A Unified Approach for Multidisciplinary Preliminary Aircraft Design

Carsten M. Liersch, Martin Hepperle

German Aerospace Center (DLR), Institute of Aerodynamics and Flow Technology,
Lilienthalplatz 7, D-38108 Braunschweig, Germany

I. Abstract

The multidisciplinary analysis of aircraft configurations is mandatory to assess the benefits of novel aeronautical technologies developed by DLR. An appropriate procedure for this task is being developed within the project TIVA, where an integration framework and a common data exchange format are used to couple numerical analysis tools from multiple disciplines. Using this set of tools, individual process chains can be created and used to evaluate the impact of new technologies.

This paper consists of three main parts. The first one provides a brief overview of concept and structure of the new data exchange file format CPACS. The second part focuses on the integration framework, discussing its requirements and selection process. Further on, it describes the extensions that were necessary to adopt the framework to CPACS usage. The third part of this paper presents an example of a simple aero-structural tool chain for wing-box sizing and the way towards future applications.

II. Introduction

In the past, several tools have been developed to support the integrated preliminary design and analysis cycle. Many of these tools were using a monolithic programming model; individual disciplines were represented by embedding corresponding modules into a main program, executed by a single user on a single computer. Examples are the well known codes *FLOPS* [1] and *PrADO* [2]. Often these codes were developed by a very small group or even a single individual [3]. Due to their compact monolithic structure they often have the advantage of short execution times, but may be less well suited for cooperative work in teams.

Over the years, with the advances in computer performance, there has been a trend to replace simplified disciplinary models by more complex, physics based models. It is desirable to arrange such models in a multi-level hierarchy so that simpler models can be replaced by more complex models while using the same dataset. Larger development groups and distributed organization structures like "center of competences" in different locations make it more difficult to accept and to maintain a monolithic code base. Also, more flexibility is often desirable to rearrange the discipline modules in different order or just use a subset of them for specific studies. These requirements led to the development of more structured and even distributed architectures for preliminary design and analysis tools. Often these developments follow the object oriented software design paradigm and implement a distributed network centric process [4], [5], [6], [7].

The German Aerospace Center DLR, which is working on technology research and development, performs preliminary aircraft design activities to assess the value and driving factors of individual technologies. The research establishment is structured into disciplinary institutes ("centers of competence"), which are spread over several sites. In the past, preliminary aircraft design activities have been pursued by individual institutes, using mostly monolithic synthesis and analysis tools, like *PrADO*. Over the last years, many multidisciplinary analysis and optimization projects using high fidelity tools (CFD, CSM) have highlighted the need for a more collaborative organization of the preliminary design activities. For this purpose, the project *TIVA* (***T**echnology **I**ntegration for the **V**irtual **A**ircraft*) has been initiated with the following main objectives:

- Develop a common parametric description of the system “aircraft” suitable for all disciplines involved in the process.
- Establish a flexible process to link tools of different developer groups located at different sites.
- Provide the required interface definitions and software infrastructure.

The first point is a fundamental prerequisite for tool coupling. Whenever tools shall be linked, a common description has to be used for their input and output data. In order to simplify typical tasks in conceptual and preliminary aircraft design like trade studies and optimization runs, this description should use a good and flexible parameterization.

The second point covers the definition of a good concept for coupling multiple tools, provided on individual, network-connected computers. It further focuses on the evaluation of available framework systems for process integration and on the selection of a suitable system for the application within the *TIVA* project.

The third point finally covers the combination of data exchange file format and selected integration framework. This combination requires additional software for the tool connection as well as special plugins to extend the integration framework. In this paper, the explanations related to this third point are included in the sections on the data exchange format and on the integration framework.

III. Data exchange file format

General aspects of the file format

One of the main tasks in multidisciplinary tool coupling is the definition of a data exchange format that has to be used as common input and output for all tools. A first step in this definition was the identification of the requirements.

The data structure was planned to be hierarchical, breaking up the data into a tree of components and these components again into their sub-components and so on. Reusable datasets like airfoil coordinates, engine characteristics, or material properties are included as catalog data. Aircraft definitions and catalog data are both stored under a common “vehicles” node. However, in order to allow for more global approaches (like simulation of air traffic with more than one aircraft), this “vehicles” node is not the root of the tree, but has siblings like “fleets”, “airports” and “missions”. An example of the root structure is shown in Figure 1. The main benefit of the hierarchical storage concept is its flexibility: Only the required parts of the tree structure have to be provided. So it is easy to extract branches of the data-tree and hand them over to an analysis module. The module then does not need to handle the whole structure – it gets just what it needs to run and returns result branches which are then inserted into the tree. Another benefit of this concept is its extensibility: As a dataset only has to contain the required branches, a later extension of the data format definition does not invalidate existing datasets.

In general, it was considered important to implement parametric definitions where possible. This allows for more flexibility and robustness of automatic variations, as changes in the parameters can propagate through the model without destroying its topology. A good parameterization reduces the effort for trade studies and optimization runs. However, as the definition of “good” here depends on the trade study or optimization problem, a quite flexible parameterization was applied. For some components, this concept even permits different behavior on parameter changes, depending on the way how the component is modeled.

For the consistency of the dataset it is important to avoid any duplication of data – each piece of information is allowed to occur only once in the dataset, even if that may create additional conversion effort for the analysis tools. With respect to consistency it was also decided to store all data in SI units. Especially in aviation, where several unit systems are common, this was considered to be essential in order to avoid later misinterpretation of the data.

The data file serves as a common language between the coupled analysis tools. So it is essential to be able to read and write such files using various programming languages without too much effort. Besides this automatic handling, it should also be possible for the user to inspect and edit a dataset manually.

For such a file format it is essential to have a good documentation in order to allow for an easy usage. Combined with the documentation, it also would be very useful to have an

automatic validation mechanism to check, whether the structure of the data tree in the file is correct.

Advantages of XML-Based file formats

In *TIVA*, it was decided to develop an XML file format [8] as basis for data exchange. The hierarchical data structure of XML allows for a good mapping of the aircraft's geometry and its more abstract parameters, as well as for adding of new result branches into the data tree. The option to add attributes with so called "meta data" (like a unique ID or a symmetry flag) to each node is also a very useful feature. Using XPath and XSLT statements [9] it is possible to navigate through the data tree and to extract parts from it. Another advantage of XML is its representation in ASCII text form, which makes easy to handle, even with a simple text editor. In order to avoid disadvantages from the handling of very large files, a concept of data outsourcing was introduced. This concept allows for putting nearly arbitrary parts of the XML-tree into subfiles (even recursively). Using a special library, it is possible to automatically include all the subfiles from their different locations (even from web sources) and build up the complete tree structure again. While doing this, the external locations of the outsourced parts are preserved in a set of attributes, so the tree can be saved back again into its original subfiles later.

For documentation and validation of XML files, the "XML-Schema" concept is used [10]. A Schema document specifies the structure of an XML document by defining each node and its optional and mandatory subnodes, or, if it is a data node, its data type and the permitted range of values. It also contains a list of the node's possible and required attributes and a documentation of the node. Using such an XML-Schema (which itself is also an XML-document) an automatic validation is possible. When using a suitable XML editor for modifying a dataset, the schema can even be used to restrict the user input to the permitted data values or subnodes. A second use of such an XML-Schema is the automatic generation of the data structure documentation, using additional tools like "xs3p" [11]. This description generator uses XSLT commands to extract the nodes, their hierarchy and the corresponding documentation text into a HTML document, displayable by any internet browser. So the main advantages of using XML-Schema are the opportunity for automatic dataset validation and that the corresponding documentation is always being up to date and in sync with the dataset structure.

CPACS format

Based on these concepts, a data exchange file format named *CPACS* (***C**ommon **P**arametric **A**ircraft **C**onfiguration **S**chema*) was defined within the *TIVA* project. This data format is continuously enhanced and extended since the beginning of the project in 2005. It contains all the necessary data required for conceptual and preliminary aircraft design and analysis. As shown in Figure 1, *CPACS* currently contains 6 top level nodes:

- ***<header>***
  This data block contains general information (e.g. dataset name, creator, version or timestamp) about the initial dataset and its updates.
- ***<vehicles>***
  In this node, the definition of one or more aircraft is stored. It will be described in more detail below.
- ***<missions>***
  Here, a list of missions can be specified. The missions are built up from mission segments, which allow for simple conceptual design definitions as well as for detailed flight-plan like descriptions. An aircraft uses a reference to one or more of these missions as its design missions.
- ***<airports>***
  This node defines airports with their location and properties like runway lengths or surface conditions. These airports can be used for more detailed mission descriptions.
- ***<fleets>***
  The <*fleets*> node currently is only a placeholder. Later, it will permit to specify whole fleets, consisting of several different airplanes (by referring to their definitions in the <*vehicles*> node).
- ***<toolspecific>***
  This data node contains a subnode for each tool that is connected to *CPACS*. Besides name and version of the tool it specifies all the necessary additional data that is needed to run the tool (e.g. tool parameters for geometry processing, amount of debug

logging and additional return files, etc.). As an example, a part of the toolspecific input for the aerodynamics tool "LIFTING_LINE" is shown in Figure 2.

To provide a deeper insight into the structure of *CPACS*, the concept used for defining the wing shape is now described exemplary in more detail. In *CPACS*, a wing consists of a set of segments having each a trapezoidal planform. This simplification was considered sufficient for a wing definition in preliminary aircraft design. Using a large number of small segments, arbitrary wing geometries can be modeled.

For a wing definition, a set of wing airfoils is needed. These airfoils are stored in the wing airfoil catalog, which is part of the <*profiles*> catalog in the <*vehicles*> node (see Figure 1 and Figure 3). Besides its name and a unique ID (UID), each airfoil definition consists of a list of points forming the airfoil's shape. Typically, the points are provided in relative coordinates for unit chordlength, but in terms of flexibility it is also possible here to enter global coordinates (e.g. if the airfoils were extracted out of an existing 3D aircraft model). To add more flexibility, it is even possible to enter a 3D airfoil geometry.

All definitions of the wing are located in the <*wings*> node inside the aircraft model specification (see Figure 1 and Figure 4). Just like an airfoil, each wing has a UID attribute to make it referable and further it has a name. Additionally, it carries an optional symmetry attribute. This permits the user to enter just one semi-wing and to assure wing symmetry automatically. In *CPACS*, symmetry is not specified as a global attribute, but individually for each of the main components. Hereby, the amount of required input data is reduced without restricting the variety of possible shapes. Each of the components has a <*parentUID*> element, which is a reference to the component's parent (the part, to which it is attached – e.g. wings and tailplanes are typically attached to the fuselage, while engines may be attached to a wing or a fuselage). This connection is essential for definitions like the wing's global transformation node (containing translation, scaling and rotation vectors to position the wing as a whole), which typically acts in the parent component's local coordinate system.

The wing definition itself is assembled from three definitions: <*sections*>, <*positionings*> and <*segments*>. Sections are 2D cuts which serve as borders of the wing segments. To allow for more complex wings with extended slats and flaps, each section can contain multiple airfoil elements. Each element finally consists of a reference to an airfoil and a transformation node to position the airfoil in the current section by translation, scaling and rotation (see Figure 4 and Figure 5). Positionings are used to position the sections in space: Using length, sweep angle and dihedral angle, each section is placed, relative to another section, or to the wing's origin (see Figure 4). The sections are referred to by their UID. The flexibility of using different references to specify the location of a section allows for the creation of models with totally different behavior on parameter changes, thus permitting a great variety of different parameterizations. The third component in the wing definition finally is the segment. The segment definition is rather simple, as it mainly consists of two element UIDs, describing which two airfoil elements serve as borders of the segment (see Figure 4 and Figure 6). The basic concept is also shown in Figure 7; an example of a more complex wing structure is depicted in Figure 8.

Besides the outer shape description, the wing's components also contain structural definitions like spars and ribs. The control surface definition (see Figure 9) permits to declare control surfaces by specifying a wing segment, relative spanwise coordinates of the inner and outer border of the control surface and the relative chordlength at both borders. The deflection law is described by discrete steps, containing each a relative deflection and a related transformation (again, consisting of translation, scaling and rotation). The idea of using a relative deflection is to abstract the absolute transformation by providing always values like "+1" or "-1" for the limits and "0" for the neutral setting, even if the deflections are non-symmetric.

Results produced by the analysis tools are fed back into *CPACS*. As long as they are aircraft specific, they are located either in the <*global*> or <*analyses*> nodes of the aircraft model (see Figure 1). At this point, the idea is to distinguish between general data like global masses and performance data (<*global*> node) and the more detailed preliminary aircraft design analysis data (<*analyses*> node). While

someone simulating the air transportation system using whole fleet of aircraft might only be interested in the <*global*> node, the aircraft designer needs data from the <*analysis*> node as well. But while he might, for instance, only need the <*global*> node of the engine specification, an engine designer would also need the engine's detailed data (but maybe only the global data from the aircraft to size the engine). The results are either stored in the form of single values, as vectors, or as arrays, containing whole performance maps in the form of lists in one single node.

In order to work with *CPACS* files, a simple ASCII reading and writing capability is sufficient. To simplify work with *CPACS* and to cover its special features (e.g. integration of external data files), DLR has developed a special support library called *TIXI* [12] *(**TI**VA **X**ML **I**nterface)*. *TIXI* is a C library, which is based on the free *libxml2* library [13]. It is independent of platform and programming language.

In order to simplify access to the parametric geometry model, a second library, called *TIGL (**TI**VA **G**eometry **L**ibrary)*, was developed [12]. Based on the free *OpenCASCADE* CAD library [14], *TIGL* generates a 3D model from the *CPACS* aircraft geometry definition and provides high-level functions to query data from this model. The *TIGL* library also supports an export of the generated geometry model into various CAD formats. An additional graphical tool allows to inspect the aircraft geometry in a 3D viewer environment. An example screenshot from this *TIGLViewer* is shown in Figure 10.

IV. Integration framework

Assessment of Process Automation Systems

Because the decision for a process automation system was crucial for the project, the assessment of available framework systems was a major milestone of the *TIVA* project. Within the *TIVA* project it was decided to use a framework system, because

- individual tools do not (and should not) talk to each other directly,
- the chain of tools may change, depending on the assessment task,
- tools may reside on different hardware and software platforms,
- tools may be located and maintained at different sites,
- some level of automation was desirable for parameter studies and optimization.

At first, a catalogue of required and desired features was composed (see Table 1):

- Systematic development of process models
  - Clear representation of dependencies
  - Hierarchically organized model structures
- Large toolbox of applications
  - Parameter sweeps (one- and multi-dimensional / DOE)
  - Optimization (gradient based, gradient free, stochastic, BLISS, ...)
  - Generation of response function models (RSM, Kriging)
  - Probabilistic design
  - Support of popular scripting languages for prototyping and as "glue"
- Technology
  - Distributed simulation models (client / server)
  - Support for parallel architectures (e.g. „grid computing")
- Documentation
  - Suitable user manuals for easy starting
  - Documented developer interfaces

Table 1: Desired features of the framework system

It should be noted that these requirements are specific for the project *TIVA* and the environment at DLR. For example, documentation and ease of application was considered important because the work force of DLR consists to a considerable percentage of young scientists spending typically 3 to 5 years at DLR and then continuing their career in industry. While this has the positive benefit of a continuous refreshing of the team it is important that the initial training period for a process automation tool should not be too long.

During the last years several different automation systems have been used in DLR. Applications for preliminary aircraft design were mostly performed by the Institute of

Aerodynamics and Flow Technology (AS), as well as by the Institute of Aeroelasticity (AE). Besides commercial products, freely available software and in-house systems have been used in various projects. A few of the known systems have either been discontinued, were not considered due to previous experience or were not yet mature enough. The final assessment procedure was applied to the first six candidates of the systems listed in Table 2.

1.	***iSIGHT*** (Engineous, USA)	*(in use at AE)*
2.	***modeFRONTIER*** (Esteco, IT)	*(in use at AS)*
3.	***ModelCenter*** (Phoenix Integration, USA)	
4.	***PaceLab*** (Pace, Berlin)	
5.	***PointerPro*** (Synaps, Bremen)	*(in use at AS)*
6.	***TENT*** (DLR SISTEC, Köln)	*(in use at AE / AS)*
7.	***Epogy*** (Synaps, USA)	*(in use at AS)*
8.	***GenOpt*** (Lawrence Berkeley National Lab., USA)	*(in use at AS)*
9.	***Dakota*** (Sandia National Labs., USA)	*(in use at AS)*

Table 2: Process automation tools considered during the assessment process in *TIVA*

All of these systems offer a rich set of features and require careful evaluation to finally produce a sound picture of the capabilities and possible limitations of each tool. Therefore, a formal evaluation process was set up. The process was intended to model a typical application in preliminary aircraft design. It was very much simplified, so that it could be implemented for each framework within a few days. The process chain consisted of eight modules, which had been prepared in form of executable programs using ASCII input and output files, as well as alternative implementations of each module written in the *Matlab* programming language. In order to leave some room for possible innovative scheduling capabilities of some of the systems, not all modules depended on each other.

To produce a realistic view on each system, teams of 2-4 persons were assembled, each group working on the implementation of the process during a two day workshop. For each framework, an individual workshop was organized so that each team member could test and compare at least two different framework systems. A questionnaire had to be answered during and after the evaluation by each participant.

In the end, about 15 persons had been actively involved in the tests. While this procedure proved to be rather time consuming, it can be hoped that it produced a relatively fair comparison of the different systems.

To distinguish between the available framework systems, two main criteria had been found during the test procedure:

- Networking capabilities and
- Process scheduling.

Networking Capabilities

Considering networking capabilities, three different groups could be identified. The first group consists of purely local systems without network support. Here, the user could only use networked applications by programming his own scripts or programs. The second group of tools offers some limited networking support, usually using mechanisms like “remote shell” or “secure shell”. The user can configure a list of hosts and user accounts to be used. Finally the third group of tools offers built-in networking support, using a client-server architecture. This architecture typically runs a server module on each networked machine. Such server modules can either be running in form of “daemons” or they are started on request. Each user then starts a client module, in which he builds up and runs his process chains. The framework systems under consideration were almost evenly distributed among these three categories.

Scheduling Capabilities

Another feature which can be used to categorize automation systems is the way how complex processes are built and driven. Here two classes of systems were found. The first class uses a sequential process model, linking components into a linear sequence. During execution, the components are executed one

after the other. Often, some additional building blocks are available to provide control logic. The second class of framework systems does not connect modules in a linear process sequence, but connects them via dependencies, usually variables or equations (or both). For the user, the first class of systems is easy to understand and to get used to. The second class requires a slightly different kind of "thinking in dependencies", especially when it comes to process control. However, the huge advantage is that the system maintains a dependency tree and only executes the modules on an "as needed" basis. This can greatly reduce analysis time and also offers the potential for automatic parallelization. Again, the tested framework systems fell to equal parts into these two groups.

Framework selection

In the *TIVA* project it was finally opted for client server based systems, because

- components can be managed locally and published by individual organization units,
- computing power can be provided by dedicated server machines as needed,
- security aspects can be implemented and authentication is possible, even down to component level,
- load-balancing and usage of grid resources can be a natural part of the system.

Within these constraints, three of the systems were highly rated. The careful consideration of all relevant details finally lead to the procurement of the *ModelCenter* software suite from *Phoenix Integration*, USA. It consists of the *ModelCenter* Client as front end and one or more distributed *AnalysisServers*. Both can interact with the *CenterLink* load balancing system. It should be emphasized here, that this selection was based on the specific requirements of the *TIVA* project and cannot be generalized.

CPACS implementation into ModelCenter

After choosing *ModelCenter* as best suited for the *TIVA* requirements, a concept of integrating *CPACS* and the desired way of tool coupling was developed. At this point, the preferred way would have been to include the whole hierarchical *CPACS*-structure as a tree of variables into *ModelCenter* and then link the appropriate parts of the structure directly with the tools. When this way was tested out, a major drawback occurred: *ModelCenter* lacked the feature of adding new variables while the process chain is running. This means, that a tool cannot add its results to the dataset, it can just replace already existing place-holder nodes. So a *CPACS* dataset would have had to contain place-holders for all the possible results before the chain could be run the first time. In addition to this, a tool coupling via the individual variables would require these place-holders to be even there when the process chain is set up and the tools are linked. In this case, a process chain would be totally fixed to one special structure (e.g. a wing with a fixed number of wing segments, a fixed number of loadcases, etc.). As these restrictions are not acceptable for a flexible preliminary design system, another approach was chosen: The whole *CPACS* dataset is currently stored inside *ModelCenter* as a single text variable. So changes in the structure, new subtrees etc. are just recognized as changes of the single text variable's content, hence, the problem of structural changes in the dataset completely disappears. A disadvantage of this concept is that the *CPACS* values and parameters are not directly accessible as *ModelCenter* variables and so, trade studies, optimization etc. would not be possible. Therefore, the concept of so called "*splitters"* and "*mergers"* was introduced. A *splitter* is a small *ModelCenter* component, which uses an *XPath* statement and the *TIXI* library to extract a specific variable from the *CPACS* string. A *splitter* can be set up even before such a string is available (or contains the requested content), but will provide the variable value when the chain runs and the required data exists. The opposite task is handled by the *merger* components. A *merger* takes an input variable and inserts its value at a certain (*XPath*-controlled) location in the *CPACS* string. Using *splitters* and *mergers*, it is possible for *ModelCenter* to change parameters and to keep track of the according results. Recently, Phoenix Integration has implemented a more hierarchical object handling and a more flexible linking architecture into upcoming *ModelCenter* versions. This will allow to get rid of the workarounds mentioned above.

The connection of the analysis tools is done in a way that each of them implements a *CPACS* interface (using *TIXI* and *TIGL*), either directly integrated into the tool, or as a wrapper around

it to leave the tool itself unchanged. The tools are connected to the AnalysisServer with a special *TIVA* wrapper. This wrapper provides the *ModelCenter* user simplified access to the tool (by selecting it from a list of tools) and it also takes care of the *CPACS*-Data exchange between *ModelCenter* and *AnalysisServer*. Besides the main *CPACS* dataset, this exchange also includes the transfer of an additional ZIP file. Using this ZIP file, the tool can return additional data, which is not part of *CPACS*, to the *ModelCenter* user.

For the handling of *CPACS* import and export to and from *ModelCenter*, two further components were developed: *CPACSSource* and *CPACSDestination*. The first one takes care of reading a *CPACS* file and providing it as a *ModelCenter* variable. This includes features like the recursive collection of outsourced parts of the dataset or an automatic validation against the *CPACS* Schema. The second component is able to save the *CPACS* dataset back to file, either on explicit user request, or, as a part of the tool chain, automatically during each iteration run. It is also able to collect and export the additional ZIP files coming from the tools. Both of the components use the *TIXI* library to handle *CPACS*.

V. First applications and future planning

In this section, a simple example for the usage of the TIVA system is presented: An aero-structural tool chain for the sizing of the primary structure of aircraft wing box boxes is created. This chain is applied to a VFW-614 aircraft configuration, in order to minimize the structural mass of the wing box.

Aero-structural tool chain

The core part of the tool chain consists of two tools: aerodynamic analysis and structural sizing. For the aerodynamics, a lifting-line method is used. This simple, fast and robust tool is based on linearized potential flow theory, further simplified to thin wings. It computes force and moment coefficients of nearly arbitrary nonplanar wing configurations. The coefficients are provided as distributions along wing span, as well as integrated to the configuration's total values. As the tool neglects the effects coming from thickness, viscosity and transonic flow, further tools (e.g. handbook methods) would be necessary to compute the total coefficients of the overall aircraft. For the example application shown here, these simplified results are sufficient. The lifting-line method itself, its usability and limits are further described in [15] and [16].

For the structural sizing process, a finite element approach was chosen. An automatic model generator reads wing geometry and spar locations and generates a finite element model of the wing box. Then, it takes the spanwise load distributions generated by the lifting-line code from *CPACS* and computes the stress on each of the elements, using the commercial *MSC Nastran* software. According to the allowables of the used materials, an iterative loop finally determines the necessary thicknesses of the elements. The mass and center of gravity of the sized wing box is calculated and written back into *CPACS*. The model generator is further described in [17].

The complete tool chain is shown as a *ModelCenter* screenshot in Figure 11. Each of the rectangular blocks represents a tool or script component; the arrows indicate the flow of data. The chain begins with a CPACS file reader component (*CpacsSource*) which reads the dataset from file and provides it as a *CPACS* variable to the subsequent components. Then, the aerodynamics tool *LIFTING_LINE* reads the surface definition and the selected loadcase from the *CPACS* dataset which it receives as input. As output, total coefficients as well as spanwise load distributions for the wings are fed back into the *CPACS* variable and passed on to the structural sizing tool *ModGen*. *ModGen* performs the structural sizing and adds the wing box structural weight and center of gravity to the *CPACS* dataset. Finally, the *CpacsDestination* component takes the resulting *CPACS* dataset and writes it back to a file.

Since this example tool chain shall perform an optimization task, some further components are necessary. These components, however, are specific to the used dataset and optimization task, but could be adopted to other configurations or tasks quite easily. At the beginning of the chain, three *merger*-components are integrated into the data flow between *CpacsSource* and *LIFTING_LINE*. They are set up with an XPath in order to vary the wing twist in the sections at 29, 69 and 100% of the wingspan. The values for the variation are delivered from the

"*twist_parameters"* assembly, which is just a collection of three *ModelCenter* variables. Changing these three variables, either by hand, or by an optimizer or trade study component, the aircraft geometry with the corresponding wing twist can now be analyzed and sized with *LIFTING_LINE* and *ModGen*. In order to give *ModelCenter* a feedback from the results, of aerodynamics and sizing, two *splitter*-components are added at the end of the tool chain. With the according XPath statements, they extract the total induced drag coefficient and the total wing box mass from the final *CPACS* dataset and provide them also as *ModelCenter* variables. The last component shown in the screenshot is the *GradientOptimizer*, which is used to minimize the wing box mass by changing the three twist values.

VFW-614 wing box sizing example

The tool chain described above is now used to optimize the wing twist of a VFW-614 transport aircraft under cruise flight conditions. The *CPACS* model of this aircraft was already depicted in Figure 10, the flight conditions are shown in Table 3.

VFW-614 Transport Aircraft		
• Maximum Take Off Mass:	18 600	kg
• Wing Reference Area:	64	m²
• Cruise Flight Mach Number:	0.65	
• Cruise Flight Altitude:	25 000	ft
• Load Case:		
○ Load factor (Maximum Take Off Mass)	1	g
○ Lift coefficient:	0.2559	

Table 3: VFW-614 cruise flight conditions for wing box sizing example

Minimizing wing box mass without consideration of drag or other requirements will – of course – not lead to a good aircraft, but can be seen as a good testcase, as one could easily imagine what the optimizer should do here: As *LIFTING_LINE* is used in target lift mode, where the angle of attack is internally iterated to reach the desired lift coefficient, the optimizer does not have to care for this additional constraint. So he is free to drive the twist of the inner section to the allowed maximum while the two outer twist values go to down to the minimum. This shifts the load as far inboard as allowed, thus minimizing bending moment and necessary structural mass. The expected behavior should be nearly independent of the used loadcase. For this reason, a simple 1g cruise flight was selected for this example.

The upper diagram in Figure 12 shows the development of the wing box mass plotted over the optimization steps: After the initial computation, each iteration consists of four steps: three steps to find the gradient for each of the three twist values and the fourth step with the resulting improved twist setting. Three of these iterations are sufficient to find the expected minimum; the fourth one shows that there is no further potential for improvements in the allowed twist range. The grouping into 4 iterations is also indicated by the major gridlines at 5, 9 and 13 optimizer steps. The diagram in the middle of Figure 12 shows the effect of the optimization on the induced drag: As expected, the more and more non-elliptical lift distribution results in a significantly increased induced drag. While the wing box mass goes down by 8.3%, the induced drag increases by 131%. The third diagram in Figure 12 depicts the history of the three twist values during the optimization. As expected, they change continuously towards their maximum (29% of wingspan), respectively minimum (69% and 100% of wingspan). So, as already mentioned above, this optimization does not lead to a good aircraft – it shows just an example case, which is easy to understand.

The next step of this process towards a more realistic application would be to trade off wing box mass versus aerodynamic drag by analyzing the fuel requirement for each of the designs along a specific mission. Such an example is presented in detail in [18].

VI. Conclusion

In *TIVA* and several related projects, DLR currently builds up a new system for conceptual and preliminary aircraft design and analysis. One of its core components is the new data exchange file format *CPACS*, which provides flexibility and extensibility within a hierarchical and parametric data structure. The second core component is a set of tools and programming

librarys which can be used to connect analysis tools to the *CPACS* data format. As third core component, the commercial *ModelCenter* integration framework is used to connect the tools on distributed servers via network to build process chains for the specific analysis and optimization tasks.

Currently, the first applications of this system are being set up and used. Besides the presented simple example, two more complex applications are documented in [18] and [19]. In the future, the TIVA system will be further developed and enhanced in DLR projects, covering preliminary engine design, environmental impact of the air transport system and specific needs for unmanned aircraft systems. A future perspective on the TIVA system as a whole is provided in [20].

VII. Acknowledgements

The authors would like to thank the *TIVA* community for their contributions to the system presented here. The help from DLR-SC (Simulation and Software Technology) in using the *TIVA* tools is greatly appreciated. A special gratitude shall be given to DLR-AE (Institute of Aeroelasticity) for contributing the structural analysis and sizing tool *ModGen* to the example process chain.

VIII. References

[1] McCullers, L. A.; "*Aircraft Configuration Optimization Including Optimized Flight Profiles"*; NASA CP-2327 Part 1; Number 87N11743; 1984

[2] Heinze, W.; *"Ein Beitrag zur quantitativen Analyse der technischen und wirtschaftlichen Auslegungsgrenzen verschiedener Flugzeugkonzepte für den Transport großer Nutzlasten"*; ZLR-Forschungsbericht 94-01; Braunschweig, 1994

[3] Kroo, I.; *"An Interactive System for Aircraft Design and Optimization"*; AIAA Paper #92-1190; February 1992

[4] Schneegans, A.; Kranz, O.; Haberland, Ch.; *"Flying objects — an object-oriented toolbox for multidisciplinary design and evaluation of aircraft"*; Proceedings of the 21st Congress of the International Council of the Aeronautical Sciences; Melbourne; Australia; 1998

[5] Baalbergen, E. H.; *"SPINEware: A practical and holistic approach to metacomputing"*; in: proc. International Conference and Exhibition on High-Performance Computing and Networking HPCN Europe 1998; Lecture Notes in Computer Science 1401; Springer, pp. 1008-1011; 1998. NLR TP98478

[6] Vankan, W. J. and Laban, M.; *"A SPINEware based computational design engine for integrated multi-disciplinary aircraft design"*; 2002; NLR-TP-2002-331

[7] Cooper, C. A.; Alderliesten, R.; La Rocca, G.; Benedictus, R.; *"In Search of a Knowledge Based Preliminary Design Method of Complex Aircraft Wings"*; 7th Annual Conference on Systems Engineering Research 2009 (CSER 2009)

[8] World Wide Web Consortium (W3C); *"Extensible Markup Language (XML)"*; http://www.w3.org/XML; 2009

[9] World Wide Web Consortium (W3C); *"The Extensible Stylesheet Language Family (XSL)"*; http://www.w3.org/Style/XSL; 2009

[10] World Wide Web Consortium (W3C); *"XML Schema"*; http://www.w3.org/XML/Schema; 2008

[11] xs3p download site; http://xml.fiforms.org/xs3p; FiForms Framework; 2009

[12] Bachmann, A.; Kunde, M.; Litz, M.; Schreiber, A.; *"A dynamic data integration approach to build scientific workflow systems"*; Institute of Electrical and Electronics Engineers; International Workshop on Workflow Management (IWWM 2009), Geneva; Switzerland; 2009

[13] libxml2 download site; http://xmlsoft.org;

[14] Open CASCADE S.A.S.; *"Open CASCADE Technology, 3D modeling & numerical simulation"*; http://www.opencascade.org; 2009

[15] Horstmann, K. H.; *"Ein Mehrfach-Traglinienverfahren und seine Verwendung für Entwurf und Nachrechnung nichtplanarer Flügelanordnungen"*; Forschungsbericht DFVLR-FB 87-51; Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt; Institut für Entwurfsaerodynamik; Braunschweig; 1987

[16] Liersch, C. M.; Wunderlich, T. F.; *"A Fast Aerodynamic Tool for Preliminary Aircraft Design"*; AIAA-2008-5901; 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Victoria, Canada, 2008

[17] Klimmek, T.; *"Parameterization of Topology and Geometry for the Multidisciplinary Optimization of Wing Structures"*; accepted for presentation at CEAS 2009 European Air and Space Conference; Manchester; UK; 2009

[18] Rodax, B.; Hühne, C.-P.; "Verwendung von ModelCenter im Projekt TIVA II"; Institutsbericht IB 124-2009/5; ISSN 1614-7790; Deutsches Zentrum für Luft- und Raumfahrt e.V.; Institut für Aerodynamik und Strömungstechnik; Braunschweig; 2009

[19] Bertsch, L.; Looye, G.; Otten, T.; Lummer, M.; *"Integration and application of a tool chain for environmental analysis of aircraft flight trajectories"*; 9th AIAA Aviation Technology, Integration and Operations Conference (ATIO), Hilton Head, South Carolina, USA, 2009

[20] Nagel, B.; Langhans, S.; Liersch, C. M.; Bertsch, L.; Gollnick, V.; *"Towards a Holistic View on Air Transportation"*; accepted for presentation at CEAS 2009 European Air and Space Conference; Manchester; UK; 2009

VIII. Figures

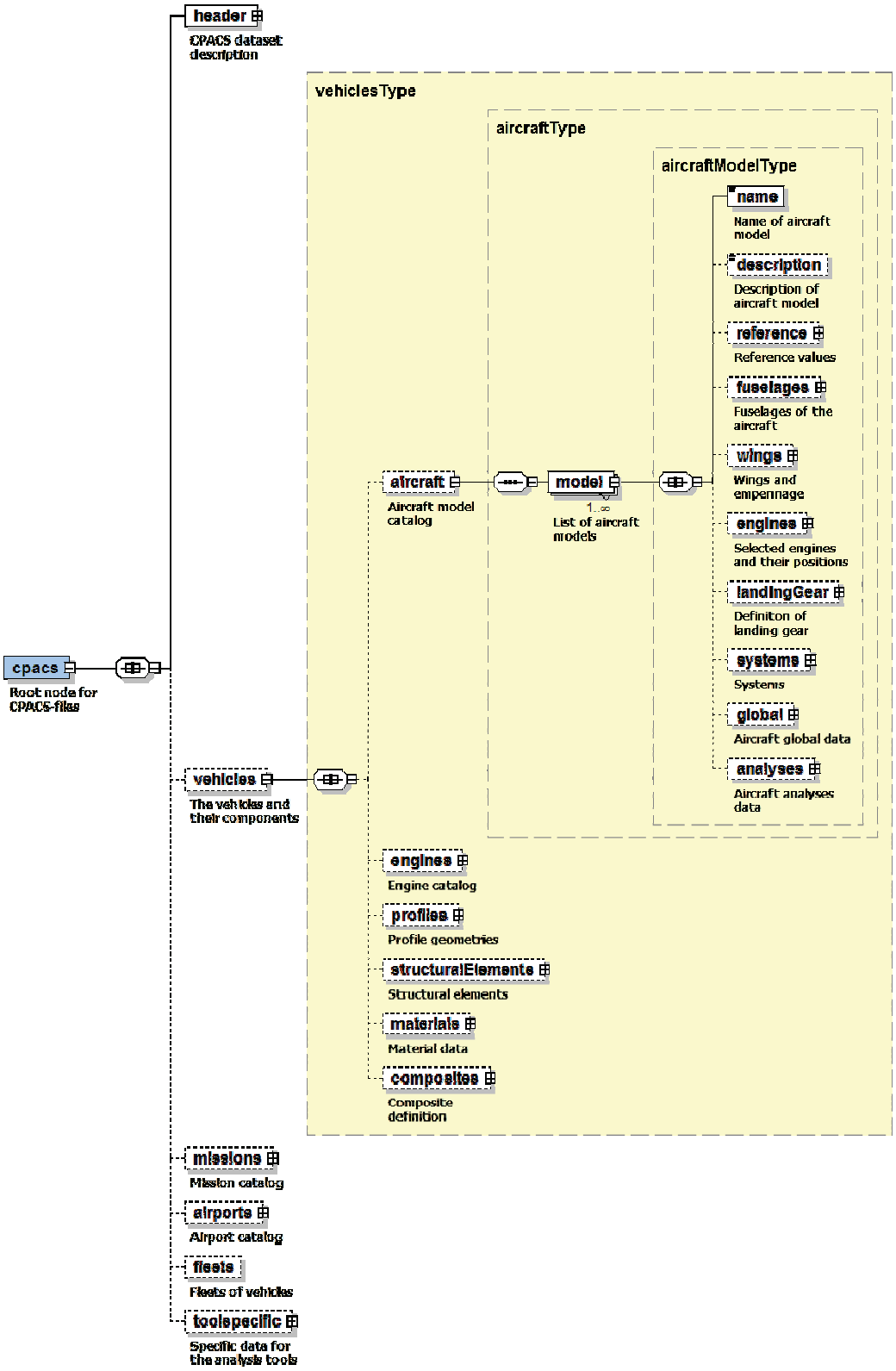

Figure 1: Root structure of the *CPACS* data format

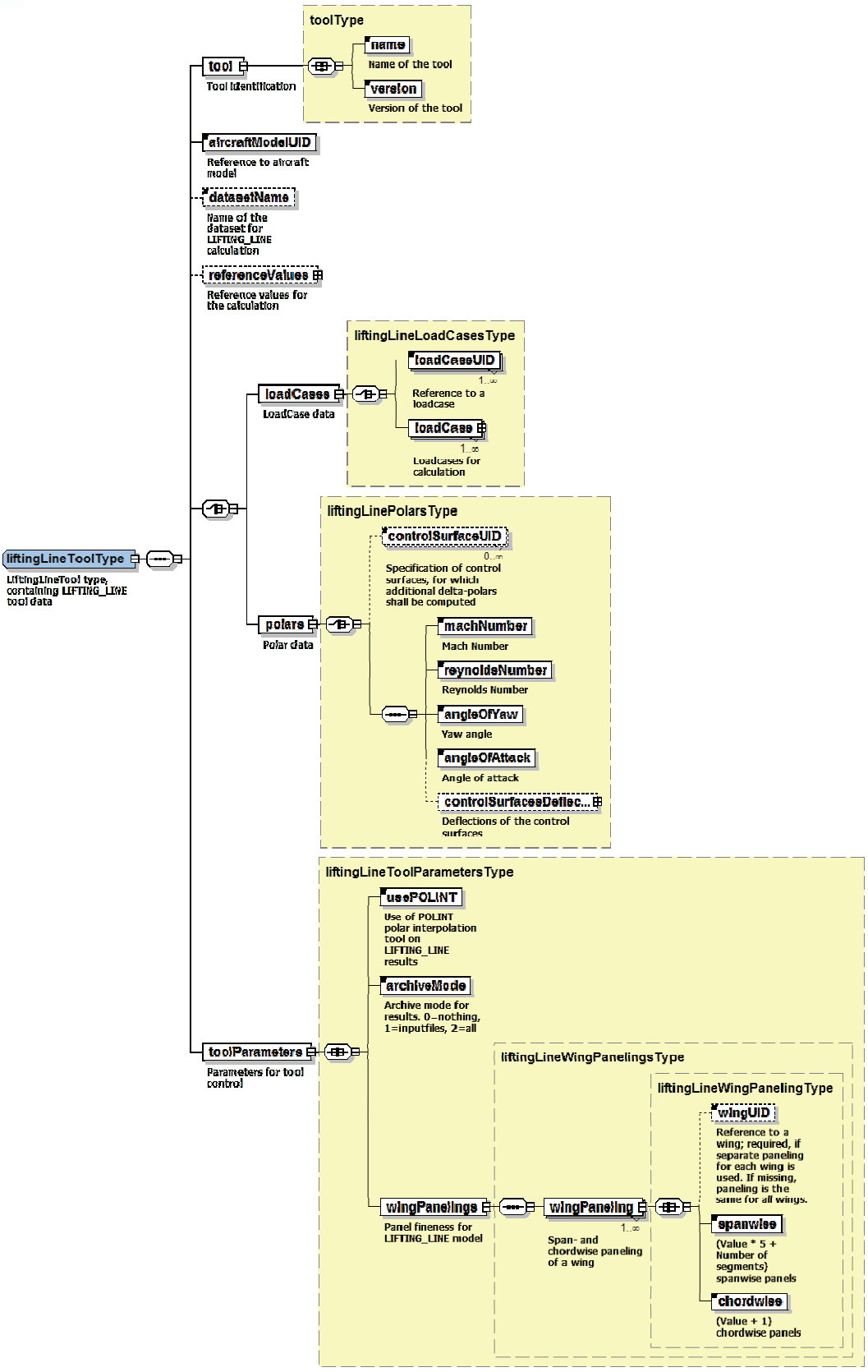

Figure 2: Structure of the LIFTING_LINE toolspecific data block

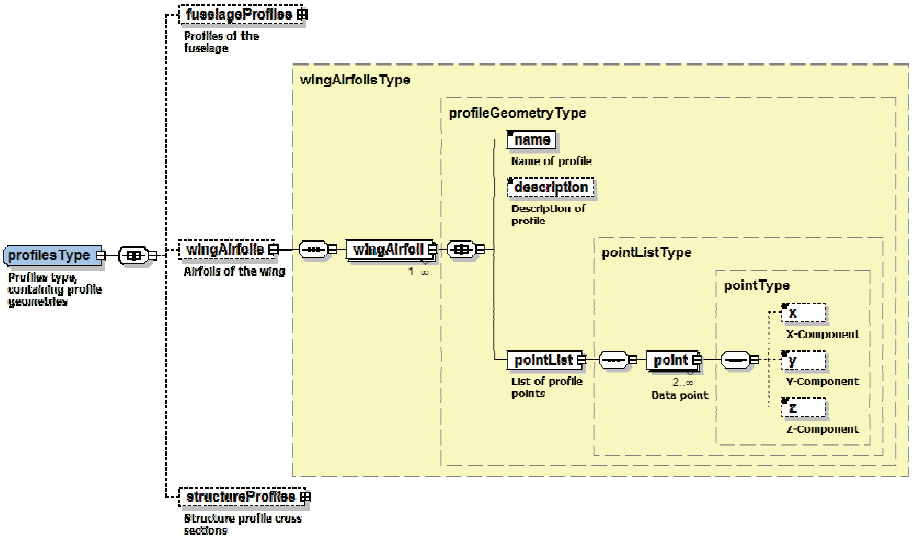

Figure 3: Structure of the airfoil catalog

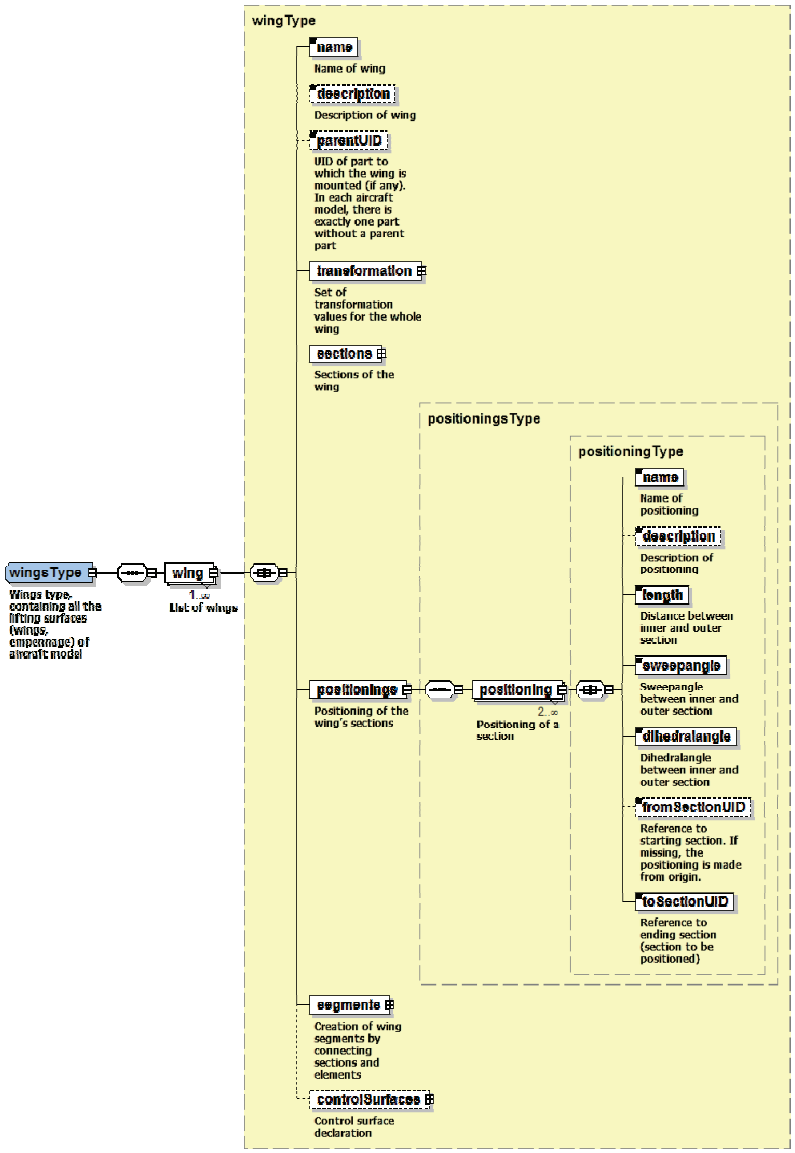

Figure 4: Structure of the wing definition

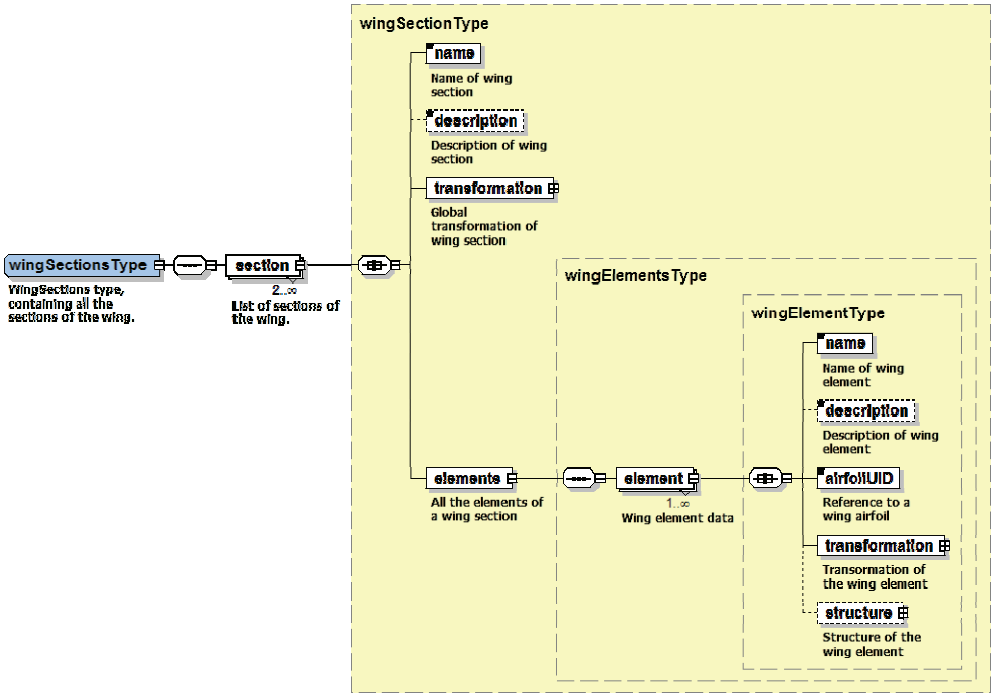

Figure 5: Structure of the wing section definition

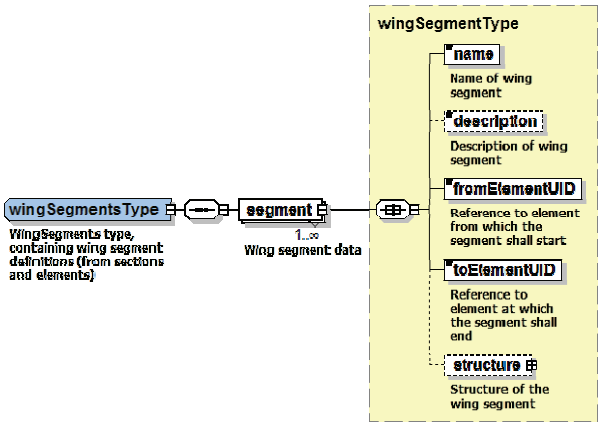

Figure 6: Structure of the wing segment definition

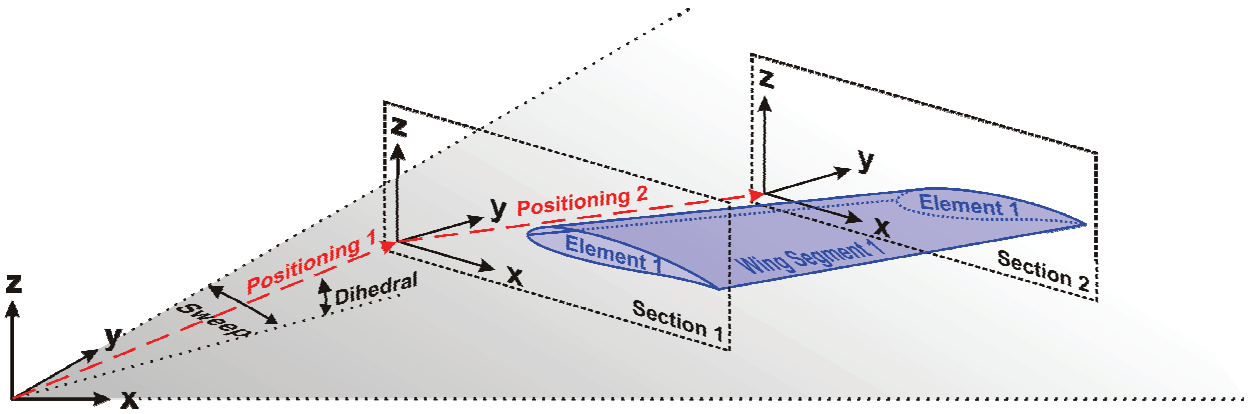

Figure 7: Concept of the wing definition

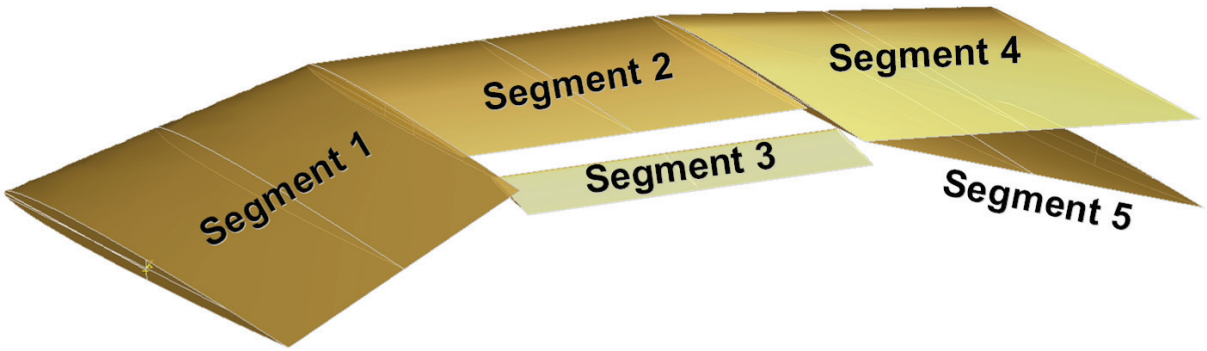

Figure 8: Example of a complex wing shape consisting of five segments

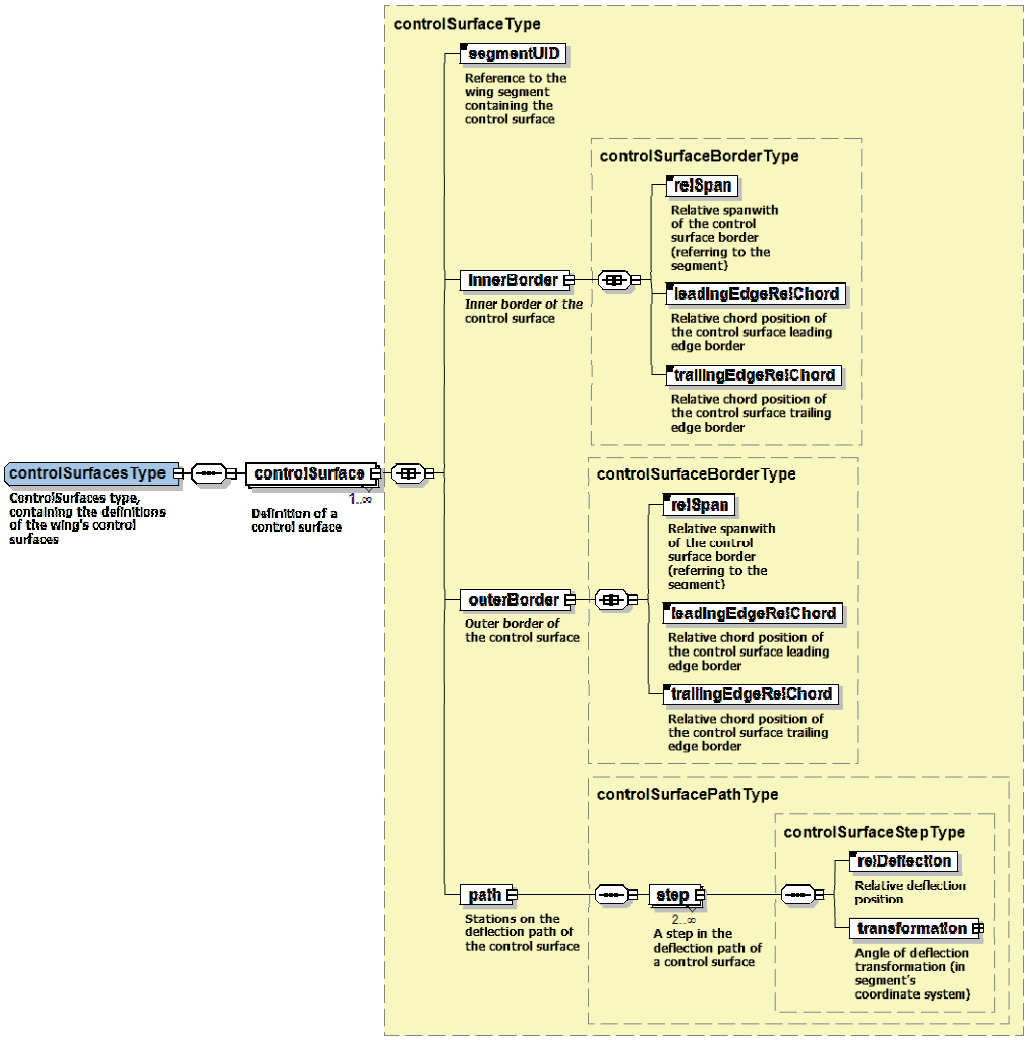

Figure 9: Structure of the control surface definition

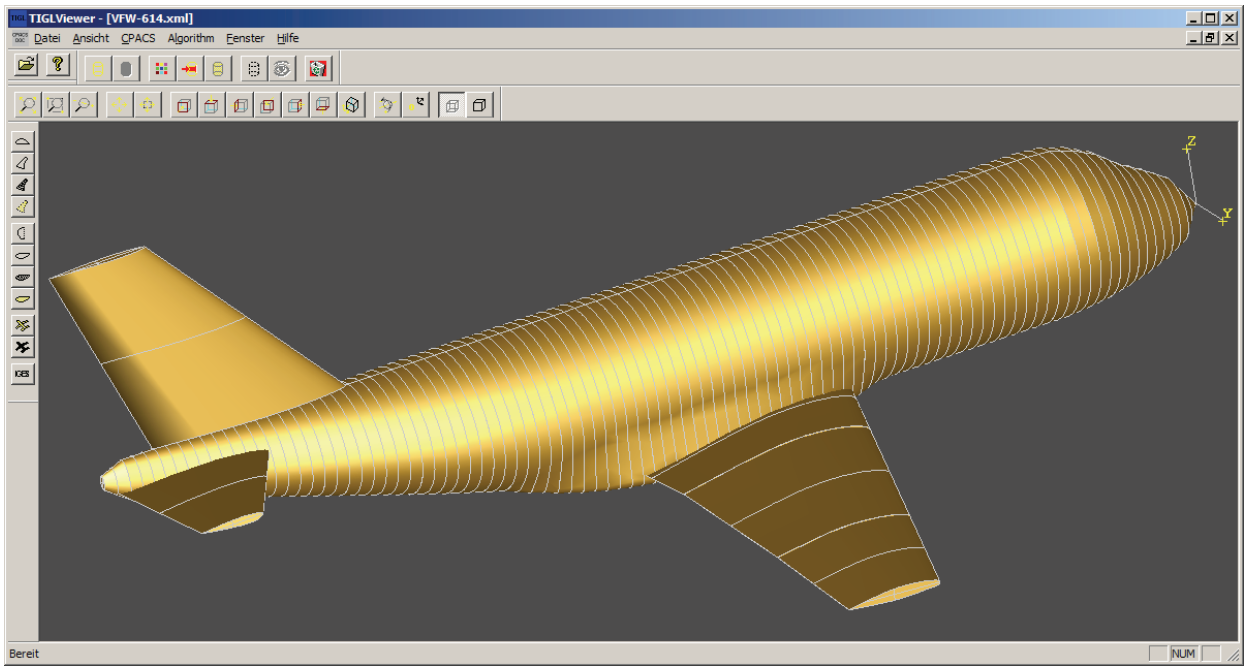

Figure 10: Screenshot from *TIGLViewer,* showing a VFW-614 transport aircraft model

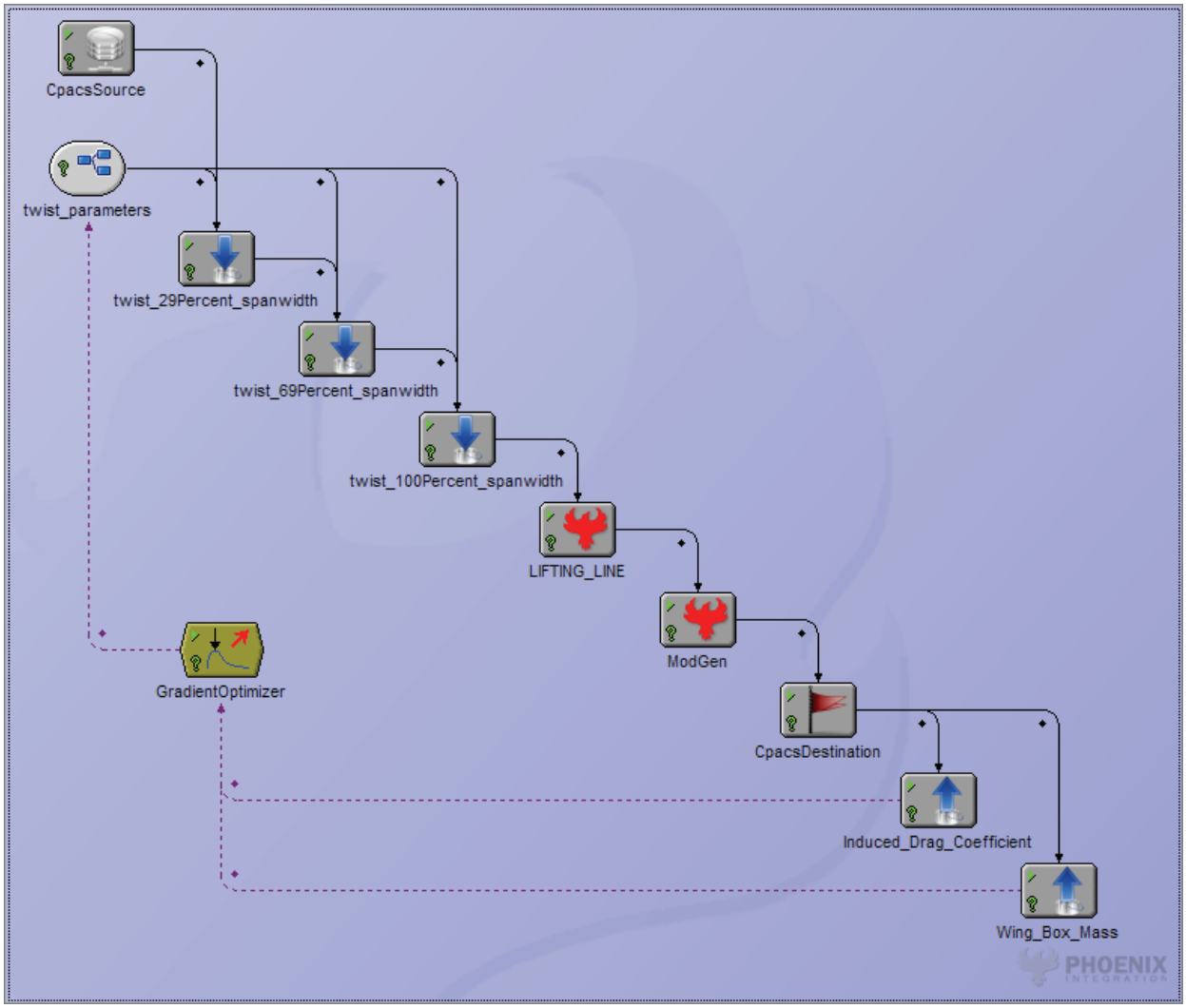

Figure 11: Screenshot from *ModelCenter*, showing an example process chain

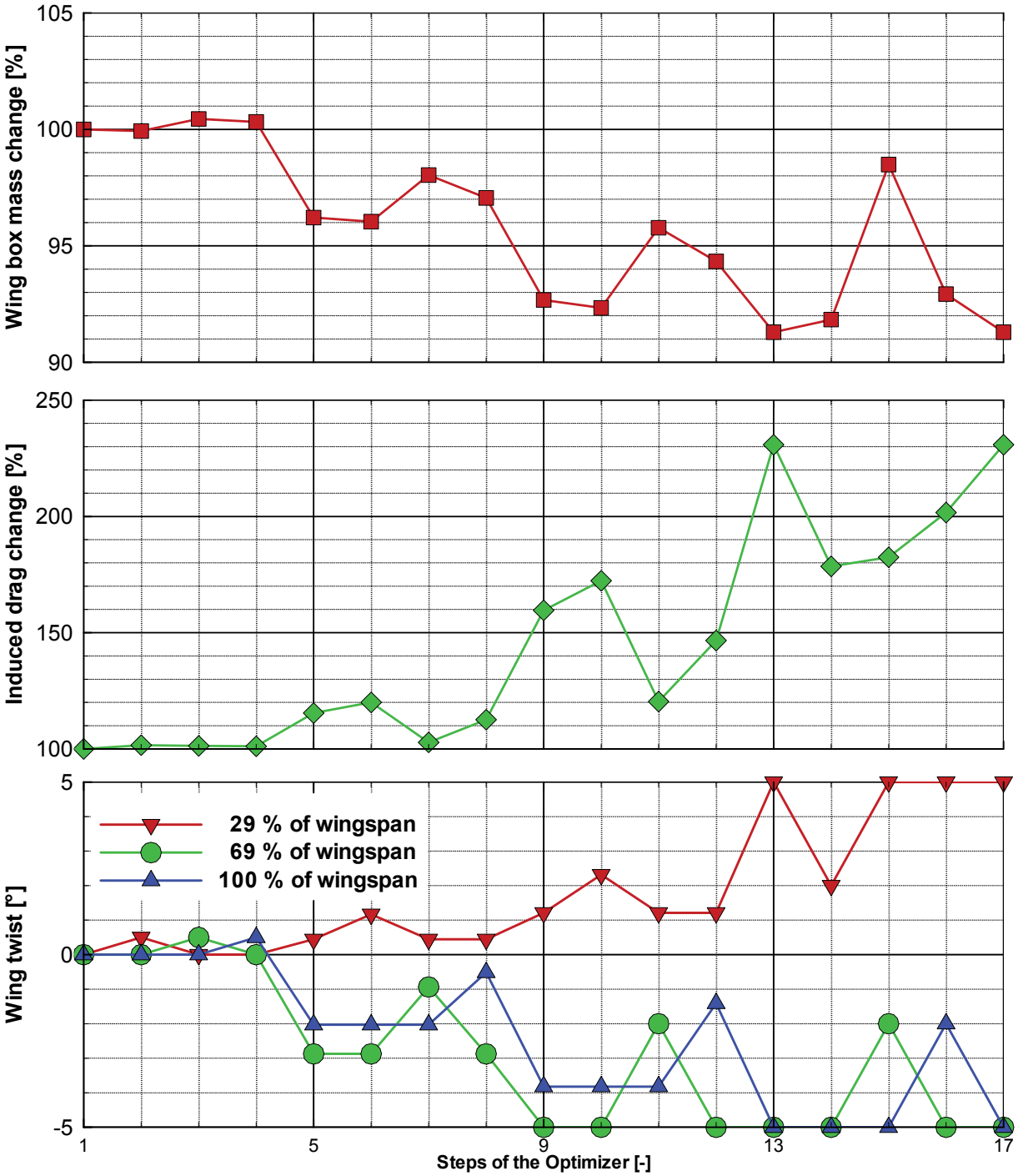

Figure 12: Optimization history of the example tool chain