WP Validation Deliverable D 532: Migration Methodology Handbook

CAPE-OPEN
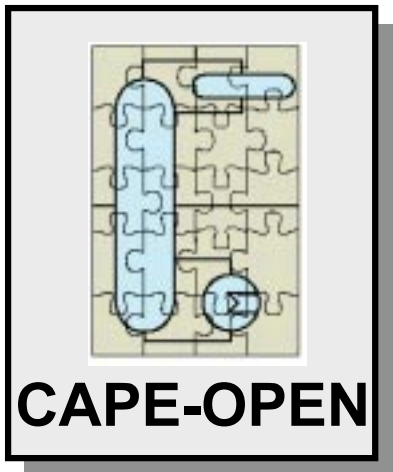
Jörg Köller, RWTH.IS

Archival Information

Reference	CO-VALI-RW2-01
Authors	Jörg Köller
Date	30 June, 1999
Number of Pages	
Version	Version 1
Filename, short	D532 Migration Methodology Handbook.DOC
Filename, long	
Location: Electronic Hardcopy 1 Hardcopy 2	
Reviewed by	S. Artlich
Approved by	
Distribution	
Originating Organisations Reference Number	

Summary

This document describes the work done in the VALI work package concerning the migration of legacy software systems to the CAPE-OPEN standard. It gives technical advice on how software migration to the CAPE-OPEN standard should be done. Additionally, it reports on several examples implemented as proofs of concept for the proposed migration strategy.

Archival Information Page

Contents

1. Introduction and Overview

This document is the deliverable on migration of legacy software to the CAPE-OPEN standard. It was created as part of the VALI work package within the Brite/EuRam BE-3512 project 'CAPE-OPEN: Next Generation Computer-Aided Process Engineering Open Simulation Environment'.

The VALI work package was formed to ensure that the standard developed within CAPE-OPEN is consistent and practically applicable. Applicability means that it must be possible to implement pieces of software which are CAPE-OPEN compliant. Furthermore, it must be assured that the benefit from the CAPE-OPEN standard is not limited to new software implemented on the basis of the standard. On the one hand this applies to commercially available simulation systems which should support the CAPE-OPEN interfaces in the future. On the other hand most operating companies have their own in-house simulation systems which have consumed a lot of resources during their development. It is important to retain the applicability of the CAPE-OPEN standard for these legacy code systems. These systems are monolithic applications, mostly written in FORTRAN, without object-oriented interfaces. Therefore, the legacy systems are hard to maintain and integration with other pieces of software is very difficult. Especially, a direct integration into a component based framework like CAPE-OPEN is almost impossible. But because a lot of resources have been invested in these systems it would be a big waste of money if a combination of legacy systems and the CAPE-OPEN standard were not possible.

Therefore, we will present a migration strategy for FORTRAN based legacy systems in this handbook. However, our strategy is not limited to mere FORTRAN migration but can be extended to integration of complete tools. We will present three examples to show how migration can be performed on different levels of complexity. We will start with a specific migration for unit operations coded in FORTRAN followed by a more generic migration for the IK-CAPE thermodynamics package. Finally, we will present an overview about the integration of a complete tool namely gPROMS.

The examples presented in this guide use CORBA as middleware approach. But our strategy can be applied to COM/DCOM as well because this is only an implementational and not a conceptual issue.

We will begin with an overview of possible migration strategies and discuss some technical issues concerning programming languages and techniques. We will tackle performance and maintenance problems. After that we will go into detail and show based on three examples how the migration was performed in the VALI work package. We will conclude with an outlook to other migration techniques.

2. Possible Migration Strategies

The most important requirements to a migration strategy can be summarised as follows: It should be easy to apply, fast to implement, and be widely applicable. Especially the latter is important because we have legacy code systems based on different platforms.

The main problem we are facing when migrating legacy code systems to component based software is the different views to a system. The component approach is based on object oriented techniques. In the FORTRAN world we have only a collection of subroutines for different tasks. Therefore, a migration strategy has to build a bridge between both worlds.

Another important aspect is that the programming language we intend to use for the migration is supported by the middleware approach we have chosen. Looking at CORBA this is only the case for Java, C++, C, Ada, COBOL, and Smalltalk. Therefore, just using FORTRAN77 or FORTRAN90 only is not an option. For COM the situation is quite similar.

We will now present different migration strategies and discuss their advantages and drawbacks. Because not all readers are familiar with the programming languages occurring in this context we will also give some technical details.

2.1.   Re-implement from Scratch

The simplest migration strategy seems to be just to re-implement the existing FORTRAN systems using one of the programming languages mentioned above. This means that we would have to create a new system that has exactly the same behaviour as the old implementation. But this goal is very difficult to achieve for two reasons: The legacy system has been used and maintained for a long time and is therefore well tested. Hence, such systems are normally very stable and it is very unlikely that new errors will occur. This is not the case for software just implemented from scratch. It will have a lot of errors resulting in system crashes or, even worse, in inconsistencies between the behaviour of the old and new implementation. Finding and fixing these bugs will take a lot of resources and is therefore expensive.

But even before the testing stage we are facing difficult problems if we try to re-implement a legacy system. To be able to write a new implementation we have to understand what the old system does and how it works. This is a very difficult task for various reasons. First, FORTRAN is a language that only offers poor support to structured programming. Therefore, large FORTRAN systems are hard to understand. This problem is made worse by the fact that often the people who implemented these systems are no longer available. Hence a lot of implicit knowledge about the system is lost. But this knowledge is very important because software documentation tends to be incomplete and hard to understand without knowing the history of a system.

Therefore, re-implementing legacy systems seems to be dangerous and expensive which is why we did not choose this way. But this does not mean that it is impossible. There could be situations where a re-implementation would be reasonable. Especially, for small software systems this could be feasible. But if we decide to implement a completely new system, modern object oriented languages like Java or C++ should be used which are directly supported by the middleware implementation.

2.2. Preserving the FORTRAN Code

As we have seen in the last section re-implementing a legacy code system is a dangerous approach for several reasons. Therefore, we have chosen the other alternative: When migrating legacy systems to the component based world we try not to touch the FORTRAN code at all. We treat the FORTRAN code as "black
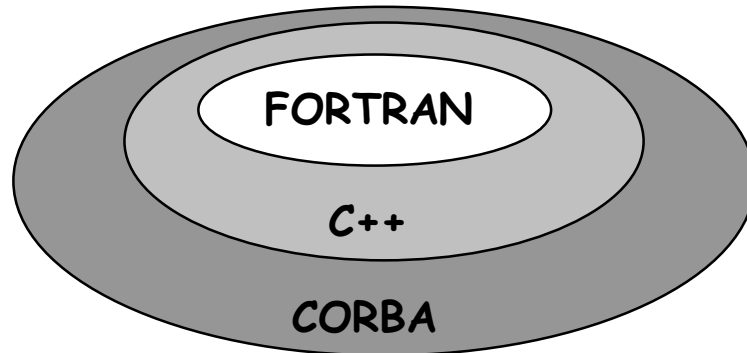
Figure 1: Preserving FORTRAN code

box" whose functionality we use. But we are not interested in how this functionality is implemented. Instead, we provide an object oriented shell around this black box integrating the legacy system into the component based framework. This shell is implemented using a modern object oriented language like Java or C++. Based on this shell we then provide a CORBA layer to make the functionality available to the outside. This layer is also implemented in Java or C++. We will present more details on the implementation of the shell and the CORBA layer in section 3.

By not touching the legacy code we avoid the problems coming along with re-implementing. As we do not alter anything in the FORTRAN code itself we will not introduce new bugs to the core functionality of our system. There will be, of course, bugs in the new shell around the legacy code. But because only the shell contains new code we know where to look for these errors. Another consequence of the black box is that the behaviour of the migrated system will be exactly the same as the legacy system. Furthermore, we preserve all work put into the legacy system regarding optimisation. We do not have to think about optimising the code because this was already done by the original developers.

Having seen all the advantages of a FORTRAN black box one important question remains: How can we integrate the FORTRAN code into our new system without directly touching it? We will present and discuss two possible solutions for this problem. One of these we have implemented but the other one could be tried as well. We did not have the time within CAPE-OPEN to try both but it could be done in a future project.

2.2.1. Cross-compiling

Cross-compilers are tools for mechanically translating one programming language into another. A large set of commercial and non-commercial cross-compilers is currently available. They are not limited to FORTRAN to C or C++ translation but have been implemented for all major programming languages. Therefore, using them for integrating legacy code into a new framework is an approach applicable to

a wide range of legacy systems. Migrating a legacy system then would consist of the following steps:

- Cross-compile FORTRAN source code to C or C++

- Write shell code in C++ for the cross-compiled code

- Integrate shell code and cross-compiled code

- Provide a CORBA layer in C++

- Compile and link everything using the same C++ compiler

By using these cross-compilers we can bypass incompatibilities between different programming languages. These incompatibilities cause a loss of performance if the wrapping strategy (next section) is applied. One example are the differing array conventions in C and FORTRAN. In FORTRAN arrays are stored in row major form and the lowest index is normally 1. C and C++ store arrays in column major form and start indexing them with 0. If we use the wrapping approach we have to take care that a conversion between these storage models has to be performed. Fortunately, in the CAPE-OPEN context these problems are unlikely to arise.

The use of cross-compilers bypasses these incompatibilities because the executables are generated by a single compiler from the same source language. But there are also some drawbacks when using cross-compilers. The first restriction is that in order to use cross-compilers the FORTRAN source code must be available. This is not always the case. Furthermore, cross-compilers usually generate source code which is nearly impossible to read and understand. If you want to maintain the code you have to do it on FORTRAN level and than again cross-compile it. This is dangerous because it can lead to inconsistencies. Another more technical aspect is that FORTRAN and C/C++ compilers do different optimisations. Therefore, FORTRAN code which includes optimisations manually written may perform a lot worse after cross-compilation than after direct FORTRAN compilation.

But all these issues would have to be investigated very closely to make a final decision whether to use this strategy or not. In CAPE-OPEN we did not choose this approach mainly for the reason that we did not want to be dependent on source code availability. But we do not claim that wrapping is the only possible thing to do.

2.2.2. Wrapping

Using the cross-compiling strategy we do not touch the FORTRAN source in the sense of not changing a line of FORTRAN. But we transform the complete source code mechanically into C or C++ code. Using the wrapping strategy we do not touch the FORTRAN source at all. We directly integrate the object code generated by a native FORTRAN compiler. Therefore the main drawback of the last approach is not present when wrapping FOTRAN code: The source can but does not necessarily have to be available.

Having generated the FORTRAN object code the wrapping strategy is very similar to the strategy presented in the last section. It can be summarised as follows:

- Compile FORTRAN source or use precompiled library

- Write shell code in C++ for the FORTRAN object code

- Integrate shell code and object code

- Provide a CORBA layer in C++

- Compile CORBA and shell layer using the same C++ compiler

- Link all object codes

When using the cross-compiling strategy the integration of shell code with the result of the cross-compilation is very easy because both use the same language. Here we have to integrate object code generated from FORTRAN with C or C++ source code. This is not trivial because of the incompatibilities mentioned in the last section. We cannot just call the FORTRAN routines in the source code of the shell layer because calling conventions and data alignments differ a lot.

We would be stuck here if there were no tools available to mechanically perform the conversion necessary here. Implementing these conversions manually is a very difficult procedure and very error prone. But there are tools available doing this work for the programmer. For CAPE-OPEN we have used a collection of macros called *cfortran* which is freeware. It provides macros for calling FORTRAN from C/C++ and vice versa. Additionally, it contains all facilities needed for data conversion between these languages. For calling a FORTRAN routine we only have to use two different macros. One of these macros is needed to declare the FORTRAN routine. The other one is used every time when the FORTRAN routine is actually called. Because *cfortran* is a simple include file using these macros and integrating the FORTRAN object code becomes quite easy.

But as discussed in the last section there can be a performance problem caused by the conversion done by *cfortran*. If we compare both strategies we can clearly see that it would be possible to combine them because after the integration with the shell code both are essentially the same. We could use cross-compiling where necessary and possible and wrapping when the source code is not available or the loss of performance is not great.

3.   The Shell and CORBA Layers

We will now present some more details on how the shell and CORBA layer are implemented. As their implementations differ only very slightly in both migration strategies we will use the same examples for both cases. These examples will show what these layers look like in different situations depending on the complexity of the system we are trying to migrate. We start with a simple wrapping of a FORTRAN unit operation which will result in a single component. The next and more complex example will be the migration of the IK-CAPE thermodynamics package. The result of its migration is a system of several components according to the CAPE-OPEN THRM specifications. Finally, we will give a short glimpse at tool integration which was prototypically performed on gPROMS.

The shell layer is responsible for providing the non-object-oriented black box with an object-oriented interface. The CORBA layer makes the objects generated by the shell layer available to the outside as components. The thickness of both layers strongly depends on how the migrated system will be used. A well structured shell

layer is important if we want to extend the functionality of the migrated software. We could just add new methods and without having to implement them in FORTRAN. Additionally a good object-oriented shell enables us to use the wrapped black box not only in a CORBA environment. Instead of putting a CORBA layer on it you could use a (D)COM layer or directly integrate it into an application without any middleware.

3.1. Specific Wrapping

If we want to use the new component in a very specific way only then there is no need for developing an elaborated object model for the black box. This especially holds for small pieces of legacy software if we are sure that we will never have to extend its functionality. In this case the shell layer can be very thin or even non-existent. We can directly use the object model imposed by the CORBA interface specifications. Therefore, the shell layer can be integrated into the CORBA layer which then translates all CORBA calls directly to FORTRAN subroutine calls. But again it must be emphasised that the solution is only possible for small legacy systems which will be used very specifically. In larger systems or systems that will be modified in the future this should be avoided under any circumstances.

As a proof of concept this strategy was applied to some unit operations. These units were successfully used in the demos on ESCAPE 9 in Budapest and the CAPE-OPEN final meeting in Paris.
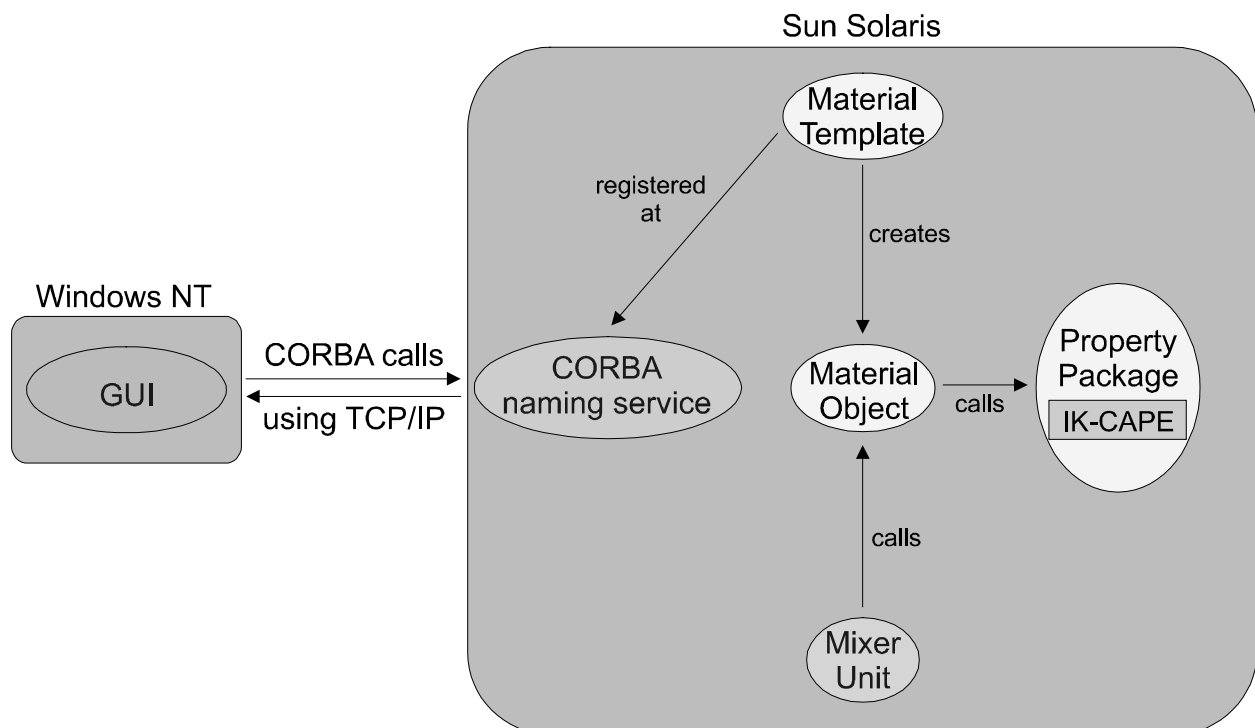

Figure 2: IK-CAPE Migration

3.2. Generic Wrapping

The next stage of complexity is a more generic wrapping of the black box. By this we mean that the migrated component is not used in a very specific context as in the latter case. Here it should be possible to put other layers than a CORBA layer only on the shell level to ensure that the wrapped black box is usable in other contexts. Therefore, the shell layer has to be a lot thicker than in the case of a specific wrapping. It must provide an elaborated object model wrapping up the legacy system's core functionality implemented using the FORTRAN subroutine model.

Developing such an object oriented model is not only important for being able to integrate the wrapped black box in a wide range of environments but also when it comes to the maintenance of the system. If we decide to extend the systems functionality we should follow our philosophy not to touch the FORTRAN code. Therefore, we must implement new functionality using the language we have used for implementing the shell layer. But extending the system in this language is relatively easy if we have developed a good object oriented model. We can extend the system just by adding new classes and by using the inheritance mechanism offered by the language we have used.

The thickness of the CORBA layers in this case strongly depends on the similarity of the object model imposed by the CORBA interface definitions and the object model of the shell layer. If both are similar or even the same the CORBA layer is very thin. In this case it only forwards the calls made to the CORBA objects to the shell layer. This is what happens in the IK-CAPE wrapping because the shell layer object model was developed with the CAPE-OPEN model in mind. But this has not always to be the case.

As an example for this approach we have migrated the IK-CAPE thermodynamics package to a CAPE-OPEN compliant properties package and embedded it into the CAPE-OPEN THRM context as shown in Figure 2. To achieve this we also had to implement CAPE-OPEN compliant material template and material object components. Finally, a CAPE-OPEN mixer unit was written which uses these components. To give a proof of concept we have generated a test scenario within the prototypical CORBA simulation environment CHEOPS. An example flowsheet consisting of the IK-CAPE based mixer and a gPROMS based reactor (see below) was successfully solved. This was demonstrated at the ESCAPE 9 in Budapest and at the CAPE-OPEN final meeting in Paris.

3.3. Tool Integration

Having seen that our strategy works for migrating pieces of legacy software we conclude with some thoughts on the integration of complete tools into the CAPE-OPEN component based framework. It is nearly impossible to make some general statements about the thickness and complexity of the shell or the CORBA layer. It strongly depends on the structure and the architecture of the application that is to be integrated into the component based framework. If the application is already object oriented or even component based it should be fairly easy to integrate them. But if this is not the case we will possibly face serious problems. But we do not want to go into detail here because tools integration is a topic that has to be looked at closely in Global CAPE-OPEN. Then we will be able to develop specific strategies for tools integration.

But to give a proof of concept that tool integration following our wrapping strategy is possible the process modelling tool gPROMS was modified in a way that it supported and implemented some interfaces developed in the NUMR work package. The modified gPROMS is now able to expose equation set object components and different solver components. This has enabled us to implement an equation based reactor unit whose equations were set up and solved by the components exposed by gPROMS. This unit was used together with the IK-CAPE based mixer in the scenario mentioned in the last section.

4.   Conclusion and Outlook

In this document we have presented a strategy for migrating legacy code software into a state of the art component based framework. Following this strategy it is possible to re-use a lot of FORTRAN based systems in the CAPE-OPEN environment without having to re-implement them completely. We have shown that the FORTRAN implementations core functionality can be preserved thereby bypassing a lot of dangers arising from re-implementing. We avoid the introduction of new bugs and inconsistencies which could otherwise cause the loss of lots of time and money.

All that has to be done in the migration process is to wrap up the legacy code and provide two layers of new object oriented code. We have given an overview on how the wrapping process is done and how these two new layers can be implemented. As a proof of concept we have presented several examples of migrated legacy systems which were successfully tested in different scenarios.

We have also discussed the integration of tools into a component based framework which will be a key topic for future investigations.