# MILAS

**M**odern**I**zing **L**egacy **A**pplications towards Service Oriented Architecture (SOA) and Software as a Service (**S**aaS)

**Hans Aage Huru**

Master Thesis

University of Oslo / SINTEF

Autumn 2009

## Acknowledgement

This is my master thesis, the last but not least piece in achieving my Master degree at the Institute of Informatics, University of Oslo.

I would specially like to thank my supervisor Dr. Arne-Jørgen Berre for his expert guidance and motivating attitude throughout the whole process of writing this thesis. I would also like to thank the SiSaS project team at SINTEF which have included and motivated me by letting me participate and contribute in the SiSaS project.

Finally, I want to thank my girlfriend Véronique and our son Brage for being patient and understanding during the process, without the support from them it would not have been possible to write this thesis.

Hans Aage Huru
Oslo, November 9th, 2009

## ABSTRACT

It has become evident that software built especially for larger organizations the past 20 to 30 years or so is much more difficult to change and replace than many initially have been believed. This kind of software called “legacy systems” remains critical to organizations, and organizations finds it expensive and difficult to maintain and control the software.

Legacy software often consists of older code written in an obsolete language, no longer thought to students and engineers. If the principles of modern software development are to be followed, several issues will arise facing this kind of legacy code. They are seldom written with general maintenance and openness in mind, but are nevertheless often under constant evolution to adapt to changing business needs.

This in addition to the accelerating changes in business environments such as, the need to meet new markets, new business goals, changes in behavior of clients/customer (fast adaption), and change of rules from government, makes the importance of a controlled, thorough and deliberate modernization of legacy systems.

This thesis will look into the challenges, methods and techniques that exist to allow for a modernization of legacy systems with the principles of SOA (Service Oriented Architecture) and demands from SaaS (Software as a Service) and Cloud Computing in mind. There are two case studies that will be referenced. The first case study is a scientific legacy software written in Fortran, which I have been working with while being at SINTEF during 2008 and 2009. The other case study is a business application written in Progress 4GL, which I have been working with while being at KrediNor AS during 2007 to 2008.

The thesis proposes the MILAS methodology as a general path to modernize a legacy system. A methodology that focuses on the whole process, following standards within the modeling community, and the generation of modernized systems that are highly serviceable and ready for SaaS.

For legacy software and components that for different reasons will not or cannot be migrated to a modern design, the thesis proposes an UML profile PIM4Wrapping. The PIM4Wrapping profile will help modeling these components as services, and support the transformation to PSM (Platform Specific Model).

## TABLE OF CONTENT

## LIST OF FIGURES

## LIST OF TABLES

# 1 INTRODUCTION

There is a waste amount of legacy applications around the world in different organizations, which introduce a wide variety of problems for the owners of these applications. Often the owners feel they have no control over the software, and that it has grown out of proportions.

These applications are often written by people no longer within the organization, and the documentation is sparse. Still the applications are fully functional and contains logic that are hard to recreate or to convert into a more modern language, yet they do work that can be of critical value to the owners.

These days, in light of the interest in SOA (Service Oriented Architecture) growing interest of SaaS (Software as a Service) and Cloud Computing, the interest and need to share business applications with others have increased. Owners want to gain the control of their software and look at it as valuable assets they can utilize and reuse in their organization, and to communicate with other organizations, rather than some "black-box" they do not want to open. Often the current IT-staff only plays the role of maintainers of these systems, and do not have the resources and/or skills to take on a modernization approach.

The common IT-world now sees solutions where applications can be put into some kind of virtual platforms and execution environments (Cloud Computing), which potentially can reach all users all over the world without the normal concerns of distribution and local management of the software. Additionally the new solutions can provide a dynamically amount of computing resources available for the different applications and their users. This makes it more important for the providers of scientific software to make their applications available over the World Wide Web.

## 1.1 PROBLEM

How to choose the best modernization approach when facing the need to change existing legacy systems in an organization? In this thesis context, a legacy system is an old application that still are being used, despite that the structure of the code, or the language of the code are out of date and obsolete. These applications can also be the backbone of the organizations business and are therefore especially crucial for the same organizations. In light of this and the fact that there exist multiple different approaches to modernize these applications, makes it important for the organizations to choose the best solution for their modernization approach.

## 1.2 SCOPE

There are many aspects to consider in this kind of tasks. Ranging from the direct digging and understanding of legacy code to the all the considerations that needs to be done regarding selecting technology, structuring, performance and maintainability of the new system. This thesis will try to take these considerations into account when evaluating approaches to modernizing legacy code.

SOA (Service Oriented Architecture) and SaaS (Software as a Service) are paradigms that will be in focus. SOA has been around for a while, but are still a necessity if organizations are to reach their goals of being adaptable to changing business needs. In addition SOA has introduced architectural concepts that are highly desirable. These concepts can vary depending on whom one asks, but there seems to be an understanding of a common set of principles.

The thesis will look into challenges one might face when introducing SOA in a modernization context.

SaaS as a more recent paradigm is more of a delivery model than an architectural model, and raises new issues to be considered for those who are evaluating SaaS as a delivery platform for their software. With SaaS new business possibilities arise, since the potential number of customers increases when given the possibility to make ones application available as services on the internet.

This thesis will talk about the consideration an organization will get, when trying to a deliver their application as software running as services.

Open source communities deliver software, development platforms, and tools that should be considered in such a context. The cost of licenses and maintenance are often a considerable amount of the total budget for an application, especially if the application is supposed to be long-lived. Open source product will be mentioned as potential target platforms for the modernized systems, since they provide a free downloadable version for the public to explore.

There is also much literature that describes modernization approaches, and the thesis will look into some of these best-practices and see if they apply for the case studies that the thesis are built around.

When it comes to production of source code and documentation the thesis will try to make spikes[1] that can illustrate different approaches. A full scale modernization will be outside the scope of this thesis.

## 1.3 Goal

The goal of this thesis is to find out how modernization towards SOA and SaaS can be applied on legacy systems using a model based approach. The goal is not to give a fully functional modernization methodology, but rather to give guidelines, suggestions and pieces of contributions on how one could proceed.

To be able to reach this goal several questions needs to be answered.

- What are the requirements of SOA and SaaS
- How to obtain a model of the legacy system
- How to migrate to a modern architecture
- How to model services for SOA
- How to model services for SaaS
- How to obtain modernized source code

The purpose with this thesis is to supply a general "methodology" of how to make legacy systems available as services that can be invoked through the internet. The "methodology" will use available modeling tools to help carrying out the task as far as possible, thus making the methodology available for the reader.

[1] Spikes; A small program pieces that are built to prove a concept.

## 1.4 RESEARCH METHOD

The research method applied in this thesis follows technology research method presented in [1]. Solheim and Stølen describes the technology research in [1] as an iterative process divided into three phases. The phases being a problem analysis phase, an innovation phase, and an evaluation phase. The phases as an iterative process is illustrated in the following figure taken from [1].

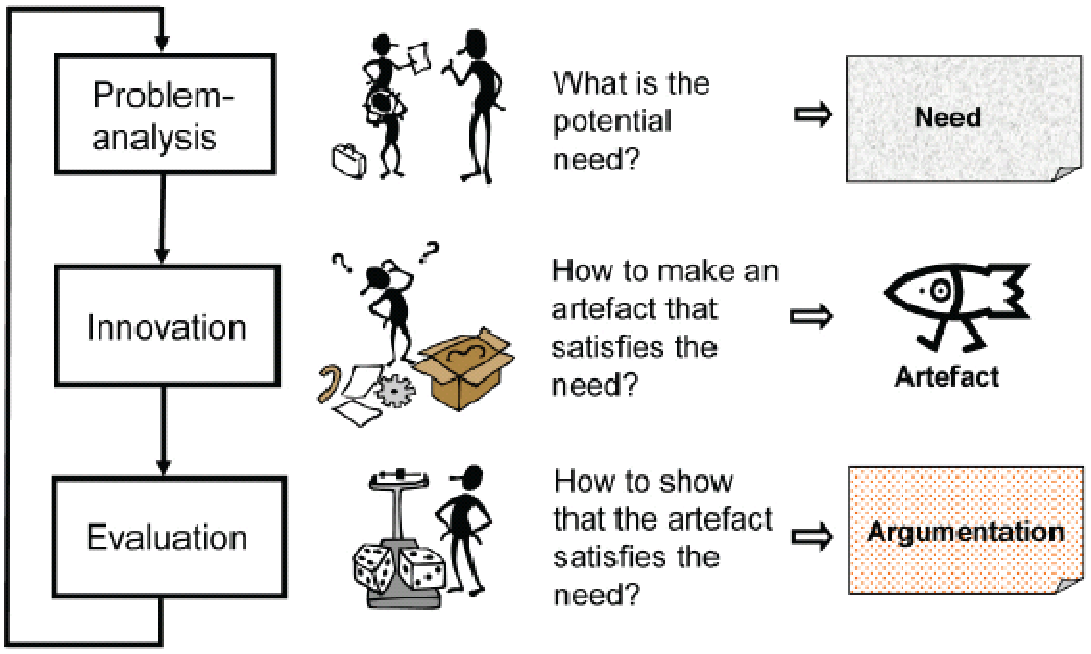

Problem-analysis
What is the potential need?
Need
Innovation
How to make an artefact that satisfies the need?
Artefact
Evaluation
How to show that the artefact satisfies the need?
Argumentation

**Figure 1 Method for technology research main steps**

The three phases are described as:

- *Problem analysis:* To do a survey on a potential need for a new or better artifact by interacting with possible users and other stakeholders.
- *Innovation:* Trying to construct an artifact that satisfies the potential need.
- *Evaluation:* To formulate prediction based on the identified problem, and to see if the predictions come true.

For this thesis the different phases were as follows:

- *Problem analysis:* During the problem analysis I worked at SINTEF with the scientific software OSCAR footnote system from 2008 to 2009, in the context of the SINTEF SiSaS (SINTEF Software as a Service) project, For the business software K90 I worked previously (from 2006 to 2008) in a business application software company that developed software with Progress.

  These systems where analyzed with respect to challenges for migration towards SOA and SaaS. In both cases there were challenges that had to be met, since there were no clear paths to solution. There was a need to see how such approaches could be carried out following a sound methodology that would lower the risks for failure and heighten the potential for success.

- *Innovation phase:* in the innovation phase the focus were on how to solve these problems with as little effort as possible, but at the same time ensuring high quality for the solution. In this phase MILAS a model driven methodology were put forward to try to satisfy the identified problems in modernization of the case-studies.

- *Evaluation phase:* The MILAS methodology was then put to a test by letting both case-studies undergo the steps in the methodology.

## 1.5 STRUCTURE

This thesis is structured in nine chapters as follows:

Chapter 1, *Introduction* explains background of the thesis, describes potential problems, lists thesis goals and presents the research method.

Chapter 2, *Problem Definition with examples* describes the challenges on might face undertaking a modernization project. It also introduces concepts and requirements of modernization project.

Chapter 3, *Requirements* describes in detail the challenges a modernization project will have, especially with SOA, SaaS and Cloud Computing in mind.

Chapter 4, *Evaluation of existing solutions*, evaluates and describes of tools from some of the leading actors in the industry, from OMG and sponsored EU projects.

Chapter 5, *MILAS,* the thesis proposal to a solution is presented

Chapter 6, *MILAS for OSCAR,* MILAS is tried out on the OSCAR case.

Chapter 7, *MILAS for Progress 4GL,* MILAS is tried out on the Progress 4GL case.

Chapter 8, *Evaluation of MILAS,* evaluation on MILAS is done for the two case-studies.

Chapter 9, *Conclusions and Further Work* summarizes contributions and conclusions of the thesis, and points out improvements in the future work.

# 2 PROBLEM DEFINITION WITH EXAMPLES

To get a feeling with the challenges and requirements a modernization of legacy systems will face, it is important to have a case study that can exemplify the different issues that will arise. In this case there are actually two case studies that will be examined to better get an overview over the needs a modernization of legacy systems demands.

The two legacy systems that are used in this thesis are very different in nature, but they have some of the same needs which are to be opened up for other applications and users to utilize. To accomplish this both systems stakeholders has expressed a need to meet SOA demands. In addition it is desirable to have one of the legacy systems modernized towards SaaS.

SOA is required since this is the de-facto standard for open systems today, and the architecture style that over several years has evolved to be the architecture that meets requirements such as openness, reusability, and maintainability best.

SaaS is a requirement for one of the systems since there is an expressed need to get the application out for everybody to use, in addition the stakeholders wants to minimize their resource usage in the fields of installation, implementation, maintenance and support. SaaS can reduce the amount of resources needed since SaaS promotes that here are only to be one installation of the application that serves all users. One application will require less maintenance and reduce installation and implementation cost to a one-time fixed value. By running the application as SaaS on Cloud Computing requirements such as scalability and performance can be removed from the responsibilities of the application developers to the infrastructure suppliers.

## 2.1 THE CASES TO BE STUDIED

The two different case studies are presented in this chapter; the first case study is about a scientific application written in Fortran[2] that predicts how an oil spill in sea water will develop over time. I have worked with this case study at SINTEF in 2008 and 2009, related to their SiSaS (SINTEF Software as a Service) project.

The second is an administrative application written in Progress 4GL[3], which is the core business application for the organization I worked with during 2007 and 2008. During that period I was involved in the development of the application, the business software k90.

### 2.1.1 CASE I, THE SCIENTIFIC SOFTWARE

At SINTEF, they experience some of the challenges of migrating existing software towards SOA and SaaS, and are currently working on projects with this in mind. In that perspective I was given the chance of looking into the challenges regarding one of their scientific applications, to see what was needed to be done to make it available as SaaS.

At SINTEF there exist several applications that can fit the description of a legacy system. The case to be studied is an application, named “MEMW” or “OSCAR”, developed by the department of “Materials and Chemistry” and consists of code made in both Fortran and

[2] **Fortran** (previously **FORTRAN**) is a general-purpose, procedural, imperative programming language that is especially suited to numeric computation and scientific computing.

[3] 4GL A **fourth-generation programming language** (1970s-1990) (abbreviated **4GL**) is a programming language or programming environment designed with a specific purpose in mind, such as the development of commercial business software.

C++[4]. The C++ code is used for the user interface and to communicate with the Fortran code that consists of mathematical procedures.

The OSCAR applications main purpose is to estimate the probable course and disintegration of oil at ocean surface after an oil spill. To do this the application considers the seas current speed, waves and direction, the type of oil, the weather in the area, and of course the amount of oil actually spilled. As an interface to the users the application uses graphical map display of the desired sea area, and several input dialogues where the user can enter in appropriate data, all written in C++. The information is then passed to the actual executable that estimates how the oil spillage will look like during the next periods of time. The executable named "Fates" is written in Fortran. The passage of data is done via pre-structured files that the executable expects and needs to be able to do the calculations. When Fates has finished the calculations it produces a set of result files that the user interface can read and show graphically in a map, as shown in the figure below.

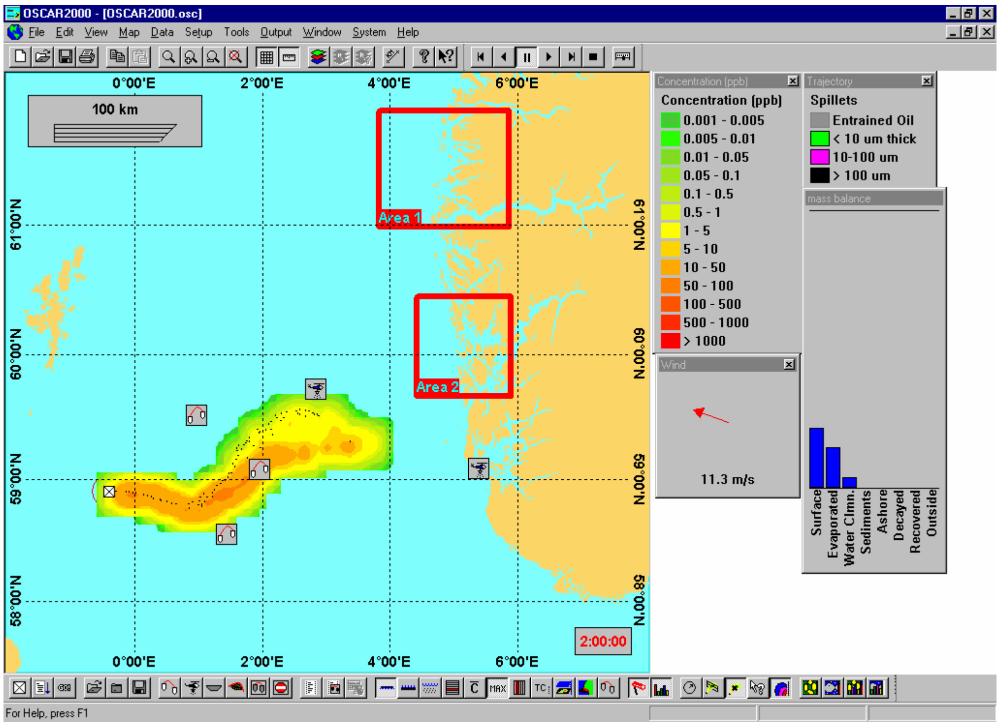


**Figure 2 OSCAR user interface showing oil spill**

[4] **C++** (pronounced "See plus plus") is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as a *middle-level* language, as it comprises a combination of both high-level and low-level language features. It was developed by Bjarne Stroustrup starting in 1979 at Bell Labs as an enhancement to the C programming language and originally named "*C with Classes*". It was renamed to *C++* in 1983.

Typically the code both in C++ and Fortran, though state of the art in their areas, is not build to satisfy SOA principles or with SaaS in mind. The code is rather built for speed and to generate a modern and flexible user-interface. Since the application initially where built to be distributed as a stand-alone application, where the customer where supposed to download and install the software on their own computers, the need for availability and reuse of software where not as present as today.

Now the goal for SINTEF is to build a general purpose platform where the software built in the various departments can be deployed such that the customers can access it on a need to run basis. To do this, the desired platform should meet the demands of SaaS. In addition, if the functionality that are inside the various application were to be made available throughout the organization, there could be a growth in new applications or services offered by SINTEF, and thus increase their income.

In my master theses I will look into different possibilities to lift existing systems, especially OSCAR, to a modern SOA/SaaS platform, and investigate in which approach that can be most efficient and give the best result. The results will be based on my experiences gained while working on the SiSaS project at SINTEF where my tasks where to find possibilities to modernize OSCAR towards a SaaS concept.

### 2.1.2 CASE II, THE BUSINESS SOFTWARE

At KrediNor, where I formerly worked, there exist several applications that can be called legacy systems. Even though they are not too old, they share the characteristics of a legacy system. They are built without composability and evolvability in mind and the code use both user interface elements with business logic and data access.

The core system, that handles most of the business transactions for the company, was started at in the late 80's, when 4GL[5] programming was popular, and 4GL was therefore chosen as the main development tool for the company. The system, internally called "k90", is a debt collection system, and handles millions of transactions every day. Basically the system read in invoices from various flat files delivered by the "creditors" and decides what to do with the invoices depended on their type and age. The business rules or logic for how to treat invoices are quite intricate compared to what one should initially believe, and some of the most important business logic for the company is present here. The interfaces of the system has been build along the way and consists of a variety of character screen menus and programs, in addition to the hundreds of reports that has evolved over time, and several more or less full modules are added to the system such as accounting, credit rating, department and user logics, and law data.

The system is built using Progress Software tools, typically their 4GL language and RDBMS[6] database, which gives client-server architecture. This means that all the user interface, business logic, data access and transactions, are bundled into the same pieces of code. In addition the user interface is character based and the validation of data done partly while editing and partly when doing transactions. The 4GL code is separated into procedures that

[5] A **fourth-generation programming language** (1970s-1990) (abbreviated **4GL**) is a programming language or programming environment designed with a specific purpose in mind, such as the development of commercial business software.

[6] A **Relational Database Management System (RDBMS)** is a Database Management System (DBMS) that is based on the relational model

reside in separate files, and all in all there are about 4000 different procedures that build up the system.

The company has been aware for a long time that this system and others is not up to date when it comes to object oriented programming and SOA, but uncertain how to attack the challenge to create a more modern system that can more easily be utilized by other applications and be more evolvable too meet modern demands. The company has traditionally done business around debt-collection and done so within Norway. Now they look for new business and geographically areas, thus the demand for flexibility, maintainability and reuse of existing business logic has become more present.

In my master thesis I will look into different possibilities to lift existing legacy business systems, especially k90, to a modern SOA platform, and which approach that can be most efficient and give the best result. While working for the company I was given the task to document the system based on the source code and running environment using modern UML design tools, and look into how the system could be modernized.

## 2.2 THE COMPLEXITY OF LEGACY SYSTEMS

Since legacy systems are written and evolved during several years, often decades, there exists a variety of code standards in it, and lack of good structure is often a due to this fact. Legacy systems can contain several factors that complicate the pictures, and a modernization approach will introduce new considerations.

### 2.2.1 COMPLICATIONS

Some of the factors that complicate the picture are listed below and broadly taken from William M. Ulrich's "Transformation strategies" [2] Ulrich.

- *Fragmentation:* Functionality can be split across several environments, which means that there can be multiple systems dealing with different aspects of the same data entities.

- *Redundancy:* Functions, that perform the same or similar jobs, are spread across the system, making it hard to get the full picture of the business logic carried out on the different data units.

- *Technical Diversity:* An organization may be using several different types of hardware, development languages, database systems and communication systems.

- *Design Diversity:* Different but similar functionality are spread among different systems.

- *Timing:* Most legacy systems started as batch oriented systems, and are not online with many of the functions that the business requires.

- *Size:* The size of legacy systems are often considerable, ranging from millions of lines of code to hundreds of millions lines of code.

- *Integrity, semantics and consistency:* The data stored can contain values that are invalid, not used, or give misleading information. Legacy systems are maintained over a long time period and shortcuts are often taken to meet new requirements. The naming and formatting of data elements are often not consistent over multiple systems which leads to confusion and possible mistakes.

- *Accessibility and security:* Disparate data makes it hard to retrieve information fast enough for new demands coming from business reporting or web applications. Multiple sign-on is often required to get security clearance, when accessing multiple systems, networks and firewalls.

- *Flexibility:* All the above factors make legacy system little flexible for changes.

### 2.2.2 DOCUMENTATION & RESOURCES

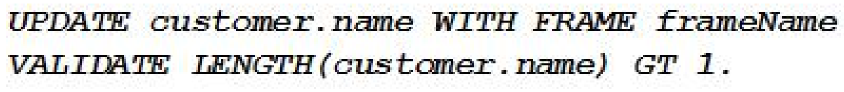
Unfortunately for the organizations that have these legacy systems the documentation of the same software is lacking, or in a condition that makes it useless for reengineering purposes and the builders of the system has long left the premises. This makes it necessary to retrieve the necessary information by oneself before starting the modernization project.

### 2.2.3 CHANGE MANAGEMENT

A modernization approach must also take in consideration the fact that the legacy system probably will undergo a lot of chances and enhancements during a modernization period. These changes must be caught by the modernization team to ensure they will be reflected in the new system. One way to ensure that changes are brought to attention is to make sure that the legacy software is put into a source repository that can track and log changes done to the system. Then these changes must be coordinated with the modernization team to ensure they are not working with old code as their requirements.

### 2.2.4 NON LAYERED

In modern software development it is common to divide the source code into multiple layers called multi-tier architecture[7], where the basic layers consists of a presentation layer, a business layer and an entity/data-access layer. Legacy software however, often lacks this distinction, especially in 4 GL applications. In 4GL languages the presentation, business-logic and data access are bundled together, sometimes all in one statement.



```
UPDATE customer.name WITH FRAME frameName
VALIDATE LENGTH(customer.name) GT 1.
```

**Figure 3 Progress code sample.**

The figure above is an example of a statement in Progress 4GL where the presentation of customer.name, the validation or business-logic and the writing of the customer.name to the database is done in one and the same statement. The "UPDATE" part displays, prompts for, and stores the input data to the database, whereas the "WITH FRAME" statement is user interface layout. In addition to this a validation statement is added at the end "VALIDATE LENGTH" ensuring something is written in the field. This makes it harder to analyze the source code, and will demand a total redesign of the same code.

[7] **Multi-tier architecture** (often referred to as *n-tier architecture*) is a client-server architecture in which the presentation, the application processing, and the data management are logically separate processes.

### 2.2.5 PERFORMANCE

Legacy systems have been tuned-up over the years to perform as well as possible on the given platforms. For a modernization team, this can be a challenge. Especially if the legacy system is build around direct access to the database through proprietary connections like 4 GL languages often has.

A multi-tier architecture will and can be slower in user response, if considerations are not taken when converting the business-logic. Special tailored user-friendly look-up lists that function well in the legacy system can be hard to implement with the same speed and accuracy in modernized environment. Especially if the new user interface is using web services for communication with the business-logic, and parameters and result sets are sent via XML[8]formatted messages.

## 2.3 MULTIPLE MODERNIZATION APPROACHES

There exist multiple modernization approaches, methodologies and tools that can help and guide to the task of moving a legacy system to a modernized system.

This thesis will define its own methodology that should be a better and more effective approach than existing ones, and include the challenges that SOA and SaaS introduces.

### 2.3.1 DO A MODERNIZATION AT ALL

Before starting the task to modernize a system a portfolio analyzes should be carried out. A portfolio analysis establishes measures of technical quality and business value for a set of systems and evaluates this set against the measures [3]

With the above analyze in mind the system can be characterized as a medium to good candidate for reengineering. The business value is potentially high, whilst the technical quality at the time being is more uncertain.

As one can see by Figure 4 legacy systems with low business value and of good technical quality should probably be left alone, while systems with high business value and low technical quality are good reengineering candidates.

[8] **XML** (**Extensible Markup Language**) is a set of rules for encoding documents electronically. It is defined in the XML 1.0 Specification produced by the W3C and several other related specifications

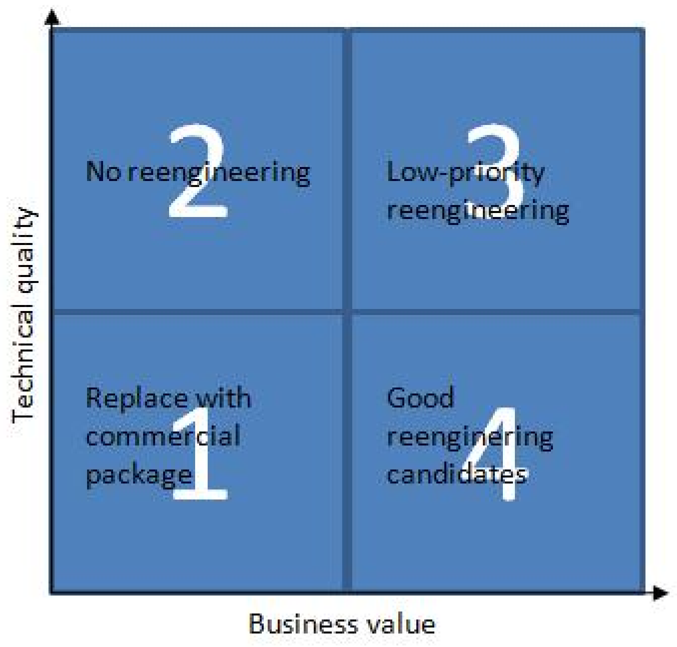

2
No reengineering
3
Low-priority reengineering
1
Replace with commercial package
4
Good reenginering candidates
Technical quality
Business value

**Figure 4 Portfolio analysis graph**

### 2.3.2 Cold Turkey versus Chicken Little

When starting a modernization project, the debate will also contain an important discussion about which approach to take. There are two main and opposite strategies that should be evaluated as possible paths to completion. The first is called "Cold Turkey" which means to rewrite the whole application from scratch, while the other method is called the "Chicken Little" approach.

The Cold Turkey approach is not recommended for several reasons, even though it can be tempting to start from scratch with clean sheets, one should consider the risks for failure listed in table 1 before one starts [4].

**Table 1 Modernizing risks**

| **Reason** | **Comment** |
|---|---|
| A better system must be promised | To get the budget, one cannot promise only to deliver the same system in a new wrapping. Additional requirements will be raised. |
| Business conditions never stand still | The legacy system will evolve in the time period it takes to develop a new system. This will lead to ongoing changes in requirement. |
| Specification rarely exist | Documentation of the old system is seldom of much use. New specification is tiresome to write. The specification is often the old system plus enhancements. |
| Undocumented dependencies frequently exists | The total overview of the legacy system is seldom present and unknown dependencies are likely to turn up. |
| Legacy systems can be too big to cut over data | Converting waste amount of live data in different formats and structure within in time limit the business can be without such, is a critical requirement that are hard to fulfill. |
| Management of large projects is hard | Brooks effect [5] central theme is that "adding manpower to a late software project makes it later" |
| Lateness is seldom tolerated | Due to the problems mentioned, lateness is a risk but not an acceptable one. |
| Large projects tends to bloat | |
| Homeostatis is prevalant | Fear of change and new technology can reduce the amount of cooperation. |
| Analysis paralysis sets in | One can ot start before one understand everything, which can lead to everlasting analysis. |

The Chicken Little approach is the recommended approach by professionals for most reengineering projects. The Chicken Little strategy is to migrate a legacy system in small incremental steps at the time. This will reduce the risk of the projects since success can be measured earlier, and the need for resources can be limited, especially in the starting phase of the project. One of the major advantages one will gain is the possibility to set these small incremental steps into production along the way of the project. In this matter the control is significantly increased compared to the Cold Turkey approach.

However, the Cold Turkey method might be the only feasible in some situations. If there exists no way of running the old and new technology together, or if this is very resource demanding to accomplish, the Cold Turkey approach could be the better approach.

### 2.3.3 TECHNOLOGY SELECTION

To do an evaluation of existing technologies, like operating systems, development languages and so on is outside the scope of this thesis. However the thesis will suggest some platform selection based on the case-studies needs.

# 3 Requirements

Since the stated goal of the case studies is to modernize and open up the legacy systems to the outer world, the requirements for the modernization approach will be much about SOA and SaaS. For the legacy business case there is also a wish to change the development language in which the software is written into another development language all together. To do this a full migration into another language must be carried out, and refactoring of the software structure into a more modern design is needed.

Therefore a main goal of this thesis is to describe, give guidelines and some contributions to a generic and standardized way to modernize legacy software into a service oriented and SaaS oriented design. To do this requirements are necessary to define which properties that must be considered when choosing a path to modernization. Also important for the thesis and modernization approach is to consider the requirement SOA and SaaS brings into the picture.

The requirements are grouped after four basic steps that have to be carried out during a normal modernization project. The basic steps involved are;

*1)* ***Recover and understand legacy architecture:*** The legacy architecture should be recovered and analysis should be possible to be performed on the recovered architecture.
*2)* ***Migrate towards SOA and :*** The legacy architecture should be refactored towards the target architecture, using analysis and refactoring tools.
*3)* ***Service modeling and MDA:*** There should be possible to design services and to extract services from the target architecture.
*4)* ***Select target platform and deploy:*** The target platform is selected and the modernized system deployed.

Legacy scientific software consists of, well scientific source code, that are likely to be hard to understand for traditionally IT people. When converting such code to another language, one will risk that the mathematical functions does not work as they used to. There are not many misinterpretations needed before the new scientific software will perform differently than the old one, and these misinterpretations can be hard to find for IT-people not skilled in the science in question.

In addition legacy scientific software has often been build in languages like C[9] and/or Fortran which means it probably are highly effective when it comes to computing speed. And for scientific software that often relies on different sets of numerical methods which are iterative, computing speed is of high importance.

Therefore requirements to scientific legacy modernization will be about as little reengineering as possible, and more on how the code can be migrated towards a SOA/SaaS platform, using wrapping techniques. If the legacy code is working well and performing its tasks to a satisfactory level, a reengineering project could very well result in a lot of unnecessary work [3].

For the business application reengineering and modernization is key issues and SOA requirements are of high importance. For the business application to be successfully

[9] **C** is a general-purpose computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system.

modernized it needs to be opened up and restructured to utilize the possibilities that SOA gives.

## 3.1 REQUIREMENTS FOR RECOVERING AND UNDERSTANDING LEGACY STRUCTURE

To understand and recover legacy software one need to use and adapt analysis tools and if possible to talk to the resources that built the software or at least resources that are expert users. A traditional administrative legacy system will normally consist of so many lines of code that it would be impossible to handle the amount of data manually within reasonable time. For scientific software the source code might not be that numerous, but it could be much harder to understand due to the complexity of the functionalities.

Based on this, there are requirements to tools and methods on how one can gain an overview of a legacy system that is correct and analyzable.

### 3.1.1 RECOVER TO ARCHITECTURAL REPRESENTATION

There should be possible to generate an AST (Abstract Syntax Tree) of the legacy system's source code. Having such a representation of the legacy source code and artifacts is a good basis to start analyzing and building an architectural view of the legacy system.

Furthermore this AST should be UML compliant so it is possible to view it using standard UML tools, and to perform analysis on the structure.

### 3.1.2 ANALYSIS OF LEGACY ARCHITECTURE

There should be possible to extract information out of the recovered architecture, such as elements, and the relationship among them. Typical elements to consider could be elements that call other programs or procedures, elements that access a data source and global variables. In this way one can for example track all updates to a specific database record.

Analysis possibilities like performing static analysis to get overview over call sequences, control flow, cross references and variable usage should be available. Static analysis can be a great help for understanding the architecture of a system, but there are items such analysis cannot detect. Late binding cannot be detected in the source code, since the relationships have not been made yet.

Slicing is a technique first described by (Weiser 84). Slicing can be done both dynamically and statically and is performed on an object or parameter of interest. The technique can then look at only the statements that affect the selected element.

### 3.1.3 RETRIEVING NEW AND OLD REQUIREMENTS

The requirements of a legacy system are often only found by digging in the source code of the system. To not having to do this job, automatic discovery of business requirements are preferable. There should be possibilities to extract business processes and rules by intelligent discovery tools so these can be separated from the architecture and placed as BPMN[10] and BMM[11] for later control and verifications.

New requirements are likely to turn up. One of the reasons for modernizing is that there exists requirements to the application that are not realized due to the difficulty in maintaining and changing the legacy structure.

Retrieving requirements must be an early stage, and on-going  activity to ensure that the result is up to date and according to the stakeholders wishes.

## 3.2 REQUIREMENTS FOR MIGRATION TOWARDS SOA AND SAAS

SOA requirements and SaaS requirements is the same for this step in a modernization project. SaaS have additional requirements to SOA, but these requirements are best covered in the next step, requirement for service modeling.

SOA requirements are set by both the stakeholders for the case-studies and it is important to be aware of what this means. There exists quite a lot of literature on SOA and the requirements to a SOA solution, but there seems to be a common understanding of these requirements.

A modern service oriented solution has many characteristics built into it, according to Thomas Erl these requirements are;

"Contemporary SOA represents an open, agile, extensible, federated, composable architecture compromised of autonomous, QoS[12]-capable, vendor diverse, interoperable, discoverable and potentially reusable services, implemented as Web Services."[6]

In this section these terms will be explained and seen in perspective of the case studies. The figure below shows a typical SOA solution and some of its principles.

[10] **Business Process Modelling Notation** (BPMN) is a graphical representation for specifying business processes in a workflow.
[11] **Business Motivation Model** (BMM) in enterprise architecture provides a scheme and structure for developing, communicating, and managing business plans in an organized manner. http://www.businessrulesgroup.org/bmm.shtml
[12] QoS (Quality of Service)

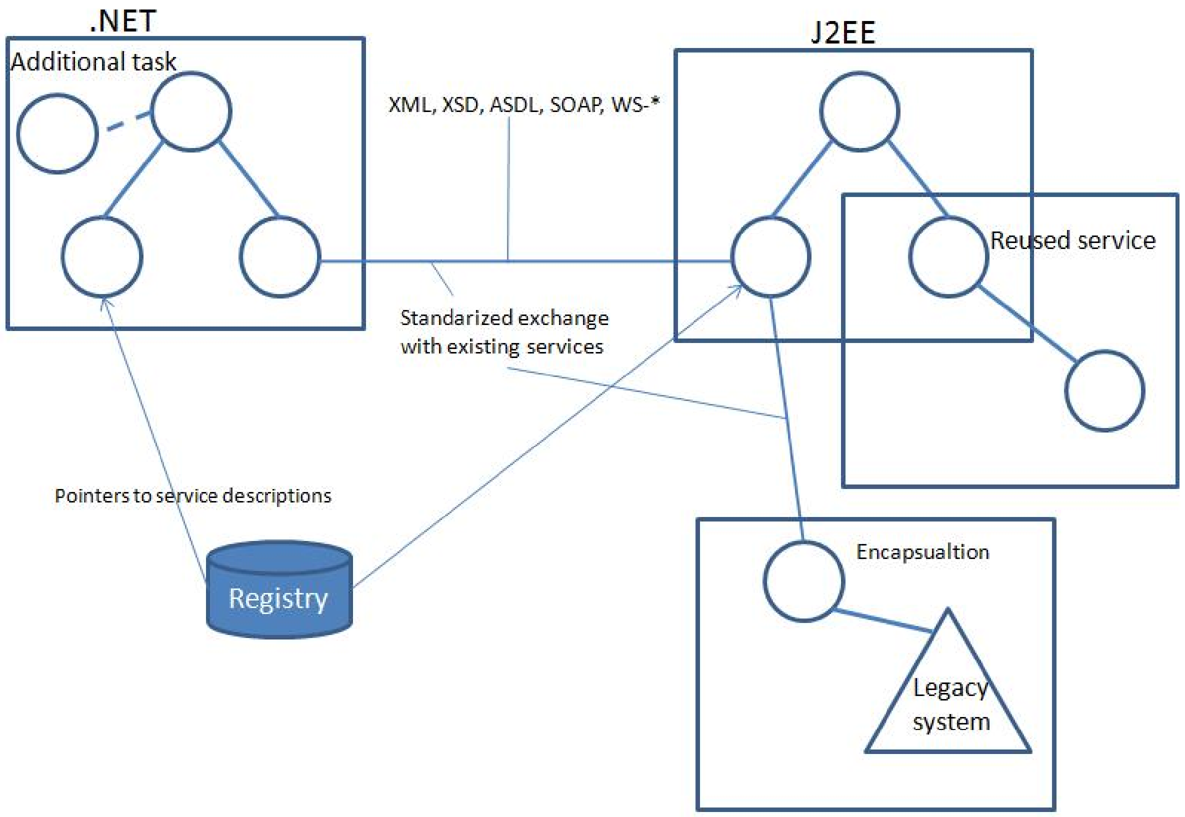


**Figure 5 A figurative image of SOA principles show**

1) ***Agility:*** When the system is structured in a SOA way, the system will be easier to change with respect to changing business needs. This can happen since one knows a SOA will provide many building blocks with a standardized and technology independent interface, instead of one large block of code that is hard to change.

   Especially the business case study will benefit from having the possibility to change the business processes and functionality in a much more rapid manner that before.

2) ***Extensibility:*** When the services are built properly, one should be able to extend the service's functionality without breaking the already established interface.

   This yields for both case studies, whether one need to change the OSCAR or the business system, it would be preferable to not change the interface in a way that will influence the way other users has implemented the interface.

3) ***Federation:*** By giving a possibility to encapsulate existing systems and functionality, and expose them as web services with following SOA, one can see how SOA promotes federation.

   This can be important in both case-studies. For the business case, it gives a possibility to not having to reengineer the whole application before the functionality within can be used by others. For the scientific study, an encapsulation can very well be the best and only solution to a modernization approach.

4) ***Reusability:*** The very nature of SOA promotes reusability, and having independent units of logic with a technology independent interface ensures reusability of the services.

   This is especially important for the business-case, and one of the base reasons to do a modernization of such. For the scientific case-study, when there exists many scientific services of finer grain this can give a huge impact on the solutions that an organization can offer.

5) ***Composability:*** Since services ideally exist as independent units, they can also easily be used as building blocks for business processes. With the second-generation web service specification, the WS-*[13], services will, become even more mature for composing services that only uses the needed features. The WS-* standard makes it possible to define in which manner one wants to interact with other services.

   Internal composability will be important for both case-studies, but at first it might not be clear whether the WS-* will be important for the case-studies. However, when services are to be composed with services outside the control of the organizations, the importance of being able to control the interaction on a more fine grained level can be imperative for the solutions.

6) ***Autonomous:*** This requires that services are as independent as possible with respect to the underlying logic. In addition it requires that the messages sent between the different services are so self-contained and intelligent that they can control the way they are processed by the receiving services.

   This is features that are desirable for both case-studies. Even though this might not be among the first demands, it will be important as the systems grow and start interacting more with external services.

7) ***QoS:*** Qos (Quality of Service) is ensured by SOA, since SOA gives the ability to secure and protect the message exchange, and gives means for reliable and transactional messaging. All this will heighten the QoS.

   Everybody would like high QoS, so this is absolutely something that will gain both case-studies.

8) ***Vendor diversity:*** Since SOA is based on open standards when it comes to communication and message exchange, it does not matter what kind of technology that is used as long as it supports the creation of standard web services.

   Especially important for the business case-study, since it is very likely this has to interact with a variety of different applications.

9) ***Interoperable:*** Interoperability is covered by other SOA requirements such as open standards, discoverability and being vendor diverse.

[13] WS-*, a common term for Web Service specifications that starts with “WS-“.

Interoperability is important for both cases for the same reasons that make the services interoperable.

What do these requirements mean for the migration process? The foundation for these requirements must be set in this step. Badly designed code will be hard to extract services from, and the services will not be agile, extensible, federated, and autonomous when they are tangled in badly designed code. The services will probably not fulfill the QoS or the reusability requirements either.

The basis for these requirements is depended on good OO (Object Oriented) design. In OO design there are two factors that stand out as important to meet such criteria and those are high-cohesiveness and loose-coupling.

Cohesion is a measure that can tell us how cohesive or focused software components are on the responsibilities it has. In other words it can with some certainty say if the software module does the job it is supposed and only that. The higher cohesion one has the better. In OO design one can say that a cohesive class performs one function. A non-cohesive class performs two or more unrelated functions. A non-cohesive class may need to be restructured into two or more smaller classes.

Coupling correlates often with cohesion and tells us how tightly a component is coupled with another. With high cohesion there are usually low-coupling, and low-coupling is desirable for our most purposes.

These two factors are the most important factors when redesigning with SOA in mind. SOA principles tell us that autonomous, re-usable and maintainable components are best suited in a SOA structure, and these factors are an indication for exactly that.

### 3.2.1 Gradually refactor code towards SOA and SaaS

When migrating old systems it is a necessity to structure the new system according to modern IT-architecture. A fundamental structure along the lines, shown in Figure 6 should be a base requirement, and is fundamental to make the new system ready for further evolution and change management.

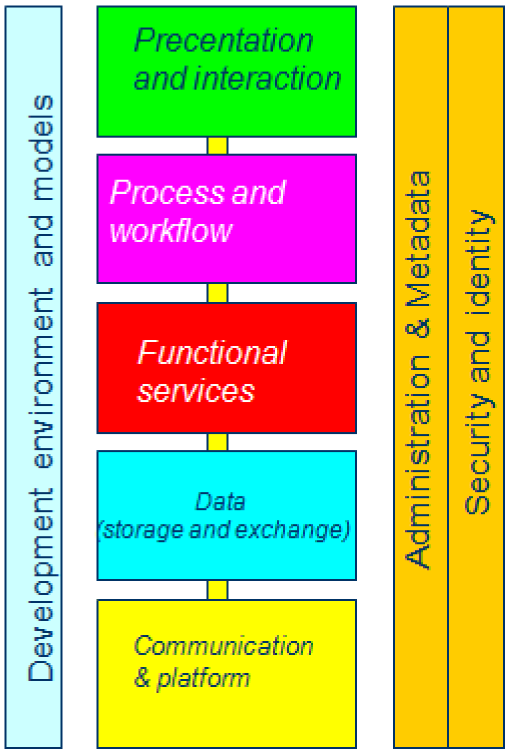

Development environment and models
Precentation and interaction
Process and workflow
Functional services
Data (storage and exchange)
Communication & platform
Administration & Metadata
Security and identity

**Figure 6 Common reference architecture**

As the figure shows, the architecture of a modern system should be build with different layers that are naturally separated by the tasks they represent. The migration tools should have build in help for detecting the different layers and to help creating a sound target architecture that follows the lines of the reference architecture.

### 3.2.2 Control your work

After a piece of the legacy code has been migrated, analysis should be made on the target architecture to verify that a sound design has been achieved. The migration tools should contain means to analyze especially with respect to cohesion and coupling.

### 3.2.3 Traceability

To help gaining an overview over the migrated code compared to the legacy, tracing capabilities should be built into the migration tool so that one can trace elements in the migrated architecture back to the legacy architecture. This will help ensuring that the old system is covered in the new and that pieces of code are not forgotten.

## 3.3 Requirements for service modeling and MDA

With service modeling we mean extended functionality compared to standard UML based modeling tools. There should be an explicit way of modeling services so that it will be possible to generate run able source code from the model.

SaaS will add some requirements to this section. A major change in structure might be necessary if the application is to run as SaaS either in a private network, a hosting company or the cloud. The most important requirement to fulfill is to make sure the application is multi-tenant.

### 3.3.1 SOA AND MDA REQUIREMENTS

The tools for SOA modeling and MDA must be platform and technology independent (PIM), and the tools should be able to allow for service modeling following the SoaML specification [7].

The MDA tool must give possibilities for specializing and adapting the transformation logic from PIM to PSM. This is necessary if the generated PSM are to inherit special model designs that the modernization project address.

### 3.3.2 SAAS REQUIREMENTS

SaaS introduces a couple of additional requirements to modeling. The SOA requirements will also apply for SaaS, but SaaS and Cloud Computing has another set of demands that are not that easily seen as automated and tool helped processes. While the multi-tenancy and security issues are something that can be handled by tools to come, the other requirements are more difficult to automate.

#### *3.3.2.1 SaaS and Multi-Tenancy*

If the goal of the modernization project is to serve as many clients as possible and outsource the maintenance and performance of the system to a SaaS/Cloud computing provider the issue of multi-tenancy arise to a priority. Since the price models for those hosting such platform are based on application instances as well as hardware resources, the importance of making one's software multi-tenant becomes the key issue.

An application that is multi-tenant can handle multiple clients' users (tenants) in the same application instance. This means that the application itself only needs to run as one instance on the hosting platform, and not as one instance per client. This will save costs for the hosting companies since fewer resources are demanded, and thus save money for the customers since they then would get lower price for the hosting.

The idea of multi-tenancy is however not new, systems that can host several clients have been built before, and there probably exist a lot of them used through the internet or at hosting companies.

For the Progress CRM system this is not relevant for the base system, there are however sub-modules or interfaces build for clients to use over the internet, and some client information already resides in the system. Multi-tenancy is at least something that must be evaluated and probably taken to the design table in an early phase.

For the scientific application multi-tenancy is a central point since the goal already has been stated to make the application ready for SaaS and Cloud Computing.

When implementing multi-tenancy several issues should be considered. According to (5) the issues are;

#### *3.3.2.2 Other SaaS requirements*

There exists several SaaS requirements, but they will not be covered in debt under the scope of this thesis. Under follows a short list with the most common ones.

*Performance isolation and scalability:* Prevents tenants to influence the performance for other tenants. Make sure that if one tenant use, or try to use, a lot of resources, this should not affect other tenants.

This seems to most valuable for the scientific application. The OSCAR system is basically about doing calculations as fast as possible, and the processing time is a crucial factor for success.

*Availability isolation:* Since so many possible users will be affected if the system goes down, it is important to detect and handle faults as early and precise as possible.

This is relevant for both the OSCAR and the k90 system.

*Administration isolation:* Any administrative task must be isolated between the different clients, so any modifications on operational level do not affect any other client.

This is relevant for both the OSCAR and the K90 system.

*On-the-fly customization:* Customizations that are needed on the system for the individual clients, needs to be flexible, isolated, easy to perform and possible during up-time of the systems.

This is relevant for both the OSCAR and the K90 system.

### 3.3.3 SERVICE MEDIATION

Having opened up the application with services, combining these services internally or with externally available services, gives possibilities for composing services with new and enhanced functionalities. However one will in most cases find that the different services are not totally interoperable. One service can be expecting a request in one format while the requesting service is using a different format. To solve this problem a way to convert one message format to another would be helpful for the service composer.

## 3.4 REQUIREMENTS FOR THE TARGET PLATFORM REALIZATION

The business application target platform must support SOA and a multi-tier architecture, while the legacy scientific application also must support SOA it must support SaaS in addition. To support these requirements there must be a way to produce a PSM (Platform Specific Model) that supports open standards, web services, layered structure, and general application storage and data management.

With open standards the emphasis is on the exchange of data, and how this data is governed. When web services communicate today, they should follow a common set of open standards based on SOAP[14], WSDL[15], XML[16] and XSD[17]. SOAP gives means to standardize the way

[14] **SOAP**, originally defined as *Simple Object Access Protocol*, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks.

messages are packaged before sending, XML is the standard way messages is formatted and XSD defines the possible data formats and message content. The WSDL defines how one can access the different methods that the service exposes. All four standards are based on XML which ensures global understanding of the documents.

For both the case studies this principle comes to play. For the OSCAR case this will be the way users of OSCAR has to communicate with the web service that will represent OSCAR. For the business case all services exposed from the system should have this as the message exchange standard to ensure that eventually users of the services meets a familiar and standardized data exchange method.

Web services are the mean to implement a SOA design, and there must be support for generating web services from the PIMs. The generation of web services should also try to be as platform independent as possible to make sure that a switch of platform does not result in a lot of rewriting of neither target source code nor the transformations that generated the source code in the first place.

The applications storage and data management should be made general, and fit into existing data management systems to reduce the diversity of maintenance. Legacy software is not unknown to use proprietary storage, often as text files or self composed database systems. These should be converted to a modern standard.

[15] WSDL (**Web Services Description Language**) is an XML-based language that provides a model for describing Web services.

[16] **XML** (**Extensible Markup Language**) is a set of rules for encoding documents electronically. It is defined in the XML 1.0 Specification produced by the W3C

[17] XSD (XML Schema Definition) An **XML schema** is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type, above and beyond the basic syntactical constraints imposed by XML itself. An XML schema provides a view of the document type at a relatively high level of abstraction.

# 4 EVALUATION OF EXISTING SOLUTIONS

In this chapter some methodologies, applications and tools relevant for legacy software modernization are selected for an evaluation. To evaluate every application and tool in the market would be a too huge job for this thesis, thus only some of the leading solutions and emerging technologies are selected.

To evaluate existing solutions the following criteria are used derived from the requirements section.

- Recovering
  - Recover to architectural representation
  - Analysis of legacy architecture
  - Retrieving requirements
- Migration
  - Refactoring for SOA
  - Refactoring for SaaS
  - Control your work (Analysis)
  - Traceability between models
- Service modeling (PIM-PSM)
  - Modeling for SOA and MDA
  - Modeling for SaaS
  - Modeling service mediation
- Platform (PSM-Realization)
  - Transform to SOA
  - Transform to SaaS

The evaluation is based on whether the solution covers the criteria or not. The symbol √ is used to mark the covered criteria, while no marking means that he criteria is not covered. The last section in this chapter contains an overall evaluation table for the evaluated solutions.

## 4.1 ADM (ARCHITECTURE DRIVEN MODERNIZATION)

When evaluating a strategy for a modernization approach, one should consider the use of ADM[18] (Architecture Driven Modernization) to see if there are applications and methods that are mature enough to be a help in modernizing the software.

OMG [19] (Object Management Group) are in the middle of developing standards for how to modernize legacy software, and their contribution should be considered before starting a modernization approach. OMG has a long record of standardization work, and if a standard is accepted, one can be ensured that there will be developed tools that support the standard, and thus ensuring continuance in the development of tools that can help a modernization project.

The new sets of standards are developed under the ADM task force and are promising a new set of standards and tools for modernization purposes.

[18] See http://adm.omg.org/ for information about ADM.
[19] See http://www.omg.org/gettingstarted/gettingstartedindex.htm for information about OMG.

The ADM task force has as its goal to find solutions for many of the challenges regarding modernization of legacy systems. The ADM task force focuses on formal transformation of existing systems into models and back into new systems, following their horseshoe model [8]. The horseshoe model basically defines three levels of transformation, the technical, architectural and business level as shown in Figure 7.

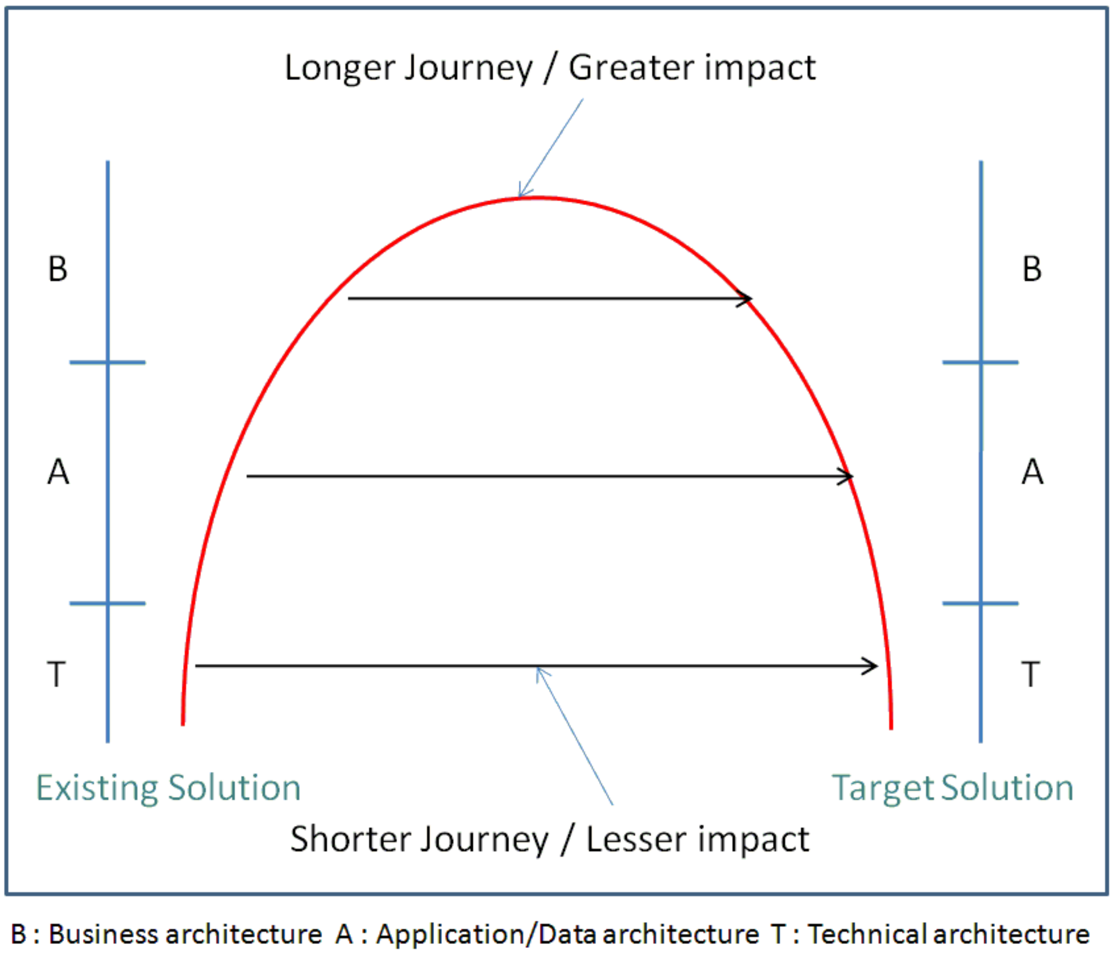

Longer Journey / Greater impact
B
A
T
B
A
T
Existing Solution
Target Solution
Shorter Journey / Lesser impact
B : Business architecture A : Application/Data architecture T : Technical architecture

**Figure 7 The ADM Horseshoe Model**

To every transformational path there are three elements, knowledge discovery, target architecture definition and transformative steps. The horseshoe model represents a variety of possible modernization scenarios and it is up to organization to figure out their best path. However the ADM recommends that the horseshoe is followed to ensure that the new system will consists of a correct representation of the organizations business processes.

By only doing a technical transformation from one programming language to another, one will still have the same un-evolvable application as before, even if the target language is more modern.

By doing an architecture and technical transformation, which is most common, one can gain a lot, but it gives no guarantee that the new system follows the organizations business needs.

Unfortunately there are not defined standards for mappings between all these types of models, especially between business and IT standards are lacking. The OMG task forces and working groups are looking into this, and have made a roadmap that states their goal in building supporting tools and definitions.

Their roadmap states the following program;

- ADM: Knowledge Discovery Meta-Model (KDM)
- Allows for exchange of application meta-data across systems. Ready in 2006.
- ADM: Abstract Syntax Tree Meta-Model (ASTM). Allows the KDM to fully represent applications and facilitate the exchange of granular meta-data across multiple languages. Ready in 2009.
- ADM: Pattern Recognition
- Facilitates the examination of structural meta-data with the intent of deriving patterns and anti-patterns about existing systems. Issued in 2009.
- ADM: Structured Metrics Package (SMM)
- The focus of the Software Metrics Package is to derive metrics from the KDM that can describe various system attributes. Roll out in 2009.
- ADM: Visualization
- Depict application meta-data stored within the KDM. No target date.
- ADM: Refactoring. Defines ways in which the KDM can be used to refactor applications. No target date.
- ADM: Transformation. Defines mappings between the KDM and ASTM and target models. No target date.

#### *4.1.1.1 KDM (Knowledge Discovery Meta-model)*

According to OMG, the KDM specification [9] defines a meta-model that can represent existing software assets, associations and operational environments.

The idea is that, whilst it exists a number of tools that can in their own proprietary ways analyze and/or extract information from existing software they do not share a common information-structure. This is a hinder for interoperability between the tools, and makes the total task of reengineering software more difficult than it could have been.

The main purpose of the KDM is to provide a common interchange format that the different tool vendors can use and implement, thus allow for interoperability between the same tools.

The KDM is divided into several packages to group different information into a more comprehensive way. There are a total of 12 packages, each package briefly explained in table 2 below.

**Table 2 The different packages of KDM.**

| **KDM elements** | **Description** |
|---|---|
| Infrastructure layer | Defines the core concepts used throughout the entire KDM specification. |
| Core package | Defines the fundamental meta-model elements |
| KDM package | Defines the infrastructure of the KDM instances. Organized in segments with specific models. |
| Source package | Inventory model, representing artifacts of the system, such as source code files. |
| Program elements layer | Provides the building blocks to represent existing software language-independent. |
| Code package | Meta-model elements representing program elements and their relations. |
| Action package | Meta-model elements representing program elements and their behavior. |
| Runtime resource layer | Represents the operating environment for existing software. |
| Platform package | Defines artifacts related to the run-time platform of the application. |
| UI Package | Meta-model elements representing the user interface of existing systems. |
| Event Package | Meta-model elements representing high-level behavior of applications, such as event-driven state transitions. |
| Data Package | Meta-model elements representing organization of data in the existing software system. |
| Abstraction layer | |
| Structure Package | Defines constructs defining the organization of software, such as subsystems, modules and architectural views. |
| Conceptual Package | Enables mapping of KDM compliant models to other models. Currently SBVR[20] |
| Build Package | Meta-model elements representing the engineering view of a system, and artifacts generated by the build process. |

[20] **Semantics of Business Vocabulary and Business Rules** (SBVR) is an adopted standard of the Object Management Group (OMG) intended to be the basis for formal and detailed natural language declarative description of a complex entity

#### *4.1.1.2 AST (Abstract Syntax Tree)*

AST (Abstract Syntax Tree) is an abstract tree view of source code written in a specific language. With abstract means that all the details in the concrete syntax are not shown. The concept AST has been around for a while, and used as help in understanding and improving source code. Compilers, converters and transformation tools represents programming language as AST's. Eclipse[21] uses an AST in their Java[22] development tools to provide help to users with for instance showing references to a field or method in the source code. Eclipse also has a plug-in that can view the AST and perform additional functionalities.

An AST is a tree representation of the source code, where every node in the tree represents some code element, such as assignments, method declarations or variable declarations. AST model structure allows for relating different language constructs and gives means for expressing properties of the same constructs.

#### *4.1.1.3ASTM (Abstract Syntax Tree Meta-Model)*

The ASTM seeks to establish a single comprehensive set of modeling elements for capturing how many software languages represent the same software language constructs and concepts.

The ASTM (Abstract Syntax Tree Meta-Model) defines a meta-model for representing information about existing software in the form of an AST. The meta-model describes the elements to be used in an AST to model the software in a formal structure. The AST does not held language specific form of expression.

The ASTM specification [10] defines three domain specific meta-models;

- Generic Abstract Syntax Tree Metamodel A generic set of language modeling elements common across numerous languages establishes a common core for language modeling, called the Generic Abstract Syntax Trees (GAST). In this standard the GASTM model elements are expressed as UML class diagrams.
- Language Specific Abstract Syntax Tree Metamodels (SAST) for particular languages such as Ada, C, Fortran, Java, etc are modeled in MOF or MOF compatible forms and expressed as the GASTM along with modeling elment extensions sufficient to capture the language.
- Proprietary Abstract Syntax Tree Metamodels (PAST) express ASTs for languages such as Ada, C, COBOL, etc modeled in formats that are not consistent with MOF, the GASTM, or SASTM. For such proprietary AST the ASTM specification defines the minimum conformance standards needed to support model interchange.

GAST can be generated by transforming a SAST or UML class diagrams or by generating a GAST directly if desirable.

By establishing one common set of modeling elements, as ASTM to represent a language, one will hopefully be given a much richer set of modernization tools. If all vendors develop their tools following the ASTM standard, the tools can be used together, working on the same set of data.

[21] **Eclipse** is a multi-language software development environment comprising an IDE and a plug-in system to extend it.
[22] **Java** is a programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform.

Another benefit gained, is that this will give possibilities for applying standard transformation rules from one language architecture to another, since the same syntax tree meta-models are used in all occasions.

**Table 3 ADM evaluation**

| **Area** | **Coverage** |
|---|---|
| **Recovering** | Yes. |
| Recover to architectural representation | Recovering to KDM and AST. |
| Analysis of legacy architecture | Planned |
| Retrieving requirements | No |
| **Migration** | Planned |
| Refactoring for SOA | Unkown (Planned refactoring and transformation from legacy to target architecture) |
| Refactoring for SaaS | Unkown (Planned refactoring and transformation from legacy to target architecture) |
| Control your work (Analysis) | Planned |
| Traceability between models | No. |
| **Service modeling (PIM-PSM)** | No |
| Modeling for SOA and MDA | No |
| Modeling for SaaS | No |
| Modeling service mediation | No |
| **Platform (PSM-Realization)** | No |
| SOA and SaaS | No |
| Application storage and data management | No |

## 4.2 MDA TOOLS/METHODS

Model Driven Architecture (MDA) and Development (MDD) has become more and more accepted in the IT community, thanks to extensive work done by several contributors over the past decades. In addition to MDA a fairly new concept the ADM has become more relevant lately, thanks to among others the Object Management Group.

Whilst the model driven approach is useful in a lot of settings, it is especially suited for the purpose of servicing scientific legacy systems. The three primary goals of MDA are portability, interoperability and reusability. A fourth and important goal is also to make the systems understandable and discussable for all concerned participants, ranging from IT architects, business people, scientist and maintainers. By applying a MDA approach one will use one common and easily readable language to model the software, and thus give the possibility to ensure that the software is not understood only by a few experts.

This thesis will look into the possibilities within in the MDA/MDD approach for making scientific legacy systems as services.

### 4.2.1 MOMOCS

MOMOCS[23] (Model driven Modernization of Complex Systems) is a European sponsored project that aims a methodology for reengineering of complex systems. MOMOCS focus on reengineering architecture, processes and data heterogeneities to make systems more predictable in terms of performance, stability and upgradability.

MOMOCS focus is not towards software as services or Cloud Computing,

MOMOCS as many other starts by extracting the legacy source code into a representation of the same code that then can be modified using modeling tools. Unfortunately for this thesis the legacy source code that can be reengineered doesn’t include Fortran nor Progress 4GL, which makes it hard to test and evaluate the outcome of the projects in light of the thesis case studies.

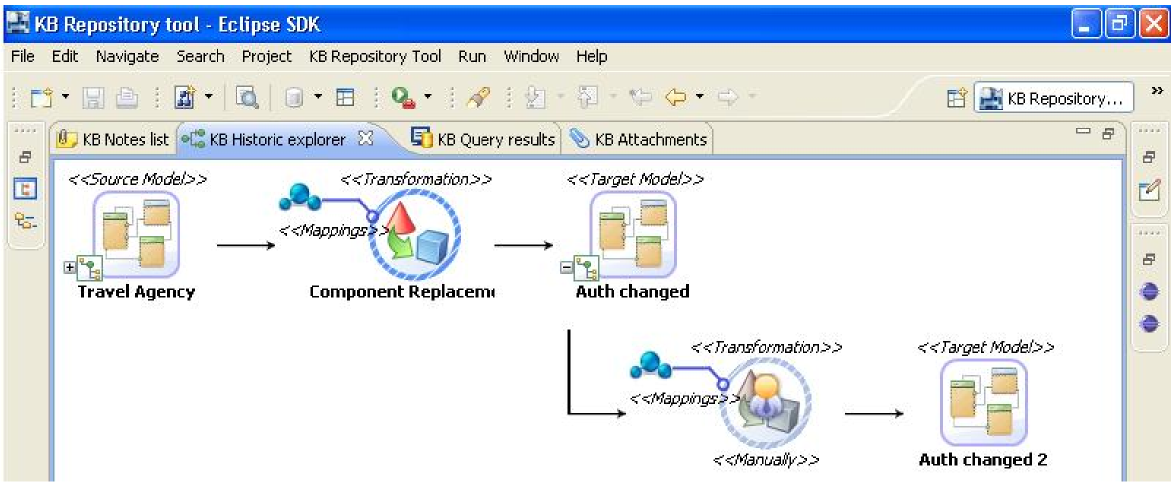


**Figure 8 MOMOCS showing transformation evolution**

The figure above visualizes the evolutionary modernization approach. By storing every

[23] MOMOCS for details see http://www.momocs.org/

modernization step applied manually or by transformation MOMOCS can keep track of and roll back the process if needed.

**Table 4 MOMOCS evaluation**

| Area | Coverage |
|---|---|
| **Recovering** | Yes. |
| Recover to architectural representation | Recovering to KDM. |
| Analysis of legacy architecture | No (Search functionality on the KDM) |
| Retrieving requirements | No |
| **Migration** | Yes |
| Refactoring for SOA | No (Can apply transformations on models) |
| Refactoring for SaaS | No(Can apply transformations on models) |
| Control your work (Analysis) | Can save modernization steps along the way in a repository. |
| Traceability between models | Do have repository for storing of artifacts, and ontology to support reuse of components. |
| **Service modeling (PIM-PSM)** | No |
| Modeling for SOA and MDA | No |
| Modeling for SaaS | No |
| Modeling service mediation | No |
| **Platform (PSM-Realization)** | No |
| SOA and SaaS | No |
| Application storage and data management | No |

### 4.2.2 MODELPRO

ModelDriven.org, a community of government, commercial and university members, has developed a method for generating executables from SoaML designs. Their solution ModelPro can in combination with SoaML and their provisioning cartridges, a plug-in for ModelPro, generate Java source code that can be executed under a JEE5[24] platform.

This means that by applying stereotypes on modeling artifacts ModelPro will generate, compile and execute the services designed using modelPro and SoaML.

To be able to run service models directly from the design board is a great advantages for the modeling community, and will hopefully drive the modeling world further towards making modeling the base language for development.

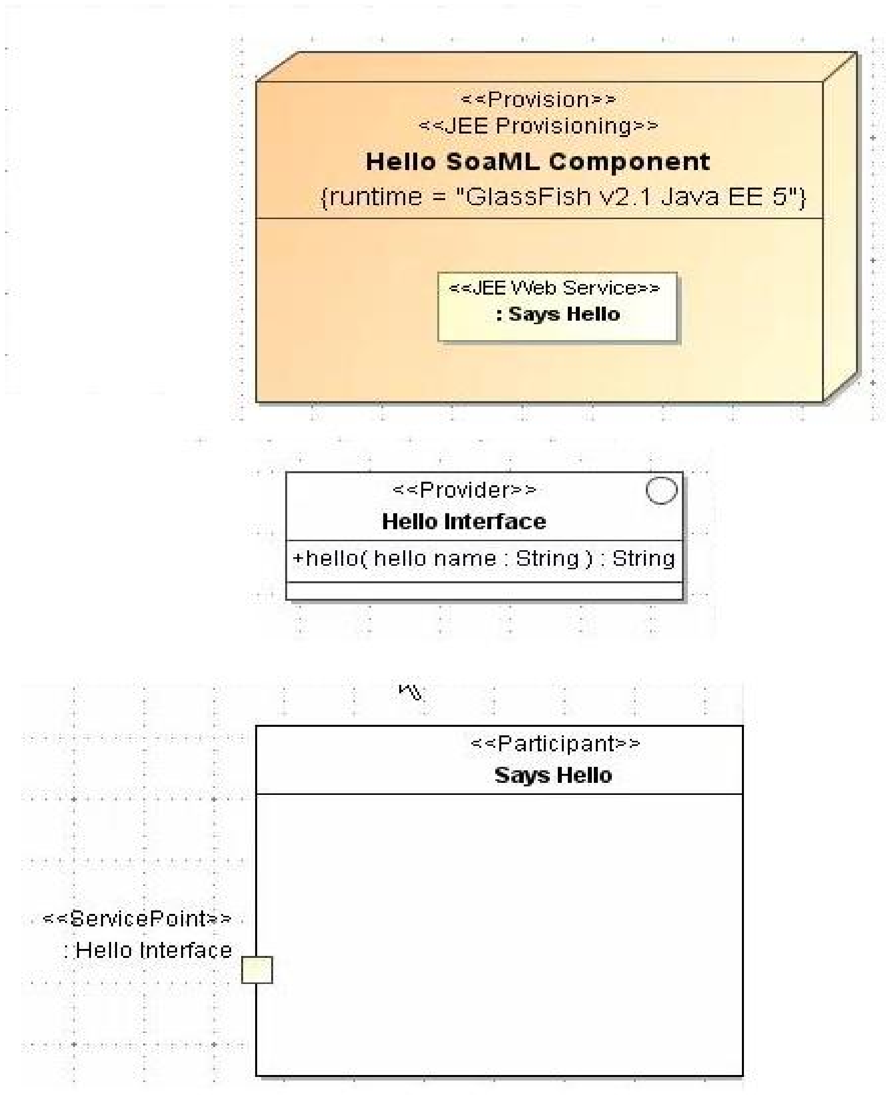


**Figure 9 Images of deployment component from ModelPro's demonstration video**

[24] Java Platform, Enterprise Edition or Java EE is a widely used platform for server programming in the Java programming language. See for more details http://java.sun.com/javaee/technologies/javaee5.jsp

As Figure 9 shows, only three diagrams are necessary to make an demonstration of a simple “Hello world” web service. The first diagram is the SoaML ”Provider” stereotyped interface, which will be used as specification for the service point at the ”participant” in the second image in the figure. The last image shows the provisioning diagram where a deployment element with stereotype ”Provision” is used to hold the information needed to generate and load the web service. As one can see, the element is also stereotyped ”JEE provisioning” to tell that the deployment is to be a JEE[25] deployment. Tag value “runtime” selects the Glassfish[26] as application server. At last an instance of the web service is put into the deployment element and stereotyped as a JEE web service. Select to execute the system and all is up and running.

However the project is still young and it is unknown whether this will work properly for larger and more complex systems than the demonstration packages supplied on their home pages.

**Table 5 ModelPro evaluation**

| **Area** | **Coverage** |
|---|---|
| **Recovering** | No |
| Recover to architectural representation | No |
| Analysis of legacy architecture | No |
| Retrieving requirements | No |
| **Migration** | No |
| Refactoring for SOA | No |
| Refactoring for SaaS | No |
| Control your work (Analysis) | No |
| Traceability between models | Unknown |
| **Service modeling (PIM-PSM)** | Yes |
| Modeling for SOA and MDA | Yes (SoaML standard implemented) |
| Modeling for SaaS | No |
| Modeling service mediation | No |

[25] JEE (Java Platform, Enterprise Edition) is a widely used platform for server programming in the Java programming language.See http://java.sun.com/javaee/technologies/javaee5.jsp for an overview of JEE5.
[26] *GlassFish* is an open source application server project led by Sun Microsystems for the Java EE platform. See for details. https://glassfish.dev.java.net/

| **Platform (PSM-Realization)** | Yes |
|---|---|
| Transform to SOA | Yes. Have good MDA transformation capabilities towards Java and J2EE, with SOA in mind. |
| Transform to SaaS | No |

### 4.2.3 UML/ SOAML

#### *4.2.3.1 UML and SoaML*

The recent SoaML (Service oriented architecture Modeling Language) specification [7] describes an UML profile[27] and metamodel[28] for design of services in a service oriented architecture.

"The goals of SoaML are to support the activities of service modeling and design and to fit into an overall model-driven development approach."[11]

The specification tries to cover several aspects of service modeling such as the specification of systems of services, the specification of individual service interfaces, and the specification of service implementations. This is done by introducing a set of terms available as stereotypes in the profile, the stereotypes includes "Service", "ServiceInterface", "Participant", "ServiceContract", "Specification" and several more. By specifying a metamodel for services as well, the use of the terms is defined and made distinct, which again will ease the task for those wanting to use the specification with MDA for automatic generation of artifacts.

This thesis will look into these modeling perspectives and see if we can build a model of the legacy system that is capable of generating the necessary information needed to build a running system.

[27] A **profile** in the Unified Modeling Language (UML) provides a generic extension mechanism for customizing UML models for particular domains and platforms

[28] A meta-model typically defines the language and processes from which to form a model.

**Table 6 Evaluation of SoaML**

| Area | UML/SoaML Coverage |
|---|---|
| **Recovering** | No |
| Recover to architectural representation | No |
| Analysis of legacy architecture | No |
| Retrieving requirements | No |
| **Migration** | No |
| Refactoring for SOA | No |
| Refactoring for SaaS | No |
| Control your work (Analysis) | No |
| Traceability between models | Partly |
| **Service modeling (PIM-PSM)** | Yes |
| Modeling for SOA and MDA | Yes |
| Modeling for SaaS | Partly |
| Modeling service mediation | Partly |
| **Platform (PSM-Realization)** | No |
| Transform to SOA | No |
| Transform to SaaS | No |

## 4.3 TRANSFORMATIONS

An important part of an ADM/MDA approach when modernizing legacy system is to have transformation tools at hand that easily can transform models to models or models to text. When modernizing one would like to be able to generate target architecture and target code without having to do a lot of manual writing. First of all it is a tiresome process to rewrite a large system by hand, and secondly it will introduce possible errors due to human factors.

To accommodate for this automation of code from models is an important part. A good transformation language and proper transformation ensures that the code or models generated will be consistent with the desired goal. To write hundreds of thousands lines of code manually will introduce a lot of errors. Studies have shown that there can be as many as one error per tenth line of code with this approach. Automated transformations can reduce this to a minimum. In addition automated transformations make it easier to keep the original architecture ideas down to code level.

### 4.3.1 MOFSCRIPT

The MOFScript[29] language s developed at SINTEF in conjunction with the European project MODELWARE[30] and is a subproject of Eclipse.

MOFScript is a tool for model to text transformation, which means it is suited for retrieving information from models and convert this information into a textual representation. Basically from UML models to source code. The latest version of MOFScript also supports model to model transformations.

The MOFScript is metamodel[31] independent that allows to use any kind of metamodel and its instances for text generation. This is making the transformation language flexible and independent. MOFScript is also based on EMF[32] and Ecore as metamodel framework, and are deployed as a plug-in for the popular Eclipse platform.

To support text generation MOFScript comes with at least three additional libraries, a string library, a collection library and a XML library. The first includes functions for string manipulation such as variations over getting a substring of a text element. The collection library support abstract data types in form of lists and hash tables. The last supports generation of XML documents with functionalities like adding elements and attributes.

### 4.3.2 ATL

ATL[33] (ATLAS Transformation Language) is a model transformation language and toolkit. ATL is specialized in model to model transformation, and suitable as a helper tool in

[29] MOFScript is a tool for model to text transformation, for details see http://www.eclipse.org/gmt/mofscript/
[30] MODELWARE for details see http://www.modelware-ist.org/
[31] Metamodeling is the analysis, construction and development of the frames, rules, constraints, models and theories applicable and useful for modeling a predefined class of problems
[32] **Eclipse Modeling Framework** (EMF) is an Eclipse-based modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI
[33] ATLAS for details see http://www.eclipse.org/m2m/atl/

migrating modelled architectures. ATL is developed by the French national research institute INRIA[34].

As for MOFScript. ATL is based on EMF and Ecore as metamodel framework, and are deployed as a plug-in to the Eclipse platform.

Rules are the central part of the language, which links the source model with the target model by defining how a source element are to be transformed into a target element. Helpers are function like and can be used to performed tasks that cannot be performed by the rules. As input to and output from the transformation ATL requires models serialized as XMI[35] files..

### 4.3.3 QVT

QVT (Query, Views, Transformations) is a MDA standard and part of the MDA framework. QVT consists of three parts;

- *Query:* Query is as it states a mean to query the model. By querying the model, much like querying a database, one will receive the result set of model elements that match the query.
- *View:* Views are similar to views in SQL[36] in the way it will reflect only a subset of the whole source model.
- *Transformations:* Transformations is a standard to create a language for writing transformation definitions. The purpose of the transformation is to generate a target model based on the source model. The transformation can be depended, which means that relationships between source and target models are kept.

[34] INRIA for details see http://www.inria.fr/

[35] The **XML Metadata Interchange** (**XMI**) is an Object Management Group (OMG) standard for exchanging metadata information via Extensible Markup Language (XML).

[36] **SQL** (**Structured Query Language**) is a database computer language designed for managing data in relational database management systems (RDBMS), and originally based upon Relational Algebra.

### 4.3.4 Evaluation of ATL, QVT and MOFScript

**Table 7 MOFScript, QVT and ATL evaluation**

| **Area** | **MOFScript Coverage** | **ATL Coverage** | **QVT Coverage** |
|---|---|---|---|
| **Recovering** | No | No | No |
| Recover to architectural representation | No | No | No |
| Analysis of legacy architecture | No | No | No |
| Retrieving requirements | No | No | No |
| **Migration** | No |  | No |
| Refactoring for SOA | Can be a tool for transforming of model to model | Can be a tool for transforming of model to model | Can be a tool for transforming of model to model |
| Refactoring for SaaS | Can be a tool for transforming of model to model | Can be a tool for transforming of model to model | Can be a tool for transforming of model to model |
| Control your work (Analysis) | No | No | No |
| Traceability between models | No | No | No |
| **Service modeling (PIM-PSM)** | No | No | No |
| Modeling for SOA and MDA | No | No | No |
| Modeling for SaaS | No | No | No |
| Modeling service mediation | No | No | No |
| **Platform (PSM-Realization)** | No | No | No |
| Transform to SOA | Can be a tool for transforming PSM to platform specific code | No | No |
| Transform to SaaS | Can be a tool for transforming PSM to platform specific code | No | No |

## 4.4 EXISTING MODELING SYSTEMS

In this chapter three different systems are selected for evaluation, Objecteering[37] which is the tools used in the SiSaS project, BluAge[38] since they have implemented the ADM taskforce KDM and TSRI[39] since they are one of the major players in the marked.

### 4.4.1 OBJECTEERING

Objecteering is a software product from Objecteering Software, a company under SOFTEAM Company[40]. Objecteering is UML 2.0 modeling tool with focus on SOA and MDA. Objecteering has good SOA modeling capabilities and they have incorporated the latest SOA standards like SoaML, BPMN and BMM. According to them self they provide full enterprise architecture solution making the solution suitable for business analysts as well as architects and developers.

The MDA module has features that make it a strong player in the MDA market. With support for several target languages like Java and .NET it is their support for J2EE that makes them interesting as a SOA MDA tool.

**Table 8 Objecteering evaluation**

| Area | Coverage |
|---|---|
| **Recovering** | No |
| Recover to architectural representation | No |
| Analysis of legacy architecture | No |
| Retrieving requirements | No |
| **Migration** | No |
| Refactoring for SOA | No |
| Refactoring for SaaS | No |
| Control your work (Analysis) | No |
| Traceability between models | No |
| **Service modeling (PIM-PSM)** | Yes |
| Modeling for SOA and MDA | Yes. Supports SoaML, BPMN, BMM, Code generation is adaptable through template programming. |

[37] Objecteering for details see: http://www.objecteering.com/
[38] BluAge application for details see http://www.bluage.com/index.php
[39]TSRI at http://www.softwarerevolution.com/
[40] SOFTEAM for details see http://www.softeam.com/

| | |
|---|---|
| Modeling for SaaS | No |
| Modeling service mediation | No |
| **Platform (PSM-Realization)** | Yes |
| Transform to SOA | Yes. Have good MDA transformation capabilities towards Java C#,C++.NET and J2EE, the last supporting SOA. |
| Transform to SaaS | No |

### 4.4.2 BLUEAGE

BlueAge corp.[41] a Netfective Technology group[42], is a software vendor focusing on Agile Model-Driven Development. BluAge is Built-On Eclipse and can transform UML 2.0[43] models into J2EE or .NET[44] applications.

BluAge has made a framework for reengineering approaches, which focuses on extracting legacy architecture into a PIM (Platform Independent Model) presentation and regenerate it to a modernized system using their MDA (Model Driven Architecture) approach.

BluAge has unfortunately not made their reengineering framework for trial purposes, so the functionalities in their framework are not tested by the author.

According to BluAge they are able to extract PSM (Platform Specific Model) models from existing legacy systems, currently those build using COBOL[45] or Java. The PSM model can then be transformed into a PIM model, then back to a PSM model representing the new system. During this process they support the use of several techniques to help the transformation from PIM to PSM, like manipulation of business rules, services and data.

The KDM is used for intermediate representation of the data structure during this process, which promises interoperability against other legacy software languages not yet implemented. They facilitate the ontology[46] concept within the KDM to ensure that the semantics of the legacy system are not lost.

Blue age seems to be a very promising tool for modernization purposes following the horseshoe model of the ADM taskforce.

[41] BluAge corp. http://www.bluage.com/index.php?cID=company
[42] Netfective Technology group http://www.netfective.com/index.php?cID=company
[43] **Unified Modeling Language** (**UML**) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed, and was created by, the Object Management Group.
[44] The **Microsoft .NET Framework** is a software framework that can be installed on computers running Microsoft Windows operating systems.
[45] **COBOL** (pronounced is one of the oldest programming languages. Its name is an acronym for **CO**mmon **B**usiness-**O**riented **L**anguage, defining its primary domain in business, finance, and administrative systems for companies and governments.
[46] In computer science and information science, an ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to define the domain.

**Table 9 BlueAge evaluation**

| Area | Coverage |
|---|---|
| **Recovering** | No |
| Recover to architectural representation | No |
| Analysis of legacy architecture | No |
| Retrieving requirements | No |
| **Migration** | No |
| Refactoring for SOA | No |
| Refactoring for SaaS | No |
| Control your work (Analysis) | No |
| Traceability between models | No |
| **Service modeling (PIM-PSM)** | Yes |
| Modeling for SOA and MDA | Yes. Supports SoaML, BPMN, BMM, Code generation is adaptable through template programming. |
| Modeling for SaaS | No |
| Modeling service mediation | No |
| **Platform (PSM-Realization)** | Yes |
| Transform to SOA | Yes. Have good MDA transformation capabilities towards Java C#,C++.NET and J2EE, the last supporting SOA. |
| Transform to SaaS | No |

### 4.4.3 TSRI

TSRI (The Software Revolution Inc.) is a company specializing in modernizing legacy software, and from their history record maybe one of the most experienced one. TSRI is also a member of the ADM task force and has implemented the ADM task force specifications of KDM and AST at the base of their tool framework JANUS[47].

JANUS is based on three high-level specification languages according to their web-site, namely;

- JPGEN™ for defining grammar system and language models.
- JTGEN™ for defining transformations between these models,
- JRGEN™, (a 5th generation artificial intelligence language), for model manipulation and analysis that supports 1st order logic and predicate calculus as well as 3GL[48] and 4GL language constructs.

TSRI have developed their own language neutral model, the IOM (Intermediate Object Model ) in which they transform all the legacy code before doing analyses, re-factoring and transformation on the legacy code into a new architecture and language.

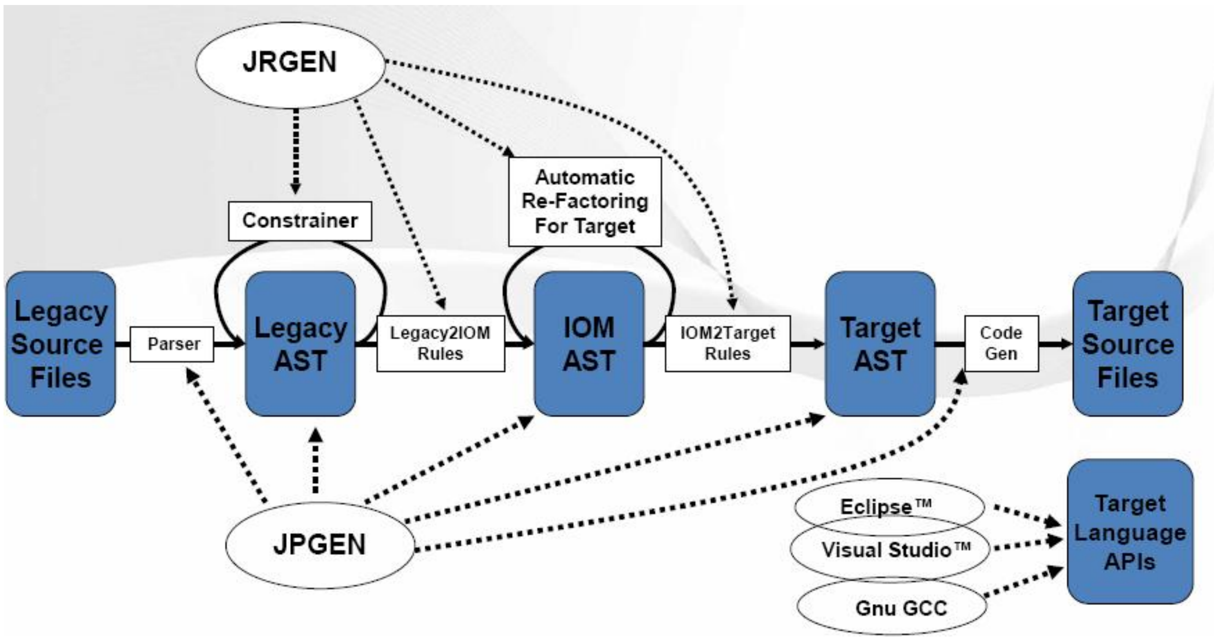


**Figure 10 Overview of TSRI's approach for modernization.**

[47] JANUS for more information see http://www.softwarerevolution.com/technology.html

[48] A **third-generation programming language** (**3GL**) is a refinement of a second-generation programming language. Whereas a second generation language is more aimed to fix logical structure to the language, a third generation language aims to refine the usability of the language in such a way to make it more user friendly.

**Table 10 TSRI evaluation**

| Area | Coverage |
|---|---|
| **Recovering** | Yes |
| Recover to architectural representation | Yes |
| Analysis of legacy architecture | Yes |
| Retrieving requirements | Yes |
| **Migration** | Yes |
| Refactoring for SOA | Yes |
| Refactoring for SaaS | No |
| Control your work (Analysis) | Yes |
| Traceability between models | Unknown |
| **Service modeling (PIM-PSM)** | No (Has extended functionality for making services out of existing code) |
| Modeling for SOA and MDA | No (Has extended functionality for making services out of existing code and transformation from models to code) |
| Modeling for SaaS | No |
| Modeling service mediation | No |
| **Platform (PSM-Realization)** | Yes (Not model based) |
| Transform to SOA | Yes. Have good MDA transformation capabilities towards Java C#,C++.NET and J2EE, the last supporting SOA. |
| Transform to SaaS | No |

## 4.5 IBM - Building web services for scientific grid applications

IBM has done a project that has the same goal as the OSCAR case with exception to the MDA & MDD approach. Whilst this thesis will look into how to model the solution, IBM describe their solution in [12] as an automated process, but the automation is based on a XML document that the users of the legacy systems must fill in. The project is in this thesis since the ideas on how to move legacy scientific software out to the cloud, or GRID[49] as described in the paper are valuable inputs.

The challenge for IBM was to solve the integration and choreography of several legacy applications, so the different applications could be utilized in calculating storm predictions. To do this they build a framework for running services on the GRID, which basically consists of a grid portal to interact with users, a generic factory service that could wrap legacy applications as services, a workflow composer to make these applications work together and a notification service for messaging and logging. The Figure 11 below shows the overall design.

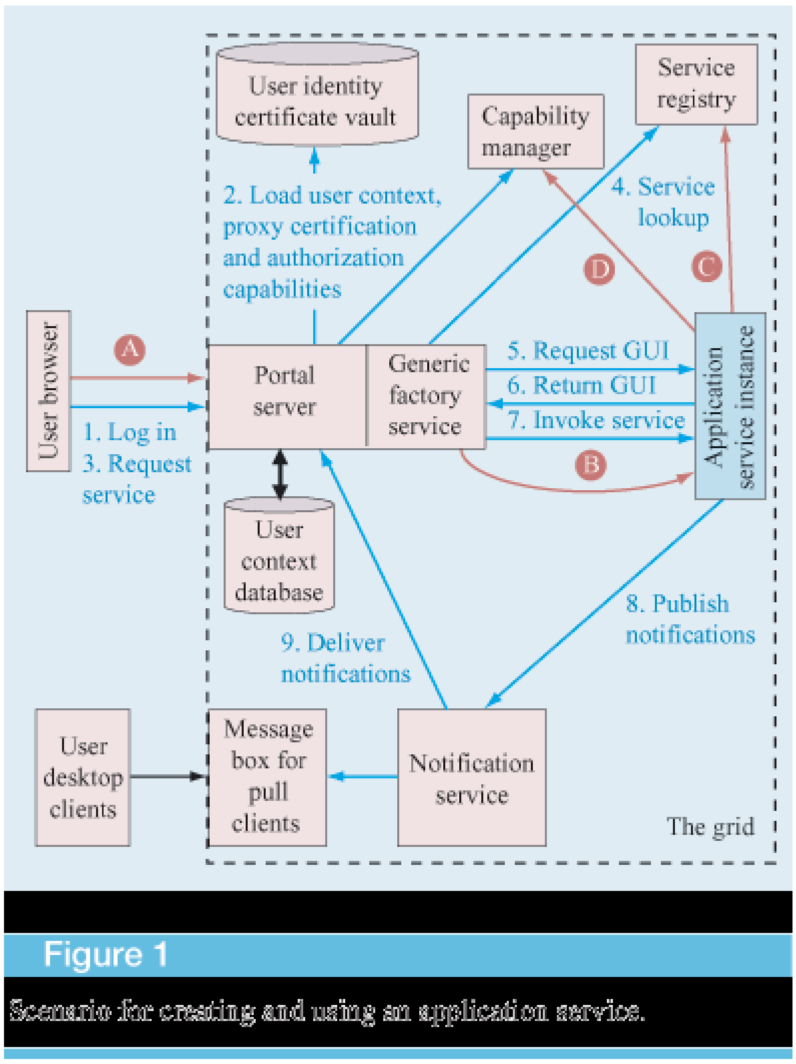


**Figure 11 From the IBM solution (1)**

[49] **Grid computing** is the combination of computer resources from multiple administrative domains applied to a common task, usually to a scientific, technical or business problem that requires a great number of computer processing cycles or the need to process large amounts of data.

The letters in the figure shows how to configure a new service

A) Upload SMD (Service Map Document**)** to portal, Portal pushes the SMD to the Factory service.
B) Factory service uses GRAM[50] (Globus Resource Allocation Manager) to launch the service, the Application service then configure itself and generates WSDL.
C) Register the service
D) Authorization registration in the capability manager

The numbers shows the users interaction, with self explaining text.

The application providers must fill out a SMD in XML, which must contain the necessary information about the legacy applications interfaces. Typically these are operations with their respective parameters and the flow of interaction.

For creating and accessing services, the factory does not create code based on the SMD, it uses  a standard message service that reconfigure itself based on the information in the SMD and then publish its WSDL to the registry.

To generate the UI (User Interface) IBM does this iin two steps

- First a HTML page is generated from the SMD which displays the different operations that can be invoked.
- Secondly, when an operation is selected, a new page is generated with information on parameters for the selected operation.

A set of parameters can be defined in the SMD, to tell how the input parameters are supposed to be viewed for the user.

Notification messages are sent to the notification service about status. If a user subscribes to this service he/she will be updated with messages from the application.

#### *4.5.1.1 Notification service*

WS-Messenger is IBM's implementation of a notification model that is compliant with WS-Notification and WS-Eventing specifications.

Publish-subscribe on topics. For clients that are not reachable a message-box holds the messages so these clients can pull messages when necessary.

Workflows subscribe to all notification for the services in the workflow. All services built by the factory has a built-in notification publisher.

A mediation approach is used to convert between notifications and events.

[50] GRAM, for details see http://www.globus.org/toolkit/docs/2.4/gram/

#### *4.5.1.2 Security*

For security in the application services, they rely on standard https-based username and password authentication plus a grid proxy certificate which is loaded after login.

Once authenticated, the portal fetches capability authorization tokens from the authorization system, called *XPOLA*, which is a fine-grained authorization infrastructure for web and grid services based on capabilities and the principle of least authority (POLA).

#### *4.5.1.3 Composing services*

To compose services on the grid IBM has build their own composer called X-Workflow, which is accessible through the net. The service composer can search for available services, connect them together in a workflow, create BPEL[51] or Jython[52] for execution and follow the execution in the same user interface.

**Table 11 Evaluation of IBM GRID solution.**

| Area | Coverage |
|---|---|
| **Recovering** | No |
| Recover to architectural representation | No |
| Analysis of legacy architecture | No |
| Retrieving requirements | No |
| **Migration** | No |
| Refactoring for SOA | No |
| Refactoring for SaaS | No |
| Control your work (Analysis) | No |
| Traceability between models | No |
| **Service modeling (PIM-PSM)** | No |
| Modeling for SOA and MDA | No |
| Modeling for SaaS | No |
| Modeling service mediation | Partly. Has a solution for composing registered services. |

[51] **Business Process Execution Language** (BPEL), short for *Web Services Business Process Execution Language* (WS-BPEL) is an OASIS standard executable language for specifying interactions with Web Services.
[52] **Jython**, successor of **JPython**, is an implementation of the Python programming language written in Java.

| **Platform (PSM-Realization)** | No |
|---|---|
| Transform to SOA | No. Yes, good wrapping techniques for generating services. |
| Transform to SaaS | No. Yes, good wrapping techniques for generating services and includes security and payment system. |

## 4.6 Cloud Computing

Cloud Computing is the latest buzz-word around, and promises a whole lot of advantages compared to standard way of computing. In this chapter we will look into some of the major players in this are and what they can offer, and how this can be utilized seen with the "legacy software glasses" on. No evaluation of the clouds are done since it falls outside the scope of the thesis, but since the topic is hot and the cloud is a possible platform for the OSCAR case the thesis will cover some basics about the cloud.

Gartner Group was early predicting that Cloud Computing will be the next big thing in the way we develops and host our software.

*"By 2011, early technology adopters will forgo capital expenditures and instead purchase 40 per cent of their IT infrastructure as a service…,Enterprises believe that as service oriented architecture (SOA) becomes common "cloud computing" will take off, thus untying applications from specific infrastructure."* –Gartner Press Release, "Gartner Highlights Key Predictions for IT Organizations and Users in 2008 and Beyond" 1/31/2008

It has to be said that this is somewhat modified later, and critical voices has predicted that the promises given by Cloud Computing vendors might not be as great as first expected. In February this year Gartner says that "Cloud Computing" needs several more years before it reaches its full potential, "Gartner Says Cloud Application Infrastructure Technologies Need Seven Years to Mature" 2/2/2009

Still Cloud Computing gets a lot of interest these days, and various vendors such as Amazon, Google etc. are offering Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) solutions. Such solutions can be a possible target execution infrastructure for software that is being migrated towards SOA and SaaS. Some of these infrastructure and platforms are shortly described below. The considerations for mappings to these are suggested as interesting future work.

### 4.6.1 Amazon

Amazon, one of the first to go world-wide commercial, when it comes to allow for anyone to use their services to a seemingly minimal cost, has become one of the leading suppliers of Cloud Computing infrastructure as-a-services. Amazon delivers several aspects of this where the major one is Amazon Elastic Compute Cloud (also known as "EC2").

###### 4.6.1.1.1 Amazon Elastic Compute Cloud (Amazon EC2)

The EC2 give customers to specify their need for computing power and the possibility to configure the system themselves on Amazon's computer park.

###### 4.6.1.1.2 Amazon SimpleDB

This gives the user a possibility to store their data in the cloud without having to pay for a database system, hardware, and needed maintenance. The simpleDB is not organized as a normal relational DB in which you have to provide db-schemas and indexing the tables yourselves. Amazon does this for you, provided that you deliver data in a structured form.

###### 4.6.1.1.3 Amazon Simple Storage Service (Amazon S3)

Amazon S3 works as a data storage in the cloud, where users can store their data, documents and application data.

###### 4.6.1.1.4 Amazon CloudFront

CloudFront is a service that works as a CDN (Content Delivery Network), which means that the content the users are accessing will be moved to the closest server for the user, and thus improving speed of retrieval.

###### 4.6.1.1.5 Amazon Simple Queue Service (Amazon SQS)

Amazon SQS works as a simple message queue. Applications or components can send and receive messages against this queue in the cloud as long as they are member of the AWS (Amazon Web Services)

###### 4.6.1.1.6 Amazon Elastic MapReduce

This is a functionality that provide ease of setting up a computing cluster for heavy data processing and calculations within the cloud. The MapReduce gives the user a possibility to select the number of computers, EC2 instances they need, and then takes care of the set-up and work-flow monitoring.

###### 4.6.1.1.7 AWS Premium Support

This is Amazons support program for the customers that use their Cloud Computing capabilities.

### 4.6.2 SUN CLOUD

Sun microsystems[53] bases their Cloud Computing technology on their OpenSolaris[54] operating system, which again is based on their Solaris operating system, which again is based on UNIX. As a database platform Sun provides the open source product MySQL[55], and as programming language Java. As application server they provide the open source community product GlassFish[56].

All products that open source developers and architects are familiar with. In addition to have the infrastructure and platform for Cloud Computing Sun will deliver several supportive products to aid any organization that wants to utilize their cloud. These products will include the following:

- Development tools
- Open API
- Partner Ecosystem
- Network scale products and technologies

[53] Sun Microsystems, for details see http://www.sun.com/index.xml
[54] OpenSolaris, for details see http://hub.opensolaris.org/bin/view/Main/about
[55] MySQL, for details see http://www.mysql.com/?bydis_dis_index=1
[56] GlassFish, for details see https://glassfish.dev.java.net/

- Service and support
- Security

### 4.6.3 GOOGLE

Google[57] which is one of the major organizations on the internet, has also launched their Cloud Computing platform, the Google App Engine[58]. Google App engine is Google's platform in which one can develop applications for running in Google's Cloud.

Google supply a java like runtime environment to support Java development, standards and frameworks, and under development is a Google plug-in for Eclipse. Their database support is limited to their own database and query language which is a restriction that can cause problems for applications with heavy database functionalities and usage.

### 4.6.4 MICROSOFT

Microsoft is, of course, also heavily investing in Cloud Computing with a platform they call "Azure Services Platform". "Azure Services Platform" delivers several kinds of services in the cloud, and among these are the following;

#### *4.6.4.1 Windows Azure*

It is in Windows Azure developers can build and deploy their web services and applications onto the internet. Windows Azure then gives the flexibility of management, on-demand scaling, computing and storage power in the cloud. This is however not commercial available before the end of this year (2009).

#### *4.6.4.2 Microsoft .NET Services*

Microsoft .NET Services provides a .NET Service Bus and .NET Access Control Service for the developers on Windows Azure. The .Net Service Bus helps the connecting of services in the cloud providing means for navigating firewalls and network boundaries. The bus is also the place to register your services and compose new services. .NET Services are web-based developer services that help developers focus on their application logic rather than deploying and managing their own cloud-based infrastructure.

#### *4.6.4.3 Microsoft® SQL Services*

Currently the service delivered under "Microsoft SQL Services" are relational database services under the name "Microsoft SQL Data Services". This is, as the name states, offering of deployment of SQL databases solutions to the cloud, with the possibility to perform the basic RDBMS functions.

#### *4.6.4.4 Microsoft Live Series, SharePoint Services, Dynamics CRM Services*

These implementations are to be delivered in the future.

[57] Google corporation, for details see http://www.google.com/corporate/
[58] Google App Engine, for details see http://code.google.com/intl/nb/appengine/

## 4.7 An overall evaluation

The table shows an overall evaluation of the selected applications and methodologies compared to the proposed methodology MILAS.

**Table 12 3 Requirements and technology/solution coverage**

| **Area** | ADM | Tansformation tools | MDA tools | Commercial actors |
|---|---|---|---|---|
| **RECOVER** | √ | | | √ |
| Recover to architectural representation | √ | | | √ |
| Analysis of legacy architecture | √ | | | √ |
| Retrieving requirements | On-going work | | | √ |
| **MIGRATE** | | √ | | √ |
| Refactoring for SOA | | √ | | √ |
| Refactoring for SaaS | | | | |
| Control your work (Analysis) | | | | √ |
| Traceability between models | | Partly | | √ |
| **SERVICE MODELING (PIM-PSM)** | | | | √ |
| Modeling for SOA and MDA | | | √ | √ |
| Modeling for SaaS | | | | |
| Modeling service mediation | √ | √ | √ | |
| **PLATFORM (PSM-Realization)** | | | √ | √ |
| SOA and SaaS | | | √ | |

## 4.8 Conclusion

The conclusion from this overall evaluation is that there is a need to put different solutions together to form a holistic solution or methodology to support a modernization process.

The ideas of OMG's ADM approach are something that seems promising, and methods to recover legacy software into a standardized target model sounds like a solution for the future. The diversity in the many software development languages and platforms has always caused problems for those who want to interact with other products. Even though this does not solve the challenges of interaction in the development and application world, it could solve the interaction problems for tools that can help viewing, migrate, analyze and transform target architecture. Since they now can have a common standardized model definition that they can relate to.

The transformation languages can be a great tool to implement in the processes of migration and model driven development. The transformation tools can be used both to build libraries and templates for source to target model transformations (PIM to PIM) and for model to source to platform (PIM to PSM to platform specific source code) transformation. Java or another 3GL language could have been chosen to perform these transformations, but the simplicity of the semantics and specialization towards transforming of models makes the transformation languages suitable for the task.

As the base modeling language UML has become the standard to use a long time ago and cannot be avoided if one are to reach out to the majority of potential users. SoaML seems to be a good and solid standard for modeling services for reuse and later transformations.

The solution provided by the IBM gives ideas on how to build a platform for running wrapped legacy software that could be something to take further if one are interested in keeping the control of the same applications.

# 5 MILAS (MODERNIZING LEGACY APPLICATIONS TOWARDS SOA, SAAS AND THE CLOUD)

There are numerous considerations and challenges that will face an organization when modernizing a legacy system based on SOA principles, SaaS platform and Cloud Computing. MILAS will show that many of these challenges can be met by following a model driven methodology and that many of the needed tools already are in place. Some of the challenges will however demand new technology and MILAS will look into these with various degrees.

Depending on the goals for the different organization, they will have to choose different paths to reach their goal. MILAS will try to cover the different steps needed in these paths for an overall modernization strategy.

As the case studies show, there can be quite different types of legacy systems and organizational goals which make the basis for these kinds of projects. While the Progress 4GL legacy system has as goals to be more maintainable and open towards other applications used by the organization, the main goal for the OSCAR case is to make the system available to as many users as possible by running the application on a platform, the cloud, which can meet performance, scalability and availability issues. Nevertheless both legacy system needs to be moved towards SOA and SaaS.

The MILAS methodology will try to cover both cases and to give the reader an option to pick and choose from the different steps involved based on their needs and interests. The main steps are listed here and explained later in their own chapters.

1) ***Recover and understand legacy architecture:*** In this step the legacy architecture are recovered and analyzed, based on source code and other available artifacts.
2) ***Migrating towards SOA and the cloud:*** In this step the legacy architecture will be refactored to towards the target architecture, using analyze and refactoring tools.
3) ***Service modeling and MDA transformations:*** Define and extract services and model them using UML, SoaML and wrapping.
4) ***Select target platform and deploy:*** The target platform is selected and the modernized system deployed.

There are many SOA principles (see chapter 3.2) that must be considered to be implemented into the development strategy and rules. The principles give good guidelines to make applications, modules and functionality much more robust to changes and better suited for maintenance than old style coding. The basic idea is to redesign processes and functionalities in the system to be as self-contained and independent as possible, which again make them appropriate for web services and communication with other systems.

When deploying the system as SaaS on a Cloud Computing environment other issues will also come into play (see chapter 3.2). If one are opening up an application from being a one client application to be multi-client application, allowing for multiple users from different organization using the same application instance and data storages, without allowing for sharing of data between them. Then the system needs to be highly scalable, flexible, manageable and secure.

MILAS will provide for methods and give guidance on how to migrate existing legacy software towards a modernization system following a SOA/SaaS paradigm.

Furthermore one will see that the approach taken for a legacy scientific system opposed to an legacy business system, can be quite different with respect to the resources and depth of reengineering needed, but still could follow the same guidelines.

As a baseline for this thesis is the horseshoe model, shown in Figure 7 and defined by SEI[59] (Software Engineering Institute) and taken into the ADM task force initiative by OMG[60] (Object Management Group).

Since most legacy systems seem to lack proper documentation, the value lay in the source code itself. Therefore the representation of the source code is a key issue, and reconstruction follows a bottom-up design view and starts from source code, to structure, to the function-level, and then architecture representation.

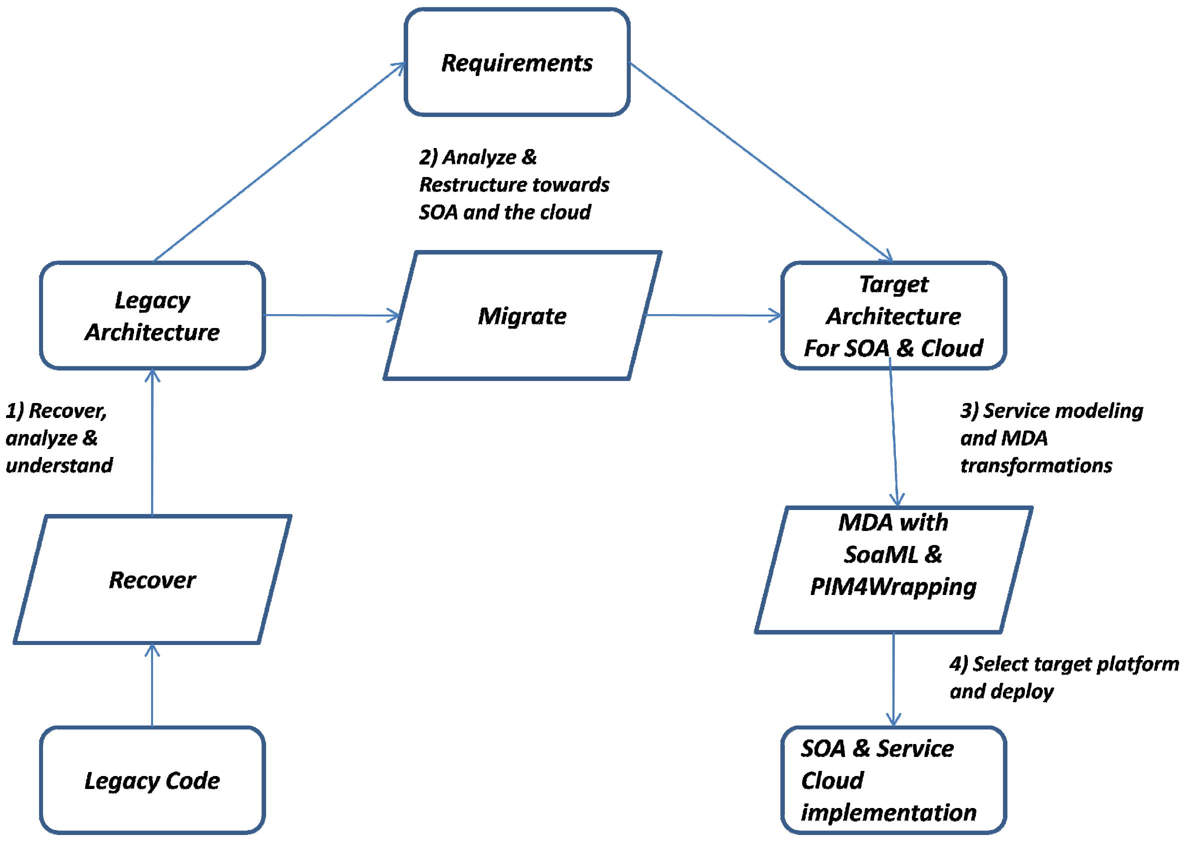

Requirements
2) Analyze & Restructure towards SOA and the cloud
Legacy Architecture
Migrate
Target Architecture For SOA & Cloud
1) Recover, analyze & understand
3) Service modeling and MDA transformations
Recover
MDA with SoaML & PIM4Wrapping
4) Select target platform and deploy
Legacy Code
SOA & Service Cloud implementation

**Figure 12 MILAS overview with PIM4Wrapping**

As Figure 12 shows, the basic steps involve recovering an architectural representation of the legacy code, and then modernize the architecture to a target architecture using refactoring and analyze tools to transform the origin to a modern structure. The target architecture will then be modified dependent on new and modified requirements plus analysis that will help construct a sound and modern architecture. Last but not least, the final step is to transform the target architecture to a PSM (Platform Specific Model) and deploy it on the new platform for SOA or Cloud Computing.

[59] **The Carnegie Mellon Software Engineering Institute** (SEI) is a federally funded research and development center. Homepage: http://www.sei.cmu.edu/

[60] **Object Management Group** (OMG) is a consortium, originally aimed at setting standards for distributed object-oriented systems, and is now focused on modeling (programs, systems and business processes) and model-based standards.

## 5.1 Recover and understand legacy architecture

As an important first step when modernizing or doing any major change to an existing system is to get an overall understanding of the same system. To get this initial understanding one should gather and read available documentation, talk to users, maintainers, developers, stakeholders and any other resources that might have valuable inputs. If available one should also explore test installations to get a better picture of how the system is and can be used. When this initial understanding is achieved the work on modernizing can start.

### 5.1.1 Recover to architectural representation

This step is about getting a model based representation of the legacy architecture and to understand the legacy software. The ideal situation is to achieve this by extracting information from the legacy system into a KDM and AST representation that can be analyzed and manipulated in a graphical model environment.

### 5.1.2 Analysis of legacy architecture

To get an understanding of the source code and to reconstruct the systems architectural overview there are several methods than can be applied.

Manual code reading is always applied in some form or another, and is necessary to verify assumptions made by the different tools used. But to manually read a large legacy system is a too huge task for a programmer.

There exist a large number of tools that can analyze different types of sources in various ways. Four basic and useful analyzing methods as described below are recommended

First of all one should have a tool that can extract information about elements, and the relationship among them. Typical elements to consider could be elements that call other programs or procedures, elements that access a data source and global variables. In this way one can track how and where fundamental elements such as database tables and fields are updated.

Static analysis is commonly used to get overview over call sequences, control flow, cross references and variable usage. Static analysis can be a great help for understanding the architecture of a system, but there are items such analysis cannot detect. Late binding cannot be detected in the source code, since the relationships have not been made yet.

Dynamic analysis on the other hand can detect such bindings and log and graphically view actual call sequences and data flow.

Slicing is technique first described by Mark Weiser in [13]. Slicing can be done both dynamically and statically and is performed on an object or parameter of interest. The technique can then look at only the statements that affect the selected element.

Having an AST and KDM representation of the legacy source code and artifacts is a good basis to start analyzing and building an architectural view of the legacy system.

### 5.1.3 Retrieving new and old requirements

Before doing a migration, new requirements from the business management must be taken into consideration. These requirements can be put into the model space using BPMN, BMM.

Combining this with tracing techniques verifications can be done automatically to confirm that the requirements are met in the target architecture.

Business requirements that are hidden in the legacy code can be extracted in various degrees using business extraction tools. These works much like control flow analysis, except that they have incorporated logic to exclude unnecessary information, such that only the essence of the logic is capture. This can then be converted into requirements for the modernized system.

The AST is also a good basis for discovering and extracting existing business process, requirements and business rules that are hidden inside the legacy system.

### 5.1.4 SELECT STRATEGY

To avoid huge project steps, that are hard to control, and that involves high risk, a Chicken Little approach [4] should be selected when possible. Chicken Little approach means splitting the system into small manageable parts, and migrate them on by one to gain control. The challenge with this method opposed a cold-turkey approach [4] is that one need the run the modernized parts of the system along with the legacy part of the system. To solve this one need to find technology suitable for the task.

Before going further it is advisable to evaluate the need for resources. A migration project towards SOA and SaaS require resources and tools that are up to the task. There are several options to select from and one should consider and select a resource strategy.

In-house, if the project can be carried out with in-house resources it will give the best benefits in a long term basis. The knowledge and know-how will stay in-house and modifications and adjustments to the system will be an easier and cheaper task compared to having consultants do these kinds work. On the other side it requires that the in-house resources are skilled in modernization approaches, and have knowledge of the target platform, service design and so on. In-house resources can also be tied up in other projects where they cannot be replaced and thus be a poor choice for the project.

Consultants on the other hand can be expected to be experts in these areas and are likely to generate a good solution faster than in-house personnel. The down-side is of course the cost and that the knowledge comes and goes with the consultants.

A combination of the above would probably in many occasions be the preferred solution. Then the in-house personnel can be trained in modern technology at the same time as one has expert help to carry out the actual work.

From an IT perspective one would wish to migrate and modernize the whole system. Economical factors might make this approach a naive one. Sometimes a modernization approach must include wrapping of legacy code, which is a good solution if resources are sparse and one need a fast way to production, but a bad one since the challenges of a legacy system will remain.

According to [14] the following can help taking the right decisions regarding wrapping or reengineering.

Good wrapping candidates have the following properties.

- Incomprehensible or difficult functionality.

- Well working
- Autonomous

Good reengineering candidates have these properties.

- Needs maintenance
- Needs changing from requirements
- Easier logic

Agile software development[61] has been popular for a while and has its obvious advantages. Short activities make room for fast results and motivation since one has accomplished something that is notable. For our purpose it would be desirable to modernize small portions of the code at time, to rapidly see the result both in form of the new architecture, but also as running code alongside the legacy software. We have already decided to prefer the Chicken Little approach before the cold-turkey, and the goal should be to try to work on sub-module level if possible. Hopefully the legacy code aren't so intertwined that such splitting of activities becomes impossible.

## 5.2 MIGRATE TOWARDS SOA AND SAAS

Having the technical understanding, modeled representation and requirements of the legacy system, one is ready to start the migration process. Migrating software are however a complex matter and using help from experts and tools are recommended.

Using the Chicken Little approach one will have a manageable piece of legacy software to migrate at hand. This is preferable if possible, since the project can be managed by a small group of people and the risk involved is much less compared to the cold-turkey method.

Several goals should be derived from this step, the main goal is to rearrange and convert the source architecture into a more decomposed, decoupled and autonomous architecture than the original. This makes the basis for being able to generate a system that follow SOA and SaaS requirements. New and existing requirements must be taken care of and any changes to the legacy software in this period must be detected and implemented in the new solution.

By being able to track the pieces of legacy AST that has undergone a transformation to the target AST, one allows for even smaller chunks of code to be transformed at the time, while still being able to see the full picture having the legacy AST intact. With traceability like this one would gain control over the process by easily being able to see which pieces that has undergone a transformation and which that hasn't.

Since an AST is build up using simple UML elements, several transformation languages can be used to build the necessary transformations.

The general idea for MILAS in this step is to use analysis tools on the legacy AST to detect potential classes, patterns, structural problems and transform them into a new modernized AST where additional analysis can be done to optimize the new source code following the above mentioned SOA, SaaS and Cloud Computing requirements.

[61] **Agile software development** refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

As mentioned in the requirements cohesion and coupling are important metrics that must be met to ensure that the new source code is serviceable. The next chapter introduces several steps that will help a project obtaining such results.

### 5.2.1 GRADUALLY REFACTOR CODE TOWARDS SOA AND SAAS

To meet non-functionality requirements such as reusability, maintainability and portability it is important to restructure the code to a good OO (Object Oriented) design

There are many ways to rearrange source code, the thirteen step approach suggested below are inspired by the articles by Ying Zou [15] [16] [17] [18] [19] [20]. For further readings on good OO refactoring [21] are recommended reading.

By following the below steps and gaining a new architecture that has high-cohesion and low-coupling most of the SOA requirements are met. Later, when doing the service modeling the last step towards independent and autonomous services will be achieved.

1. Create main classes derived from the overall understanding. This can be a good starting point.

2. Examine the flow of information between modules, looking at parameters and globally shared data. This will help figuring out the boundaries and to obtain utility classes. Classes should be created in the new AST with linkage to a UML class model.

3. Organize the potential classes and objects found in step 1 and 2. This can be carried out using four immediate steps. First determine concrete classes by identifying the objects that are instances of those classes. Secondly identify duplicate classes and objects, thirdly classes with no objects can be potential abstract classes and finally re-evaluate objects that seems not to belong to any classes.

4. Identify attributes. Global attributes and local attributes. If the legacy system contains a relational database, most of the attribute are given there.

5. Identify methods. To identify methods, there are several approaches to take, due to the nature of the legacy source code. Five different scenarios might occur. First there are the modules, which can be regarded as a cohesive method, but also can involve functionality that belong in several different classes. Then one should examine existing functions and classify them as behavioral, associative, unassigned or virtual. The different types can be explained as;
   **Behavioral**: Functions that uses or manipulate data that belong to one specific class.
   **Associative**: functions that can belong to more than one class.
   **Unassigned**: Functions that cannot easily be identified, belonging to any class.
   **Virtual**: Functions that belongs in an abstract class, so the appropriate method can be executed at run time (polymorphism[62]).

6. Decide on entity objects and control objects and user interface objects. MVC (Model View Control)

[62] Polymorphism, in the context of object-oriented programming, is the ability of one type, A, to appear as and be used like another type, B.

7. Re-evaluate unsigned functions. Since it is not desirable to have independent functions in an object model, these functions should be re-evaluated and fit into existing classes or new ones.

8. Look for patterns. Discovering patterns can save a lot of coding if different chunks of code really performs the same job, and therefore can be put into the same methods.

9. Refine class definition, (and database structure). A more thorough examination on the found classes should be carried out. One should evaluate if attributes are specific to the class, or implementation specific. For instance array declarations should be inspected closer, in case these actually should be defined as classes themselves.

10. Move direct access to files or the database to entity classes, and replace the access them with method calls.

11. Build in transaction. If necessary transaction methodology should be taken into consideration and built into the system.

12. Determine class relationship. Relationship among the classes should be established. Especially inheritance and aggregations should be determined.

13. Re-evaluate any potential virtual functions.

### 5.2.2 CONTROL YOUR WORK

Every iteration on the above steps should be followed with analysis of the generated AST and UML models. Analysis should be carried out to ensure that the new system is manageable and reusable. Measuring the amount of cohesion and coupling helps determine the robustness of the new system.

## 5.3 SERVICE MODELING AND MDA

While the AST is ideal to perform transformation and analysis on, additional models of the new system should also be built. Models can be used for several purposes in addition to the obvious purpose of viewing the desired system in a commonly understandable language.

Basic class models helps to understand the structure of the system and if linked to the AST model can be a powerful tool for visualizing, rearranging and refactoring target code. Newer model standards like BPMN and SoaML can help to visualize and structure parts of the code that needs to be exposed to the outer world. SoaML ensures that the models will be contract centric and that boundaries are kept. BPMN diagrams are not only helping the business management to define and understand systems, they can also be transformed into BPEL code that can control the process flow that the new services will participate in.

New and existing services should be modeled using SoaML, SoaML extend the standard UML 2.0 with terms like “service interface”, “participant”, “service contract”, “service architecture”, “message type” [7] and more and provides the means to model well designed services.

Figure 13 in the next section shows the basic elements that are needed to design a service with SoaML. The service interface realizes a standard UML 2.0 interface, which will be the methods that the service offers to the outside world. The required interface tells if the service is depended on other services and last the service has a behavior element attached to it that specify how to interact with the service.

### 5.3.1 SOA, MDA & WRAPPING (FEDERATION)

Federation is one of the SOA requirements and a necessity if a modernization of parts of the legacy system is not an option. In short terms federation is the same as wrapping of code in this thesis context. Wrapping is a technique often used in IT-technology to make legacy pieces of code available to a newer system. The basic idea is to build new software around the existing so that the legacy component is exposed to the outer world as a modern component.

To avoid manual coding every time a new legacy system has to service enabled it is necessary to create a general way of modeling the wrapper. The idea is to model this wrapper as normally and simple as possible, to ensure that the people that actually will perform the task, finds it a fairly easy job. To solve this, a good starting point would be the generic service interface definition in SoaML [7].

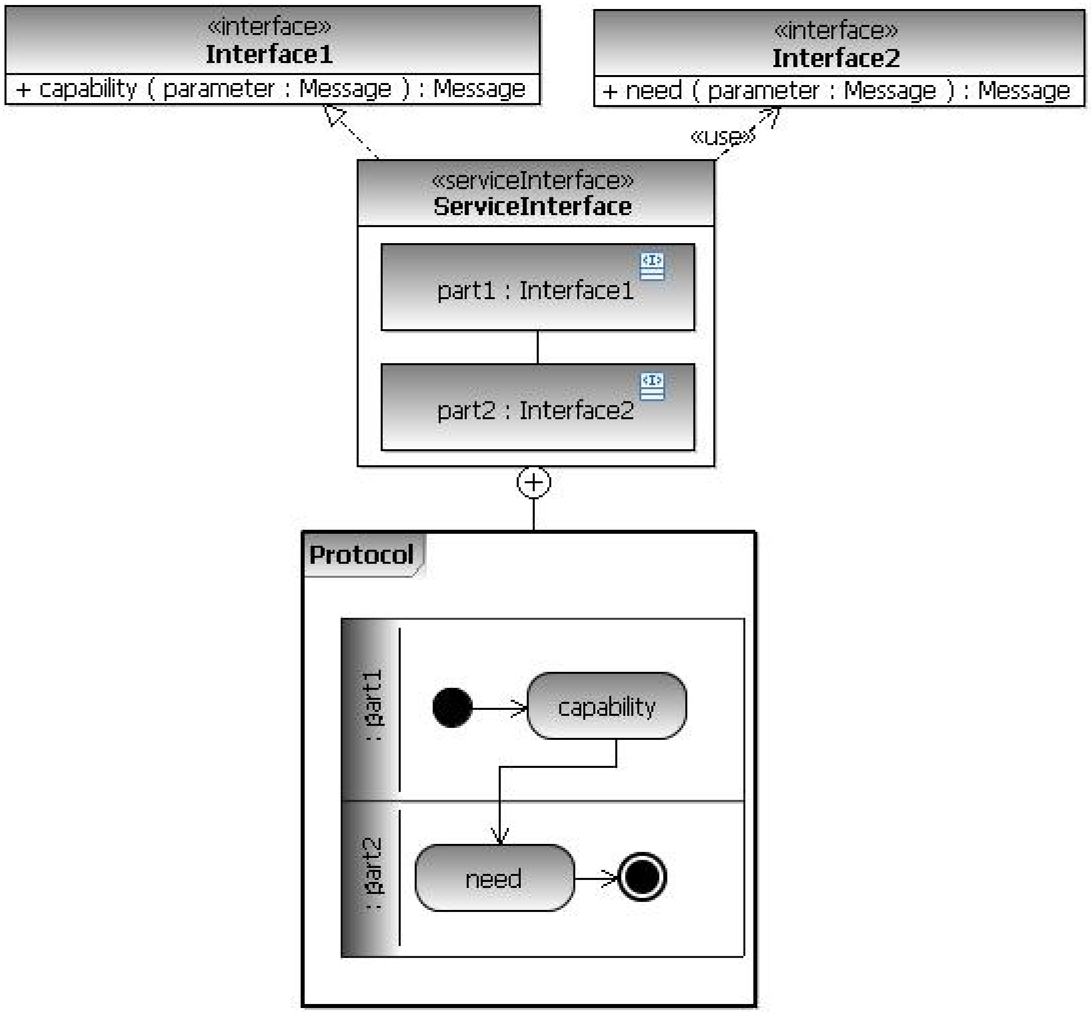


**Figure 13 A generic service interface**

Figure 13 A generic service interface shows how a service can be defined in the simplest, yet complete, way. The SoaML stereotype ”serviceInterface”, is applied on a standard UML 2.0

class object, and defines the service itself. Then the SoaML specification makes it possible to both define required and provided interfaces for the service, using the stereotype ”interface” that modelers are common with. In addition the SoaML specification states that the behavior of the service should be modeled using an activity or interaction diagram as shown in the figure. In this way the necessary information to generate code skeletons are available.

Even though the UML language is rich there are needs that standard UML cannot tackle. To accommodate for this UML provides a generic extension mechanism that can be used to add special meaning to the UML elements called profiles. A profile is defined using stereotypes, tag definitions[63] and constraints[64] to existing model elements. A profile consists of a collection of such extensions that typically are applied for a particular domain. In our case we will make a new profile that we call “PIM4Wrapping” (Platform Independent Model for Wrapping), and add this as an extension to help identify the elements that are special to wrapping legacy system code. The model elements can later be detected by the transformation logic, and thus ensure that appropriate transformation is performed for the right model instances.

To be able to run external components or executables from within a service, additional information in the SoaML model is needed. Information such as the location and initial parameters for the component must be present in the model to be able to transform the model into run able artifacts.

#### *5.3.1.1 Wrapping components*

Often when it comes to software it is the whole application that needs to be put into the cloud, and therefore wrapping of an executable is necessary to expose it as a service. To accomplish this, the first that must be in place is the location of the executable and the necessary start up parameters. The information elements will be put into PIM4Wrapping, the new UML profile, which has defined stereotypes with tags to it.

The new profile needs a stereotype that can be put on a component, and this stereotype will have tag values to describe the location of the executable. The stereotype is ”WrappedExecutable” and the belonging tag name is “PathToExecutable” where path and executable name are placed.

[63] A tag definition is a definition of a parameter in UML. Each stereotype has zero or more *tag definitions*, and all stereotyped UML elements have the corresponding number of tagged values.
[64] A constraint is a way to define an limitation on an UML element.

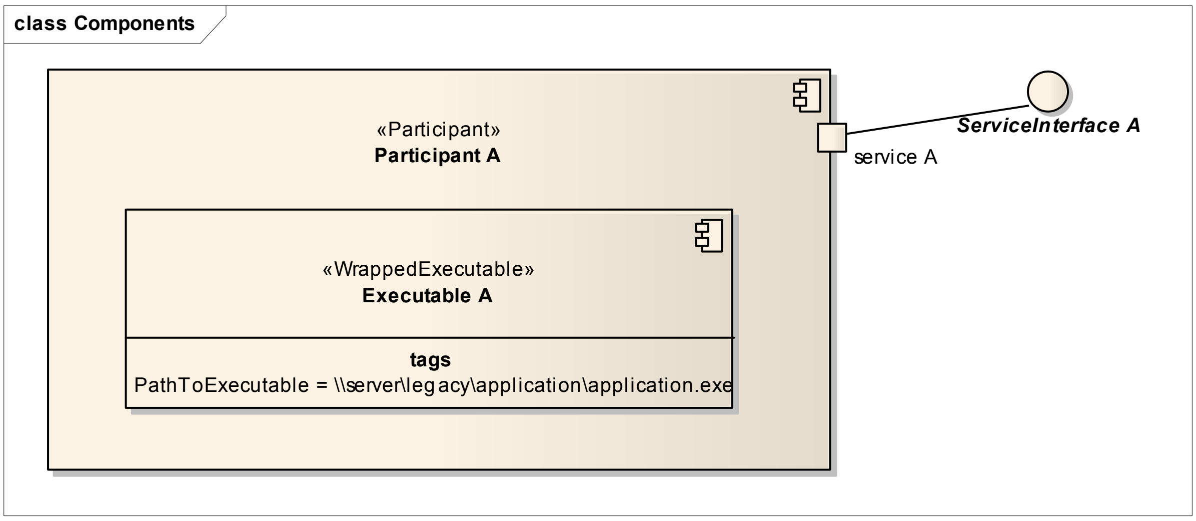


**Figure 14 Wrapping profile example 1**

An executable can have start up parameters and this information must also be included in the model. For this purpose MILAS suggest using the message stereotype in SoaML and use this as the parameter for the service interface method.

The message will be identical for every executable and only contain a string parameter.

The message structure must be specified for every different executable that will be modeled.

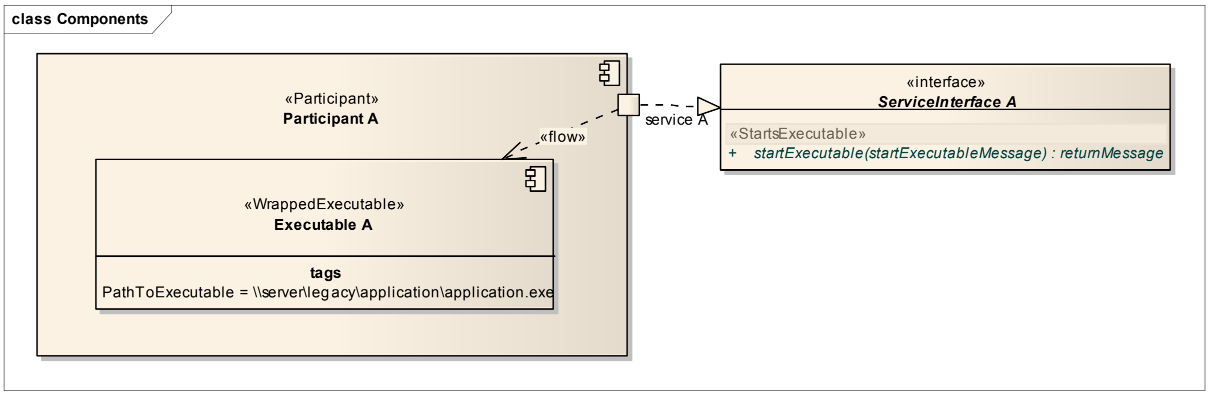


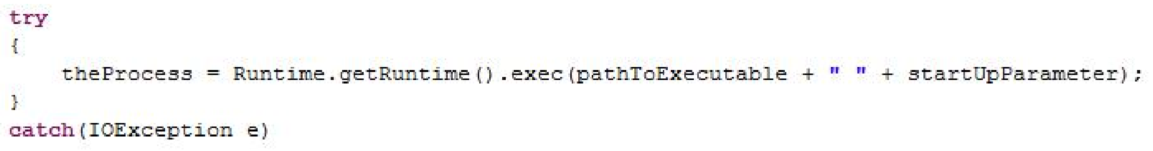
**Figure 15 Wrapping profile example 2**

The minimum amount of information needed to apply in the model is to mark the executable component and the method that will start it up with the appropriate stereotypes from the PIM4Wrapping profile.

To accomplish this apply the ”WrappedExecutable” stereotype on the executable component, and fill in the tag value, then place this component into a SoaML participant. Next apply the stereotype ”StartsExecutable” on the method in the service interface that actually will start the executable. One are now almost ready with basic wrapping.

Having the PIM4wrapping profile ready, generating a transformation from the PIM model to a PSM model is necessary. To accomplish this MOFScript can be used as the tool. MOFScript is a language specially designed to transform models to textual representation, and is well suited for looping through model elements to find candidates for a transformation.

For this scenario, when the transformation code reach an operation stereotyped ”StartsExecutable” it will insert the needed code fragments for executing an external application. If the target language is Java such code fragments could be a method starting up the executable. Part of the generated code could look like this;



```
try
{
    theProcess = Runtime.getRuntime().exec(pathToExecutable + " " + startUpParameter);
}
catch(IOException e)
```

**Figure 16 Java code starting an external executable**

This code piece will be responsible for firing off the external executable component. The values from the tag and message are inserted in the statement. Since the “startUpParamter” is in the message received from the user, it can be specified at run time ensuring the flexibility of the executable. The “startUpParamter” should be initialized in the model to aid any users of the system.

Starting up an executable is fairly simple, a bit more troublesome is to invoke methods in a legacy application of a different language that the selected target language. To solve this on a Windows platform the generation of a DLL[65] (Dynamic Link Library) as a mediator to the legacy code can solve the challenge. For Unix platforms a shared object (.so file) can be used as the Windows equivalent.

Java is delivered with a framework they call JNI[66] (Java Native Interface) which gives Java application possibilities to communicate with other applications written in another language such as C, C++ and Fortran. JNI supply a possibility to take control over the legacy code, by communicating directly through DLL’s on Windows based system and through shared objects on Unix based systems.

A third possibility is to call external components that can communicate through Java RMI (Remote Method Invocation) this is not very different from calling executables and wrappers can be made without too much effort.

#### *5.3.1.2 Interaction with wrapped legacy services*

It is not unlikely that the external component needs input in a sequence. For instance it could need some additional information after processing the parameters at start-up. It could be as simple as a question like “Do you want to continue?” or some form of parameter input.

This can be specified in the contract between the participants, the client and the web service wrapper. Every contract should contain a behavioral element that typically would be specified using an interaction diagram or an activity diagram. The diagram will then define how the participants will interact.

[65] **Dynamic-link library** (also written without the hyphen), or **DLL**, is Microsoft's implementation of the shared library concept in the Microsoft Windows and OS/2 operating systems.
[66] JNI, for details see http://java.sun.com/javase/6/docs/technotes/guides/jni/index.html

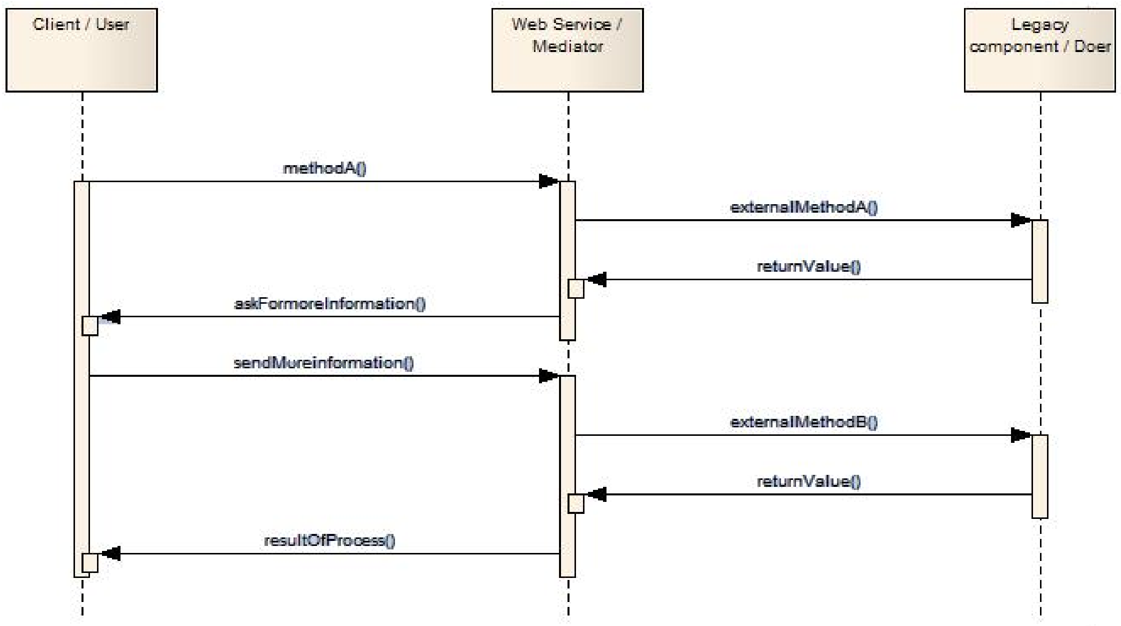


**Figure 17 Behaviour diagram example**

Figure 17 shows a simple example interaction diagram that specifies that after the client have requested that method "methodA" the component will ask for more information, which must be supplied and sent invoking the "sendMoreInformation" method. (The life-line "Legacy component" is not necessary for our purpose, since the requests from the client will be mimicked to method calls by the service towards the external component).

One way specified by [7] to attach this behavioral information uniquely for this scenario is to attach it to the service contract between the participants. A service contract is modeled using SoaML modeled as Figure 18 show.

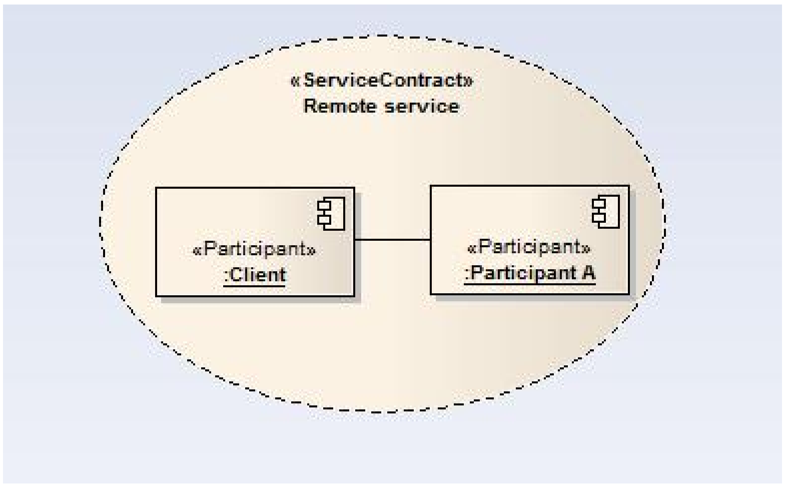


**Figure 18 A Service contract**

### 5.3.2 SOA AND SAAS REQUIREMENTS

Most of the SOA requirements are met by following the MILAS methodology up to this point. Requirements such as agility, extensibility, federation, reusability, composability, autonoumity, vendor diversity and interoperability are met up to a certain level, while requirements such as QoS, and discoverability has not been dealt with yet.

*Discoverability:* Discoverability is about publishing your services to a common repository which makes it possible for external users to discover and implement your service. Services can be made discoverable within an organization as well as outside the organizations by using standard registers in which the service definition are stored. UDDI[67] is such a registry made for registration of web services. It is however not widely used, and many organizations use an implementation of a LDAP[68] registry, since that often already is established.

*QoS:* QoS (Quality of Service) is about several matters such as security, performance, availability and transactions. These matters are only partly covered in this thesis when intersection with other relevant issues. To cover them in detail would not be possible within the scope of a master thesis.

While the SOA requirements are met in the above steps, SaaS and Cloud Computing has another set of demands that are not that easily seen as automated and tool helped processes. While the multi-tenancy and security issues are something that can be handled by tools to come, the other requirements are more difficult to automate.

#### *5.3.2.1 Multi-Tenancy*

Maybe the most important aspect of designing for SaaS and the cloud is security. Unless the legacy system should have an appropriate security design that already take into account the challenges of running an application as SaaS in the cloud, such security design must be implemented now.

Such security concerns affect the whole application and some thoughts must be made regarding how it should be implemented in the design. There are two main solutions to the security problem which both should be present. One is to add in security with public and private keys to control access to the different services that a customer can and cannot use. Another is to add client identification into the necessary data tables to build in means to prevent one client from looking into data that belong to another client.

The last approach is probably the easiest to implement. By adding a unique client id to the tables that are searched on directly by an user, one can easily restrict that user from getting access to table rows that does not have his or hers client id. A user accessing a for instance a CRM (Customer Relationship Management) system in the cloud would typically do search in the customer table and retrieve data related to that customer. By adding a client id into the customer table and add that id in all the searches on the customer table one will ensure that a user can only access data belonging to his or hers customers. A prerequisite is that the database is well designed.

[67] UDDI (Universal Description, Discovery and Integration) is a platform-independent, Extensible Markup Language (XML)-based registry for businesses worldwide to list themselves on the Internet.
[68] LDAP (Lightweight Directory Access Protocol) is an application protocol for querying and modifying directory services running over TCP/IP

If the migration process demands a restructure or move of a database to another platform, a client id can be inserted at migration time to prevent any extra steps with testing and verification. If not, then this step will imply some overhead, but a fairly small one.

The other requirements scalability, availability, performance, extensible data model, customizable GUI, customizable business logic and workflow is not covered in this thesis, but must be addressed before the application can be put into the cloud. All these factors can generate quite a lot of redesign work and solutions should be found now. We have gained a well understanding of the system and redesigned it to fulfill the basic SOA principles, and new design patterns and concepts are ready to be added to our model.

### 5.3.3 SERVICE MEDIATION

When modernizing legacy systems towards SOA and Cloud Computing, new possibilities for creating functionality will arise. By having opened up the application with services, combining these services internally or with externally available services, one can compose new services with new and enhanced functionalities. This is one of the strengths of SOA and combining this with model driven composition can ease the burden for developers since they do not have to know the details of the implementation.

There is a challenge to composition of services that are not addressed in UML or SoaML. When trying to have two or more services to communicate, it is likely that the exchange formats differs, and the need to transform a message from one format to another will appear.

To solve this challenge a service mediator can be inserted between the different services, as shown in Figure 19, which will perform the necessary transformation of the message that are being exchanged.

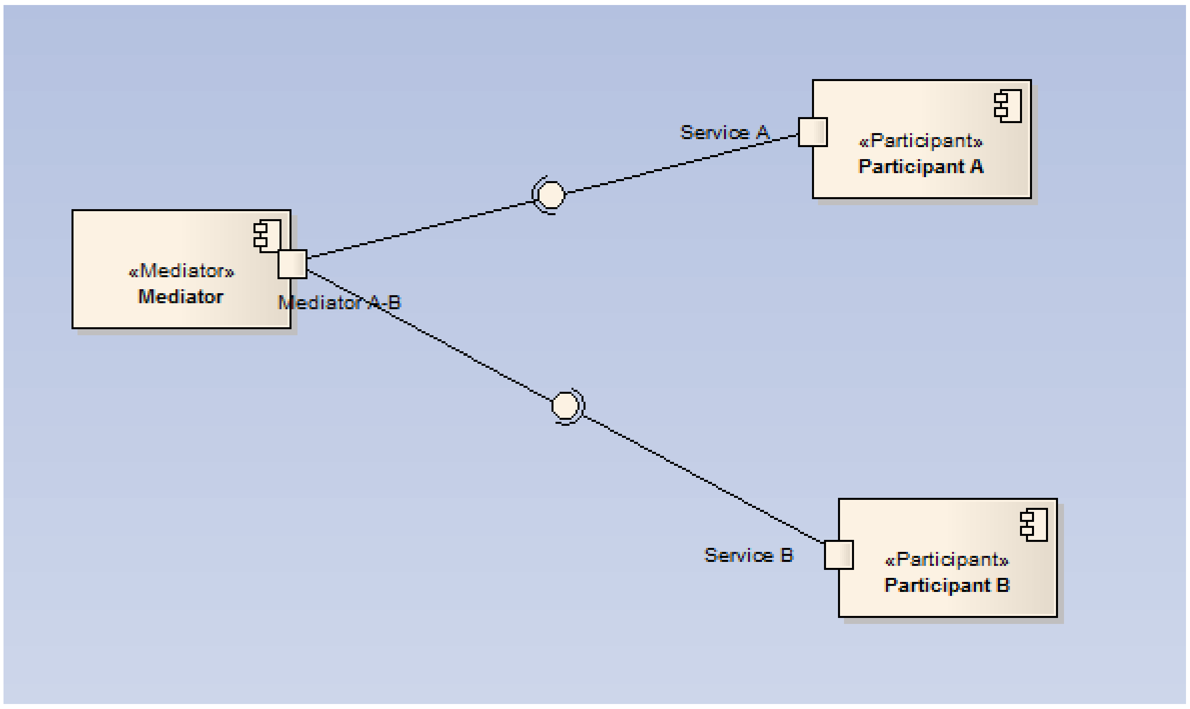


**Figure 19 A modeled mediator**

To transform the message from “Service A” to “Service B” in the figure, the mediator service needs to be capable of map the incoming data elements to the appropriate outgoing data elements. The transformation of data from one format to another can range from simple on to one mapping to complex rearranging of data. To accomplish the task a mapping tool is necessary which visualize the mappings, and help writing the transformation rules for later automatic generation of the mediator service.

Transformation rules could be written using Java directly to ensure richness of the language. By applying such a language one will ensure that the challenges one will face can be met. Typically challenges in a mapping situation is formatting, splitting and merging of fields, all which Java has a strong support for.

When visually mapping fields in a graphical editor, the mediator program should insert an mediator element between the services in question where the designer can add transformation rules. Rules that later should be transformed to source code residing in the mediator service.

When the time comes to generate the code supporting this the transformation program can look for any defined mediators for the services that are to communicate and generate the mediator service and change the call sequences so that the mediator are addressed during run time.

The definition of such a mediator tool could be an extension to SoaML, making it an universally standard and thus increasing the possibility for the tool to be developed and possible to implement in most UML designer tools.

## 5.4 SELECT TARGET PLATFORM AND DEPLOY

This step involves selecting appropriate technology for the modernized system and technology that can run alongside and communicate with the legacy system. The second part is crucial for a Chicken Little approach success.

However when doing a model based modernization one will generate the target architecture as a PIM (Platform Independent Model), and the selection of target platform can be done later in the project.

### 5.4.1 SOA AND SAAS

If the system to be deployed as an in-house system, as the case is for the Progress legacy software then a traditional platform for multi-layered database applications can be selected. Such a platform selection consists of evaluating suitable operating systems, database systems, application servers and web servers. If communication with other systems are high, or expected to increase in volume suitable messaging system, enterprise buses and systems for orchestrating and managing composable services must be evaluated when needed.

To explore and evaluate the different platforms in general terms are outside the scope of this thesis, but some considerations are done for the individual case-studies.

### 5.4.2 SAAS AND CLOUD COMPUTING

For the OSCAR case and for many scientific applications alike, and for business applications where the goal is to reach as many users as possible and the effort of maintaining such systems with respect to performance, scalability and availability is desirable to keep to a minimum, a Cloud Computing platform should be evaluated.

As the OOPSLA[69] workshop paper “Migrating Legacy Applications to the Service Cloud” [22] states, the most popular Cloud Computing platforms today are: Sun Open Cloud Platform [23], Microsoft Azure [24], Amazon EC2[25], and Google App Engine [26]. The EU project RESERVOIR [27] is also a good choice for providing services on the cloud. A general approach to select and prepare for the Cloud will be described here.

To run an application on the cloud is more or less like renting a virtual server and managing code in other organization's platform. We can take other advantages of cloud technologies to make our application more scalable and effective to use. The generic processes of configuring and running a Web service on different clouds are quite similar. Take Sun Open Cloud Platform as an example, for which the implementation steps include [23]:

1) Choose the running environments from a pre-defined virtual machine images, like load balancer, web server, database server appliances;
2) Configure the running environments to make a custom image, like configuring the load balancer, uploading static content to storage cloud, loading web content from Web Server, deploying database server to generate dynamic content;
3) Program with the redesigned architecture to satisfy specific application requirements;
4) Choose a pattern that takes the images for each layer and deploys them. Also handle networking, security, and scalability issues.

With this kind of migration steps, it is possible to move the legacy systems to a Cloud Computing environment. The more self-contained the system is, or the less dependent on supportive applications like databases, report engines, communication protocols the system is, the easier it is to deploy the system on the Cloud. Careful considerations should be done before moving more complex systems to the Cloud, to see if the system is realizable with today’s Cloud technology.

[69] **OOPSLA** (Object-Oriented Programming, Systems, Languages & Applications) is an annual ACM conference.

# 6 MILAS FOR OSCAR

For the scientific legacy system which is a part of the SiSaS project, there are different requirements to solution than for the business legacy system. There is no demand that the scientific system should be rewritten into another language, on the contrary it is desirable that scientific software should undergo as little change as possible. This is basically because such reengineering could be very difficult, since the code written are done by highly professionals in the specific science, and therefore would be hard to understand for resources of mere IT-skills. Secondly the language which is used originally is one of the best for writing mathematical procedures and thus well suited for scientific programs.

The requirements still state that the software should be able to run as SaaS software with service architecture. To accommodate for this, a strategy to solve the challenge, is based on wrapping the interfaces of the legacy software in software used by the target architecture.

In addition this task is likely to be carried out several times, so an automated process as possible is desirable. The automation can be carried out in several stages of the process, but the thesis will focus on automatic transformation from code to model to model to code.

## 6.1 RECOVER AND UNDERSTAND LEGACY ARCHITECTURE

How much work needed to get an overview of the system depends on what is needed to be done with the same system. If the legacy software is performing a task that doesn't need to be split into modules or rewritten in any parts, then an overview of how the interfaces and environments works, could be sufficient.

As it is the OSCAR system consists basically of five components that together build up the application, as Figure 20 shows. The C++ component is a client with graphical user interface that starts a Fortran executable when the initial data are entered into the system. The access database is used to store information on different types of oil, which will carry an impact on the calculations done by the Fortran program. Most of the information the Fortran executable needs is stored in structured text files, that the executable imports when needed during processing. The C++ dialogue is updated with small intervals telling the progression of the calculations.

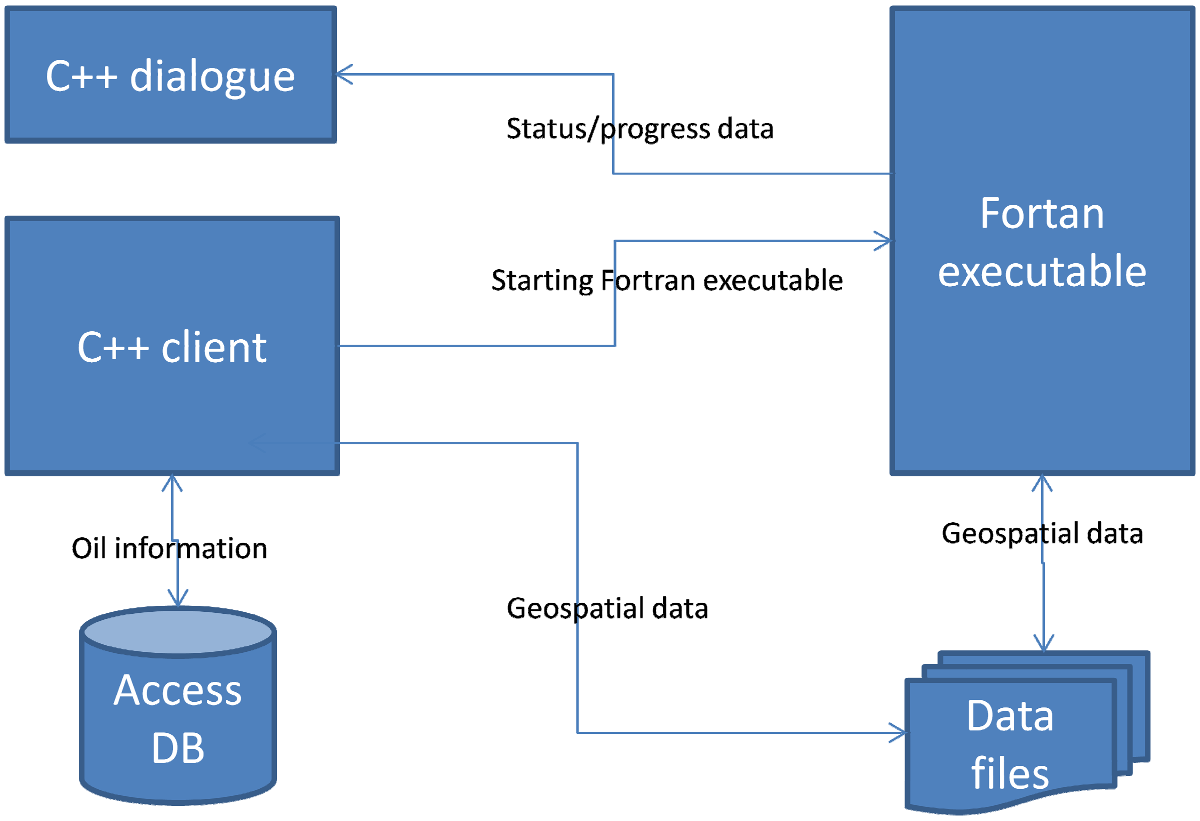


**Figure 20 OSCAR technical overview**

### 6.1.1 Recover to architectural representation

For the OSAR case there is no need to do this step. However if the Fortran part of the code were to be split up in sub modules the steps described in the MILAS methodology.

### 6.1.2 Analysis of legacy architecture

As for the step above, there is no need to do this step for OSCAR. However if the Fortran part of the code were to be split up in sub modules the steps described in the MILAS methodology should be followed.

### 6.1.3 Retrieving new and old requirements

The old requirements for the application are that it acts as it always has done. The new requirements are about fulfilling SOA and SaaS requirements that will be handled by the surroundings.

### 6.1.4 Select strategy

Depending on the result from step one, a strategy can be selected. For our case the strategy will be to wrap the Fortran part of the legacy system into web services, so that the functionality of the legacy system can be reach through standard service interfaces that modern software knows how to communicate with.

## 6.2 MIGRATE TOWARDS SOA AND THE SAAS

For our purpose these steps are not necessary, since no changes to the legacy software are needed. However if the Fortran part of the code were to be split up in sub modules the steps described in the MILAS methodology should be followed.

## 6.3 SERVICE MODELING AND MDA

This step is the fundamental step for this case-study, to make the legacy system available as a SaaS running in the cloud. By using the SoaML standard as basis for the models generated in this step we will ensure that the model is compliant to the SOA requirements.

The focus will be to make the Fortran part of the OSCAR application running as a service, and to accomplish this some modeling must be done.

First the initial service is modeled using the SoaML profile.

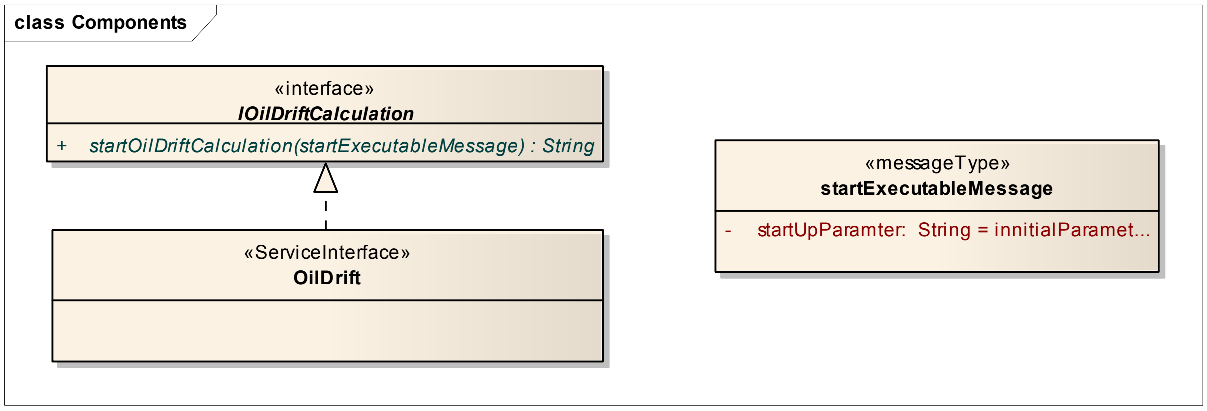


**Figure 21: An simple service model for OSCAR using UML with SoaML**

As the image in Figure 21 shows the service definition model is very simple to make. One element using the SoaML "ServiceInterface" stereotype realizing a standard interface with one operation. This and the message definition is all there is to with SoaML, since all we want to do is to start the executable. For our purpose this could have been enough information for the transformation program if it was a normal service, but following the SoaML standard, and to ensure reusability the model should define the participants and their contract.

The below figures shows a contract between a user of the system, and the service represented as a participant in fFigure 22 plus an owned behavior that is an interaction diagram Figure 23 that defines how the two participants should interact.

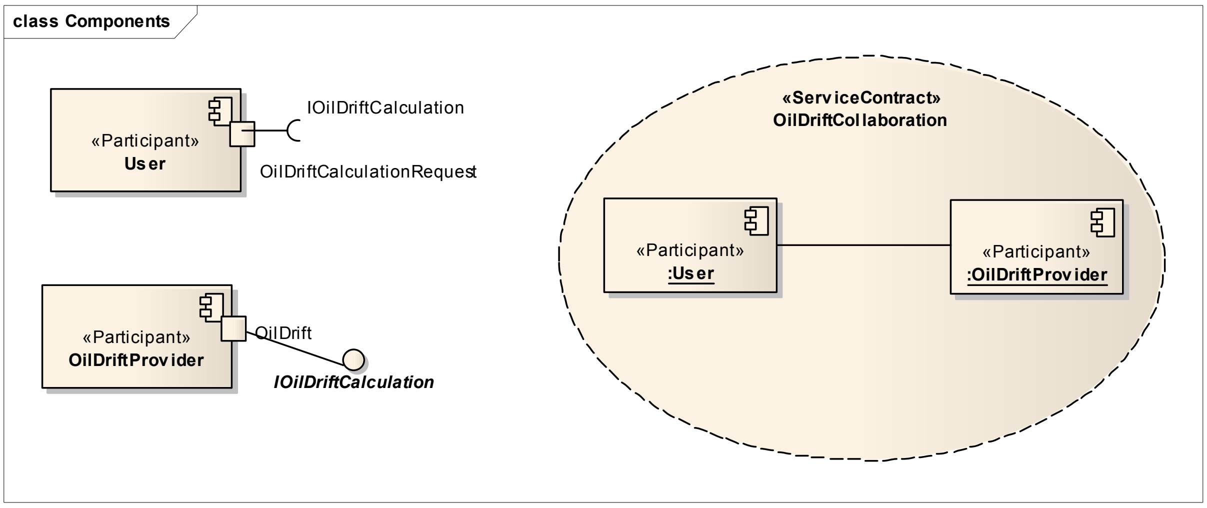


**Figure 22 Contract modeling using UML with SoaML**

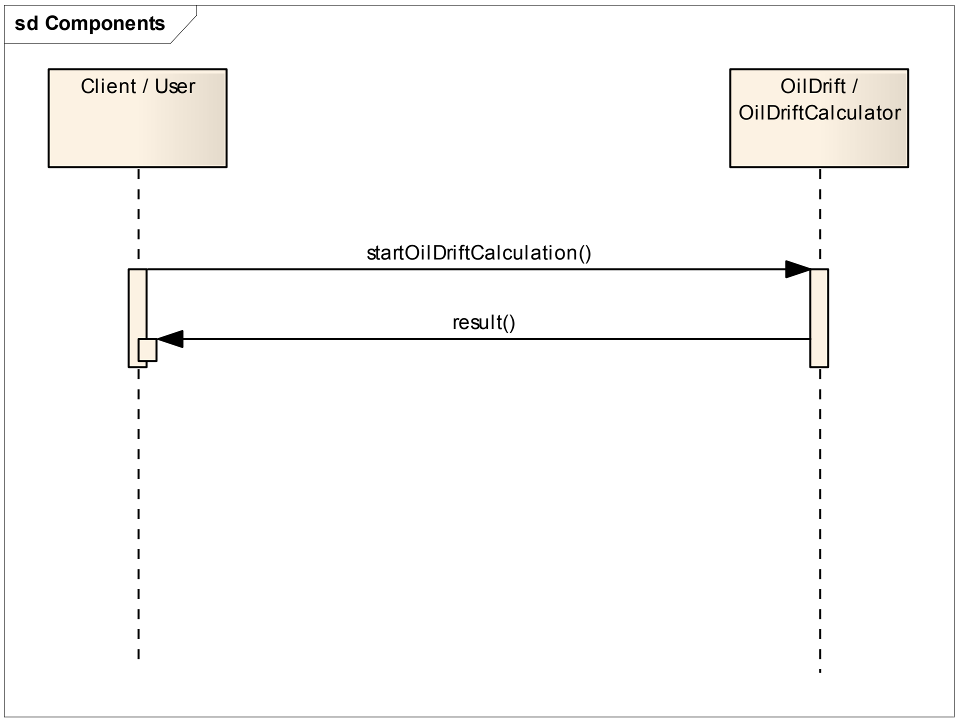
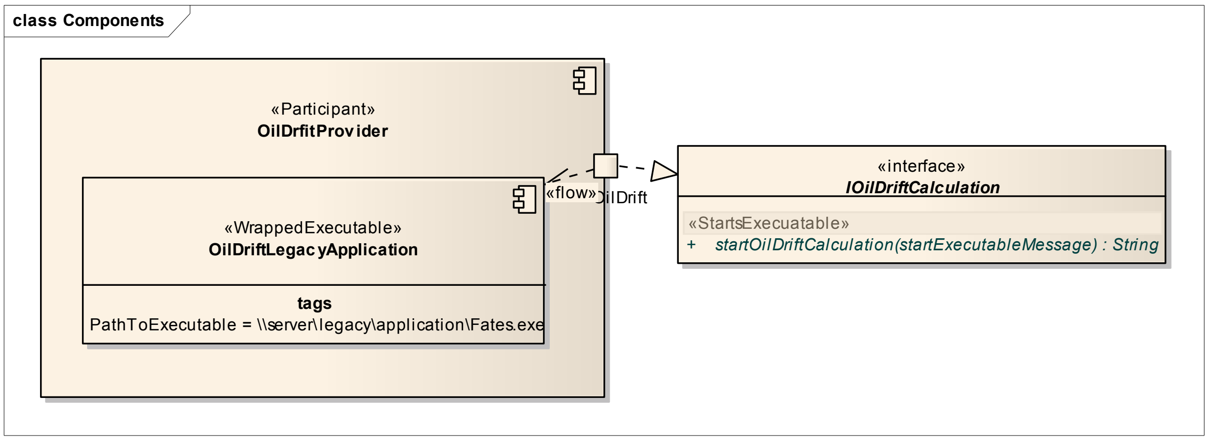


**Figure 23 Interaction model showing the interaction between the client and the service**

This would be enough to make an transformation form the service model to WSDL or POJO objects. However there is no information telling us how to implement the Fortran component into the "OilDrift" web service. To accomplish this we need the PIM4Wrapping UML profile as described in the MILAS methodology chapter 5.3.

### 6.3.1 SOA, MDA & Wrapping (federation)

With the PIM4Wrapping profile added to the model we can put the legacy component into the model and stereotype it for recognition. We add an component to the already defined participant "OilDriftProvider" and steretype it with "WrappedExecutable" and connect it to the port with an information flow relationship. To make sure we can find the executable we add the tagged value "PathToExecutable" as well. The start-up operation must be stereoyped with "StartsExecutable". Figure 24 below show the new participant.



class Components
«Participant»
OilDrfitProvider
«WrappedExecutable»
OilDriftLegacyApplication
tags
PathToExecutable = \\server\legacy\application\Fates.exe
«flow»
OilDrift
«interface»
IOilDriftCalculation
«StartsExecuatable»
+ startOilDriftCalculation(startExecutableMessage) : String

**Figure 24 Wrapped OilDrift legacy application**

Now we have the needed information to transform the model to code that can generate the service and start up the legacy application from within the service.

#### *6.3.1.1 File transfers*

Often scientific systems rely on information in files of various types and sizes to hold necessary data that are needed in their calculations, as it is for the OSCAR case. The information in these files can be anything from simple control parameters to geographical data describing the area in which the calculations take place. Since a file or a file system is not part of the base UML data types, it is necessary to especially mark these operations as well, to later be able to generate appropriate code. One should also have the possibility to differ between three types of transfer. Alternatively make sure that all three types can be carried out within the same operation. The three types are;

- Transferring one specific file.
- Transferring all files in a directory.
- Transferring all files in a directory and its sub-directories.

For the purpose of the case-study, the last option needs to be implemented. The Fortran executable expects to find certain files stored under a specified directory. In the root directory configuration files are stored while the data files are stored in several sub directories. In other words the generated web service needs a way to transfer client generated files to a location that are unique for the Fortran executable in question. The uniqueness of the location is imperative so that different users will not share the same files, something that will cause the legacy application to fail.

By stereotyping the data type to "TransferAllFiles" the transformation program can generate the appropriate methods to perform this task. The value of the input parameter of the message must then contain a valid directory containing all necessary files and reachable from the web service' server. Files could be transferred as attachments, but in our case there are many files to be transferred and the transfer will be binary. The generated service code must also make sure that the files are transferred into a unique directory for the session and that the legacy application is started up relatively to that directory so the files will be found by the legacy application. The uniqueness will fulfill the multi-tenancy requirements for data treatment.

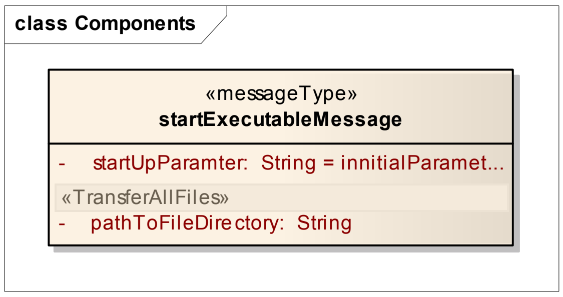

class Components
«messageType»
startExecutableMessage
- startUpParamter: String = innitialParamet...
«TransferAllFiles»
- pathToFileDirectory: String

**Figure 25 Message with steretyped data element**

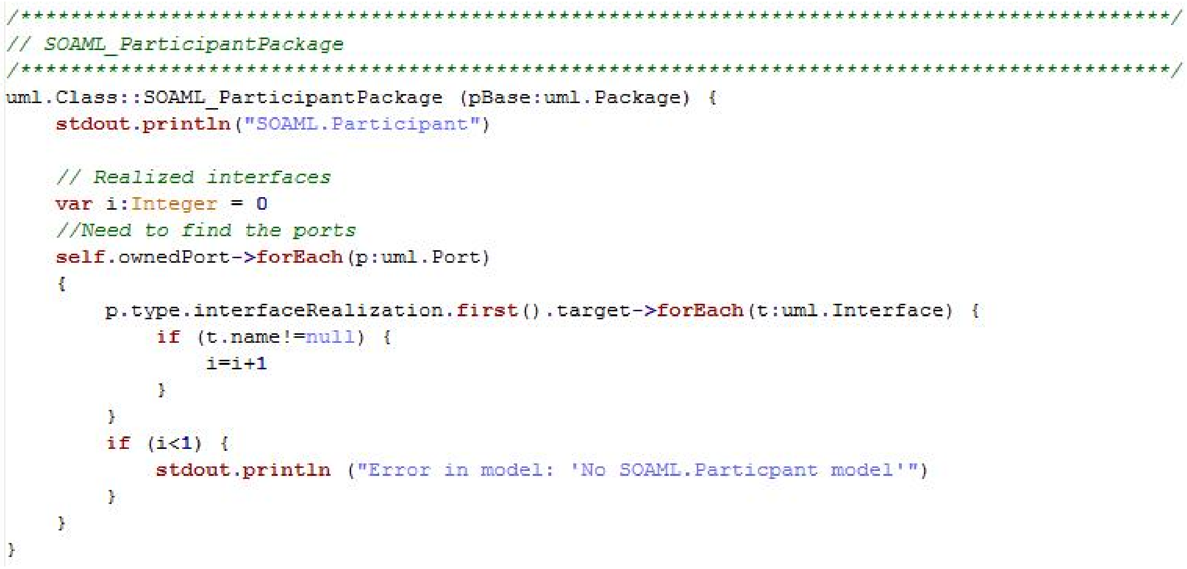
### *6.3.1.2 Write transformation*

A natural transformation for the model could be a transformation to a WSDL[70] **(**Web Services Description Language) file, but for the case-study we will transform the model into POJO[71] (Plain Old Java Object) code. This is done to utilize existing tools made for the eclipse[72] platform. Since the eclipse platform has built-in wizards that can generate web services from POJO code using web frameworks. The generated web services will then define the WSDL files. The following chapters explains how to write the transformations.

### *6.3.1.3 General transformation*

Since there can be several different services modeled within the same model space, one should be given the option to specify the package[73] or a specific an element before performing any transformations. Apart from that the transformation program should be able to retrieve necessary information from anywhere within the model. This is of importance to allow for reuse of other model elements such as general classes, and other dependent architecture artifacts.

[70] WSDL (Web Services Description Language) http://www.w3.org/TR/wsdl
[71] ***POJO*** is an acronym for **Plain Old Java Object**. The name is used to emphasize that the object in question is an ordinary Java Object, not a special object, and in particular not an Enterprise JavaBean (especially before EJB 3). The term was coined by Martin Fowler, Rebecca Parsons and Josh MacKenzie in September 2000
[72] http://www.eclipse.org/
[73]Package, in computing, a type of file format where software programs and installation material is grouped together.



```
/***************************************************************************************/
// SOAML_ParticipantPackage
/***************************************************************************************/
uml.Class::SOAML_ParticipantPackage (pBase:uml.Package) {
    stdout.println("SOAML.Participant")

    // Realized interfaces
    var i:Integer = 0
    //Need to find the ports
    self.ownedPort->forEach(p:uml.Port)
    {
        p.type.interfaceRealization.first().target->forEach(t:uml.Interface) {
            if (t.name!=null) {
                i=i+1
            }
        }
        if (i<1) {
            stdout.println ("Error in model: 'No SOAML.Particpant model'")
        }
    }
}
```

**Figure 26 MOFScript example, finding interfaces to a participant**

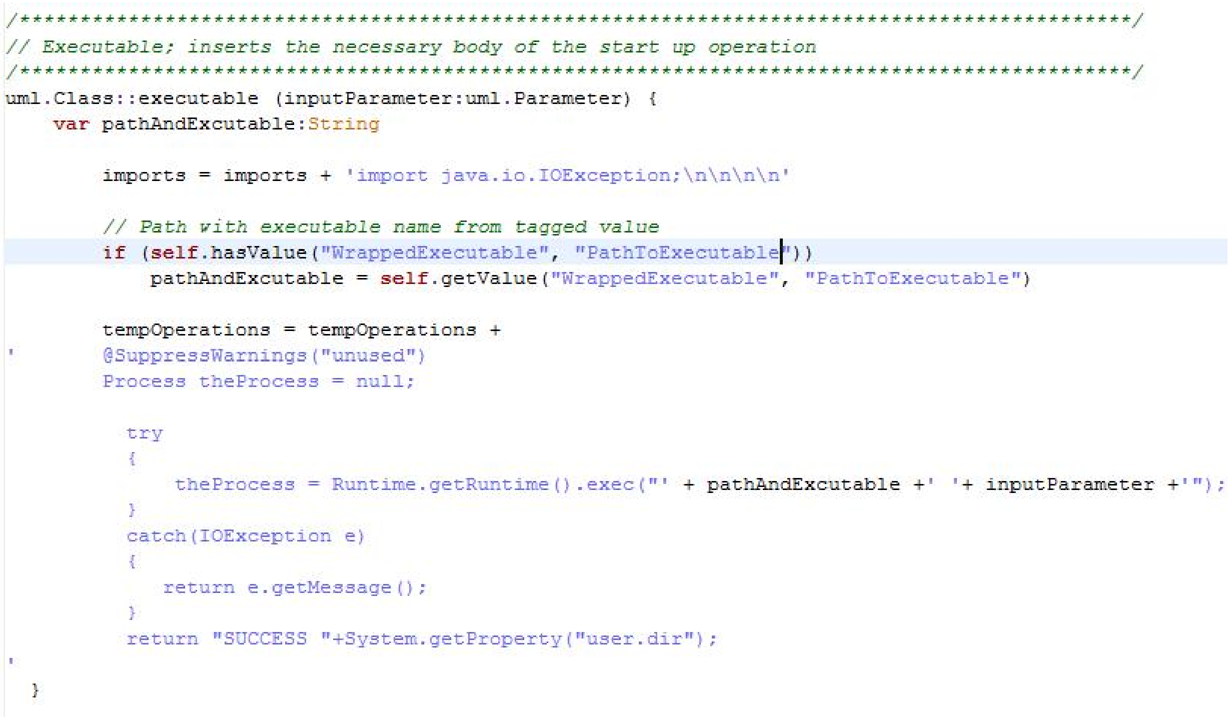
In the SoaML specification [7] there are multiple elements that can be thought of as a service definition. One have the service definition itself, stereotyped with “ServiceInterface”, the service specification implementing the service stereotyped with “ServiceSpecification”, a participant implementing the service stereotyped “Participant” or collaboration stereotyped with “ServiceContract”. The best choice as a starting point for transformation can be debated and are dependent on how collaborations among different services play a role or not. For our purpose we have collaboration between two participants, the user and the control service and we will use the collaboration (Figure 22) as a starting point.

The figure above shows an example MOFScript that having found a participant loops through the realized interface for the participant. For every interface found POJO code will be generated for later writing to file.

#### *6.3.1.4 Special transformation*

Since we want to wrap the scientific code in POJO, we need to transform the elements stereotyped by the PIM4Wrapping profile, like start execution operation and file transfer operation into more than a skeleton. One needs to fill in code chunks capable of performing along the lines of the requirements given by a SaaS platform.

In Figure 27 one can see a MOFScript code-piece that implements the body of the start legacy application method. As one can see the path and executable name are retrieved from the tag value of the executable component stereotyped “WrappedExecutable”.



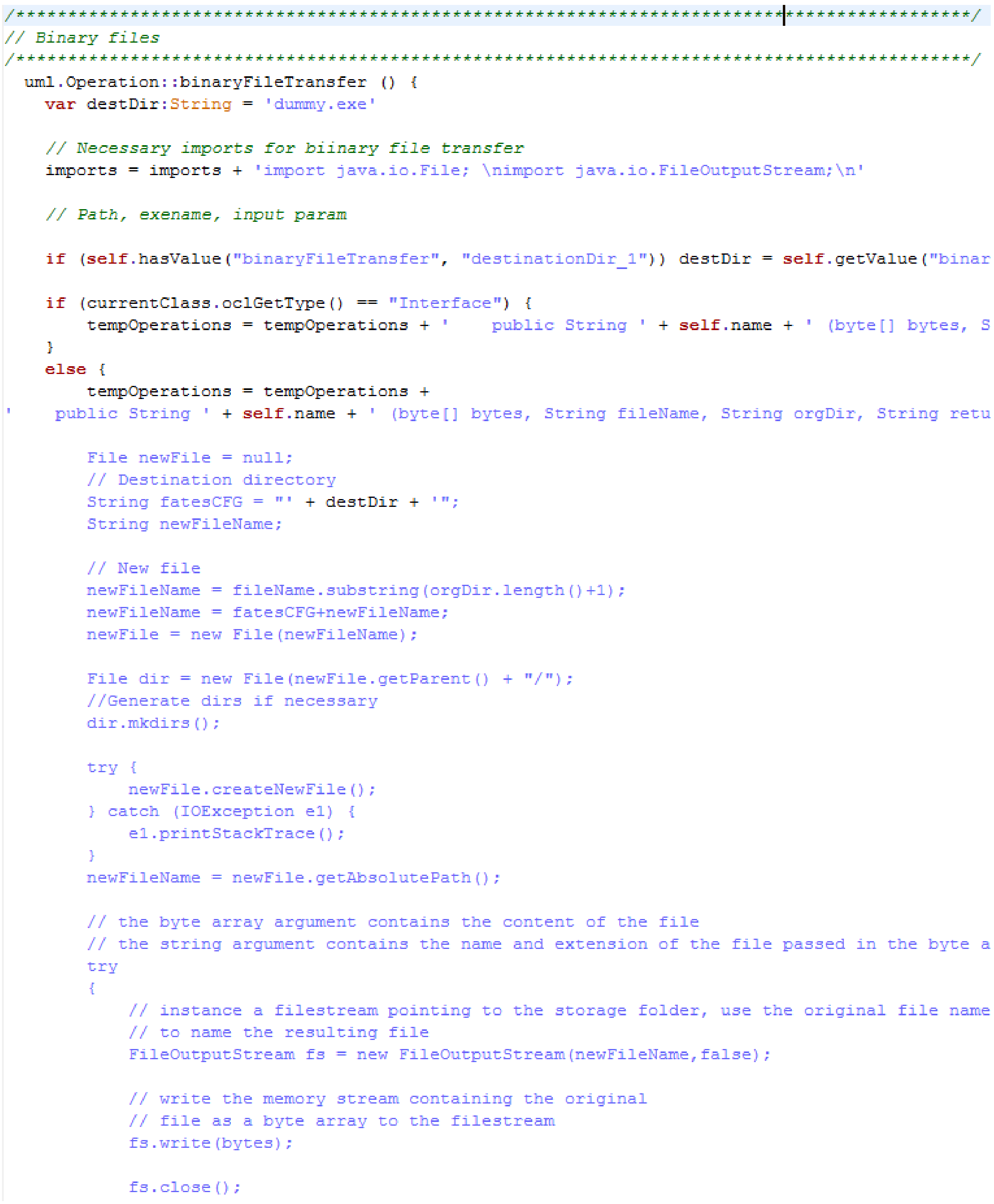
```
/******************************************************************************************/
// Executable: inserts the necessary body of the start up operation
/******************************************************************************************/
uml.Class::executable (inputParameter:uml.Parameter) {
    var pathAndExcutable:String

        imports = imports + 'import java.io.IOException;\n\n\n\n'

        // Path with executable name from tagged value
        if (self.hasValue("WrappedExecutable", "PathToExecutable"))
            pathAndExcutable = self.getValue("WrappedExecutable", "PathToExecutable")

        tempOperations = tempOperations +
'       @SuppressWarnings("unused")
        Process theProcess = null;

          try
          {
              theProcess = Runtime.getRuntime().exec("' + pathAndExcutable +' '+ inputParameter +'");
          }
          catch(IOException e)
          {
             return e.getMessage();
          }
          return "SUCCESS "+System.getProperty("user.dir");
'
  }
```

**Figure 27 Sample MOFScript code generating the body of the start up method.**

To generate the method necessary for file transferring a similar approach is taken. After detecting the input parameter of the base message with stereotype “TransferAllFiles” the method with body of the transfer files operation can be generated.

To prove the concept in the case-study the file transfer operation were simplified to retrieve one file at the time, letting the client loop through any subdirectories to ensure that all files were transferred to the destination folder. This is not unlikely the same solution as one would select for the real implementation. This is because it is not easy, and not advisable to let applications on the internet have access to ones private file structure. In the figure below is a sample from the MOFscript generating the transfer file method of the web service, or as for our case, the receive file method.

After the POJO’ have been generated one can use Eclipse and create a new web service project and import the generated code. Select the class that is meant to be the web service and use the built in wizards to generate a web service automatically by eclipse. To test the web service, use the generated test client or build test client according to ones liking.



```
/*****************************************************************************************/
// Binary files
/*****************************************************************************************/
  uml.Operation::binaryFileTransfer () {
    var destDir:String = 'dummy.exe'

    // Necessary imports for biinary file transfer
    imports = imports + 'import java.io.File; \nimport java.io.FileOutputStream;\n'

    // Path, exename, input param

    if (self.hasValue("binaryFileTransfer", "destinationDir_1")) destDir = self.getValue("binar

    if (currentClass.oclGetType() == "Interface") {
        tempOperations = tempOperations + '    public String ' + self.name + ' (byte[] bytes, S
    }
    else {
        tempOperations = tempOperations +
'    public String ' + self.name + ' (byte[] bytes, String fileName, String orgDir, String retu

        File newFile = null;
        // Destination directory
        String fatesCFG = "' + destDir + '";
        String newFileName;

        // New file
        newFileName = fileName.substring(orgDir.length()+1);
        newFileName = fatesCFG+newFileName;
        newFile = new File(newFileName);

        File dir = new File(newFile.getParent() + "/");
        //Generate dirs if necessary
        dir.mkdirs();

        try {
            newFile.createNewFile();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        newFileName = newFile.getAbsolutePath();

        // the byte array argument contains the content of the file
        // the string argument contains the name and extension of the file passed in the byte a
        try
        {
            // instance a filestream pointing to the storage folder, use the original file name
            // to name the resulting file
            FileOutputStream fs = new FileOutputStream(newFileName,false);

            // write the memory stream containing the original
            // file as a byte array to the filestream
            fs.write(bytes);

            fs.close();
```

**Figure 28 MOFScript sample showing file transfer method**

### 6.3.2 SOA and SaaS requirements

As mentioned in the MILAS methodology, most of the SOA requirements are met by following the MILAS methodology up to this point.

#### *6.3.2.1 Multi-Tenancy*

The multi-tenancy part of the OSCAR case is covered by ensuring that the configuration and parameter files that the Fortran executable needs are stored in a unique location for every run of the program. This means that a new directory structure must be created each time the Fortran code is executed and that that directory structure is isolated from other users (tenants).

This can easily be achieved since the target location of the files is an input parameter to the executable, and controlled by the web service itself.

### 6.3.3 Service Mediation

Service mediation was not a part of the case-study, and not an option in these early steps for the OSCAR case. Some consideration is however covered in the MILAS methodology's service mediation chapter.

## 6.4 Select Target Platform and deploy

For the OSCAR case a suitable platform could be an existing in-house platform, and then expose the web service interfaces out so it can be used by anyone with access to the site. This is probably the easiest approach since the platform of choosing will be a known one to the organization.

A hosting company would also fit as a platform for the OSCAR case, and the burden of scalability, performance and availability requirements can be lifted to the hosting company.

A better alternative, at least if performance and scalability requirements are high or expected to be high is to deploy the application on a Cloud Computing platform as described in previous chapters. With this as a platform the SaaS requirements such as scalability, performance and availability will have considerable potential since the resources are virtually unlimited.

Different popular cloud platforms are compared in [28]. We could see that it is easier to move a legacy system to Amazon EC2 than to Google App Engine (GAE). Amazon EC2 nearly supports most of the environment, applications, programming languages, runtimes, database servers etc. It offers much more freedom to developers. While with GAE, developers have to use either Python or a JVM-language (like Java), and also have to use the database system provided by Google itself, and communicate with Google database query language. Some tools like Cloud Foundry[74] are used to support the process of migrating legacy systems to Cloud more easily and effectively.

The Sun Open Cloud Platform would be interesting to test the application on, especially since the analysis done by Zach Hill and Marty Humphrey in [29] shows that the Amazon EC2 has some performance problems regarding high computing software.

[74] Cloud Foundry, for details see http://www.cloudfoundry.com/

Unfortunately there were not time enough to test the OSCAR case on a Cloud Computing platform, something that would be very interesting with respect both to clarify the complexity of such migration, and especially to see how the performance of the application would be.

## 6.5 Conclusion

Following the steps in the MILAS methodology for the OSCAR case results in a new application that is wrapped in a service design and deployed as a web service that can be accessed from anywhere on the internet and run as SaaS either on in-house platforms or hosting platform or on the cloud. The advantage of this methodology is that it is highly reusable, guarantees that standards are met and understandable for people that are not experts in developing services. There are probably thousands of legacy scientific applications out there that could benefit from this methodology and save a lot of man hours and investment in expertise.

# 7 MILAS FOR PROGRESS 4GL

The ideal goal for the modernization of k90, the Progress 4GL system, is to reach an open, well structured, reusable, serviceable, available, maintainable system. To reach this goal the system must undergo a lot of transformations in architecture, and a new development language must be selected. Even though Progress software has products that can help a modernization approach towards openness and SOA, one problem remains for the organization, and that is available resources. Resources in Norway that are familiar with Progress software products are scarce. To make sure that resources are available a new development language must be selected.

In addition, the development is until recently based on character based user interface with a client server technology, which mixes user interface, business logic and data access in one and the same programming space.

For these reasons the organization wants to modernize their old legacy code.

## 7.1 RECOVER AND UNDERSTAND LEGACY ARCHITECTURE

To get an overview of the system we had access to resources who wrote it, several current users and the source code and test facilities. So to gain an overview of the system were fairly easy. Unfortunately the next step turned out to be a problem. There did not exist any open source or freeware that could help extracting the architecture out of the source code and obtain an AST based on KDM or not. There exist a couple of companies that could do the job and had experience with Progress, but that was not an option to be taken at the time.

### 7.1.1 RECOVER AND UNDERSTAND LEGACY ARCHITECTURE

Ideally one should have had access to one of the major contributors of modernization software that supports the modernization of Progress 4GL, unfortunately these software products are highly commercial, and so no such test or trial can be carried out. The different tools that is available for testing has unfortunately no support for Progress 4GL either.

Gaining an understanding of the existing legacy system is therefore mostly a manual process. To gain a proper understanding some code reading and look-ups in the database must be carried out. Fortunately the 4GL is easy to read compared to many other languages, since the syntax is more academically laid-out and the 4GL commands more course grained than for the normal 3GL languages.

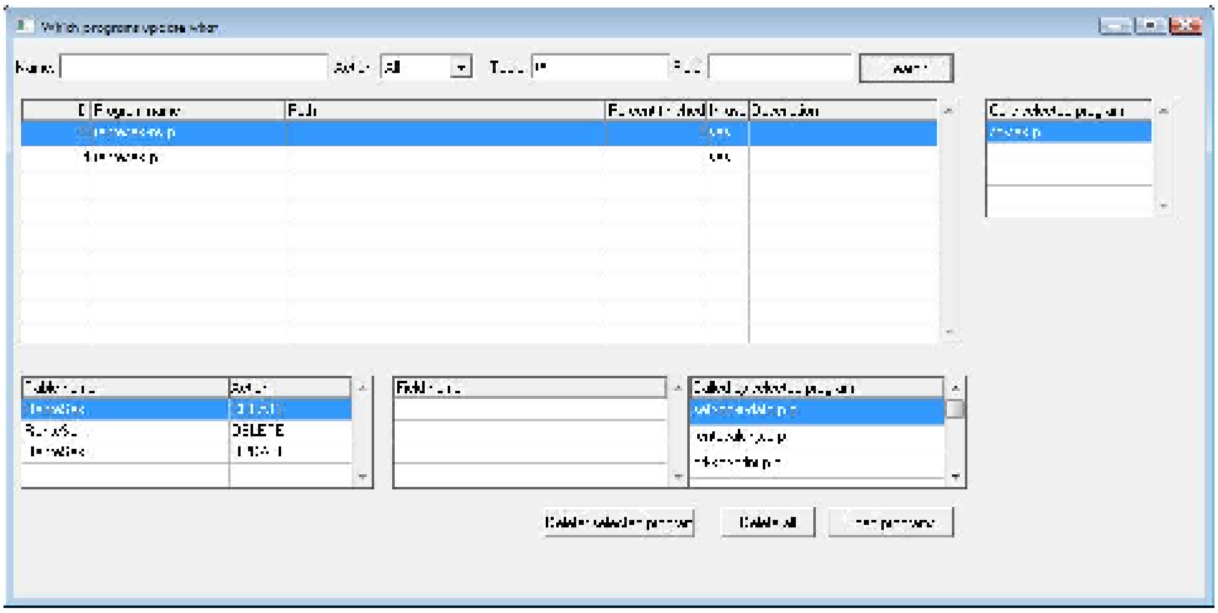
It would have been desirable to extract an AST from Progress 4GL, to be able to use the functionalities of the different modernization tools that exist. But Progress 4GL does not seem to have been a widespread language enough for the modernization companies and projects to invest time in this.

### 7.1.2 ANALYSIS OF LEGACY ARCHITECTURE

Analysis of the architecture had for most parts to be done manually. Some analysis programs were built to help getting overview over certain aspects of the architecture. One tool where built to look into where in the source code updates and reads from the Progress RDBMS took place.

The Progress 4GL code, at least the ones written during the 80's and early 90's, mix user interface, business-logic and data logic in one and the same procedures. Because of this to explicit distinguish the entities from the rest of the code can be a tidy process. Fortunately Progress can create a cross reference document for every compiled procedure, a structured text file with the same name as the procedure file, except that it has the file extension XREF. This file can then be investigated to find every reads, updates and creations of database tables and fields. Since it is well structured, it is a fast operation to generate an automatic analyze of the XREF file and put the result into some form of reference list that can be search on later. Figure 29 Analyse program for Progress 4GL

In the case study a program (Figure 29) where made that lists all source code files and their respective reads, updates and creations to the database. With some simple search functionality one could with a mouse-click view every place where a table or a table field where updated. This is a great help when designing the new data entity layer, since one can easily find all the necessary update and creation criteria that exists in the old system.


Which programs update what

DELETE

Figure 29 Analyse program for Progress 4GL

In addition the XREF file contains information regarding which other procedures that are called from the current procedure, thus making it possible to add in functionality that displayed static call graphs.

One substational gain done by this was the fast discovery over where writes to the database where. If for instance one wondered how many places a table was written to the database, one could just write the name of table in the "Table" field in the program window and retrieve a list over all procedures containing such writes. This was to great help when understanding how things worked.

### 7.1.3 RETRIEVING NEW AND OLD REQUIREMENTS

New requirements in this context are requirements to the applications in forms of functionality and processes.

New requirements are retrieved from the business management and super users of the legacy system. By letting they put the requirements as business process diagram using BPMN we can link the different tasks to components in the target architecture later, to verify that the requirements are met.

Old requirements are retrieved in a combination of exploring the legacy test application, talk to the experts on the system, and by extracting different type of information from the system using specially build tools.

To better get an overview I developed a new tool. The tool focused on getting the business logic and process out of the system by reading and analyzing the source code. The output from this process where XMI files that could be imported into a modeling tool and displayed as activity diagrams that reflected the flow in the source code and the static process flow. The tool read the syntax and built a proprietary AST of the source code and transformed that into a XMI format that is readable for modeling tools using the XMI standard.

Below in Figure 30 an activity diagram automatically generated is shown. (The source code program trims characters that are not numbers away from a text string).

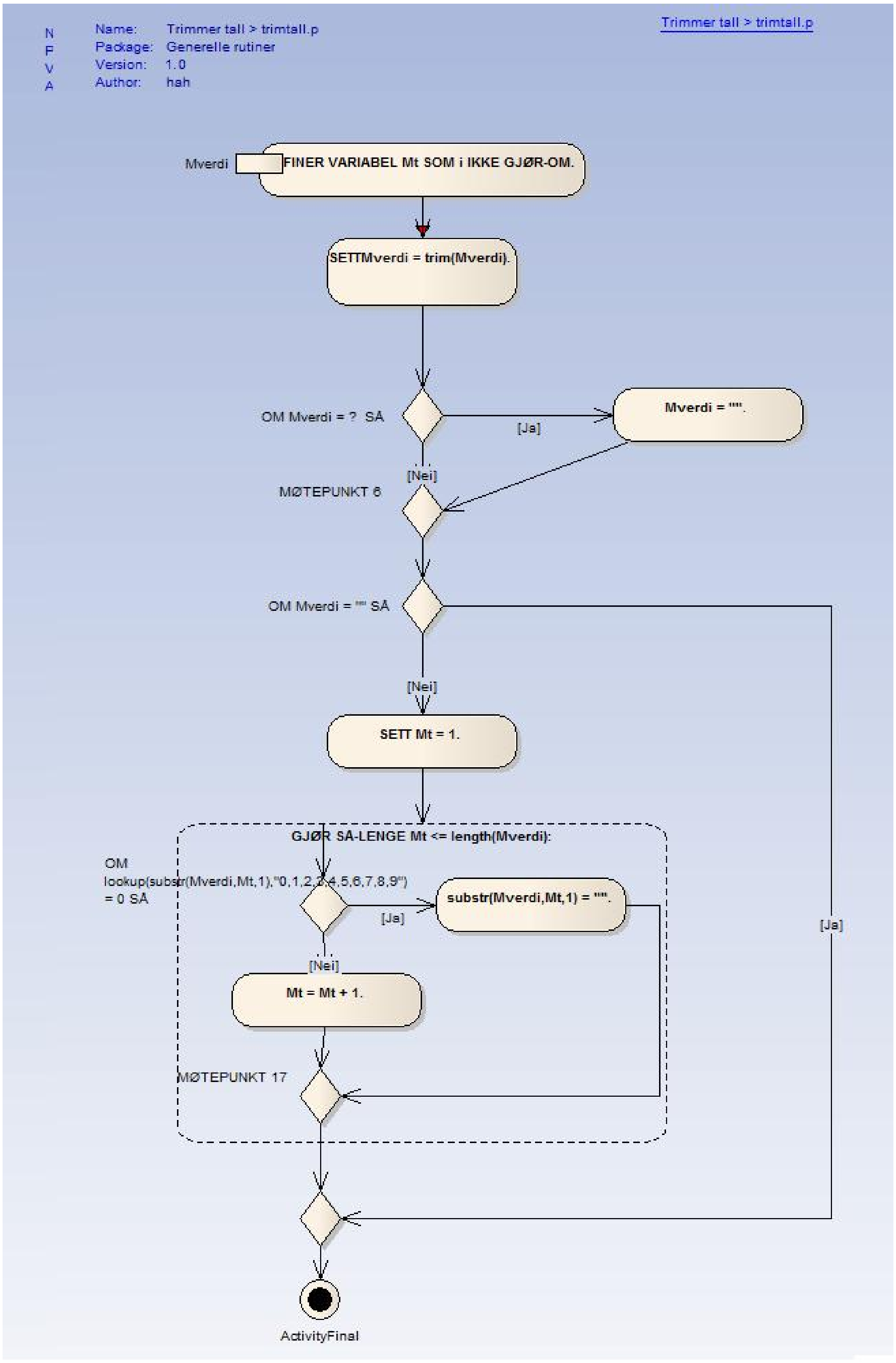

N
F
V
A
Name: Trimmer tall > trimtall.p
Package: Generelle rutiner
Version: 1.0
Author: hah
Trimmer tall > trimtall.p
Mverdi
FINER VARIABEL Mt SOM i IKKE GJØR-OM.
SETTMverdi = trim(Mverdi).
OM Mverdi = ? SÅ
[Ja]
Mverdi = "".
[Nei]
MØTEPUNKT 6
OM Mverdi = "" SÅ
[Nei]
SETT Mt = 1.
GJØR SÅ-LENGE Mt <= length(Mverdi):
OM lookup(substr(Mverdi,Mt,1),"0,1,2,3,4,5,6,7,8,9") = 0 SÅ
substr(Mverdi,Mt,1) = "".
[Ja]
[Nei]
Mt = Mt + 1.
MØTEPUNKT 17
[Ja]
ActivityFinal

**Figure 30 automatically generated activity diagram from a small Progress procrdure**

Unfortunately this was not such a good idea if the Progress procedures were long. The result of a medium long procedure could the look like something as shown in figure 31 below, which is not very readable or more understandable than the source code.

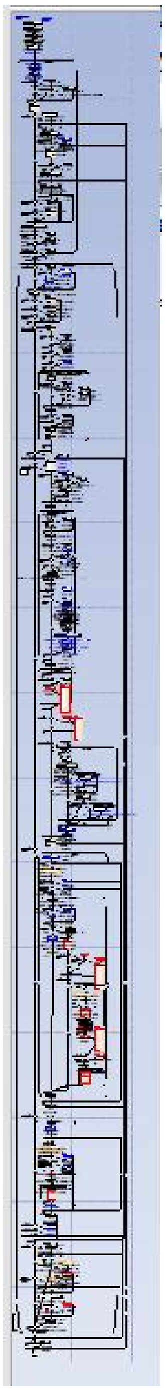


**Figure 31 A medium sized activity diagram reengineered from Progress 4GL procedure**

### 7.1.4 SELECT STRATEGY

Progress Software has in their software portfolio, both an application server[75], web server and has also introduced object oriented coding for their 4GL language. This makes it possible to select a modernization strategy that actually is a combination of several strategies.

The Chicken Little approach is fully feasible because there are several possibilities for communication between modernized code and legacy code. The Progress Application Server has the ability to generate a java proxy interface from the native code run within. This means that in some cases one can put legacy 4GL procedures directly in the application server and generate an interface that Java can interchange with. In addition the application server can generate web services based on the same 4GL procedures. This will require some modernization effort on the 4GL but is a fast way to service some functionality from the legacy system.

The legacy system is quite complex and the organization depends on from a day-to-day basis. A modernization effort following the Chicken Little approach must ensure that the application is up and running 24x7. This can be a challenge when the complex pieces of the code are to be modernized. If these pieces are too large, the risk of making error prone code is high.

When selecting a suitable sub-module for migration, there will be components in the legacy system that must communicate with the migrated software. A wrapper technique towards these components must be made.

Sometimes the sub modules selected will be larger than desired, and heighten the risk for failure. To avoid having to large pieces of software to migrate at a time, some of the functionality within the legacy code can also be candidates for wrapping techniques.

All in all a combination of migrating and wrapping are chosen to keep the projects down to manageable level.

## 7.2 MIGRATE TOWARDS SOA AND SAAS

### 7.2.1 GRADUALLY REARRANGE CODE TO MEET MODERN STANDARDS

An overall analysis of the existing system has been carried out, and together with examining existing documents one can derive the first set of classes for the system. As criteria for selection objects one should use the normal object selection criteria as external entities, things, occurrences or events, roles, organizational units, places and structures [30].

#### *7.2.1.1 Create main classes derived from the overall understanding.*

Progress is a procedural language, and the general structure of the code is to place every main procedure and/or functionality into procedural files. In the old days, the eighties, the main procedures or functions where put into separate files. Later Progress introduced internal procedures in their language, which made it possible to structure the code better.

In a procedural language there is also a tendency to write the code in a much more process (or procedural) manner that with object oriented coding.

[75] Progress software: http://web.progress.com/index.html

Even though one cannot expect the procedures to map one to one with objects, it is a good start to look into the possibility that the procedural files can define main classes in the modernized system. Internal procedures if used maps on the other hand better directly to operations.

Progress also comes with a RDMS system, which together with the 4GL language makes the core of Progress Software. The RDBMS is probably a much better candidate for classes than the procedural files, and should be the starting point. Some tables will map directly to classes, but the RDBMS tables are not build with object orientation in mind, and there is always tempting to put more into a table than would naturally belong in a similar class.

For our case a typical example would be the table named object. With object the system thinks of every possible person, company or client that has anything with debt collection to do with the organization. An object can in other words be a person with outstanding payments, a company with the same, or a client which the other owes money to. In addition an object can be jurisdictional instance, the government or the tax department. For an object oriented system this would typically be divided using polymorphism.

Another structure in the 4GL language to look for is temp-tables. These are often defined to hold data in-memory for fast processing or to organize data in a more convenient way for business-logic programming. Either way these can often reveal lack in good database design and should be considered when making the new class structure.

#### *7.2.1.2 Examine the flow of information between modules.*

By looking at parameters and globally shared data. This will help figuring out the boundaries and to obtain utility classes.

The XREF file produced from Progress compilation can be used in this context also. The file contains information regarding every call to external and internal procedures made from the procedure in question. By compiling every procedure with respect to the XREF file and then read and interpret the information, one can generate a static call-graph for further analysis.

#### *7.2.1.3 Organize the potential classes and objects found in step 1 and 2.*

This can be carried out using four immediate steps. First determine concrete classes by identifying the objects that are instances of those classes. Secondly identify duplicate classes and objects, thirdly classes with no objects can be potential abstract classes and finally re-evaluate objects that seems not to belong to any classes.

Typically one has found objects corresponding to the information needed in a procedure. This information can be information derived from several database tables and a conflict between the database and the information used will probably arise.

*Example:* A procedure retrieves information from a table used to store ad-hoc information and tables of more structured data like a customer table. Objects identified should be one, as a combination of the tables. While classes identified are two, one for each table. These conflicts must be resolved to satisfy good OO design.

Another issue that must be handled, is the likelihood of the database containing tables with a role field in them. This is typically a relational database design, but for OO it would initially be more correct to split the table (class) up in several classes with one master class, which the other classes inherit from.

#### *7.2.1.4 Identify attributes.*

To refine this structure a bit one can look at several elements in the code and naturally the database.

Since Progress RDBMS is a relational database and probably represents entities in a fairly good fashion, one can retrieve the database schema from the database and generate an UML 2.0 class model based on this information. In this way one will have a jump start in defining the entities for the new system.

In addition if there are defined temp-tables[76] in the source code a look into these might reveal new entities or entities that should be changed compared to the current database definition. A temp-table in Progress are used for different purposes, but mainly they exists for one of three reasons; To hold database tables in memory for faster processing, either in batch jobs or at the client side, to build new relations among existing data similar to views in standard SQL, or to create entirely new structures. The latter can be as an easy way out when doing changes to the system, without having to do serious changes to the database.

Global variables where a popular way of getting access to information needed throughout the system in the earlier days. These variables typically belong in utility classes, and should be moved into such.

#### *7.2.1.5 Identify methods.*

To identify methods, there are several approaches to take, due to the nature of the legacy source code. Five different scenarios might occur. First there are the modules, which can be regarded as a cohesive method, but also can involve functionality that belong in several different classes. Then one should examine existing functions and classify them as behavioral, associative, unassigned or virtual. The different types can be explained as;

**Behavioral**: Functions that uses or manipulate data that belong to one specific class.
**Associative**: functions that can belong to more than one class.
**Unassigned**: Functions that cannot easily be identified, belonging to any class.
**Virtual**: Functions that belongs in an abstract class, so the appropriate method can be executed at run time (polymorphism).

The Progress 4GL source code differs in structure, depending on when it was written. The oldest pieces of code are not structured into sub-procedures or functions, since the language did not have these terms in the old days. These pieces of code are structured in files, or as Progress calls them procedures, and every file is one block of code containing everything needed for the specific purpose. These files can be regarded as modules in Progress and must be analyzed to get an overview over potential methods. Newer pieces of code can be well designed procedures or files, with internal-procedures that can be examined and classified into the four different categories.

[76] Temp-table. A table that resides in memory only at run time, but are structured as a RDBMS table.

Discovering virtual functions can be done with respect to common "if" statements. There are several examples of similar procedures, doing the same with minor variations due to some type differences such as customer type.

#### *7.2.1.6 Decide on entity objects and control objects and UI objects. MVC (Model View Control)*

The discovering of the user interface of a character based Progress 4GL application could have been a straight forward operation if it weren't for one issue. At least in over case study a lot of the communication is done through messages. In this context messages means the plain old message that appears on the screen with a message text and an option for answering if necessary. This message construct of Progress 4GL makes it hard to distinguish plain messages, messages as a question, like "do you want to continue?", messages where data is updated, like "Enter in the new amount:", and messages functioning as menus with multiple choices.

There is at least possible to discover these artifacts and separate from the rest of the code.

#### *7.2.1.7 Re-evaluate unsigned functions.*

Since it is not desirable to have independent functions in an object model, these functions should be re-evaluated and fit into existing classes or new ones.

This step is not applicable for the business case-study with Progress 4GL.

#### *7.2.1.8 Look for patterns.*

Discovering patterns can save a lot of coding if different chunks of code really performs the same job, and therefore can be put into the same methods.

Patterns can be discovered in Progress 4GL especially by looking at the database statements. There would probably exist several similar statements for retrieving lists of data from tables, and also create and assign statements. On simple way to discover pattern is to do further analysis with respect to the XREF compilation output. In this file are updates and searches on database objects listed.

#### *7.2.1.9 Refine class definition, (and database structure).*

A more thorough examination on the found classes should be carried out. One should evaluate if attributes are specific to the class, or implementation specific. For instance array declarations should be inspected closer, in case these actually should be defined as classes themselves.

There is no array definition in Progress.

#### *7.2.1.10Move direct access to files or the database to entity classes, and replace the access them with method calls.*

In 4GL languages and in Progress 4GL there exists special proprietary ways to access the database. Progress Software delivers much more than just the 4GL, and from the start back in the seventies they delivered a relational database system along with the 4GL language. Because of this they have developed their own ways to manipulate data in the database directly from within the 4GL. The language provides a rich set of commands that are similar to the more known SQL. Instead of the familiar SELECT statement in SQL Progress supply two main commands for accessing the database. The ”FIND” command and the ”FOR” loop.

| | View | Control | Model | |
|---|---|---|---|---|
| FIND | | x | x | |
| FOR | | x | x | |
| CREATE | | x | x | |
| UPDATE | x | x | x | |

Update statement is a good representation of a 4GL statement. Update carry out functionality that normally would be split into the three different layers in a MVC designed system. Update retrieves information from the database records in question, displays it to the screen, and assigns them to database when the user leaves the statement. This statement must be divided and rewritten into the three MVC layers.

##### *7.2.1.11Build in transaction. If necessary transaction methodology should be taken into consideration and built into the system.*

Transactions must be handled in the new and modernized system. In Progress 4GL the transactions can be found either by explicit search on the “TRANSACTION” key-word, which will be attached to a block statement which again defines the scope of the transaction.

Unfortunately, this key-word is often left out by the programmer since Progress will automatically scope any transactions it handles. In other words, to detect where transactions are carried out, a search for the statements capable of doing transactions must be done. This would typically be the “CREATE”, “UPDATE” and “ASSIGN” statements, but also reads from the database can define a transaction block. If the read is carried out without the “NO-LOCK” key-word, then the read will be handled as an transactional read.

##### *7.2.1.12Determine class relationship. Relationship among the classes should be established. Especially inheritance and aggregations should be determined.*

The found classes and their relationships should be evaluated up against the existing relational database to see if there are candidates for restructuring the database. The existing database is the best starting point for establishing entity classes.

##### *7.2.1.13Re-evaluate any potential virtual functions.*

Not applicable for Progress 4GL.

### 7.2.2 CONTROL YOUR WORK

Every iteration on the above steps should be followed with analysis of the generated AST and UML models. Analysis should be carried out to ensure that the new system is manageable and reusable. Measuring the amount of cohesion and coupling helps determine the robustness of the new system.

## 7.3 Service Modeling and MDA

This step is important to make sure that the modernized system will open and flexible with respect to services. In the OSCAR case study mostly wrapping of existing software where necessary to make the system ready as a SaaS, for the Progress case however there will also be service models that are pure UML 2.0 and SoaML based.

Service modeling in this perspective is not that interesting since it is covered by the service wrapping techniques. Therefore this thesis will focus on wrapper techniques also for the k90 case study.

### 7.3.1 SOA, MDA & Wrapping (federation)

The same technique as for the OSCAR case will be applied here. The differences are that instead of wrapping an executable as for the OSCAR case we will now wrap a Progress procedure running on a Progress AppServer[77].

Information needed to wrap the Progress AppServer procedure is covered by applying a profile and message definitions as for the OSCAR case. To call a Progress procedure in Progress AppServer one need to know the following;

- Which element that is the external Progress procedure.
- The location (URL) of Progress AppServer.
- The name of the external Progress procedure.
- The input and output from the Progress procedure.

The external Progress procedure are recognized in the model by applying the stereotype "WrappedProgressAppServerProcedure", the rest of the information is stored in the tagged values that are added to the stereotype. The location (URL) is placed in the "RemoteMethodInvocationAddress" tag and must be initialized in the model. The name of the Progress procedure are placed in a tag named "RemoteMethodName" and prefixed "createPO" something expected by Progress AppServer. The operation that calls the Progress component must be stereotyped with "RunRMIComponent" to get the correct entrypoint when generating code. See Figure 32 for a visual explanation.

[77] Progress AppServer, for details see http://web.progress.com

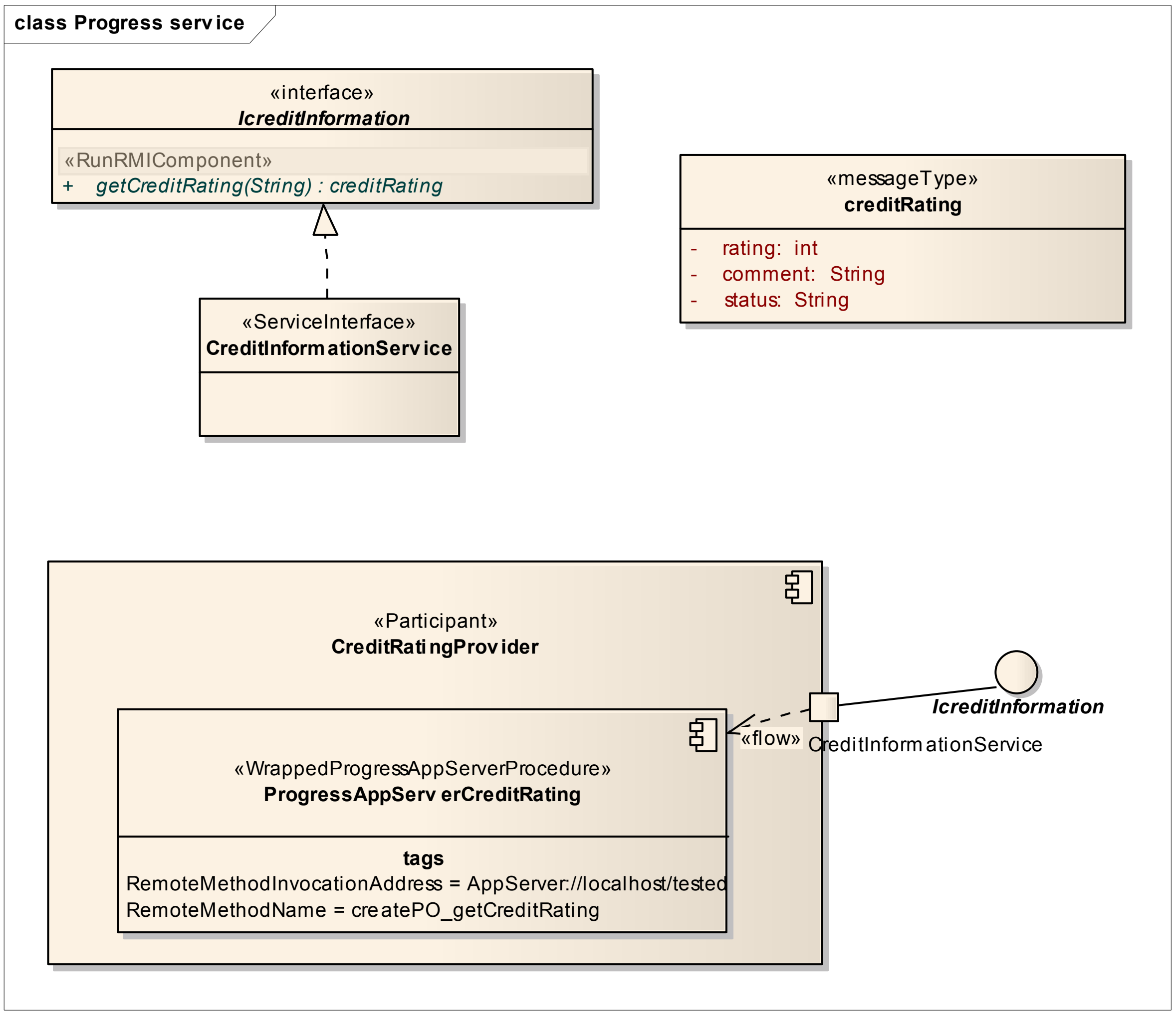
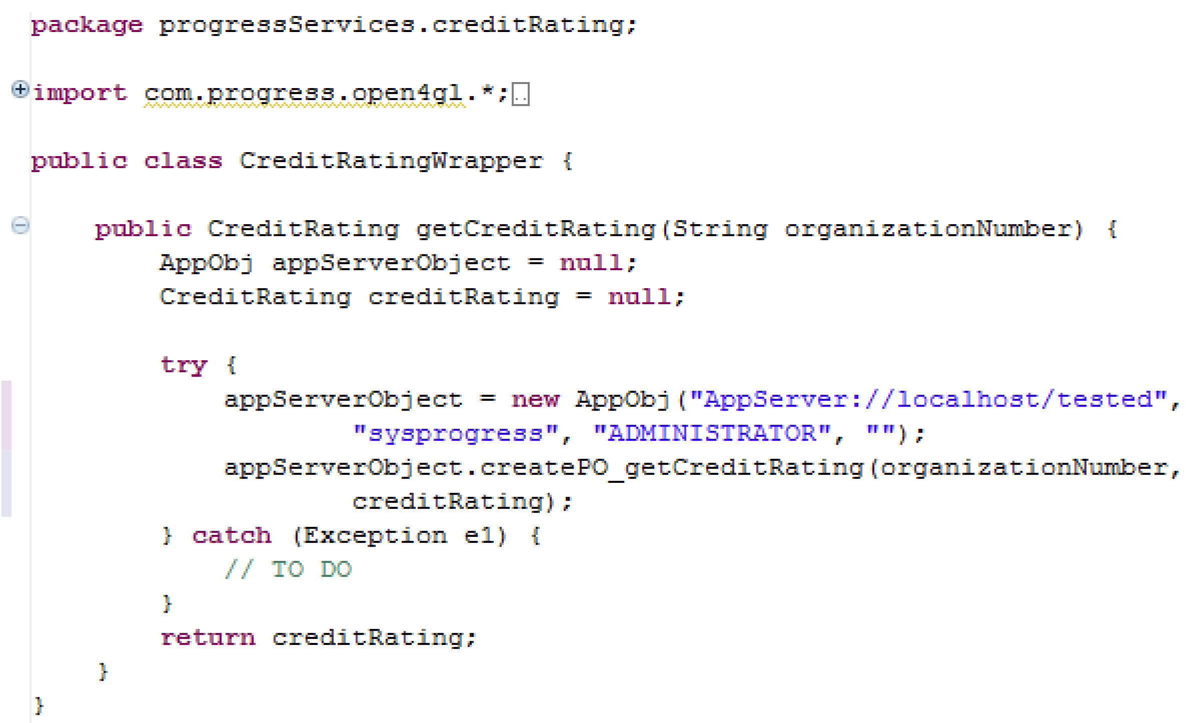


**Figure 32 Wrapping progress AppServer procedure with SoaML and a profile**

By writing new method in the MOFScript program from the OSCAR case that is invoked when finding the stereotyped element, and insert the special source code features needed to start the external Progress procedure, one can apply this code in a JEE environment and call the progress procedure by using the generated code. In the figure below is an example of such produced code.



```java
package progressServices.creditRating;

import com.progress.open4gl.*;

public class CreditRatingWrapper {

    public CreditRating getCreditRating(String organizationNumber) {
        AppObj appServerObject = null;
        CreditRating creditRating = null;

        try {
            appServerObject = new AppObj("AppServer://localhost/tested",
                    "sysprogress", "ADMINISTRATOR", "");
            appServerObject.createPO_getCreditRating(organizationNumber,
                    creditRating);
        } catch (Exception e1) {
            // TO DO
        }
        return creditRating;
    }
}
```

**Figure 33 Code produced by MOFScript applied on the service model.**

### 7.3.2 SOA and SaaS Requirements

SaaS requirements do not apply to the Progress case-study at this moment in time. It might be a requirement later to share the application among several clients or to sell it as SaaS. When it comes to the SOA requirements discoverability and QoS these are met for the in-house system. If the service is to be exposed to the outside world some sort of registry registration should be carried out and WS-Security could be implemented to ensure the safety of data.

### 7.3.3 Service Mediation

Since no service mediation modeling software is available as described in MILAS, other solutions must be used. The target platform selected as described later gives possibilities for adding service mediation inside of the products.

The Mule[78] enterprise bus is an open source product that gives possibilities to add in transformers, which are java components, between services that Mule acts as a transporter for. In this way one can add transformers or mediators outside the main service design and implementation, and achieve a separation of concerns. The advantages one would gain by modeling this into the service model is however lost.

When orchestration of services are required, another open source product can be used that has specialized on this task. Intalio[79] is an orchestration tool that has some support for mapping of data between services, as foreseen by MILAS. The disadvantage is that one need a separate tool to perform the task.

[78] Mule Ebterprise Service bus. For details see http://www.mulesoft.org/display/COMMUNITY/Home
[79] Intalio deliver an integrated cloud computing platform designed for the Enterprise. For details see http://www.intalio.com/

## 7.4 Select Target platform and deploy

As a target platform open source products are selected. At least in the beginning of a project when one might not be aware of problems that can arise, it makes sense not to invest heavily in third-party software or to bind up with a specific supplier. There are many open source products to choose from for our purpose. To help chose a proper platform the "open source SOA" by Jeff Davis [31] is recommended reading.

To make the target platform as flexible as possible the following platform is selected. The Progress Application server is selected to have possibility for a smooth modernization approach. The Progress application server can expose native Progress procedures for use with Java RMI[80] and as web services. This will ensure communication between Progress 4GL and Java components or web services.

Even though the goal is to modernize the whole legacy application one need to be able to chop it up into smaller pieces. To be able to move native Progress 4GL into an application server will make it easier to keep the Chicken Little approach a little at a time. It does not solve all of the problems, since user interface code cannot run in the application server, and has to be separated from the rest of the 4GL before it can be moved.

Also transaction can be harder to handle when mixing code, if parts of the same transactions are handled both in the modernized java code and in native Progress 4GL code. Typically the Progress 4GL will have its own transaction scope and modification must be done to make the procedures return a transaction status, that the Java code can interpret and act upon.

As a server side platform for the modernized code a JEE[81] (Java Enterprise Edition) is selected. The latest version JEE5 supports most enterprise variation of coding in a simplified manner compared to its predecessor J2EE. JEE5 comes with API[82] that supports JDBC[83] , RMI, JMS[84], web services, XML and other technologies, and of course its own EJB[85], and servlets[86] specification.

As an application server for JEE5 the Glassfish Enterprise Server[87] is chosen since it is Sun Microsystems[88] own application server build with support for the JEE5 specification.

[80] **Java Remote Method Invocation** API, or **Java RMI**, is a Java application programming interface that performs the object-oriented equivalent of remote procedure calls (RPC). For more details see http://java.sun.com/docs/books/tutorial/rmi/index.html

[81] Java Platform, Enterprise Edition or Java EE is a widely used platform for server programming in the Java programming language. See for more details http://java.sun.com/javaee/technologies/javaee5.jsp

[82] An **application programming interface** (**API**) is an interface that a software program implements in order to allow other software to interact with it; much in the same way that software might implement a user interface in order to allow humans to interact with it.

[83] **JDBC** is an API for the Java programming language that defines how a client may access a database.

[84] The **Java Message Service** (**JMS**) API is a Java Message Oriented Middleware (MOM) API for sending messages between two or more clients.

[85] **Enterprise JavaBeans** (**EJB**) is a managed, server-side component architecture for modular construction of enterprise applications.

[86] **Servlets** are Java programming language objects that dynamically process requests and construct responses

[87] **GlassFish** is an open source application server project led by Sun Microsystems for the Java EE platform.

[88] **Sun Microsystems, Inc.** is a multinational vendor of computers, computer components, computer software, and information technology services, founded on February 24, 1982.

To select a database server the choices are among using Microsoft's SQL Server 2008[89], since it is already in the house, or to stay with the original Progress RDBMS. The Progress RDBMS has performed well over the last 20 years or so, and has given the maintenance people only small challenges. The RDBMS is also SQL compliant and the database is kept as target database. This will save a lot of work concerning data access, transactions, replication and so on, since all data will stay in the same database system and easily accessible from both Progress 4GL and JDBC.
The arguments for not using Progress RDBMS are usually that resources are scares, nobody knows it and the skepticism that follows when choosing something outside of peoples familiarity.  Until new and strong arguments turns up, the projects stays with Progress RDBMS.

Since the organization has a multitude of applications that are communicating and there are strong wishes for more communication and reuse of functionality among the applications an enterprise bus is chosen to accommodate for the messaging among the same applications. A messaging system such as JMS could also be chosen, but will be implemented later when and if necessary.

As an enterprise bus MULE is selected since it is open source and is chosen as the best open source bus by Jeff Davies in [31]. For the same reasons as with MULE, Intalio is chosen as the orchestration tool.

[89] **Microsoft SQL Server** is a relational model database server produced by Microsoft. Its primary query languages are T-SQL and ANSI SQL.

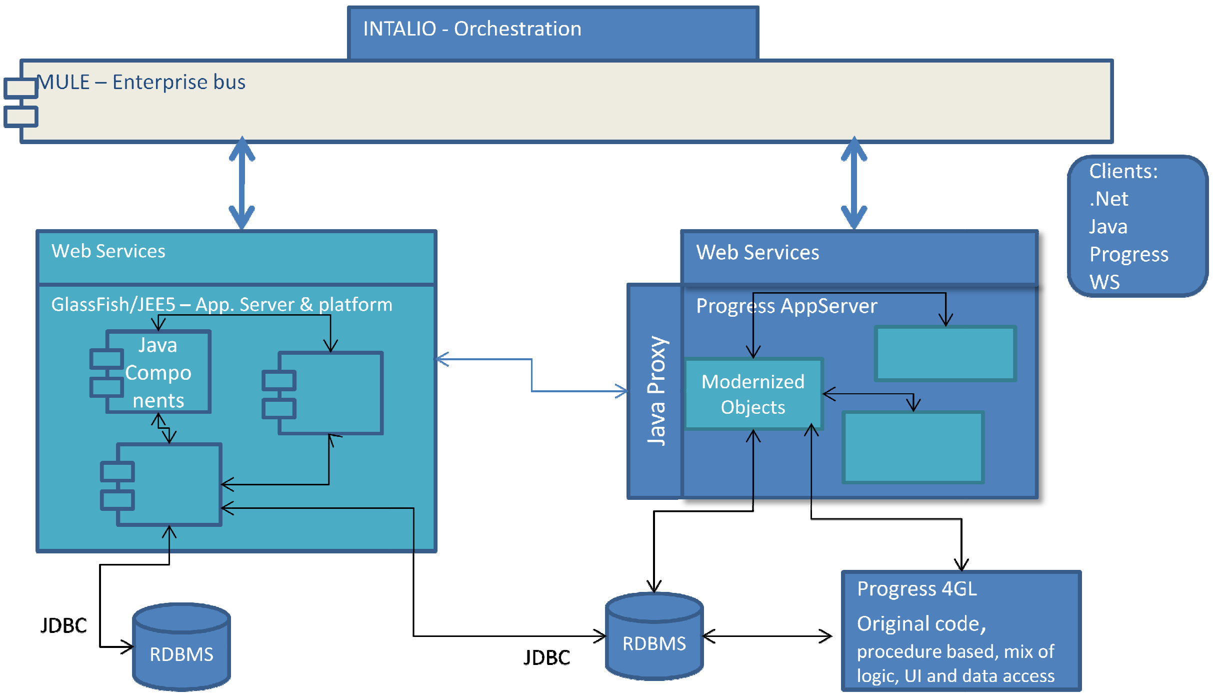


**Figure 34 Target platform for Progress modernization.**

As one can see in Figure 34 there are many possible communication paths between the modernized part of the legacy system and native Progress environment. Communication can be carried out as RMI between Java and Progress components in the Progress Application Server, between web services generated at both sides, through the enterprise bus as messages or directly to the legacy database through JDBC.

A lot of different platforms and combinations of supporting products, could have been chosen, and there are probably many that would work as well as this and even better. However the platform chosen have proven track record and should be a good starting point that are easily defended.

## 7.5 Conclusion

Following the steps in the MILAS methodology for the k90 case, results in a new application that is highly serviceable, follows standards and ensures openness and reusability. By doing the modernization stepwise one will always have a running system that is gradually upgraded towards the desired target architecture. This limits the risks of failure and therefore increases the trust among the stakeholders and users of the system compared to a Cold Turkey approach. Furthermore MILAS allows for a combination of migrated, legacy and wrapped code to run simultaneously which makes it easier and faster to migrated parts of the system at a time.

# 8 EVALUATION OF MILAS

The two case studies show that there are different needs for the different type of legacy software. While the OSCAR case did not undergo any migration, Progress 4GL procedures were both migrated fully to another language or wrapped to be available as services or RMI. Both case studies could follow the MILAS methodology, by selecting the steps important for the respective case study.

To evaluate MILAS the same criteria are used as for the evaluation of the different selected solutions in chapter 4. The criteria being the ability to perform:

- Recovering
  - Recover to architectural representation
  - Analysis of legacy architecture
  - Retrieving requirements
- Migration
  - Refactoring for SOA
  - Refactoring for SaaS
  - Control your work (Analysis)
  - Traceability between models
- Service modeling (PIM-PSM)
  - Modeling for SOA and MDA
  - Modeling for SaaS
  - Modeling service mediation
- Platform (PSM-Realization)
  - Transform to SOA
  - Transform to SaaS

A three level evaluation scale is selected to rate MILAS up against the criteria. The scale will use the following levels;

| | |
|---|---|
| √ | : The methodology is covering the criteria, but should be extended. |
| √√ | : The methodology is covering the criteria, to a satisfying level. |
| √√√ | : The methodology is covering the criteria, to the highest level. |

**Table 13 Evaluation of MILAS**

| **Area** | **MILAS Coverage** |
|---|---|
| **Recovering** | Yes |
| Recover to architectural representation | The support of OMG's ASTM and KDM meta-models makes it possible for independent vendors of analyzing software to develop new or adapt existing analyzing tools for MILAS. Even though the availability of such tools is limited today, using the OMG standards will ensure a multitude of tools in the future.<br><br>Converting legacy code to a KDM and AST representation is a fairly huge job, and need expertise in the fields of programming and UML. If no ready conversion programs exist, either as open source or commercial products, an alternative approach might be better. There exist commercial software that can help the modernization process for almost any language and a research to reveal these could benefit for the project.<br><br>Methodology evaluation: √√ |
| Analysis of legacy architecture | Yes. By using an open standard as architecture representation, any number of tools is and will be available to perform necessary analysis on the legacy architecture.<br><br>Methodology evaluation: √√ |
| Retrieving requirements | Yes. By using an open standard as architecture representation, any number of tools is and will be available to perform necessary analysis on the legacy architecture.<br><br>In addition by using BPMN and BMM these requirements can be verified against the new architecture by using linking between the different modeling levels (CIM[90] to PIM).<br><br>Methodology evaluation: √√ |
| **Migration** | Yes |

[90] CIM (Computational Independent Model) CIM is about capturing business context and business requirements.

| | |
|---|---|
| Refactoring for SOA | Yes. By following the thirteen steps in the migration section of MILAS, one will achieve a good foundation for creating services later. A sound OO design is a basis for good SOA design, something which is ensured by MILAS.<br><br>Methodology evaluation: √√√ |
| Refactoring for SaaS | Partly. A good SaaS design is based on a good SOA design, however several new requirements are introduced with SaaS and these are not properly covered or covered at all in this step.<br><br>Methodology evaluation: √ |
| Control your work (Analysis) | Yes. By transforming the source architecture to a target architecture expressed with UML and ASTM/KDM one will obtain a target architecture where there exists analysis tools and where new tools will be supplied when the new standard is being adopted by the market.<br><br>Methodology evaluation: √√√ |
| Traceability between models | Yes. Following the approved standards from the different organizations such as OMG will ensure that traceability can be applied on all levels of the total model.<br><br>Methodology evaluation: √√√ |
| **Service modeling (PIM-PSM)** | Yes |
| Modeling for SOA and MDA | Yes. Using UML, SoaML, BPMN, BMM, the basic and new model standards are covered, ensuring flexibility and reusability. Especially the SoaML and BPMN are well suited and designed for SOA modeling.<br><br>Methodology evaluation: √√√ |
| Modeling for SaaS | Partly. By providing the PIM4Wrapping some SaaS modeling features are covered to make legacy components act as SaaS applications. |

| | |
|---|---|
| | Methodology evaluation: √ |
| Modeling service mediation | Partly. If the service mediation extension to UML and SoaML are realized, one will gain control over service composition and orchestration.<br><br>Methodology evaluation: √ |
| **Platform (PSM-Realization)** | Partly |
| Transform to SOA | Yes. By using open languages such as MOFScript ensures that model elements following a defined metamodel can be transformed into practically any textual representation.<br><br>The transformation from model to text can be developed further, covering more complex situations and produce a more robust system. Ideas from the IBM paper [12] could be taken into the solution to gain a more dynamic and flexible solution.<br><br>Methodology evaluation: √√√ |
| Transform to SaaS | Yes. By applying transformations on the stereotypes defined in the PIM4Wrapping profile it is possible to transform the models to a PSM realization.<br><br>The MILAS methodology could need a profile to cover different platform specific implementation details that the different Cloud Computing suppliers require. A PIM4Cloud could solve many of the issues raised here.<br><br>Methodology evaluation: √ |

# 9 CONCLUSION & FURTHER WORK

This thesis introduces the MILAS methodology for modernization of legacy application. The methodology is based on an ADM/MDA approach were the initial goal were to have focus on the ADM part. As it turned out, most reading and studying was on the ADM part while most of the practical work were done on the MDA side.

While working on these matters I found that there exists a lot of theoretical and practical work done in both areas, but that the results of these contributions has not been put together in a standardized package that can be utilized by the potential consumers. Some packages exists that state that they contain most aspects of the modernization requirements, but these are highly commercial and not available without the aid from consultant or training by the same commercial actors.

## 9.1 CONCLUSION

However the conclusion is that many of seemingly resource heavy tasks in modernizing can be carried out with lesser effort and with higher degree of quality using a suitable methodology with accompanying tools. Conclusions in accordance with the stated goals are as follows

- What are the requirements of SOA and SaaS
- How to obtain a model of the legacy system
- How to migrate to a modern architecture
- How to model services for SOA
- How to model services for SaaS
- How to obtain modernized source code

Requirements for SOA were easy to find, since there were a waste amount of literature on the subject, and most of the requirements could be met by obtaining good OO architecture during the migration process. Thereafter SOA requirements are met through service modeling using SoaML standard and transformations of the SoaML models to source files. SaaS requirements were harder to find literature on, and especially literature that could help solve the challenges of SaaS. SOA requirements are SaaS requirements, but SaaS introduces several new ones such as multi-tenancy. These requirements are described in details in chapter 3.

To obtain a model of the legacy system turned out to be far more difficult than expected. While there exists a lot of applications that can aid in this task for major languages like Java and C, there were few that could obtain a model from Fortran or Progress 4GL software. Unless one wants to invest in expensive commercial packages there were few if any options. The hope is that the OMG ADM initiative will force the producers' of such software to develop versions that can read Fortran and Progress 4GL.

For the migration of legacy system to a modern system, several interesting books and papers were read and a thirteen step process was drawn out of this. By following these steps, or the ones that apply for the individual legacy software, one will apply a sound target architecture of good OO design that are ready for SOA and SaaS.

To the modeling of SOA the quit fresh OMG standard SoaML became a big help, and gave a good basis for the transformations from model to source.

To model SaaS, the basis of SOA modeling were used, and with the addition of our new profile PIM4Wrapping made it possible to wrap legacy code and expose it as services.

Modernized source code was obtained by transformations written in MOFScript. MOFScript is well suited to browse through models and generate textual output, and were used to generate the source code from the UML, SoaML and PIM4Wrapping models.

## 9.2 CONTRIBUTION

- *MILAS a general methodology:* The thesis delivers a general methodology that will help modernizing legacy software in a standardized and flexible way.
- *PIM4Wrapping:* The PIM4Wrapping UML profile will aid architects in design of wrapping legacy code following service modeling standards through SoaML.
- PIM4Wrapping will make it possible to generate the needed wrapping code to run legacy components as services.
- *Transformations:* Transformations done in MOFScript will automate generation of standard services and wrapping components as services.

## 9.3 FUTURE WORK

Since the field in which this thesis is written are huge when it comes to possible solutions, future work only addresses some issues the author think has most relevance to achieving a stronger methodology related to the case-studies.

- *PIM4Wrapping:* The profile added to UML for wrapping of legacy code can be extended to have better support for the different legacy objects that are to be wrapped. For instance support communication through DLL's and shared objects, and different application server objects. The PIM4Wrapping profile can be considered for future inclusion in an OMG SoaML standard extension for deployment and platform specific model mappings.
- *PIM4Wrapping:* Better support for Cloud Computing, through how the legacy object is instantiated. Locally on the same system as the controlling web service, external as a normal executable, or threads for better control.
- *PIM4Wrapping:* Direct support for selected Cloud Computing environment through support for native databases, file systems operating systems and so on, also known as IaaS (Infrastructure as a Service).
- *PIM4Wrapping:* The WS-Standard come into play in a modernization project, as soon as we are talking about services, and services will play a role in almost any modernization approaches, since the reusability of functionality, always are a priority in light of SOA/SaaS. Further work should look into the WS-standard and see how modeling can aid the implementation of various WS-Standards..
- *AST/KDM:* For the particular case-studies an AST and KDM extraction program are definitely needed to be able to follow the guidelines of MILAS. Today there are not that many extraction programs developed and none for the two development languages in the case-studies.
- *Service mediators:* More work can be done to create a good tool for service mediation that could work together with the SoaML specification. Such a tool would increase the productivity when composing new services.
- *Payment system:* One of the reasons for customer to use a SaaS system is that they just have to pay for what they use, therefore a payment system is needed to be able to invoice accordingly. A payment system could be an easy one that just registers logins and logouts, and the usage of the different services. However since the payment that the SaaS provider

has to do to the Cloud Computing platform provider is based on CPU usage, disk space and memory, the payment system must be more intricate and probably link to the Cloud Computing usage registration system.
- *Analysis tool:* A tool for SaaS applications that could analyze the modernized application with respect to how the database are accessed from the source code to see how whether access to records can be done without going through the use of a client id.

Future work should concentrate on integrate a complete toolset to support the MILAS methodology based on use of appropriate ADM/MDA and transformation tools.

# APPENDIX

## APPENDIX A: MOFSCRIPT EXAMPLE. TRANSFORMING MODEL TO POJO

```
/**
 *
 * Template Transformation Example
 *
 */

/*
  Contributors:
     Hans Aage Huru, SINTEF (Norway)
       Spring 2009
     Developed as part of the SiSaaS Project
*/

/*
*
* MOFScript UML2/SOAML to POJO Transformation
*  This  program  loops  through  the  input  model  to  find  any  "classes"
stereotyped with the
*    SOAML    stereotypes    "SOAML.ServiceInterface,    SOAML.Participant,
SOAML.Specification"
* and generates POJO based on these classes. The intention is the to use
* eclipse to generate web services based on the generated POJO's.
*/

textmodule UML2WSDL (in uml:"http://www.eclipse.org/uml2/2.1.0/UML")

//import uml_std ("uml_std.m2t")


//property nameSpaceBase:String = "http://mofscript.org/"
//property business_service_pack:String = "BusinessServiceSpec"
//property type_namespace:String = "types"
//property hostname:String = "http://localhost:8080/"

property rootDir:String = "C:/hah/uio/Sintef/MOFGen/"

property serviceInterface:String = "SOAML.ServiceInterface"
property participant:String = "SOAML.Participant"
property specification:String = "SOAML.Specification"

// Alternative approches for generation of wsdl.
// SOAML.ServiceInterface, SOAML.Participant, SOAML.Specification
property generationApproach:String = serviceInterface

property generateReferences:boolean = true

//  Used  to  hold  information  already  written  to  the  wsdl,  to  prevent
repetitive information
var referencedClasses:List
// Data types/classes to be written to the types xsd
var transformedClasses:List

var InterfaceList:List

var packageName:String
var imports:String
var classHeader:String
var classProperties:String
var classOperations:String
var tempOperations:String

var currentClass:Object

/************************************************************************
******************/
// main
/************************************************************************
******************/
uml.Package::main () {
      // Need to loop through classes later. (from operation parameters)

      imports = 'import javax.jws.WebMethod; \nimport javax.jws.WebService;
\n'

      packageName = 'package '+self.name+';'

      stdout.println("Package "+self.name)

      // Main routine
      self.ownedMember->forEach(c:uml.Class){
            stdout.println("  Class "+c.name)

            classHeader = ""
            classProperties = ""
            classOperations = ""

            if (generationApproach == serviceInterface) {
                  if (c.hasStereotype(serviceInterface)) {
                      classHeader = classHeader + '@WebService\n'
                        c.SOAML_ServiceInterfacePackage()
                  }
            }
            else if (generationApproach == participant) {
              if (c.hasStereotype(participant)) {
                  c.ownedPort->forEach(p:uml.Port) {
                        p.SOAML_ServiceInterfacePackage()
                  }
              }
            }
          else if (generationApproach == specification) {
              if (c.hasStereotype(specification)) {
                        c.SOAML_ServiceInterfacePackage()
              }
          }
      }

    // Generate POJO's for the interfaces
      self.ownedMember->forEach(o:uml.Interface) {
        if (InterfaceList.contains(o)) {
                  o.Interface()
        }
      }

      // Gets associated classes to the referenced classes
      self.ownedMember->forEach(o:uml.Class) {
        if (referencedClasses.contains(o.name)) {
          // Has this class any associated classes?
          o.getAssociatedClasses(o)
        }
      }

      // Generate POJO's for refernced classes
      self.ownedMember->forEach(o:uml.Class) {
        if (referencedClasses.contains(o.name)) {
          stdout.println("Referenced class "+o.name)
          if (!transformedClasses.contains(o.name)) {
                stdout.println("Printed referenced class "+o.name)
               o.SOAML_ServiceInterfacePackage()
          }
          transformedClasses.add(o.name)
        }
      }

      stdout.println (referencedClasses)
      referencedClasses.clear()

}

/***************************************************************************
*******************/
// SOAML_ServiceInterfacePackage
/***************************************************************************
*******************/
uml.Class::SOAML_ServiceInterfacePackage () {
//  if (self.name = "Interface Model") {
    stdout.println("SOAML.ServiceInterfacePackage")
    currentClass = self

      // Participant port
    if (self.oclGetType()=='Port') {
            currentClass = self.owner
      }


    // Class header info
    classHeader = classHeader + 'public class ' + currentClass.name

    // Extends...

    // Properties
      currentClass.ownedAttribute->forEach(p:uml.Property) {
            p.privateProperty()
      }

    // Realized interfaces
      var i:Integer = 0

      currentClass.interfaceRealization.first().target-
>forEach(t:uml.Interface) {
            if (t.name!=null) {

                  // Prevent writing the interface more than once...
                if (!InterfaceList.contains(t)) InterfaceList.add(t)

                classHeader = classHeader + ' implements ' + t.name + ' {'
+ '\n'
```

```
                  // Interface operations
                  i=i+1
                  t.ownedOperation->forEach(f:uml.Operation) {
                        tempOperations = ''
                  f.Operation()
                  if     (currentClass.hasStereotype(serviceInterface)     ||
currentClass.hasStereotype(participant)) {
                    classOperations = classOperations + '    @WebMethod\n'
                  }
                  classOperations = classOperations + tempOperations + '\n'
+ '    }\n\n'
                }
            }
      }
    if (i<1) {
        stdout.println   ("Error   in   model:   'No   SOAML.ServiceInterface
model'")
    }

    // Required interfaces

    // Own operations
    currentClass.ownedOperation->forEach(o:uml.Operation) {
            tempOperations = ''
      o.Operation()
      if    (currentClass.oclIsTypeOf('Interface'))    classOperations    =
classOperations + tempOperations + '\n\n'
      else classOperations = classOperations + tempOperations + '\n' + '
}\n\n'
    }

    '
    '
    // Writes the POJO
    file (rootDir + currentClass.name + ".java")
    packageName
    '\n\n\n'
    imports
    classHeader
    '\n'
    classProperties
    classOperations
    '\n}'
}

/***************************************************************************
******************/
// SOAML_ParticipantPackage
/***************************************************************************
******************/
uml.Class::SOAML_ParticipantPackage (pBase:uml.Package) {
    stdout.println("SOAML.Participant")

    // Realized interfaces
      var i:Integer = 0
      //Need to find the ports
      self.ownedPort->forEach(p:uml.Port)
      {
            p.type.interfaceRealization.first().target-
>forEach(t:uml.Interface) {
                  if (t.name!=null) {
                        i=i+1
                    }
                }
        if (i<1) {
            stdout.println ("Error in model: 'No SOAML.Particpant model'")
        }
        }
}

/*****************************************************************************
******************/
// Referenced classes
/*****************************************************************************
******************/
uml.Package::refClasses () {

   self.ownedMember->forEach(o:uml.Class) {
        if (referencedClasses.contains(o.name)) {
          // Has this class any associated classes?
          o.getAssociatedClasses(o)
        }
        if (referencedClasses.contains(o.name)) {
          // Has this class any associated classes?
          o.SOAML_ServiceInterfacePackage()
        }
   }
}


/*****************************************************************************
******************/
 // This is a recursive method,
 // going through the class hierarchy to find all associated classes of the
original one.
/*****************************************************************************
******************/
uml.Class::getAssociatedClasses (c:uml.Class) {
      c.ownedAttribute->forEach(a:uml.Property | a.association == true)
      {
          if (!referencedClasses.contains(a.type.name)) {
                  //a.type.wsdlTypeMapping()
                  referencedClasses.add(a.type.name)
          }
            a.getAssociatedClasses(a.type)
      }
}

/*****************************************************************************
******************/
// Interface
/*****************************************************************************
******************/
  uml.Interface::Interface () {
    stdout.println("Interface")

    currentClass = self

    // Class header info
    classHeader = classHeader + 'public interface ' + self.name + '{\n'

    self.ownedOperation->forEach(o:uml.Operation) {
            tempOperations = ''
            o.Operation()
            classOperations = classOperations + tempOperations + '\n'
      }

    '
    '
    // Writes the POJO
    file (self.name + ".java")
    packageName
    '\n\n\n'
    //imports
    classHeader
    '\n'
    classOperations
    '\n}'


}


/*************************************************************************
******************/
// Properties
/*************************************************************************
******************/
  uml.Property::privateProperty () {
    if (self.upperValue = 1) {
    classProperties = classProperties + 'private ' + self.type.name + ' ' +
self.name ';
    '
    }
    else {

    classProperties = classProperties + 'public Set<' + self.type.name + '>
' + self.name + ' = new HashSet<' self.type.name '>();
    '
    }
  }

/*************************************************************************
******************/
// Operation
/*************************************************************************
******************/
uml.Property::Operation () {
    var i:Integer

      // Binary file transfer?
      if (self.hasStereotype("BinaryFileTransfer")) {
            self.binaryFileTransfer()
      }
      else {
            tempOperations   =   tempOperations   +   '          public   '   +
self.type.name + ' ' + self.name + '('
            self.ownedElement->forEach(p:uml.Parameter) {
                  // For later references
                  if (!referencedClasses.contains(p.type.name)) {
                        referencedClasses.add(p.type.name)
                  }
```

```
                    if (p.direction == "in") {
                          if  (i>0)  {  tempOperations  =  tempOperations  ',  '  }

                          tempOperations = tempOperations + p.type.name + ' '
+ p.name
                          i+=1
                    }
          }
          if  (currentClass.oclGetType()  ==  "Interface")  tempOperations  =
tempOperations + ');\n'
          else tempOperations = tempOperations + '){\n'

            // Executable?
            if          (self.hasStereotype("ContainsExecutable")          &&
currentClass.oclGetType() == "Class") {
              self.executable()
            }

      }
}

/*************************************************************************
******************/
// Executable
/*************************************************************************
******************/
uml.Operation::executable () {
    var execName:String = 'dummy.exe'
    var path:String
    var input:String

      imports = imports + 'import java.io.IOException;\n\n\n\n'

            // Path, exename, input param

            if   (self.hasValue("ContainsExecutable",   "ExecutableName_1"))
execName = self.getValue("ContainsExecutable", "ExecutableName_1")
            if   (self.hasValue("ContainsExecutable",   "ExecutablePath_1"))
path = self.getValue("ContainsExecutable", "ExecutablePath_1")
            if    (self.hasValue("ContainsExecutable",    "ParameterList_1"))
input = self.getValue("ContainsExecutable", "ParameterList_1")


            tempOperations = tempOperations +
'         @SuppressWarnings("unused")
            Process theProcess = null;

            try
            {
                theProcess = Runtime.getRuntime().exec("' + path + execName
+' '+input+'");
            }
            catch(IOException e)
            {
               //System.err.println("Error    on    exec()    method    "    +
e.getMessage());
               return e.getMessage();
            }
            return "SUCCESS "+System.getProperty("user.dir");
'
  }
```

```
/***********************************************************************
*****************/
// Binary files
/***********************************************************************
*****************/
  uml.Operation::binaryFileTransfer () {
    var destDir:String = 'dummy.exe'

      // Necessary imports for biinary file transfer
    imports    =    imports    +    'import    java.io.File;    \nimport
java.io.FileOutputStream;\n'

      // Path, exename, input param

      if  (self.hasValue("binaryFileTransfer",  "destinationDir_1"))  destDir
= self.getValue("binaryFileTransfer", "destinationDir_1")

      if (currentClass.oclGetType() == "Interface") {
            tempOperations  =  tempOperations  +  '      public  String  '  +
self.name  +  '  (byte[]  bytes,  String  fileName,  String  orgDir,  String
returnDir);'
      }
      else {
            tempOperations = tempOperations +
'    public String ' + self.name + ' (byte[] bytes, String fileName, String
orgDir, String returnDir) {

            File newFile = null;
            // Destination directory
            String fatesCFG = "' + destDir + '";
            String newFileName;

            // New file
            newFileName = fileName.substring(orgDir.length()+1);
            newFileName = fatesCFG+newFileName;
            newFile = new File(newFileName);

            File dir = new File(newFile.getParent() + "/");
            //Generate dirs if necessary
            dir.mkdirs();

            try {
                  newFile.createNewFile();
            } catch (IOException e1) {
                  e1.printStackTrace();
            }
            newFileName = newFile.getAbsolutePath();

        // the byte array argument contains the content of the file
        // the string argument contains the name and extension of the file
passed in the byte array
        try
        {
            //  instance  a  filestream  pointing  to  the  storage  folder,  use
the original file name
            // to name the resulting file
            FileOutputStream fs = new FileOutputStream(newFileName,false);

            // write the memory stream containing the original
            // file as a byte array to the filestream
            fs.write(bytes);

            fs.close();

            // return OK if we made it this far
            return "OK";
        }
            catch (Exception ex)
            {
                // return the error message if the operation fails
                return ex.getMessage();
            }
'
        }
    }
```

# APPENDIX B - PIM4WRAPPING PROFILE

**PIM4Wrapping stereotypes**

| **Stereotype** | **UML element** | **Tags** | **Description** |
|---|---|---|---|
| WrappedExecutable | Component or Class | PathToExecutable | To distinguish the executable component. |
| StartsExecutable | Operation | | To distinguish the operation that starts the executable. |
| TransferAllFiles | Data element | | To tell the transformation program how to deal with file transfers. |
| TransferFilesIn-Directory | | | To tell the transformation program how to deal with file transfers. |
| TransferOneFile | | | To tell the transformation program how to deal with file transfers. |
| WrappedProgress-AppServerProcedure | Component or Class | RemoteMethod-InvocationAddress | To distinguish the remote component. |
| | | RemoteMethod-Name | To be able to call the remote method correctly. |
| RunRMIComponent | Operation | | To distinguish the operation that interact with the remote component. |

## APPENDIX C - BIBLIOGRAPHY

1. Solheim Ida, S.K., *Technology Research Explained.* SINTEF, 2006. **A313**
2. M., U.W., *Legacy systems, transformation strategies*. 2002: Prentice Hall.
3. Warren, I., *The Renaissance of Legacy Sytems*. 1999, London: Springer.
4. Michael, B.L. and S. Michael, *Migrating legacy systems*. 1995, San Fransisco: Morgan Kaufman Publishers Inc.
5. Frederick P. Brooks Jr, *The Mythical Man-Month: Essays on Software Engineering.* 1975.
6. Thomas, E., *Service-Oriented Architecture*. 2006: Prentice Hall.
7. OMG, *Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS)*. 2008, OMG.
8. Ulrich W., K.V., *Architecture-Driven Modernization: Transforming the Enterprise.* 2007.
9. OMG, *Knowledge Discovery Meta-Model (KDM)* 2009.
10. OMG, P.N., Ed Gentry, Carlos Araya,Ivan Sanazria,Charles Dickerson, William Ulrich, *White Paper on RFP II: Abstract Syntax Tree Meta-Model* in *OMG White paper*. 2004.
11. *Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS)*. 2008, OMG.
12. G. Kandaswamy, L.F., Y. Huang, S. Shirasuna, S. Marru, D. Gannon, *Building web services for scientific grid applications.* IBM Journal of Research and Development, 2006. **2006**(2).
13. Mark Weiser, *Program Slicing. IEEE Transactions on Software Engineering*, 1984. **10**(4): p. 5.
14. Seacord, R.C., D. Plakosh, and G.A. Lewis, *Modernizing Legacy Systems*. 2003: Addison-Wesley.
15. Ying Zou. *Quality Driven Software Migration of Procedural Code to Object-Oriented Design*. in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. 2005.
16. Ying Zou, *A Framework for Migrating Procedural Code to Object-Oriented Platforms*, in *Software Engineering Conference, 2001. APSEC 2001. Eigth Asia-Pasific*. 2001. p. 390-399.
17. Ying Zou, *Incorporating Quality Requirements in Software Migration Process*, in *Software Technology and Engineering Practice*. 2003. p. 175-185.
18. Ying Zou, *Incremental Quality Driven Software Migration to Object Oriented Systems*, in *20st IEEE International Conference on Software Maintenance (ICSM'04)*. 2004. p. 136-146.
19. Ying Zou, K.K. *Incremental Transformation of Procedural Systems to Object Oriented Platforms*. in *Computer software and Applications Conference, 2003, COMPSAC 2003, Preceedings. 27th Annual International*. 2003.
20. Ying Zou, T.C.L., Kostas Kontogiannis, Tack Tong, Ross McKegney. *Model-Driven Business Process Recovery*. in *Reverse Engineering, 2004, proceedings, 11th Working conference on*. 2004.
21. Fowler, M. and K. Beck, *Refactoring : improving the design of existing code*. The Addison-Wesley object technology series. 1999, Reading, MA: Addison-Wesley. xx1, 431 p.
22. W. Zhang, A.J.B., D. Roman, H.A. Huru, *Migrating Legacy Applications to the Service Cloud*, in *OOPSLA'09*. 2009: Orlando. p. 59-68.

23. Sun Microsystems, I., *Introduction to Cloud Computing architecture.* Sun Open Cloud Computing Document, 2009.
24. Micrsoft Corportation. *Windows Azure platform*. Available from: http://www.microsoft.com/azure/services.mspx.
25. Amazon web services. *Infrastructure services*. Available from: http://aws.amazon.com/.
26. Google corporation. *Google App Engine*. 2009; Available from: http://code.google.com/intl/nb/appengine/.
27. The RESERVOIR Seed Team, *RESERVOIR - An ICT Infrastructure for Reliable and Effective Delivery of Services as Utilities.* IBM Technical Library H-0262, 2008.
28. Radu Prodan, S.O., *A Survey and Taxonomy of Infrastructure as a Service and Web Hosting Cloud Providers.* The 10th IEEE/ACM International Conference on Grid Computing, 2009.
29. Zach Hill, M.H., *A Quantitative Analysis of High Performance Computing with Amazon's EC2 Infrastructure: The Death of the Local Cluster?* 2009.
30. P. Coad, E.Y., *Object Oriented Analysis*. 1990: Prentice Hall PTR.
31. Davis, J., *Open source SOA*. 2009, Greenwich, Conn.: Manning. xxiv, 425 p.