ICAS-98-6,4,4

A98-31680

# FLYING OBJECTS – AN OBJECT ORIENTED TOOLBOX FOR MULTIDISCIPLINARY DESIGN AND EVALUATION OF AIRCRAFT

A. Schneegans, O. Kranz
PACE Aerospace Engineering and Information Technology GmbH, Germany

## Abstract

The following paper presents an innovative development project of a software program family for aircraft modelling, design and evaluation. The overall goal is to provide for a consistent data model of the aircraft geometry and the associated attributes. The data model shall supply the information to a toolbox of evaluation and design sub-processes. This program architecture shall form the basis for a family of program applications which share one single data model. Software design drivers which determine the underlying architecture include the demand to smoothly integrate arbitrary data structures into a single project hierarchy, the automatic communication of relevant changes in the data structure and the providence of uniform visualisation techniques, nevertheless keeping up the access performance. The introduction of an object and attribute oriented data management, a commercial 3D geometric modeller and powerful interfaces for the graphical presentation give the basis for the program system. First derivatives of the program family include a mission performance calculation application with flexible input filters for legacy data, a cabin configuration tool for automated generation of 2D cabin drawings and 3D visualisation of the cabin interior as well as the first version of a comprehensive design and evaluation tool which shall replace existing, conventional systems.

## 1. Introduction

The assessment of the technological, operational and economic viability of an aircraft is crucial in today's aviation industry no matter whether an aircraft manufacturer, a systems supplier, an operator, a research institution or a certification authority is considered. The ability to evaluate aircraft performance, cost as well as the impact of new technologies is a basic requirement to all of them.

The scope of related tasks extends from market and competitor analysis, sensitivity studies and trade-off investigations to the matching of engines and airframes as well as to conceptual design studies; the latter mostly in case of aircraft manufacturers. A multidisciplinary approach is required to accomplish these tasks. Other requirements stem from ongoing trends in the industry towards concurrent engineering within a computer network environment. The ability to communicate within this environment is essential.

The *FLYING OBJECTS* development project has been started intending to establish an object oriented framework of design and evaluation relevant processes operating on widely adaptable aircraft components[1]. Overall development goal is to provide for a consistent data model of the aircraft geometry and the associated attributes like weights and aerodynamic characteristics and to easily supply this information to a toolbox of evaluation and design sub-processes, see Figure 1. This program architecture is the basis for a family of program applications which share one single data model.

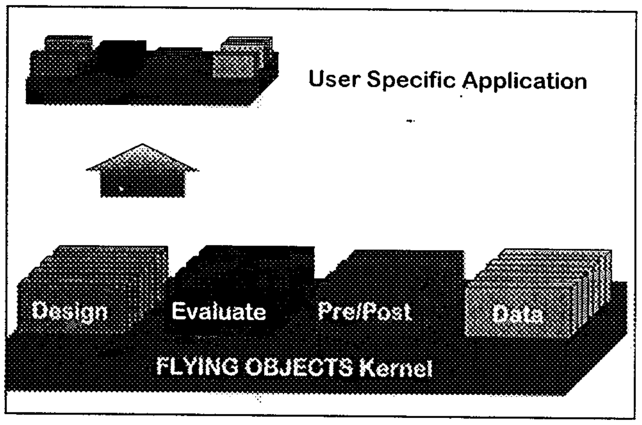

**Figure 1:** Toolbox Architecture

Besides the improved flexibility from the computational point of view, the underlying architecture with its open data & programming interfaces as well as the flexible parametric geometric model are a clue feature for the design and the calculation of a wide variety of aircraft configurations, see Figure 2. While not inherently providing every specific analysis

21st ICAS Congress
13-18 September 1998
Melbourne, Australia

method which might be required for the calculation of unconventional aircraft configurations, the system is able to grow with the data and the methods provided by the user.

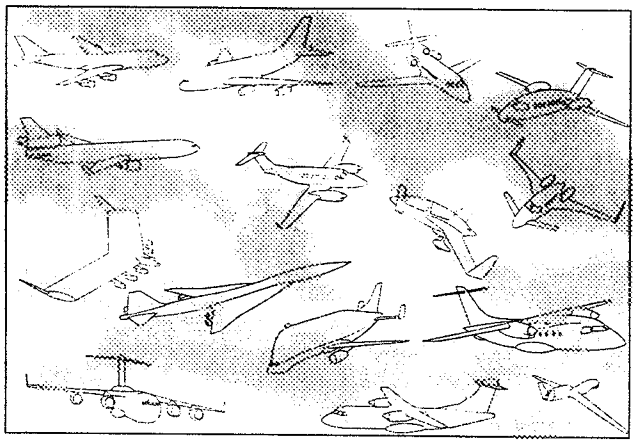

**Figure 2:** Configuration Flexibility

Following proven concepts of object oriented software design, the aircraft components are described in a highly generic way, providing for flexible data structures and interfaces. Attributes may be assigned to dynamically created. The ability to freely define aircraft components along with their attributes and the access to a powerful database allow for configuration flexibility and quick initialisation of standard aircraft configurations.

In order to calculate the desired component attributes (weights, aerodynamic parameters, etc.) the component and attribute dependencies or breakdowns can dynamically be defined. This concept allows the design disciplines to share a common data structure and to handle differing analysis levels. In case of weight estimation methods, a wing outer shell could provide for sufficient data for simple statistical methods while more sophisticated methods would require the definition of structural details. Dynamic adaptation of the geometry and the associated assembly structure gives the required flexibility.

Elementary calculation or design procedures are embedded in adaptable program flows, which give quick access to singular functionality but also to complex processes like aircraft synthesis, parametric studies or optimisation procedures. The combination of specific functions to complex processes and the ability to also access each of them alone - provided the required information is available - gives maximum user support and allows for the integration into existing work procedures.

A basic requirement for high analysis levels, the modelling of unconventional configurations and for configuration assessment (cabin, kinematics) is a detailed description of the outer and inner geometry. An object oriented modeller (ACIS™) provides for solid modelling and analysis capabilities.

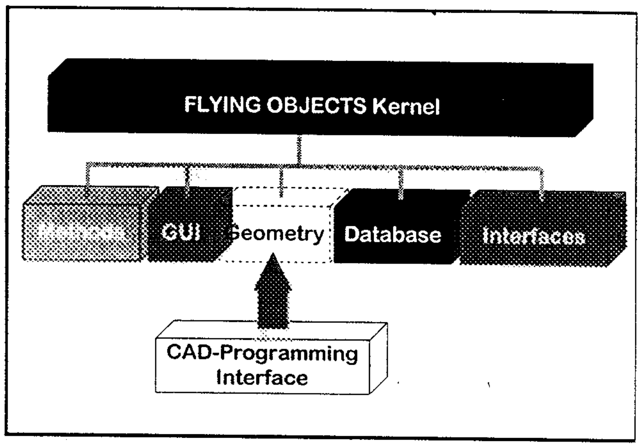

**Figure 3:** Geometry Interface

The abstraction of basic geometric properties gives the ability to model the desired configurations with manageable effort. Although shipped with a powerful 3D modeller, the software architecture allows the integration of *Flying Objects* into existing CAD environments, provided the CAD system offers appropriate programming interfaces, see Figure 3. Alternatively, the built-in modeller offers several data export facilities.

## 2. System Design Drivers

The design of the *Flying Objects* system architecture is driven by several needs. These include the access to a database, the offering of import facilities for legacy data and the providing of a powerful, but easy to use graphical interface (GUI). The GUI shall provide the mechanisms one is used to from PC programs, e.g. add or remove functions applied to project components.

Other design drivers include the extensibility with respect to detailing of functional components or completely new functionality. *Flying Objects* is intended to become an entire suite of applications, covering areas like conceptual aircraft design, detailed geometric modelling of the airframe structure for comprehensive analysis with respect to space allocation, aerodynamics, weights and structure as well as operational and economic evaluation.

The aim of the *Flying Objects* development is to provide a consistent program architecture which can be tailored to the customers specific needs by simply adding or removing program modules. For example, a mission performance module will be able to communicate with all other components which are envisaged in the scope of the program suite. For this reason it necessary to provide for uniform data input and viewing facilities. The design drivers sum up as in the following:

- ✈ Only provide the information the user needs.
- ✈ Give quick access to data visualisation and manipulation.
- ✈ Offer powerful, but uniform visualisation techniques.
- ✈ Utilise modern data input techniques (drag and drop).
- ✈ Offer data templates for storage of often used data structures.
- ✈ Store program session information for easy continuation of the work.
- ✈ Give instant access to project model or calculation processes as well as to the database.
- ✈ Give transparent access to the calculation methods, e.g. in case of a performance calculation routine which uses either drag and thrust data or alternatively performance data given by the aircraft manufacturer

These demands lead to an architecture, which aims to integrate proven engineering algorithms and strategies for aircraft modelling with modern software technology, giving a new software basis for engineering applications and ensuring the further development and the extensibility of the program system.

## 3. General Architecture

The design drivers lead to an architecture which is based on the tree representation of a project. A *Flying Objects* project is built up as a hierarchy of tree nodes, which represent the functional units of the program version. This architecture allows for management of parent / child relationships and offers the basis for intelligent data navigation, see chapter 4. The tree gives instant access to the entire project functionality and is capable of integrating the database handling, too. This allows the user to freely define and edit project components and simultaneously browse and reference database items. Thus, different members of the *Flying Objects* program family only differ in the number and the detailing of the project tree branches as well as in the subjects they address. The two base elements - project tree and the database - are explained in the following.

### 3.1 Project Definition

The definition of a *Flying Objects* project is schematically explained on the example of the flight mission calculation module. The content of a mission project is briefly illustrated in figure 4.

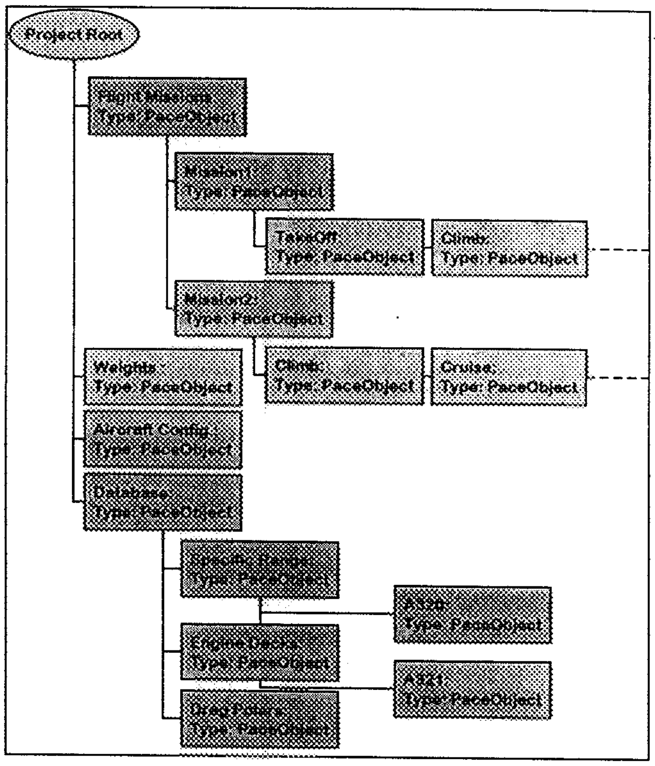

**Figure 4:** Project Components

The illustration displays the objects which will belong to a project file. Each object has to be regarded as a shell for data and program functionality. Every flight mission project will consist of four main objects which reside directly under the project root. These will be the node for the missions, the node for the supply of aircraft geometric and configuration data, the node for the weight breakdown of the aircraft under investigation and the node for access to the database.

The project will refer to database elements (e.g. engine card decks, drag data, specific range data) through the settings in the mission nodes. There will be two possibilities to link to the database. One will be to integrate a reference into the project, pointing to the database. This policy will allow for lean project files and quick initial program loading.

The other way of referring to the database will be to save the data with the project into one file. Although slower in loading, this feature would permit to exchange an entire project just by distributing the project file without the need to share the database.

The missions node will allow to specify several missions within one project. This node will be the starting point for functions which apply to all defined missions. The missions node will allow to initiate the calculation of several missions at a time, it will provide diagrams for the comparative plotting of mission specific data or the generation of comparative tables.

Children of this node will be the missions themselves. The mission nodes will provide the functionality for calculation of a single mission and generation of the respective diagrams and tables. Clicking on the mission segments will give access to segment specific information and processes, as there are manipulation of the segment parameters or calculation of single segment performance. The number and order of the mission segments will be highly modifiable; the mission will be sub-dividable into an arbitrary number of segments:

The order (or more general, the hierarchy) will only be restricted by 'real-world' constraints, so that the beginning of a mission should start with either a take-off or a climb segment, but not with cruise or descent segments. The aircraft geometry and configuration node will allow the user to input geometric and configuration data, which are important for mission and drag calculation. The parameters comprise the reference wing area, the lever of the outer engine for engine out drag calculation etc. The data will not be derived from a geometry but will only be input by the user. The implementation will be performed by attaching an extendable 'Info Class', which will be responsible for providing aircraft data. This implementation strategy allows for the integration of a geometry module for a more comprehensive program version, which would derive geometric parameters from a 3D geometry based on the ACIS™ modeller, but nevertheless offer those parameters through a compatible 'Info Class'.

Alike the aircraft node, the weight breakdown will be implemented as editable values in case of a mission performance software; however, when relying on a detailed aircraft geometry, the weight breakdown will initialise its nodes through triggering weight estimation methods which operate on the 3D aircraft geometry.

Although not really a part of it, the database node will be a compulsory element of each project. This node is the gateway to data import facilities as well as for the database browsing functionality. The reason to implement this node within the project structure is to provide the same visualisation techniques as for the project specific nodes. Both project and database will share the same user interface, identical editing capabilities and the same diagram or spreadsheet interfaces. Detailed information on the structure of the *Flying Objects* database is given in the following chapter.

## 3.2 Database

In its first version, the *Flying Objects* database is organised as a directory structure with sub-directories for specific contents and ASCII files for the data storage within these sub-directories. It will be presented transparently to the user as one single database containing the entire set of data, that is system wide as well as user owned data. Future program versions will substitute this database by commercial, relational or object-oriented database (e.g. Oracle™, Poet™, ObjectStore™).

The database files will be marked as writeable or non-writeable, depending on the access rights which will be controlled by the operating system. This feature will allow to simultaneously use (read) the system and user database for calculation purposes, but give manipulation authority (write) only to specific users.

The *Flying Objects* program suite defines its own internal database file formats (e.g. engine performance, aircraft performance and drag data) The file format will not only include the data, but will provide a consistent header which describes the range of application of the files. In case of engine performance data, the header will give quick information about the covered altitudes, mach numbers and thrust settings, providing for automatic checking of the applicability to an entire mission or a single mission segment.

For flexible integration of new data file formats and the necessary filters, a base import class will be defined. Integration of new file formats will thus consist in deriving a specific class for the new format, maintaining the same interfaces for data visualisation and storage to a database file.

# 4. Project & Data Management

A key feature of the *Flying Objects* software family is the internal data management. Overall design driver for the architecture was the requirement to handle very heterogeneous data structures but nevertheless provide uniform access interfaces and data navigation functions. Regarding the functionality which is addressed by the *Flying Objects* software suite, the program system with all modules will have to handle data such as 3D geometric, CAD-suitable models or numeric data like weight estimation methods, aerodynamic flow solvers and of course, performance calculation routines.

The program modules need to communicate changes of their parameters to one another; e.g. the geometry module will provide information for aerodynamic and structural weight methods, or simply supply its information to a two- or three-dimensional viewer. The data structure must be capable to dynamically grow and keep up a satisfying performance. All theses requirements lead to a design, which can be named 'object and attribute oriented, hierarchical data management'.

## 4.1 Object and Attribute Oriented Data Management

According to the above demands, design constraints were defined as in the following:

- *Flexible:* Arbitrary data structures smoothly fit into one project structure
- *Powerful:* Data structures have easy and standardised access to all attributes
- *Consistent:* Relevant changes of the data structures are automatically communicated to the objects
- *Transparent:* Data structures can access arbitrary viewers for optimal presentation to the user and comfortable manipulation of data
- *Fast:* Data management allows for high performance access to necessary information

This lead to an architecture, where standardised objects act as shells around the data structures which shall be managed. These objects provide for functions which handle parent-child relations and the navigation within the hierarchy. The objects have a standardised interface for persistence (storage to a file or a stream), which gives the basis for a flexible project loading and storage mechanism. In order to implement specific functionality, e.g. the performance calculation routines, import filter functions or visualisation routines, the objects can be extended with 'object attributes'.

These attributes may be simple values (strings, real or integer numbers) but also complex object oriented data structures (classes). The attributes are responsible for the functionality which is normally understood as a program. These object attributes are stored as templates and can be assigned to more than one object.

Functionality which has a more global character is integrated as so called 'base attributes'. These base attributes act as managers for object attributes and maintain a list of objects which have been assigned the respective attributes. This feature gives two main benefits:

Data navigation cannot only be performed by going up and down the object tree hierarchy, but also by referencing to a base attribute and asking for objects, which attributes are managed by the base attributes. This allows for direct and quick data search in a dynamic project structure. In case of weight chapter breakdown, objects which are assigned a weight can hence be collected without the need to browse the entire, potentially complex project structure.

Standardised, program wide functionality can easily be accessed by any project component which is added. This applies e.g. to the data viewers, which are not specifically designed for one single object. The diagram viewer is thus capable of displaying diagrams from database files (performance brochures, engine card decks) but also diagrams which are generated from the project tree components, such as the mission node or the node responsible for the entire set of defined missions.

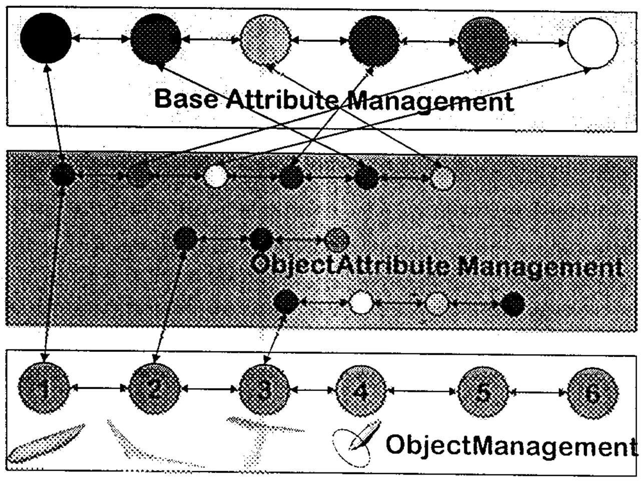

**Figure 6:** Object-attribute relations

Typical examples for base attributes are the data viewers, which comprise the project tree itself, the 2D and 3D graphics viewer, the spreadsheet viewer and the diagram viewer. The object-attribute relations are symbolically depicted in Figure 6. The objects in this figure are all real aircraft components; however, they can also be non-geometric components like mission segments within a mission calculation.

The objects are connected through two-way arrows, symbolising double-linked lists, which enable the navigation within the project tree. The circles representing the object attributes are not only assigned to a single object. A circle which appears in every object attribute list might for example stand for the description of the objects name, hierarchy position and the associated icon for the project tree. The displaying of the objects with this attribute within the project tree is then managed by the correspondent circle which appears in the base attribute list.

The links between the objects, their attributes and the associated base attributes give high flexibility with respect to the addition of project components and their implicit management. On the other hand, manipulation of the project tree implies complex updates of the link relations in order to maintain the consistency within the project. This management is performed automatically in the background.

## 4.2 Project Loading and Storage Mechanism

An important issue within a program system is the loading and storage of project files / structures. If a high flexibility with respect to the definition of an arbitrary number of project components is demanded, the persistence mechanism has to cope with varying project structures. Moreover, if requirements such as the restoring of project session settings or the management of generated diagrams have to be met, the loading / storage procedure has to be highly standardised.

By defining a virtual storage function in the object attribute abstract class, the persistence of a project is realised by browsing the project tree (base attributes) and subsequently calling the standardised 'save' function which triggers the object attributes to write their incorporated data to a single data stream and thus to a file.

How does a standardised 'save' function know about the details of an object attribute? This is explained by a key feature of the C++ programming language. The object attributes are 'derived' from a common base class. This means, that by calling the uniform function as defined in the abstract base class, the function which is really executed is the one defined in the object attribute which knows about the data it needs to store. The loading of a project is simply the inversion of this process. The project file contains identifiers for the objects which have been stored; their 'constructors', that means their initialisation functions, are subsequently called in the order of appearance in the file.

This mechanism is highly flexible with respect to the contents of project. It will thus be possible to store the reference to a database entry. When reloading the project, the reference will be resolved and the data be read out of the file. Alternatively, the data file itself could be send to the file stream. The identifiers in the project file would then tell the 'constructors' of the object attributes to interpret the following lines as the data which was originally stored in the database. A clear benefit will be, that the user can decide whether to vote for a lean and thus quickly accessible project file with references to the database, or to decide for the advantage of having an 'all in one' storage of the project, which would make the data transfer to remote computers very comfortable.

It is obvious that by moving the authority of data storage from a central unit to the components, the extension of the system with new functionality and / or additional project components (new flight missions, for example) imposes no changes to the project management. The central persistence unit only handles the scheduling of the save-actions according to the structure of the project tree.

## 5. Graphical User Interface

The graphical user interface (GUI) is characterised by a clear and standardised structure. It consists of a few main components, which give access to the entire program functionality. A project tree displays all project components, which are added to the project via component template dialogues. Standardised data and component viewers provide for 2D and 3D graphic visualisation, spreadsheet editing capabilities for data sets and textual output presentation as well as diagram generation facilities. Context dependent 'right-mouse button' menus present the specific functionality for each component, 'drag and drop' allows for quick and comfortable project initialisation.

### 5.1 Project Tree

No matter what subject a project component will be related to, it has its representation in the project tree. Unlike the data and component viewers, see chapter 5.3, which are only available for suitable project components, the project tree is the central visualisation and access gateway for the entire program functionality.

The project tree visualises the hierarchy between the components and sub-divides the project into its functional units. Additionally to standard, but 'old-fashioned' user-interface elements like the menu bar, the project tree provides context-related menus by clicking on the right mouse button. These menus give access to the functions which apply to the specific tree node. Double clicking on a project node allows for expansion or folding of the underlying node structures. With growing project size, this feature ensures a comfortable project overview.

The project tree also supports 'drag and drop' technique. For example, entire flight mission templates or mission segments are added from component template dialogues via drag and drop. Also, basic aircraft configurations like conventional aircraft or canard layouts can be initialised by dragging of appropriate template icons. The initialisation is then performed on the basis of defaulted values or database entries; however, the parameters can be edited at any time.

## 5.2 Component Templates

The FLYING OBJECTS software family will store components (symbolised as nodes in the project tree) and even complex component hierarchies as templates. Those templates will allow for quick initialisation or adaptation of the project. The templates will be presented in 'template gallery dialogues' as depicted in Figure 7, which illustrates cabin item templates.

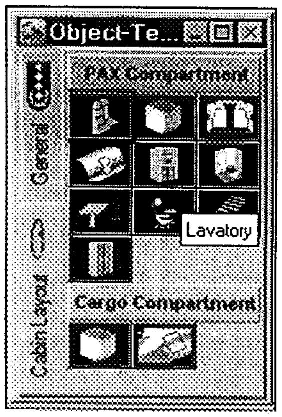

**Figure 7:** Template gallery

In analogy to the depicted cabin items which are organised within a folder tab, mission segments of the flight performance module will themselves form a folder tab with entries for generic mission segments (take-off, climb, cruise, step-climbing cruise, etc.). The templates will be presented as coloured icons and will provide a bubble help on mouse focus, describing the purpose of the template through a help text.

Application of a template to the project will be performed by dragging the template to the project tree or a suitable viewer (2D or 3D) and dropping it on the targeted position. Validity checks will ensure a consistent definition; in case of a mission segment, e.g., the placement of a take-off segment directly behind a cruise segment would trigger a warning or error message. In other words, the template knows about valid drop sites and / or drop constraints which have to be fulfilled. The parameters of a template will be defaulted by the template or initialised through a database reference, relieving the user of tedious initialisation.

## 5.3 Data & Component Viewers

Additionally to the project tree, which is used by all project components, *FLYING OBJECTS* provides for standardised data viewers. The viewers comprise a 2D sketcher, a 3D OpenGL viewer (see Figure 8), a spreadsheet and diagram viewer.

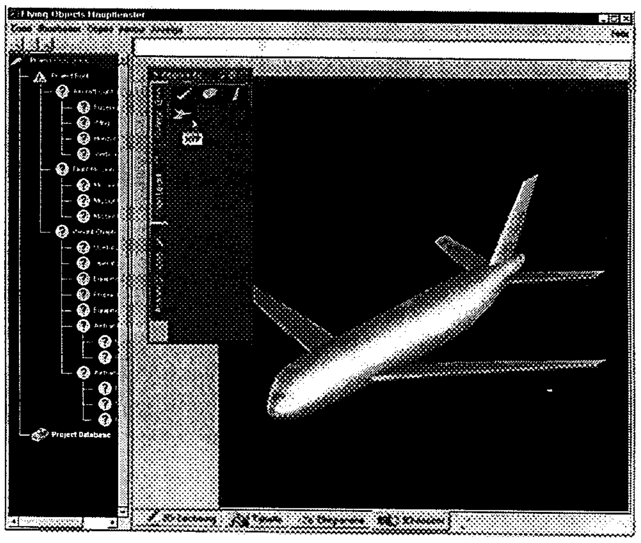

**Figure 8: 3D Viewer**

The viewers are organised within a folder, residing right hand side of the project tree. The window division between project tree and viewer folder is customisable through a vertical slider. The viewers are attached as 'attributes' to the project nodes, see chapter 4.1. As a consequence of this attribute handling, the viewer folder only displays those viewers which are associated with the active (marked) project node. Again, as in the case of context dependent 'right-mouse' button clicks, only the relevant information is presented to the user.

The 2D sketch viewer is used within *FLYING OBJECTS* for any kind of 2D non-diagram presentation which requires interactive modification and needs drag and drop capabilities. In case of the performance software, this viewer will be used for the presentation of the mission profile, either for one single mission when clicking on a mission node within the project tree or for the entire set of missions within a project when pointing on the flight missions node.

The viewer not only provides a graphical sketch of the mission which is defined in the project tree, but gives interaction capabilities for mission segment modification and visualisation of the mission status. For example, a red sketch of a mission segment might indicate that this segment is not fully defined with respect to the required input and is thus not valid for calculation.

The viewer is a 'drop-site' for the template gallery, just like the project tree. Thus, the user is able to access the mission segment functionality either by right-mouse button click on the node in the tree or on the respective segment line in the 2D graphics.

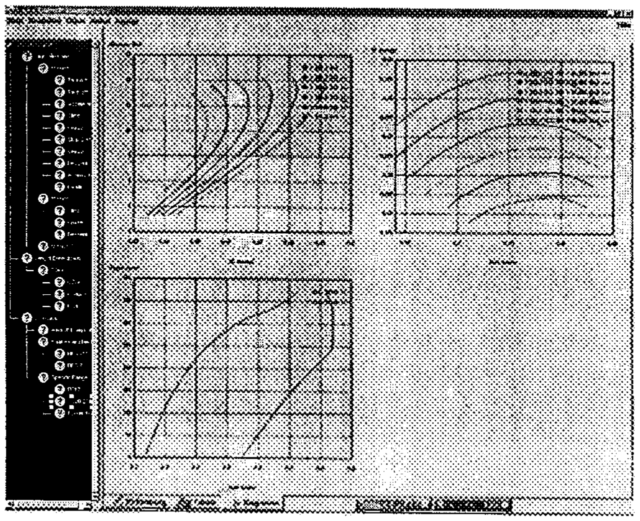

**Figure 9:** Diagram Viewer

The diagram viewer in Figure 9 displays project node specific diagrams, e.g. SAR-curves when clicking on the mission or any kind of data when pointing on a database entry.

The graphics output of the diagram library are 'piped' into the diagram viewer; this allows for displaying several diagrams into one worksheet. The diagrams are automatically generated with a pre-set formatting; however, the user has the possibility to entirely customise the look of the diagrams. This includes the formatting of line types, colours and thickness as well as the insertion of arbitrary text, the generation / deletion of legends, insertion of bitmap logos (Enterprise logo) etc.

The diagrams can be exported via output filters to the following file formats:

- Postscript (a vector file format)
- Windows Metafile Format (a vector / bitmap file format)
- DXF (a vector file format)
- Portable Bitmap Format (a bitmap file format)
- All X-Window bitmap formats

Depending on which project node currently has the focus, the diagrams might display thrust curves for a single engine or compare the curves of several, selected engines, if the engine-database node would be the active one. By using the same diagram viewer, the generation of diagrams from a data stream can be generalised and thus be applied to arbitrary components. The extension of the functionality for future program versions will thus cause relatively little effort.

The spreadsheet viewer serves both for data and result visualisation as well as for editing / manipulation purposes. This viewer is available for imported data files such as drag data, thrust data or aircraft performance brochures as given by the aircraft manufacturers. The viewer performs certain input checks, for valid digit number input for example. The user has the possibility to add / remove data sets, insert or remove rows and columns, mark regions, copy or remove regions, etc. Resulting changes in the data sets can be checked with the diagram. For output presentation, the spreadsheet viewer provides for pre-built tables and formatting.

## 6. Program Family

The software architecture as described in the previous chapters can be regarded as the core of an engineering workbench, which is capable of providing for the necessary components for several applications or program versions around aircraft assessment and design. Core elements of the system are the object & attribute oriented data management system, the project tree and the viewers for 2D, 3D, spreadsheet and diagram visualisation.

In the first development phase of the *FLYINGOBJECTS* project, three main application domains have been identified. For aircraft mission performance, a program version with a highly flexible mission calculation module has been developed. *FLYINGOBJECTS- MISSION* gathers the required input data either from drag data and engine cad decks or from aircraft manufacturers performance brochures.

Import facilities for several manufacturers formats are provided.

The second project derivative addresses the automated layout of cabin interior items and the generation of the necessary drawings and tables. The cabin module is characterised by its easy handling and a high performance. Besides two dimensional cabin layout plots, three dimensional shaded models of cabin items and interior geometry are provided.

The last program version reflects the integrative character of the system. *FLYINGOBJECTS* Design provides for a 3D, component based aircraft model, analysis methods and interfaces for all relevant design disciplines and processes for conceptual design which can be applied to the defined aircraft geometry.

These first three applications show the potential of the presented architecture and the prove the concept which intends to provide for application specific views on a consistent data model. It is intended to extend the application of the *FLYINGOBJECTS* core system to other engineering domains.

## Bibliography

A. Schneegans, Ch. Haberland, M. Kokorniak, B. Domke. *An Object Oriented Approach to Conceptual Aircraft Design through Component-Wise Modelling.* ICAS-96-1.3.1, Sorento, 1996.