

# SEGIS - Ship Design and Simulation System

Sandra Schrödter, FSG, Flensburg/Germany, sandra@fsg-ship.de,  
Thomas Gosch, FSG, Flensburg/Germany, gosch@fsg-ship.de

## Abstract

Within the research project SESIS (Ship Design and Simulation System) a structured, modular and flexible software platform for early design of complex ships has been developed. Several simulation and design tools are linked by clear defined interfaces and ergonomic graphical user interfaces. The presentation will describe the IT technology used in the software framework for graphical user interfaces, distributed data management, etc.

## 1 Introduction

The early design phase of a ship is a multidisciplinary process. All technical aspects of the ship's performance, like fuel consumption, safety standards, deadweight, sea keeping behaviour, manoeuvrability, comfort, etc. are elaborated. During this process the yard's engineers define a technical specification for a specific ship design which forms the basis of the building contract between yard and owner. Thus, the main part of the technical and financial risk of owner and yard is depending on the results generated in the early design phase. Typically 85% of the total building costs of a ship are fixed during the first 4 weeks of the design process.

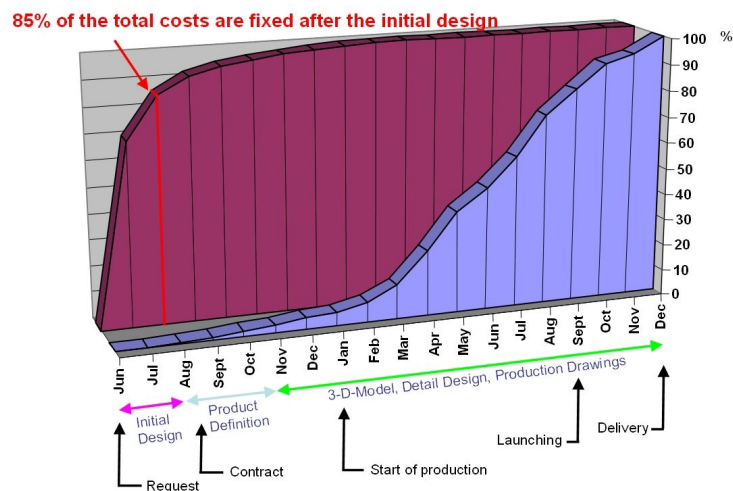


Fig.1: Design costs versus building costs

To enable the design engineers to generate precise results within the very limited time available in early design, IT support for their work is highly recommended. The range of tools used goes from "simple" engineering formulas for single technical problems to complex 3D models for FEM or CFD analyses. As the performance of computers and tools available is rising up constantly, more and more complex problems can be solved during the early design phase. Thus, engineers are enabled to design an optimized ship for the owner's needs today.

One of the key issues during the early design process is the information flow. Typically several design engineers of different disciplines are working in parallel on one specific ship design. Thus, they have to get and serve information all time to ensure their work is done on actual and consistent data whenever possible. An efficient data management system is essential for this kind of work.

To ensure efficient work in the early design process of a ship, an integrated software platform giving a standardised interface to different design programs, tools and their related data, is needed. The main requirements for such a platform are:

- Efficient data management, enabling concurrent engineering
- Flexible but safe data handling
- Possibility to combine programs into workflows
- Parallel use of programs running on different platforms (Windows, Linux, Unix)
- Robust system performance, enabling work with even inconsistent data

To create a software platform fulfilling the requirements mentioned above, the SESIS development was started spring 2005. SESIS is a collaborative project driven by DLR<sup>1</sup>, SCAI<sup>2</sup>, SAM<sup>3</sup>, TUHH<sup>4</sup>, CMT<sup>5</sup>, Lindenau Yard<sup>6</sup> and FSG<sup>7</sup>. The development of the SESIS project is funded by the German Ministry of Economics and Technology.

The SESIS system aims for one database to manage all data created by the programs and tools used in early design. An authentication and authorisation service will control the access to design data and programs available, even from outside the yard's environment. Further more the framework will allow easy integration of existing and new design programs from different platforms by use of wrapper technology. A standardised User Interface (UI or GUI, Graphical User Interface) will give the engineers a harmonised access to the available design programs. The general idea of the SESIS data management and programs interfacing is shown in the pictures below.

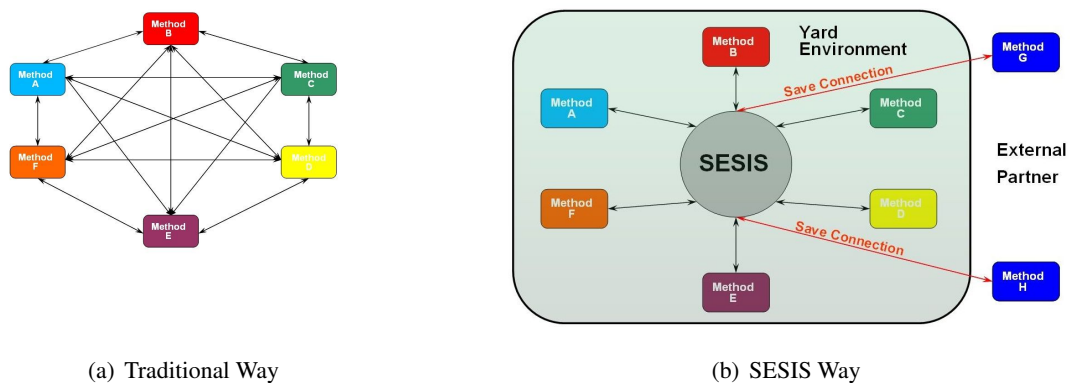


Fig.2: Interfaces and Data Exchange

While the traditional way depends on program methods to directly communicate with each other, the SESIS way promotes a centralised approach where data is accessed through a standardised interface. Here, the methods lose their dependency on other methods, granting more flexibility while additional allow to replace and maintain methods more easily. To achieve these advantages, an adequate IT infrastructure has to be designed.

<sup>1</sup>Deutsches Zentrum für Luft- und Raumfahrt – German Aerospace Center

<sup>2</sup>Fraunhofer-Institut für Algorithmen und Wissenschaftliches Rechnen – Fraunhofer-Institute for Algorithms and Scientific Computing

<sup>3</sup>SAM Electronics GmbH

<sup>4</sup>Technische Universität Hamburg-Harburg

<sup>5</sup>Center of Maritime Technologies e.V.

<sup>6</sup>Lindenau GmbH, Schiffswerft & Maschinenfabrik

<sup>7</sup>Flensburger Schiffbau-Gesellschaft mbH & Co. KG

## 2 Requirements from an IT Point of View

The requirements for the IT infrastructure and the framework concretise into the following:

### Requirements for the Basement:

- Flexible and extendable
- Based on standard, state-of-the-art software components
- Easy maintenance
- Data management
- User management
- Communication services
- Security levels for data and integrated programs
- Platform independent or at least functional on different platforms

### Requirements for Integration:

- Integration of existing software and applications
- Different levels of integration possible, depending on the availability of sourcecode or an API<sup>8</sup>
- Common user interface
- Centralized data handling
- Defining and carrying out of workflows

### Requirements for Development:

- Software development kit (SDK) to support the development of new software for the framework
- Providing tested components
- Aid in keeping standards

Where to look for the solution? The market for collaboration software is growing. The world is shrinking, communication is *the* topic of today, not only connecting humans but hardware, programs and datamodels. New software design paradigms are evolving, facing the problems of designing complex system in a manner that keeps them scalable and maintainable.

Only there is no such tool that combines the points above yet, nor is it likely to evolve at short notice, because the above mentioned requirements are fairly extensive. Considering the task of integrating existing inhouse software solutions, it is often tried either by throwing away and rewriting the sourcecode basis using new paradigms (high potential, but risky and time consuming) or patching it with software-glyue little by little (downwards compatible but reduced maintainability and potential).

Both ways are examples of extremes that, as experience tells, often fail to reach their goal. But there is a third way: keep the well proven, gain the new and combine both into a framework.

Although developing software inhouse commits resources and has the peril of producing isolated applications or reinventing the wheel, the list of benefits is long.

Once the know how is built up – which may take some time – it is possible to specialise on an area of expertise and react faster to new or changing requirements than would ever be possible with software off the shelf and waiting for the next version. *If* the requirements make it into the new version. If there *is* a new version.

So being independent may allow you to be the first – or only – one able to comply to a customer's request.

---

<sup>8</sup>Application Programming Interface, a source code interface provided by e.g. a software library.

### 3 Creating SESIS

The collaborated research project SESIS was started to create a framework as described above. The partners bring in the required experiences in IT development, engineering processes and how to connect both sides.

The general conditions for the project were manifold and can be split into three sections:

#### 3.1 Keeping the Well Proven

Regarded applications range from the inhouse written software where the sourcecode is available and adaptable, called white box, to the black box software without sourcecode, which is only changeable by the vendor. Between them exists a span of grey box software, which can be, to an degree, customized and controlled by input files or an programmable API.

Both the *know how* in grown applications, which is more or less palpable, and the *efficiency* of established processes, deliver reasons the keep them.

But the fact that software is in use for a longer time doesn't make it well proven per definition. There are numerous reasons for remaining using it, like running contracts, licences or sometimes the fear of change which always contains a risk. These reasons have nothing to do with reliability.

Examples of well proven programs include the optimized and thoroughly tested solver or the efficient and for the internal processes highly customized prognose tool.

#### 3.2 Gaining the New

A 20 year old car is still running, not least because its character is known so thoroughly that almost every issue can be fixed. It keeps its driver warm and dry and gets him from A to B. There seems to be no reason to change this.

But moderns cars are more efficient, safe and faster. They contain a lot of fancy stuff – well that is a matter of taste, but the first points will show in the car owners purse and the safety of him and his passengers.

The statement from this is easy: being part of technological progress has advantages.

What needs to be carefully considered – in particular in the rapid changing IT – is not to fall for a “hype”, meaning a paradigm or software that shines brightly the one day, with its name all over the internet, and disappears silently within a year or two.

Examples of technological progress include visualisation toolkits, communication support or data storages. Due to advancements in computer hard- and software the modern realizations are much more powerful and extensive and allow both software developers and engineers to concentrate on their actual challenge.

#### 3.3 Combining both in a Framework

Neither all new nor all established is the solution, but a combination of both it is. The main challenges to be met on this road are:

- Find a common platform where all applications can be run and managed from. Common possibilities for this include Linux or Windows
- Present the user with as much of a common user interface as possible. This is always possible for new software, once a standard is kept, but somewhat harder for existing ones
- Share data bases of the integrated applications, which may be solved via direct or indirect exchange of data

Common to all points are their possible consequences for existing software, which may include porting and refactoring of white box code and buying new licences for black box software. So another challenge can be formulated:

- Keep changes to existing software as minor as possible

#### 4 Realisation of SESIS

With the right decisions regarding fundamental technologies a number of achievements come for free. Open source software offers a heap of proven applications or functionalities and the support is, thanks to forums and communities often faster and more detailed than with commercial hotlines. On the other hand some precaution should be put into the selection of technologies because this rapidly changing market carries the above mentioned risk of falling for “hypes”.

More details will follow but some words about limiting implementation parameters first: The main programming language was chosen to be Java. Java is modern, object oriented and platform independent, thus supporting among other things maintainability and integration of applications that may be limited to a platform. Apprehensions about poor performance are no matter anymore.

The base for flexibility and maintainability is delivered by the underlying Rich Client Platform (RCP) – a subset of Eclipse<sup>9</sup>. One part of RCP’s flexibility is based upon the OSGi standard that supports *pluggable* functionality<sup>10</sup>

Considering these technologies as base pillars, still more functionality needs to be added for the above mentioned requirements, as there are remote usage, security, data mangement, workflow and the integration of different softwares on the list. Part of them can be found as stable and standarized open source software, others need to be implemented.

Bundling all together defines the multi purpose framework RCE (Reconfigurable Computing Environment), which can be utilized in diverse industries. SESIS extends RCE by adding applications for the shipbuilding industry.

##### 4.1 Layered Architecture

An overview about the architecture of the new framework (see Figure 3) shows that it is made of different layers to separate components, which is an important constraint for complex software systems.

The basement is made up of 3 Components:

- As Java is the common programming language, the application runs in a *JVM (Java Virtual Machine)*.
- *Eclipse Equinox*, which is part of the above mentioned RCP, is an implementation of the OSGi core framework specification. OSGi specificates an application life cycle management model which enables a modular and customizable application with *pluggable* functionality and defines a scope for modularity: the *bundle*.
- The *Reconfigurable Computing Environment* is based on the open tools platform RCP and extends it in two ways: a core layer (yellow) and a number of additional bundles (blue).

The vertical blue bundles provide additional functionality like: data mangement, communication, security and distribution.

---

<sup>9</sup>Eclipse itself started as an Integrated Development Environment for Java years ago and advanced to a popular open-source software framework

<sup>10</sup>The OSGi standard was formulated by the OSGi Alliance (formerly known as the Open Service Gateway initiative) which is an open standards organization founded in 1999 with companies among their members like IBM, Oracle, Sun Microsystems, or Siemens.

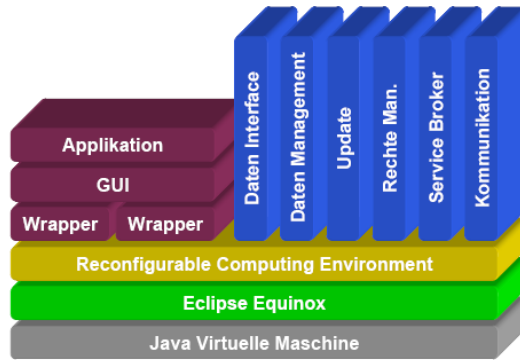


Fig.3: Layered Architecture

The red block only symbolizes the manifold possibilities of different applications and integration levels. These applications may contain user interfaces or consist only of a calculation kernel. Most will need a *wrapper* to be integratable, but the level of integration may differ considerably.

#### 4.2 User Interface

The user interface of SESIS as a framework is based on the UI toolkit SWT/JFace which became popular with Eclipse. While the efficient but rather low-level SWT (Standard Widget Toolkit) provides standard GUI elements for Java applications (e.g widgets like textfields, comboboxes, menus or buttons), JFace extends it and provides complexer components like a menu bar, tool bar, status line, different views that can be independently opened, closed or rearranged and perspectives, combining views (see Figure 4), all following the Model-View-Controller design pattern.

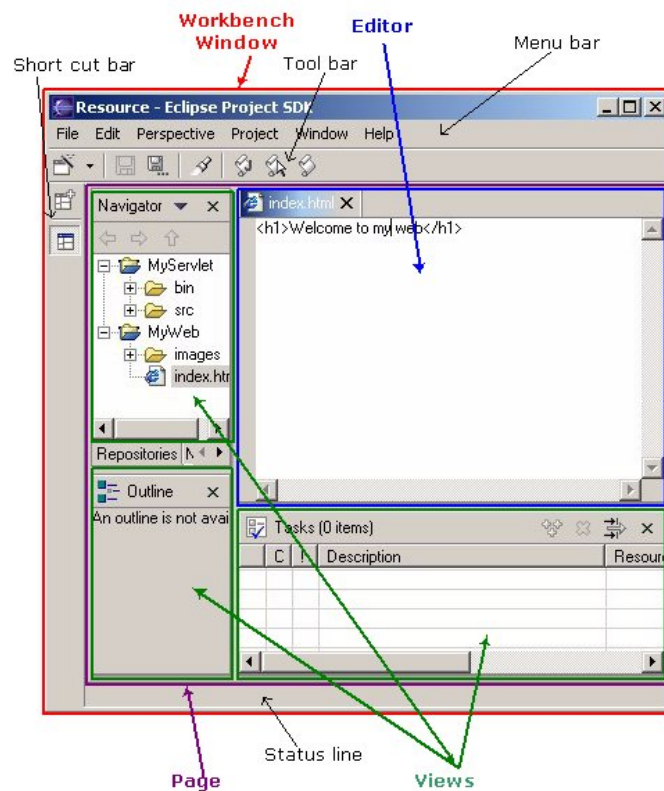


Fig.4: GUI Elements of eclipse

Because SWT made considerable effort to present a platform dependent Look&Feel to the user it is not platform independent. But that doesn't reduce the platform independence of SESIS much, because implementations of SWT exist for numerous platforms including Windows, Linux, Unix, MacOS and even Pocket PC.

This user interface will be automatically used for new developed applications and for managing the framework itself, but what about existing applications? It is quite possible to adopt their own UI but the effort depends on the availability of sourcecode and the language used. This has influence on the decision of the integration level.

### 4.3 Integration

What can be integrated into the new platform? To start with the easiest: bundles that were developed for Eclipse. Functionality of existing plugins (Eclipse's name for an OSGi bundle), range from low level system tools to highly sophisticated applications. Those bundles can be added to the framework and immediately be used from within the system.

For software that is not already provided as a bundle it is a bit more complicated. This software needs to be *wrapped* by a bit of sourcecode to form a bundle that then again can be used from within the framework.

While the best you can realize with a wrapper for black box applications is starting, stopping, and providing needed data, white box and often even grey box code can be integrated nearly seamlessly, e.g. utilizing the above mentioned new UI toolkit and wrapping the UI-less functionality of the existing software, though it may be necessary to refactor it beforehand to separate UI and logic.

If the existing software is not written in Java, the integration demands a higher effort to bridge the language gap. But there are tools available. The main one is called JNI (Java Native Interface), which is shipped with Java. Another one, JNA (Java Native Access), a free library that provides easy access to native shared libraries, encapsulating JNI. Both tools enable communication from Java to C or e.g. to Fortran (via C) and vice versa, thus allowing direct access to functionality of the other side.

Once applications are integrated there are various ways of starting and managing them. One is the well-known way of an *Open with* on a file, which is provided from the platform. Another is a new implemented bundle that adds a browser for the integrated applications, from where they can be started or stopped. Also, the workflow management connects applications to be automatically executed, while the succession may comply to additional conditions.

### 4.4 Distributed Environment

SEGIS is designed for a distributed environment so it comes with communication support. This enables distant SESIS-Instances to communicate with another, transferring data or starting remote applications (see Figure 5).

The communication bundle of SESIS supports a number of protocols, including RMI (Remote Method Invocation), CORBA (Common Object Request Broker Architecture) and SOAP (former Simple Object Access Protocol). This enables the application to choose the most suitable communication for a task, balancing need for performance, security and existence of firewalls.

Further more SESIS aims at the world of Grid Computing planning to support the grid computing middlewares UNICORE (UNiform Interface to COmputing RESources) and Globus Toolkit.

### 4.5 Datahandling

One critical part of the collaborative environment is the data. Integrated applications lose a bigger part of their benefit if their data is isolated. Different steps can be taken to avoid this.

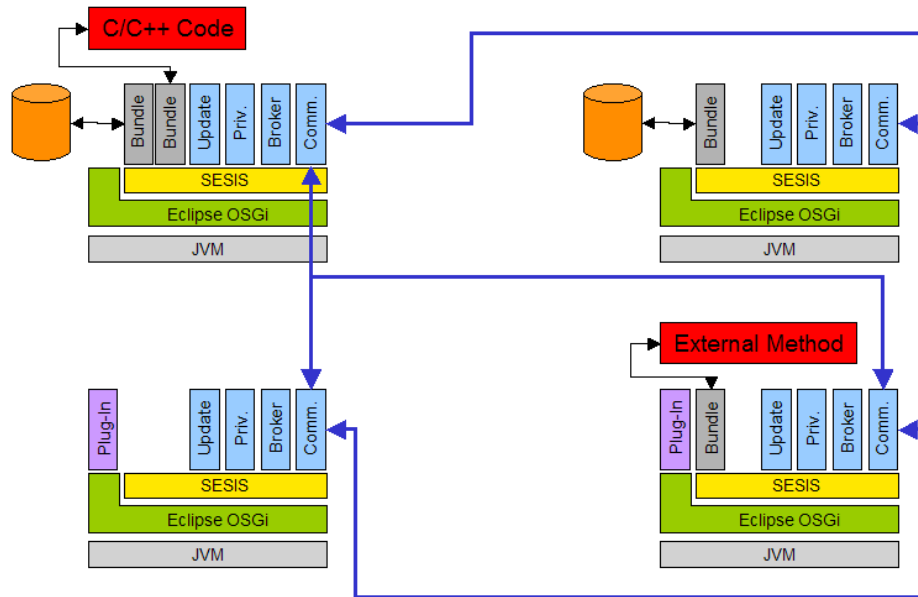


Fig.5: Communicating SESIS Instances

They could be directly connected to a common data base (e.g. Apache Derby, a relational database management system or just the filesystem). This is easiest for white and may be possible for grey box code. If the direct connection is not possible (which is the case most times) the data should still be saved in a common database. The next step would then be to provide conversion tools to adapt the data for the different applications.

The SESIS data management is located in the data management bundle and provides a common layer, hiding the various storage types.

One possible exception to the central management of data can be the use of workflows. It is possible and reasonable to pass data from one step to the other directly, to connect output data with input data.

Once all data is managed by a common data management, it is possible to expand it by further functionality. The first important improvement would be metadata, that is additional information kept together with the data, e.g. about which engineer last changed this data and why. SESIS implements them with its tailor made data management that handles metadata as Key-Value-Pairs for each file.

The second one would be versioning. This gives additional safety because it adds a *history* to data. SESIS enables versioning with branching, merging and locking.

## 4.6 Security

The higher the amount of users, the further their location from the company and the more valueable the business data, the safer a system needs to be.

SEGIS requires a high amount of security, because remote and grid operations open the system to the world. This needs to be taken care of, so a combination of security mechanisms is integrated into SESIS that is similar to what is used in Grid Computing.

First of all comes the authentication: the user needs to log on when SESIS starts. As a single-sign-on system no further authentication is necessary, each successive bundle can check against this user. Similarly the access to data is checked – if it is handled by the data management. Each instance of the system *can* have its own privilege configuration. That enables each company to regard safety from a local point of view.

## 5 Conclusion

Although the requirements for a flexible and extendable framework for a distributed collaborative environment seemed ambitious, the majority of the requirements are solvable or already solved. Because of careful weighted design decisions it was possible to gain a lot from the open source market in terms of sourcecode or standards and momentum from communities like Eclipse. This is also a potential for the future, for the community takes care of advancements and bug fixes.

Of course there is still work to be done. Especially regarding tests, integration tests in particular. Performance and scalability can best be tested during stress situations. Creating them is on the agenda for the next half a year.

Considering the general conditions from the beginning, *Keepin the Well Proven* was realized with the integration of white and black box code. One example presents the black box integration for ANSYS. A wrapper was implemented that fetches necessary data, asks the user for additional information it couldn't gather from that data base and starts a calculation with ANSYS. Another example shows the white box integration of E4 Methods utilizing JNI, which will be replaced by JNA in the next step, because it is easier to maintain.

*Gaining the New* brought a new harmonized GUI based on JFace/SWT, standards like OSGi or CORBA, new potential with the use of existing Eclipse plugins and the application of object oriented design paradigms and Design Patterns.

In order to *Combine both*, integrated applications can be managed from a single point with a new project and data management organizing their data. Thanks to the communication support, SESIS is able to connect different platforms, diverse applications and distributed users.

An example for an application developed for the new framework is the Propeller Design Tool. It allows interactive modifications and hydrodynamic assessment of the propulsor, offering a flexible and modern GUI and keeping its data downwards compatible.

The resulting conclusion of developing a customized software framework is positive. It is possible to keep the well proven in form of existing applications and functionalities so that no setback regarding required and established functionality is to be feared. On the other side the potential of new technologies could be gained, supporting both the engineers and the developers with new standards and technological achievements.

Basing on tested, standardized and maintainable components, SESIS supports the demanding and time-critical concurrent engineering of the early design phase of a ship.

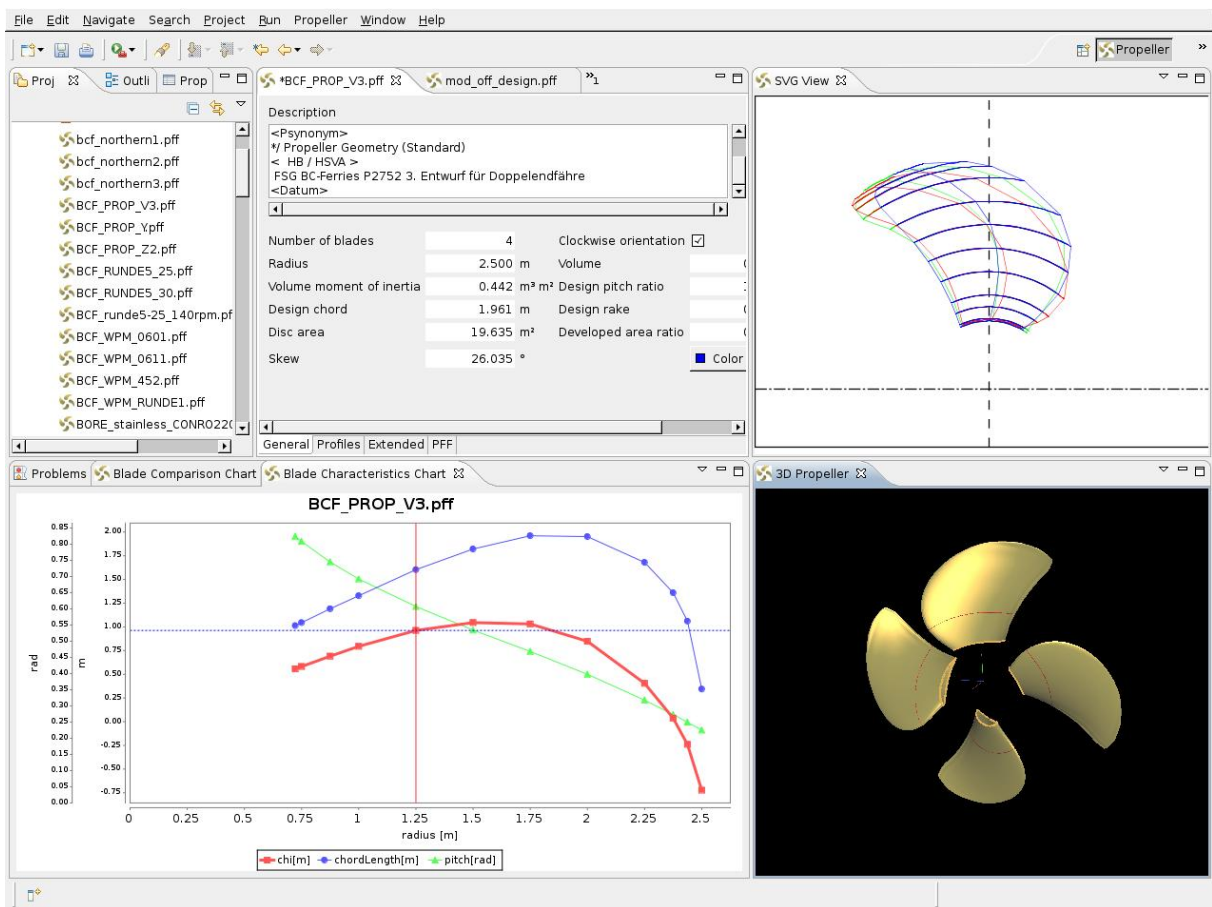


Fig.6: New Propeller Design Tool

## References

*SEISIS – Integriertes Schiffsentwurfs- und Simulationssystem*, [www.sesis.de](http://www.sesis.de)

KRÜGER, S. (2003), *The Role of IT in Shipbuilding*, COMPIT'03, pp.525-533

FREEMAN, E.; FREEMAN, E.; SIERRA, K. (2004), *Head First Design Patterns*, O'Reilly Media

*Java*, [java.sun.com](http://java.sun.com)

*Eclipse*, [www.eclipse.org](http://www.eclipse.org)

*OSGi*, [www.osgi.org](http://www.osgi.org)

*JNA - Java Native Access*, [jna.dev.java.net](http://jna.dev.java.net)

*Wikipedia*, [www.wikipedia.org](http://www.wikipedia.org)