DOI: 10.13009/EUCASS2023-097
Aerospace Europe Conference 2023 – 10TH EUCASS – 9TH CEAS

A multidisciplinary modelling system for aircraft structural components

Tobie van den Berg, Ton van der Laan*†, Bas van Manen*, Ioana Ciobotia, Darpan Bansal, Jente Sonneveld*
** GKN Fokker, Papendrecht, 3351LB, the Netherlands*
ton.vanderlaan@fokker.com
† Corresponding Author

Abstract
This paper describes the development of a Multidisciplinary Modelers (MDM) package. This is a python based software package used in trade studies and optimizations of aircraft components. It is capable of modelling wing-like aircraft components such s wings, movables and flaps. It also provides data for and links to analysis tools that determines the performance of such aircraft components. These analyses are for example finite element analyses, stress analysis and manufacturing cost analyses. The MDM enables full automation of the engineering process as is shown in the application cases included in this paper.

1. Introduction

Tier 1 suppliers like GKN Fokker need to have the ability to rapidly respond to new customer requests for aircraft component trade studies and exploit product optimization opportunities **[1]**. New market opportunities such as electrical, hydrogen and Urban Air Mobility (UAM) platforms have recently increased the need for a rapid and agile product development process. To support this, GKN Fokker is working towards a high level of multidisciplinary design and analysis automation and the application of Multidisciplinary Design Optimization (MDO) and Design Space Exploration (DSE) techniques. In [2], the Knowledge Based Engineering (KBE) application MoveableGenerator was introduced: an application for defining and analyzing aircraft moveable structures. This paper presents advancements that have been made since that publication and introduces the larger framework that application is part of: the Multidisciplinary Modelers (MDM) package.
To ensure that design trade studies and optimizations can be supported, all engineering tasks in such efforts must be automated. In the GKN Fokker environment, the MDM forms the heart of design automation tasks. It models the necessary engineering features and generates data required by analysis modules. These analysis modules determine the main characteristics of a design concept. Results of the analysis concepts can be shown in the MDM, strengthening its position within GKN Fokker's automated design efforts. The goal in the automated studies is to achieve a level of confidence that is required for contract binding commercial proposals. This is enabled by using high fidelity models and analyses based on those normally used in the aircraft component certification phase. This paper shows how MDM supports the analyses; the analyses themselves will not be described in detail.

2. The Multidisciplinary Modelers (MDM) package

To cope with the many utilization scenarios for the design and analysis framework, the Multidisciplinary Modelers (MDM) package was developed. Instead of setting up design and analysis studies in case-specific workflows, the model definition and analysis methods are integrated in a KBE application. This is an intelligent executable central product model with rules implemented to ensure the completeness and consistency of generated models. Automatic dependency tracking (see [3]) removes the need to specify the exact sequence of events that need to take place.

The high-level architecture adopted in the MDM package is based on the Multi Model Generator (MMG) concept from [12]. The MMG is a KBE application that provides a designer a central parametric product-modeling environment and is capable of generating disciplinary aspect models. Where La Rocca [12] implemented an MMG for conceptual aircraft design, the concept has been applied to other product concepts as well [13].

The MDM package provides a library of building blocks, i.e. Python classes and functions, allowing the construction of multiple MMGs for different types of aircraft structural components. Each of these MMGs is a central product

DOI: 10.13009/EUCASS2023-097

model for a typical component developed at GKN Fokker. Each product model is composed of building blocks or 'primitives' that provide a definition model for product features and links to various disciplinary models. The latter provide inputs for disciplinary tools and are in most cases able to automatically execute these tools and read back results to the central product model. The MDM classes are set up using the ParaPy KBE Software Development Kit [4], which has all the typical capabilities of KBE applications such as dependency tracking, runtime value caching and lazy evaluation. Refer [3] for more details.

2.1 MDM product models

At the time of writing the previous paper on MDM [2], the only available product model was the MoveableGenerator. Since then, the available product models have been extended to allow the creation of flaps (FlapGenerator) and wing boxes (WingboxGenerator). At the moment of writing, efforts are underway to set up a tip-to-tip wing box model including the center section (WingGenerator) and a fuselage model (FuselageGenerator).

The class diagram of Figure 1 shows the available product model classes. Each is a specialization of the BaseWingGenerator class, which provides components needed in each product model:

- *Libraries*: this class aggregates libraries needed throughout the product model, such as a material library and a standard parts library. These libraries are typically fixed for an aircraft project.
- *Specifications*: this class aggregates specifications for parts like spars, ribs and stringers, independently of the part instances. These specifications describe certain dimensions and default material allocation for a type of part. This could for example be the shape and dimensions of an L-stiffener, which are then used to create the actual stiffeners in the product.
- *InputHandler*: some inputs to the MDM application could be from a geometry data file such as STEP, BREP, IGS or CPACS. In these cases, the InputHandler converts the geometry in the file to ParaPy geometry objects that are then used in the rest of the application. This ensures the MDM can obtain geometry input from various different sources without affecting the entire application.
- *OutputHandler*: aggregates methods to generate outputs of the model (e.g. STEP, IGES or STL geometry files).
- *BaseGeneratorLoads*: aggregates loads to be applied on the structure, e.g. point loads or constraints and pressure distributions.

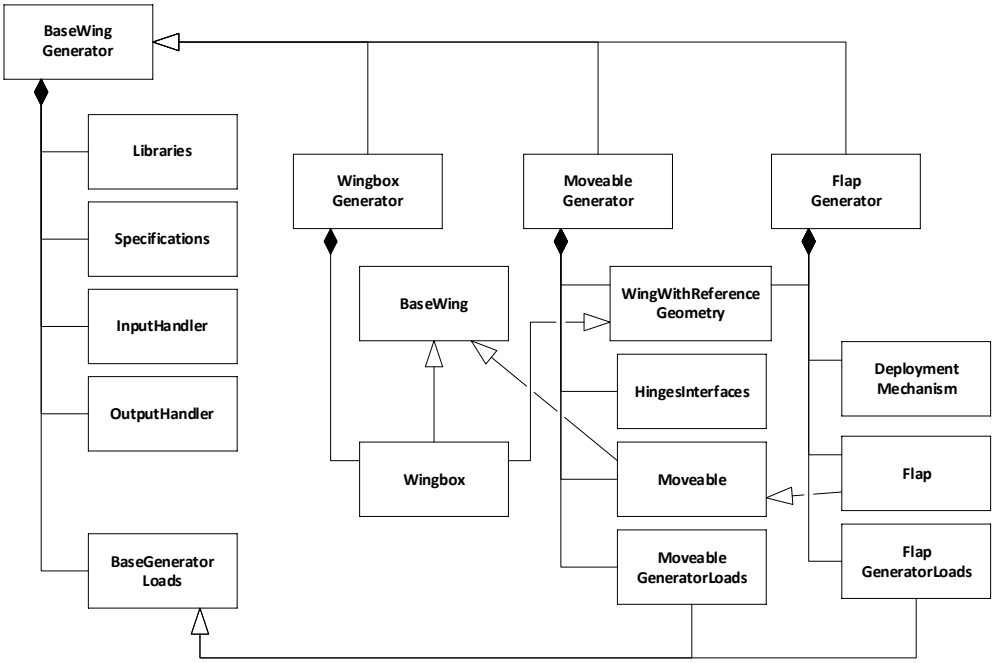

Figure 1: Class diagram of the MDM product models WingboxGenerator, MoveableGenerator and FlapGenerator

DOI: 10.13009/EUCASS2023-097

The WingboxGenerator, MoveableGenerator and Flap product model classes are specializations of the BaseWingGenerator class and add the following product classes:

- *Wingbox*: component class representing a (half) wingbox model, it is a specialization of the *BaseWing* class, which includes spars, ribs, stringer stations and skins, and the *WingWithReferenceGeometry* class, which adds a reference geometry node (input geometry).
- *Moveable*: component class representing a simply hinged moveable (e.g. aileron, elevator, rudder, ruddervator). It is a specialization of the *BaseWing* class.
- *HingesInterfaces*: component class representing the hinges and actuators connecting a moveable to a fixed wing.
- *MoveableGeneratorLoads*: Adds moveable specific constraints to the BaseGeneratorLoads class
- *Flap*: component class representing a flap. It is a specialization of the *BaseWing* class.
- *DeploymentMechanism*: component class representing the deployment mechanism for a flap, linking the flap to the fixed wing.
- *FlapGeneratorLoads*: Adds flap specific constraints to the BaseGeneratorLoads class.

Section 3 describes how analysis nodes are added to generator classes.

2.2 Primitives, the MDM building blocks

An MDM product model is built on functional parametric building blocks, called primitives. As was introduced in [2], primitive classes can represent a physical part or component such as a *SkinPanel* or a more abstract concept such as a *MaterialZone*. The primitive classes are used in all of the product models component classes (*Wingbox*, *Moveable*, *Flap*). This section will showcase some of the primitives used to create a wingbox model.

Figure 2 shows the product tree in the ParaPy app for a typical wingbox case along with the corresponding class diagram. The reference geometry can be provided by different types of classes, which are all specializations of the *BaseReferenceGeometry* class which ensures the reference geometry available for the primitives has a planform and an oml_shapes definition, see Figure 3. This setup allows reference geometry to be defined in the MDM input in completely different ways while returning the same standard geometry objects, enabling other primitives to be independent of the way input geometry was provided.

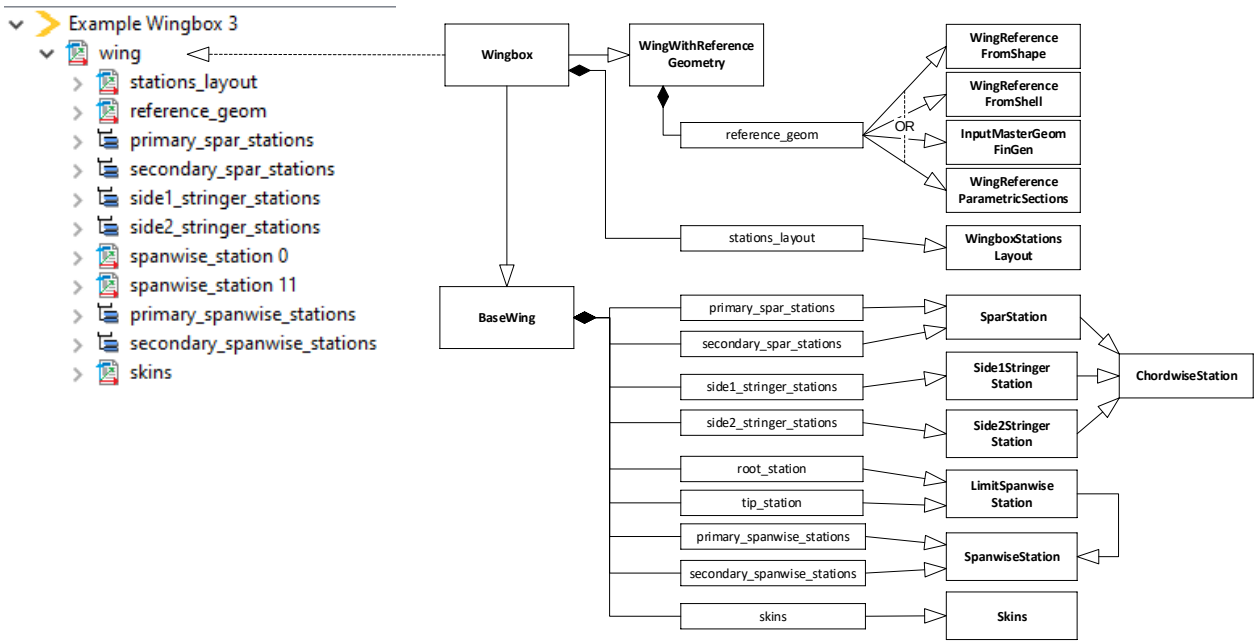

Figure 2: Left: ParaPy product tree of the example_wingbox_3 Wingbox instance; right: Wingbox class diagram

DOI: 10.13009/EUCASS2023-097

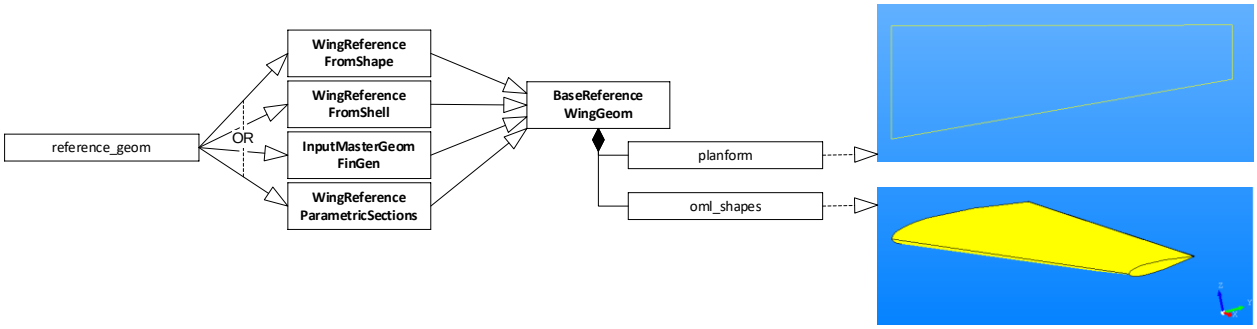

Figure 3: Class diagram showing the different available options to obtain wing reference geometry

The overall layout of the wingbox is set in the station_layout (*WingboxStationsLayout* in Figure 2). Relative to the 2D planform definition, 'chordwise' and 'spanwise' stations are defined to accommodate potential primitives such as spars, ribs and stringers. Figure 4 shows a planform with different stations. The primary_spanwise_stations, root_station and tip_station are all *SpanwiseStation*s. In this class the position of ribs and joints within the stations reference frame can be configured and the corresponding primitives (BasicRib, Joint) are created. Note that the exact position of a rib web does not have to match the stations reference plane, as for various reasons (e.g. alignment with respect to a joint) offsets and rotations may be applied. The side1_stringer_stations and primary_spar_stations are *ChordwiseStations*. In a *ChordwiseStation* the position of spars, stringers and joints within the stations reference frame can be configured and the corresponding primitives (*BasicSpar*, *Joint*, *LStringer*, *HatStringer*, etc.) are created.

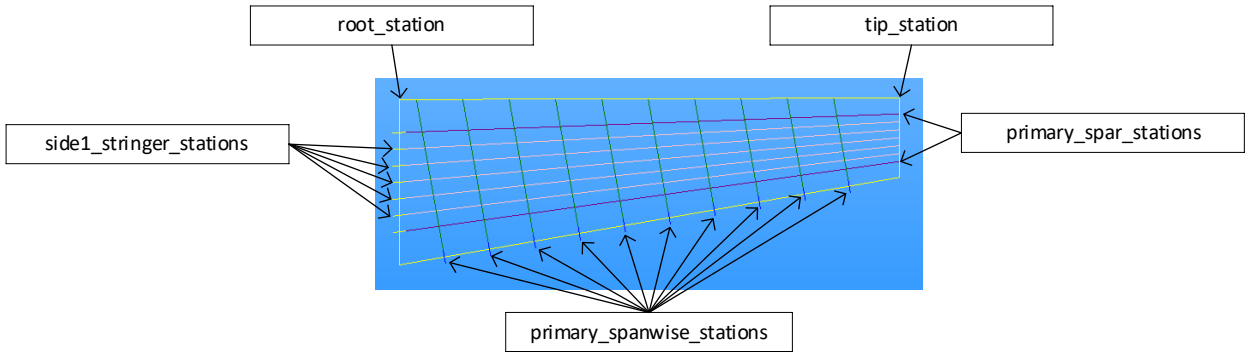

Figure 4: In MDM position chordwise and spanwise stations to accommodate part primitives such as spars and ribs

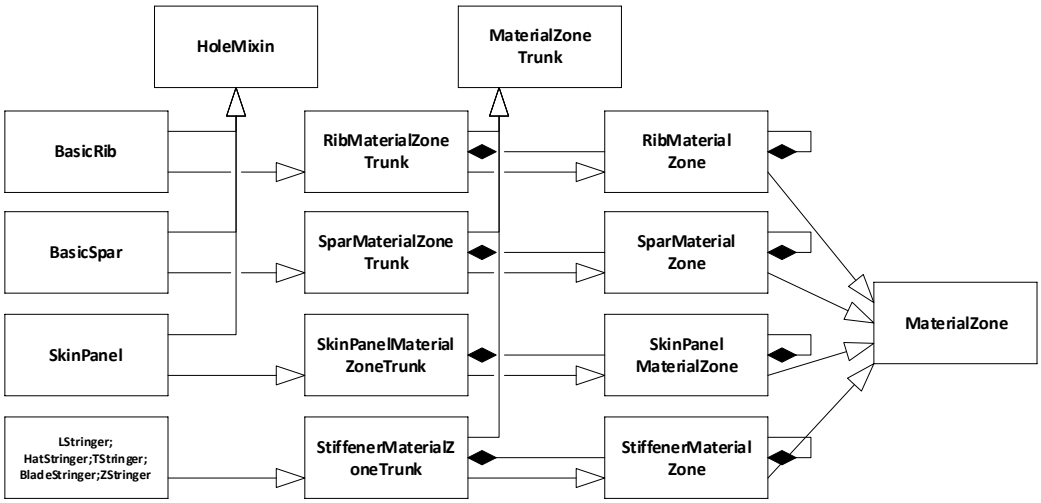

Figure 5: Class diagram of the main part primitives *BasicRib*, *BasicSpar*, *SkinPanel* and **Stringer* classes, each of which is a *MaterialZone*

DOI: 10.13009/EUCASS2023-097

The spar, rib, stringer and skin panel primitives consist of features most aeronautical engineers will expect and these are therefore not elaborated on here. However, to enable flexible subdivision, assignment of material properties and linking to analysis models, the abstract primitive *MaterialZone* was introduced. Depending on the aircraft or research project, there may be different needs in the level of detail of the material assignment or thickness distribution. The MaterialZone allows both defining very coarse models and very detailed models. Figure 5 shows the class diagram of the main part primitives on the left, which can all be seen to be specializations of the *MaterialZoneTrunk* and *MaterialZone* classes. The trunk is the highest level material zone, it is where the material library can be selected. Figure 6 illustrates the use of material zones. At the trunk level the complete primitive geometry is the material zone, in the example a spar_pattern has been applied to it, which means that based on reference planes from the spars, the trunk material zone is divided in three. When a *MaterialZone* is composed of *MaterialZones*, the parent material zone becomes a 'branch'.
All 'leaf' material zones are collected at the trunk level. In the example of Figure 6, since the outer material zones are not subdivided further, these are the leaves. A rib_pattern is applied to the center material zone, which becomes a branch zone. Finally a stringer_pattern is applied resulting in leaf material zones. Note that besides the patterns mentioned, other patterns are available such as from external geometry, span and chord fractions, absolute offsets, etc. Adding a new pattern is relatively straightforward.

When an aircraft project required the modelling of holes in skin panels, the *HoleMixin* was implemented (see Figure 5). As it applies on any primitive which is a MaterialZone, simply adding it as a mixin allowed holes to be available for ribs and spars as well. In many cases, analysis methods can be mapped to material zones instead of a very specific primitive, making it easy to add analysis capabilities (e.g. see Figure 7 and Figure 14).

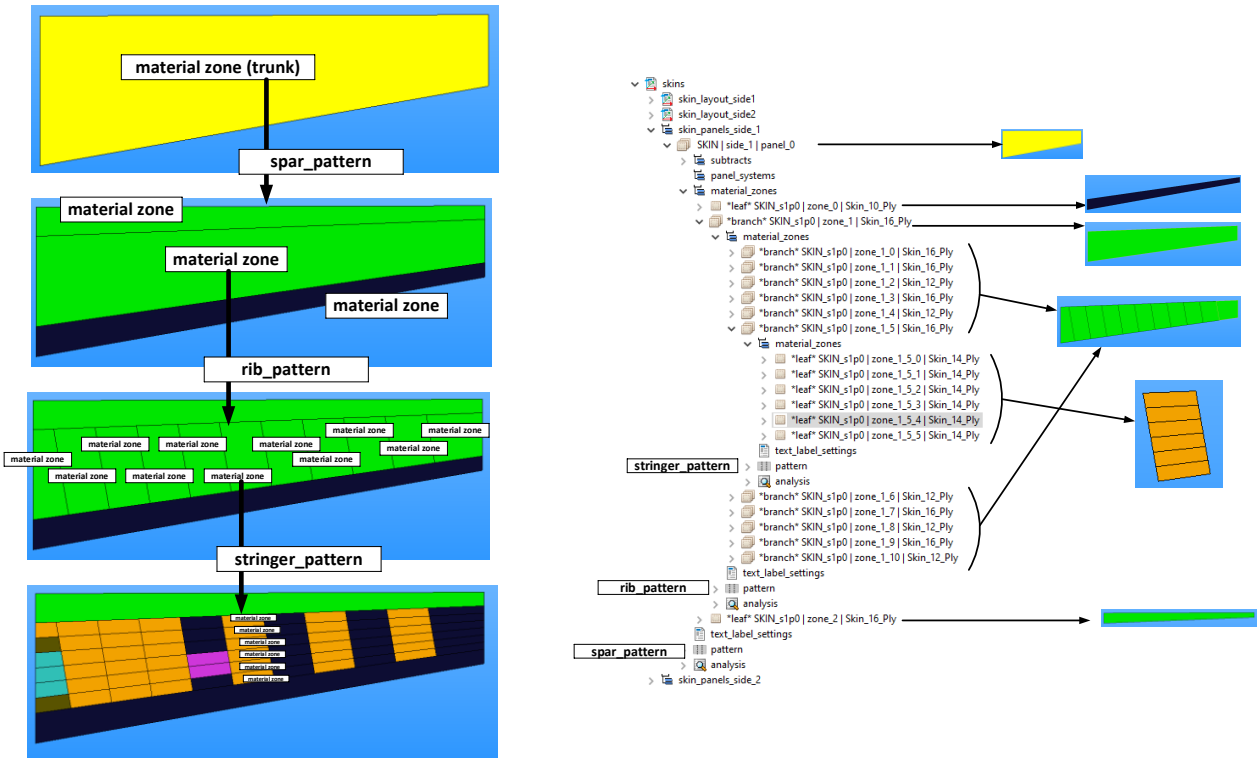

Figure 6: Material zones of the example_wingbox_3 case. Left: representation of the material zones in the nested tree; Right: material zones in the ParaPy product tree

3. Analysis

The main objective of the MDM is to rapidly generate consistent multidisciplinary models. A disciplinary model will cover the creation of inputs for a specific disciplinary tool, the capability to trigger the execution of the tool and read back relevant results.

DOI: 10.13009/EUCASS2023-097

3.1 Integration in the product model

The integration of analysis models is designed to anticipate for future changes in disciplinary tools and their API changes. As was explained in [2], each primitive in MDM has a primitive-specific class mixed in of the *PartWithAnalysis* type, where for each discipline a class type can be specified. This class effectively assigns disciplinary models for a specific primitive. The generic *PartWithAnalysis* class creates an instance of the *Analysis* class where each of the discipline specific class types is instantiated. No disciplinary models are instantiated if no class type has been specified in the *PartWithAnalysis* specialization.

This mechanism is shown in Figure 7 for the *BasicRib* primitive. As the *BasicRib* inherits from a material zone trunk, it can be composed of sub-material zones. Due to the analysis model mapping shown, a *BasicRib* instance will have an instance of the *RibOpenSourceCost*, *RibStress* and *RibC2F* being instantiated in its analysis node. The sub-material zones will each have an instance of the *MaterialZoneStress* and RibC2F instantiated in the analysis node. As no open source cost model class is specified, no model will be created for the individual material zones.

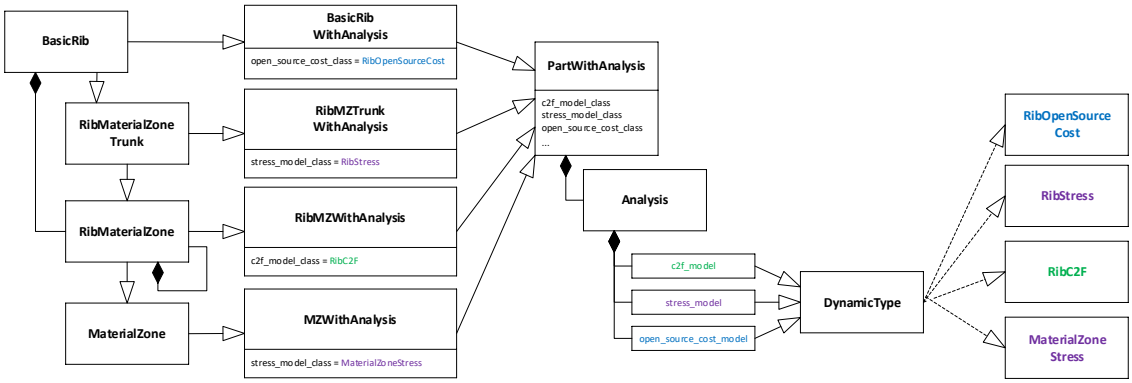

Figure 7: Class diagram of the BasicRib primitive and the integration of analysis models

This mechanism allows disciplinary analysis models to be associated to primitives without making the primitive definition model dependent on any of these disciplinary models. By not providing settings to the individual disciplinary models, they can be suppressed when generating an instance, hence allowing the use of the primitive without necessarily creating all the potential disciplinary models.

The root of the product model has an analysis node where for each of the active disciplines a disciplinary model is instantiated that aggregates the disciplinary models of the primitives; is able to construct the product-level disciplinary model, execute the disciplinary tool if applicable and retrieve results. The aggregation is facilitated through a collector part that contains the logic to extract specific primitives from the product model. The root level disciplinary model provides discipline specific application settings, which can be accessed by the primitive-level disciplinary models. How this works is depicted in Figure 8.

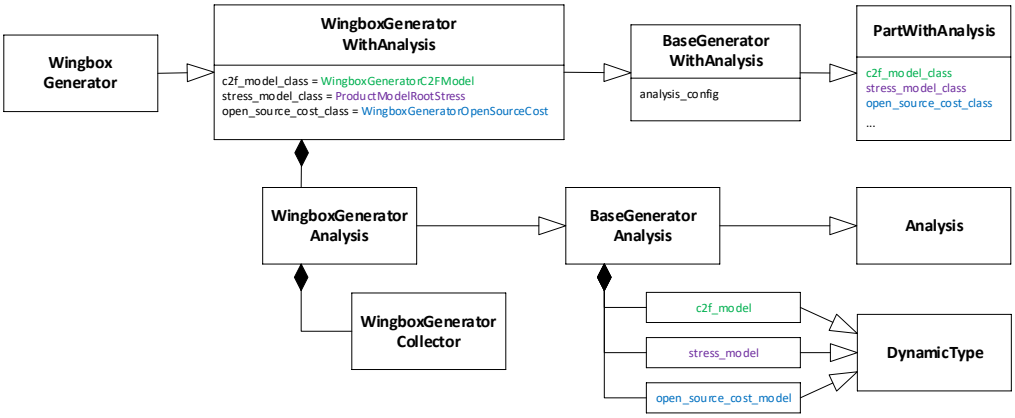

Figure 8: Class diagram of the root product model and the integration of analysis models

DOI: 10.13009/EUCASS2023-097

The architecture implemented allows flexible linking of disciplinary models to primitives and makes it possible to add any number of disciplinary models without affecting the primitives or other disciplinary models. The approach can initially be complex to grasp for starters, although multiple people at the time of writing have been able to use and extend the disciplinary models.

3.2 Available analyses

The following disciplinary models are available in MDM:

- Mass properties model: generates input for the in-house Python based mass estimation tool. Results can be obtained both at individual primitive level (e.g. only calculate the mass of a single stringer) and at complete product level. The latter effectively adds up the masses of primitives.
- Project based cost model: generates input for the proprietary Excel based cost estimation tool. The tool can be executed to obtain results both at the complete product level and individual primitive level.
- Open source cost model: inputs are created for the Python based CATMAC [10] tool and results can directly be obtained.
- Finite Element Method (FEM) segmentation model: a fully segmented representation of the product surface geometry to be used for finite element meshes.
- C2F model: interface to the in-house FEM pre- and postprocessor using PATRAN. Uses geometry from the FEM segmentation model.
- FEM model: Model providing a ParaPy and Salome based finite element mesh and an interface to ABAQUS, see section 3.3.
- Stress model: Model of stress analysis specific views of the product model, including a failure mode model and representation of results, see section 3.4.

3.3 Finite element model

The C2F model provides a Finite Element (FE) model of Critical Design Review level quality and is used in various aircraft programs. However, the application has relatively high runtimes and is limited to PATRAN/NASTRAN. In order to obtain more rapid mesh generation and an interface to the ABAQUS FE solver, the MDM FEM model has been developed. ParaPy provides libraries to interface with the Salome meshing package as well as interfaces to simulation solvers such as ABAQUS. These have been used to create a generic mesh model and an interface capable of generating ready-to-run ABAQUS input (.inp) files. Figure 9 shows an instance of the mesh for one of the standard MDM example cases and Figure 10 shows the imported model in ABAQUS. A comparison between cases evaluated in the C2F tool and in ABAQUS from the same MDM model showed only small difference. These differences are being studied as the model is under continued development.

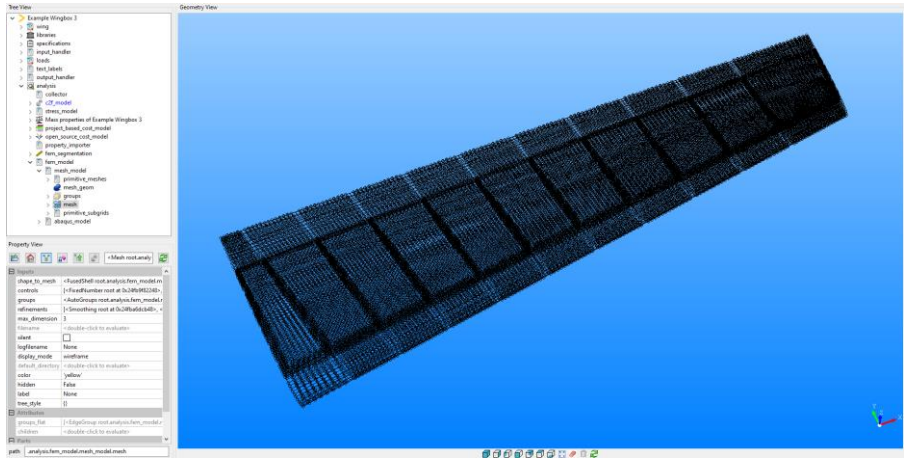

Figure 9: ParaPy developer GUI showing the MDM example_wingbox_3 case product tree and the mesh model

DOI: 10.13009/EUCASS2023-097

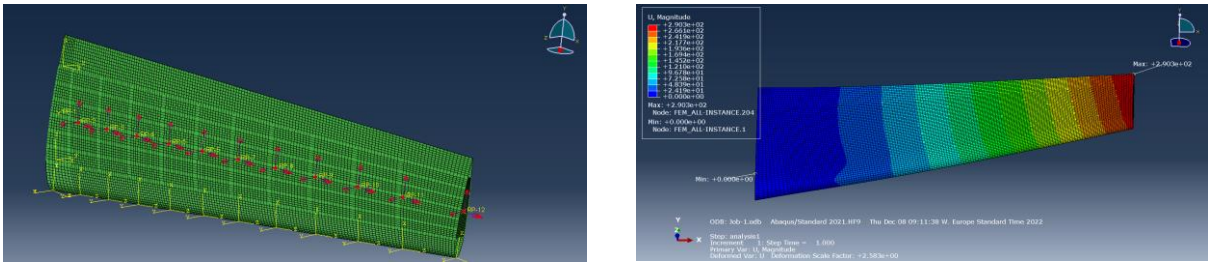

Figure 10: Screenshots of the example_wingbox_3 case exported to ABAQUS. Right side shows loads on the model, left side the total displacement magnitude.

3.4 The stress model

The stress model allows the evaluation of structural constraint, i.e. the calculation of reserve factors (RFs) for failure modes applying to a specific case. In practise, the failure modes that need to be evaluated can differ per case, or different methods/tools are to be used for their calculation. Some Original Equipment Manufacturers (OEMs) actually require their in-house tools to be used over GKN Fokker tools. The stress model implementation therefore provides stress analysis specific representations of the model, allows selecting specific failure modes, interfaces to various stress analysis tools and retrieval of results, i.e. the minimum RFs and corresponding loadcase and failure mode.

Stress specific model representation: split material zone model
Material zones are not necessarily split at each rib, spar and stringer, such as is the case in the left side of Figure 11 where the trailing edge and leading edge are a single material zone. Some stress analysis models used require all material zones to at least be split at each spar, rib and stringer position. To handle these cases without bothering the user with the definition of irrelevant material zones, the split material zone model automatically adds virtual material zones, as depicted on the right side of Figure 11.

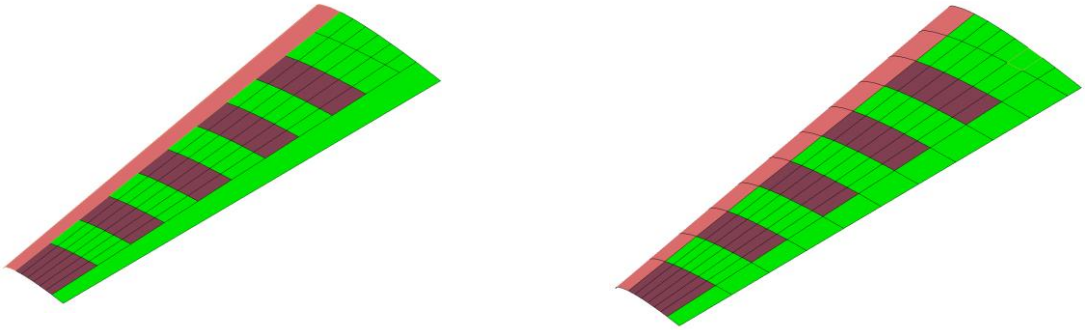

Figure 11: Screenshots of the example_wingbox_3 case skin side 1. Left side: Main material zone model; Right: split material zone model.

Stress specific model representation: buckling panel model
Some stress methods such as the analytical calculation of buckling, require an idealized or simplified representation of the actual geometry. The buckling panel model identifies which material zones from a panel enclosed by stiffening elements (ribs, spars, stringers) and determines the equivalent flat rectangular panel dimensions. These dimensions can used in specialized buckling analysis software. Figure 12 shows the geometrical representation of the buckling panel model.

DOI: 10.13009/EUCASS2023-097

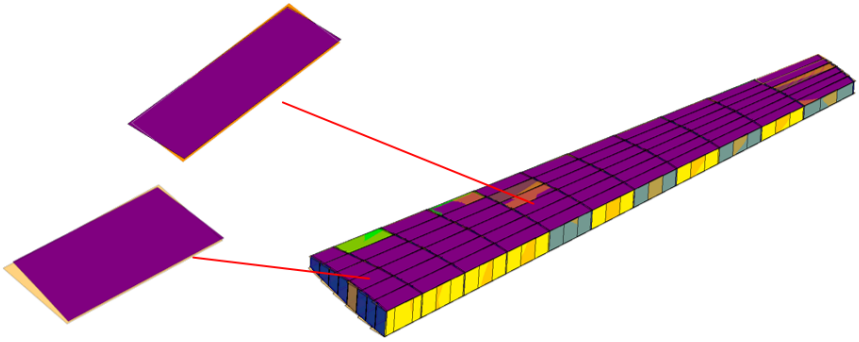

Figure 12: Screenshot of the example_wingbox_3 cases buckling panel model.
For each skin panel (in purple) the equivalent rectangular panel is shown in orange.

Interfaces to stress analysis tools

The Python based 'Stress Generator' (SG) is a GKN Fokker proprietary tool to calculate reserve factors (RF's) for a range of typical failure modes. This tool requires a number of XML and CSV files describing the model, load and other settings. The MDM stress model has a stress_gen_model node (class: *SGRoot*) of which the class diagram is shown in figure 13. *SGRoot* provides various settings as for example which version of SG to use, run directory definition, etc. As the tool needs load output from the CAD2FEM model, a dedicated fem_results node (class: *StressGenC2F*) ensures all CSV paths leading to the required C2F output data are available. For each primitive type category an *SGPartGroup* is created, where all the input files (XML and CSV) with part data and run settings are created. These are saved to the SG run directory. A method is available to directly execute the tool, which will subsequently produce various output files. The *ResultsExtractor* class (see Figure 13) has methods to extract specific data from the SG output files.

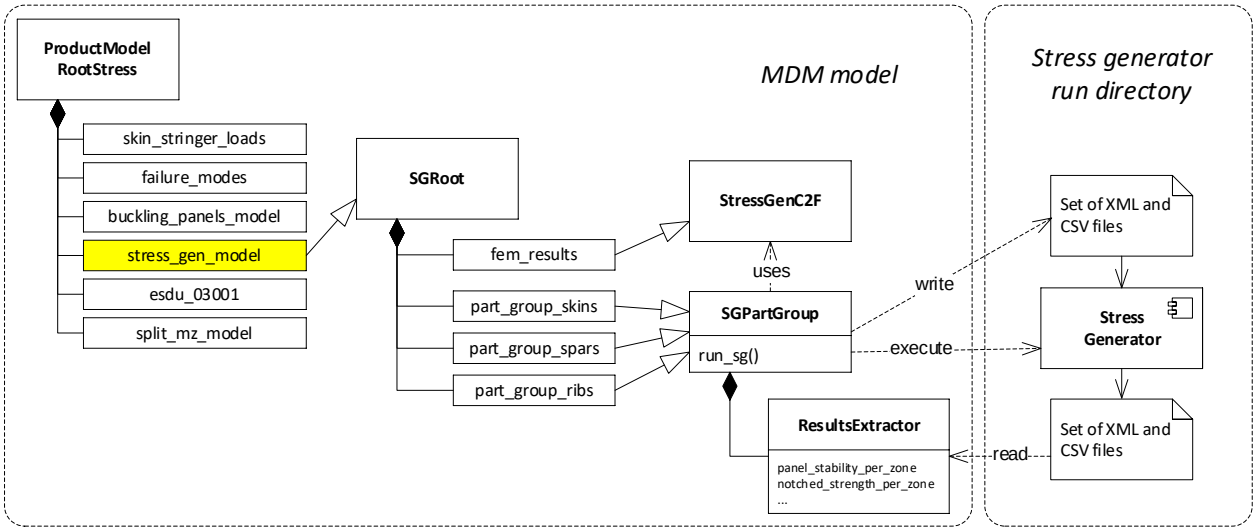

Figure 13: Class diagram of the Stress Generator interface model in MDM

Another stress analysis tool that has been integrated is for the ESDU 03001 method (elastic buckling of composite stiffened panels and struts in compression). Using the buckling panel model (Figure 12) as input, the *ESDU03001Model* class defines 'stringer regions', i.e. sets of stringer sections with the adjoining skin panel sections from which the inputs required for the ESDU method are extracted. For each of the stringer regions, the ESDU executable can be automatically executed and the resulting panel buckling allowable values are extracted. Other stress tools can of course be added to the overall stress model as needed.

Failure modes model

The failure modes models class diagram is shown in Figure 14. For each primitive type a *FailureModesGroup*, visible at the top-center of the figure, is created where the failure modes that are to be analysed are selected. The selected failure modes are the active_failure_modes and for each of these an object is created where the calculation method can be selected. The method has to correspond to an extracted output from one of the stress analysis tool interfaces, such as the SG of the previous section. For example, if 'notched_strength' is set to active in *CompositeStrength*, the

DOI: 10.13009/EUCASS2023-097

CompositeNotchedStrength class is instantiated and allows the selection of a calculation method. Each calculation method corresponds to a specific tool and/or tool calculation settings. For example, for a SG method, the SG interface from the previous section will use these settings to properly set up the inputs for the tool. The output from the selected analysis method is collected in the failure mode object per material zone (min_rf_per_mz in Figure 15).
Each individual material zone can retrieve its min_rf from the model, and at each level up in the failure modes product tree the minimum of the underlying RFs is selected as the min_rf, up to and including the *FailureModesRoot*. A dedicated visualization class allows the representation of the failure mode analysis results, including load case and failure mode. In the example of figure 15, material zones in red are violating the RF margin of 1.0, green zones have 1.0 < RF < 2.0 and blue zone RF > 2.0. It also shows that for some zones material strength is the critical failure mode, while for other zones it is buckling.

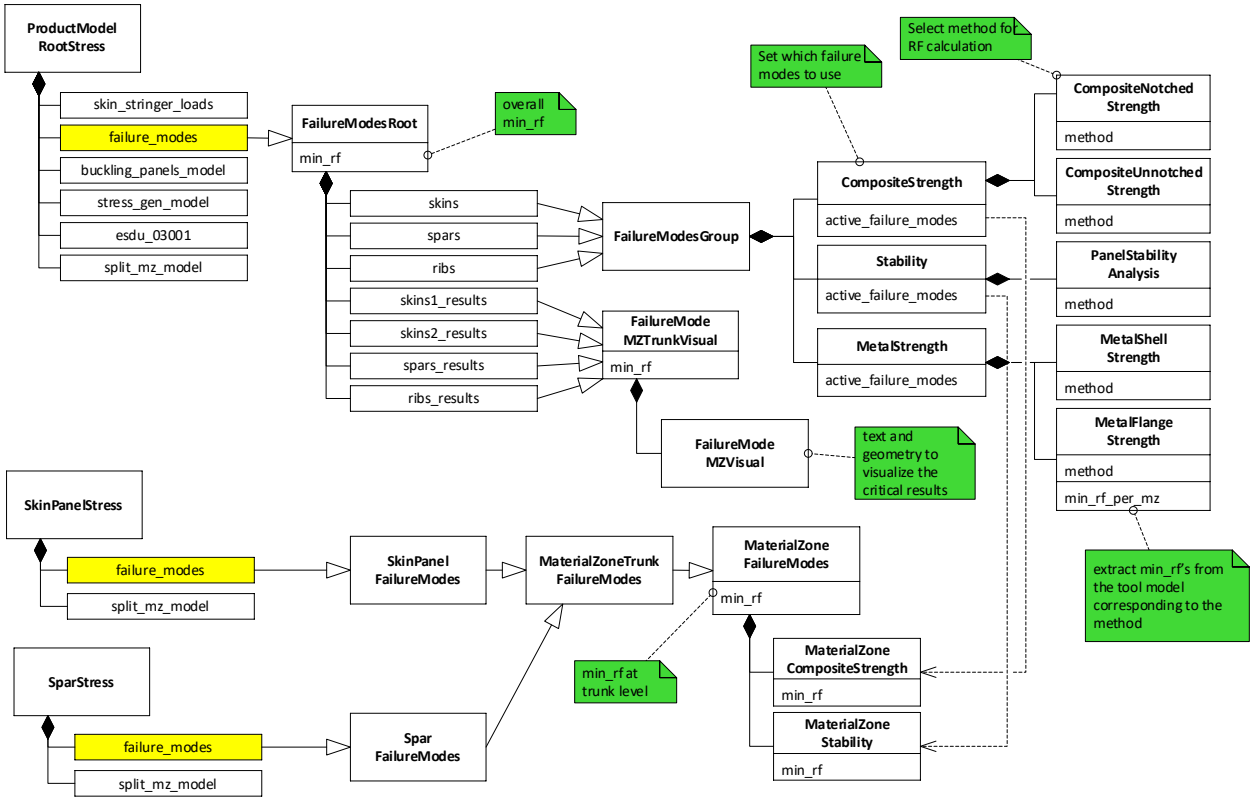

Figure 14: Failure modes model

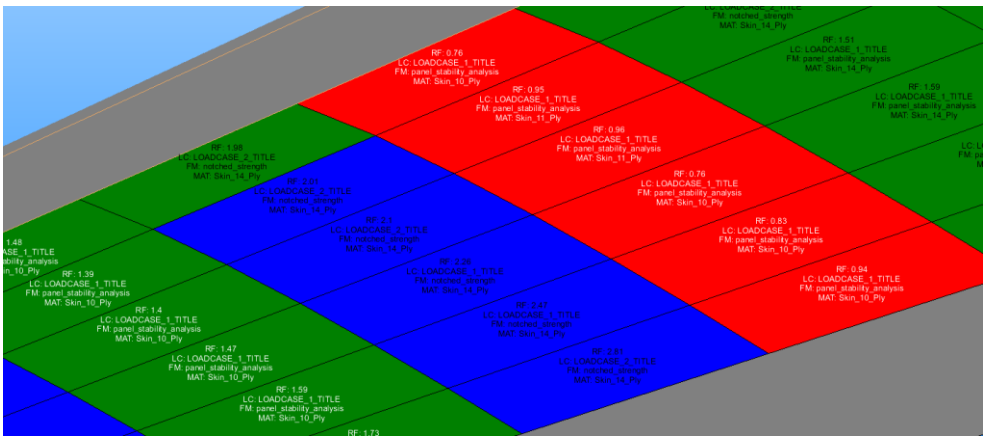

Figure 15: Screenshot of the example_wingbox_3 cases stress analysis results visualization.

The structure of the failure modes model allows obtaining RFs using various method without needing to set up the individual tools as these are integrated according to standard ways of working. A new calculation method/tool can easily be integrated by adding a new calculation method to the failure mode object. This is advantageous, as changes in method do not affect MDM use cases that use the min_rf results. If, for example, notched strength needs to be

DOI: 10.13009/EUCASS2023-097

calculated, but instead of the SG method an OEM tool is required to be used, only this method needs to be added. Any sizing workflow would keep on using the min_rf value that was already available, only now it would be calculated by a different method.
To support rapid sizing of an architecture in MDM, a basic pattern search has been implemented that will change material assignments for a material zone until a minimum viable RF has been found. This method can be executed with or without FEM convergence (i.e. re-evaluating the FE model based on a new material allocation).

3.5 The manufacturability model

Besides interfaces to stress models, the MDM also features interfaces to manufacturing models. Currently the most prominent features of these models are the Project based cost model and the Open source cost model. Both models are capable of estimating part cost of an aircraft component modeled in the MDM. In [10], the integration of the open source cost model named CATMAC is described in more detail. The integration of the cost model in MDM has shown that manufacturing can be taken into account on a similar level as the FEM and stress analyses presented in the previous sections. However to achieve the same fidelity level more detailed models are required that model not only part cost, but also the manufacturing workflows and other essential manufacturing aspects. Currently developments are underway to extend the MDM in this direction and some examples can be seen in Figure 16.

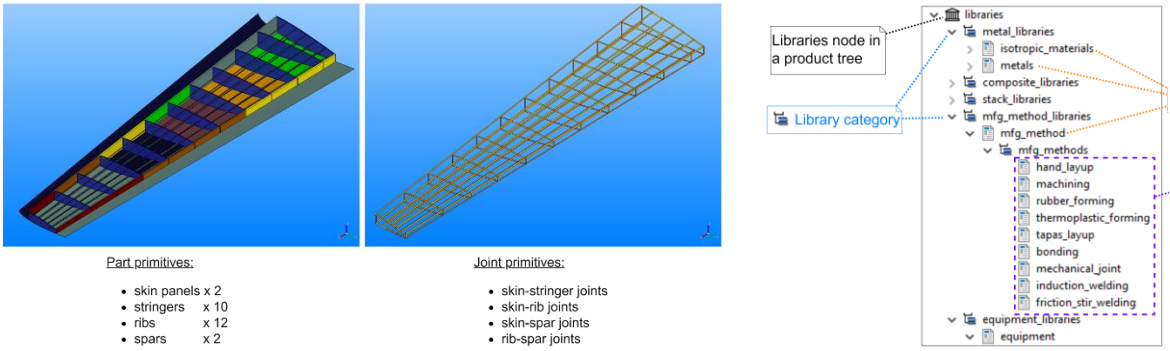

Figure 16: Manufacturing items and methods library represented in the MDM

4. Interacting with MDM

There are several different ways of interacting with MDM, which interaction is used depends on the context in which MDM is used. In the most basic case, a user can interact directly with the MDM using file based or Graphical user interface (GUI) inputs. MDM can also be used as a service where MDM is running on a server and a user can set inputs and/or get data produced by the MDM from the server. Finally, MDM can also be packaged to work directly in optimization workflows.

4.1 Direct Inputs and operation in a ParaPy development GUI

To create the various models, MDM requires many inputs to operate. However many of these inputs are standardized and can be reused with minimal adjustment. When initializing MDM directly, so running MDM in a python script, the inputs are assigned to MDM via files. MDM uses python dictionaries as input items. These can be stored directly as a dictionary in a python file or they can also be stored as a JSON file. MDM can handle both types of input. The input dictionaries describe the aircraft component to be analysed and the environment in which to analyse. For example, the environment can set the variables for the Finite element or stress analyses. Besides the .py or JSON files containing the MDM configuration, other files might also be required. For example if the MDM bolts a structure inside an existing Outer Mould Line (OML) or uses other geometrical references, this OML and or references have to be provided using a STEP file.

Once the MDM is loaded, it is visualized in the development GUI of ParaPy. In this GUI the user can adjust variables and inspect the geometry of the model. By adjusting and inspecting, the user can further fine tune the model. In the GUI, the user also has the possibility to apply certain filters. Enabling them to, for example, hide reference geometry and only focus on their items of interest. Once the user is satisfied with the configuration, they can also trigger the different analysis modules from the GUI and inspect the results from the analysis modules. The ParaPy GUI is shown in Figure 9.

DOI: 10.13009/EUCASS2023-097

Finally, the user can also store a configuration created in the GUI, including the analyses executed. The configuration is stored in a set of files consisting of an MDM JSON file and some files referenced from this main file. By using this option, a user can store the work from a ParaPy GUI session and continue at a later date.

The direct input and operation approach is currently the most used at GKN Fokker and used extensively to support multiple aircraft component design projects. The interface is quite complex with many options and buttons and therefore needs trained users. In future more dedicated user interfaces will be developed to allow disciplinary specialists, such as stress or manufacturing specialists, to interact with the MDM model.

4.2 MDM as a service: the pyMDM client

Having most disciplinary tools integrated in MDM, these disciplinary tools no longer need to be added to the MDO workflow explicitly. Instead, the engineering service requesting a certain parameter such as the total cost can be called, which will automatically trigger the execution of the cost tool through the cost model. If the mass is required to calculate the cost, Parapy's dependency tracking will automatically trigger the calculation of the mass without needing to add an additional step for it in the workflow. The product model remains available in memory and any changes to the model variables will invalidate the dependent analysis models. This approach allows describing the MDO workflow with service calls specific to the objectives and constraints of the MDO problem formulation.
In order to make it possible to run an MDM instance and change it at runtime without needing to completely instantiate the model, a Flask based server has been created to host MDM instances. A separate package 'pyMDM' has been created to interact with the server to create instances, remove them, change variables and retrieve properties. Pymdm has been used to run Design of Experiments from RCE as explained in [11]. Figure 17 shows the main workflow where the initialize_mdm block adds an instance to the server and the set_get_mdm block changes specific variables and retrieves specific results, as specified in an XML input file, thereby making the workflow setup generic.

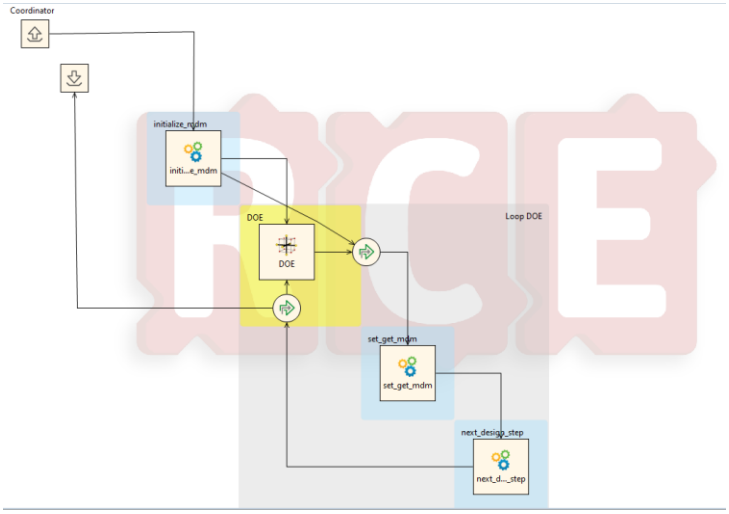

Figure 17: Screenshot RCE workflow with an initialize_mdm block and set_get_mdm block

4.3 Optimization

MDM can also be used in optimization workflows. How MDM is packaged in such a case depends on the workflow manager used and the characteristics of the workflow that is run.
One case of using MDM in a workflow is described in [5]. In this case, the MDM is part of a workflow to optimize a flap. In the workflow, MDM is used several times. Firstly to create the flap geometrical and FEM model and secondly to estimate the flap weight. In both cases, MDM is run as a python script with a file input describing the flap to be analyzed. In case of AGILE 4.0, CPACS is the standard communication format, this is an open source XML format for describing aircraft systems. MDM interprets this CPACS configuration file plus a set of standard .py files to generate a model and update the CPACS file with analysis results.
Another case where MDM is incorporated into a workflow is the DEFAINE use case. In this case the MDM server is used and services are requested from this sever when needed. This process is described in detail in [11]

DOI: 10.13009/EUCASS2023-097

5. Application cases

The MDM is actively used in various aircraft programs, for which the data is confidential. In order to develop new features and test the MDM models, an example case representative for an Electric Vertical Take-off and Landing (eVTOL) aircraft wing ‘example_wingbox_3’ has been added to the standard MDM examples. Figures 6, 11 and 12 show representations of this model. Besides aircraft programs the MDM has also been used in EU-subsidized projects, two of which are briefly described in the sections below.

5.1 AGILE 4.0 use case

The AGILE 4.0 project focused on applying Model Based Systems Engineering (MBSE) techniques to multi-objective optimization problems. In this context, the MDM was used to model flaps of a 90 passenger regional jet. Detailed results of this study are available in [4]. To support the creation of a surrogate model of the flap weight and cost ,multiple flap sizes were modeled using the MDM. Besides the wing structure, the structure of the flap mechanisms were also modeled and two different mechanism types were evaluated, examples of which can be seen in Figure 18.

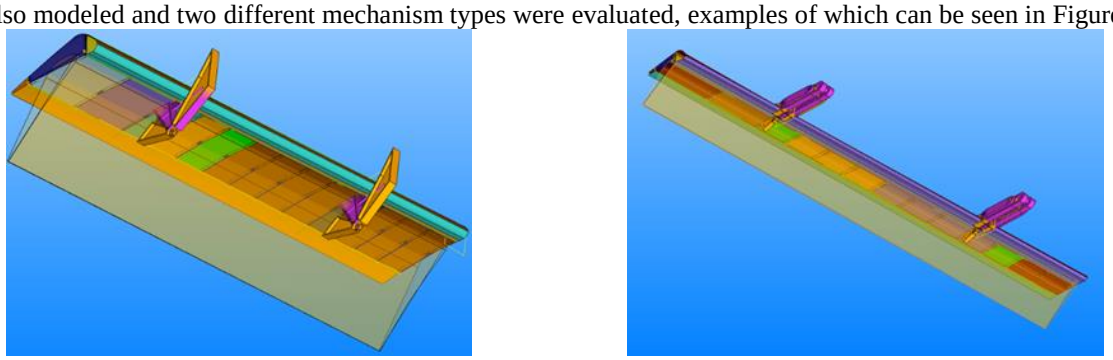

Figure 18 Two flaps modelled for the Agile 4.0 project. Left a flap with a large chord and a standard kinematic solution. Right a small chord flap with an advanced kinematic solution

5.2 DEFAINE use case

In the DEFAINE project, the objective is to perform design space exploration for the aileron structure architectures of an Unmanned Air Vehicle (UAV). The objectives of this optimization are total mass of the aileron and total cost. In this case, the number of ribs, number of hinges, hinge positions of the aileron and material types (AL2024, thermoplastic composites, thermoset composites, thermoset with honeycomb) are varied, and the aileron is sized. This means that the material properties are determined, for which a reserve factors of more than 1.0 is achieved. The resulting material specifications can be inspected in the MDM (Figure 19). As a preliminary step towards full design space exploration using the new architecture optimization methods, multiple design concepts were analysed in a Design Of Experiments (DOE), resulting in a plot shown in Figure 20. The pareto front consists of configurations made from thermoplastic composites.

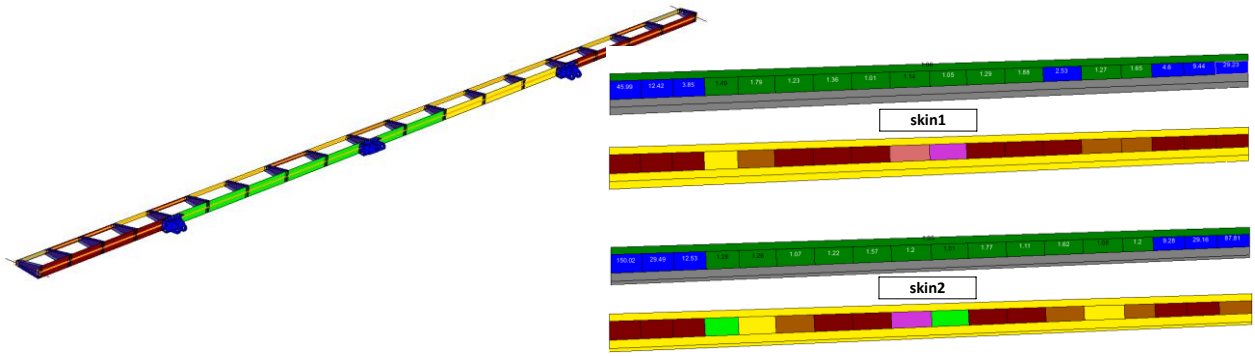

Figure 19: DEFAINE aileron with sized material properties. Each colour represents a different material lay-up.

DOI: 10.13009/EUCASS2023-097

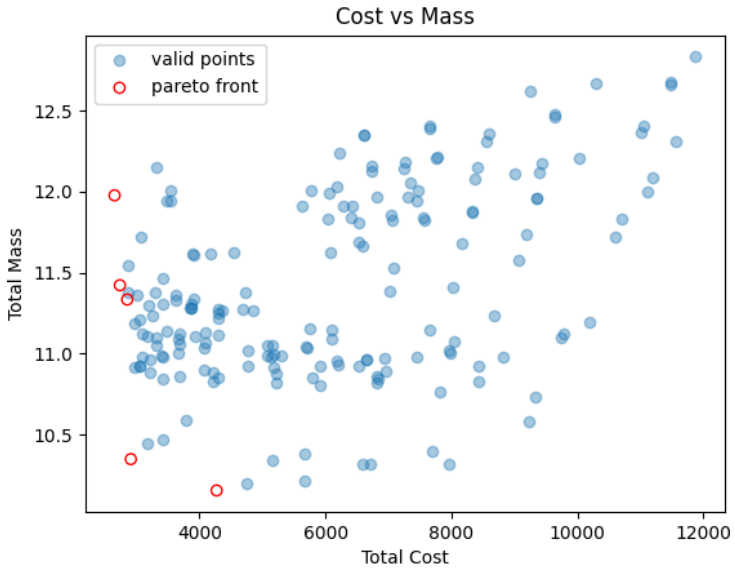

Figure 20 DOE results for the DEFAINE aileron architecture optimization case

6. Conclusion

This paper has shown how the Multidisciplinary Modelers (MDM) package is at the heart of design automation at GKN Fokker. Its ability to model wing-like structures and to link these models efficiently to analysis tools has enabled the automation of the engineering design process for aircraft components. The MDM is set up modularly, which will facilitate extending its capabilities. The multiple ways MDM can be used or be provided data, via a server or direct, has enabled it to be applied to multiple industrial design cases and in multiple subsidized collaboration projects.
To increase its capability MDM will be continuously developed. Some of these future developments will be:

- *Web Graphical User Interface*, to make using MDM easier for disciplinary specialist from for example stress or manufacturing a web GUI will be developed that provides specific views on a product for the specific specialist.
- *Tip to tip wing*, currently only one side of a wing is modelled. In the future, the modelling of a complete tip-to-tip wing will be enabled. This will eliminate certain problems that currently arise, for example when that are symmetries in a wing.
- *Fuselage model*, a big next step is the addition of a fuselage model to the MDM, this is described in more detail in [9]

MDM will continue to be at the heart of the engineering automation effort at GKN aerospace. Enabling to respond quickly to customer requests and supply the best engineering solutions to its customers.

Acknowledgement

The research presented in this paper has partially been performed in the framework of the DEFAINE (Design Exploration Framework based on AI for froNt-loaded Engineering) project and has received funding from ITEA 3 programme.

References

[1] A.H. van der Laan, T. van den Berg, L. Hootsmans, "Integrated Multidisciplinary Engineering Solutions at Fokker Aerostructures", 5th CEAS Air and Space Conference, Delft, 2015
[2] T. van den Berg, A.H. van der Laan, "A multidisciplinary modeling system for structural design applied to aircraft moveables", AIAA AVIATION 2021 FORUM, VIRTUAL EVENT, 2021.
[3] La Rocca, G. “Knowledge Based Engineering: Between AI and CAD. Review of a language based technology to support engineering design.”, Advanced Engineering Informatics, vol. 26, no. 2, pp. 159-179., 2012.
[4] Parapy, "The ParaPy platform " [online]. URL : https://parapy.nl/features/ [retrieved 10 January 2023]
[5] A.H. van der Laan et. al., “Bringing Manufacturing into the MDO domain using MBSE”, AIAA AVIATION 2022 FORUM, Chicago, 2022

DOI: 10.13009/EUCASS2023-097

[6] A.M.R.M. Bruggeman, “An MBSE-Based Requirement Verification Framework to support the MDAO Process”, AIAA AVIATION 2022 FORUM, Chicago, 2022

[7] M. Baan, et. al., “DEFAINE – Design Exploration Framework based on AI for front-loaded Engineering: Achievements and Open Challenges”, submitted to Joint 10th EUCASS-9th CEAS Conference,

[8] J.S. Sonneveld, A.M.R.M. Bruggeman, G. La Rocca, T. van den Berg, B. van Manen, “Dynamic workflow generation applied to aircraft moveable architecture optimization”, submitted to Joint 10th EUCASS-9th CEAS Conference, Lausanne, 2023

[9] B. van Manen, T. van den Berg, A.H. van der Laan, B. Timmer, “A multidisciplinary modeling system for designing fuselage structures: Extending the Multidisciplinary Modeler”, submitted to Joint 10th EUCASS-9th CEAS Conference, Lausanne, 2023

[10] A.H. van der Laan et. al., “An open source part cost estimation tool for MDO purposes”, AIAA AVIATION 2020 FORUM, online, 2020

[11] Jente S. Sonneveld et. al, “Dynamic workflow generation applied to aircraft moveable architecture optimization“, CEAS 2023, Lausanne

[12] La Rocca, G. “Knowledge Based Engineering Techniques to support Aircraft Design and Optimization”, PhD thesis, Delft University of Technology, 2010.

[13] van den Berg, T., “Harnessing the potential of Knowledge Based Engineering”, PhD thesis, Delft University of Technology, 2013.“