Integration Framework for Preliminary Design Toolchains

M. Litz[1], D. Seider[1], T. Otten[2], A. Bachmann[1], M. Kunde[1],
{ Markus.Litz, Doreen.Seider, Tom.Otten, Arne.Bachmann, Markus.Kunde }@dlr.de
German Aerospace Center
[1]Simulation and Software Technology
[2]Institute of Propulsion Technology
Linder Höhe, Cologne

1. Abstract

The goals defined by the ACARE Vision 2020 present a major challenge to aeronautic research. Besides developments in new aerodynamics and structures, advancements in aero engine research will account for a major part of the required reduction in fuel, emissions and noise. To enable a commercial launch of new aircraft and engine concepts, complex and often contradictory demands have to be fulfilled.

Strong dependencies between the individual technical disciplines exist so that the optimization in a single discipline may not lead inevitably to a global optimum. Therefore it is necessary to look at the overall system in order to evaluate the potential of new technologies realistically.

This article presents a typical design task in aeroengine predesign and a software solution which supports and enables multidisciplinary cooperation on the engineer side. A common data format based on XML, necessary for data exchange, as well as supporting programming libraries for the processing of this data format are introduced. Furthermore it is described, how a parametric representation can be realized for various geometries with the help of XML. A programming library with C and FORTRAN Interfaces supports geometrical computations for these representations. Finally it is demonstrated that the tools used by the different technical disciplines can be connected to a process chain within a framework.

2. Introduction

In the predesign phase of airplane and aeroengine development many different technical disciplines are involved. Each of these disciplines comes with special simulation programs that fit perfectly to the associated questions within the technical field of the discipline. In this article such simulation programs will be referred to as application. Often these applications are used for optimization of technical problems within these disciplines. However, due to the strong dependencies between the different disciplines a global optimum can rarely be obtained by the combination of optimal solutions that result from separate optimization processes within every single discipline. For example, an aerodynamic optimization of a component may result in an infeasible design due to stability problems. For this reason, it is essential to keep track of the complete system.

Furthermore, the applications are usually operated manually and isolated from each other. Therefore, an automated tool chain in which the applications are executed in consideration of their existing dependencies is required. A similar approach has been used in 3m-XML [1]. Such automated toolchains are necessary to determine the mutual dependencies of all applications and perform multi-disciplinary simulations of a complete system with minimized user interaction, especially when dealing with global optimization.

Since data exchange between the technical disciplines is crucial, different approaches to couple such applications are possible:

- The combination of all simulation algorithms involved into a single pro-gram results in a very strong but inflexible coupling, from the software technical point of view, but may be desirable due to algorithmic reasons.

- Coupling applications via MPI or CORBA [2] gives a slightly more flexible system due to a general interface defined by the communication libraries.
- A flexible system can by designed using flexible data exchange via a modern software integration framework. These frameworks could handle most technical functions like remote code execution, data management between different computing nodes or visualization of result data.

At the German Aerospace Center (DLR) the last approach was chosen. The easy substitution of an existing application by another one with the same functionality or adding a new application to the system are essential design issues. A common database which is accessed by the applications and used for data storage and exchange was realized via a global XML [3] data set.

At this time, there is no framework available that could handle all the tasks necessary for a distributed execution of simulation tools with a common data set that meets DLR's reqirements. A larger number of free and commercial products tries to address the specific needs of scientific data management, flexibility to change workflows at any time, real-time monitoring and powerful post-processing facilities, but often they fail in one or more point. The best known solutions for this software field are Kepler & Taverna [4], [5], iSight (http://www.simulia.com/academics/isight.html) or ModelCenter.

The solutions developed for these tasks are introduced in the following sections starting with an introduction of the DLR project EVITA that will use the developed software environment, followed by a review of the format of the common data base in section 4. Section 5 describes the software libraries developed to access this database. Section 6 is about the developed framework and about the integration of our tools and our data format. In Section 7 we present some conclusions and directions of future work.

3. The DLR use cases

Interdisciplinary projects have become more and more important. The need for a common data exchange and problem solving platform had became eminent for a global view on multidisciplinary problems. At DLR, in each research area advanced tools are available to cope with specific problems. Most of these tools were developed over many years and are highly elaborated and accurate. In the past, multidisciplinary studies have been conducted mostly on an ad-hoc basis. This approach enabled single multidisciplinary analyses but was not powerful and flexible enough for modern demands on collaborative projects. The need to be able to build up inter-department cooperation's was well known at DLR and led to a number of projects since 2005, which were involved in developing the presented software integration development:

- Technology integration for the virtual aircraft (TIVA/TIVA-II), 2005-2008, to perform technology assessment regarding collaborative predesign of civil aircraft. The main goal of this project was the definition of a common parametric data model of an aircraft, suitable for application in the preliminary design phase. A suite of tools was adapted to this data format and integrated into a process automation framework;
- Climate optimized air transport system (CATS), 2008-2012. The goal of this project is to optimize aircraft and aircraft missions regarding their climatic impact. In this project the technology described in this paper is combined with another tool developed for climate assessments;
- Evaluation of innovative aero-engines (EVITA), 2008-2011. The technology and data used in this project is quite similar to TIVA, but is focused more on the preliminary design of engines rather than on the whole aircraft;
- Unmanned Combat Air Vehicle (UCAV-2010/FaUSST), 2007-2012. This project simulates the development of a military aircraft optimized for stealth capabilities. Existing knowledge gained in the TIVA project is taken into account. The central data set was adapted to the additional requirements, and an expert system was added.
- Virtual Aircraft Multidisciplinary Analysis and Design Processes (VAMP), 2009-2012. This project is the successor of the TIVA projects and concentrates on the extension of the developed technology and more common problem solutions like data provenance or data management. Also non-technical problems like usage limiting are engaged.

3.1. The DLR Project EVITA

The DLR started the internal project EVITA (Evaluierung innovativer Turbo-Antriebe) to bundle the existing competencies in the area of aero engine predesign at DLR. The ambition of EVITA is to enable DLR to conduct preliminary multidisciplinary aero

engine predesign studies by taking into account – amongst other disciplines – aero- and thermodynamics, acoustics, structural design and material science. This will expand the DLR capabilities to evaluate and analyze different new engine concepts at a very early stage of development. In order to reach this goal in a limited timeframe, detailed methods like CFD-analyses are not suitable and too complex. Instead simple, robust and reliable tools are developed and will be combined to a scientific process chain.

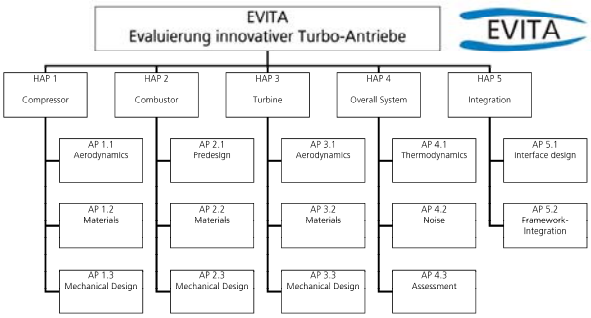

Figure 1: Work Breakdown Structure of EVITA

The Work Breakdown Structure of EVITA (Figure 1) has been selected to cope with the variety of work-packages and disciplines required. The first three main work packages (HAP) are dealing with the main aero engine components, whereas the forth main work package contains all cross-cutting issues.

Each of the main work packages 1-3 is split in three disciplines. The Aerodynamics/Predesign work packages (WP) deal with the aerodynamic design of the individual components. These work packages are led by the DLR Institute of Propulsion Technology. The goal of the materials work packages is to supply the mechanical design WPs with the required material data and to provide research results on new materials at a very early stage of engine design. These WP are led by the DLR Institute of Material Research. The aim of the mechanical design WP (DLR Institute of Structures and Design) is split into two parts. First, it is necessary to ensure that the aerodynamic predesign results are feasible in terms of solitidity and stability. Second, the layout of discs, spools and casing is defined here.

In HAP4 the Thermodynamics WP (DLR Institute of Propulsion Technology) is responsible for the performance evaluation and synthesis and - in collaboration with the aerodynamic disciplines - for the layout of the cross section. The Aeroacoustics work package (DLR Institute of Propulsion Technology) contributes methods for noise estimation whereas the assessment WP (Institute of Air Transportation Systems) contributes methods to evaluate the engine concepts from an aircraft point of view in terms of costs and snowball-effects.

The goal of HAP 5 is the integration of the tools of HAP 1-4 into the software framework described in this paper.

Many predesign tools applied in the EVITA project are new developments. Additionally, in some disciplines, existing tools are refined and updated. The major tools of this project (aerodynamic compressor design, aerodynamic turbine design, gas turbine performance analysis code and combustor design) will be presented in detail on this years DGLR conference [6], [7], [8].

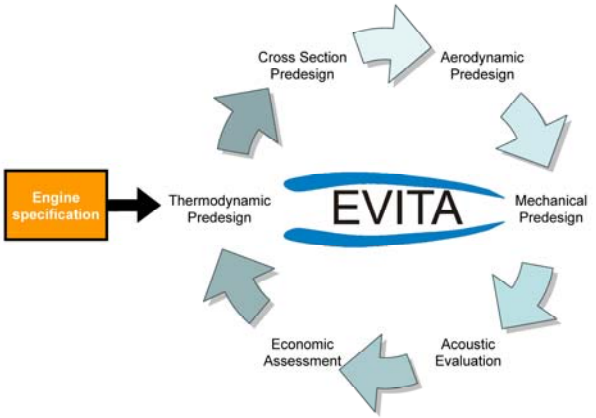

Figure 2: Simplified typical process chain in Aero-engine predesign

Until now, the focus of EVITA was the development of the individual predesign tools as well as the identification and the development of the requirements for preliminary aeroengine design. Currently, the selection and buildup of a predesign process chain between the disciplines, the experts and the tools has become the main focus of EVITA. In a first step, the participating disciplines have to be linked together very closely. For each engine component, predesign groups are initiated in which the specialists of the DLR will work closely together to establish understanding for the requirements and limits of the adjacent disciplines. A brief overview of one of the first test cases is given in [7].

The design process (Figure 2) is basically a highly iterative loop over all the disciplines involved in EVITA. This process is very complex and must react to specific design requirements. Thus, the process is not one fixed process-chain, but a flexible system which is adaptable to different tasks.

At the beginning, detailed engine specifications are required in order to start the predesign process. In a first step the thermodynamic predesign uses a

knowledge database with simple estimations for the component behavior. After a thermodynamic cycle and corresponding mach numbers are set, the simplified engine cross section can be derived. The aerodynamic tools are now able to generate the aerodynamical design of the individual engine components. The aerodynamic predesign is on the one hand iteratively connected to the mechanical design to avoid infeasible results at a very early stage of development and on the other hand to the thermodynamic cycle design, updating the simple estimations for the component behavior. Depending on the engine specifications an acoustic evaluation is very essential in the predesign phase to take noise effects into account. The economic assessment of an engine is another important point to evaluate in engine design. Each of the disciplines has an effect on the overall design, thus many complex iteration loops and sub-loops are required to reach a feasible result.

The enormous complexity of such predesign tasks requires a well planned system. Besides a good collaboration of the individual specialists, the multitude of very different tools must be linked together more closely. Therefore a suitable software framework is essential and will be described in the following.

In the last years, a software solution for such tasks has been developed within DLR, especially in the project TIVA. This framework presented in the next chapters is currently focused on conceptual aircraft design and will be adapted to the requirements of aeroengine design.

4. A Common Database for the Preliminary Predesign Phase

The common database for all applications involved in a multidisciplinary simulation is realized by a central data set. The central data set comprises a main file formatted according to the newly designed XML data format CPACS (**C**ommon **P**arametric **A**ircraft **C**onfiguration **S**chema) [9]. Additional data files in arbitrary formats can be linked into the central data set via file paths. Each application reads its input data from the central data set. Also the applications have to write back their results to this file. A software suite, described in Section 6, has been developed to handle the data transfer between the central data set and the applications.

A file stored in the CPACS format could describe one or several configurations with necessary data for the initial design phases. The configuration is defined by different, hierarchically ordered construction units, for example engines, wings or fuselage. Additionally, the format defines how mission data and simulation results of the applications are stored in the central data set.

5. Supporting Software Libraries for Implementing the Common Data Format

This section describes the software tools developed to make the communication of the simulation tools with each other via the CPACS data format possible. As shown in Figure 3, the following libraries and software components were developed:

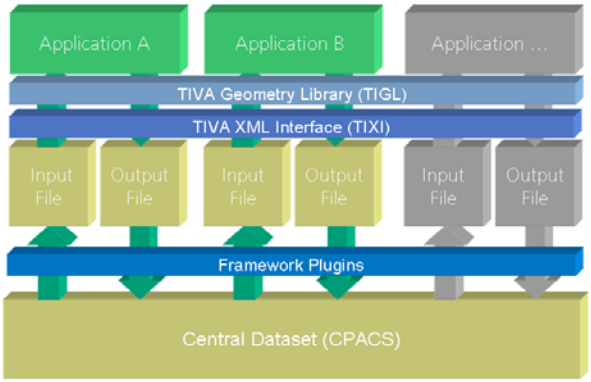

Figure 3: Overview of the data flow between the central data set and the applications.

5.1. TIXI: A Library for simple XML Access

The interface to the central data set is completely based on the XML for-mat. To enable already existing applications without the ability to access XML formatted files to read and write XML based data there are two possible approaches. The first approach is to enable the application itself to read and write XML data. The second is to write filter programs that translate XML data into the format native to the application for input and again translate the native output into XML. This approach shields the application developer completely from dealing with XML when the filter programs are written by XML specialists based on the description of the native format. However, this procedure has the disadvantage that changes of the native format or changes in the application must reflect themselves in the filters which will inevitably lead to additional effort to adapt the filter programs regularly. The first approach is

liable to suffer from the complexity of the API of libraries used to process XML data like libxml2 [10]. On the other hand this approach has the advantage that there is only one place where changes have to be implemented.

A requirement analysis showed that file formats used by the applications are based on quite simple data structures. Input files often comprise single floating point or integer numbers only and their meaning depends solely on the position in the file. More advanced input parsers implemented in these applications are able to handle name value pairs. Sometimes lists of numbers and other aggregates comprised of simple types like vectors, arrays, or matrices have to be read and to be written.

Based on these requirements, it was decided to design a simple API to access XML files for performing these tasks. The provision of a simplified interface to an existing XML library hides the complex API of a full-fledged XML library from the application programmer. Nevertheless, advanced users will be able to use the complete API of the underlying XML library for manipulating the XML files. Another reason to supply an own API was that existing XML- libraries provided very little support to read and write XML files from Fortran programs.

The simplified XML processing library can be directly integrated into a code by the application owner or can be used to create filter software which can be maintained by the application owner himself. Although simplified and some-what restricted compared to a full-fledged XML processing library the user can, for example, create documents, create and delete nodes, and add and remove element attributes. Routines to read and write simple text nodes and addi-tionally specialized nodes holding integer and floating point numbers are part of this API. Furthermore, routines to process aggregates of these simple types have been implemented. For the processing of geometric data, reading and writ-ing of multidimensional arrays or arrays of vectors, i.e. coordinates of points are supported. The C API and the API for Fortran provide the same functionally to the programmer. Additionally the library could be used from JAVA program, Python scripts and Matlab routines. The library has been designed to hide the implementation details so that the underlying XML library, currently libxml2, can be replaced by another one without changing the XML processing API in the applications.

5.2. TIGL: A Library for Processing Geometric Data

In order to perform the modeling of wings and fuselages as described in Section 2 as well as the computation of surface points effectively, a geometry library was developed in C++. The library provides external interfaces for C and FORTRAN and could be used from JAVA, Python and Matlab as well. Some of the requirements of the library were:

- Ability to read and process the geometry information stored in a CPACS file,
- Possibility to extend to other geometrical characteristics (i.e. engine pods, compressor/turbine blades, landing gear),
- Ability to build up the three-dimensional (airplane) geometry for further processing,
- Ability to compute surface points in Cartesian coordinates by using parameters, segment numbers or unique identifiers,
- Possibility to be expanded by additional functions such as area or volume computations,
- Possibility to export the airplane geometry in the more common CAD file formats like IGES or VTK [11], [12].

The developed library uses the Open Source software OpenCASCADE [13] to represent the airplane geometry by B-spline surfaces in order to compute surface points and also to export the geometry in the IGES format. OpenCASCADE is a development platform written in C++ for CAD, CAM, and CAE applications which has been continuously developed for more than ten years. The functionality covers geometrical primitives (for example points, vectors, matrix operations), the computation of B-spline surfaces and boolean operations on volume models.

Apart from the already specified requirements above, the geometry library offers query functions for the geometry structure. These functions can be used for example to detect how many segments are attached to a certain segment, which indices these segments have, or how many wings and fuselages the current airplane configuration contains. This functionality is necessary because not only the modeling of simple wings or fuselages but also the description of quite complicated structures, i.e. in aero engines, is targeted. For an example see Figure 4.

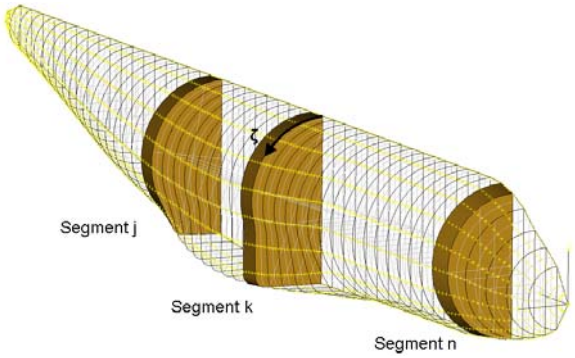

Figure 4: Parameterization of the aircraft fuselage.

5.3. TIGLViewer: A tool for Visualizing Geometric Data

In order to review the geometry information of the central data set a visualization tool, TIGLViewer, was developed. A screenshot is shown in Figure 5. The TIGLViewer allows the visualization of the used airfoils and fuselage profiles as well as of the surfaces and the entire airplane model. Furthermore, the TIGLViewer can be used to validate and test the implemented functions of the geometry library, for example the calculation of points on the surface or other functions to check data that belong to the geometry structure.

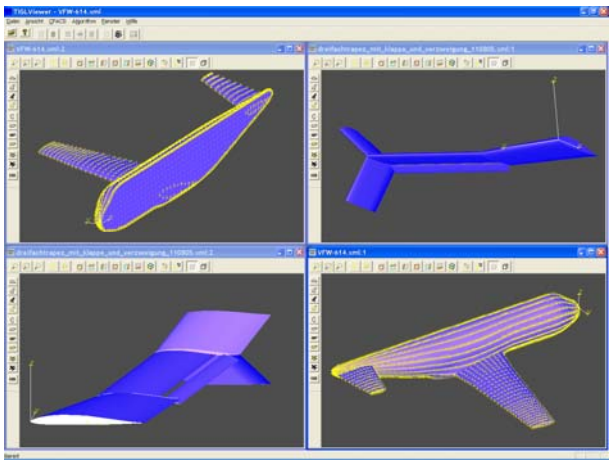

Figure 5: Screenshot showing the visualization of various airplane geometries with the TIGLViewer.

6. Integrating the Applications into a Framework

In order to run applications within an automated workflow that also considers their dependencies a software integration framework. Such a framework offers an environment for the integration of existing software components and take care of network communications, data transfer and management and centralize the overview of all parts of the running system. Modern integration frameworks such as the open source framework RCE [14] or Phoenix Integration's ModelCenter [15] provide a graphical editor for setting up workflows visually. The different applications are assembled to a workflow by the framework and are linked via their input and output variables. A change in an output variable triggers the execution of all components which depend on this variable as input data.

At first ModelCenter was chosen in DLR for integration of the CPACS data format to enable scientists to use the CPACS data schema within a framework. This was done because ModelCenter was ready to go and comes with a lot of build-in optimizers and helping components so that we only needed to develop a set of plugins for data integration and code execution. Unfortunately we learned that the possibilities with a closed system are limited and that ModelCenter is not capable of handling such a flexible system that is needed. There are lots of challenges not genuinely tackled by the initially chosen integration platform, e. g., management of data sets larger than a few dozen MB or user privilege management. Because of this inadequateness ModelCenter may be phased out during coming projects and replaced by the Remote Component Environment (RCE), which provides a more powerful and versatile platform to integrate simulation software together with optimal fitted post-processing and visualization.

6.1. Remote Component Environment

The purpose of Remote Component Environment (RCE) is to provide a framework with base software components where diverse and specific applications can be embedded [16]. This allows an integrated application to simply use the already realized base software components like data management, workflow engine, or privilege management. Thus, the integration platform enables basically independent applications to interact with each other and to operate on the same data.

RCE was developed by DLR at the Simulation and Software Technology division in cooperation with the Fraunhofer Institute for Algorithms and Scientific Computing (SCAI). It was designed for the engineering

area, which mainly consists of simulation applications that utilize numeric algorithms which run on different resources by different users. Therefore, it includes the base software components workflow engine, data management, privilege management, and distribution. RCE adopts the OSGi technology (Open Services Gateway initiative) [17], the industry standard for modular dynamic Java applications. Thus, it is an extensible, component-based platform written in Java. The component-based architecture allows RCE to adapt to the environment it is deployed in. In order to ease development of graphical user interfaces (GUI), it is also based upon the Eclipse Rich Client Platform [18], a GUI framework providing base graphical elements.

In the following RCE will be explained in more detail with focus on distribution with regard to workflow engine and data management.

The distribution component facilitates a distributed RCE setup. In this case the integration platform consists of multiple RCE instances, at least one running on every participating host. Different applications can be integrated into every RCE instance, which hence may have different characteristics (e.g., application server, client (GUI), database server). All the RCE instances communicate with each other in a decentralized way. This allows all kinds of network topologies at the application level.

RCE is transparent regarding access and location. This means that remote and local services can be accessed with the same operation and that the location of a service s concealed from the use [19]. E.g., the workflow editor provides a list of available applications to couple, which is transparently aggregated by calling the local and all remote workflow engine services. Thereby, all RCE instances will be considered which are known by the client via its configuration. Next to instances which run on servers, it is also possible to involve RCE instances which run on a colleague's laptop as long as this laptop is reachable on network layer. Another example illustrating the distribution concept of RCE is the data management. Among others it is used to store data produced by applications as part of a workflow. This data can be transparently accessed by other clients. Hereby, a view on the data exists, which again is provided by calling the local and all known remote data management services. Next to listing the data, it is also possible to directly open the underlying files within RCE even if they are stored remotely.

The distributed setup of the Remote Component Environment at DLR looks as follows:

Users install an RCE instance on its desktop computer pre-configured as client knowing all RCE server instances at DLR.

Each contributing institute or department at DLR installs an RCE instance configured as server running on a physical server machine and offers their domain specific simulation software as services available inside DLR's network. It can then be accessed from and RCE client graphical user interface.

RCE is designed as a universal problem solving environment. In order to make use of it in specified application domains it is required to extend it with domain specific components and user interfaces. RCE plus these additions form an application domain specific software. For EVITA, the following components are added to RCE:

For the server software a Java components were developed to enable scientists to remotely run any tool or application they might need from within the workflow building environment of ModelCenter. This functionality is originally built into ModelCenter, but the plugin contains additional data handling necessary to provide an easy way to exchange and access CPACS data.

This again is simplified by a set of easy to use Java plugins within the graphical project editor (Figure 7). Some of these are described below and in the user handbooks that come along with the component suite.

6.2. The CPACS integration component suite

The CPACS integration component suite encompasses all Java plugins designed to simplify working with CPACS data from within the RCE integration environment. The following list sums up the most important plugins developed so far, with more to come:

Tool-Wrapper. The tool wrapper is the main interface between the framework and the requested application. It consists of several abstraction layers and a separate configuration file. The concept itself is hereby called 'tool wrapping'. Wrapping a tool or application is hereby a piece of software which encapsulates the base application and hides it from the environment outside of the tool wrapper. The primary advantage is to have an abstraction layer for the outside environment which again can use the interface provided by the tool wrapper. In this implementation type the tool wrapper needs to be customized for the

wrapped application. The CPACS data format represents the whole set of all exchange data and builds the base for all operations.

Splitter Components help to extract single values from the complete CPACS data set for connection with optimizers.

Merger Components complements the splitters by updating single values in the CPACS data set, e.g. to build up a loop over a chain of tools.

Joiner Components are to merge the CPACS of two process branches together. Therefore a user must define the logical part of the merge.

Source Components enables the user to read in a data set from various places like the local file system, making use of the TIXI library internally and checks the validity of the initial data set.

Destination Components makes it possible to store computation results. Additionally the intermediate results of each run of an optimizing loop can be logged with this component for later offline analysis.

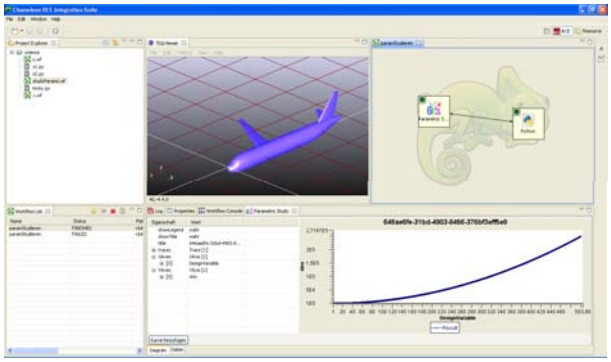

Figure 6: Screenshot of the RCE integration

7. Summary

Using the example of preliminary engine design, the demand of a flexible software solution to support complex predesign tasks has been shown.

The software tools described in this paper have been used to setup a process chain consisting of applications with a low level of complexity for computing lift, structural parameters for wings, the resistance against buckling, and the behavior at forced landing on water. Also, on base of the more advanced framework RCE, it is planed to model the Preliminary Aero Engine Design phase of the previous described EVITA project.

Planned extensions of the CPACS format include the storage of abstract aspects for example cost factors or the description of the mission to be simulated.

The wide field of collaboration is, beside the core integration technology, the next main approach we want to address in our software suite. Beside of an adaption of our existing software tools to the preliminary aeroengine design, our future work will be about the integration of provenance information and to provide a possibility to semantic checks during creation of workflows.

8. References

[1] Hoheisel, A., *Model coupling and integration via xml in the m3 simulation*, International environmental modelling and software society IEMSs, Lugano, Switzerland, 2002, pp. 611-616.

[2] OMG. CORBA Architecture and Specification. OMG, 1998.

[3] Extensible Markup Language: http://www.w3.org/XML.

[4] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock. *Kepler: An extensible system for design and execution of scientific workflows*, In SSDBM, pages 21–23, 2004.

[5] Hull D., Wolstencroft K., Stevens R., Goble C., Pocock M.R., Li P., Oinn T. *Taverna: a tool for building and running workflows of services*, Nucleic Acids Res. 2006;34:W729–W732.

[6] [PLANNED] Tietz, S.; Behrendt, T.: *Development and Application of a Pre-Design Tool for Aero-Engine Combustors*, DLRK2011-241190, DGLR, 2011.

[7] [PLANNED] Becker, R. ; Wolters, F.; Nauroz, M.; Otten, T. : *DEVELOPMENT OF A GAS TURBINE PERFORMANCE CODE AND ITS APPLICATION TO PRELIMINARY ENGINE DESIGN*. DLRK2011-241216, DGLR, 2011

[8] [PLANNED] Krumme, A.: *CONCEPTION OF A PROCESS CHAIN AND DEVELOPMENT OF A PROGRAM FOR AXIAL TURBINE PREDESIGN*, DLRK2011-241181, DGLR, 2011

[9] Böhnke, Daniel und Litz, Markus und Rudolph, Stefan (2010) *Evaluation of Modeling Languages for Preliminary Airplane Design in Multidisciplinary Design Environments.* DGLR 2010, 31.08.2010 - 03.09.2010, Hamburg, Germany.

[10] libxml2 Homepage: http://xmlsoft.org .

[11] Initial Graphics Exchange Specification (IGES): Initial Graphics Exchange Specification (IGES) Version 4.0, NBSIR 88-3813, 1988.

[12] Visualizing with VTK: a tutorial, Schroeder, W.J. Avila, L.S. Hoffman, W. Computer Graphics and Applications, IEEE On page(s): 20 - 27 , Volume: 20 Issue: 5, Sep/Oct 2000

[13] OpenCASCADE homepage: http://www.opencascade.org/

[14] German Aerospace Center (DLR), “Remote Component Environment,” [accessed, May 29, 2009]. [Online]. Available: http://www.rcenvironment.de/

[15] ModelCenter product homepage: http://www.phoenix-int.com/products/modelcenter.php

[16] Forkert, Tomas und Kersken, Hans-Peter und Schreiber, Andreas und Strietzel, Martin und Wolf, Klaus (2001) *The Distributed Engineering Framework TENT*. In: Vector and Parallel Processing - VECPAR 2000: 4th International Conference, 1981, Seiten 38-46. Springer-Verlag. VECPAR 2000, 2000-06-21 - 2000-06-23, Porto, Portugal.

[17] OSGi Alliance, OSGi - The Dynamic Module System for Java. http://www.osgi.org .

[18] Eclipse Foundation. Eclipse Plug-in Development Environment. http://wiki.eclipse.org/pde/.

[19] J. Dollimore, T. Kindberg, and G. Coulouris, *Distributed Systems: Concepts and Design* (4th Edition) (International Computer Science Series). Addison Wesley, May 2005.