Integration of Software Packages with Different Programming Paradigms, Exemplified by a Ship Model Generating Tool

Wilfried Abels, TU Hamburg-Harburg, Hamburg/Germany, w.abels@tu-harburg.de

Abstract

The paper describes problems and solutions for integrating old existing software in Fortran with new programming paradigms in the context of a specific tool to simulate a new ship design in a bridge simulator. A system kernel in pure Java application is combined with simulation tools written predominantly in Fortran.

1. Introduction

Many large software products in use today represent continuous development since 10 or 20 years, reflecting accumulated knowledge from many research projects. All this know-how is part of this software and should be used in future without a total new implementation. But information technology (IT) is a young science. The way to develop software has changed over the years. Modern techniques are very different from traditional or historic programming paradigms.

These differences create problems, when an integration of modern and classical software packages is necessary. Especially in engineering, powerful and large simulation tools need to be integrated in a design environment. The simulation tools may come from different environments and philosophies. Apart from that, we have today software at our disposal to handle problems of distributed databases, communication by distributed computation problems and other information technical tasks. However, the integration of large simulation tools from different sources in one system still poses challenges.

Our particular case here is the E4 ship design software, a complex software for different tasks in ship design, Krüger (1999). The system consists at present of a set of 116 different methods, and further simulation and user-interface modules may be added in the future. The flexible approach of E4 allows an engineer with little IT background to write his own application in a short time. The engineer can concentrate on the physical problem, instead of the IT part. Due to historical reasons, E4 is largely based on a procedural way and not event driven, as is practice today. For an engineer, this procedural approach is easy to understand, because the behaviour of a program can be analysed and structured directly and is not based on complex modern design pattern, Gamma (2004), as required by event-driven applications. It is not useful to demand that all engineers, who want to implement software, have to be familiar with complex IT concepts. Instead, it is more practical to use a framework with a clear restricted functionality, which allows implementing software with only a short training period. The task of a software-developing engineer is to build numerical models for a physical problem and not to waste time for computational problems. A naval architect who uses the E4 framework has his know-how in ship design and its physical behaviour. By the use of the above-described framework, this know-how can be used in an easy way.

Since programming E4 started, the technical environment has changed significantly. Computational power has increased and distributed work came with wide availability of computers. Aspects of user authentication and a management of privileges are now very important. The distributed use of data generates problems of informational consistency. Computational networks with high bandwidth worldwide allow now distributed computational power and data. For example, shipyards and suppliers want to have their computational environments to communicate directly. Time consuming calculations may be executed remotely at dedicated computer centres. The new demands require updating of E4. Generally, simulation software frequently does not consider complex IT handling tasks.
This was the motivation to launch the BMBF funded project SESIS, http://www.sesis.de. This is an IT project to solve such problems. SESIS combines naval architect with pure IT specialists, Fig.1. The project will design a system kernel for technical applications like the ship-design software E4, which will be called in the following RCE (Reconfigurable Computing Environment). The RCE should handle all the tasks of data handling, user-authentication, management of privileges and network communication. A further important aim is that existing numerical methods can be integrated to the RCE without a new implementation. Therefore, the interfaces between the new RCE and existing methods are vital.

In contrast to the existing numerical methods, normally written in FORTRAN and compiled to system depending executables, the RCE is a pure Java application, which needs a Java-Virtual-Machine for executing the byte-code of a Java program. Java has been chosen because its hardware independence makes it easier to handle different platforms and the communication between them. Moreover there are many free add-on packages available, which can be used without own implementation effort. But these two concepts are based on total different software paradigms. In a classical environment of compiled executables, different computer languages can interact in a relative simple way, because all compilable languages generate object files, which can be connected by a linker to one application. In contrast to this, an interpretative language generates no executables but it needs an interpreter for executing the application. Therefore, object files cannot be connected to the byte-code of a Java application, but such a connection is necessary for integration.

Fig.1: Partners of the SESIS project

E4 is continuously under development and in use by different partners. Taking E4 out of operation for several months to implement major updates is not an option. With several E4 users not participating in SESIS, the danger of developing incompatible versions must be avoided. Therefore the migration path from a stand-alone executable to an integrated version is absolutely important. Without a clear strategy for this task it is impossible to migrate a continuously used software system of similar complexity.

A further task by the migration of such software is to handle the complexity of the total system. Because there are many different technologies, which have to interact homogeneously, they all have to be supported and controlled by the developers. But nobody can be familiar with all these different IT techniques. There are experts for numerical problems of physical FORTRAN methods, experts for designing complex object-oriented software architectures, and experts for databases. Nevertheless a system has to be built, in which all these different people can work, although they can not overview the complete system in all details. This is only possible if there are clear boundaries with simple interfaces. Every section of the global system has to support and to regard the facilities of the different experts. The layout of the system has to reflect these aspects of psychologies.

Not only should existing software be integrated, but the framework should also adapt to future developments. Most of these future developments will come from naval architects and not by computer scientists. Therefore it is also important for new implementations to have a development
framework, which regards the facilities of technical engineers. This group of persons, who are not pure users of the software and who want to implement themselves software for their physical problems, will be called in the following power-user.

One of these sub-projects is to develop an interface from the new SESIS system to the bridge simulator Simflex, http://www.simflex.dk, from the company FORCE Technology. The aim is to have an automatic link between a ship-design tool and a bridge simulator. This link allows testing a ship design in a virtual environment before the ship has been built. The classical use of a bridge simulator is training the crew. The numerical models are adjusted mostly by results of model and full-scale tests. Such test results are not available during the design stage of a ship. Therefore numerical simulations have to identify the behaviour of the projected ship. In the past, many methods have been already implemented within the ship design software E4. There are methods for resistance, manoeuvring and seakeeping of the ship. These methods allows a ship designer to estimate the performance of a new design. However, the generated information is hard to understand for people not familiar with engineering. Therefore discussions between project engineers of a shipyard and customers are sometimes a little bit complicated, because a customer does not think in terms like lift coefficients of a rudder. The customer has special missions for the ship he wants to own. For example, he wants to transport a special charge between two ports. Such a mission should be realized with a maximum of benefit for the investor. Therefore, the customer is not interested in technical terms but for example in questions, how often he needs tug support within a port and which technical actions and costs are necessary to avoid this need. For a customer it would be much more useful to let his own captain navigate within a bridge simulator than to discuss about simulation results based on first-principle methods. For the communication between engineer and customer, a bridge simulator could thus be useful. Unfortunately, at present the preparation of a manoeuvring model for a bridge is time consuming, because the physical flow simulation uses completely different data representation from the bridge simulator. The task at hand is to interface the two applications within SESIS, allowing virtually automatic data transfer. This will offer two benefits. The application as such is useful and it demonstrates the feasibility of the SESIS approach.

2. Architecture of the RCE (Reconfigurable Computing Environment)

The new RCE is a platform for other software tools. The aim is that these tools can be connected to the RCE and that they can use its functionality for their own purpose, Fig.2. The level of integration is not constant for every method. It is possible that some tools are connected only over file exchange, whereas other methods are directly part of the RCE in form of a Java package. Between these
extremes all levels of integration are possible. Further there are three different roles of people who want to use the SESIS platform:

1. The Naval Architect as User only wants to use the SESIS platform for the design applications. He does not want to program. If he encounters a bug, he will wait until stable versions are available before he updates his platform.

2. The Power-User primarily wants to use the software. But he also develops new methods (respectively plug-ins) for a technical task. His focus is on the physical effects of his engineering problems. He wants a developing framework, which allows building standard methods basing on the RCE without having to be familiar with all the software techniques, which are working in the background. He is responsible for the technical use of the SESIS system.

3. The Computer Scientist develops and supports the total RCE platform. He is familiar with the software techniques used within the RCE, as they are the data management, the user identification and the network communication. He is not an expert for questions of engineering methods.

The RCE is based on a flexible OSGI plug-in architecture allowing different developers to build their own functionality without having to build the RCE. It is possible to build only the own plug-in and add it to the already existing RCE. A set of plug-ins is part of the RCE as shown in Fig.3 (data management, update, privileges, service broker, communication). The power-user can use this platform by developing his own plug-in. For example, the wrapper plug-in, which has an interface for external code, would be such a plug-in. The external code could be a classical E4 method.

This plug-in architecture has two main advantages. (1) There is a clear interface definition between the RCE system and the numerical methods and (2) it is possible to distribute the complete SESIS system in different configurations. Not everyone who uses SESIS has to get every functionality. If there are licence problems by some functionality, the corresponding plug-ins can be distributed by their own, directly to authorized partners. This allows an organisation to develop functionality on the RCE platform without having to make this functionality becomes part of the complete SESIS system, with the consequences of an automatic distribution to all SESIS partners by the next update. A system based on a plug-in structure allows a much better protection of own investments.
Fig. 4 shows another important feature of the RCE architecture. The system kernel of the RCE is designed for a distributed network. If different computers have installed the RCE kernel, these kernels can build a virtual network by themselves. A user working with his own client is able to use functionality and data from the distributed RCE network if he has the correct permissions. This permission handling will be managed by a system of certifications.

3. Technical problems and solutions of integration

This section explains which technical problems appeared and how they were solved. The main problem at the beginning was the incompatibility between interpreted byte code of a Java application and a classical stand-alone FORTRAN application with object files compiled to a specific hardware. Because the changes to the existing source code should be minimized, it had been discussed, whether it is useful to let the new basic kernel RCE and the existing software E4 as two separated processes on a system. After all, the main task of the SESIS project is not to improve the methods for ship design but to develop a basic framework to support modern IT demands. Therefore principally it is acceptable to keep the classical method. Even if the user-interface is not completely up to date, it has been proven its use in practise. An approach would be to run the naval architect software stand-alone as up to now and to realize the communication with the RCE in the background by a special communication library over inter-process communication. After initial discussions, some performance tests showed that inter-process communication is very expensive and a use of naval architect functionality within the RCE system would be impossible. Thus a closer connection between E4 functionality and the RCE was necessary. The only way to connect compiled object files with a Java application is to use JNI (Java Native Interface). This means to put all needed functionality (compiled object files) to a shared-library, to add interface descriptions and to load this library during runtime directly to the Java-Virtual-Machine. The difficulties of this approach will be discussed now.

3.1 Modifications of the existing methods

It is difficult to modify a software system continuously being in use. However, it was necessary to modify the E4 methods to generate shared libraries for the RCE system without splitting off a separated version of E4 methods. The consequence was to develop a build system for the source code, which automatically allows generating, based on the same source code, stand-alone methods for the E4 system and shared-libraries for the RCE system. An aggravating fact is that parallel to the adjustments for the RCE system a porting of E4 from the UNIX operating system HP-UX to Linux is in process. Although both operating systems are UNIX systems, there are many problems in detail. The existing system was not able to handle all these different demands. Based on the GNU tool Make, it has been implemented. The problem of different target platforms and different types of output
(executables/shared-libraries) will be handled by the build system, which uses the correct compilers and options for the special purpose. Needed differences in behaviour of the programs are handled by a configuring of the source code with a pre-processor. Among other things this approach was used to distinguish between executables, static and shared libraries. Fig.5 shows such a part of starting an E4 method. The original version contains only the section program. The macro variable E4USECPPMAIN triggers now whether the E4 method will be compiled as a stand-alone executable with a "program" statement, or if a "subroutine" statement will be used for generating the start-point. This little switch allows building automatically the different types of software.

```c
#ifdef E4USECPPMAIN
program E4StandaloneFortranMain implicit none
   call E4StandaloneFortranEntry()
end program
#endif

#ifdef E4USECPPMAIN
subroutine E4StandaloneCPPEntry
#else
subroutine E4StandaloneFortranEntry
#endif

... 

dend subroutine
```

Fig.5: Excerpt from E4 code, source: FSG

Because now a defined starting point is available, a standardized interface to a Java-Virtual-Machine can be implemented. This interface can be developed independent from the actual source code of a method. Even if this interface is in development and not usable for a real purpose, it is continuously possible to generate working methods.

3.2 Coupling a Java Virtual Machine with compiled object code

This section describes the handling of shared libraries in more detail. On a first view, it is only necessary to use the "-shared" option of the compiler to produce shared libraries. Fig.6 shows two examples of linking an executable. "Prog1.exe" is linked static. All functionality is connected to one big binary. Different executables, using the same libraries, have to store all needed object code by their own. Therefore the binaries of executables are relatively big. In contrast "Prog2.exe" is linked shared, i.e. this application only has to store the object code of the main program. All source code of the libraries is stored in the shared libraries. This object code will be loaded by the scheduler of the operating systems (OS) on starting the process. Thus the OS has to solve all dependencies as shown in Fig.6 during runtime and not during compilation time. This approach needs a strict version control system. Otherwise it cannot be guaranteed that a method is useable when needed. One incompatible library is enough to prevent the execution. Apart from that, this approach allows saving memory, because the OS can share such a library to different processes, i.e. the OS has only to load a shared library one time and all different processes can use this one. This approach is possible because different processes have different address spaces by the scheduler of the OS. In combination with the MMU (Memory Management Unit) of the processors chip set, every process has its own set of base addresses, Fig.7. Because all memory references within the object code of a shared library are offsets to these base addresses, there are no memory conflicts if a library often uses static memory as global variables. By activating a sleeping process, the scheduler activates at the same time the corresponding bases addresses for the MMU. Therefore, the activated process has automatically its own memory environment as symbolised by the red arrow. This safe mechanism has proven itself for a long time and is described e.g. in detail in Tanenbaum (1992).
In combination with a Java-Virtual-Machine (JVM), there are the same problems. The concept in the SESIS project is that the RCE is a Java application, which supports the running naval architect applications. These applications are plug-ins, which do not run as stand-alone processes but as threads within the JVM. As a consequence, all applications are part of the same process. I.e. the object code of used shared libraries has a reference to the same base registers as shown in Fig.8. Therefore, two applications using the same library can destroy the memory state of each other. The security mechanism of the operating system in combination with hardware is not available by using threads for different applications instead of processes.

An E4 method has to be transferred into a shared library. An obvious approach would be to compile all needed libraries of an E4 method not static but shared. Additionally, the source code of the main program can be compiled to a shared library too, as Fig.5 shows by using the pre-processor. In this case the “Lib1” and “Lib3” in Fig.8 are the main programs of two E4 methods and “Lib2” would be e.g. a numerical functionality used by both methods. Resulting from this conflict, a design decision was made to limit one E4 method has to be transferred into exactly one shared library. Thus a shared-library used within SESIS has more analogy with the static link executable of Fig.6 as the shared-library one. Fig.9 explains this mechanism. Because an application library is a container of all object code of an E4 method, the compiler has already allocated memory space relative to the generated library. A second library, using the same object code of a classical E4 library, labelled “Lib2” in Figs.7 and 8, will get an own memory space relative to itself. Although, the memory spaces of the
different application libraries are part of the memory space of the same process (they are still part of the JVM process), they have within this process-memory an own part, because they are two different shared-libraries. This allocation is handled by the dynamic link loader and not by the process scheduler of the operating system.

This approach has the further advantage of easy version handling. Because all object code needed by a method is included in one library, there are no problems in resolving dependencies between needed libraries. This allows an easy distribution of updates in a distributed environment without maintaining a complex dependency database. The application library can be stored as one item within the JAR file of an RCE plug-in. The plug-in concept allows differing between the main RCE system and additional plug-ins. Because one plug-in is only one JAR file, it is easy to handle these plug-ins. These files can be combined with a certification strategy, whereby it is easy to implement different licensing and update strategies. In terms of memory effort, this approach is sub-optimal, but it avoids a total redesign of the existing E4 methods to eliminate all memory access of global or static variables. In a classical FORTRAN environment this is nearly impossible. The approach explained here and shown in Fig. 9 allows using the existing source code without radical changes.

To summarize, it should not be asserted that the duplication of object code is the best way of software design, but under practical decisions it is a good choice in view of the constraints in a large integration project as SESIS. Furthermore, there is now a clear interface between the system kernel and the physical/numerical part. Thus software scientists and engineers can concentrate on their own problems with only a small coupling to the problems of the other partners, which allows an efficient workflow. Meetings between all project partners can be focused to design decisions, which are relevant for the global system. Because of the strict partitioning between the different software worlds, problems of one part have normally no side effects on the other part.

3.3 Loading / Unloading of Application Plug-ins

After explaining the mechanism of integration of classical FORTRAN/C/C++ methods in a Java system, we will focus on the handling of the shared libraries. At first glance, there is no problem, because the Java standard has the function “System.loadLibrary(<LibName>)”. However, this function is problematic to use in the SESIS environment. The main problem is that there is no unload function. But such a function is fundamental. The application libraries described in the previous section are big because of the inclusion of all relevant object code in one library. If a user is working with the SESIS system and is starting one method after another, these methods will be not automatically unloaded after use. Thus over runtime the memory of the used computer will be filled continuously. The pure Java functionality does not allow to control which libraries are really necessary during the use of SESIS. A second problem is that some older methods are not properly initialised. Historically, simulation software in FORTRAN was developed without stringently initialising all variables. Compilers often performed the initialisation automatically, but this cannot be guaranteed, particularly not when using optimisation flags during compilation. It can happen that the functionality of a method depends on the random memory state at starting time of the method. Such bad software design has to be fixed in the source code of the method, but because of limitations in man-power these bugs can be fixed only in a long process. As a work around, such methods are often compiled with the debug flag of a compiler, which forces the compiler to include initialisation code within the binary. But this initialisation code is only executed once at starting time. In combination with the shared-library concept of SESIS there is another problem. In a classical environment methods are stand-alone processes. If a user needs different methods in a sequence or in a loop, these methods are started by the operating system as a new process and therefore the compiler included initialization is executed. Within the SESIS system the RCE is responsible for starting the application bundles. This means, the initialization code of the library will be executed only once at loading time. If the user of SESIS starts an application plug-in twice during a session, the shared-library will not be initialised a second time. This can result sometimes in corrupt computational output. The forced compiler included initialization code is only a work around for not fixed software bugs! But a system, which should be designed as a framework for a numerous software, has to be stable enough to handle methods, which
do not satisfy high quality standards. E4 alone consists already of more than 100 methods. SESIS will be an even larger framework, where we do not know yet which software will be integrated in future.

Resulting from these assumptions, an own dynamic library loader and unloader has been implemented in the RCE. By using this loader the RCE has the possibility to control exactly the application libraries. The RCE can guarantee that the start of a plug-in enforces a new load of the corresponding library and that the initialisation code is executed before use by the user. A software design problem was that such functionality needs a direct communication with the operating system. But the main RCE concept is to develop the RCE framework as a pure Java application to guarantee platform independency. The dynamic link loader has to be native code in the same way as the application libraries. Thus a shared-library of this loader is needed for every platform on which a plug-in with an application library is available. At the moment the focus for this loader is on a Linux system, because E4 methods selected for integration, are running on this operating system. But for future purpose a loader for Windows would be useful, too.

4. Link-up of the bridge simulator Simflex

So far only the structure of the SESIS system has been explained. Now we focus on how this framework can be used for naval architectural applications. We want to develop a link to the bridge simulator Simflex. This requires extraction of physical characteristics of a ship design from the ship design tool. In the SESIS project, this means using the functionality of the E4 methods within the new SESIS environment. Therefore a new SESIS application has to be implemented. This application is a new naval architectural application, which is not part of the integration task of the existing E4 methods. It will be a pure SESIS application and should demonstrate how naval architectural software can be developed within the new framework. This issue needs the following tasks:

1. Java plug-ins basing on the RCE have to be designed.
2. Fortran functionality of existing E4 methods has to be used.
3. A useful flow control between the Java and FORTRAN world has to be designed.
4. All necessary information has to be prepared and exported to Simflex.

![Fig.10: Screen-shot of the new RCE platform](image)

After solving the second item, as described in Section 3, the actual Simflex application could be designed. This application has been designed as shown in Fig.9. A RCE plug-in controls the user interaction, and a shared-library imports all needed naval architectural functionality. This shared library is a collection of existing E4 libraries and newly developed Simflex relevant functionality. The corresponding Java plug-in controls the communication with the RCE. It loads and unloads the shared
application library and handles the workflows of the application. Especially the last point is a main
difference in contrast to the existing E4 methods, which are integrated to the RCE framework.
Because these are existing methods, the workflow is already implemented within the main program of
the FORTRAN part of the application. These methods are classical procedural programs with a
workflow symbolised in Fig.10 (A). The control loop is part of the algorithms and user interaction
depends from state of the workflow. Modern applications are normally event driven as shown in
Fig.10 (B). This workflow does not have a main-loop, which controls the application. The main-loop
in event-driven application is only needed to poll continuously I/O-devices and react to events from
them. Such events are for example mouse- or key-events. The task of the software engineer is not to
define exactly every sequential step of user-interaction and numerical processing, but the way of
handling the user input.

For existing methods it is nearly impossible to redesign the workflow. Moreover the benefit of such a
redesign would be less than the effort invested. The disadvantage is that only little functionality of the
RCE is available within a method. But because the existing methods are already designed without the
new RCE functionality this is no disadvantage at the moment. Just in the moment, when these
methods will be adjusted with functionality of the RCE there will be some problems. But such
adjustments are at the moment not part of the project. Nevertheless they will be not ignored. Concepts
are available to implement an interface for calling RCE functionality from the side of the application-
library. The Java-Native-Interface allows such calls from native code to Java, but the problem is to
handle objects from a procedural language like FORTRAN. Approaches as described in Abels (2005)
can be used to handle object-oriented tasks within FORTRAN software, if they are necessary for
communication with the Java part of the RCE framework.

But differing from existing methods, which should be integrated in SESIS, new methods are less
restricted. Because the Simflex method is totally new, the workflow has to be designed from scratch.
Now the Java part of the application will be used not only for loading and starting of the method, but
to control the complete application. I.e. the interface is more complex as one of a classical method.
Fig.10 (B) shows that there are lot of calls from the GUI plug-in to the FORTRAN library. By
implementation of the main control on the Java plug-in part, the software designer can use in an easy
way both, the Java RCE functionality and the numerical FORTRAN libraries with all naval architect
know-how.

The RCE framework handles the database of ship information. Corresponding to the privileges of the
user, it is possible to initialise the FORTRAN application library with the needed data. Due to
historic reasons, the E4 library functionality need a set of binary files, which represent the database of
all ship information. A long-term target is to map this binary file database to a relational or object-
oriented database. In a first step, the binary format will be used further, but the files will not be longer
handled by the clients’ file systems. Each binary file will be stored as one item within a RCE
controlled database. This database does not need to be located on the local client. It can be somewhere
in the distributed RCE network. The RCE will handle the communication with the database, while the
FORTRAN libraries will only see a temporary local copy of the database files. These files will be
locked during use of the library and written back after use to the real location of the database,
wherever it will be. These details are no longer relevant for the numerical FORTRAN library. E.g.
information about resistance curve, manoeuvring or seakeeping behaviour can be used as before. But
in contrast to the existing integrated E4 methods the new one can change the database during runtime
of the application. Because the control loop is on the side of RCE, the Simflex method triggers the
loading and initialising process not only once at starting of the application, but every time it is needed.
This allows to use rudder, propeller or other information not only from the actual project. This is
momentary no feature of the classical data model. At present, there are special import/export
functions necessary to transfer explicit information from one project to another.

Further the Java plug-in is able to use directly all file input/output and network functionality of the
RCE. The FORTRAN part of the application will load and calculate all characteristics of the ship
design which have to be passed to the bridge simulator Simflex. This is primary a numerical task
which is already implemented within the existing E4 system. The post-processing and transferring of
information to the correct destination is primary an IT task, which will be supported by the RCE.
Depending on whether it has to be stored just to a special destination within the file system or perhaps
transferring it directly with email to someone, the corresponding output module can be used.

This software design structure allows large flexibility. The power-user can develop functionality as
before. But if the classical software environment is not enough, one can use the new features of the
RCE. While work on the Simflex method continues, the main structure has been implemented and
proven its functionality. Ship design characteristics have been extracted from the classical E4
environment. The new SESIS application has shown that the software set-up is able to handle the task
of building a link from the design tool to the bridge simulator within the SESIS framework. Soon it
should be possible to create new designs and simulate those in bridge simulations as shown in Fig.11.

5. Conclusion

Different types of software can be integrated, but there are many detail problems. Especially the
combination of interpreted Java byte code and compiled object code is complicated, requiring closer
examination of the memory handling. Features, which are normally handled by the operating system,
have to be handled within a JVM by the software developer. This requires effort, but is feasible. The
integration of a big software system like E4 is possible. Theoretically, there are no principal problems.
The Java-Native-Interface between a Java and native code allows implementing all types of
interfaces. In practice there are many problems, because the existing source code of such projects
imply many constraints. Besides aspects of software design, restrictions in man-power drive solutions
in practice. A balance between clear and quality software and required effort in man-power has to be
found.

The Simflex link-up allowed to demonstrate that the problem of software integration can be handled.
Using shared-libraries as a container for the needed object code of existing FORTRAN methods
allowed to solve problems of memory consistence. Further, such a container allows an easy handling
of loading and unloading of functionality which is necessary for the JVM, by memory handling.

The project has also shown that a complex system, composed of such many different technologies,
needs a close communication between different project partners. Despite an overall coherent structure,
the structure must contain a separation between IT parts and engineering parts to allow individual
specialists to work on their fields of expertise. Suitable interfaces require communication between the partners.

Beside the integration of existing FORTRAN based ship design methods, it was investigated, in which way a power-user (i.e. an engineer with average know-how in software techniques) is able to implement a naval architectural application within a complex IT framework. A design structure for using existing numerical functionality was developed. The link-up from SESIS to the Simflex bridge simulator is under development, with the principle functionality being already proven.

6. Acknowledgements

The author wishes to thank the German BMBF for the funding of the project SESIS. Without the support of BMBF, the research work would have not been possible.

References


