ULTRAFLOADS – A FRAMEWORK FOR HIGH-FIDELITY LOADS COMPUTATIONS

J. Feldwisch*, L. Reimer†, M. Ritter*
* German Aerospace Center DLR, Institute of Aeroelasticity, Göttingen, Germany
† German Aerospace Center DLR, Institute of Aerodynamics and Flow Technology, Braunschweig, Germany

Abstract

High-fidelity simulations play a key role for aircraft design, virtual flight testing and simulation based certification. As computational resources increase and numerical methods progress, highly detailed geometries can be investigated and complex multi-physics models can be applied. Research projects of the past decades have demonstrated the capabilities towards virtual design and virtual testing of aircraft. To channel the experience and software developments from the past years, a new software framework for high-fidelity, multidisciplinary analyses with a focus on loads computation has been developed. The backbone is the FlowSimulator and FSDataManager (FSDM) jointly developed by DLR, Airbus and ONERA. The FSDM provides a comprehensive HPC library for the management of multiple parallel processes and data objects. This work focuses on the architecture of the new loads analysis software, which features a base class for all analyses, a centralized data model and a registry to inject user-defined scenarios. The capabilities are demonstrated by comparing a steady turn maneuver of a free-flying, fully elastic transport aircraft to real flight test data, showing good agreement for the resulting displacements.

Keywords

Loads, Software, Framework, CFD, CSM, MDA, Aeroelasticity

1. INTRODUCTION

High-fidelity methods for gust and maneuver loads computation enrich modern aircraft design, as less assumptions like inviscid, subsonic flow have to be made. Compared to industrialized lower fidelity methods, those simulations quickly become complex, time consuming and therefore need to be solved in a multiprocessing environment on high performance computers (HPC).

In the past decade a prototypical code has demonstrated and progressed the capability of high-fidelity loads simulation [1,2]. Now, this predecessor is being replaced by a new software for high-fidelity loads analysis within DLR. The software must be flexible as modifications and new methods are developed or investigated in a scientific research environment. The requirements for the new loads framework comprise multiple sets of simulations like trimmed steady maneuver loads, unsteady maneuvers and gust encounters. For all of those scenarios, aerodynamic solvers of various fidelity levels, trim algorithms and coupling strategies should be available. Also, researchers frequently need to implement new methods to answer new questions. Thus prototypical programming must be possible using the framework. As multiple levels of fidelities are requested for loads computation, the name of the framework is ultra fidelity loads or short UltraFLoads. The framework has already been used for high-fidelity loads computation for quasi-steady pull-up and roll-maneuvers [3].

The solver for high-fidelity aerodynamic simulations is currently the DLR TAU code [4]. In the near future the solver will be replaced by the next-generation CFD solver CODA, which descends from DLR's solver prototype implementation Flucs [5]. CODA is currently being diligently developed in a joint effort by DLR, Airbus and ONERA. The FlowSimulator DataManager (FSDM) [6], jointly developed by DLR, Airbus and ONERA, provides an HPC library for CFD-based simulation workflows, data models, data manipulation and multiprocessing. As the core library includes Python interfaces, additional convenient control layers (FSDLRControl) have been developed to organize data and workflows in Python. Those layers simplify the volume mesh deformation, interaction with TAU, MSC.Nastran and many more. As the FSDM is highly modularized and a centralized data model is available, each user or researcher can combine the data and methods individually for their needs. This was used for example in a proposed approach by Backhaus et al. [7], where the optimization capabilities of OpenMDAO[1] [8] are combined with the FSDM.

With a focus on multidisciplinary analysis, UltraFLoads depends heavily on the FSDM and

[1] https://openmdao.org/

doi: 10.25967/550264

FSDLRControl to increase code robustness and reduce duplicate, additional work. As all of the simulations can be interpreted as some kind of analysis, object oriented programming is used for all analysis classes. There are abstract analysis classes to solve explicit or implicit problems. All analysis classes are based on the explicit analysis class.
This paper provides an architecture documentation of UltraFLoads, containing the functional and user requirements, the architecture constraints, the architecture solution, and implementation details of the generic scenarios. A demonstration is provided for a fully elastic, free flying aircraft in a steady 2-g turn maneuver.

2. REQUIREMENTS

The requirements for the new framework were derived from user stories of the type: *"As a ...., I want ...., such that ..."*. The stories were written by the scientists at the department of Loads and Aeroelastic Design and then categorized. The functional requirements list all the analyses and simulations the user wants to run. Further expectations towards input/output and process management are combined in the user expectations. Furthermore, constraints, the research context and the technical context have to be considered for the new framework.

2.1. Functional

The main objective is to compute loads using high-fidelity methods. The aerodynamic, inertia, external and other loads need to be distinct from each other and a cut-loads computation should be available. The loads should be provided for steady maneuvers, dynamic maneuvers and gust encounters in the time domain. Also, flight control should be considered for the dynamic simulations.
Furthermore, different aerodynamic solvers like TAU [4] and CODA [5] should be combined with structural solvers like MSC.Nastran, B2000++[2] or a modal approach. Analyses of free-flying, flexible aircraft require rigid body flight mechanics and flight dynamics. A trim algorithm of arbitrary trim targets and any trim control variables should be available, which is required for all quasi-steady maneuver simulations.
Additionally, unsteady gust encounters and maneuver simulations should be possible with the new framework.

2.2. User expectations

Based on the user stories, additional expectations about the framework can be summarized. Only one input file should be needed, which is simple to setup and for which input checks are conducted.
Furthermore, as the simulation may take several hours to days, live monitoring of the analyses should be available, to check for example the convergence of the fluid-structure coupling or the current actuator settings.
Also, restarting from a previous simulation should be enabled in order to reduce computational effort.
Additionally, it was excepted that the scenarios should be modifiable and that own implementations can be integrated.

2.3. Constraints

The framework is constrained by the usage of the FlowSimulator and FSDataManager (FSDM).
The FSDM is jointly developed by Airbus, DLR, Onera and other research institutes with the goal to provide a high-performance computing library for high-fidelity simulations [9, 10].
For example the flow solver TAU, the next flow solver CODA [5], mesh deformation plugins, and other high-fidelity related developments are integrated or will be integrated in the FlowSimulator environment. Thus, a lot of plugins can be used and less new implementations are necessary. As the number of users has been increasing constantly over the recent years, the different plugins are tested more intensively and the software is developed carefully.

2.4. Research context

The new framework is a research software for a research environment. Hence, it needs to be considered that new questions arise which may require new implementations or modifications to the already existing code.
Also, code prototypes may become necessary to quickly investigate a new method, which first must prove to be useful. Hence, modifications and prototyping should be enabled by the new code. Instead of modifying the code directly, the modifications should take place only at run-time. This allows some freedom to the researcher to adjust analyses to the current project while maintaining a stable code base.

2.5. Technical context

To use already implemented plugins and being able to include future developments, the FSDM ecosystem must be used. With the *FSDataManager*, multiple data objects can be centrally stored, allowing for a simple modularization of code. The most important data objects for the new framework are the *FSMesh* objects and the *FSRelationsModel*.

2.5.1. Mesh objects

The *FSMesh* object is a data container for distributed meshes. Different mesh-types can be imported and exported for structured and unstructured meshes.
Different element types are available for which cell attributes and datasets can be stored. For example, cell attributes can be used to define components, which is useful for coupling, or boundary markers, which help to identify solid walls in a volumetric mesh.

[2] https://www.smr.ch/products/b2000/

As all meshes are *FSMesh*-objects, different mesh partitioning algorithms can be used and general mesh deformation techniques can be applied.
Visualization is simply achieved by built-in export methods to VTK or Tecplot formats. Thus, the simulation results can be visualized immediately, without additional postprocessing.

2.5.2. The relations model

The *FSRelationsModel* is intended to provide relations between models and to hold simulation parameters. It can be received from the *FSDataManager* instance and is always synchronized over all processes. This allows to access the relations and the parameters anywhere in the process, and enables a centralized data approach for light-weight meta data.
It is a hierarchic data model, similar to an XML format, with nodes, sub-nodes and elements like attributes, parameters, *FSDataLog* and IDs. The relations model can be imported from and exported to an XML file. Thus, exporting the relations model allows to monitor the current state of any simulation or using it as a restart file.
Simulation settings or input can be realized by attributes or parameters. Monitoring of data is accomplished via the *FSDataLog*, which may hold multiple categories with again multiple quantities, where each quantity can be a multidimensional array.

2.5.3. The layers

The FlowSimulator ecosystem can be seen as different layers, where the data-objects are on the lowest level and the fully defined scenarios are on the upper level, as shown in Figure 1. Data-objects like the *FSArray*, *FSDataSet*, *FSMesh*, *FSRelationsModel* build the backbone, but they may not be convenient to interact with for the upper-level user. Thus, different plugins are used to manage the data and different features like the mesh deformation. On top of that, the convenient control layers can be used for easier access and execution of for example the TAU solver. Here, it is only expected that the user has some basic Python knowledge to work with the convenient control classes. UltraFLoads aims at the upper level, to harmonize different control classes in general scenarios. Thus, the script does not require too much knowledge about the FSDM and the control classes. However, when it comes to further developing the code or deviating slightly, it quickly becomes necessary to know all the different layers.

3. ARCHITECTURE

With the *FSDataManager* a centralized data model is available, allowing to split the data-flow from the process flow and making modularization easier. This core idea is used for UltraFLoads as well by using FSMesh objects for all models, which can be interpreted as some discrete, three-dimensional object.

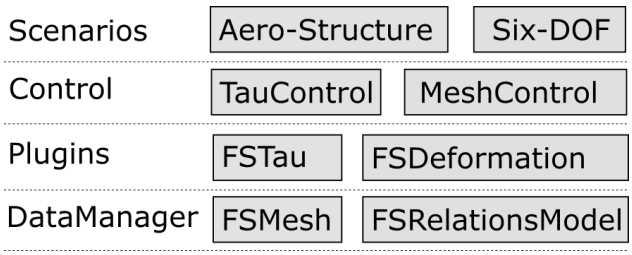

FIG 1. The different layers of the FSDM. Starting from the data objects on the lowest level, the user needs less and less expertise with the FSDM going up to the scenario level.

So *FSMesh* objects have to exist for all structural models, coupling models , loads model, etc.
Furthermore, the *FSRelationsModel* is used for all meta information. The hierarchy, node names, attribute names, parameter names etc. are stored in relations interface classes. Therefore analysis classes receive their input and store their output in the *FSRelationsModel* via the relations interface classes.
On the downside, most of the packages or classes of the FlowSimulator do not use the *FSRelationsModel* yet and hence there is no definition for them. Thus, the input parameters for those classes and functions have to be defined by UltraFLoads. This might change, once the *FSRelationsModel* is utilized for those plug-ins as well.

3.1. Relations-model interface

As the *FSRelationsModel* provides a hierarchic data structure with different elements, the structure and the names have to be defined.
This structure is defined by each relations interface class, which inherits from a base interface class. The *FSRelationsModel* can be accessed from the *FSDataManager* and therefore, each interface object can be instantiated using the *FSDataManager* only. Each interface class defines a domain node, to which all other parameters, attributes, *FSDataLog* are stored.
A simple rule of one task, one node, one relations interface class is followed. Thus, there is one class for the flight point definition, one for the trim analysis, one for general information about the vehicle and so on.
Monitoring of any scalar quantity is achieved by filling an *FSDataLog*, stored below each domain node. Which monitoring variable to be stored has to be explicitly requested by the user. All interface classes have access to an object, which centrally stores iteration numbers for any analysis. When monitoring variables are written, all up-to-date iteration numbers are stored as well. This is necessary, as the different analysis may be executed multiple times.

3.2. Analysis base class

For UltraFLoads, every simulation, workflow and analysis share some common behavior like incrementing iteration numbers or triggering a monitoring event. Therefore, those code blocks can inherit from one slim base class, the *AnalysisExplicit*, which is demonstrated in Figure 2. The base class provides public interfaces like *preprocess*, *process*, *postprocess*, *exportModels*, *chainrun*, *getVar*, *registerAnalysis*, *registerMonitoring*, *registerPostProcessingEvent*, *getRelationsInterface* and *fromRelations*. Also, an implicit analysis class exists, which is based on the explicit analysis as well. Note, the implicit, explicit analysis idea is based on the explicit and implicit component idea of the open multidisciplinary analysis and optimization framework OpenMDAO [8].

3.2.1. Using the relations interface

The relations interface object is constructed and stored as an instance variable, when the analysis class is initialized. This object is used to increment the iteration number and do the monitoring. Which relations interface class to use is defined by the *getRelationsInterface* class method, which can be overwritten by any child class.
When initializing the analysis object based on the information stored in the *FSRelationsModel*, the class method *fromRelations* is used, which first collects the input for the analysis, initializes it and then returns the new instance as outlined in Algorithm 1.

Algorithm 1 Usage of the class method **fromRelations(dataManager)**

```
1: RelClass = cls.getRelationsInterfaceClass()
2: rel = RelClass(dataManager)
3: arguments = rel.getInput()
4: return cls(arguments)
```

3.2.2. Process flow

Each analysis instance can be executed by the instance method *chainrun*, which simply calls consecutively *preprocess*, *process* and *postprocess*. The *preprocess* calls the private method *_preprocess*, which can be defined by a child class. A call to *process* increments the iteration number and then executes the private method *_process*. This private method has to be overwritten by any child class, otherwise the initialization fails. Similarly, the *postprocess* calls a private method, which may be overwritten by any child class, then triggers a monitoring event, if requested stores the *FSRelationsModel* as XML and or calls the *exportModels* and finally triggers further postprocessing events.

3.2.3. Variables

Variables are just quantities which may be required by an analysis or are just interesting to be monitored. Calling the method *getVar* with the variable name as input, will first check whether an up-to-date value can be found in the monitoring log. If not available, the variable is computed using a dictionary which maps the variable name to the corresponding method. Otherwise the variable is received from the sub-analyses by calling *getVar*.

3.2.4. Analysis registry

The analysis registry is a core component of each analysis class, as it is used to identify other analysis classes, which may be used. For example the fluid-structure analysis may combine different aerodynamic and structural solvers. The registry is realized by a dictionary which maps analysis keys to analysis classes. As the registry is stored as a class variable, it is shared by all instances and can be accessed or modified easily.
Users can register own analysis classes to each analysis registry, allowing for quick modifications of already existing analysis classes. This way, a user can define an own, new analysis, register it and still use all the other analyses classes. The new analysis has to be a child of the base class as well. The benefit is that the modification only exists at run-time and the source code remains untouched. This also simplifies the automatic testing, as dummy analysis can be registered as well.
In combination with the class method *fromRelations*, the registry can be used to chain the analyses automatically, visualized by Algorithm 2. The name of the sub-analysis to be used is stored in the corresponding domain of the *FSRelationsModel*. Then, the analysis name is used to receive the class of the sub-analysis from the registry. Now, the analysis can be instantiated by calling the *fromRelations* method of that sub-analysis.

Algorithm 2 Using the **analysis registry** in the method **fromRelations(dataManager)**

```
1: RelClass = cls.getRelationsInterfaceClass()
2: rel = RelClass(dataManager)
3: analysisName = rel.analysisName
4: Analysis = cls.fromRegistry(analysisName)
5: analysis = Analysis.fromRelations(dataManager)
6: return cls(analysis)
```

An example of the analysis registries and the inheritance is shown in Figure 2 by the green boxes. Following the green arrows, it becomes apparent that each analysis may be in an analysis registry of another analysis. For example the *SixDOF* analysis, which allows for rigid-body flight dynamic simulation of the free-flying aircraft, may use the *TAU* analysis or the *SteadyFSI* analysis. The SixDOF class itself is then part of the registry of the *TrimAny* class to enable the trimming of the free-flying aircraft.

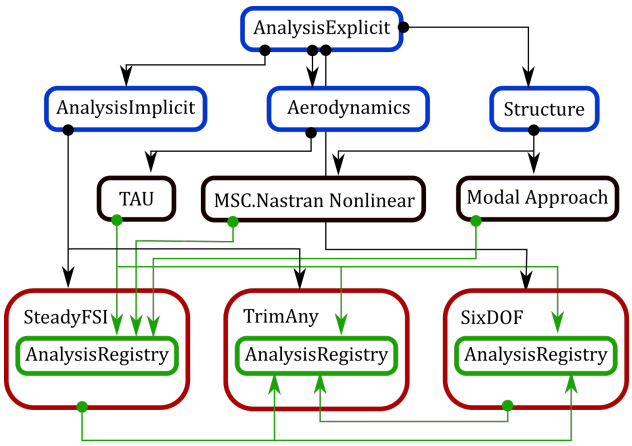

FIG 2. Inheritance diagram of the abstract analysis in blue and the fully defined child classes in black and red. The analysis registries are visualized as green boxes, which may hold other analyses definitions.

3.3. Implicit analysis class

Another analysis class provides solvers for implicit problems. The solvers are damped Gauss-Newton solvers which can be used for trimming and a sequential solver, applicable to loosely coupled fluid-structure simulations.
The implicit analysis has the methods *getX*, *setX*, *getY* and *setY*, to build simple input-output models. The child classes may also provide a way to compute the jacobian, which is done using forward finite differences otherwise. Alternatively, the jacobian can be computed using a good or bad Broyden approach. It is up to the user to decide whether or not to freeze the jacobian or allow for re-computation.
If activated, the input variables can be relaxed by a predefined sequence of relaxation factors, or the relaxation may be based on an Aitken relaxation method.

3.4. Input

As all necessary information for the analyses are stored in the *FSRelationsModel*, it plays a central role for UltraFLoads. Thus, to use all the features and analyses, the *FSRelationsModel* has to be set up correctly, meaning the names and hierarchy of nodes, attributes, parameters have to be known.
One possibility is to load the *FSRelationsModel* from an XML-file, which is shifting the problem towards the XML file creation. As the XML format is not particularly user friendly, this option is only considered for restarting from previous simulations.
Another approach is to use the interface objects for setting up the *FSRelationsModel*, which requires knowledge about UltraFLoads. With each interface object, the data can be written without knowledge about the structure of the *FSRelationsModel*.
A third way is a simplified input file in the lightweight, text-based JSON-format [11]. Also, a schema is available for UltraFloads based on the JSON-schema[3] [12], enabling simple auto-completion, dependency checks and data type checks. Thanks to the Python package jsonschema[4], the input is validated at the start of the simulation. Then, the input is translated to the *FSRelationsModel* using the interface objects.

4. IMPLEMENTATIONS

4.1. Nodal based mesh interpolation

Interpolation from one mesh to another is necessary for the coupling of the different disciplines. Forces from the aerodynamic surface mesh need to be transferred to the structural mesh and displacements from the structural to the aerodynamic mesh.
Based on element attributes of each *FSMesh* object, the interpolation can be split into different pairings to ensure that for example loads from the left wing are transferred to the respective wing structure only and not to the fuselage as well. For each coupling pair, a different interpolation method can be selected. Different radial basis functions and nearest neighbor methods are available and the user can register own interpolation functions.

4.2. Meta models

In order to not mix different disciplines with each other, meta models are used for storing loads and displacements. The structural analysis receives the loads from a loads model and not from an aerodynamic analysis. This way, the structural solver does not need to have access to the aerodynamic model. The same applies for the aerodynamic analysis with respect to the displacements. Also, it allows to define artificial external loads, which may act on the structure only.
Furthermore, the loads model is equipped with features to integrate the forces and moments along the loads-reference axis.

4.3. Actuators and controls

Actuators are used to change the state of an moving object. Depending on the target state, multiple actuators may be applied to reach it. The state of each actuator is logged in an *FSDataLog* to monitor the history of the actuators state.
Multiple actuator types are available, which either manipulate the model directly or just change the input of an analysis. Aerodynamic control surfaces and control planes can be selected, which then alter the volume mesh for the CFD simulation. At the moment only the blended, mesh-morphing approaches for control surfaces [6] can be selected; Chimera mesh-movement is not considered yet.
Furthermore, external force distributions can be related to a set nodes of the loads-meta model to mimic

[3] https://json-schema.org/
[4] https://python-jsonschema.readthedocs.io/en/stable/

a horizontal tail plane for example. Additionally, artificial forces can be applied to the flight mechanic model to account for a thrust force.
As the Euler-angles, translatory and rotatory speeds are input parameters for the simulation, they can be treated as actuators as well.
Control variables are then defined, which may affect multiple actuators simultaneously. Each actuator may be scaled by a different participation factor, which may change over time. Those control variables can then be used for a flight-control system, which has not been implemented yet, or for a trim algorithm.
Also, each actuator state and control variable state may have different upper and lower limits. All those information are light-weight enough to be stored in the *FSRelationsModel*.

4.4. Modal projection

For small deformations, the structural degrees of freedom u relate linearly to the applied forces F by the stiffness matrix K.

$$Ku = F \tag{1}$$

However, solving this equation for a large number of degrees of freedom and different element-types may require a finite element analysis with an external solver. Instead, the degrees of freedom can be reduced, by just using a small set of n structural modes of the model. With $u \approx \sum_i^n \phi_i q_i$, the displacements are approximated by scaling the eigenvectors ϕ with the generalized coordinates q.

$$\Omega^2 q = \Phi^T F \tag{2}$$

Then, only the eigenvalues Ω^2 and eigenvectors ϕ need to be available at run time. This method is especially beneficial for the free-flying vehicle as shown in the work of [13]. Also, the FEM solver is only needed once in a model-preparation phase to set up the modal base. Here the user has to decide which structural modes should be included.

4.5. Fluid-Structure interaction

The available structural and aerodynamic analyses are stored in the analysis registry of the *SteadyFSI* class. The generic fluid-structure interaction solver only weakly couples the two disciplines, by executing the aerodynamic solver, then transferring the aerodynamic forces via the loads model to the structural mesh. Then, the structural solver is executed and the displacements are interpolated over the displacement model to the aerodynamic surface mesh. The problem is solved if the residual of the forces and the residual of the displacements are small enough. Note, it is up to the user how to set up the aerodynamic solvers and the structural solvers. At the moment, the available structural solvers are the modal approach as described in Section 4.4, the linear and the nonlinear solvers of MSC.Nastran. It is planned to include the B2000++ solver in the near future. For the aerodynamic solvers, only TAU is available at the moment. However it is planned to include a low-fidelity, potential flow based solver and the new CODA solver as well.

4.6. Rigid body motion

For the free flying aircraft the six degrees of freedom of the rigid body motion are based on the equations for forces (Eq. 3) and moments (Eq. 4). Forces F and moments L in the body reference frame $_b$ are evaluated at the center of gravity. The mass matrix M and the moment of inertia tensor J are computed at the center of gravity as well. The degrees of freedom for the translatory motion u and rotatory motion Ω contribute by the gyroscopic terms, leading to a nonlinear differential equations. Further details can be found in [14].

$$M\dot{u}_b = (Mu_b) \times \Omega_b + F_b^{aero} + F_b^{ext} + F_b^{grav} \tag{3}$$

$$J\dot{\Omega}_b = (J\Omega_b) \times \Omega_b + L_b^{aero} + L_b^{ext} \tag{4}$$

Note, external forces can be defined like artificial loads to mimic the engine's thrust or an artificial lift distribution for a horizontal tail plane.
As the load factor plays an important role for maneuver simulations, it is calculated by the rigid body acceleration, the centrifugal acceleration and the gravitational acceleration.

$$n = \frac{1}{g}\left(\dot{u}_b - \Omega \times u_b + \ddot{z}_b^{\text{grav}}\right) \tag{5}$$

4.7. Flight dynamics

For the flight dynamics analysis, the equations 3 and 4 are rewritten in a state-space form. The system of equations is nonlinear due to the terms $(Mu_b) \times \Omega_b$ and $(J\Omega_b) \times \Omega_b$. The time integration is achieved by the linear multistep methods Adams-Moulton or Adams-Bashforth [15], where the implicit Adams-Bashforth method may use the explicit Adams-Moulton method in a predictor step. Furthermore, applying the mean-axis condition to the fully-elastic, free-flying aircraft, the rigid body degrees of freedom and the elastic degrees of freedom can be decoupled [16].

4.8. Trim

For several quasi-steady maneuvers, a target state must be reached in order to asses the correct loads. For example, the load factor is prescribed and the pitch acceleration should be zero. Those two conditions constrain the lift distribution on the wing and therefore the aerodynamic loads. The target state is reached by altering trim variables, which have to be

determined during the trim analysis. Thus, the implicit problem can be formulated as a root-finding problem, or interpreted as a minimization problem [17].
The trim analysis class inherits from the *AnalysisImplicit* and therefore the implemented solvers can be selected.
As all relevant parameters and input values are stored in the *FSRelationsModel*, the trim analysis only needs to modify it. Therefore, the control variables (section 4.3) can be selected as trim variables, which will then alter several actuators. Those actuators then influence the different analysis blocks.
The states of the trim targets are received through the public method *getVar* (section 3.2.3) of the trim analysis object, enabling setting up trim tasks with arbitrary trim targets.
The available analyses in the registry are so far, the *TAU* analysis class, the *SteadyFSI* class and the *SixDOF* analysis class. Depending on the analysis class, different target states must be defined, as each analysis class can only return the values for variables, which are stored in the variable map or can be found in a sub-analysis.
For example, having selected the *TAU* class, only C_L, C_D, C_{my} etc. are possible candidates for trimming. If a load-factor is requested, then the *SixDOF* class needs to be selected, as the load-factor is related to flight mechanics and not pure aerodynamics. Of cause, if the aerodynamic class *TAU* was the sub analysis of the *SixDOF* analysis, the variables C_L, C_{my} can then be used for defining the trim target state as well.

5. RISKS

All the advantages of the FSDM, the plugins and the different control classes come at the cost of dependencies. Smaller to larger changes to the different codes, may affect UltraFLoads and could break it. This risk is minimized as all the release notes are available and planned changes are communicated.
Even though the centralized data approach is advantageous for modularization, it poses a risk, as data has to be implicitly available and must be up to date. It is more difficult to read when, where and how the data is modified. However, standard Python optimized IDEs and code editors help and the documentation of the code and its architecture as well.
Another risk poses the analysis base class, as all other analysis classes inherit from it. Modifications may cause unforeseen behavior of the child classes. One answer to that is testing and keeping the base class as slim as possible. Using continuous integration and continuous deployment together with a git-flow branching model further reduces risk of unexpected behavior.
Although software is the backbone of every simulation-based research department, the main task is still the actual analysis. Hence, software development remains a task of few people in a research environment and users may need to become developers at some point to ensure continuity. This can be achieved by training, documentation and example test cases.
Training with Python and the FSDM is necessary for the scientific usage of UltraFLoads. For simpler tasks, the pre-defined workflows, which just rely on the analysis registries, can just be used. Here the acceptance is improved by a user manual and a guided, simplified user input.

6. STEADY TURN MANEUVER

The steady turn maneuver, visualized in Figure 3, is a good testcase for the trim method, because it can be flown steadily by a pilot. In order to push virtual flight testing as a tool for simulation based certification, different flight tests, including steady turn maneuvers, have been performed with the DLR ATRA[5] aircraft in the projects VicToria[6], SimBaCon[7] and VitAM. Also, the simulation capabilities for the virtual aircraft have been progressed as shown by [6] and [2].

6.1. Flight mechanics

The goal is a trimmed, horizontal, slip free, coordinated turn with a load factor of $n_z = 2$. This is a standard text-book scenario, relating the bank angle with the loadfactor $n_z = 1/\cos(\phi_b)$. However, this angle may not be mistaken with the Euler-angle ϕ. While trimming the free variable θ, the other two angles ϕ and ψ are constrained by the slip-free condition and the target loadfactor. Thus, all three Euler angles are different from zero [17].
The other trim variables are aileron deflection, rudder deflection and the rotation angle of the complete horizontal tail plane. The target state is that a load factor of $n_z = 2$ is reached and that all rotatory accelerations are zero. Note, this does not force the longitudinal acceleration u_b^x to be zero as no thrust from an engine is considered.

6.2. Simulation

For this scenario, a hybrid mesh with 11.6×10^6 nodes, and 19×10^6 prism cells was used. The airflow is modeled by the RANS equations, using the negative Spalart-Allmaras turbulence model [18]. The equations are solved by the DLR-TAU code [4].
The Mach number is 0.7; the true airspeed is $222\,\mathrm{m\,s^{-1}}$. In order to match the dynamic pressure of the experiment, the altitude was altered.
The structural model developed for the project VicToria [2] is reused and the modal approach is selected for the analysis.

[5] https://www.dlr.de/content/de/artikel/luftfahrt/forschungsflotte-infrastruktur/dlr-flugzeugflotte/airbus-a320-232-d-atra.html
[6] https://www.dlr.de/as/desktopdefault.aspx/tabid-11460/20078_read-47033/
[7] https://www.dlr.de/as/desktopdefault.aspx/tabid-13016/22740_read-52834/

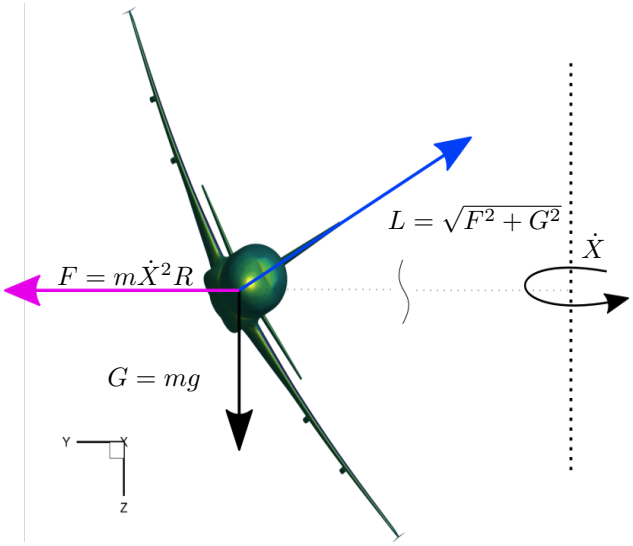

FIG 3. Forces acting on the aircraft in a coordinated steady turn maneuver.

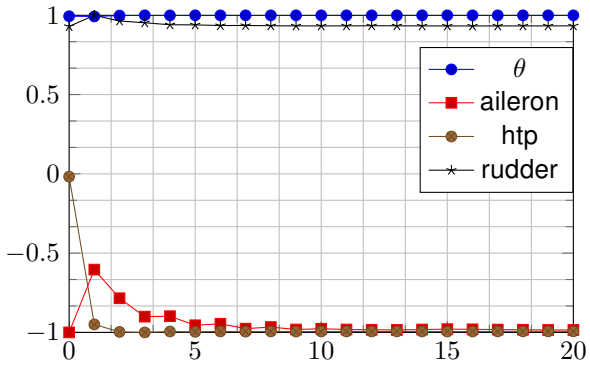

FIG 4. Convergence history of the control variables used for trimming the aircraft. The variables are normalized by their maximum value.

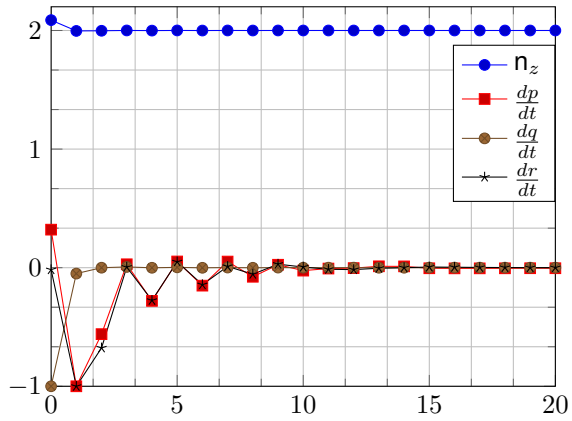

FIG 5. Convergence history of the load factor and the rotational accelerations.

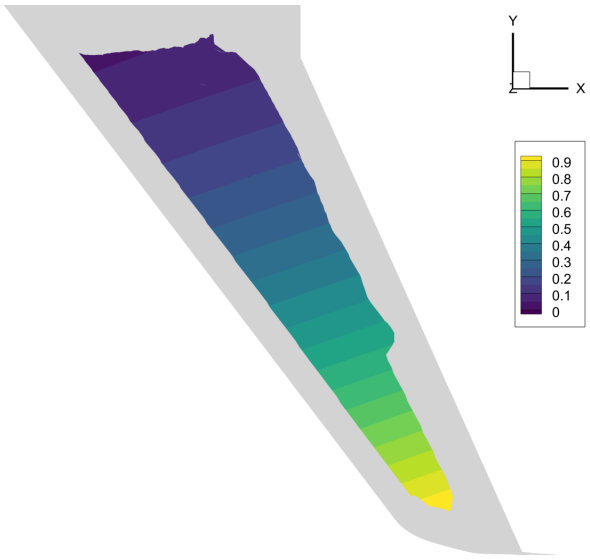

FIG 6. Magnitude of the simulated displacements.

Just altering the pitch angle θ in the *FSRelations-Model* does not consider that the other angles ϕ and ψ have to be adjusted for the slip-free condition and the target load factor. Therefore, the trim class has been modified on the fly in order to adjust for the steady turn maneuver. This modification only applies to the *setX* method, in which the other two Euler angles are adjusted depending on the pitch angle.
The problem is trimmed using a damped Gauss-Newton method with a frozen jacobian matrix. The converged trim history is shown for the different variables in Figure 4 and the trim states are depicted in Figure 5.

6.3. Comparison to flight test

For the flight test, a stereoscopic camera setup was used to reconstruct a deformation field based on a patterned foil on the wing applying the image pattern correlation technique [19]. The magnitude of the displacements is shown in Figure 6 for the simulation and in Figure 7 for the flight test. From the contour plots, the agreement between both results appear to be perfect on first sight. The difference between both plots, normalized by the simulation results is shown in Figure 8. It reveals that the error in displacement magnitude is between $5.0\,\%$ and $6.5\,\%$.
Even though the simulation results are in good agreement with the flight test data, it has to be pointed out that there are multiple uncertainties of the simulation models but also in the flight test data. The structural model has not been fully adjusted to the actual aircraft structure. Furthermore, the aerodynamic model does not include the engine's thrust, and the flow around the wing is not affected by the nacelle. Also, the mass distribution of the finite element model does not match the one of the actual aircraft perfectly. Furthermore, the total weight, the moments of inertia and the center of gravity of the DLR-ATRA have only been estimated for the flight tests. So it is likely, that the assumed mass-matrix for the simulation deviates from the mass-matrix of the actual flying aircraft.

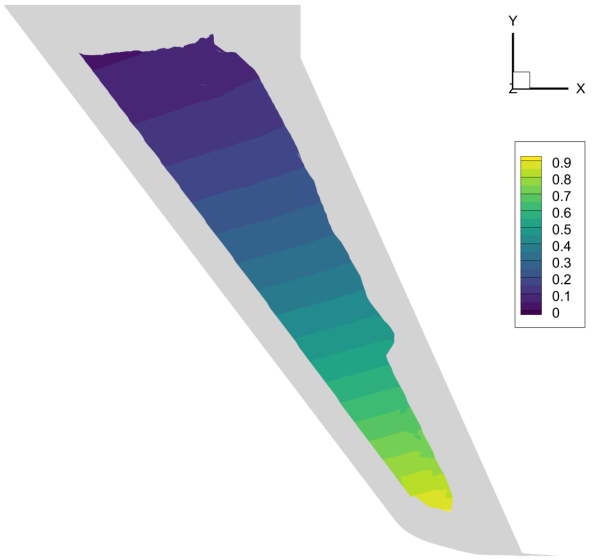

FIG 7. Magnitude of the displacements from the flight test.

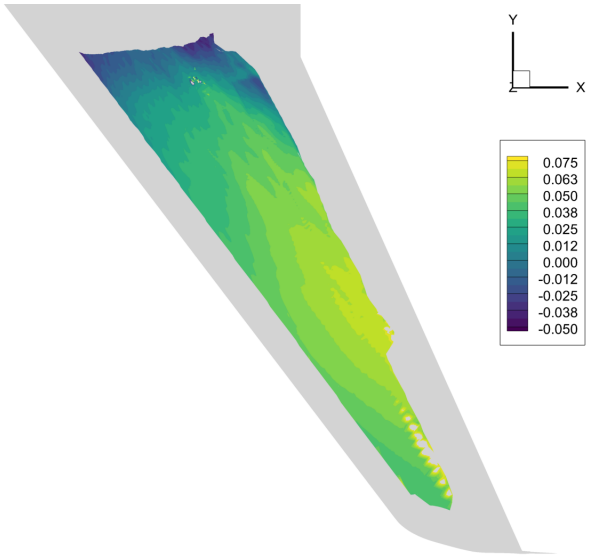

FIG 8. Difference between simulated and experimental results, normed by the simulation results

7. CONCLUSION

Concluding, the requirements, the architecture solution and some implementation details of the loads framework UltraFLoads have been presented.
A centralized data approach is followed using the FSDM and the *FSRelationsModel*, which can be interacted with using one interface class per task. All analyses can be instantiated anywhere in the process, as they receive their relevant input from the centrally stored *FSRelationsModel* via interface classes. This enables constructing analysis instances from an analysis registry, which can be modified by the user to integrate own implementations.
All analysis classes are derived from a single abstract analysis class. Each analysis class provides an analysis registry, from which sub-analyses can be selected. Also, users can inject new analysis prototypes into the different registries in order to test new analysis methods.
As a testcase, demonstrating the framework's capabilities, a steady turn maneuver of a free-flying, elastic, transport aircraft was discussed. Good agreement with the flight test data was achieved, even though some uncertainties exist.

8. OUTLOOK

In the next phase, the new flow solver CODA and the structural solver B2000++ will be integrated.
Investigations of maneuvers at the border of the envelope will be intensified to evaluate the application of viscous CFD applications for structural design.
Then, virtual and real flight tests with DLR's research aircraft iSTAR[8] will be performed, for which model and operational uncertainties may be reduced.
Furthermore, work about geometrically nonlinear structural models, coupled with nonlinear aerodynamics is ongoing and will be presented soon.

Contact address:

johan.feldwisch@dlr.de

References

[1] Markus Ritter, Mathias S. Roeser, and Lars Reimer. CFD-based multi-axis maneuver simulation for system identification of flexible transport aircraft. In *AIAA Scitech 2020 Forum*. American Institute of Aeronautics and Astronautics, jan 2020. DOI: 10.2514/6.2020-2125.

[2] Markus Ritter, Vega Handojo, Wolf Krueger, Mathias S. Roeser, Wulf Mönnich, Tania Kirmse, Lars Reimer, Ralf Heinrich, Sven Geisbauer, and Stefan Goertz. Virtual aircraft technology integration platform: From virtual flight testing towards simulation-based certification. In *AIAA Scitech 2021 Forum*. American Insti-

[8] https://www.dlr.de/content/de/bilder/2018/2/forschungsflugzeug-istar_30302.html

tute of Aeronautics and Astronautics, jan 2021. DOI: 10.2514/6.2021-1203.

[3] Johan Moritz Feldwisch and Matthias Schulze. High-fidelity aeroelastic loads calculation for a transport aircraft configuration including pitch and roll maneuvers. In Andreas Dillmann, Gerd Heller, Ewald Krämer, and Claus Wagner, editors, *New Results in Numerical and Experimental Fluid Mechanics XIII*, pages 527–536, Cham, 2021. Springer International Publishing.

[4] Dieter Schwamborn, Thomas Gerhold, and Ralf Heinrich. The dlr tau-code: Recent applications in research and industry. In *ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics, Egmond aan Zee*, Egmond aan Zee, The Netherlands, September 2006.

[5] Tobias Leicht, Jens Jägersküpper, Daniel Vollmer, Axel Schwöppe, Ralf Hartmann, Jens Fiedler, and Tobias Schlauch. Digital-x: Next generation cfd solver flucs. In *Deutscher Luft- und Raumfahrtkongress 2016*, Februar 2016.

[6] Lars Reimer, Ralf Heinrich, Sven Geisbauer, Tobias Leicht, Stefan Görtz, Markus Raimund Ritter, and Andreas Krumbein. Virtual aircraft technology integration platform: Ingredients for multidisciplinary simulation and virtual flight testing. In *AIAA SciTech Forum*, Januar 2021.

[7] Thomas Backhaus, Sebastian Gottfried, Andrei Merle, John T. Hwang, and Arthur Stueck. Modularization of high-fidelity static aeroelastic MDO enabling a framework-based optimization approach for HPC. In *AIAA Scitech 2021 Forum*. American Institute of Aeronautics and Astronautics, jan 2021. DOI: 10.2514/6.2021-1236.

[8] Justin S. Gray, John T. Hwang, Joaquim R. R. A. Martins, Kenneth T. Moore, and Bret A. Naylor. OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization. *Structural and Multidisciplinary Optimization*, 59(4):1075–1104, April 2019. DOI: 10.1007/s00158-019-02211-z.

[9] Michael Meinel and Gunnar O. Einarsson. The flowsimulator framework for massively parallel cfd applications. In *PARA2010*, Juni 2010.

[10] Lars Reimer. The flowsimulator—a software framework for cfd-related multidisciplinary simulations. In *European NAFEMS Conference Computational Fluid Dynamics (CFD) – Beyond the Solve*, Dezember 2015.

[11] Tim Bray. The javascript object notation (json) data interchange format. https://tools.ietf.org/pdf/rfc8259.pdf, December 2017. Accessed: 10.08.2021.

[12] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of json schema. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, page 263–273, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee. DOI: 10.1145/2872427.2883029.

[13] Markus Ritter. Nonlinear numerical flight dynamics of flexible aircraft in the time domain by coupling of cfd, flight mechanics, and structural mechanics. In Andreas Dillmann, Gerd Heller, Hans-Peter Kreplin, Wolfgang Nitsche, and Inken Peltzer, editors, *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, volume VIII of *new Results in Numerical and Experimental Fluid Mechanics*, pages 339–347. Springer Berlin Heidelberg, 2013.

[14] Martin R. Waszak, Carey S. Buttrill, and David K. Schmidt. Modeling and model simplification of aeroelastic vehicles: An overview. Technical Report NASA TM-107691, NASA Langely Research Center, 1992.

[15] J.C. Butcher. Numerical methods for ordinary differential equations in the 20th century. *Journal of Computational and Applied Mathematics*, 125(1):1–29, 2000. Numerical Analysis 2000. Vol. VI: Ordinary Differential Equations and Integral Equations. DOI: https://doi.org/10.1016/S0377-0427(00)00455-6.

[16] Jan R. Wright and Jonathan E. Cooper. *Introduction to Aircraft Aeroelasticity and Loads*. John Wiley & Sons, Ltd, Chichester, 2nd edition, December 2014. DOI: 10.1002/9781118700440.

[17] Agostino De Marco, Eugene Duke, and Jon Berndt. A general solution to the aircraft trim problem. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*. American Institute of Aeronautics and Astronautics, June 2007. DOI: 10.2514/6.2007-6703.

[18] Steven Allmaras, Forrester Johnson, and Philippe Spalart. Modifications and clarifications for the implementation of the spalart-allmaras turbulence model. *Seventh International Conference on Computational Fluid Dynamics (ICCFD7)*, pages 1–11, 01 2012.

[19] Tania Kirmse. Recalibration of a stereoscopic camera system for in-flight wing deformation measurements. *Measurement Science and Technology*, 27(5):054001, apr 2016. DOI: 10.1088/0957-0233/27/5/054001.