WITH PUBLIC DOMAIN SOFTWARE TO INTEGRATED DESIGN AND ANALYSIS TOOLS

M.C.Haupt, W.Heinze, P.Horst
Institut of Aircraft Design and Lightweight Structures, TU Braunschweig,
Hermann-Blenk-Str. 35, 38108 Braunschweig, m.haupt@tu-bs.de

Keywords: *software tools, multidisciplinary analysis, airplane design*

Abstract

This papers describes a tool integration approach for the airplane design, using public domain software. The developed architecture, the applied tools and applications are shown to demonstrate the power of this approach for engineering applications.

1 Introduction

Through the increasing use of the internet, the source codes of software projects are more often made available and can be used (almost) free of charge. This way is interesting from view of the licensing (costs) and the extendability. The high quality of such software was proofed by the development of the operating system Linux, which is accepted as a professional software system. How to achieve flexible and challenging software tools for complex design tasks, is shown with examples from the airplane design.

2 Motivation

2.1 Requirements of the Airplane Design

The goal of the airplane design is the determination of the best airplane layout for a defined transport task by a multidisciplinary analysis. The classic kind of the assessment often happens by means of the DOC, other criteria particularly in case of part tasks are conceivable.

In the context of these objectives, there are therefore different design scenarios, which are to be accomplished depending on:

- Examination of the feasibility of a configuration (basis solution)
- Optimization of the basis solution
- Assessment of changes in the transport task
- Rate the employment of new technologies
- Comparison with alternative configurations at equal analysis precision

Further tasks arise through new challenges:

- Recognizing and analysing problem fields already in the concept phase, e.g. by using more precise analysis models during the concept analysis that are based on automated model generation.
- Early simulation of technical problem fields of the current aircraft design with visualization possibilities such as
  - emergency evacuation
  - dynamic behaviour of the structure
  - kinematics of the landing gear
  - shower water impact on the engines
  - flight dynamic of critical manouvers
- Consideration of further, nonclassic aspects of the market success
  - design and variability of the cabin
  - ground service of the airplane (operation of standard ground vehicles, optimization of ground times)

– compatibility of the airplane with the airport infrastructure (simulating these problems with the customer)

2.2 Problem

The total task the computer-aided aircraft design is of high complexity, both on the part of the single models with their extensive description data and on the part of the model interactions. In general, different self developed and commercial modeling and analysis tools are used, which have an intensive data transfer with various interfaces between them.

Essential aspect of the computer-aided design also is the possibility of the easy transition of simple to higher-order modeling and analysis tools in order to increase the precision or validate the results of simple procedures if required.

For new design tasks, the diverse modeling and analysis tools must be connected newly together. A manual execution of such new workflows is error-prone and must be avoided.

2.3 Objectives

A software environment, which is able to implement such design processes, must fulfill some elementary requirements, so that the design processes can be performed efficiently and reliable.

The possibility of the integration of individual tools is important for the automation of the work flow of the design tasks in a single environment. For that purpose, a flexible data transfer between the tools is necessary and the ability to embed arbitrary tools that are available as different source codes in `C, C++` or `Fortran` or as complete executables.

Important aspects for the applicability are the simplicity and the clearness as well as the flexibility and the robustness the integration procedure. It therefore also guarantees an efficient rapid prototyping for new part tasks.

For such a software environment, which was intented here, available and powerfull technologies and tools are used. In order to put the development from the beginning on a broad basis, the approach depends on suitable public domain software.

3 Public Domain Software

The name public domain software refers here on open source software and not on Freeware, shareware etc. An exact definition the open source is (still) not available and therefore different license models of the open source model (GNU Public License, retain the copyrights, free only for research,...) exist under it may be used.

Essential aspect the open source idea is, that the source code is free for interested people at no charge for the use and to the understanding, the extension and the error search and last but not least for the documentation and portation to other hardware platforms or operation systems.

Due to a very diverse and great number of users, one hopes for functionality and stability that remains secured over the complete life cycle. In spite of the openness of the sources, and therefore the free use, exist the possibilities of commercialization. They arise through additional services, e.g. during the support and training as well as during special portations and extensions.

Problems have proven to be: the discover of suitable software in the internet as well as the judgment of the capacity and the documentation, to facilitate the selection of available tools.

The success of this software development model is demonstrated by the operating system `Linux` (Unix on Intel-, Alpha-hardware ..) or by the widely-used webserver software `Apache`.

4 Integration Environment

The integration environment presented here is to be understood as a collection of suitably arranged tools. The basis is formed by the core parts, which provide the needed software technologies. They do not contain any engineering-specific aspects, therefore this environment can be also used for arbitrary other tasks.

Extension packages are added, which were developed for the pursued design and analysis work or which were adapted as independent tools

at the kernel of the environment.

The environment is completed through a universal, graphic users interface. Through an interactive working and the integrated possibility for visualization the user-friendlyness increases.

4.1 Core Parts

The four central core parts that form the software-technical platform of the integration environment are:

- `Python`, a script language,
- the standard packages of `Python` (tkinter, Numerical Python, ...),
- the Visualization Tool Kit `vtk`,
- Interface generators: `pyfort` and `swig`.

These general parts are all open source software and are used unchanged. In the following, they are characterized by some keywords.

4.2 `Python`

`Python` is an object-oriented script language that mixes features of the software technology of traditional languages with the usability of script languages [1].

- Features:
  `Python` has an extensive set of diverse data types and supports namespaces, exceptions, dynamic loading and object-oriented programming. A huge number of modules exists that make all modern information technologies availably.
- Syntax:
  `Python` has a simple syntax that does not make it necessary to use cryptic symbols ($, %, etc.) or a completely different syntax than that of `C`.
- Small core:
  `Python` has a modular structure that makes it easy to load or delete modules as required or to reload newly after a change. By this means, it can be used only with the needed modules; monolithic systems are therefore avoided.
- Documentation:
  is available in the book shops in form of many books. Tutorials, language and library references and many further documents are availably in the internet.
- Support and stability:
  `Python` has proven to be very stable. A fast support is given through the news groups and special interest groups (SIG).
- Free available and portable:
  The available source's code (Ansi `C`) is modifiable as required and runs on every common platform (Unix, Windows, ...).
- Increasing acceptance:
  For multitude of program libraries `Python`-bindings were carried out. Many new scientific computing projects use `Python` as script language.
- Integration:
  `Python` can be integrated into a number of further languages and vice versa. Thus the power of other languages can be utilized and their weaknesses can be circumvented.

4.3 Visualization Tool Kit

The Visualization tool Kit `vtk` is an object-oriented, portable software for 3D computational graphics, visualization and image processing, which is comparable with `AVS` or `OpenDX` in their graphic functionalities [2].

- Architecture:
  `vtk` consists of two major parts: A compiled core (`C++`) and wrapper that were produced automatically for `Tcl`, `Java` and `python`. Learning and use in connection with script languages is very simply and efficient.
- Graphical model:
  `vtk` has render-windows, renderers, 2D/3D actors, properties, lights, cameras and mapper for geometric or volumetric data.

- Visualization pipeline:
  It transforms information into graphic data with the help of data objects (general data, structured and unstructured grid data) and Process objects (source, filter, mapper) that convert according to a variety of methods input into output data (Fig. 1).
- Documentation:
  An extensive online documentation is available. Two reference books explain the functions, techniques and backgrounds very well. In order to understand the numerical techniques implemented in filters the look in the source texts very is helpful.
- Support und stability:
  `vtk` has proven to be stable. A fast support is available through news groups, special interest groups (SIG) or commercially.
- Free available and portable:
  The available `C++` source code uses `OpenGL` and runs on every common Win-Tel or Unix platform.
- Increasing distribution and extendability:
  `vtk` is used in many projects and can be extended simply due to the object-oriented `C++` implementation.

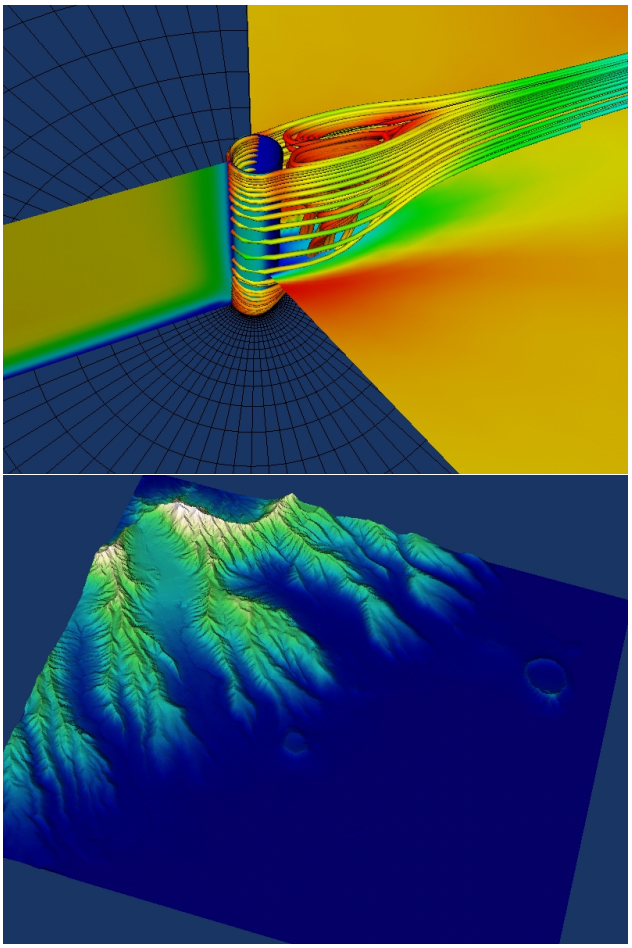

Fig. 1 `vtk` Visualization examples:
flow visualization / geometry of Hawaii

The combination of the different objects to a network and/or a pipeline can be done without deeper knowledge of object-oriented programming. The implicit execution mechanisms of `vtk`, when a process object must transform input data into new output data, are adopted for the self developed, `vtk` independent extension packages. `vtk` defines therefore decisively the design of the integration environment.

4.4 Standard Packages of `Python`

- `tkinter`
  provides the widgets of `Tk` under `Python` [3]. It is contained in the `Python` distribution, so that powerful, platform-wide graphic users interfaces (GUI) can be made without further packages.
- `Numerical Python`
  provides vector- and matrix-objects and a multitude of methods to their manipulation [4]. These methods are implemented in `C` and use numerically efficient techniques e.g. from numeric libraries like `Lapack`.
- `Scientific Python`
  is a collection at scientific tools [5]. Emphasized are the bindings of the `netCDF`-library, so that writing and reading `netCDF`-formated files is very simple. Bindings for `MPI`, `ArrayIO`, `Fortran-Format`, mathematical, statistical and physics functions, which are efficient implemented in `C` or even use `Numerical Python`, are in `Scientific Python` contained as well.

4.5 pyfort and swig

The programs pyfort and swig are essential tools in order to directly integrate existing source codes in Python. They produce from simple declaration files C-code for extension modules that were integrated completely.

- pyfort
  is a tool for the Fortran-Python integration [6].
  - Simple and robust automatic construction of Python extension moduls.
  - Fortran declarations are transformed into Python / Numerical Python equivalents.
  - Declaration defines the in-/in-out-/output variables in Fortran90 syntax.
  - The used Fortran libraries can be linked statically or dynamically.
- swig
  With swig it is possible to integrate C/C++-codes in Python or as well in Perl and Tcl/Tk [7].
  - Automatic construction of Python extension modules.
  - C declarations are transformed into Python equivalents.
  - C data-types are mapped in suitable Python representations.
  - Complex C/C++ objects are handled with references.
  - C++ classes are transformed into shadow classes.
  - Static and dynamic linking is possible.

4.6 Extension Packages

The special extension modules of the integration environment may be divided up roughly into two categories. One category contains modules based only on the core parts without any engineering-specific functions. These are e.g. utk and usr.

- utk
  These modules serve for connecting vtk and pure Python objects easily. For this purpose some essential base classes were copied from vtk to Python, so that classes that were derived from these have the same base methods like the corresponding vtk-classes. This is important to garantee the pipeline functionalities, if necessary also without the use of vtk.
- usr
  Among them are modules that increase the functionality of the core parts and which are derived only from them. E.g. filters for special visualization techniques can be found here.

The packages of the second category contain modules, which provide modeling- and analysis specific classes.

- dtn
  Modules of this class make the functions of the spline library DTNURBS of the US Navy available [8]. For this purpose geometric data objects were implemented which hide the DTNURBS representations of curves, areas etc. The special process objects execute geometric operations using the DTNURBS library such as the generation or editing of geometric objects. Extensions with the help of the GridLib (NASA) allow a grid generation on the DTNURBS geometries.
- pdo
  Different simulation- and analyses techniques become availably through this package. Corresponding data- and process objects allow analyses with flow- and structure-analysis programs, as well with tools for the movement simulation of dynamic systems.

4.7 Graphical User Interface

The graphic user interface (GUI) fulfills two important tasks within the integration environment.

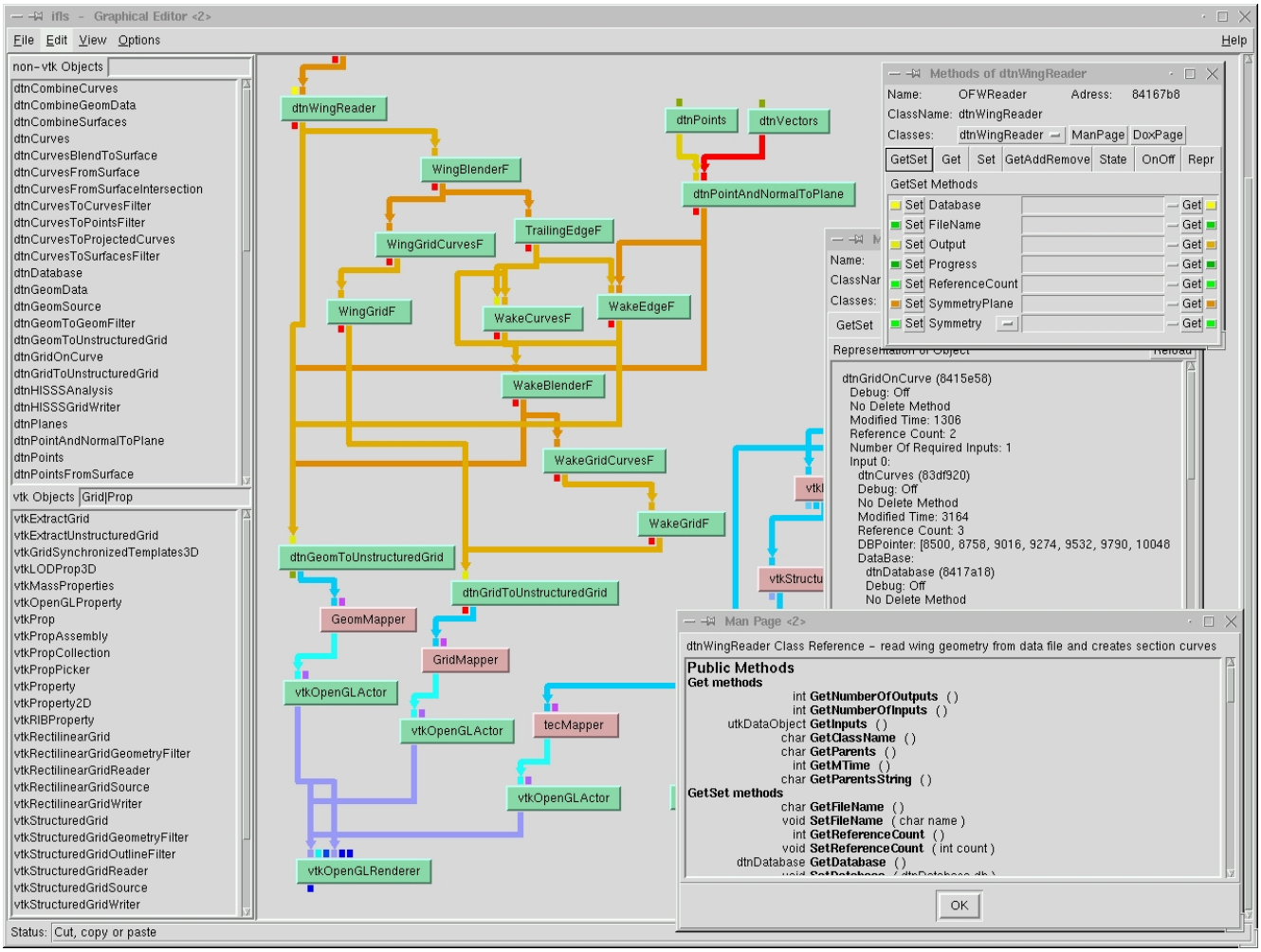

Fig. 2 Graphical User Interface with a pipeline of a geometry und grid generation (right top: object editor, right bottom: documentation based on the class implementation)

On one hand, it makes the visual programming possible, i.e. the objects can be created, deleted and interrelated interactively. On the other hand, the properties of the individual objects are presented in an uniform environment. By their methods the objects can be interactively manipulated Fig. 2. If an existing `Python`-script is executed within the GUI, the interface analyzes the existing object-relationships and presents these interactions as a graph or as a tree.

Because the GUI handles all objects in a generic way, some coding guidelines should be observed, so that new classes can be treated easily in the GUI without any integration expense. Similar holds for the automated creation of the documentation of the modules and their classes in different formats (`html, latex, postscript, pdf.`)

The GUI is not even a necessary component of the extension packages and therefore `Python`-scripts run also without the GUI in the batch mode.

5 Applications

The following examples demonstrate how the integration environment can be used and which potential lays in the open source's software. In the context of several projects, these examples and the newly required extension modules were developed specifically. They indicate how flexible the integration environment is. Arbitrary expan-

sions are conceivable in all directions. Whether existing open source can be used, should be investigated case-by-case.

In this example the main aspect is the connectivity between the integration environment and the preliminary aircraft design and optimization program PrADO of the IFL [9]. The aim is the specific examination of partial problems or the development of new methods.

5.1 Aerodynamic Analysis of an Oblique Flying Wing

Subject of this example is the aircraft configuration of the Oblique Flying Wing (OFW) type [10]. A data base in the PrADO-format describes the geometry with all control-surfaces parametrically. From these data the surface geometry of the configuration is produced with the help of the DTNURBS - package. By means of an algebraic grid generation on these geometries, at which functions from the GridLib were used, a surface discretization is produced (Fig. 3).

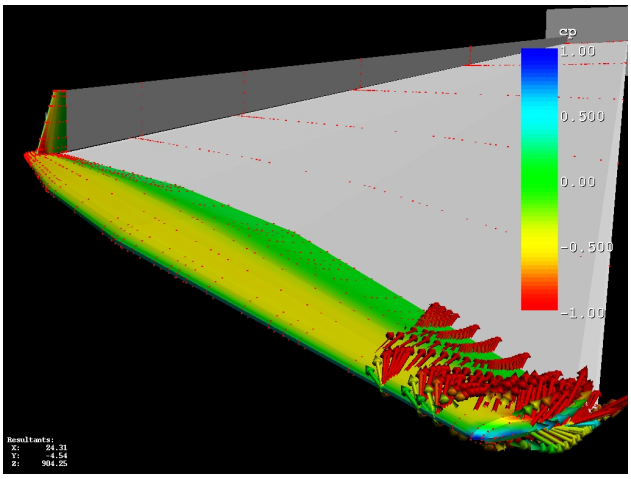

Fig. 3 Aerodynamic analysis of an OFW, pressure distribution and velocity-arrows

Since a panel method is used for the aerodynamic analysis, a wake surface between trailing edge and the Trefftz-plane is generated and also discretized. The analysis is carried out by the program HISSS from EADS (Munich) and delivers the velocity and pressure distribution on the surface. The simulation modules of the integration environment produce the input files of the HISSS-programms, start it locally or remotely on an other computer, postprocess the result files for the required data and make these available in the integration environment e.g. for visualization.

The parametric description of the geometry makes it possible to change it interactively so that variations of e.g. the profile distribution of the wing, the floor plan etc. are feasible easily and the aerodynamic results immediately. In the batch mode, the complete set of derivativa of the configuration for flight-mechanical examinations is determined this way for different flight states and flap angles. By the way, the DTNURBS library has the export possibility in the IGES format. Therefore the geometry in the PrADO format can be converted into this one.

5.2 Static Aeroelasticity of a Wing

A half wing is the subject of this application, for which the static deformation should be calculated under cruise conditions. Raw data are existing independent discretizations on the common surface for the structural and aerodynamic analysis. Optionally, the aerodynamic grid can be generated as described above by a PrADO data base and the DTNURBS extensions modules. For the analysis independent solvers for fluid and structure are applied - on the structure side FEM code NASTRAN, on the fluid side panel code HISSS -. For the coupling of these codes a control of the iterative solution and numeric methods for the mapping of the deflections and loads between the two surface grids are necessary.

For the mapping of the deflections and loads two existing filters of vtk are used: vtkThinPlateSplineTransform and vtkProbeFilter. The first filter masters - unlike the name says - the Volume-Spline method, which presents a classic technology the aeroelasticity (Fig. 4). The second performs an interpolation of the state variables at a given point with the help of the shapefunctions of the grid, for which the state is known. This

corresponds to the technology, implemented in coupling libraries (e.g. `MpCCI` [11]) The non trivial projection of a point on a surface grid is also contained in this filter in order to determine the weight factors. Furthermore, self developed conservative mapping techniques were integrated into the environment, which are described in [12].

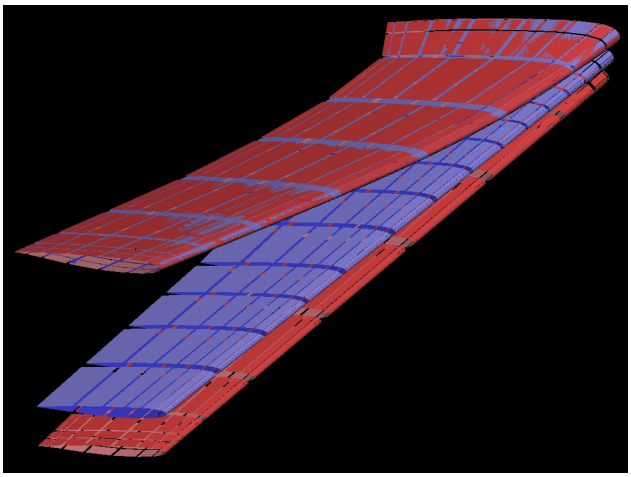

Fig. 4 Deflection transfer with the volume-spline method; bottom: undeformed (offset), top: deformed Grid (overlayed)

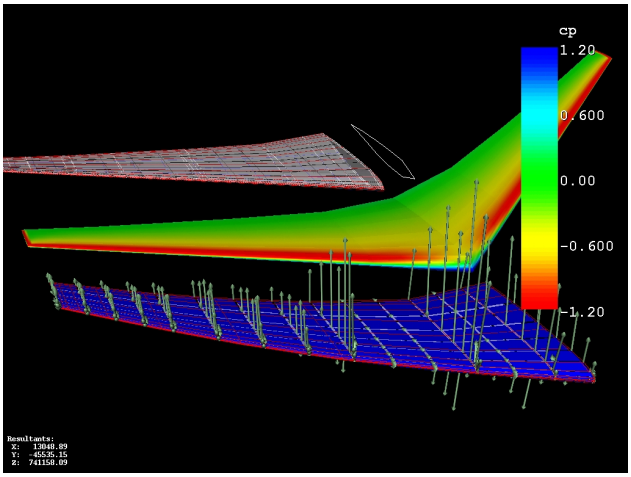

Fig. 5 Visualization of the coupled fluid structure analysis of a wing

Fig. 5 shows the visualization of the simulation results after the first coupling step. Grey in the background are the surface grids (blue: structure, red: fluid). In the middle the c_p pressure distribution can be recognized on the undeformed geometry of the symmetrized wing. The nodal forces, which result from the integration of the pressure over the surface elements and the transfer by the `vtkProbeFilter` from the aerodynamic grid to the structural grid, can be seen in the foreground as vectors, which deform the wing structure.

5.3 Airport environment

At new configurations, special aspects often must be examined specifically. This could be only the visualization of the calculated data and the simulation models, that were developed, with respect to feasiblity and correctness. For example in application of Fig. 6, based on the results by `PrADO`, the geometry model of a A380 configuration is combined and displayed with the results of the flight dynamic simulation, at which the airplane is considered as a mass point. The relationships in between the runway and the configuration (runway length, undesirable collisions with undercarriage or stern etc.) can be examined visually.

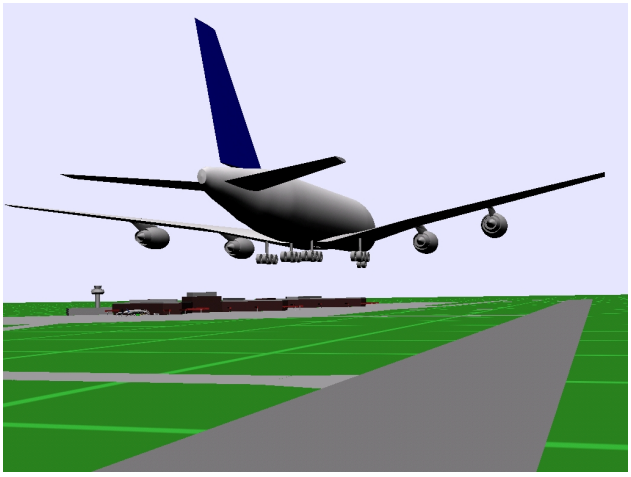

Fig. 6 Visualization of a A380 geometry in a dynamic landing simulation

A further topic that is demonstrated here in an application is the compatibility proof of new configurations with an existing airport infrastructure.

Fig. 7 shows a sonic cruiser configuration during the service at the airport terminal. Questions how fingers of the terminals dock or how the aircraft can be served by standard ground vehicles are studyable. For this purpose, every object has its own geometry and dynamics model which can be visualized in real time.

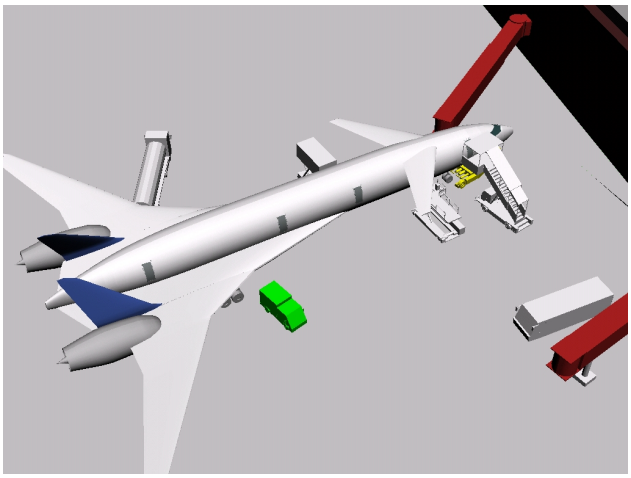

Fig. 7 Service simulation of a new airplane configuration in an airport environment

For complex service scenarios an application is under development in which the vehicles have to fulfill individual missions, having their own intelligence. The mission tasks are defined prior to the simulation and the vehicles are programmed accordingly. The paths of the vehicles are planned in the airport environment with rigid and moveable objects automatically so that one can optimize the time schedule of the service with this model. During the simulation disturbances can be introduced (e.g. further objects) and the vehicles with their own intelligence react on these. For this application the package `MissionLab` is applied.

5.4 Flight Simulation

The class library of the `JSBSim` project whose aim is it, to carry out a flight dynamic simulation in object-oriented programming [13] also is found among the integrated simulation codes. The classes include the complete three-dimensional equations of motion of the rigid airplane with ground reactions and a wind model as well as a flight control system (Fig. 8). Due to the object-oriented programming, arbitrary extensions of single classes can be integrated in order to use e.g. alternative jet engine models. The physics of the `JSBSim`-classes base on the flight dynamic model of the `LaRCSim` code of the NASA whose theory is described in detail. The flight simulation can be steered interactively for real-time applications or can be scriptbased. The complete parameters of the aircraft are provided in files in the `xml` format and can be taken e.g. in large parts from `PrADO` investigations.

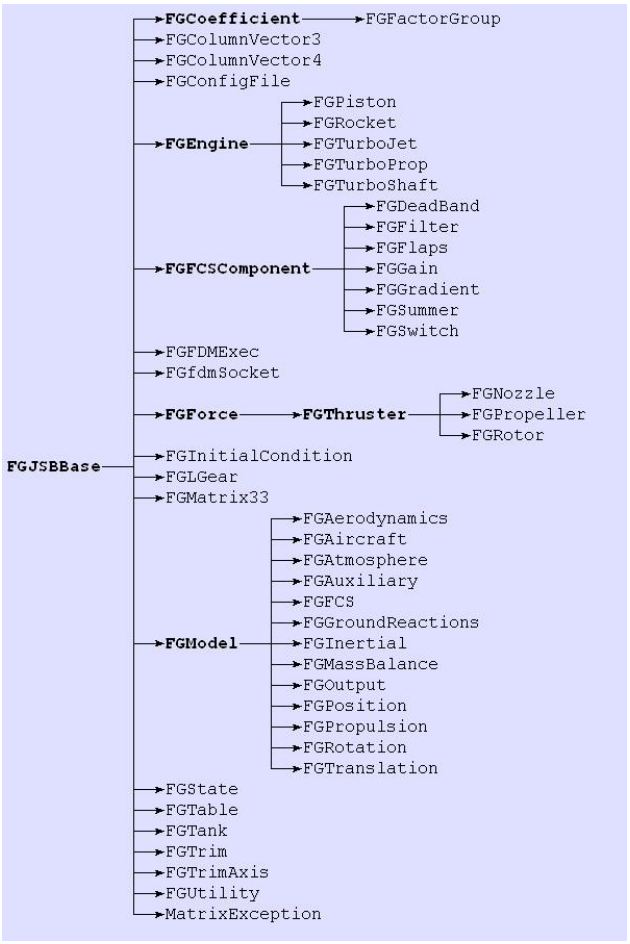

Fig. 8 Class diagramm of the `JSBSim` library

For the essential classes that describe the flight dynamic behavior, `Python` bindings were generated with `swig` so that this dynamic model can also be combined to the geometry object of the aircraft as described above. Therefore, a

flight simulator is available in the integration environment by which investigations and visualizations can be carried out, that are not possible with other simple flight simulators. Furthermore, the classes of the aircraft parts can be used, to study flight dynamic properties like the trim, the start and landing behaviour (Fig. 9).

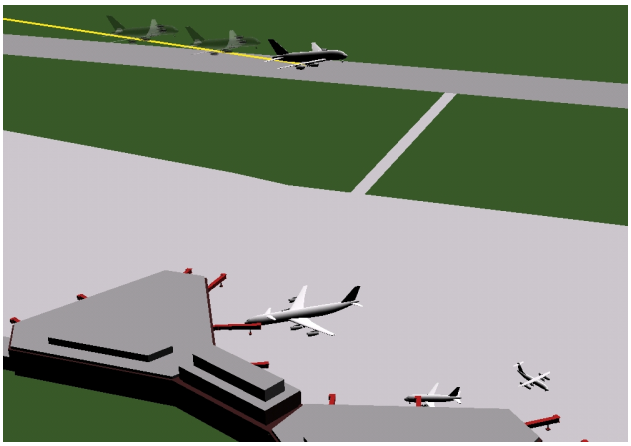

Fig. 9 Simulation of a landing with `JSBSim` moduls connected to a PrADO geometry

6 Concluding Remarks

At the current time, at which the presented integration environment is still under development, the conclusion can be drawn, that an efficient integration environement with public domain software is possible. The core parts that were outlined above just proved to be a good choice during the diverse applications due to the flexibility and the integration behavior. They are well developed and represent the state of the art of the software technology, because of the large developer and user community.

The situation is more difficult at the available open source analysis codes. Although, they are often very efficient but limited to specific problem classes. Here it necessary, to make a careful selection and not to be afraid, to create required extensions. It appears to be important, that with open source software the engineering problems are not solved automatically at no expense. The application of the scientific software requires the technical understanding of the user.

The shown integration environment will be used during research projects in the future. These come from the areas of the fluid structure coupling at spacecrafts and from the detailed examination of OFW configurations. The benchmarking, the embedding and the extension of analysis codes are also scheduled in this context. Synergy effects with a view to the architecture of similar environments are expected. The projects `Salome/OpenCascade` [14] and `ActionFactory` must be mentioned here, too [15].

References

[1] http://www.python.org

[2] http://www.kitware.com/vtk/

[3] http://www.python.org/topics/tkinter/

[4] http://sourceforge.net/projects/numpy

[5] http://starship.python.net/crew/hinsen/scientific

[6] http://sourceforge.net/projects/pyfortran

[7] http://www.swig.org

[8] http://ocean.dt.navy.mil/dtnurbs/dtnurbs.htm

[9] W.Heinze, C.M.Österheld, P.Horst: Multidisziplinäres Flugzeugentwurfsverfahren PrADO - Programmentwurf und Anwendungen im Rahmen von Flugzeug-Konzeptstudien, DGLR-Jahrbuch 2001, Band III, DGLR 2002

[10] A.J.M. Van der Velden: Aerodynamic Design and Synthesis of the Oblique Flying Wing Supersonic Transport; PhD Thesis, Stanford, USA, Juni 1992

[11] http://www.mpcci.org

[12] M.Haupt, P.Horst: Coupling of fluid and structure analysis codes for air- and spacecraft applications; First MIT Conference on Computational Fluid and Solid Mechanics 2001, pp.1226-1231, Elsevier Science, 2001

[13] http://jsbsim.sourceforge.net/main.html

[14] http://www.opencascade.org

[15] http://sourceforge.net/projects/actionfactory/