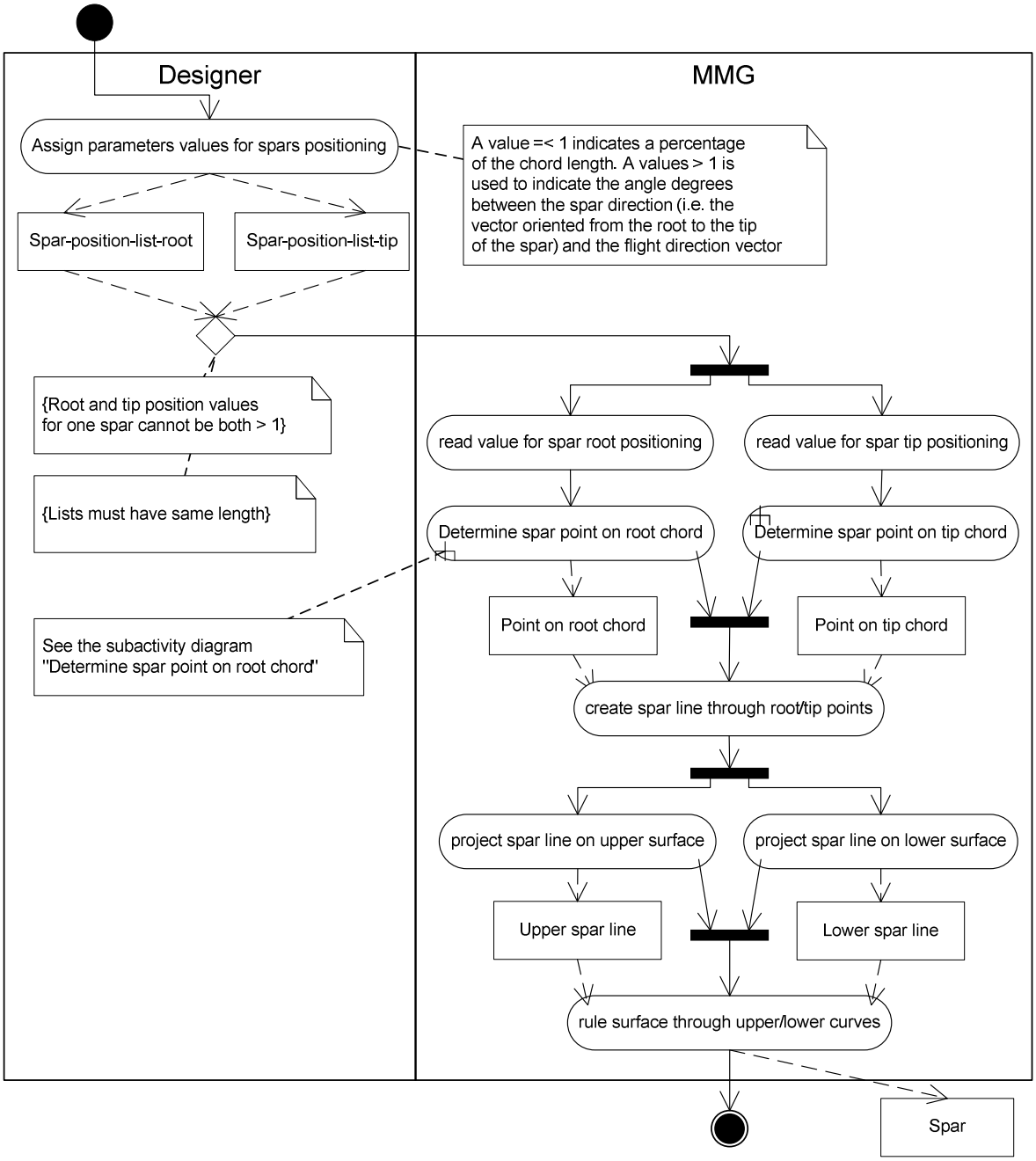


Appendix D MMG input file. Definition of the outer surface of a wing-like element

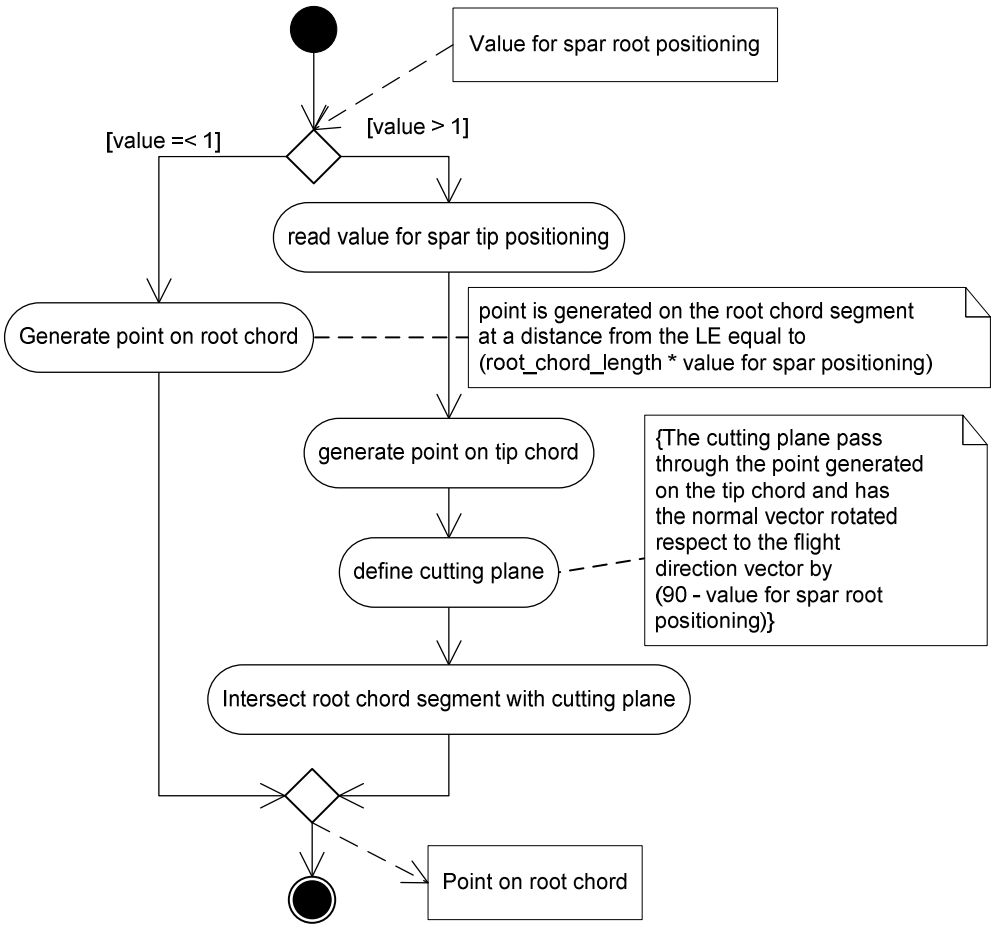
In the insert below a snippet of the MMG input file is provided to show an example of wing surface definition.
In this case, the wing is composed of three wing-parts, as can be seen by the three values contained in the lists. For each wing-part only two airfoils have been selected, one for the root and one for the tip section.
The wing is not twisted, is swept and features a winglet.

```
;;;;------------------ input-data for the MAIN WING ------------------

:span-wing-list        (list 2000 792 2398)
:chord-root-wing-list  (list 10569 2982 1500)
:chord-tip-wing        (list 800)
:sweep-wing-list       (list (degree 7) (degree 29) (degree 29))
:dihedral-wing-list    (list (degree 1) (degree 3) (degree 0))
:twist-wing-list       (list (degree 0) (degree 0) (degree 0))
:twist-angle-wing-root (degree 0)
:winglet? t
:wing-airfoil-list (list (list  "NACA0012"   "NACA0012")
                         (list  "NACA0012"   "NACA0010")
                         (list  "NACA0010"   "NACA0009"))
:wing-airfoil-thickness-list (list  (list 0.90 0.90)
                                    (list 0.90 0.80)
                                    (list 0.80 0.90))
:wing-offset-list (list (list 0.0 1.0)
                        (list 0.0 1.0)
                        (list 0.0 1.0))
```

Appendix E Spars generation process activity diagrams



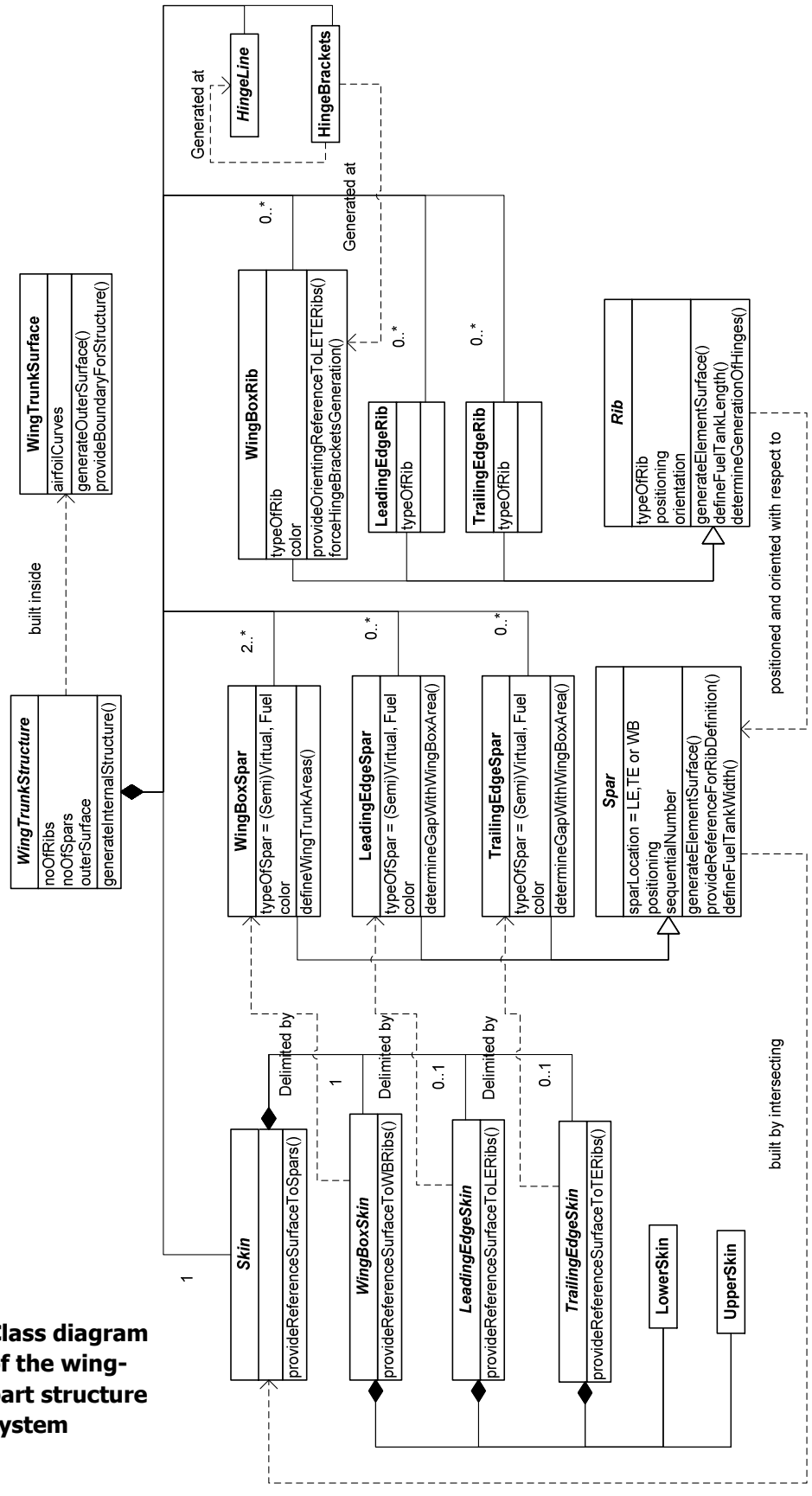
Activity diagram for the spars generation process.



Sub-activity diagrams for the generation of spar-points on root chord.

Appendix F Wing-Part Structure class diagram

(see next page)



Class diagram of the wing-part structure system

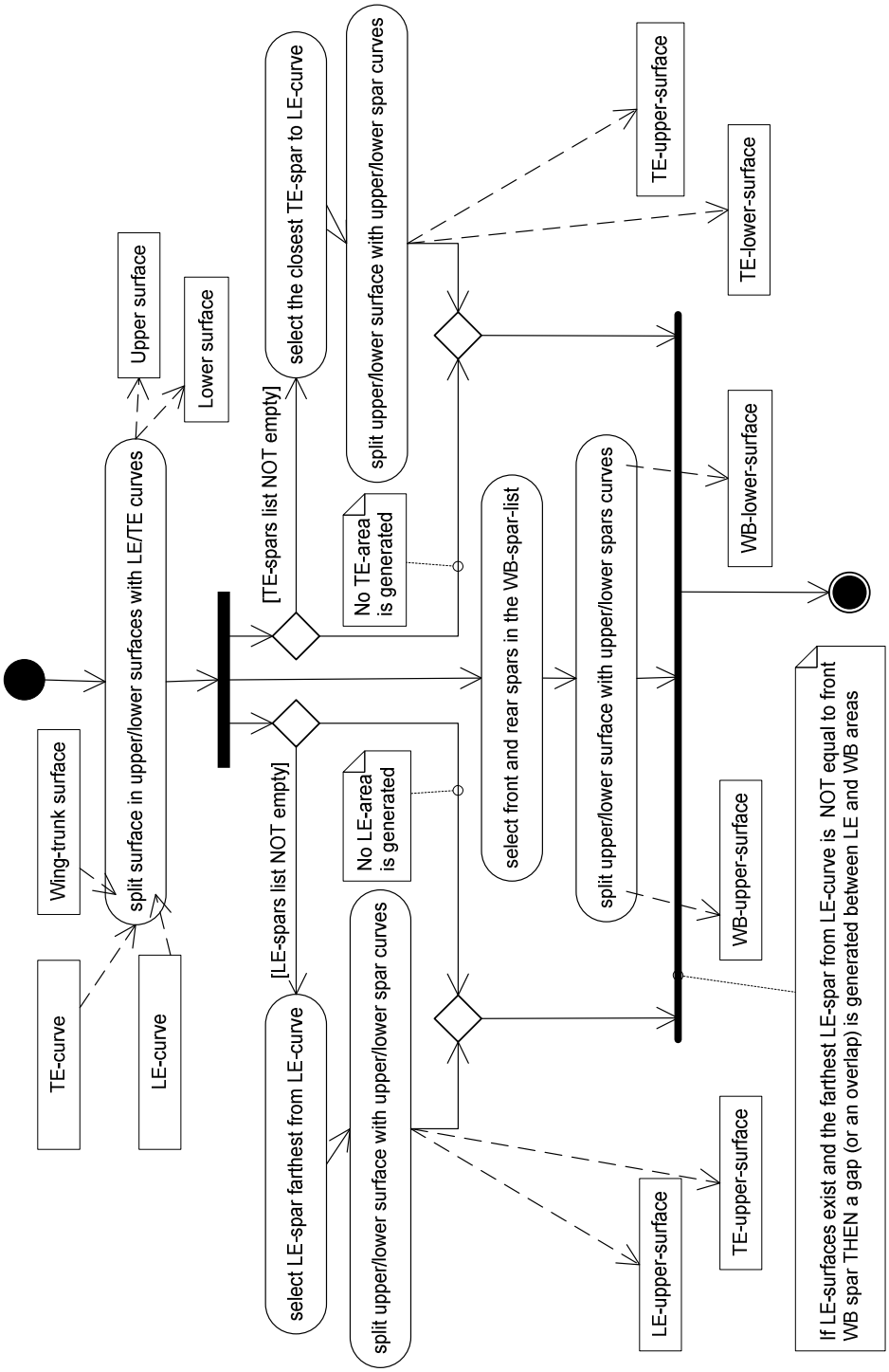
Appendix G MMG input file. Spars definition

A snippet of the MMG input file relative to the vertical tail of a transport aircraft is reported in the insert below, to show examples of spar definitions.
Each parameter is defined as a *list of lists*, being these *lists* as many as the wing-parts used to build up the vertical tail. In this case two wing-parts have been used.
No real spars are present in the LE/TE area (just virtual spars to make sure the LE and TE areas structure is generated).
Three spars are defined in both the wing-parts of the Wing Box area. Note the definition of the semi-real spar in the Wing Box area of the second wing-part.
The index number of each spar (which is assigned automatically by the MMG) has been indicated in the insert below, but only for the first wing-part.

```
;;;------------ spars LEADING-EDGE AREA ----------------------
:fin-type-of-spar-le (list (list 'v)
                           (list 'v))
:fin-spar-offset-list-root-le (list (list 0.1652)
                                    (list 0.1652))
:fin-spar-offset-list-tip-le  (list (list 0.1652)
                                    (list 0.1652))

;;;------------ spars WING-BOX AREA --------------------------
:fin-type-of-spar-wb (list (list 'r 'r 'r)
                           (list 'r (list 'r 0. 0.25) 'r))
:fin-spar-offset-list-root-wb (list (list 0.1652 0.3655  0.5708)
                                    (list 0.1652 0.3655  0.5708))
:fin-spar-offset-list-tip-wb  (list (list 0.1652 0.3655  0.5708)
                                    (list 0.1652 0.3655  0.5708))

;;;------------ spars TRAILING-EDGE AREA ---------------------
:fin-type-of-spar-te (list (list 'v)
                           (list 'v))
:fin-spar-offset-list-root-te (list (list 0.5708)
                                    (list 0.5708))
:fin-spar-offset-list-tip-te  (list (list 0.5708)
                                    (list 0.5708))
```

Spar 0
Spar 1
Spar 2
Spar 3
Spar 4

Appendix H Activity diagram of the LE/WB/TE areas identification process

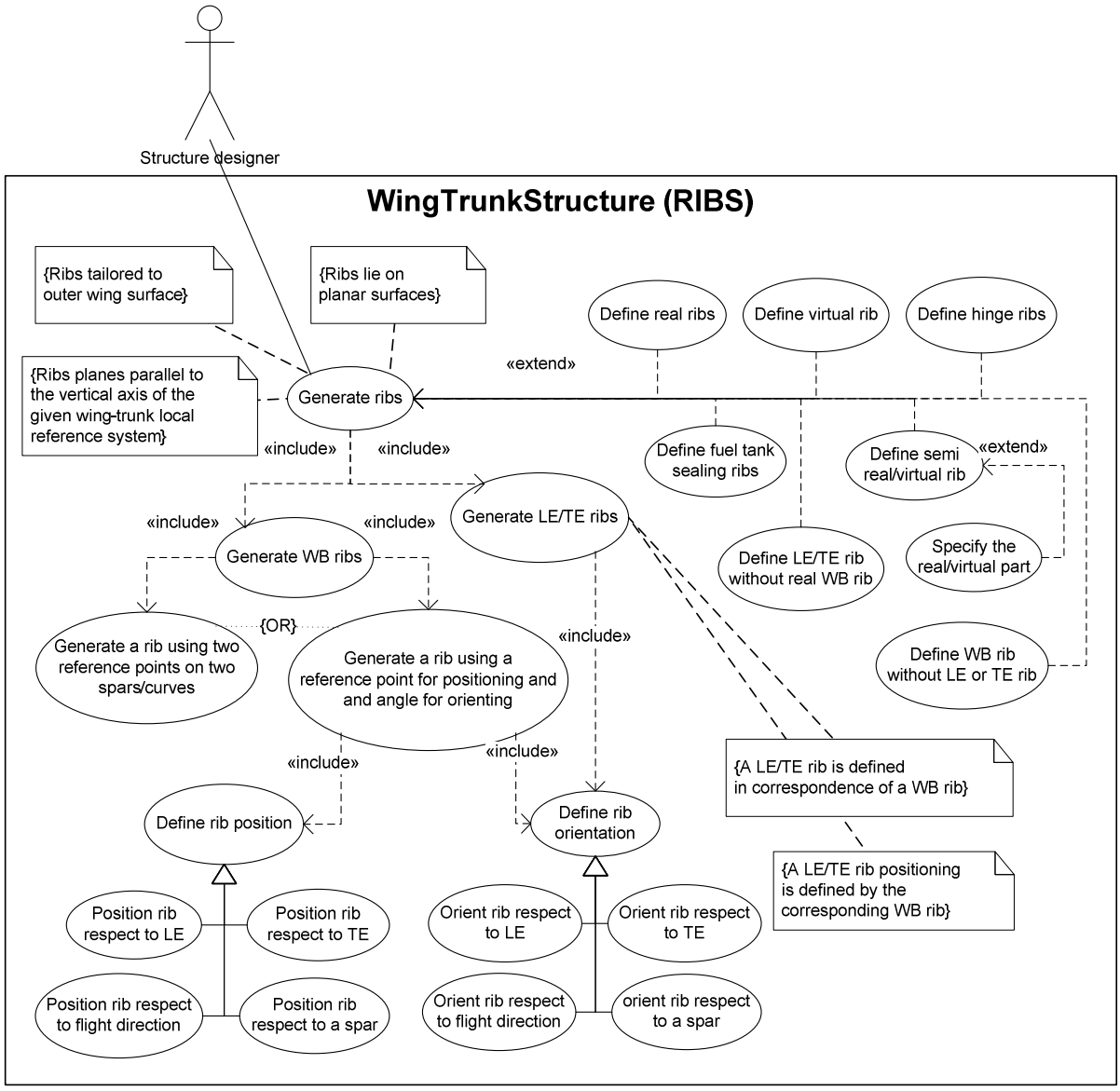


Appendix I MMG input file. Ribs definition

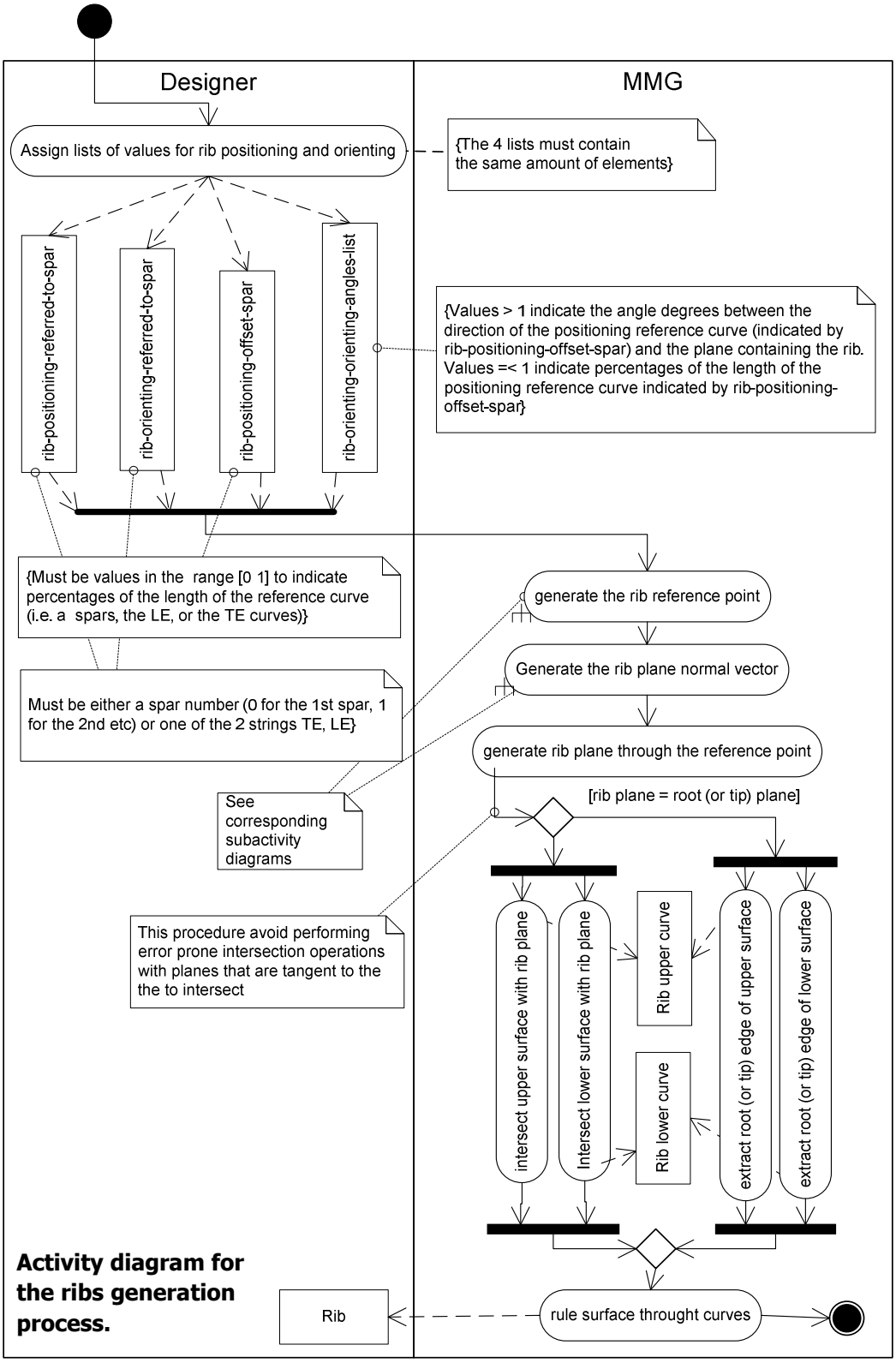
The insert below contains a snippet of the MMG input file containing the definition of the LE/TE/WB ribs in the vertical tail of a transport aircraft. As usual, each parameter is defined as a *list of lists*, being these *lists* as many as the wing-parts used to define the vertical tail. In this case two wing-parts have been used.
Seven and four ribs have been defined in the first and second wing-part respectively. All ribs have been oriented in flight direction; hence the values of the *tail-rib-positioning-referred-to-spar* parameter are ignored.

```
;;;----------------- ribs LEADING-EDGE AREA --------------------------
:tail-type-of-rib-le (list (list  'r  'r  'r  'r  'r  'r  'r)
                           (list  'r  'r  'r  'r))
:tail-rib-le-orienting-referred-to-spar
                     (list (list  'fd  'fd  'fd  'fd  'fd  'fd  'fd)
                           (list  'fd  'fd  'fd  'fd) )
:tail-rib-le-orienting-angles-list (list (list  0  0  0  0  0  0  0)
                                         (list  0  0  0  0) )

;;----------------------- ribs WING-BOX AREA ---------------------
:tail-type-of-rib (list (list  'l  'l  'l  'l  'l  'l  'l)
                        (list  'l  'l  'l  'l) )
:tail-rib-positioning-referred-to-spar (list (list  0  0  0  0  0  0  0)
                                             (list  0  0  0  0) )
:tail-rib-orienting-referred-to-spar
                  (list (list  'fd  'fd  'fd  'fd  'fd  'fd  'fd)
                        (list  'fd  'fd  'fd  'fd) )
:tail-rib-positioning-offset-list
                  (list (list 0.07  0.216  0.35  0.50  0.64  0.78  0.92)
                        (list 0.12  0.37  0.62  0.87))
:tail-rib-orienting-angles-list (list (list  0  0  0  0  0  0  0)
                                      (list  0  0  0  0) )

;;;-------------------- ribs TRAILING-EDGE AREA -----------------
:tail-type-of-rib-te (list (list  'r  'r  'r  'r  'r  'r  'r)
                           (list  'r  'r  'r  'r))
:tail-rib-te-orienting-referred-to-spar
                     (list (list  'fd  'fd  'fd  'fd  'fd  'fd  'fd)
                           (list  'fd  'fd  'fd  'fd) )
:tail-rib-te-orienting-angles-list (list (list  0  0  0  0  0  0  0)
                                         (list  0  0  0  0) )
```

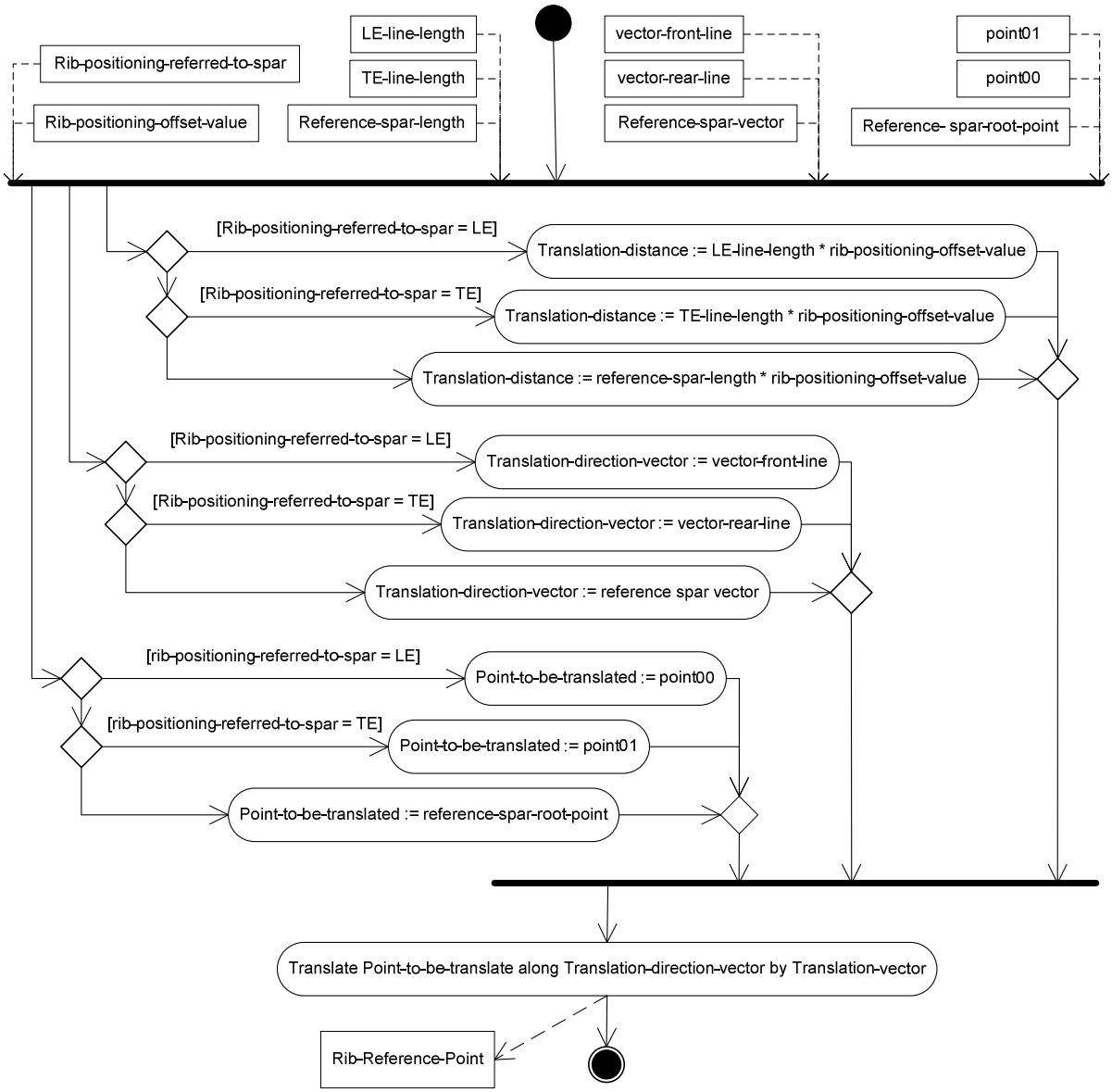
Appendix J UML Use case relative to ribs definition



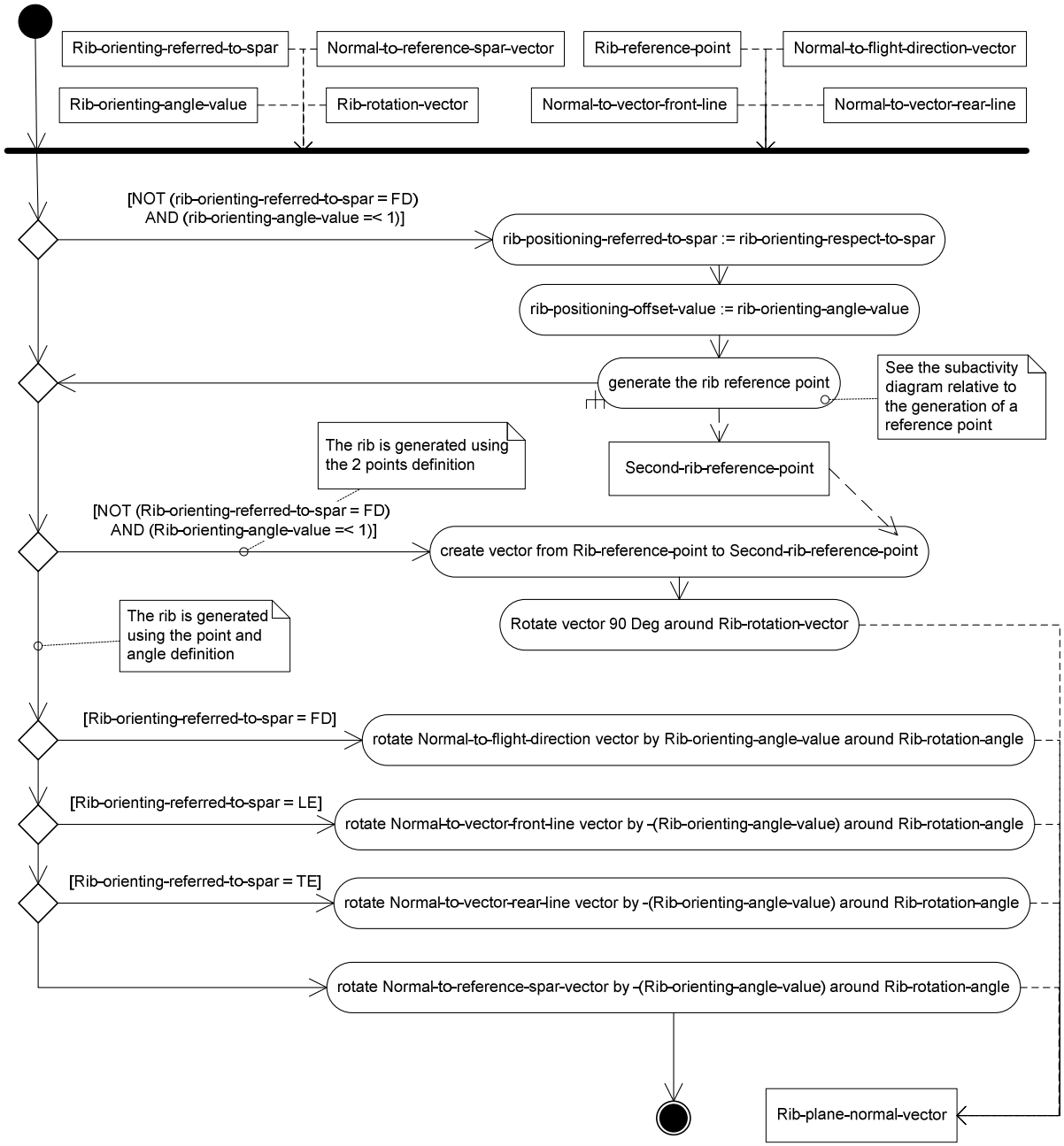
Appendix K Ribs generation process activity diagrams



Activity diagram for the ribs generation process.

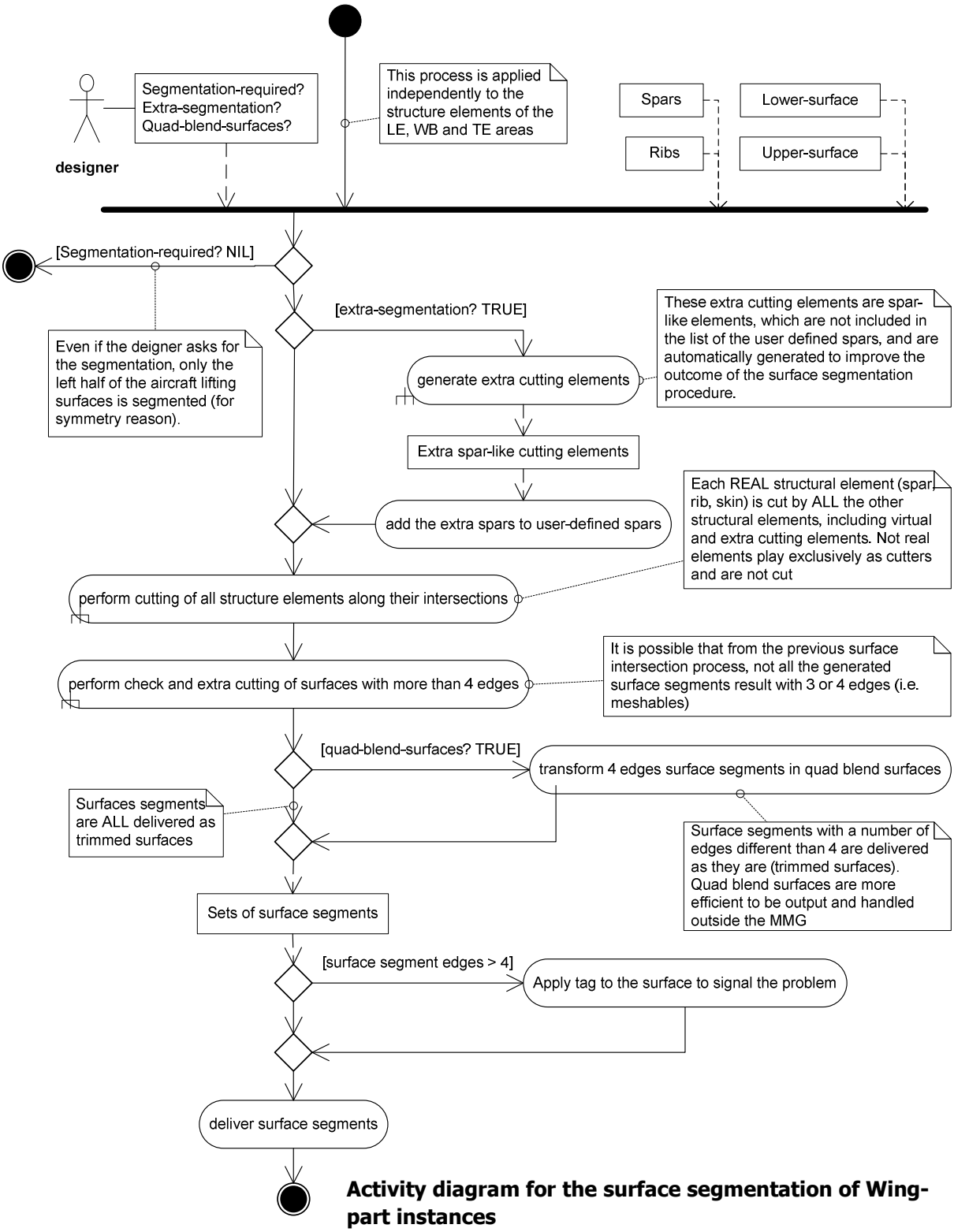


Sub-activity diagram for the generation of *Rib-reference-point*

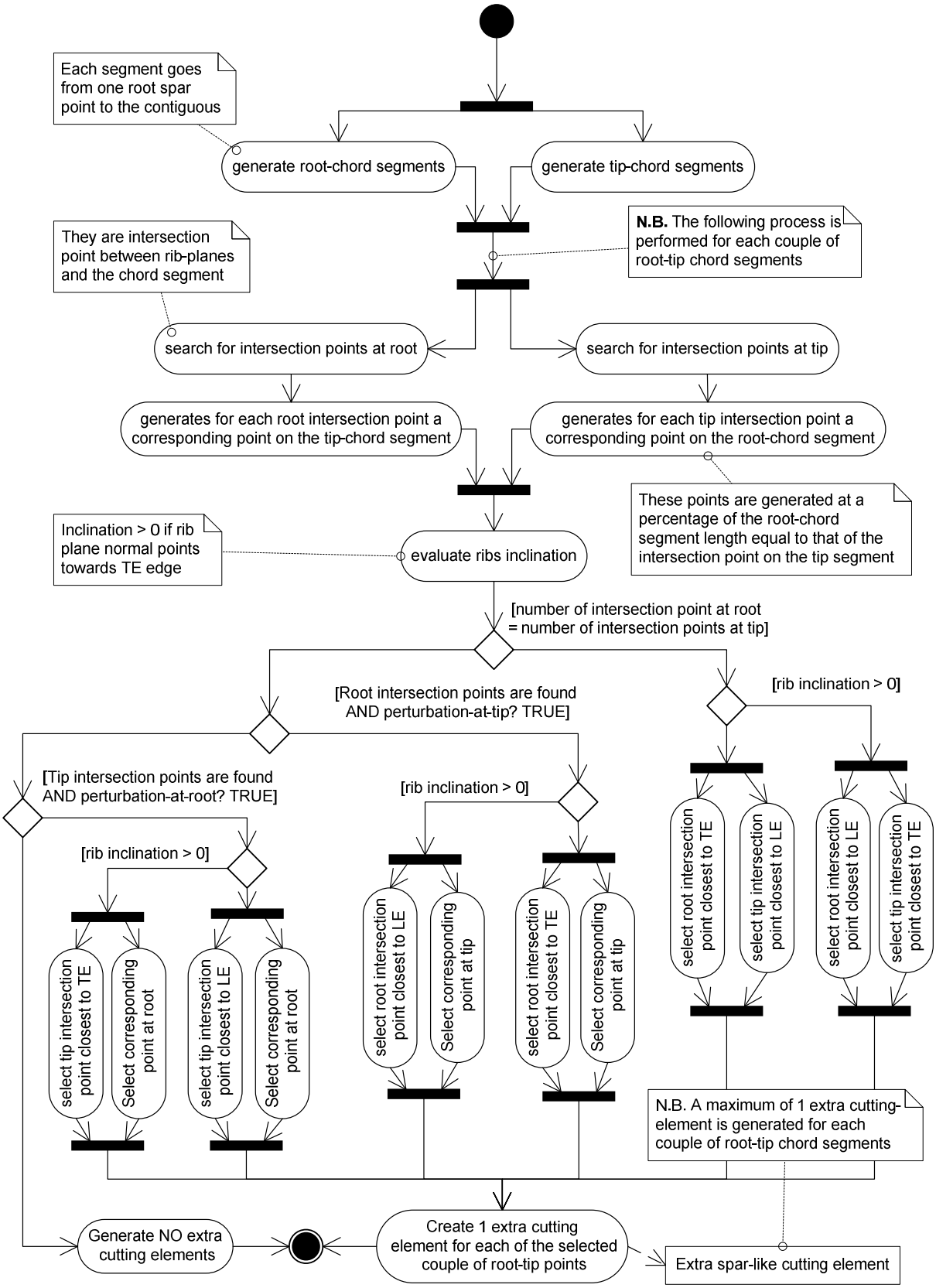


Sub-activity diagram for the generation of *Rib-reference-plane*

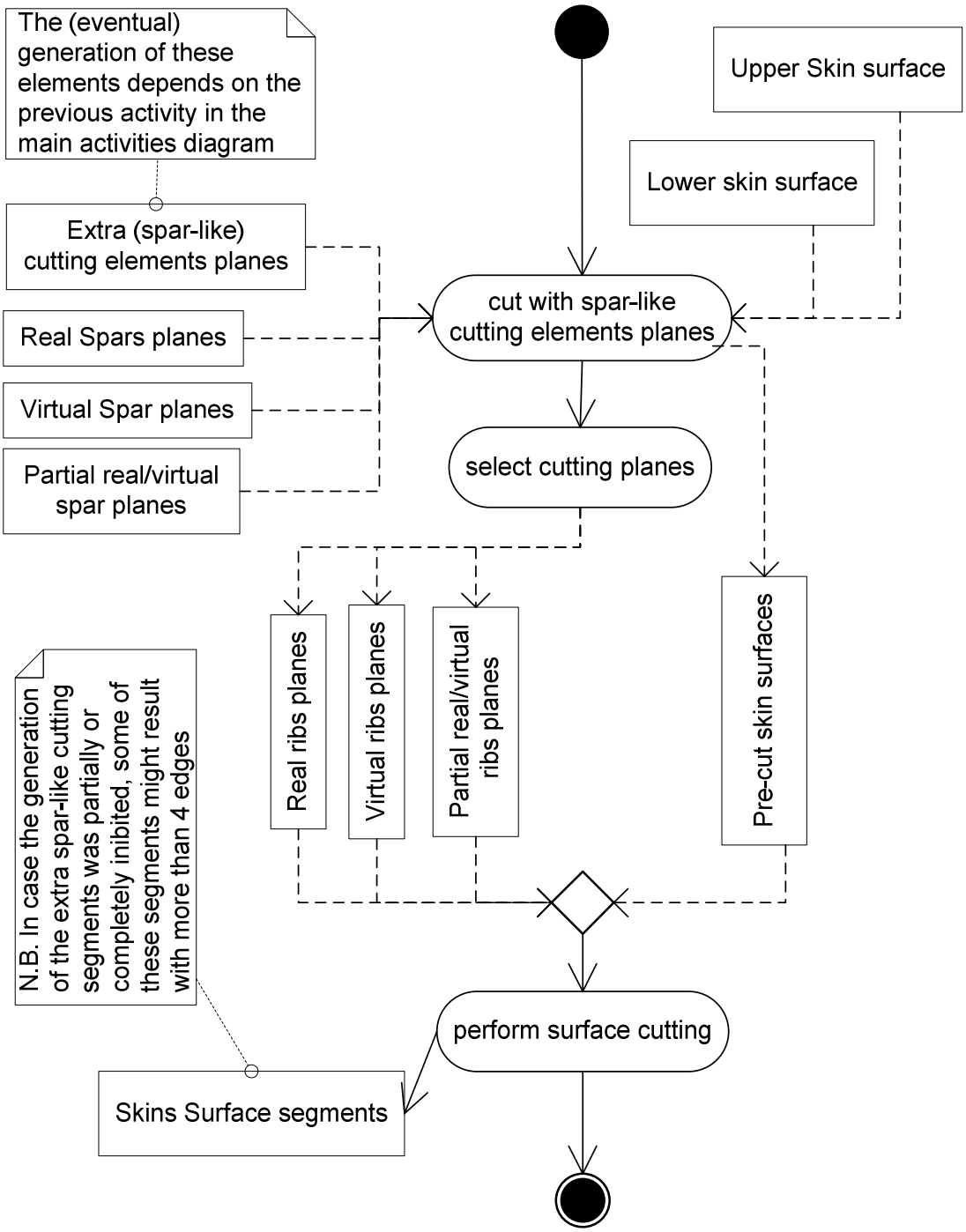
Appendix L Wing-part segmentation process



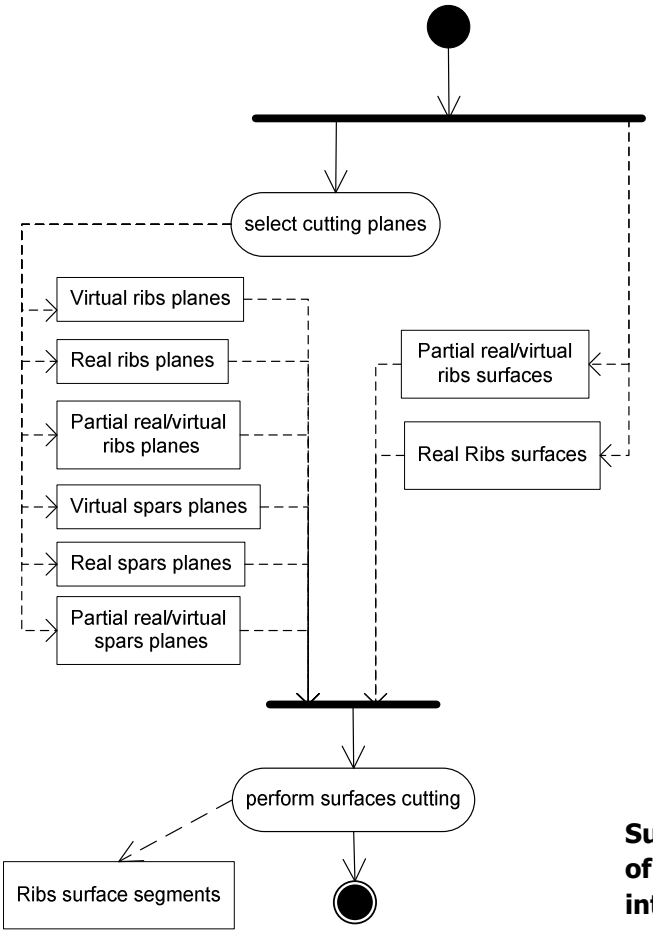
Activity diagram for the surface segmentation of Wing-part instances



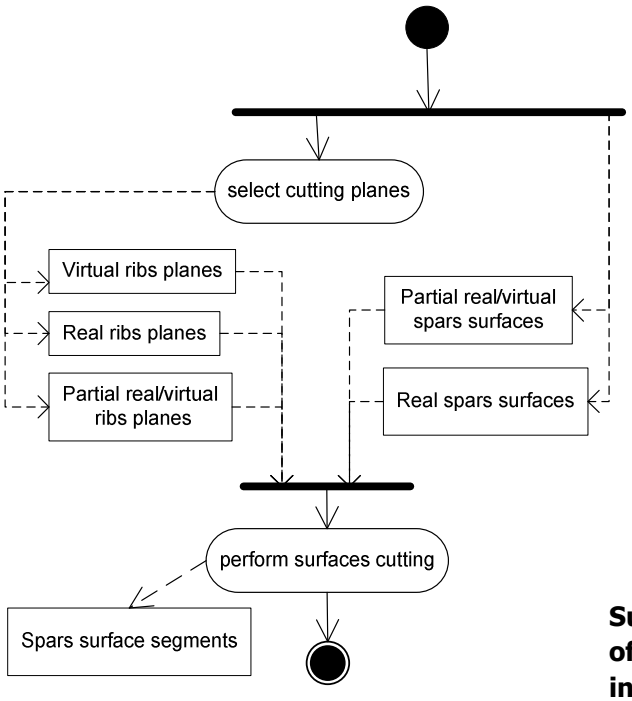
Sub-activity diagram "generate extra cutting elements"



Sub-activity diagram "perform cutting of all structure elements along their intersection". Skin panel segmentation.



Sub-activity diagram "perform cutting of all structure elements along their intersection". Ribs segmentation.



Sub-activity diagram "perform cutting of all structure elements along their intersection". Spars segmentation.

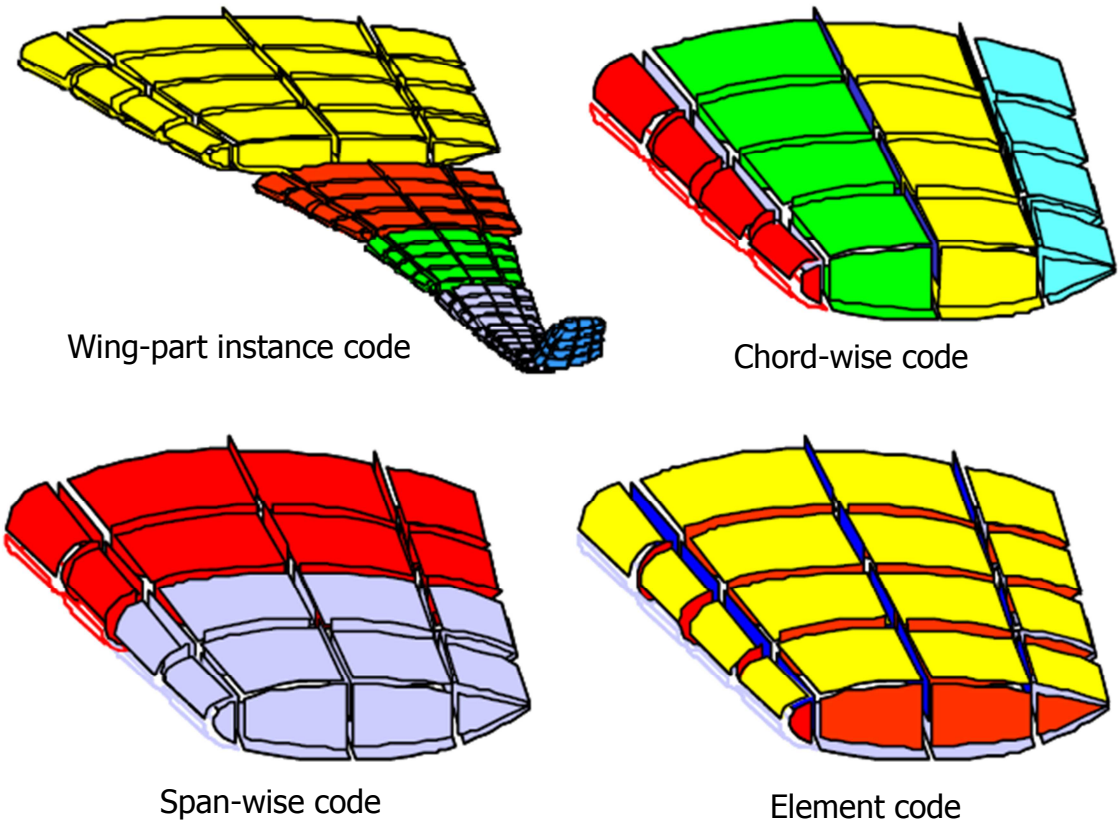
Appendix M Definition of design variable areas for structural FE-based optimization

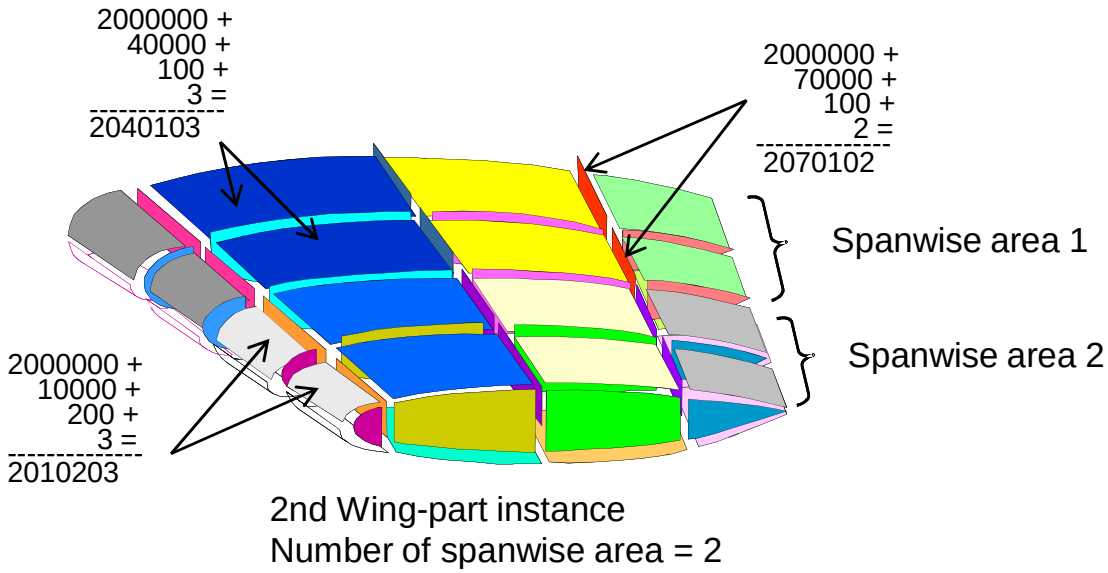
A method has been developed to assign all the structural surface segments generated by the Capability Module Surface-splitter to a number of so called *design areas*. The thickness of all the segments belonging to a certain same area corresponds to one design variable in the structural optimization process. That is to say, all the segments belonging to the same design area will end up with the same thickness value.
The grouping method is based on the automatic generation of design area identification codes, which are attached to each surface segment and then stored in their respective FEM-table. Via a limited amount of settable parameters in the input file, the user has the possibility, to define the amount and the extension of the design areas, hence, to affect the total amount of design variables for the optimization process.

In the MMG developed for the MOB project, the design variable code is calculated by adding 4 different sub-code numbers (see next page for an example)):

1. *Wing-part instance code*: from $1 \cdot 10^6$ to $n \cdot 10^6$ (from root to tip in the given lifting surface)
2. *Chord-wise code*: from $1 \cdot 10^4$ to $99 \cdot 10^4$ ($1 \cdot 10^4$ and $2 \cdot 10^4$ reserved for the LE and TE zone respectively, the other zones are comprised from spar to spar)
3. *Span-wise code*: from $1 \cdot 10^2$ to $99 \cdot 10^2$ (from root to tip in the given wing part, number of zones defined by the user via input file)
4. *Element code*: from 0 till 99. Each code is reserved to a type of element:
   - 1 for ribs and riblets segments
   - 2 for (real) spars segments
   - 3 for upper-skin segments
   - 4 for the lower-skin segments
   - *n* for other segments, with $n \leq 99$

Parameters are also available in the input file to *reduce* the amount of design variables: forcing al the ribs in a chordwise area to get the same variable; forcing all the upper skin panels in the same spanwise area to get the same variable; idem for the lower skin panels. To *increase* the amount of design variable areas, the user can define extra virtual spar or ribs.
The design variable areas identification approach described here, and initially developed for the MOB BWB, has been subsequently extended to conventional aircraft configurations as well.





Coding scheme for design variable areas identification and examples

List of Publications

Journal Papers

1. **La Rocca, G.** and M.J.L. van Tooren, Knowledge-based engineering to support aircraft multidisciplinary design and optimization. Proceedings of the Institution of Mechanical Engineering, Part G: *Journal of Aerospace Engineering*, 2010.
2. **La Rocca, G**. and M.J.L. van Tooren, Knowledge-Based Engineering Approach to Support Aircraft Multidisciplinary Design and Optimization. *Journal of Aircraft*, 2009.
3. **La Rocca, G**. and M.J.L. van Tooren, Enabling distributed multi-disciplinary design of complex products: a knowledge based engineering approach. *Journal of Design Research*, 2007.
4. Vlot, A., E. Kroon, and **G. La Rocca**, Impact Response of Fiber Metal Laminates. Key Engineering Materials, 1998.

Book Sections

1. **La Rocca, G.**, Seaplanes and Amphibians, in *Encyclopedia of Aerospace Engineering* 2010, John Wiley & Sons, Ltd.
2. **La Rocca, G.**, and Nick Milton, KBE Systems, in *Knowledge Technologies*, Sica, Editor 2008, Polimetrica, Monza, IT
3. Torenbeek, E. and **G. La Rocca**, Civil Transport Aircraft, in *Encyclopedia of Aerospace Engineering* 2010, John Wiley & Sons, Ltd.
4. Van Tooren, M.J.L. and **G. La Rocca**, Design Criteria: Resources, Constraints, and Objectives, in *Encyclopedia of Aerospace Engineering* 2010, John Wiley & Sons, Ltd.
5. Van Tooren, M.J.L., **G. La Rocca**, and T. Chiciudean, Further steps towards quantitative conceptual aircraft design, in *Variational Analysis and Aerospace Engineering*, G. Buttazzo and A. Frediani Editors, 2009, Springer.
6. Van Tooren, M.J.L. and **G. La Rocca**, Systems Engineering and Multi-disciplinary Design Optimization, in *Collaborative product and service life cycle management for a sustainable world*, R. Curran, S.Y. Chou, and A. Trappey Editors, 2008, Springer, London.

Conference Papers

1. **La Rocca, G.**, L. Krakers, and M.J.L. van Tooren, Development of an ICAD Generative Model for Blended Wing Body Aircraft Design, in 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimisation, 2002, Atlanta, GA.
2. **La Rocca, G.**, L.A. Krakers, and M.J.L. Van Tooren, Development of an ICAD Generative Model for Aircraft Design, Analysis and Optimisation, in 13th International ICAD User Group IIUG, 2002, Boston, MA.
3. **La Rocca, G.** and M.J.L. Van Tooren, Development of Design and Engineering Engines to Support Multidisciplinary Design and Analysis of Aircraft, in Design Research in the Netherlands, 2005, Eindhoven.

4. **La Rocca, G.** and M.J.L. van Tooren, A modular reconfigurable software tool to support distributed multidisciplinary design and optimisation of complex products, in 16th CIRP International Design Seminar, 2006, Kananaskis, AB, Canada.
5. **La Rocca, G.** and M.J.L. van Tooren, A Knowledge Based Engineering Approach to Support Automatic Generation of FE Models in Aircraft Design, in 45th AIAA Aerospace Sciences Meeting and Exhibit, 2007, Reno, NV.
6. **La Rocca, G.**, A. Vlot, and A. Frediani, Residual strength evaluation of impacted carbon/epoxy sandwich panels with Glare protection, in 13th ICCM Conference, 2001, Beijing.
7. **La Rocca, G.**, A. Vlot, and A. Frediani, Development of a hibrid material for aerospace application, made of C/E sandwich panels with Glare protection; an experimental evaluation of impact behaviour, in 13th ICCM Conference, 2001, Beijing.
8. van Tooren, M.J.L., **G. La Rocca,** L.A. Krakers, and A. Beukers, Design and technology in aerospace. Parametric modeling of complex structure systems including active components, in 13th International Conference on Composite Materials, 2003, S.Diego, CA.
9. Morris, A.J., **G. La Rocca,** P. Arendsen, M. Laban, R. Vos, and Honlinger, MOB - A European Project on Multidisciplinary Design Optimisation, in 24th ICAS Congress, 2004, Yokohama, Japan.
10. Cooper, D.J. and **G. La Rocca**, Knowledge-based techniques for developing engineering applications in the 21st century, in 7th AIAA Aviation Technology, Integration and Operations Conference, 2007, Belfast, Northern Ireland.
11. Voskuijl, M., **G. La Rocca**, and F. Dircken, Controllability of blended wing body aircraft, in ICAS 2008, Anchorage, Alaska.
12. Chiciudean, T., **G. La Rocca**, and M.J.L. Van Tooren, A Knowledge based engineering approach to support automatic design of wind turbine blades, in Design Synthesis, CIRP Design Conference, 2008, University of Twente, NL.
13. van Dijk, R.E.C., R. d'Ippolito, G. Tosi, and **G. La Rocca**, Multidisciplinary Design and Optimization of a Plastic Injection Mold Using an Integrated Design and Engineering Environment, in NAFEMS World Congress, 2011, Boston.

Acknowledgements

Financial support for this research has been partially provided by the European Commission under the GROWTH Programme for the research project *MOB - A Computational Design Engine Incorporating Multi-Disciplinary Design and Optimisation for Blended Wing Body Configuration* (Contract Number G4RD-CT1999-0172) and by the Dutch Technology Foundation STW for the research project *Parametric Modelling and Meshless Discretisation Methods for Knowledge-based Engineering Applications* (Contract Number DLR.6054).

A PhD research work needs much more than bare financial support! Indeed, I am greatly indebted to many people that, in a way or another, helped me to finalize this work and become a better scientist, a better teacher, a better man.

First of all, I have to thank Michel for offering me this great opportunity. I still remember the day I asked whether you had some work for me, given all the free time I had during my fatigue testing in the lab. You came out with something about "an engine" to be developed within a just started European project…An engine? Was propulsion something for me? Well, it took me a couple of days before realizing it was about the computational design engine of the MOB project…. Thanks again for all your energy, inspiration, support and the trust you always had in me during all these years. It was never boring working with you, it was never doing research for the sake of doing research.

Thanks to Lars K., my very first companion in the ICAD and MMG adventure and to Paolo L. and Ton v.d.L., the other major victims of the MMG disease. You know the pain and the satisfaction of making the ICAD thing working! Thanks to Valeria A. and Marco B., my very first personal FEM and CFD experts. The Capability Modules of my MMG still contain some of your genes.

Thanks to all the MOB project mates, in particular to Prof Morris, Armando V., Helen C. and Prof. Ning from Cranfield University, Dave P. from BAE System, Martin S. from EADS, Marco N. and Martin L. from NLR, who supported and encouraged my work with enthusiasm and appreciation. It was a lot of fun working with you and, let's be honest, we did really some good stuff there! I am still waiting for the next MOB to come…

Thanks to all the user committee members of the STW project: S. Allwright from Airbus, T. Ros from Dassault Systemes, F. van Dalen and J. Baan from Fokker, P. Arendsen from NLR, E. Kappel from Thales, B. Knops from MSC Software, A. Konter

from NIMR and L. Goossens from UGS. Your critical comments and suggestions helped me, Marco, Ton and Giampietro to better understand and aim at the real needs of industry.

Steve A., extra thanks for your continuous encouragement since my first IIUG participation and the opportunity you gave me to follow the ICAD and KA courses at Airbus UK.

Thanks to Chiara C., Jochem B. and Joost S., my good TAILORmates and fellow PhD candidates. You were the kind of users that any developer would love to have for testing, verifying and improving his tools.

Thanks to D. Cooper from Genworks International for all the interesting discussions and sharing on the noble art of KBE. Thanks to N. Milton from Epistemics for making me co-author of the KBE systems chapter in your Knowledge Technologies book.

Thanks to all the SIA/DAR/SEAD students who have helped me during the years to develop, extend, test and disseminate the ICAD MMG: Mathieu v.R., Remko S., Peter M., Wietse K., Martijn, v.d.B, Jasper B., Joris G., Frank D. and Reinier v.D. This extends to Durk S., Justin K., Tobie v.d.B. and Maarten v.H., the new generation of GDyeLlers/DARwing-ers. With you guys around, there is big hope for KBE.

Thanks to all my colleagues and friends at SIA/DAR/SEAD (yes, you included!) for keeping our group a pleasant and exciting place to work for all these years, even during the recent turbulent times. In the end, doing a PhD is not just blood and tears… Eating freshly caught salmon in a Alaskan fishing hut, night skiing in Reno, strolling along the Freedom Trail in Boston, walking the Chinese Great Wall, peeking in the final assembly line of the Eurofighter, drinking beer at the Coyote Ugly in Atlanta, admiring the A380 "aerobatics" at Le Bourget, enjoying the Battle of Britain commemoration under the wing of an Avro Vulcan, driving the Icefields Parkway on the way to Jasper, for instance, can be very pleasant side effects!

Special thanks go to my brave paranymphs Meo and Chiacchiera who came all the way to make this day even more special.

Last but not least, my greatest gratitude go to my parents and to Mariska for always being there, always (always?) sure that one day I would have made it!

Well, I made it!

About the Author

Gianfranco La Rocca was born on the 26th December 1970, in Messina, Italy. Until 1989 he attended the Liceo Scientifico Foresi in Portoferraio, Isola d'Elba, where he graduated with a score of 60/60. Afterward, he moved to Pisa where he started his studies in Aerospace Engineering. In 1997, he visited the TU Delft as Erasmus student, and worked on a thesis assignment defined by the universities of Pisa and Delft and Augusta Helicopters. The result was the development of a new hybrid material for aerospace applications and the assessment of its impact behaviour and residual strength. After that he returned to Italy where he first served the Italian Army for his conscript military service and then finalized his Master Degree in Aerospace Engineering with a final grade of 107/110.

In 1999, he returned to the TU Delft, as researcher in the GLARE Research Group. There he worked for two years on the investigation of the hole-to-hole assembly principle, a project commissioned by Fokker Aerostructures, related to the Joint Strike Fighter Demonstrator program.

At the end of 2001, he joined the System Integration Aircraft group with the position of lead researcher for the 5th Framework EC project MOB, on Multidisciplinary Design and Optimisation of Blended Wing Body Aircraft Configurations. There he started working on Knowledge Based Engineering and developed the Blended Wing Body Multi Model Generator, for which, in 2002, he received the *14th KBE Innovation Award*. His KBE application contributed also to the success of the MOB project, which, in 2003, was selected finalist for the *4th edition of the Descartes Prize*, for Outstanding Scientific and Technological Achievements Resulting from European Collaborative Research.

In 2003, he officially became a TU Delft PhD student under the supervision of Prof.dr.ir. van Tooren, and continued his work on the development of KBE applications to support aircraft multidisciplinary design optimization, sponsored by the Dutch Technology Foundation STW.

Since July 2005 he is Assistant Professor at the TU Delft Faculty of Aerospace Engineering, where he teaches Aircraft Design and Advanced Design Methods and pursues research in the field of Knowledge Engineering. In 2008, together with 4 colleagues he became co-founder of KE-Works, a company that offers customers Knowledge Engineering solutions to improve their engineering process.

Gianfranco lives in The Hague with his girlfriend Mariska and Joey the cat.

E quindi uscimmo a riveder le stelle

Dante Alighieri

Finally we could see the stars again. Inferno, Canto XXXIV