FliPASED
WWW.FLIPASED.EU

D1.4 Information and interfaces definition for Collaborative Work Process

Thiemo Kier, Yasser Meddaikar, Matthias Wuestenhagen (DLR), Fanglin Yu (TUM), Balint Vanek, Bela Takarics (SZTAKI), Charles Poussot-Vassal (ONERA)

GA number:	815058
Project acronym:	FLIPASED
Project title:	FLIGHT PHASE ADAPTIVE AEROSERVO-ELASTIC AIRCRAFT DESIGN METHODS
Funding Scheme: H2020	**ID:** MG-3-1-2018
Latest version of Annex I:	1.1 released on 12/04/2019
Start date of project: 01/09/2019	**Duration:** 40 Months

Lead Beneficiary for this deliverable:	SZTAKI
Last modified: 18/11/2021	**Status:** Delivered
Due date: 30/09/2021	

Project co-ordinator name and organisation:	Bálint Vanek, SZTAKI
Tel. and email: +36 1 279 6113	vanek@sztaki.hu
Project website:	www.flipased.eu

Dissemination Level		
PU	Public	X
CO	Confidential, only for members of the consortium (including the Commission Services)	

“This document is part of a project that has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 815058.”

Glossary

ASE	Aeroservoelastic
CAD	Computer-aided Design
CPACS	Common Parametric Aircraft Configuration Schema
DLM	Doublet Lattice Method
FEM	Finite Element Model
SW	Soft-ware
HW	Hard-ware
VV	Verification and Validation
GLA	Gust Load Alleviation
MLA	Manoeuvre Load Alleviation
MDO	Multidisciplinary Design Optimization
MLA	Manoeuvre Load Alleviation
PID	Proportional-Integral-Derivative
RCE	Remote Component Environment
HIL	Hardvare-in-the-loop
FCC	Flight Control Computer
TCL	Tool Command Language

Table of contents

List of Figures

1 Executive Summary

The deliverable ”D1.4 Information and interfaces definition for Collaborative Work Process” lays the foundation of the MDO toolchain developed in WP2 of the project, in correspondence with the demonstrator aircraft. In the beginning of the project, several key factors have been identified and objectives as well as performance metrics have been proposed to show the benefits of the MDO tool-chain. The interdisciplinary teams within the project share models, data and tools among them. D1.4 formalizes these steps within the iteration loops and establishes a document to define their interdependency and their standard interfaces (CAD, NASTRAN, Dymola, Matlab/Simulink, python, embedded C code). In addition to the MDO toolchain, an interface to the HIL tests needs to be defined as well. The HIL tests serve as crucial investigation on whether the developed models and controllers are implementable on hardware as well as their final assessment before the flight tests. Finally, the developed tools in the MDO toolchain are evaluated in flight tests as well after the HIL tests. The lead beneficiary for the deliverable is SZTAKI, but all other consortium partners TUM, ONERA,and DLR contributed significantly to the deliverable by various aspects of the interface definitions and data sharing definitions and MDO toolchain setup tests.

2 Interdisciplinary Design Architectures and Status

There are three main toolchains developed within FLiPASED, which can be seen in Figure 1.

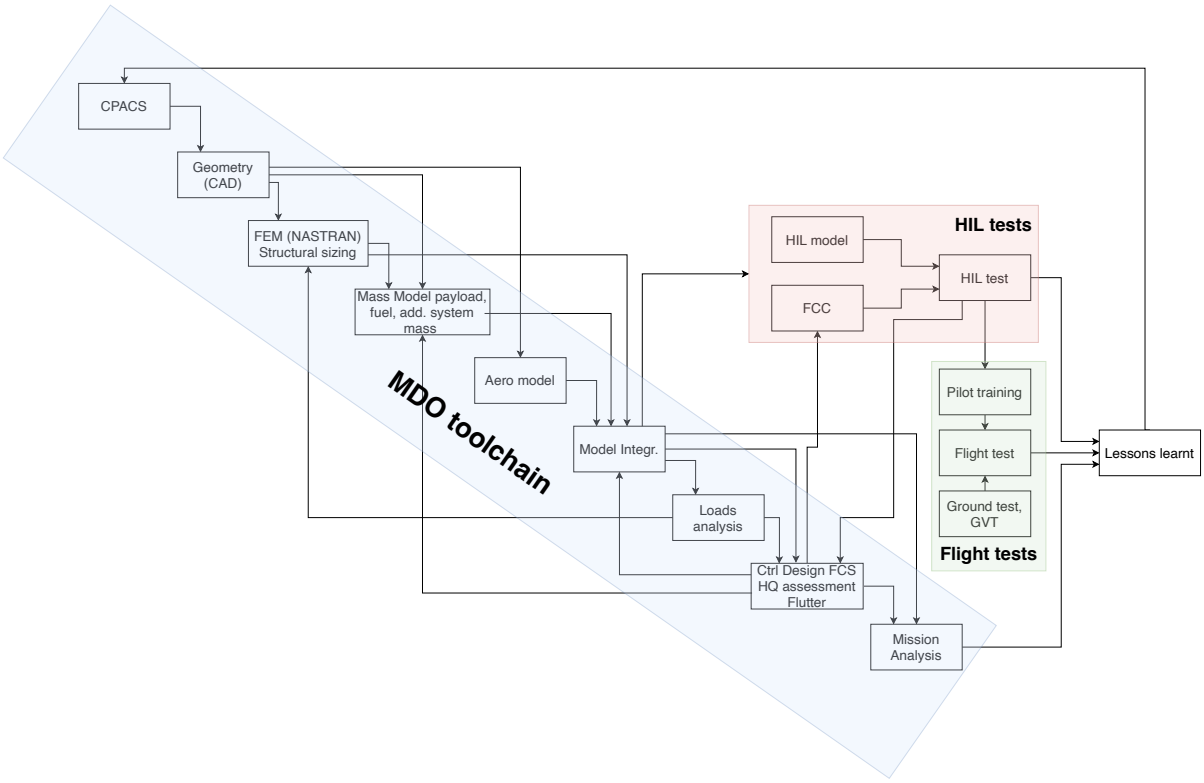

Figure 1: Toolchains developed in FLiPASED

The main toolchain is the MDO toolchain that starts from the CPACS aircraft geometry definition and finishes with the developed aircraft geometry/parameter set tightly coupled with the baseline, manoeuvre load alleviation, gust load alleviation and flutter suppression controllers evaluated in the mission analysis. Based on a successful mission analysis the controllers and the models need to be handed over to the HIL test block. The HIL model needs to be real-time executable and the controllers in discrete from. The HIL tests asses the performance of the controllers in addition to the implementation aspects. The third main block are the flight tests. This block serves as the final maturity test of the developed methodologies. Conclusion are drawn from all three blocks which are then feed back to the CPACS model generation step.

2.1 MDO toolchain development status

The MDO toolchain development is carried out in two parallel branches. In the first branch each partner implements its own set of tools into the RCE framework locally. This step requires capturing the requirements of the input data coming from the previous MDO block and defining the output data the actual block is creating. This branch of the MDO toolchain is mostly finalized and only minor adjustments are needed. This is the milestone when the current deliverable can be finished.

In the second branch, the communication between the partner blocks and data sharing is implemented. In this case first a simple toolchain is set up for creating variables that are then shared among the partners. The example workflow is shown in Figure 2. This test was carried out successfully. As the final step, the full blocks of the first branch need to be set up to communicate between various partners and to be able to share data. This step is currently under ongoing development and in the next FLiPASED meeting (22/11/2021) at TUM the implementation aspects will be verified and the initial MDO toolchain run is expected to be carried out.

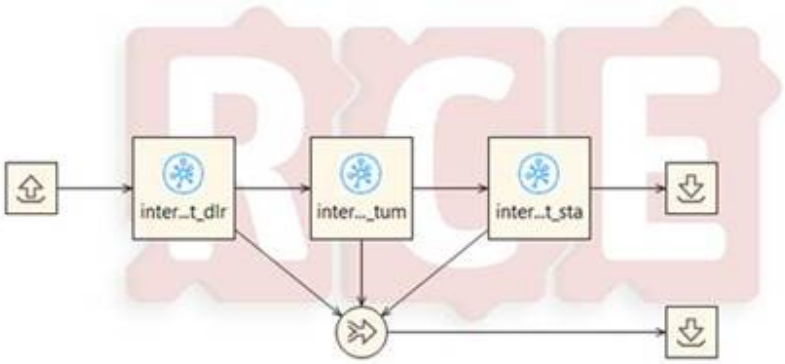

Figure 2: RCE workflow for the testing the communication and data sharing between partners

2.2 Hardware-in-the-loop (HIL) testing status

Due to the significant effort of conducting flight tests the main method to clear any newly developed system component or software function is to test it in the HIL simulation platform. It has two distinct versions, one is hosted on a legacy Windows 10 based PC, running Simulink Real Time, and inerfaced with external devices via the standard PCI cards of a desktop PC. This system has two instances: one at SZTAKI (for software development) and another one at TUM (for pilot training). The other HIL is based on a Speedgoat target machine, which is a turnkey solution with dedicated hardware interfaces and dedicated software implementation of the required communication protocols between the simulation and hardware components, this is also available at two locations (SZTAKI uses it for SW/HW development and another one is under commission at DLR to develop the necessary real-time capable simulation platform for V&V).

The inputs (controllers and models), input and output interfaces have been defined to the HIL tests. The only remaining items are the discrete versions of the MLA and GLA controllers, what are under fine-tuning, to be able to test the complete ASE system in the HIL environment.

2.3 Flight testing status

The flight test with the demonstrator aircraft are running in parallel with the MDO toolchain development. First the operation of the aircraft is investigated, then system identification tests are carried out and finally the maturity of the developed controllers within the MDO toolchain will be evaluated. The detailed scope and test schedule of these flight test campaigns are discussed in D3.1, D3.2, D3.3, D3.6, D3.8 and 3.11 respectively.

3 Aircraft Geometry and FEM Model generation

Since 2005 DLR develops the Common Parametric Aircraft Configuration Schema, short CPACS. It contains a parametric description of aircraft configurations as well as the complete transport system, e.g. fleet and airport descriptions. On the other hand control system related layout and parameter information is not standardized in it.

The number of interfaces in multi-disciplinary aircraft design is crucial for a flexible and efficient flow of information. Along with CPACS the number of interfaces between analysis modules is not only reduced but also do these become replaceable, as all adapt to one common definition.

The CPACS format allows the automatic generation, validation and documentation of data-sets. As a part of the Release Kit, CPACS format, documentation and sample configurations are made available at https://www.cpacs.de/ or at https://github.com/DLR-SL/CPACS.

The Common Parametric Aircraft Configuration Schema (CPACS) is a data definition for the air transportation system. CPACS enables engineers to exchange information between their tools. It is therefore a driver for multi-disciplinary and multi-fidelity design in distributed environments. CPACS describes the characteristics of aircraft, rotorcraft, engines, climate impact, fleets and mission in a structured, hierarchical manner. Not only product but also process information is stored in CPACS. The process information helps in setting up workflows for analysis modules. Due to the fact that CPACS follows a central model approach, the number of interfaces is reduced to a minimum.

3.1 CPACS Generation

CPACS Generation block is the first block in the MDO toolchain. It will generate a CPACS file as shown in Figure 3 which holds aircraft configuration and optimisation variables, for instance, airfoil, planform, structure layout and so on. All these information and optimisation variables needs to be given as the input for this block. A Matlab script is used to initialize the CPACS data.

3.2 Geometry Model Updating

Geometry Model Updating block takes the CPACS data as the input. Tixi (https://github.com/DLR-SC/tixi) and Tigl (https://github.com/DLR-SC/tigl) libraries are used to extract geometry and structure information from CPACS. A Catia macro is used to update the existing wing model to the latest parameters. A Catia model will be the only output of this block, as shown in Figure 4.

3.3 FEM Model Generation

FEM Model Generation block takes the updated Catia model as input and uses TCL programming language in HyperMesh to generate a FEM model. All relevant meshing parameters, modelling techniques and interfaces with fuselage model and empennage model are predefined in macro script of HyperMesh. A Nastran wing model is generated with an established numbering scheme and outputted to several bdf files in a folder with predefined structure, as shown in Figure 5.

Node	Content
?-? xml	version="1.0" encoding="utf-8"
cpacs	
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation	CPACS_Schema.xsd
header	
vehicles	
materials	
aircraft	
model	
uID	FLEXOP
name	FLEXOP
reference	
fuselages	
wings	
wing	
uID	WR
symmetry	x-z-plane
name	WR
description	WR
parentUID	FU
transformation	
sections	
positionings	
segments	
componentSegments	
wing	
analyses	
profiles	
wingAirfoils	
fuselageProfiles	
toolspecific	

Figure 3: CPACS data

3.4 Aerodynamic Model Generation

Aerodynamic Model Generation takes the CPACS file as input and extracts airfoil and wing planform using Tixi and Tigl library. The TiGL library uses the OpenCASCADE CAD kernel to represent the airplane geometry by NURBS surfaces. The library provides external interfaces for C, C++, Python, Java, MATLAB, and FORTRAN. A Python script is used to generate DLM panel model and written out to bdf files as predefined file structures as shown in Figure 6

3.5 RCE Integration

All aforementioned blocks are integrated into RCE as shown in Figure 7. RCE is a distributed integration environment for scientists and engineers to analyze, optimize, and design complex systems like aircraft, ships, or satellites. It is especially suited for multidisciplinary collaboration. Handling complex systems requires many experts and several tools for analysis, design, and simulation. Using RCE, these tools can be shared between team members and integrated into automated, executable workflows. RCE is extensible and supports different scientific applications with a wide range of requirements.

Corresponding wrappers are written in python to enable the integration. An additional Tigl viewer block is added to the workflow to visualize the aircraft configuration.

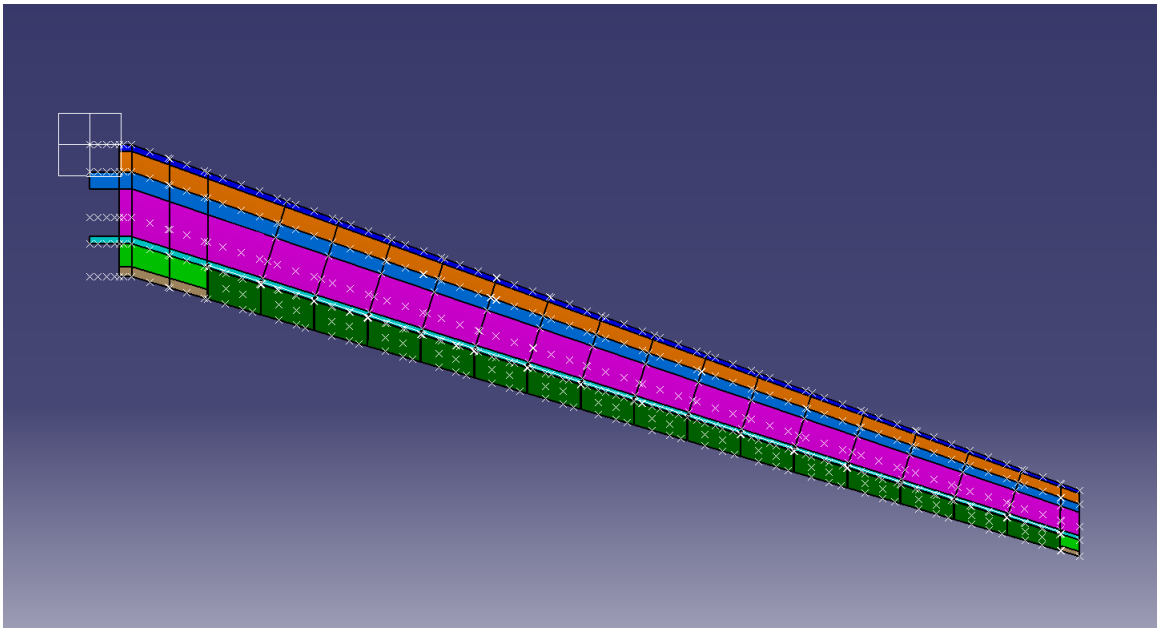

Figure 4: Catia wing model

rbe2_LW.nastran	07.10.2021 10:12	NASTRAN-Datei	8 KB
rbe2_MW.nastran	07.10.2021 10:12	NASTRAN-Datei	4 KB
rbe2_RW.nastran	07.10.2021 10:12	NASTRAN-Datei	8 KB
rbe3_LW_NoUM.nastran	07.10.2021 10:12	NASTRAN-Datei	10 KB
rbe3_MW_NoUM.nastran	07.10.2021 10:12	NASTRAN-Datei	4 KB
rbe3_RW_NoUM.nastran	07.10.2021 10:12	NASTRAN-Datei	10 KB
rbe2_LF.nastran	07.10.2021 10:12	NASTRAN-Datei	6 KB
rbe2_RF.nastran	07.10.2021 10:12	NASTRAN-Datei	6 KB
rbe3_LF_NoUM.nastran	07.10.2021 10:12	NASTRAN-Datei	7 KB
G_rbe2_LW.nastran	07.10.2021 10:12	NASTRAN-Datei	13 KB
G_rbe2_RW.nastran	07.10.2021 10:12	NASTRAN-Datei	13 KB
rbe3_RF_NoUM.nastran	07.10.2021 10:12	NASTRAN-Datei	7 KB
G_rbe2_MW.nastran	07.10.2021 10:12	NASTRAN-Datei	4 KB
G_rbe2_RF.nastran	07.10.2021 10:12	NASTRAN-Datei	9 KB
G_rbe2_LF.nastran	07.10.2021 10:12	NASTRAN-Datei	9 KB
set1.nastran	07.10.2021 10:12	NASTRAN-Datei	5 KB
set1_RW.nastran	07.10.2021 10:12	NASTRAN-Datei	4 KB
set1_LE.nastran	07.10.2021 10:12	NASTRAN-Datei	5 KB
set1_LW.nastran	07.10.2021 10:12	NASTRAN-Datei	4 KB
set1_MW.nastran	07.10.2021 10:12	NASTRAN-Datei	4 KB
set1_LF.nastran	07.10.2021 10:12	NASTRAN-Datei	4 KB
set1_TE.nastran	07.10.2021 10:12	NASTRAN-Datei	5 KB
csm.nastran	07.10.2021 10:12	NASTRAN-Datei	1.656 KB
set1_RF.nastran	07.10.2021 10:12	NASTRAN-Datei	4 KB
rbe3_LW.nastran	07.10.2021 10:12	NASTRAN-Datei	14 KB
rbe3_LF.nastran	07.10.2021 10:12	NASTRAN-Datei	9 KB
rbe3_MW.nastran	07.10.2021 10:12	NASTRAN-Datei	4 KB
rbe3_RW.nastran	07.10.2021 10:12	NASTRAN-Datei	14 KB
csm_right.nastran	07.10.2021 10:12	NASTRAN-Datei	882 KB
rbe3_RF.nastran	07.10.2021 10:12	NASTRAN-Datei	9 KB

Figure 5: Defined file architecture for wing FEM model

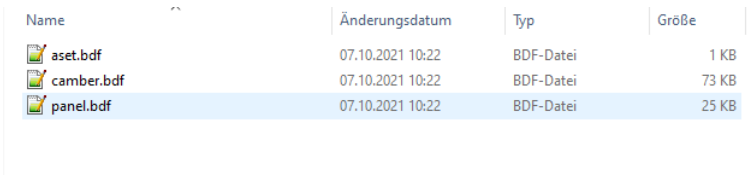

Figure 6: Defined file architecture for wing DLM model

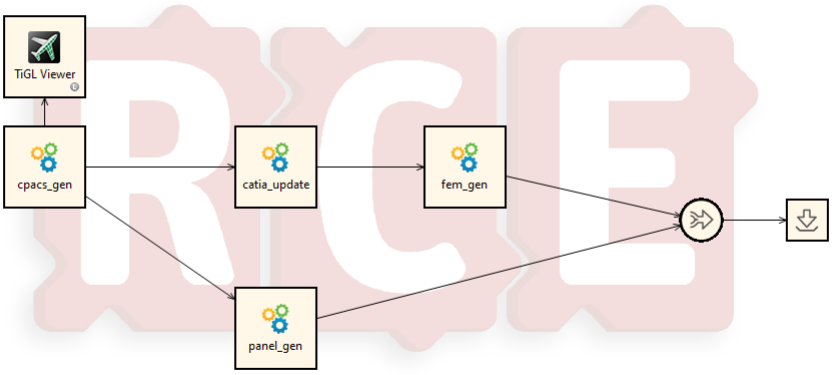

Figure 7: Integrated blocks in RCE

4 Aeroelastic Model Generation

4.1 Aeroelastic Model Integration - NASTRAN

From the perspective of the RCE workflow, the input to the NASTRAN aeroelastic model generation block are the following.

1. *CPACS.xml* - containing the most recent aircraft CPACS dataset
2. *wingFE* directory - directory containing the FE and DLM models of the wing, generated by TUM
3. $principal\_angle\_shifts_{1,2}$ float variables - outer-level optimization variables that define the prinicipal angle with respect to which the laminates in the upper and lower skin are oriented

The wing models are generated by the preceding block following an established numbering scheme for the entire aircraft, together with defined interfaces for assembly with the fuselage and empennage models. This ensures that different configurations of the wing model are compatible with the existing fuselage and empennage models, generated based on FLEXOP data. The input wing model to this RCE block has a defined file-folder hierarchy as shown in Figure 8.

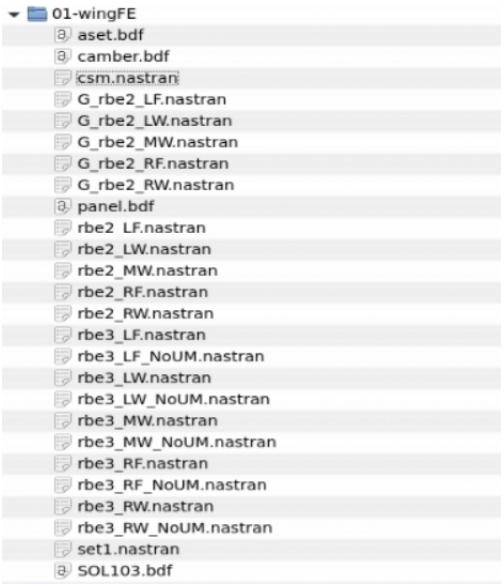

Figure 8: Defined folder-file architecture for wing models from TUM

The NASTRAN aeroelastic model integration block primarily performs the following tasks.

1. Create a modified wing FE model by rotating the existing laminate definitions on the upper and lower skins according to the input variables $principal\_angle\_shifts_{1,2}$
2. Assemble the aerodynamic model of the aircraft by merging the panel definitions, spline sets and the camber correction entries for the wing, fuselage and empennage
3. Run pre-defined NASTRAN decks corresponding to modal, aeroelastic, flutter analyses and a static Guyan reduction

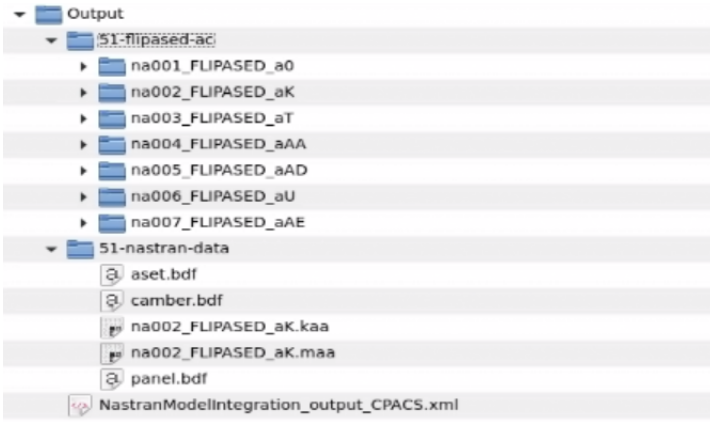

Figure 9: Defined folder-file architecture of NASTRAN aeroelastic models to DLR-SR

4. Aggregate the output data, including mass and stiffness matrices, and pre-defined aerodynamic bulk data into the output directory

The outputs from this block include two directories and the CPACS dataset as shown in Figure 9.

1. *51-nastran-data* directory - contains the outputs required by the next partner in the RCE workflow, DLR-SR in this case. Files include the mass and stiffness matrices, aerodynamic bulk data - panel definition and camber correction, and other outputs needed for the tools downstream.
2. *51-flipased-ac* directory - contains different NASTRAN solution decks for various analyses in order to aid in debugging.
3. output CPACS dataset - for the demonstrator workflow, the CPACS dataset is not altered during the execution of the tool. For the scale-up workflow, information from analyses such as structural weight, thickness and material properties of the various structural entities can be appended.

4.2 Aeroelastic Model Generation and Simulation

The aeroelastic model generation and simulation workflow implemented in RCE is given in Figure 10. The workflow is executed from the left to the right. All the corresponding functions are executed in Matlab. The result of each individual block is saved in a Matlab struct. First the aerodynamic, structural and spline grid information as well as mass and stiffness matrices are provided to the first block called ”varloads model”. VarLoads is a tool created in Matlab for defining flexible aircraft models by e.g. setting-up aerodynamic influence coefficient matrices and performing an eigenvalue analysis of the aircraft structure. The results are passed on to the block ”create model input”. The data is then downsized and provided in a specific form, so it can be used with the Simulink simulation environment.

In the block ”trim lin model” the simulation environment is initialised and also linearized. It is possible to adapt the simulation environment based on various parameters, that have to be defined. First of all the model order is selected by deciding on a model with unsteady aerodynamics or steady aerodynamics, flexible dynamics or rigid dynamics. Furthermore, dynamics coming from sensors, actuators, airbrakes and the engine can be switch on or off. Dependent on the simulation to be performed or the type of controller to be synthesized gust inputs and load outputs can be added. Finally the operating point for which the aircraft model should be trimmed and linearized has to be selected by defining the indicated

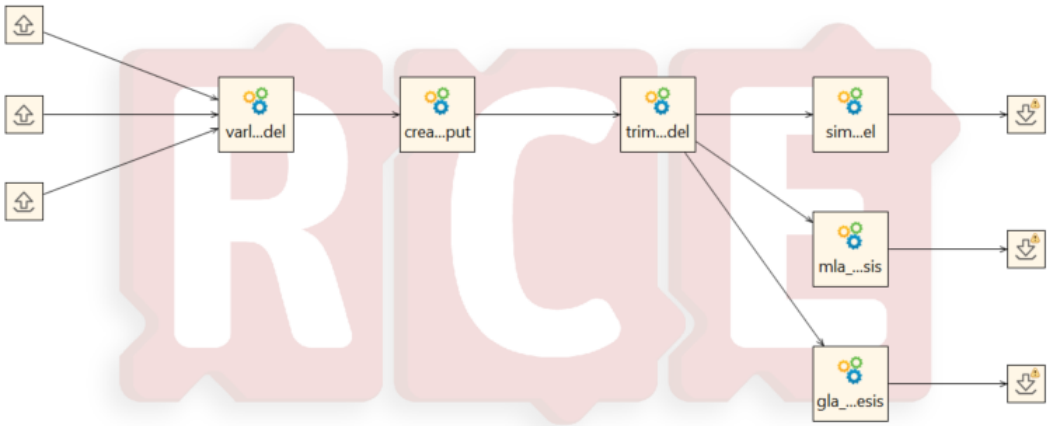

Figure 10: RCE workflow for the aeroelastic model generation and simulation

airspeed, the barometric height, the roll angle and others. Subsequent to the block ”trim lin model” the model can be simulated with the block ”sim model” by means of the trim results. It creates a time series for dedicated inputs commanded to the control surfaces, the engine rotational speed and so on.

4.2.1 Models to be generated

As already stated before, several flavors of the same model can be generated. The goal is to keep the number of the models as small as possible while satisfying the requirements of the control design blocks. The generated models are the following:

- Model for baseline control design – 12 state rigid model, set of LTI models parameterized by velocity;
- Model for manoeuvre load alleviation – LTI models;
- Model for gust load alleviation – LTI models;
- Model for flutter suppression controller design – low order flexible model, set of LTI models;
- Model for mission analysis – LTI and nonlinear model;
- Model for HIL tests and pilot training – real time capable nonlinear Simulink model.

5 Control Design Blocks

In accordance with the workflow in Figure 10, after the ”trim lin model” block has finished, the synthesis of the various controllers follows. The linearised state-space systems offer the opportunity to synthesize linear controllers or gain-scheduled.

5.1 Baseline and Flutter Suppression Control Design Blocks

The model generation, the control synthesis and the analysis of the resulting controllers for the baseline and flutter controllers is shown in the workflow presented in Figure 11.

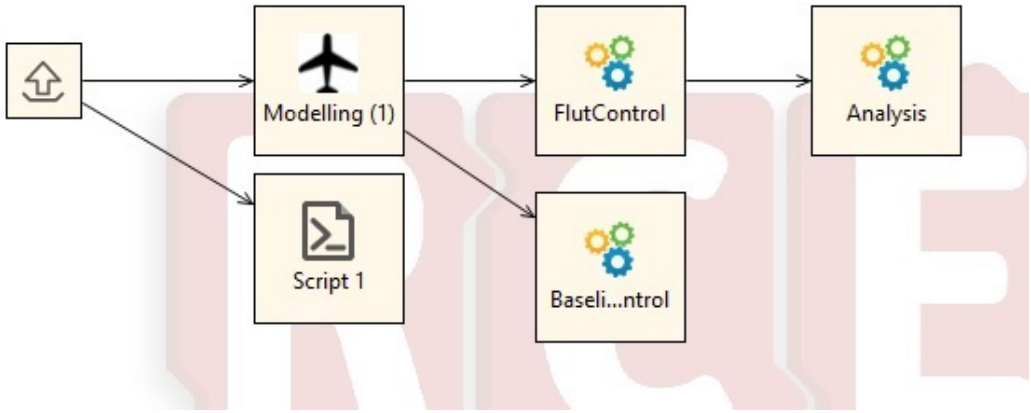

Figure 11: RCE workflow for the aeroelastic model generation for baseline and flutter suppression control design

The RCE workflow takes the nonlinear Simulink model with the configuring struct file from DLR-SR as the input files. These files are shared through RCE compressed files and are referenced withtin the CPACS file. The ”Modeling” block generates two models in this case. The flutter control synthesis block requires a low order aeroelastic model as an input. The aeroelastic model is obtained by the bottom-up modeling approach ([3, 4]) which provides a sufficiently low order model for the control design. The generated model is a set if linearized model obtained from trimming and linearizing the simulink model. The linearized state space models are parameterized by the airspeed, but also by uncertainties in the FEM model of the aircraft. This model is saved as a mat file with name FlexACModel. The baseline controller accepts the rigid body, 12 state linearized models as input. This model is also generated in the ”Modeling” block from the aeroelastic moodel by residualizing the flexible and aerodynamic states. The baseline control design model does not contain any uncertainties, but a set of LTI state space model parameterized by the airspeed. This model is saved as a mat file with name RigACModel.

Based on the FlexACModel the flutter suppression controller is generated in the ”FlutControl” blocks ([2]) and sets the controller as a state space model at the output in a mat file. Based on the RigACModel the baseline controller is synthesized in the ”BaselineControl” block ([1]) and sets the simulink block with the configured PID controllers as the output.

Once the flutter suppression and baseline control design blocks finish the synthesis they pass the resulting controllers and the FlexACModel model to the ”Analysis” block. This block then runs a fre-

quency domain analysis in two aspects. First, it assesses the performance of the two controllers acting together simultaneously. Second, it checks the robustness margins and flutter margins of the resulting controllers and if the minimum requirements are satisfied a pass flag is set and a PDF report is automatically generated. The pass flags and the PDF are finally set as the outputs of the block.

The main algorithms of each block and their adaptation to the MDO/RCE framework is given in deliverable D2.2 Report on tool adaptation for collaborative design.

5.2 MLA and GLA Control Design Blocks

The second part of Figure 10 shows the manoeuvre load alleviation controller block ”mla control synthesis” and a gust load alleviation controller block ”gla control synthesis”. Both seek to reduce the wing root bending moment corresponding to manoeuvres and gust encounter. Their structure is predefined with specified inputs and outputs. The pitch angle and rate, the commanded and real vertical acceleration are needed for the manoeuvre load alleviation controller. Based on these measurements it calculates the necessary aileron and elevator deflections. The gust load alleviation controller takes the pitch rate, the vertical acceleration in the fuselage and on both wing tips as an input. It likewise provides aileron and elevator deflections. Both controllers are synthesized based on the structured H_∞ synthesis method with a full order model including unsteady aerodynamics, gust inputs and load outputs. Before the synthesis takes place, the order of the state-space model of the aircraft is reduced removing irrelevant dynamics. As an objective function for the MLA and GLA controller the weighted transfer function from gust input to wing root bending moment has to be reduced.
Output of the RCE blocks are state-space models of the controllers. More controller types, like an active flutter suppression controller, could be synthesized subsequent to the ”trim lin model” block as well. The resulting controller state-space systems can then be fed to a closed loop model in order to analyse the overall aircraft performance.

6 Mission Analysis

The frequency based analysis of the resulting controllers are carried out within the control design blocks. Therefore, the mission analysis can only be started in case all controllers have satisfactory performance and robustness. The mission analysis block takes the controllers and models as the input. The controllers are provided in a Matlab struc file from the control design block, the nonlinear model is given as a Simulink file with the configuration struc file. All the files are handed over via RCE as a compressed folder and are indexed in the CPACS file.

One of the goal of the mission analysis is to minimize the aerodynamic drag. Specifically, the induced drag is addressed by high aspect ratio wing designs. For this the induced drag has to be modeled for the reference aircraft and an optimal wing shape needs to be determined which results in the minimal induced drag.

The second goal is to assess the benefits of the improved aircraft and of the developed controllers. This goal requires a model that is of high fidelity, contains gust inputs and load outputs as well as all four controllers.

To analyse up to which speed it is safe to operate the aircraft, it is necessary to assess the speed at which flutter becomes unstable. For a flutter analysis the nonlinear aircraft model is linearized at several speeds. The poles of the model linearized at the highest speed are analysed at first. As flutter is most likely to be unstable at that speed, the analysis of the unstable poles reveal the flutter poles. Thus the flutter eigenvalues and eigenvectors are determined. By using a mode matching algorithm the flutter mechanism can be tracked for the linearized models with stepwise decreasing speed. With larger speed steps the tracking algorithm is more likely to fail. It is therefore recommended to choose small speed steps. The flutter speed is the speed, at which the flutter poles become stable. For more accuracy the flutter speed can also be estimated by interpolation between the flutter poles at the different flight speeds.

For the overall aircraft performance the aircraft is considered to operate in cruise. The flight conditions within cruise only changes due to defueling. To account for this change in mass a few discrete mass cases of the current aircraft configuration are provided. They represent different fuel levels. The necessary thrust for a mass case is then estimated by means of the overall drag, which is minimised through allocation of the control surfaces. The fuel consumption can be determined based on the required thrust. As soon as a certain level of fuel is consumed, a new mass case representing the predominant fuel level best is chosen. The sum of the distances the aircraft flies per mass case then provides the overall aircraft range.

7 Hardware-in-the loop Tests

The second major toolchain in FLiPASED is the HIL test. The main purpose of the HIL test is to test the controllers running on the FCC - flight control computer. With this simulation, the FCC hardware and the controllers are testable and it can be assessed whether the designed controllers have any implementation limitation and also how they work in a realistic environment. The HIL architecture consists of two main components: a PC that runs the simulation model and the FCC running the control algorithm.

Requirements for the nonlinear model:

- The model is in continuous time and it has to run in real time. The real time capability of the model can be tested by running the simulation in External mode with Simulink Desktop Real-Time option set under menu item Simulation/Model Configuration Parameters/- Code Generation.
- The inputs of the model are the 19 controlled inputs. In addition to these inputs, the GLA controller tests also require the gust inputs.
- The outputs of the model are the sensors that can be used by the controllers. In addition to these outputs, the model also need to contain the loads as output in order to assess the MLA and GLA controllers.

Requirements for the Controller:

- The controller needs to be transformed to discrete time in case it was designed in continuous time. The sampling time is 5ms.
- The controller block designed by each partner has to be a static map between $(U_C, x_c[k])$ and $(Y_C, x_c[k+1])$, where x denotes the state of the controller (see Figure 12.). The input and output signals equal to the output and input of the model, i.e. $Y_C = U_M$ and $U_C = Y_M$.

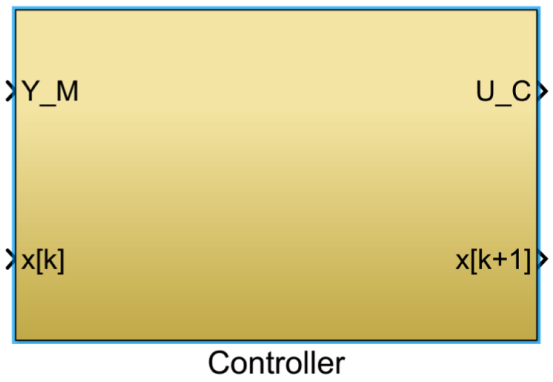

Figure 12: Controller inputs and outputs for the HIL test

The MDO toolchain needs to provide the model and controllers to the HIL environment. The model is provides as a Simulink model with fixed structure and a configuration file that contains all the necessary data of the model. The order of the HIL model is not reduced in order to retain as high fidelity as possible.

The control design blocks of the MDO toolchain provide the baseline, MLA, GLA and flutter suppression controllers in discrete time. All the controllers are given as state space models that are ready to be used for automatic C code generation in order to be uploaded to the FCC.

The HIL tests provide time domain simulation results after evaluating the controllers. The responses are evaluated numerically and based on the evaluation pass/no pass flags are set for each individual controller.

8 Flight Tests

The third major toolchain consists of pilot training, ground tests and flight tests. The controllers and the models are the passed on from the HIL tests which can directly be used for pilot training. Ground tests and flight tests can only be carried out with the physical aircraft and not after each iteration of the MDO toolchain.

The ground testing serves to evaluate the structural properties of the newly developed wing in order to clear its airworthiness and to produce a comprehensive modal model of the aircraft. This modal model can then be used for Finite Element (FE) model updating,flutter calculations and controller updating.

The main goal of the flight test is to validate the maturity of the developed controllers and control design methodologies. The baseline controller has already been validated in the legacy FLEXOP project. Therefore, the main focus in FLiPASED is on the testing of the MLA, GLA and flutter suppression controllers.

The details of the ground and flight test plans are given in deliverable “D1.3 Demonstrator Ground and Flight Test Requirements Definition”.

9 Overall Architecture Evaluation

The goal of this section is to explain the connection between the MDO, HIL test and flight test toolchains.

The MDO toolchain is the main block in this case which has its own optimization and gets back to the CPACS generation block after each iteration. In this tool one of the main focus for the model generation, model reduction blocks and control design blocks are the robustness aspects of the underlying algorithm. These need to run automatically, without human interaction in the presence of changes in the aircraft. This comes at an expense that the individual controllers do not achieve the highest possible performance. The other main goal of the MDO toolchain is to show the improvements of the optimization, which involves aircraft geometry, sizing, modeling and control design simultaneously, with respect to the reference aircraft. The HIL test and flight test block come as an auxiliary tool to validate the developed methodologies.

The HIL tests evaluate the implementation aspects of the controllers and serve as a final step before flight testing the controllers.

The flight test goals are twofold in case of the MDO toolchain. The main goal is to validate the control design technology maturity. This is especially valid for the MLA, GLA and flutter suppression controllers of the project since they not have been flight tested yet (using the model based design methodology within FLiPASED). The other goal of the flight tests, as opposed to the MDO toolchain, is that the resulting controllers can be fine tuned by ”hand” to achieve optimal performance and provide lessons-learnt to the designers and to the aviation community in general. In this case the robustness of the synthesis algorithms to be able to be run in an automatic manner is not of a paramount criterion. In addition, the fine tuning of the controllers is to be done based on the aircraft model that has been updated via flight test data.

At the end of the cycle, the lessons learned from the HIL tests and flight tests need to be fed back to the MDO toolchain. This is done via engineering considerations. If the HIL tests indicate that some controller has implementation difficulties, the corresponding control design algorithm needs to be updated. Similarly, if the flight tests show that a controller has lack of performance or robustness during flight tests, the algorithms need to be adjusted as well.

10 Conclusion

The main output of the deliverable is the definition of the functional division, data flow and specific data types exchanged among the partners in the collaborative design. This is especially important in case of the types of models generated throughout the workflow since one of the key goals of the project is to reduce the overall number of models in the development. The other main result is to connect the ”lessons learned” from HIL and flight test to the MDO toolchain to be able to update the underlying algorithms if required.

The MDO tools are being integrated into the RCE framework by the respective tool owners based on the interface definitions laid out in the deliverable. Once the integration is finished the MDO toolchain will be tested and fine-tuned by the consortium. The present workflow is developed for the demonstrator, but the overall methodology is almost the same for the scale-up task within WP4. The main difference comes from the objective function and the inner convergence loop for structural sizing - aeroelastic tailoring, what is not present in the demonstrator workflow, where only a structural check is established.

11 Bibliography

[1] Tamás Luspay, Daniel Ossmann, Matthias Wuestenhagen, Dániel Teubl, Tamás Baár, Manuel Pusch, Thiemo M. Kier, Sergio Waitman, Andrea Ianelli, Andres Marcos, Balint Vanek, and Mark H. Lowenberg. Flight control design for a highly flexible flutter demonstrator. In *AIAA Scitech 2019 Forum*. AIAA, jan 2019.

[2] Bálint Patartics, György Lipták, Tamás Luspay, Peter Seiler, Béla Takarics, and Bálint Vanek. Application of structured robust synthesis for flexible aircraft flutter suppression. *IEEE Transactions on Control Systems Technology*, pages 1–15, 2021.

[3] Bela Takarics, Balint Vanek, Aditya Kotikalpudi, and Peter Seiler. Flight control oriented bottom-up nonlinear modeling of aeroelastic vehicles. In *2018 IEEE Aerospace Conference*. IEEE, mar 2018.

[4] Béla Takarics, Bálint Patartics, Tamás Luspay, Balint Vanek, Christian Roessler, Julius Bartasevicius, Sebastian J. Koeberle, Mirko Hornung, Daniel Teubl, Manuel Pusch, Matthias Wustenhagen, Thiemo M. Kier, Gertjan Looye, Péter Bauer, Yasser M. Meddaikar, Sergio Waitman, and Andres Marcos. *Active Flutter Mitigation Testing on the FLEXOP Demonstrator Aircraft*.