A Software Migration Concept for ASP-based Web Applications to Java

04. January.2010

Minor Thesis

Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
Institute of Systems Architecture
Faculty of Computer Science
Technische Universität Dresden

Supervisor: Dipl. Inf. Marius Feldmann

Le Hai Dang
S2980966@inf.tu-dresden.de

Declaration of Academic Honesty

I, Le Hai Dang, declare that this minor thesis on the topic of **“A Software Migration Concept for ASP-based Web Applications to Java”** is wholly my own work and conducted without any assistance from third parties.

Furthermore, I confirm that no sources have been used in the preparation of this thesis other than those indicated in the thesis itself.

Dresden, 04. January.2010

Signature

Table of Contents

LIST OF TABLES AND FIGURES III

ABSTRACT 1

1 INTRODUCTION 2

1.1 Heterogeneous IT-Infrastructures in Companies 2
1.2 Problems of Heterogeneous Environments 3
1.3 Application Class: Custom Business Web Applications 4
1.4 Primary Requirements 5
1.5 Available Migration Strategies and Related Migration Works 6
1.6 Motivation and Goals 7

2 USE CASE: ROBERT BOSCH CORPORATION 8

2.1 Situation and the Migration Task 8
2.2 Secondary Requirements 8
2.3 Best-Practice Migration Task: TICOS Web Application 9

3 ANALYSIS ON ASP WEB APPLICATIONS 11

3.1 Sample Codes 11
3.2 MVC & Programming Styles in ASP Web Applications 13

4 BASICS: TECHNOLOGIES, ARCHITECTURES AND TOOLS 17

4.1 Technologies 17
4.1.1 ASP – Active Server Pages 17
4.1.2 JSP – Java Server Pages 18
4.1.3 Java Servlet 18
4.1.4 Component Object Model 18
4.1.5 Web Service 19
4.1.6 JAX-WS 19
4.1.7 Model View Controller Design Pattern 19
4.1.8 Enterprise Service Oriented Architecture – SAP NetWeaver 20
4.1.9 Portal System 20
4.2 Migration Tools 21
4.2.1 NetCoole J-ASP 21
4.2.2 COM4J 24

5 SOFTWARE MIGRATION 25

5.1 Common Deciding Factors 25
5.2 Migration Strategies and Methods 25
5.2.1 Migration Strategies 26
5.2.2 Migration Methods 27

6 MIGRATION CONCEPT 28

6.1 A General Migration Concept for Software Systems 28
6.2 The Stages of the Migration Concept 30
6.3 Software Evaluation 33
6.4 Object Modeling 35
6.5 Software Post-Documentation 37
6.6 Object Refinement 40
6.6.1 Vertical Migration vs. Horizontal Migration 41
6.6.2 Model View Controller Structure 41
6.6.3 Refinement and Design of MVC Components 42

6.6.4 Design of Service Interfaces ... *55*
6.7 SOFTWARE IMPLEMENTATION ... 56
6.7.1 Mapping of ASP files to Structures ... *56*
6.7.2 Business Package Class & QueryList class ... *57*
6.7.3 Mapping of Model code ... *58*
6.7.4 Mapping of Controller code ... *60*
6.7.5 Mapping of View code ... *63*
6.7.6 Implementation of Web Services ... *64*
6.7.7 Business Package Dependencies ... *64*
6.8 SOFTWARE TESTING ... 65
6.9 SOFTWARE INTEGRATION ... 65

7 USE CASE: MIGRATION OF TICOS ... 67

7.1 MIGRATION PROCESS ... 67
7.2 PROBLEMS AND ADVANTAGES ... 74

8 OUTLOOK ... 76

REFERENCES ... A

BIBLIOGRAPHY ... A
WEB REFERENCES ... B

List of Tables and Figures

Fig. 1.2.a : Tight coupling of applications in a heterogeneous environment [IBM SOA] 3
Fig. 1.3.a : Classification of Custom Business Web Applications 5
Fig. 2.3.a : TICOS application structure 10
Fig. 3.1.a : Request- Response-objects in ASP 11
Fig. 3.1.b : ADO.DB example in ASP 12
Fig. 3.1.c : COM components in ASP 12
Fig. 3.2.a : Mixed MVC-style 13
Fig. 3.2.b : Separated View via XSLT 14
Fig. 3.2.c : Complete MVC separation with COM, ASP and XSLT 15
Fig. 3.2.d : MVC-styles in ASP applications 16
Fig. 4.2.1.a : createMember.asp 22
Fig. 4.2.1.b : createMember.jsp 23
Fig. 6.1.a : Migration Concept by Sneed 29
Fig. 6.1.b : Incremental Migration Concept 30
Fig. 6.3.a : Software Evaluation 33
Fig. 6.4.a : Object Modeling 35
Fig. 6.4.b : Tight coupling between Web applications 36
Fig. 6.5.a : Software Post-Documentation 37
Fig. 6.5.b : Example of source code documentation 39
Fig. 6.6.a : Object Refinement 40
Fig. 6.6.2.a: MVC structure in Java 42
Fig. 6.6.3.a : MVC-styles in ASP applications 43
Fig. 6.6.3.b : Wrapping of SQL queries – Model component 45
Fig. 6.6.3.c : SQL queries in Controller component 46
Fig. 6.6.3.d : Target Design with encapsulated legacy components 47
Fig. 6.6.3.e : Object oriented calls on the data model 48
Fig. 6.6.3.f : SQL statement with subqueries 49
Fig. 6.6.3.g : Encapsulation of SQL statements 49
Fig. 6.6.3.h : Conversion and Redevelopment 50
Fig. 6.6.3.i : View component in ASP 51
Fig. 6.6.3.j : View component converted to JSP 51
Fig. 6.6.3.k : Sample of ASP & XSLT 52
Fig. 6.6.3.l : Target Design with separated View 53
Fig. 6.6.3.m : Encapsulation of COM business objects 54
Fig. 6.6.3.n : Target Design for separated MVC 55
Fig. 6.6.4.a : Service Interface design 56
Fig. 6.7.1.a : Example of pageflow mapping from ASP to Java 57
Fig. 6.7.2.a : Example of QueryList class 58
Fig. 6.7.3.a : Mapping of Business Functions to Java 59
Fig. 6.7.4.a : Mapping of built-in ASP objects to Java 62
Fig. 6.7.4.b : Redirection of data to Views 63
Fig. 6.7.6.a : JAX-WS Web Service 64
Fig. 6.9.a : UDDI Model for Web Services 66
Fig. 7.a : The legacy TICOS application 67
Fig. 7.1.a : TICOS application structure 68
Fig. 7.1.b : TICOS Worktime management use-case diagram 68
Fig. 7.1.c : TICOS data export via files 73
Fig. 7.2.a : Deployment of a migrated business package on the SAP Portal 75

Abstract

In many of today's large-scale enterprises, IT-administrations often times find their IT-infrastructure heavily "departmentalized" [IBM SOA]. This happens primarily, because enterprises tended to leave their business departments and local IT-administrations to take care for their own needs, instead of relying on a centrally managed IT-organization. For that reason local departments ended up creating applications in isolation from the global context. As a result the IT-infrastructure became heterogeneous and complex, as applications were coupled in a direct manner, due to the lack of common interfaces. This situation is called *tight coupling* of applications, which is a major source of maintenance and development problems for enterprises.

A modern solution for this situation is the *Service Oriented Architecture (SOA),* which is widely implemented in enterprises nowadays. The *Service Oriented Architecture* is an IT-infrastructure, which introduces *loose coupling* to the environment and decouples applications by providing common interfaces. A group from the set of enterprise applications consists of Web-based applications, which are used in the intranet environments to support business departments with customized functionalities. These Web Applications are often times implemented in ASP 3.0 (or classic ASP) and are built in a monolithic manner. As a result, Web applications are often times part of the enterprise's migration strategy to migrate legacy IT-systems to a modern SOA-based IT-infrastructure.

Therefore this paper focuses on the migration of the class of intranet-based Web Applications and presents a migration concept for Web Applications implemented in ASP, in the context of a migration to a *SOA*-based IT-infrastructure. Based on a migration project conducted at Robert Bosch Japan, this paper summarizes analysis, deciding factors, migration approaches and best practice experiences in a concept, with the goal to provide a systematic approach to migrate ASP-based Web applications successfully.

1 Introduction

1.1 Heterogeneous IT-Infrastructures in Companies

A significant part of a company's IT-infrastructure is characterized by management and controlling systems, which is usually composed of professional ERP software (Enterprise Resource Planning) and custom applications. These custom applications are tailored to complement software of the shelf and are often times realized as web applications in the company's intranet environment. They can be found in all business departments of companies and are applied in many different use-cases. In a large-scale company, it is often the case that the local IT-departments develop customized applications, for different departments of the regional branch, and thereby do not follow a strict IT-policy (sometimes due to a lack of a global IT-policy). This often times leads to an overall heterogeneous IT-infrastructure which consists of monolithic and tightly coupled applications tightly. Due to this "departmentalized" way of developing software, applications are built in a monolithic way in an isolated manner, which causes the applications to lack extensibility and reusability.

Through this constellation many companies struggle to cope with the increasing maintenance problems and the demands for modern IT services. In order to solve these problems the companies IT-administrations are changing their IT-infrastructures to a more homogeneous environment, by using common technologies and common interfaces. Along with a change to a homogeneous environment, a so called *Service Oriented Architecture* is often times established by the companies. Thereby a Service Oriented Architecture is a widely applied solution to provide extensibility and reusability to the IT-environment as it is based on the idea of interoperable and reusable services which encapsulate functionalities of the underlying business applications. Therefore companies can benefit from such an architecture by:

- Reducing *tight coupling* of applications via changing middleware connections to Service interfaces, which introduce *loose coupling* and decrease maintenance costs of legacy software
- Reusing Service interfaces to reuse functionalities in order to speed up development time and lower development costs of applications
- Implementing an *Enterprise Service Bus* to increase flexibility in the communication between applications and allow better integration of applications
- Introducing a central presentation platform, such as a portal system, that is based on the aggregation of contents from business applications and services.

1.2 Problems of Heterogeneous Environments

In a “departmentalized” legacy IT-infrastructure the problems lie in the many custom applications used in the departments. Due to their independent development, they are heterogeneous and require middleware to be interoperable with each other. The resulting picture of such an IT-environment resembles a complicated network of heterogeneous nodes with customized middleware connections. This kind of environment is depicted below.

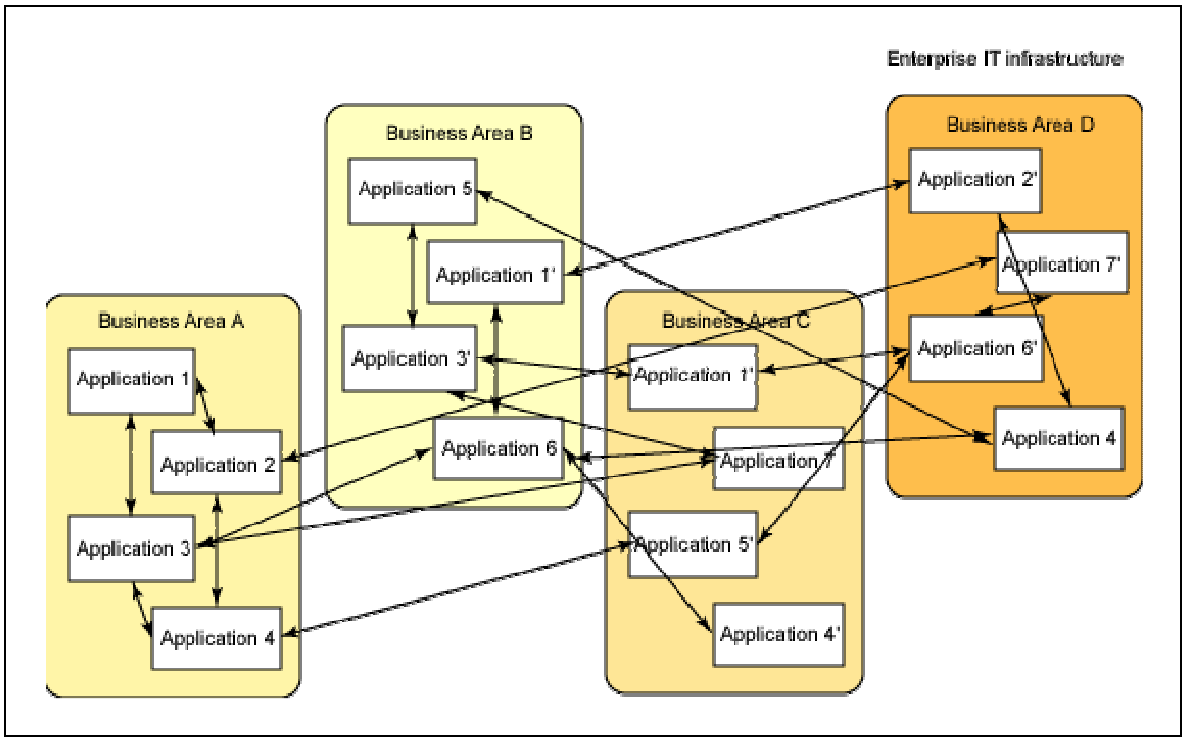

Fig. 1.2.a : Tight coupling of applications in a heterogeneous environment [IBM SOA]

What is apparent in such a case is the numerous quantity of middleware used to link applications together, whereby middleware is required to connect applications with each other, which are implemented in different technologies (such as Java, ASP, PHP or native technologies like COM) and operate on different data sources (such as relational databases, data files, messages) with each other. This introduces strong dependencies to the applications and result in inflexibility and inextensibility in regard to changes to the systems. Therefore the maintenance costs for such a constellation is high. Additionally the potential of redundant and inconsistent data exist, since applications often times store overlapping set of data locally. Therefore middleware connections have to deal with different technologies and different data formats in these environments. In fact it is often the case that information can not be synchronized properly due to the heterogeneous infrastructure.

Through these reasons companies have begun to modernize their IT and transform their infrastructure to a SOA-based IT-infrastructure. Consequently as a part of this movement,

the class of Web Applications, which constitutes the majority the described applications, are also being migrated to a SOA-based platform. In the following the class of these mentioned Web Applications will be characterized.

1.3 Application Class: Custom Business Web Applications

As an own category of applications in a company's IT-environment, *Custom Business Web Applications* can be found in the intranet environment of many business departments. There, they serve many kinds of requirements, ranging from controlling, resource planning tasks to information or messaging services. Thereby *Custom Business Web Applications* are small-sized or middle-sized monolithic Web Applications, which operate on local data sources or retrieve portions of data from external sources via some sort of data service. Typically *Custom Business Web Applications* are implemented in different (legacy) programming technologies, depending on the expertise available in the different business departments or local IT-administrations. The common web technologies used are "classic" ASP, JSP or PHP. Therefore it is difficult for the companies to come up with a general migration process or concept, because of the differences in the technologies.

Through the isolated development of *Custom Business Web Applications*, the software quality can vary very much, depending on the maturity of the system. The range can span from applications with poor implementation and documentation quality, e.g. where typical Model-View-Controller components are mashed up together and documentations are insufficient, to matured applications, where components are well structured and documented. Though in reality within a department, there are more small-sized and poor implemented applications than matured applications. From a technical point of view *Custom Business Web Applications* are used by a small quantity of users (usually by those in the business department) on a regular basis, as for example in time accounting or project management Web applications. Therefore optimization issues such as load balancing, availability etc. are less relevant.

From the data-centric point of view, *Custom Business Web Applications*, typically operate on local datasources. If ever data must be shared, applications tend to retrieve portions of data and store them locally for further processing, therefore synchronization services are mandatory in many applications, although usually less synchronization services are implemented. Additional third party services can be used such as directory services (e.g. LDAP) or (Windows) authentication services.

Classification	Custom Business Web Applications
Type	- Specialized Web Applications - Intranet environment - Small-sized to middle-sized legacy Web Applications
Examples/Fields of usage	- Controlling (resource planning, time accounting, meeting room reservation) - Resource Planning (warehouse management) - Information and communication services (web translator, web-based instant messaging, conference chat, maintenance notification, employee/customer information) - knowledge management (wiki, collaboration)
Properties	- use of middleware to communicate with other applications of different technologies - monolithic - specialized and redundant data - non-uniform user interfaces - decentralized login - non-uniform login mechanism and identity management
Implementation	- Implementation in different programming languages - commonly ASP, PHP, JSP - AJAX, COM, ActiveX components
Software Quality/State	- less usage of common programming conventions - no (rare) application of MVC model - usually no object orientation - poor documentation of code

Fig. 1.3.a : Classification of Custom Business Web Applications

1.4 Primary Requirements

Resulting from the named problems with heterogeneous IT-infrastructure and *Custom Business Web Applications*, the primary goals for requirements are:

- **Migration of Custom Business Web Applications to a target programming technology (Java)**

  This is an IT-policy specific requirement, which goal is to generalize the programming technology used at business departments in order to provide better support and resolve heterogeneity of the IT-infrastructure.

- **Generation of SOA-based interfaces from business functions**

  The goal is to provide reusability of business functions and resolve exponential dependencies between applications. By providing SOA-based interfaces the

quantity of middleware/bridges between applications will be reduced and therefore enables loose coupling between applications.

- **Modernization of Custom Business Web Applications to a Model-View-Controller based structure**

  As *Custom Business Web Applications* are implemented by local business departments, the quality of code is usually not good. As applications should be reusable, they must be able to cope with future changes, therefore an implicit modernization along the migration process is required.

1.5 Available Migration Strategies and Related Migration Works

The topic of software modernization is a complicated problem in software engineering, which has produced different kinds of solutions by the time, especially in case of software migration of procedural programs to object oriented programs (such as in ASP to Java). From the perspective of automatism, one can find different approaches ranging from full- or semi-automatic code-to-code transformation to complete manual approaches. Works that are based on automatic code-to-code transformation rely on re-engineering techniques in order to generate dataflow and function call graphs, examples of works that fall into these categories are [Martin/Müller] [Ping et al.] or [Cimitile et al.]. Although the solutions named in [Martin/Müller] refer to C-to-Java migration it is apparent that automatic solutions often times result in unsatisfactory codes that are not readable for humans. This is also apparent in the direct context of ASP-to-Java migration, namely in the [J-ASP] tool, which will be introduced in *4.2.1 NetCoole J-ASP*. Besides full automatic solutions, semi-automatic code-to-code transformation tools can be found. Those tools rely on user interaction in order to retrieve ambiguous information in the re-engineering process. The aim of such systems is to enhance the quality of the resulting code. In the work of [Ping et al.] a code-to-code transformation tool for Net.Data* to JSP was developed, which migrated Net.Data programs to JSP while enhancing the structure to a Model-View-Controller based design. The tool developed relied on the analysis of the Net.Data Abstract Syntax Tree (AST) and the mappings of dataflow and function call graphs, additionally data access via SQL queries are analyzed and dynamic variables are mapped with the help of dataflow models. The advantages of automatic and semi-automatic approaches in comparison to manual approaches lie in the fast transformation of code, however often times the result of those works are unsatisfactory for industrial standards. In [Sneed] an overview of the reasons and problems in the migration of procedural programs to object oriented programs is given.

Due to the complexity of the topic and the limitations found in the industry, it is often times not feasible to follow the research heavy path of automatic or semi-automatic migration. Therefore a pragmatic goal is often set in companies, which encourages the

* Net.Data is a server-side scripting language developed by IBM

development of manual migration concepts. In this manner this paper tries to provide an extensive migration concept for ASP based Web applications.

However, in the context of Web applications migration concepts, materials and references are scarce. Through research on the internet the author had to rely on Web sources in order to gain valuable experiences in Web application migration. Therefore the information and approaches presented in this paper relies heavily on the experiences of the author. Related works that have been done to this subject can be found in [Jeenicke] or [MSDN Migration].

1.6 Motivation and Goals

In the concrete case of a company-wide migration process in the context of a SOA-based IT-infrastructure, companies must provide a migration procedure for *Custom Business Web Applications*. Due to the different technologies used in different departments, it is not easy for a company to provide a general migration procedure for all technologies and all application types. Instead the companies must provide local departments and IT-administration a guideline which is customized for the technologies used within these departments. Thereby one must take into account, that different departments and IT-administrations have different fields of expertise, therefore the guideline must define a systematic decision process to migrate Web applications from one environment to another.

Since the paper is based on a project conducted at Robert Bosch Japan, where ASP was the major web technology used, “classic” ASP is chosen as the legacy web technology to represent the migration of *Custom Business Web Applications*. Therefore the goal is to provide a holistic migration concept for ASP-based Web Applications, which discusses the following problems of migration:

- How to analyze and what factors have to be considered in preparation for a migration process
- Identification of reusable components in ASP-based Web applications
- What are the problems in the migration of ASP-based Web applications
- What kind of migration methods can be applied to ASP Web applications
- How to modernize the structure of legacy Web applications
- How to migrate towards a SOA-based infrastructure

2 Use Case: Robert Bosch Corporation

2.1 Situation and the Migration Task

Due to the heterogeneous IT-infrastructure of the Robert Bosch Corporation, the company has introduced a custom Service Oriented Architecture, called “Robert Bosch SOA”, and deployed a portal system called SAP NetWeaver Portal as a central platform for users to access business applications. In regard to its custom Web applications, the company’s goal is to migrate those Web applications to a base technology, which is Java, and integrate the applications with the portal system.

In this paper, a project conducted at Robert Bosch Japan will serve as the best-practice example for the migration of Web Applications. Since this project was part of a migration process at Robert Bosch Japan, the goal was to migrate the application called TICOS (“Task-Information-Charging-Operation-System”) to Java and integrate it with the SAP NetWeaver Portal. As in the case of Robert Bosch Japan, the local IT department is responsible for the whole IT administration and controlling of Robert Bosch Japan. The local IT department develops small- and middle-sized customized Web applications to support the business processes of the local business departments. Most of the applications produced are kinds of controlling tools, to supervise projects and account times/costs. They all are designed as stand-alone applications and have only few interaction interfaces with each other, such as with COM (Microsoft Component Object Model) interfaces for ASP Web applications.

However at Robert Bosch Japan applications are mostly connected together at data level via shared databases or with custom data services, which often times is used to interchange data with external clients, such as other equivalent departments of different regions (China, Australia and Europe etc.). Just until recently all these applications were used in an intranet environment, but due to the company’s IT policy, it had been decided to migrate and integrate all applications to a new portal environment, in order to allow users a centralized application access. The goal of the company is to provide central access to resources and applications by a common architecture in a standardized and service oriented manner.

2.2 Secondary Requirements

Due to the transformation from the old heterogeneous infrastructure to a new homogeneous infrastructure and the introduction of Robert Bosch SOA and the SAP NetWeaver Portal system, the company has defined new IT-policies in regard to the custom Web applications used at subsidiaries. These requirements include the migration of Web applications to Java and the exchange of traditional middleware for modern service oriented Web Service interfaces. Additionally to that, Web applications should be integrated with the portal system and should provide certain functionalities of the portal system.

Under the term “Minimized Portal Integration” the company has introduced the minimal requirements for the migration process of custom Web Applications to provide the minimal set of functionality that all applications must support when integrated into the company’s portal system, in order to gain the added value of portal systems.

- **Single Sign On (SSO) and Identity Management**

  For applications used in the portal system of Robert Bosch Company, the company requires additional functions to be implemented. As for improving efficiency and security, applications integrated into the portal system have to implement a Single Sign On mechanism.

  Single Sign On is a mechanism where users are only required to login once to the portal system and then be automatically logged into integrated applications subsequently. Therefore users do not need to care anymore about a great multitude of passwords and hence work more safely and in an efficient manner.

- **Corporate Design**

  Since the portal system is the company’s presentational platform and in order to provide a uniform user interface in the web applications, it is necessary to introduce a common user interface design which is defined in the corporate design. Therefore, it is required from applications to support the portal’s corporate design.

- **Linking with Portal Navigation**

  Web applications from different departments and locations are required to be accessible from a single central platform (SAP NetWeaver Portal). Therefore web applications must be linked with the portal application in a personalized manner. The web applications must then be integrated with the portal in a form, which allows the portal to initialize the Web applications in a personalized manner.

2.3 Best-Practice Migration Task: TICOS Web Application

TICOS (“Task-Information-Charging-Operation-System”) is a Web-based information system deployed at the Information Systems department of Robert Bosch Japan and is used for accounting and charging of IT services. TICOS stores, among other things, master data, such as project, employee or customer data and manages the accounting, charging and planning of these resources. Essentially TICOS works with personal data and manages project workflows. Projects can be created by managers and then planned with team members. The system provides project planning (work packages, milestones, resource planning) and time accounting (daily time accounting, time management) mechanism and presents them in personalizable views. On regular basis TICOS’ business

data is exported to external systems for further processing, this is done by an scheduled service, which retrieves data from the database and exports data to different formats such as Microsoft Excel Sheet (*.xls) or XML to a shared data server.

From the technical point of view, TICOS is implemented with ASP in combination with XML, XSL and JavaScript. The application's structure is inflexible, since TICOS components mix up Model-View-Controller codes and functionalities are scattered into many files.

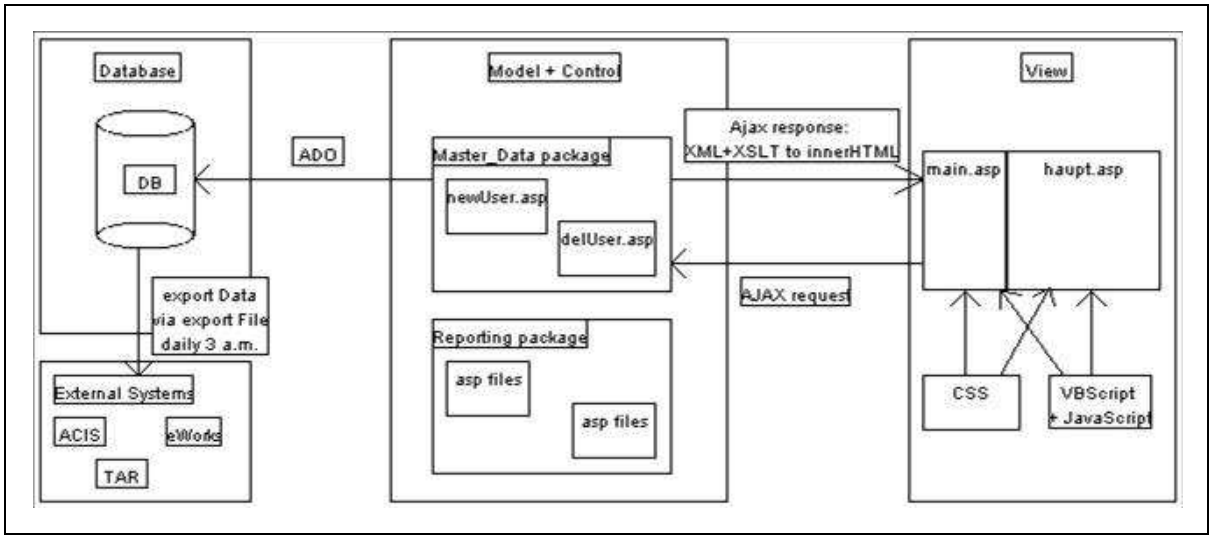

Fig. 2.3.a : TICOS application structure

As described previously the goal was migrate TICOS to the Java platform with integration to Robert Bosch SOA and portal system. Hence the tasks included analyzing appropriate migration strategies and conducting migration with focus on the redesign of functionalities for SOA and implementing required functionalities in order to integrate with the portal system. Furthermore the system must be redesigned in order to be flexible for future changes.

3 Analysis on ASP Web Applications

In order to provide a migration concept for ASP-based Web applications, an analysis of the ASP technology and programming styles is necessary. In the next sections samples of ASP specific techniques will be presented and programming styles will be described.

As mentioned in the previous chapter, ASP is a server-side scripting technology. As such ASP scripts are interpreted on server-side by a scripting runtime, called “Active Scripting Engine”. This scripting runtime supports the development of ASP application in different programming languages, such as VBScript, JScript, with the requirement that each programming language must be interpreted by an associated Active Scripting Engine which is implemented as a COM-class. In this paper we will only consider VBScript types of ASP Web Applications, as the VBScript Active Scripting Engine is the standard and the most common implementation available on Microsoft IIS server (Internet Information Services). Since the Active Scripting Engine is itself implemented as a COM component, it supports COM components to be used in ASP pages. As such ASP has standard components already implemented by default. In the following the programming style and usage of COM components in ASP will be presented.

3.1 Sample Codes

Syntactically VBScript is a derivate of Visual Basic (VB) and Visual Basic for Applications (VBA). The following code snippet will show VBScript and standard components implemented in VBScript ASP:

```
<html>
<body>
<%    Set username = Request.Form("username")
      Dim greeting

      greeting = "Hello "  & username & " !"

      Response.write(greeting)
%>
</body>
</html>
```

Fig. 3.1.a : Request- Response-objects in ASP

A COM object which is regularly used by ASP is ASP ADO (ActiveX Database Object), which is an ActiveX (COM) object that provides access to database management systems and comes with the Microsoft IIS server. Therefore data can be queried with SQL statements in ASP pages, like this:

```
<html>
<body>
<%
Dim sConnection
sConnection = "DRIVER={MySQL ODBC 5.1 Driver}; SERVER=localhost;
DATABASE=test; UID=root;PASSWORD=admin"

Set objConn = Server.CreateObject("ADODB.Connection")
objConn.Open(sConnection)

Set resultSet = objConn.Execute("SELECT * FROM test.table1")

objConn.close
%>
</body>
</html>
```

Fig. 3.1.b : ADO.DB example in ASP

As ADO is a COM component the DB object (COM class) can be created via the Server object in ASP. In the same way any other COM component can be referenced in ASP like this, therefore oftentimes application logic are implemented as COM components in ASP Web Applications.

```
<html>
<body>
<%
Set calculator =
Server.CreateObject("COMCalculator.VectorCalculator")
Set vector1 = Server.CreateObject("COMVector.Vector")
Set vector2 = Server.CreateObject("COMVector.Vector")
vector1.x = 1
vector1.y = 2
vector2.x = 3
vector2.y = 4

Dim result
vector1 = calculator.vAdd(vector1, vector2)

result = "v.x=" & vector1.x & "v.y=" & vector1.y
Response.write(result)
%>
</body>
</html>
```

Fig. 3.1.c : COM components in ASP

Regarding programming styles, programming with COM components or with SQL queries are both widespread. Whereas modeling business objects via SQL is not ideal but accessing data is more straightforward. On the other hand COM components are hard to

implement and require additional programming skills. Through these technical features of the ASP technology, there can be several programming styles be described. The programming style of a ASP Web application is one of the main characteristics which describes the software state of an application, thus it is necessary to analyze the main programming styles which are widespread in ASP applications, in order to be able to generate a migration concept for ASP-based applications.

3.2 MVC & Programming Styles in ASP Web Applications

With the capabilities of ASP shown in the previous section, there are different styles widespread how ASP applications can be developed. Regarding this, it is different from project to project, in which style (and therefore in which quality) ASP applications can be found. That must be kept in mind when migrating ASP applications as the quality of code affects the outcome of the migration. The different styles range from completely mixed MVC components to completely strict separation of MVC components. Generally a strict separation of Model View Controller components is considered the best solution for (big) Web Applications, however in order to achieve that, additional technologies must be utilized, as ASP alone is not suited for every MVC component. In the following some possible styles of ASP applications will be presented.

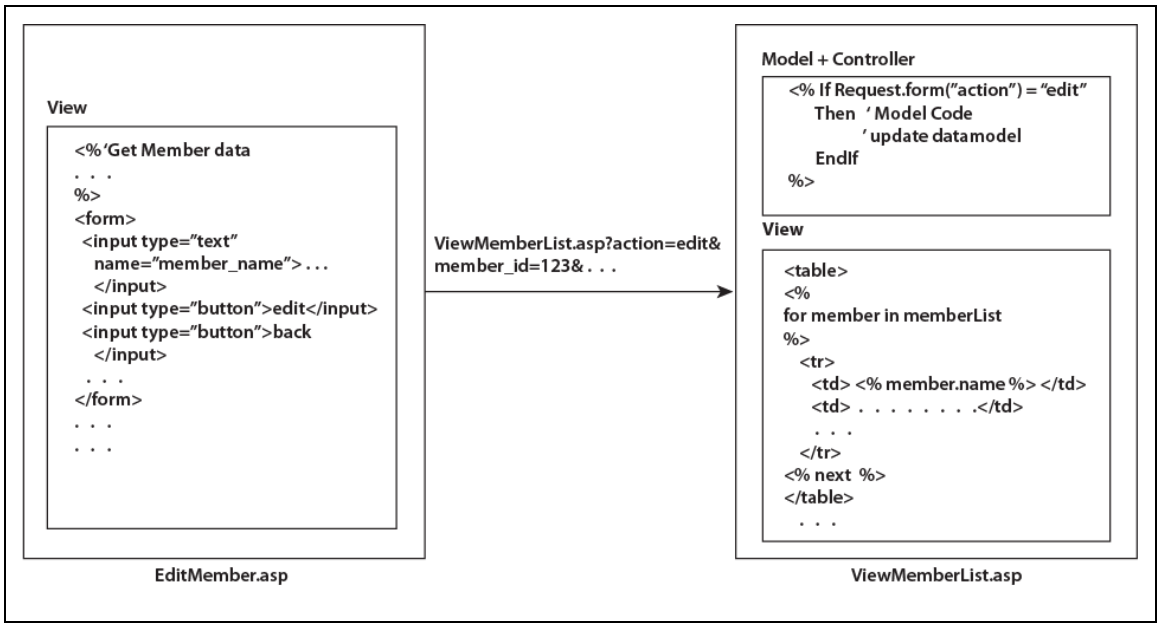

Fig. 3.2.a : Mixed MVC-style

In the first case, MVC components can be found in a completely mixed state. This is apparent as "EditMember.asp" retrieves "Member" data via SQL queries and forwards the new input data to "ViewMemberList.asp". There, the incoming data will be validated (Controller) and applied to the data model (Model) and the actual "Memberlist" (View) will be rendered.

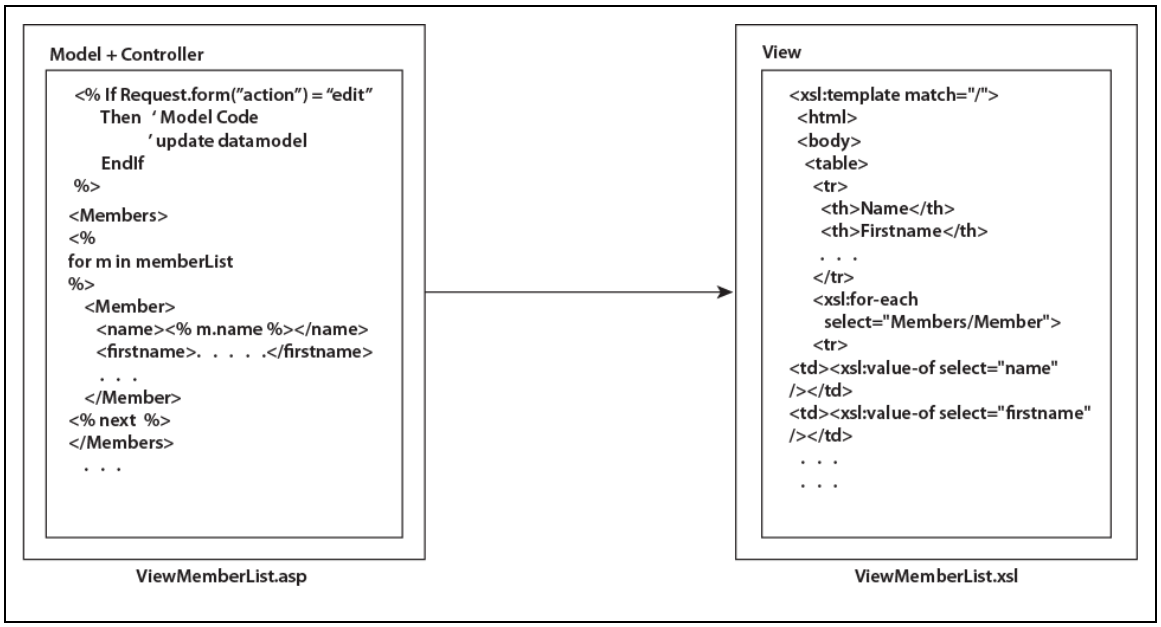

Fig. 3.2.b : Separated View via XSLT

Between complete mixed components and strict separation of components, forms of partial separation of MVC components can be found. One form of partial separation is when View components are separated via a combination of ASP pages and XML transformations. In such a case Controller and Model Components are separated to dedicated ASP pages and View Components are processed by separated ASP pages and XSL files, where ASP pages hold the data to be presented in XML structure and XSL files transform those XML structures to HTML. On pages where Controller and Model components reside, SQL queries are used to access data from the data model. In such a design business functions and control flow functions are not separated into different files, therefore business logic and control logic are mixed with each other.

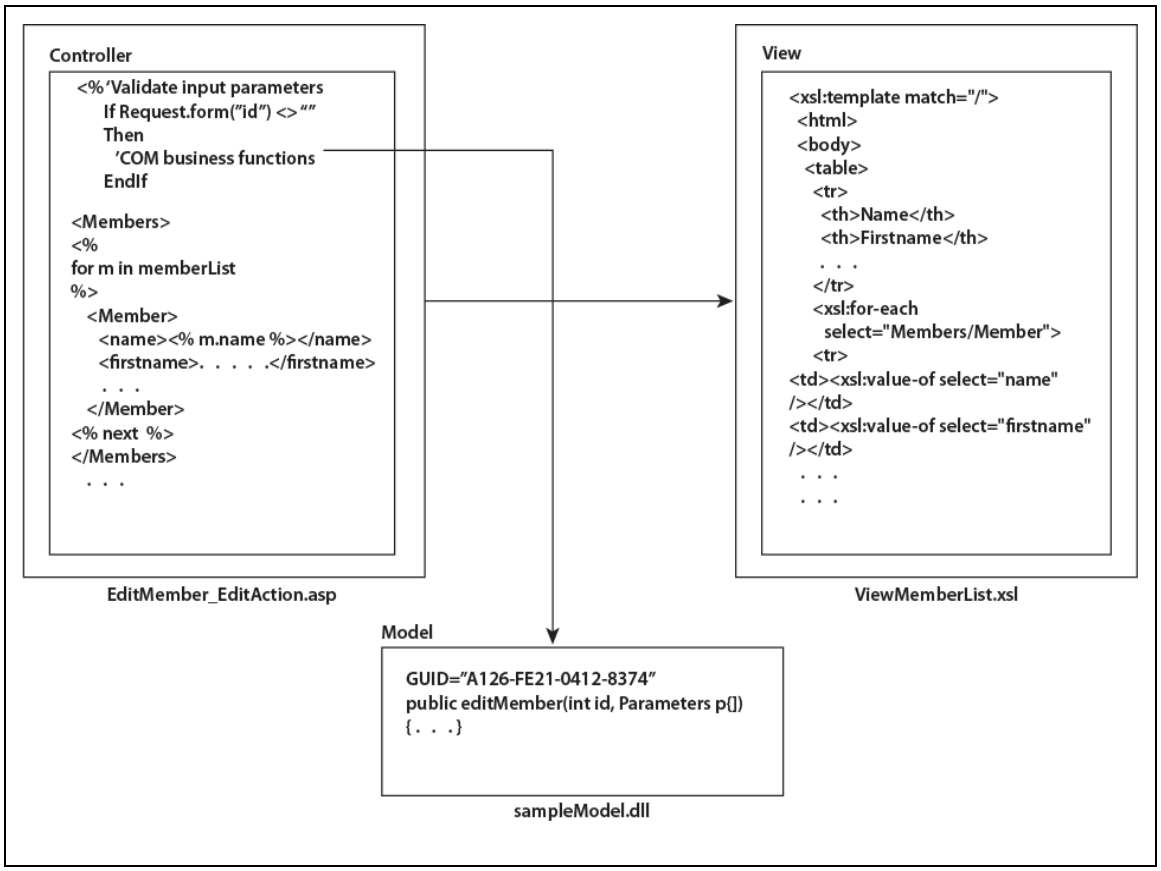

Fig. 3.2.c : Complete MVC separation with COM, ASP and XSLT

In complete separation of MVC components, Model components are implemented by COM classes, Controller components are separated by dedicated ASP pages and View Components are implemented by independent ASP pages. In such a design COM components access data and provide business functions via object methods. Controller components are separated to dedicated ASP pages, where they process input parameters to handle the control flow of the application. Based on the input parameters the business logic of the application is directed and the result is sent to the dedicated View page.

ASP Style	Page Structure	Model	View	Controller
Completely Mixed MVC	• Model, View & Controller elements are mixed together within an ASP file	• No object oriented business object model • Business functions in one page with other components • Data access via SQL	• Data entities from Model are presented with ASP and HTML	• Control flow is based on SQL queries to the data model or with COM objects • External functionalities with 3rd party COM components
Separated View	• Business functions are mixed with control flow functions in one page • View is separated to a dedicated page	• No object oriented business object model • Business functions in one page with Control components • Data access via SQL queries	• Data is presented in a separated ASP page with SQL queries • View can be additionally separated with XML and XSL files	• Control flow is based on SQL queries to the data model • External functionalities with 3rd party COM components
Complete Separation of MVC	• Business functions are separated into COM components • Control flow functions are separated from View pages • View is separated	• COM components access data model • Business functions are implemented in COM components • Data access via COM DAO	• Data is presented in a separated ASP page with SQL queries or COM DAO • View can be additionally separated with XML and XSL files	• Controller pages manage control flows of the application based on input parameters and business logic • Control flow is managed in separated files

Fig. 3.2.d : MVC-styles in ASP applications

4 Basics: Technologies, Architectures and Tools

In this chapter relevant Web technologies, used in the best-practice migration project, will be described. The legacy Web technologies used in Robert Japan are ASP and JSP. Through the desired target platform by Robert Bosch Corporation, J2EE (Java Enterprise Edition) technologies are mandatory, therefore J2EE standards and modern J2EE Web frameworks will be introduced. Additionally SOA concepts and products used at Robert Bosch will be described. Finally available migration tools for the relevant technologies will be presented.

4.1 Technologies

4.1.1 ASP – Active Server Pages

Active Server Pages is a legacy web technology by Microsoft, which is used to create dynamic HTML pages. Nowadays ASP is replaced by ASP.Net as the successor technology of Microsoft. The mechanism behind Active Server Pages lies in the embedment of ASP codes into a HTML structure, which can be evaluated dynamically during run-time. As a server scripting technology, ASP codes are interpreted by the Web server (commonly by the Microsoft "Internet Information Services" server) and the resulting HTML documents are returned to the clients. The programming language used by ASP can be any scripting language (commonly VBScript and JScript), which is compatible with the Active Scripting Engine[*]. Through the support of COM-components, the functionality of ASP scripts can be extended via COM/ActiveX components. Therefore the functionality of ASP is not limited to built-in components such as (*"Session", "Request", "Response", "FileSystem"* etc.), but can also be extended by custom COM-components. This is often done in order to implement business logic in an ASP application.

Advantages/Disadvantages:

One of the main limitations of ASP is that it easily allows mixing up presentation and business logic. As for example one can often find SQL statements directly embedded inside of ASP pages, which violates the common Model-View-Controller design pattern in Web Applications. This is usually done when programmers access data sources via the ADO (ActiveX Data Objects) COM-component, and implement their SQL-Statements directly in ASP pages instead of COM-components itself. Hence in practice, ASP pages are often implemented in the "dirty" way. Another problem regarding the migration of COM-components of ASP applications to Java is that COM-components are bound to Microsoft Windows and can not be reused by Java for other operating systems easily.

[*] The Active Scripting Engine itself is a COM component, because of that it provides Scripting Languages to access COM components natively

4.1.2 JSP – Java Server Pages

Java Server Pages (JSP) is a technology for the presentation layer that allows the dynamic creation of contents by using Java code, in similar way to ASP. JSP pages are located on Java Web servers that evaluate Java-code fragments with a JSP compiler. JSP is designed to separate application logic from presentation, therefore code for application logic can be implemented in standard Java classes (POJO – "Plain Old Java Object") and be called by Java tags in JSP pages. This is a difference to ASP, as ASP allows COM-components to be called from Active Server Pages, instead of Java classes, which are usually implemented in the JavaBeans convention. JSP applications are usually written in combination with the JSTL (JSP Standard Tag Library). This custom tag library is an extension of the JSP specification, which allows the JSP compiler to evaluate tags defined by the JSTL. Custom tags are sorted by category such as "*core*", "*xml*", "*sql*" etc.

Advantages/Disadvantages:

The advantage of JSP is the implicit integration of Java technology as compared to ASP with COM support. Hence the technology is not bound to the requirements of the underlying backend. The limitations of JSP are similar to ASP, such as the danger of mixing up business logic code with presentation code easily. In regard to the migration of legacy Web Applications from technologies such as ASP to Java, JSP as a target technology is an ideal replacement for the presentation layers of an ASP application, since there are many similarities between the two technologies.

4.1.3 Java Servlet

Java Servlets are Java classes, which are configured by the Java-Server to process HTTP requests. These Java classes process HTTP requests and return HTML code to the client via a HTTP-Response stream. Since Servlets are implemented at class level they are not suited as View components (since the HTML code generation would be tiresome) but instead as Controller components, because the control flow can be managed in the code directly. Therefore Servlets are often used in combination with JSP to implement the MVC pattern. From a technical point of view, JSP pages are equal to Servlets, since JSP pages will be processed as Servlets by the Webserver.

4.1.4 Component Object Model

Microsoft Component Object Model (COM) [COM] is a technology, which enables applications to reuse software components in the Microsoft Windows environment. COM components can be created with many different programming languages, such as C++, Visual Basic. The technology is used to provide interfaces in the local Windows environment, which can be reused by other applications. COM components are compiled binary files, often in form of a .dll or .exe file, which must be registered in Windows in order to be reused. The COM technology itself is limited to the local environment, however functionalities can be extended by *Distributed Component Object Model (DCOM)* in order to be reused remotely.

4.1.5 Web Service

Web services are remote software components that can be accessed via networks, such as the internet or enterprise intranet. Web services in the regular configuration, are based on the 3 open standards, Simple Object Access Protocol (SOAP), Web service Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI), which define the communication, the interfaces and the registration of a Web service. Web services provide remote procedure calls and can be used to build up a Service Oriented Architecture (SOA). Web services share similarities with DCOM as they provide remote computation and describe interfaces for such communication, however the biggest differences lie in the communication protocol[*] and interface descriptions[†]. [Web Service vs. DCOM]. In comparison to DCOM, Web services provide loose coupling to a software environment, therefore Web services are the technology of choice for Service Oriented Architectures.

4.1.6 JAX-WS

Java API for XML – Web Services (JAX-WS) [Sun JEE 5 Tutorial, part III], is a Java API for the creation and consumption of Web Services. The API was introduced in the Java Platform - Enterprise Edition 5 and is a replacement for the legacy Web Service API, Java API for XML-based RPC (JAX-RPC). In JAX-WS Web Services can be created in a much simpler way than it was in JAX-RPC, thereby JAX-WS makes use of Java annotations heavily and simplifies the deployment of Web Services notably. Compared to JAX-RPC, interfaces must not be created to generate Web Service stubs.

4.1.7 Model View Controller Design Pattern

The Model-View Controller (MVC) design pattern is an architectural pattern for structuring the software development process of applications into Model (data model or business object model) View (presentation) and Controller components (control logic). The goal is to increase flexibility in software development, by facilitating future modifications and extensions on the current software. The design pattern employs separation of concerns for components, which helps to replace or reuse components from each other independently. In the context of Web applications, the MVC pattern is widely used as a reference, for modern design. Thereby each of the three components can be implemented by different standards (of a technology). In the case of Java, oftentimes Java Beans/EJB is used as Model, JSP as View and Java Servlet as Controller. In such a configuration, a normal HTTP request is processed starting at the Java Servlet, which process the request and calls Java Beans to return data to the JSP View component. In legacy Web applications MVC components are usually mixed up together, therefore it is

[*] Web services communicate over HTTP via SOAP, while DCOM is based on a proprietary DCOM protocol

[†] DCOM is based on remote objects and interfaces of COM objects defined in type libraries, while Web services rely on loosely coupled WSDL interfaces

a goal in software migration to identify and to decouple MVC components from each other, since the objective of software migration is the modernization of software.

4.1.8 Enterprise Service Oriented Architecture – SAP NetWeaver

The software development in companies is more difficult, when functionalities can not be reused and data can not be accessed seamlessly. Due to such obstacles, the modeling of business processes can be slow and inflexible. Through the years, business processes have changed at a fast pace and demands from inside or outside a company have increased, thus a flexible and adaptable component based software environment have become necessary, which could provide compositioning and reusing of existing functionalities in a loose coupled environment. Such an architecture is the Enterprise Service Oriented Architecture.

The concept of the Enterprise Service Oriented Architecture is based on the compositioning of business services during the software development process. Other than that loose coupling is also introduced by the Enterprise Service Oriented Architecture. Thereby loose coupling is the basis for companies to integrate their incompatible systems with each other. In a Service Oriented Architecture, a Service is a reusable functional component, which encapsulates fine to coarse grained application functionalities that can be called remotely and is defined by interface specification meta-data. Beside that Services can be stored in repositories that can be looked up via naming and look up functionalities. As described by [Nicolescu/Klappert/Krcmar, p.37 – p.40], an Enterprise Software Architecture is composed of a Service Repository, Service Bus and a Frontend Application. The latter one is usually represented by a portal application. In the case of the Bosch SOA this instance is filled by the SAP NetWeaver Portal.

4.1.9 Portal System

On top of the Robert Bosch IT-infrastructure, the company has introduced the SAP NetWeaver Portal, which acts as a front-end component to allow user-computer interactions with the systems integrated to it. By definition SAP NetWeaver Portal is an *enterprise portal*, which belongs to the group of *closed and process oriented portals* [Nicolescu/Klappert/Krcmar, p. 23]. Whereby a closed portal is, in contrast to an open portal (e.g. to which web portals belong to), a web system that is focused on a closed group of users. An enterprise portal is defined as by [Vlachakis/Kirchhof/Gurzki, p.11] as follows:

"An enterprise portal is defined as an application, which based on web technologies provides a centralized access to personalized contents as well as processes. Therefore enterprise portals offer the possibility to support processes and collaboration between heterogeneous groups. Characteristic for portals are the links and data transfer between heterogeneous applications via a centralized platform and a uniform user-interface. A manual log in on individual applications integrated to the platform is not necessary through Single Sign On."

By definition portal systems aggregate contents created from integrated applications and display them in a uniform manner on the portal page. Portal systems display personalized web-contents, depending on user roles and manage those roles in via their own identity management system. Furthermore portal systems can increase efficiency and security by proving SSO (Single Sign On) functionality to all integrated applications and can add many services to support business related tasks, such as virtual conferencing, collaborating, document sharing and etc.

4.2 Migration Tools

Migration processes can be supported by migration tools, which can implemented in a form of a migration wizard or a stand-alone application that transforms code-to-code. In the case of ASP-to-Java migration one of few migration tools that can be found is NetCoole J-ASP, which transforms ASP applications into JSP/Servlet applications. In the following this migration tool, along with a the wrapper tool COM4J will be presented, which is an extremely useful tool when it comes to wrapping COM components to Java classes.

4.2.1 NetCoole J-ASP

J-ASP is an ASP to JSP transformation tool from NetCoole company [J-ASP], which can migrate ASP web applications to JSP applications. The tool takes ASP pages as input and transforms them into JSP or Java Servlet applications. Thereby the tool works in this way:

1. Takes a ASP project as input and analyzes each ASP page of it
2. Gets variables defined in an ASP page and declares them in the JSP page
3. Maps predefined ASP objects (Request, Response, Session, ADODB etc.) to J-ASP objects in Java (jasp.Request, jasp.Response, jasp.Session, jasp.AdoDB etc.)
4. Transform ASP code to JSP code and create the JSP files

Essentially the tool works by mapping the standard ASP objects into Java classes and transforming ASP codes to JSP codes based on the J-ASP classes. Following is a sample of J-ASP transformation:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />
<title>Sample create Member</title>
</head>
<body>
<div id="container">

<%    'Controller

      Dim member_id
      randomize()
      member_id = int(rnd*9999999)+1
      Set member_name = Request.Form("member_name")
      Set member_firstname = Request.Form("member_firstname")

      Dim sConnection, objConn , objRS
      sConnection = "DRIVER={MySQL ODBC 5.1 Driver};
SERVER=localhost; DATABASE=ticos; UID=root;PASSWORD=; OPTION=3"
      Set objConn = Server.CreateObject("ADODB.Connection")
      objConn.Open(sConnection)
      Set rs = objConn.Execute("INSERT INTO tbl_members " & _
                           "(member_id, name, firstname) " & _
                         "VALUES ('" & member_id &"','" & _
               "member_name" & "','" & member_firstname & "')" )

      objConn.close
      Response.Redirect "ViewMemberList.asp"
%>

</div>
</body>
</html>
```

Fig. 4.2.1.a : createMember.asp

```
<%@ page contentType="text/html; charset=iso-8859-1" %>
<%@ page import="jasp.buildin.*" %>
<%@ page import="jasp.util.*" %>
<%@ page import="jasp.vbs.*" %>
<%@ page import="jasp.adodb.*" %>
<%@ page extends="jasp.servlet.JspBase" %>
<%
 try {
    jspinit(request,response,application,out);
%>

<%
    //Controller
    double member_id = 0;
    IStringList member_name = null;
    IStringList member_firstname = null;
    String sConnection = "";
    Connection objConn = null;
    variant objRS = new variant();
    Recordset rs = null;
%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />
<title>Sample create Member</title>
</head>
<body>
<div id="container">

<%
    vb.Randomize();
    member_id = vb.Int(vb.Rnd() * 9999999.0) + 1.0;
    member_name = Request.Form("member_name");
    member_firstname = Request.Form("member_firstname");
    sConnection = "DRIVER={MySQL ODBC 5.1 Driver};
SERVER=localhost; DATABASE=ticos; UID=root;PASSWORD=; OPTION=3";
    objConn = new Connection();
    objConn.Open(sConnection);
    rs = objConn.Execute("INSERT INTO tbl_members " +
"(member_id, name, firstname) " + "VALUES ('" +
vb.CStr(member_id) + "','" + member_name.toString() + "',' " +
member_firstname.toString() + "')");
    objConn.Close();
    Response.Redirect("ViewMemberList.jsp");
%>

</div></body></html>
```

Fig. 4.2.1.b : createMember.jsp

Advantages/Disadvantages:

NetCoole J-ASP works by mapping ASP functionality to Java, therefore ASP developers can easily read J-ASP fragments in Java code. The problem is that ASP-style code is inserted in JSP files, which is confusing for JSP developers. Furthermore as the transformation method of J-ASP is rather simple, the tool does not provide migration of COM components. Therefore the usage of J-ASP is limited to simple ASP applications. The applications that can be migrated with J-ASP must not have the business logic implemented by COM-classes, therefore usually these applications implement business logic by directly querying data from the data model. Another problem resulting from the working method of J-ASP is that, the JSP applications generated will have the same structure as their ASP counterparts, therefore Model-View-Controller patterns must be added afterwards if possible. As a bottom line, it can be said that the use cases of J-ASP are very limited, as J-ASP can only be applied for small-sized and simple ASP applications. As for the working method of J-ASP, it can be said that it is rather simple than sophisticated and can not serve as a good migration strategy for the migration of *Custom Business Web Applications.*

4.2.2 COM4J

COM4J is a Java library which enables interoperability with Microsoft Component Object Model. With COM4J one can generate Wrapper classes (Java classes that implement native methods for COM interfaces) for COM interfaces automatically. By doing so COM4J generates Java Wrapper classes at design-time and automatically creates the necessary native implementation code during run-time [COM4J]. Therefore developers can just reuse COM components in a convenient manner. In the case of ASP-to-Java migration, COM4J can be used to encapsulate COM components that are being used by ASP applications to enable interoperability with Java.

In order generate Java proxies, COM4J can be invoked via a command line tool, thereby one has to specify the location of the COM file, which can be of a type of .ocx, .dll, .exe or .tlb. As an example a COM4J command could look like this:

- ” java -jar tlbimp.jar -o wsh -p test.wsh c:\windows\system32\wshom.ocx”

Where “wshom.ocx” is the COM component, tlbimp.jar is a COM4J jar, wsh is the output folder and test.wsh is the package name.

5 Software Migration

Due to the limitations in an industrial environment, a manual migration* approach in form of a migration guideline is chosen (*see 1.5 Available Migration Strategies and Related Migration Works*). In this chapter, the different migration strategies and methods for manual migration will be presented. Additionally common deciding factors that influence the decision making process of software migration will be named.

5.1 Common Deciding Factors

The common goal of any software migration project is to transform a given software program from a legacy environment to a target environment by reusing or converting existing components as much as possible. Software migration is about how to reuse the most of existing components and how to rewrite existing components for the new environment. In this manner the capability to conduct migration depends on common deciding factors, which are described in [Sneed 1999, p. 24]:

- Costs – money to spend
- Time – deadline when to exchange the software
- Human resource - expertise, experience and quantity of members
- Software state – technical aspects of software such as code structure and modules

Monetary and time limitations are universal factors that affect any software project. Human resource is an important criterion in the decision process of a software migration process. For any migration project a balanced combination of expertise and experience must be found in order to conduct the migration well. This means that both, expertise and experience in the legacy technology as well as in the target technology must be available. Beside those common criteria, the most important criterion is software state, which describes the quality and complexity of a software program. This deciding factor must be analyzed thoroughly as it defines which components can be reused and which migration approach can be applied. Therefore the majority of analysis is spent on this aspect in any software migration project.

5.2 Migration Strategies and Methods

Migration strategies and methods are the tools in a migration concept, which allows the migration concept to rely on for different situations depending on the common deciding aspects in context of a migration task. Thereby a migration strategy describes the overall conduction of a migration concept and migration methods define the ways how actual

* In this way, the term “software migration” will, from this point on, be referred to as in the context of manual migration

code can be migrated in a given case. In the following common migration strategies and migration methods will be described.

5.2.1 Migration Strategies

- **“Horizontal” and “Vertical” migration [MSDN Migration] [Jeenicke]**

  By horizontal and vertical, the distinction between application layers (horizontal) and functionalities (vertical) are defined. While in the horizontal migration strategy, a migration from certain layers of abstraction, such as presentation layer, application layer, data layer, is targeted, it is the goal to migrate whole functional modules in the vertical migration strategy. As for example in the case of horizontal migration, there is a given situation where the application is well separated into abstraction layers and components can be reused seamlessly in these layers. On the other hand if functionalities are separated by functional packages, it can be more efficient to migrate these functionalities, each as a stand alone entity by applying the vertical approach.

- **Incremental and All-At-Once migration**

  Other forms of migration can be identified by the incremental (or “Chicken Little” [Brodie/Stonebreaker]) and the All-At-Once (“Big Bang” or “Cold Turkey” [Brodie/Stonebreaker]) migration approach. Thereby, the incremental approach, describes the migration of an application in incremental steps or migration cycles, where components or functionalities will be migrated incrementally and deployed in parallel to the current application. In incremental approaches the principle of divide-and-conquer is applied to the migration. This approach has the advantage that users will have more time to get familiar with the changes and thus will have a higher acceptance to the application. Also, the application can be tested more thoroughly because of the iterating steps. On the other side the disadvantage lies in the parallelism of the two systems, as by the incremental approach the migrated components and legacy components must be able to run in parallel to each other. This can be difficult in certain situations, such as described in [Jeenicke, p. 11], where target technology and legacy technology are incompatible. The “Big Bang” strategy, describes the process as an all-at-once migration, where all functionalities will be migrated in one big procedure. This approach may be faster, when applied in the ideal circumstances (such as for well documented small sized applications), but however this kind of migration strategy involves great risks as changes to the requirements can not be reverted easily during the migration process. Therefore this kind of strategy is more suitable in small and less complex applications.

5.2.2 Migration Methods

As for migration methods, generally there can be three types of methods named [Sneed 1999, p. 20] [Gimnich/Winter].

- **Conversion**

  Conversion is the approach to re-use functionalities by code-to-code transformation, often times applied when legacy technologies share similarities with the target technologies, such as in case of ASP and ASP.Net. or ASP and JSP. In such cases the syntax of the legacy technology resembles the syntax of the target technology, which makes it easier to convert code from one environment to another.

- **Encapsulation**

  Encapsulation is the method to re-use existing components, by providing wrappers or bridges for the legacy component. Encapsulation can be applied when it is possible to run both, the legacy component and the target application in parallel. Usually encapsulation can be considered the least extensive approach to migrate software, since wrappers and bridges can be easily created. However it is also the least clean solution, since legacy software are not modernized through encapsulation and additionally must be run in parallel with the target system.

- **Re-development or Re-engineering**

  Re-development defines the whole re-creation of software components. Thereby program structures as well as data models can be subject of re-development. Usually re-development is the most resource-consuming approach. Whenever re-development is chosen, there must a situation exist where conversion or encapsulation is not available for migration, such as in cases where source codes are not available or whenever it is not allowed to transform code-to-code (e.g. by 3rd party software).

6 Migration Concept

In this chapter the migration concept will be presented, it is structured in migration stages which describe the analysis, design, implementation and integration phases in a migration process. Additionally the concept will incorporate the design process for Web Service interfaces for legacy *Custom Business Web Applications*, and will provide recommendations to solve the dependencies between legacy application, in order to move from the legacy environment to a Enterprise SOA-based environment.

6.1 A General Migration Concept for Software Systems

The migration concept presented in this paper is based on the migration concept by Harry M. Sneed and described in his book “Object Oriented Software Migration” [Sneed 1999], in which the author describes a migration approach for legacy procedural business software, written in COBOL, from a procedural and monolithic system to an object oriented and distributed system. Thereby the migration process is structured in stages which cover analysis, design, implementation and integration of the migrated system. However, the concept is generally applicable, since it is based on steps that are summarized by [Gimnich/Winter]. Therefore, this concept is based largely on the stages described by Sneed and applies them in the context of Web applications. According to [Gimnich/Winter], a manual migration process generally consists of the following migration workflow: *choosing a migration strategy*, *defining the target environment*, *analyzing differences*, *defining the migration complexity*, *defining and executing transformations*, *deploying the system*, *“migrating employees” and quality assurance*. These steps can be found in the stages of the migration model of Sneed, which is presented below.

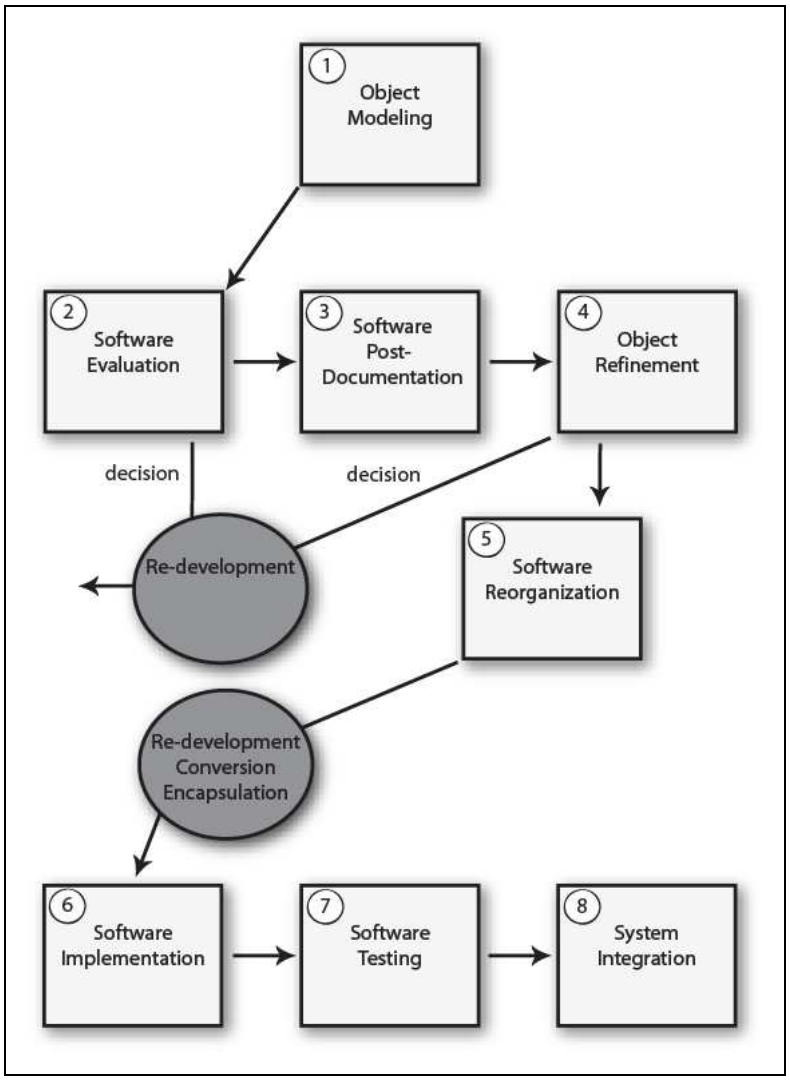

Fig. 6.1.a : Migration Concept by Sneed

The development process described by Sneed is sequential, as it is separated into 8 stages which are conducted stage by stage, therefore it is similar to software development processes like the *Waterfall model* [Waterfall model]. Because of this reason Sneed's migration model inherits the higher risks, as compared to incremental software development models, since the concept implies an "all-or-nothing" characteristic (meaning that systems are developed in one big process), which is not suitable for big and complicated projects, where business requirements may change during the development process. Although the migration of *Custom Business Web Applications* is concerned with small-sized to middle-sized applications, it is better, for the named reasons, to migrate applications in an incremental approach. Following an incremental migration model based on Sneed will be presented.

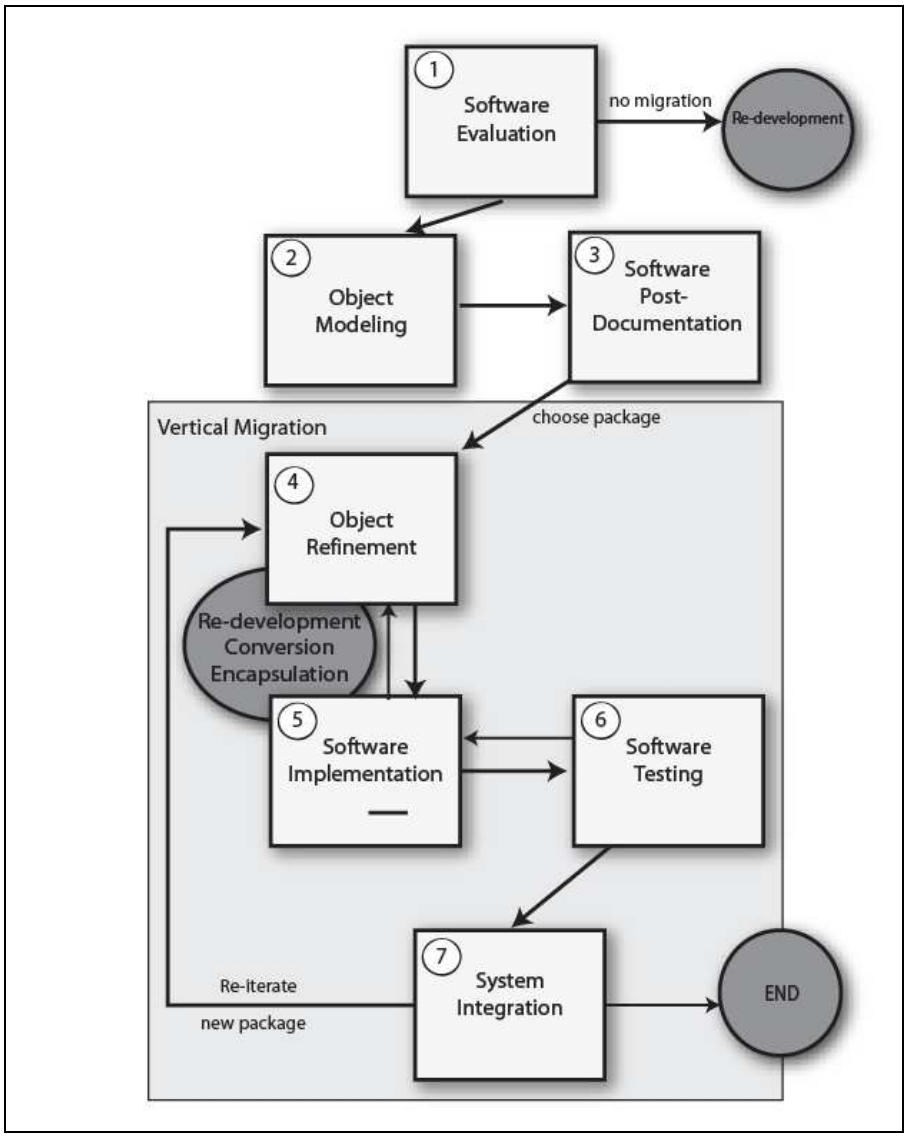

Fig. 6.1.b : Incremental Migration Concept

This migration concept encourages the migration of single functional packages, which can be in coarse or fine grained form, over an incremental development cycle. In the course of the migration process, additional packages can be migrated and integrated together or migrated packages can be further refined in additional iterations. Because the data model is mostly untouched by this migration concept, the migrated packages can be run in parallel to the legacy system.

6.2 The Stages of the Migration Concept

The migration process is structured into 7 stages which span analysis, design, implementation, testing and integration. In the following the migration stages will be introduced briefly.

Software Evaluation

In the *first stage* the legacy system must be analyzed thoroughly, the goal is to identify the structure of the legacy system, in order to decide which migration methods can be applied. Therefore it must be analyzed how Model View Components are implemented, and decided how to migrate them effectively with regard to the limiting factors (such as costs, time, human resources etc.) and the company's requirements (such as technical requirements, functional requirements, software quality etc.). The goal at the end of this stage is a vision of how components of the legacy systems will be looking like in the new system and if it is feasible with the resources given.

Object Modeling

The purpose of the *second stage* is to analyze the functionalities of the system, and to identify its functional modules and business objects. The goal is to define a coarse grained business object model in order to modernize the legacy application and to prepare the interfaces for the Service Oriented Architecture. As the overall goal of the migration of *Custom Business Web Applications* is to reduce heterogeneity and to support reusability and flexibility via a SOA interfaces, it must be kept in mind to analyze which business parts of the legacy system can be exposed as SOA interfaces.

Software Post-documentation

In order to conduct migration properly in the implementation phases, the results of the software evaluation stage must be documented in the *third stage*. The goal is to document functionalities and structures of the legacy system, if it was not done in the original development. The documentation of structures and components can be done in the source code in order to support the mapping of Model View Controller components to the new structure.

Object Refinement

In the *fourth stage* of the migration concept, a functional module or package will be selected to be migrated. The goal of this stage is to design the classes and the packages in preparation for the incoming implementation phase. The focus lies on the refinement of the business objects and the design of packages for SOA interfaces. As the goal is to migrate a functional package to the new environment, Model View Controller components must also be refined and be prepared for the implementation stage. The overall goal is to redefine the business object model so that future business changes can be adapted more flexibly, therefore an object oriented design of classes is favored. However it is not the goal to migrate the legacy application to a fully object oriented Java application as it would reduce the chances to reuse components of the old system. A common problem in this phase is that functional packages have references to each other. Therefore it can not be

assured that functional packages can be migrated isolated from each other, furthermore it must be kept in mind to define functional dummy stubs so that they can be integrated with each other in later phases and migration cycles (in Software Implementation phase of the later migration cycles).

Software Reorganization

The Software Reorganization stage, was described by Sneed as a step between Object Refinement and Software Implementation, which purpose is to prepare the legacy software for migration. It was stated that components exist that do not fit and are not supported by the target environment should be reorganized or redeveloped before handed in order to be migratable to the target environment [Sneed 1999, p.20 - 31]. Such components as in the example of Sneed can be components that operate on legacy data sources such as hierarchical databases, which are not supported by the target technology (such as Java). Therefore it is necessary to reorganize the legacy system before handed (in the sample case the hierarchical data model would be reorganized to a relational data model). However in the context of Web applications and in the migration of ASP to Java, such a case is insignificant since Web applications in both technologies usually operate with similar and common technologies nowadays. Therefore the software reorganization stage as described by Sneed will be left out of this migration concept.

Software Implementation

In the *fifth stage* of the migration concept, the chosen package will be implemented in the target technology. Dependent on the software evaluation stage and the refined business object model, the legacy components of the Web Application will be encapsulated, converted or redeveloped. Additionally dependencies between functional packages can be resolved by implementing the now available functions that were not migrated in earlier migration cycles. Therefore packages can be integrated with each other if necessary. At the end of this stage the chosen package can be tested in cooperation with the already migrated packages.

Software Testing

The testing stage is an essential part of every software development process, thereby the migrated applications functionalities will be tested for errors and bugs. Because of the incremental development process of the migration concept the migrated packages can be tested on the data source, while running in parallel with the legacy system.

Software Integration

The last stage of the migration concept is aimed to integrate the migrated package to the new environment. This includes the integration with the portal system, whereby application pages must be linked with the portal’s application repository or

Single Sign On functionalities must be implemented. Furthermore SOA interfaces can be registered and integrated with the companies Service Oriented Architecture. After the migrated packages are integrated with the new environment the migration process can be furthermore conducted incrementally or can be finished at this point.

6.3 Software Evaluation

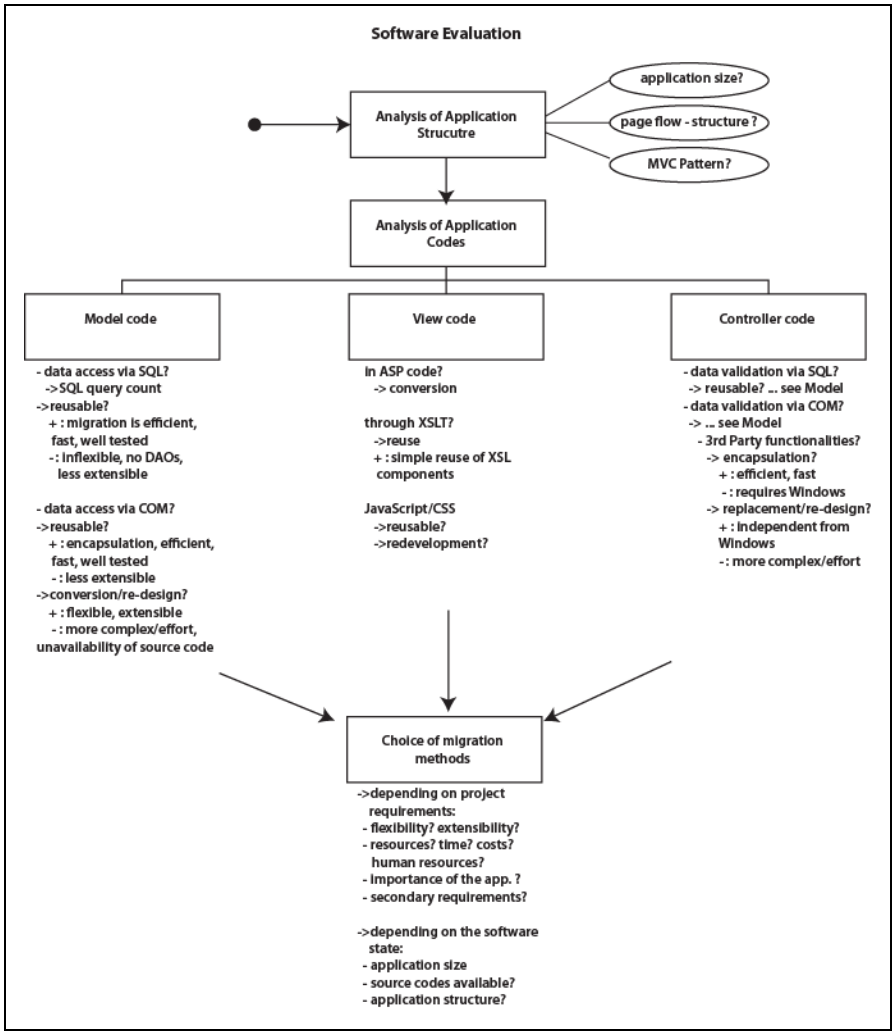

Fig. 6.3.a : Software Evaluation

The software evaluation stage starts with the analysis of the application. It has to be determined how big and complicated the application is, therefore the source code and the Lines of Code (LOC) must be checked. After this short analysis the structure of the application must be analyzed. It has to be checked in which form the application is linked with its data model, e.g. oftentimes legacy *Custom Business Web Applications* found in business departments are connected with a simple database management system such as Microsoft Access. As the data model usually relates to the complexity of an application, it can serve as an indicator for the structure of the application. As the size of the application defines the migration strategy, it can be already decided at this stage whether

the application should just be redeveloped or not, this is the case when the application is of smaller size and have simple complexity.

Of greater importance is the analysis of the MVC components of the legacy application. It should be determined in which programming style (*see 13.2 MVC & Programming Styles in ASP Web Applications*) the application was developed, whether it has a mixed MVC style or if MVC components are strictly separated. In the following the MVC components must be analyzed in which technology they are implemented. As for the View components the most common implementations involve ASP, XML/XSLT, Cascading Style Sheets (CSS) and JavaScript. As ASP and JSP (the Java equivalent to ASP) are quite similar in syntax and capabilities, and XML/XSLT, JavaScript and CSS are reusable in JSP, it can be said that the mapping of View components from ASP to JSP is rather trivial. The only aspect that has to be analyzed is that whether View components must be adapted due to company requirements (e.g. new corporate design of portal system), and following which components of the View can be reused or converted (adapted).

As for Controller components, in ASP Web Applications these components can be implemented in ASP fragments which manage the control flow of the application for example by validating input parameters of the ASP page request. In ASP Web applications such validation is usually done by accessing the data model either via SQL queries or COM. In the same manner Model components must be analyzed, whether they are modeled via COM components (in an object oriented form) or modeled directly by the data model (e.g. the data model stores session or workflow data). In this sense the quantity and complexity of SQL statements must be analyzed, in order to decide whether SQL statements can be reused to query data or be remodeled via new Data Accessing Objects (DAO) in the target system. Usually this is a choice between better software design (redevelopment) and migration effort (reuse). In the case of COM components, firstly it must be checked whether source codes of the original COM component exist or whether the required COM components are developed by an 3rd party. In case that no source codes are available COM components can only be wrapped to be reused by Java or be redeveloped. If source codes are available, the COM components can also be converted to Java. Encapsulation is the most efficient way to migrate legacy codes to Java, however this approach is less flexible in the long run since functionalities must be redeveloped in the Java wrapper classes (or changed in the original code) if the company request changes in the Model. This solution is therefore suited for cases where migration must be conducted immediately and future changes in the business logic occur not frequently. In the case of a conversion, the components would be more flexible and more efficient, which is due to the absence of Java-to-COM bridges. However the effort spent to convert COM components to Java would be significantly higher. As the last alternative, COM components can be completely redeveloped in Java. This solution gives the most freedom in design, but requires the most effort to migrate functionalities from the legacy application. The choice whether it is better to encapsulate, convert or redevelop COM components is dependant on the project requirements. As for example in the case where COM components are not accepted on the target server, encapsulation of COM would be not a choice, therefore conversion or redevelopment would be suitable

for platform independent solutions. Additionally, in regard to 3rd party COM components, it must be analyzed which services (e.g. LDAP, Middleware etc.) could be wrapped by Java or replaced by equivalent Java services.

Depending on the limiting factors costs, time, human resources and project requirements one can decide which approaches to follow and where the focus should be, which can be either on efficiency (reusability or conversion) or quality (redevelopment). In the common case View components can be reused (JavaScript, XML/XSLT) and converted (ASP to JSP), Controller components can be reused (SQL, COM) and converted (ASP to Servlet) and Model components can be reused (SQL, COM).

6.4 Object Modeling

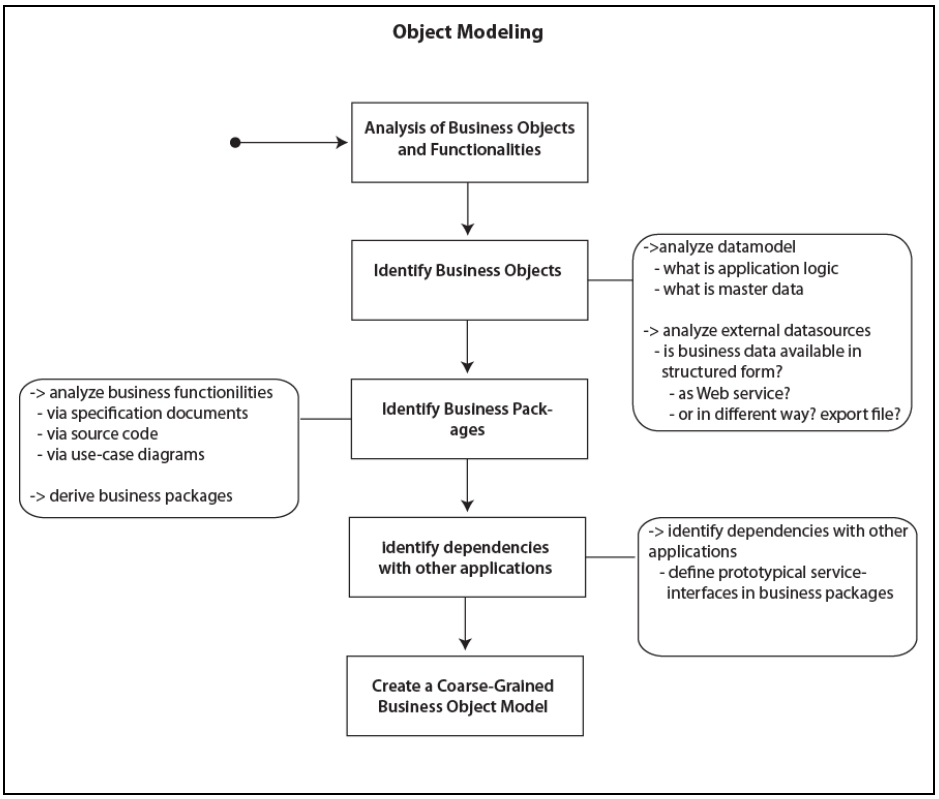

Fig. 6.4.a : Object Modeling

After the software evaluation stage the software's technical state is analyzed and well understood, therefore one can decide which migration methods can be applied for the parts of the application. However in order to bring the legacy system up to date, the application structures must be modernized (application of MVC pattern) and adapted to the new enterprise environment (integration with SOA). Therefore the task in this stage is to analyze the functionalities of the legacy system and to identify business objects and functional modules. Thereby business objects are entities on which the application operates.

Business objects are essential entities in an object oriented* application structure, which is necessary in the migrated application in order to be flexible and extensible for future requirements. Because oftentimes legacy applications are not object oriented and not flexible, as they do not have an object oriented business object model and do access data structures via SQL queries only (*see Fig. 3.1.b : ADO.DB example in ASP*), it is necessary to map the underlying data model to an object oriented model at application level. Therefore one has to analyze the applications master data tables of the data model and retrieve business objects from this information. Otherwise, when COM components are used to access the data model, COM objects have to be analyzed. The goal of this step is the generation of a coarse grained business object model that is object oriented in order to cope with future changes and demands more flexibly.

When business objects have been identified and a basic business object model has been generated, business packages must be defined. Business packages are functional modules that group a set of related functionalities together. This can be done by analyzing the code of the legacy applications, but as well by studying specification documents and use case diagrams, if available. The purpose of this step is to define packages that are loosely coupled with each other so that they can be migrated incrementally (as far as possible) in the later migration stages. Additionally the goal is to allow the packages to be exposed as SOA-based interfaces for the common reuse of functionalities.

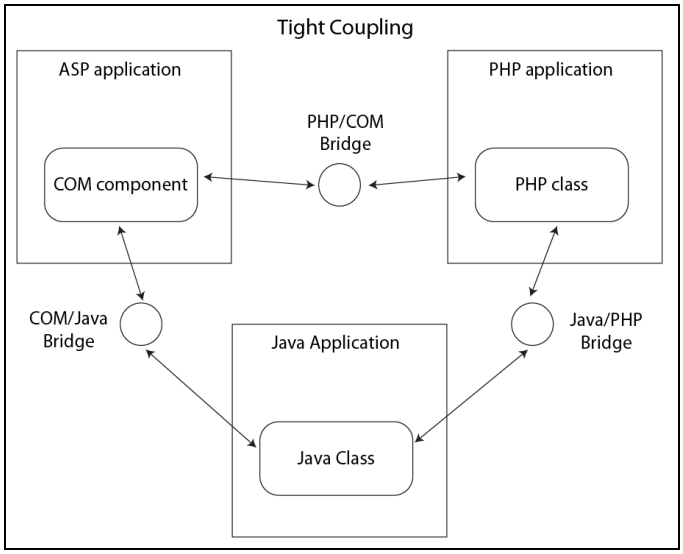

Fig. 6.4.b : Tight coupling between Web applications

Since the goal of the migration process is to reduce heterogeneity and dependencies between applications (*see Fig. 6.4.b : Tight coupling between Web applications*), it must be identified which functionalities are being used by external systems. The goal is to define business packages for these functionalities, and expose them as services (as Web

* In the context of Web applications, object orientation is often applied for the representation of database entities in a structured manner. Object orientation of the business object model is a requirement for the application to return data entities in a reasonable manner in SOA interfaces.

services) in an SOA-based manner, so that those services can be integrated into the Service Oriented Architecture in order to be re-used and accessible in a more uniform way. This will improve efficiency of the company's IT infrastructure as less middleware/bridges have to be maintained. As described in *1.2 Problems of Heterogeneous Environments*, a problem of *Custom Business Web Applications* is redundant or inconsistent data. This was caused by the isolated development of systems and inappropriate methods used for data transfer (such as data file transfer *TICOS, see Fig. 7.1.c : TICOS data export via files*). These problems should also be addressed in the creation of a business object model during the identification of business objects, thereby it must be identified which data are being imported or exported. It should be denied to store global master data locally or redundantly. Usually a company's SOA-based IT environment is based on a central interaction platform (the portal system) and a central data provider, which provides integrated systems with common master data such as employee information/business department codes etc, therefore these kind of data shall not be stored locally. Therefore legacy data transfer mechanism shall be re-defined and modernized if possible, in this cases the communication between departments is essential to be able to design appropriate interfaces for common data exchange and common data structures.

In summary, the overall goal of this stage is the generation of a coarse grained business object model and related business packages. It has the purpose to solve problems of heterogeneity and redundant data in the legacy environment and is aimed to build a foundation for the later refinement (e.g. definition of Web service interfaces, methods etc.) of the business object model in the later design and implementation phases.

6.5 Software Post-Documentation

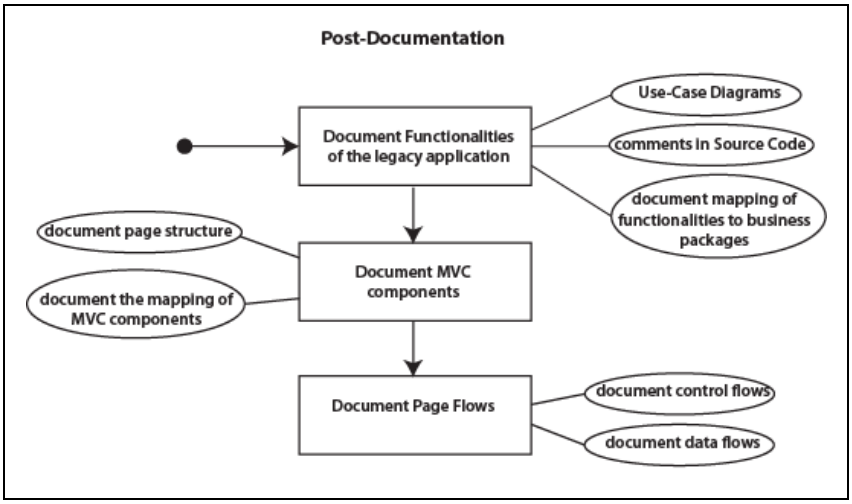

Fig. 6.5.a : Software Post-Documentation

In the Software Post-Documentation stage, the goal is to document the technical characteristics of the legacy application in order to support the implementation phase of the migration and improve the understanding of the legacy application for all team members conducting the migration. Therefore the decisions made and components

identified in the software evaluation process must be documented. This can either be done in specification documents or in the source code of the application. Important aspects that must be documented are MVC components and business functionalities, especially those that are subject to be exposed as service interfaces. In the documentation, Model, View and Controller code fragments must be distinguishable by comments and tags. Furthermore it should be made apparent how these codes will be dealt with in the implementation phases of the migration as available options were already discussed in the software evaluation stage. The 3 options that are available for migration are redevelopment, conversion and encapsulation/reuse, therefore the application of these options on application components, must be documented in a document such as in a specification document.

```
<%
‘ViewMemberList.asp
Response.ContentType="text/xml"

‘MODEL PART – implement connects to DB and gets all Members
Dim sConnection, objConn, objRS ‘ODBC Objects
sConnection = “DRIVER={Microsoft OODBC for Oracle};” &_
              “SERVER=localhost; DATABASE=ticos;” &_
              “UID=root;PASSWORD=;”

Set objConn = Server.CreateObject(“ADODB.Connection”)
objConn.Open(sConnection)
Set rs = objConn.Execute(“SELECT member_id, name, firstname” &_
                                .
                                .
                           “FROM tbl_members”)
                                .
                                .
‘CONTROLLER PART – creates and maps the model data to the XML
‘and pass ‘control relevant Data if necessary
xmlString = “<?xml version='1.0' encoding='UTF-8'?>” &_
            “<?xml-stylesheet type='text/xsl' “ &_
            “href='viewmemberlist.xsl'?>”

Response.Write(xmlString)
Response.Write("<Members>")
                                .
                                .
do until rs.EOF

      Response.Write(“<Member>”)
      for each x in rs.Fields

            Response.Write(“<” & x.name & “>” & x.value &_
                           “</” & x.name & “>”)
      next
      Response.Write("</Member>")
      rs.MoveNext
loop
                                .
                                .
Response.Write("</Members>")
%>
```

Fig. 6.5.b : Example of source code documentation

In addition to the documentation of MVC components and migration methods, the functionalities of each component must be documented, if they are not already done by the original developers. Since the migration of *Custom Business Web Applications*, in context of an integration to a SOA-based IT environment and portal platform, involves additional functionalities (e.g. portal integration via Single Sign On or Web service definitions), those functionalities must be documented in any case in the software post-documentation stage.

By the end of this stage, the result will consist of a specification document, which gives information about functionalities and functional modules of the legacy application and new functionalities required by the requirements of the company. Additionally documentation at source code level is available which helps to identify MVC components for the mapping components during the implementation phase. In the following stage, the object refinement stage, the documentations will help to refine the business object model and to design interfaces.

6.6 Object Refinement

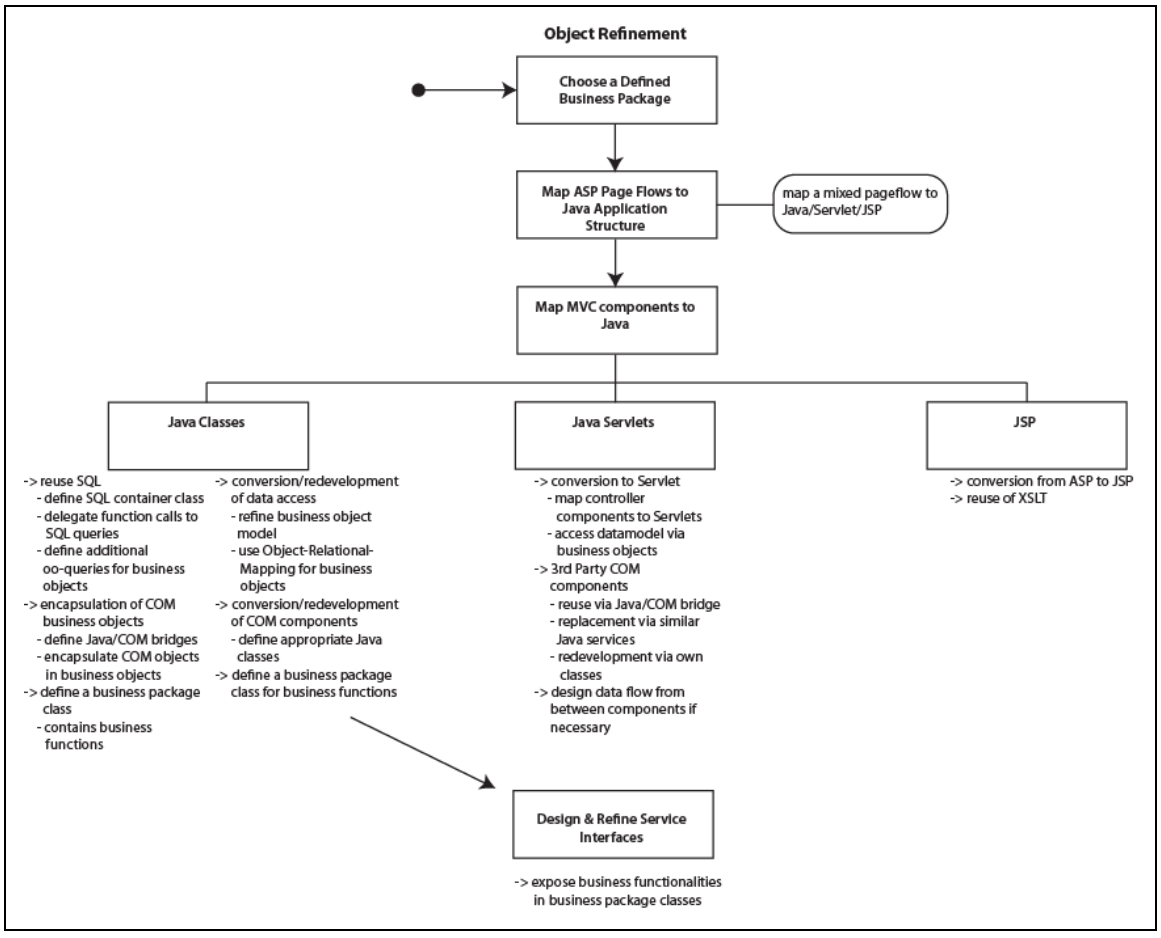

Fig. 6.6.a : Object Refinement

At this stage of the migration process, the legacy application will be prepared to be migrated. Starting from this stage the legacy application will be migrated in an incremental cycle by migrating functional packages, package by package. The goal of this stage is to refine the business object model, defined in the object modeling stage, with the purpose to modernize the application and solve the problems related to the legacy technologies.

6.6.1 Vertical Migration vs. Horizontal Migration

As an incremental migration strategy, the vertical migration strategy is suited at best for the purposes of *Custom Business Web Application* migration, since it allows the application to be separated into functional packages and to be deployed in parallel to the legacy application. The advantage of vertical migration is that, it separates the migration process into incremental cycles in which small packages can be migrated and tested incrementally. In comparison the *big bang*, it is more flexible and less error prone during the migration process, since the *big bang* approach migrates applications in an all-at-once manner which is attached with higher risks. While the *horizontal* approach is also an incremental migration strategy, it can not be applied well to Web applications, since different technologies in Model, View or Controller components (such as a combination of JavaBeans, Servlet and ASP) will cause incompatibilities and introduce additional work. This is the case when for example a Servlet instance wants to pass its session data to a ASP view component. For this reason the most suitable migration strategy for the migration of Web applications is the vertical approach, where Model, View and Controller components of a functional package will be migrated and tested in one migration cycle.

In a related work, Jeenicke has also applied the vertical migration in his approach and concluded from his experience that it was “quite easy to integrate rewritten parts with each other” [Jeenicke p.13]. However this can not be generalized, as the migrated functional components may have complex dependencies with each other. This aspect leads to a main deciding factor which affects the efficiency of vertical migration, which is how to separate business packages from each other. In an ideal case the packages are independent from each other, so that the migration of one package can be viewed as a complete process. On the other side, if packages have dependencies to each other, the packages have to be implemented with temporary stubs so that packages can be migrated and deployed. Following the process those stubs have to be changed to real functionalities when the dependent functions are available. Therefore it is a task of the implementation and integration phases to handle such problems.

6.6.2 Model View Controller Structure

The MVC components in an ASP-based Web Application must be mapped to appropriate technologies on the Java platform. As a possible configuration, a combination of JavaBeans, Servlet and JSP is considered as the most suitable solution for the direct migration of ASP-based Web applications. While Java frameworks such as JSF [JSF], Seam [Seam] etc. provide a modern workflow for Web development, they add additional complexity to the migration process, which will not be dealt in this paper. Therefore JavaBeans, Servlet and JSP will be used to map equivalent ASP components, as the direct mapping of those technologies is simpler as when certain framework mechanics are involved additionally. As a result the object refinement stage is focused on the design and refinement of components in the named technologies, whereas JavaBeans will be used for Model components, Servlet used for Controller components and JSP used for View components.

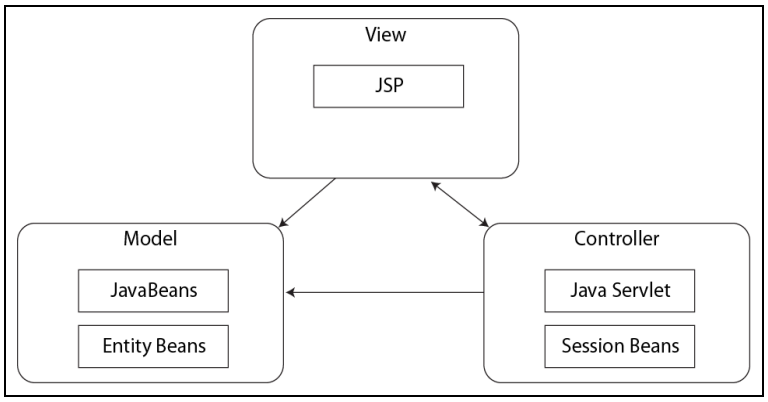

Fig. 6.6.2.a: MVC structure in Java

6.6.3 Refinement and Design of MVC Components

In the software evaluation stage the legacy components were analyzed and in the object modeling stage a new coarse grained model was defined. The goal of this stage is to combine the information and to refine the business object model (of the chosen package) and design the MVC components involved with it. As analyzed in the software evaluation state, the possible migration methods for different components are dependent on the software state of the legacy application and the project requirements.

Depending on the programming styles of ASP Web applications (*see 13.2 MVC & Programming Styles in ASP Web Applications*), MVC mappings of legacy applications follow different designs at this stage. In the following the different refinement and design scenarios of MVC components will be described depending on programming styles that may occur in ASP applications.

ASP Style	Page Structure	Model	View	Controller
Completely Mixed MVC	• Model, View & Controller elements are mixed together within an ASP file	• No object oriented business object model • Business functions in one page with other components • Data access via SQL	• Data entities from Model are presented with ASP and HTML	• Control flow is based on SQL queries to the data model or with COM objects • External functionalities with 3rd party COM components
Separated View	• Business functions are mixed with control flow functions in one page • View is separated to a dedicated page	• No object oriented business object model • Business functions in one page with Control components • Data access via SQL queries	• Data is presented in a separated ASP page with SQL queries • View can be additionally separated with XML and XSL files	• Control flow is based on SQL queries to the data model • External functionalities with 3rd party COM components
Complete Separation of MVC	• Business functions are separated into COM components • Control flow functions are separated from View pages • View is separated	• COM components access data model • Business functions are implemented in COM components • Data access via COM DAO	• Data is presented in a separated ASP page with SQL queries or COM DAO • View can be additionally separated with XML and XSL files	• Controller pages manage control flows of the application based on input parameters and business logic • Control flow is managed in separated files

Fig. 6.6.3.a : MVC-styles in ASP applications

Completely Mixed MVC Style

In a completely mixed structure of an ASP application, the data access is usually realized via inline SQL queries. In such a structure, there is actually no Model or object oriented business object model existent, therefore the data records returned from such queries must be mapped to object oriented business objects if possible. Like the Model the Controller and View code is implemented with a combination of ASP and SQL queries. In order to migrate such an application structure to a modern SOA-based environment, the applications structure must be mapped to a MVC-based design and provide an object oriented business object model. This can be done by reusing, converting or redevelopment of components as analyzed in the software evaluation stage. In the following the different migration methods for each component will be described.

Encapsulation

The most efficient way to migrate SQL statements (which involves Model, View and Controller) is to reuse the statements completely as they are, since the statements are already well tested in practice. The idea is to copy these SQL queries to a central Java class from where Model, View and Controller components can call those queries like in the way they were used in the legacy application. However the result set of queries which return attributes of a business object (or a set of business objects) should be objectified if possible, which means that the containing Java class must provide a method which returns the results of the query as a business object type (*see Fig. 6.6.3.b : Wrapping of SQL queries – Model component*).

```
<% 'Sample ASP code - EditMember.asp

'get Parameters
Set member_id = Request.Form("member_id")
query = "SELECT member.name, member.lastname FROM member" &_
        "WHERE member.id =" & member_id
Set results = adoConnection.execute(query)

... write results to HTML/XML ...
%>
```

```
//SQL Query wrapped as method in Java

public QueryList{
...

   public static Member edit_member_asp_query1(int member_id){
      ...
      String query = "SELECT member.name, member.lastname" +
      "FROM member WHERE member.id =" + member_id;
      ResultSet rs = jdbcConnection.execute(query);
      ...
      Member member = new Member();
      member.name = (String) rs.getValue(0);
      member.lastname = (String) rs.getValue(1);
      return member;
   }
...
}
```

```
<% //The JSP View - EditMemberView.jsp

  //get Parameters
  int member_id =(int)this.getRequest.getAttribute("member_id");
  Member m = QueryList.edit_member_asp_query1(member_id);

  ... write results to HTML/XML ...
%>
```

Fig. 6.6.3.b : Wrapping of SQL queries – Model component

However those kind of design, would only apply to SQL queries that result in attributes of a business object. For SQL queries that result in different values of different business entities, such as in the following:

```
<% 'ASP Sample – sample.asp
   'Controller code

Dim member_name, role_name

query = "SELECT member.name, role.name" &_
        "FROM member INNER JOIN role" &_
        "ON member.role_id = role.role_id" &_
        "WHERE member.member_id =" & member_id
Set rs = ado_conn.execute(query)

... member_name = rs(0).value ...
... role_name = rs(1).value ...

if role_name = "manager" then
   ... Manager Code ...
End if

if role_name = "employee" then
   ... Employee Code ...
End if
%>
```

```
//Controller mapped to Java Servlet

public class Sample_ASP_ControllerServlet extends HttpServlet{

 void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
 {... get Post Parameters ...

  ResultSet rs = QueryList.sample_asp_query1(member_id);
  ...
  String member_name = rs.getAttribute("member.name");
  String role_name = rs.getAttribute("role.name");

  if(role_name.equals("manager"))
  {... Manager Code ...}

  if(role_name.equals("manager"))
  {... Manager Code ...}
}
```

Fig. 6.6.3.c : SQL queries in Controller component

The result set of the query can not be objectified and must be used as it is. This kind of SQL query is typical for SQL statements of Controller components. Therefore those queries will be reused in Java Servlets of the target applications often. As mentioned before, this way of reuse is the most efficient way to migrate SQL components. The advantages are that, SQL calls from the legacy application can be mapped to MVC components of the target application very easily, without any redesign of queries and

function call structures. Additionally it provides a semi-object oriented programming style to the application, since business objects queries return the actual objects instead of data records. This can be used when implementing SOA interfaces where object structures must be returned.

As Controller components may utilize external functionalities provided by 3rd party COM components, those components must be encapsulated or their functionalities must be replaced by equivalent Java components (redevelopment or replacement). Conversion of 3rd party components is not feasible, since those components are only available in binary form. The only way to convert COM components without the original source code is to re-engineer the binary files, but this would be far less efficient then wrapping components into Java classes. In the following a target design for legacy components to a MVC based Java design is presented.

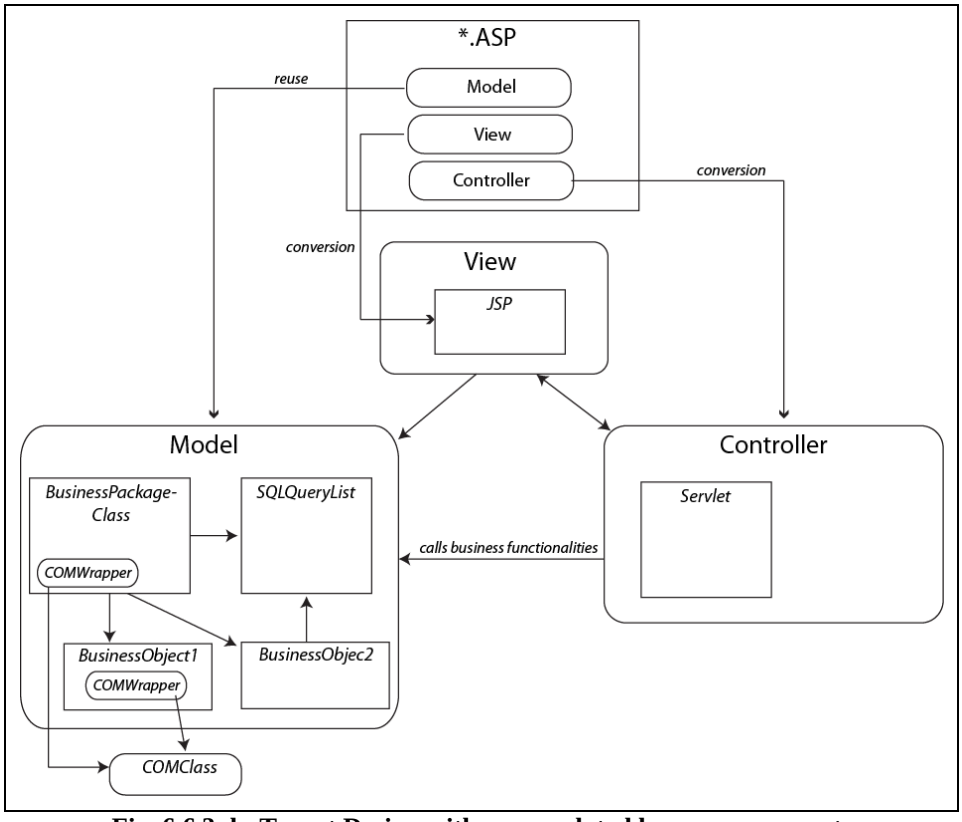

Fig. 6.6.3.d : Target Design with encapsulated legacy components

As business functions, control flow functions and presentational functions are mixed together into a single ASP page, those 3 components must be separated from each other and stored separately in different files. View components in a mixed MVC style application are implemented with ASP, therefore they can be mapped to JSP as both technologies provide a set of very common tags. Code that manages the control flow in legacy applications will be mapped to Java Servlets, where SQL statements and COM components are wrapped into Java classes which allow a nearly equivalent style of programming as in the original state. Those container and wrapper classes can be called from any layer (Model, View, Controller) of the application, therefore the original

function call structures of the legacy application will remain. Model components, such as business objects and business functionalities that were implemented in the same ASP pages with Controller and View elements must be mapped to Java classes and Java packages. As the vertical migration strategy encourages the migration of functional packages (modules), it is appropriate to define such a package as a central Java class, which contains the business functionalities of the module. In regard to business objects, the corresponding data structures must be mapped to Java classes, in order to provide an object oriented way to work with business objects.

Conversion & Redevelopment

In the design approach based on reusability, the focus is set on efficiency and fast migration of legacy applications. This approach is suited for complex and unstructured applications, as most components (SQL, COM) can be reused by encapsulation and used in a similar way as they have been used in the legacy application. However the resulting design of such an approach may not be flexible and extensible from the modern perspective of Web development, as there is no complete object orientation behind the design. As an example, when reusing SQL statements, there is no way to access business data in an object oriented way such as:

```
//As Java Code
//Object oriented business object model
Project p = member.getProjects(0);
Customer c = p.getCustomer();
```

Fig. 6.6.3.e : Object oriented calls on the data model

The reason behind this lies in the SQL statements that are used in the ASP applications. In the example above, the query to fetch the *Customer* from the *Project* which is assigned to a *Member*, is split into 2 object oriented method calls. However in practice, such kind of query is not separated in the ASP application. Therefore when reusing SQL statements, the whole statement is encapsulated into a Java containment class (*see Fig. 6.6.3.g : Encapsulation of SQL statements*).

```
'ASP CODE
'Query for Customer with 2 parameters member_id and project_id
Dim query

query = "SELECT * FROM customer WHERE customer.id =" &_
        "(SELECT project.customer_id FROM project" &_
        "WHERE project.id = " &_
          "(SELECT member_projects.p_id FROM member_projects" &_
          "WHERE member_projects.member_id =" & member_id " &_
          "AND member_projects.project_id =" & project_id & "))"
```

Fig. 6.6.3.f : SQL statement with subqueries

As a result the encapsulated data access method remains in the procedural form.

```
//Java CODE
//Reuse of SQL queries

Customer c = SQLQueryList.getCustomer(member_id, project_id);
```

Fig. 6.6.3.g : Encapsulation of SQL statements

A solution to this problem is to convert and redevelop SQL queries found in the legacy application. This approach would allow the target application to be more flexible and extensible to future changes, however it is less efficient and slower as the encapsulation approach. Following such migration approaches every single SQL query must be analyzed and mapped one by one. In such a case the business object model designed in the object modeling stage, will be refined with methods that allows such objects to query its relationships with their neighbors. Therefore it should be mentioned that it would be equally efficient to design business object classes based on an Object Relational Mapping framework such as Hibernate, if the data model allows such a design. As in regard to business functions, such kind of combined SQL queries can also be found in business functions of legacy applications. In such cases the SQL queries can be analyzed and be replaced by object oriented calls from business objects. In the following the migration mapping for mixed ASP applications based on conversion and redevelopment will be presented.

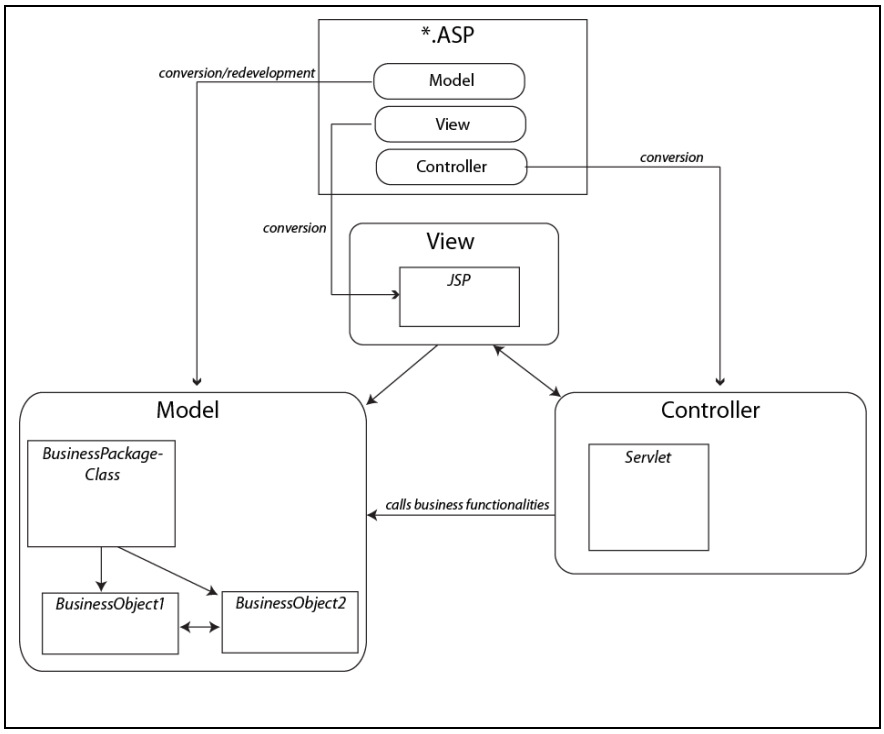

Fig. 6.6.3.h : Conversion and Redevelopment

Separated View Style

Comparable to ASP applications that are implemented in a mixed MVC style, this kind of applications have a similar structure of Model and Controller components. As Model and Controller components are usually implemented in the same ASP file and are based on SQL queries and 3rd party COM components, the possible migration methods remain the same as with the mixed MVC kind of applications, which are either encapsulation of SQL and 3rd party COM components or conversion/redevelopment of SQL queries in the business object model. The difference between those two styles lies in the View component, which is separated into a dedicated file in this kind of style. As like with Model and Controller components, View components in this kind of style may or may not be reusable, depending on the way how they are implemented. In the first regular form View components can just be implemented with regular ASP tags, such like in the following example:

```
<html><body>
<% 'View component, get result data from Controller components
   'which are appended to the session object

Set membernameList = Session("membernameArray")
For membername in membernameList
  Response.Write("<h1>Membername: </h1>" & membername & "<br></br>")
next
%>
</body></html>
```

Fig. 6.6.3.i : View component in ASP

When this form is encountered the only migration method available is to convert the ASP page to a JSP page. Following the example above the JSP page would be mapped like this:

```
<html><body>
<% //View component, get result data from Controller components
   //which are appended to the session object
   //Member is migrated to a JavaBean structure
   ArrayList<Member> memberList =
                        session.getAttribute("memberArray");
   for(Member member: memberList){
%>
<h1>Membername: </h1> <%= member.name %> <br></br>

<% } %>

</body></html>
```

Fig. 6.6.3.j : View component converted to JSP

In the second form View components can be separated into a XSL part. In such a case, the mixed Model/Controller page will return the data records in an XML structure (goal is the structuring of those data records) which will be processed via XSL Transformation to a HTML page.

```
<html><body>
<% 'sample.asp
   'Data is fetched via SQL query
   Dim xmlString
   xmlString = "<?xml version='1.0' encoding='ISO-8859-1'?>" &_
      "<?xml-stylesheet type='text/xsl' href='sample.xsl'?>"
   Response.Write(xmlString)

   Set rs = ado_conn.execute(query)
   Response.Write("<Members>")
   For x in rs
   Response.Write("<Member><name>" & x.value & "</name></Member>")
   next
   Response.Write("</Members>")

%>
</body></html>
```

```
<!-- sample.xsl -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html><body>
    <h1> Memberlist</h1>
    <table>
     <xsl:for-each select="Members/Member">
       <tr><td>
       Membername: <xsl:value-of select="name" />
       </td></tr>
     </xsl:for-each>
    </table>
   </body></html>
```

Fig. 6.6.3.k : Sample of ASP & XSLT

Since XSLT is not dependent on ASP, the View component can be reused as it is in the target application. This solution is the fastest and most efficient solution to migrate View components. However the generation of the necessary XML structure must be mapped from the legacy applications ASP pages to the target applications Java Servlet. In the following the design and mapping of View components will be presented.

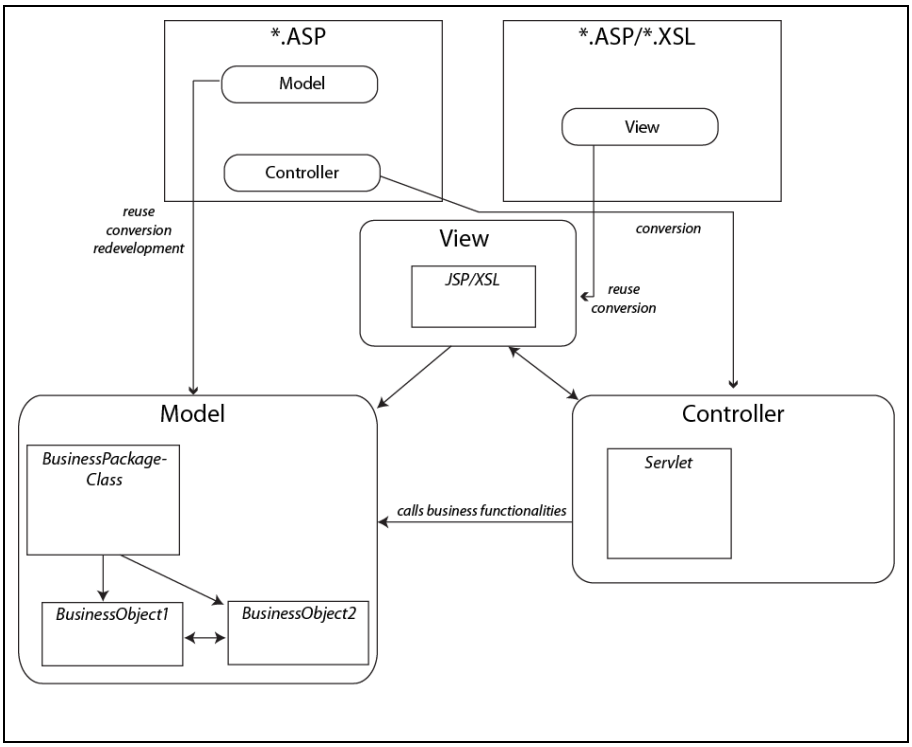

Fig. 6.6.3.l : Target Design with separated View

Separated MVC Style

In ASP applications where a strict separation of MVC components is applied, the object model and business logic is implemented by COM components. Through this design Controller and View components of legacy ASP applications can make use of an object oriented design of the object model. Usually Controller components are implemented in separated ASP pages that validate incoming values of requests and manage the control flow of the application. View components can be implemented in dedicated ASP pages or in separate XSL files.

The fastest and most efficient migration method for this kind of applications is a combination of encapsulation of COM components in the Model, conversion of ASP pages and encapsulation of SQL statements in the Controller components and a conversion or reuse of View components. Following this method, business objects should be designed in such a way that they encapsulate their equivalent COM objects. By this design, methods of the business object can delegate their functionalities to the COM object or implement new functionalities by their own. Furthermore object creation and destruction of the business object should be also delegated to the COM object, this allows the business object to be used transparently from any location. Additionally business functions can be encapsulated into business package classes, by the same pattern.

```
//Encapsulation of COM components

class SampleClass{
   private COM_sampleClass com_sampleClass;

   public SampleClass(){
      com_sampleClass =
                    COMWrapperClass.create("COM.SampleClass");
   }

   public String method(){
      return com_sampleClass.method();
      //otherwise functionalities can be adapted
   }

   public void newMethod(){
     //new functionality can be implemented here
   }
}
```

Fig. 6.6.3.m : Encapsulation of COM business objects

As an alternative to encapsulation, COM components can be converted or redeveloped to Java, if the original source code is still available. In case of conversion, when source code is available, COM components can be implemented by different programming languages. Therefore it would exceed the scope of this paper to describe the conversion of COM components in detail. However, when source code is available, the conversion can either be done manually or automatically via code converters from various programming languages to Java. Through automatic code conversion the structure of the legacy code will be maintained therefore it can be disadvantageous if the original code was implemented in a procedural style. Hence it would be beneficial to convert the code manually and map the procedural functions to object oriented classes. In the following the possible designs for the target application structure can be found.

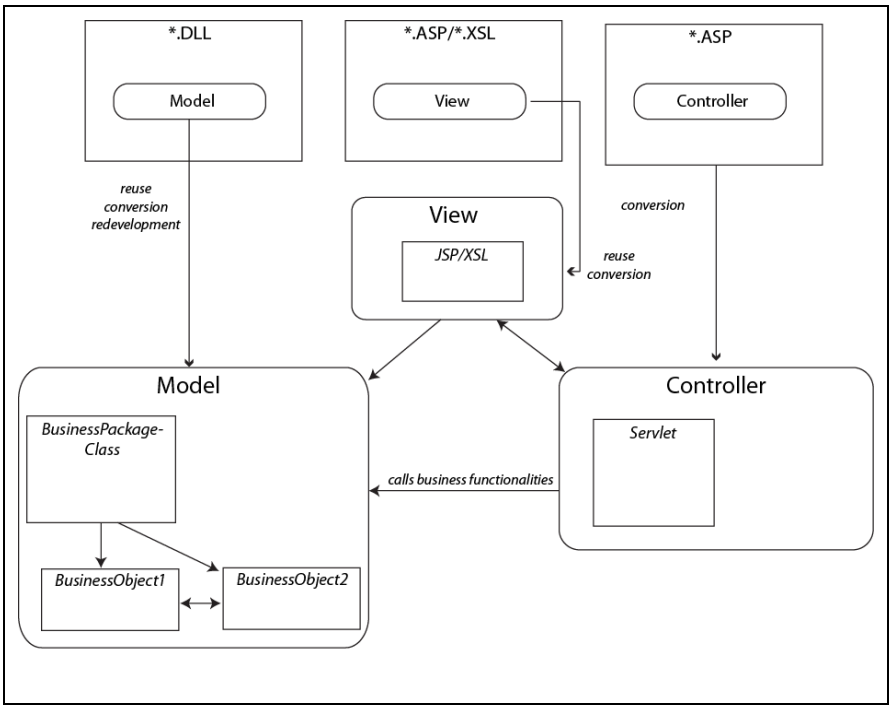

Fig. 6.6.3.n : Target Design for separated MVC

6.6.4 Design of Service Interfaces

At the last step in the object refinement stage, the analysis and design of SOA-based interfaces will be conducted. In order to design interfaces suitable for the Service Oriented Architecture it must be analyzed if the exposed interfaces should be fine grained or coarse grained, this is usually defined by the company's IT-policy and hence should be considered in this step. Additionally it must be analyzed which functionalities must be exposed explicitly according to the primary requirement of the company, which states that SOA-based interfaces must be provided to reduce the dependencies between applications (*see Fig. 1.2.a : Tight coupling of applications in a heterogeneous environment* [IBM SOA]). Therefore it must be analyzed if business functions are being used by external applications. This can happen between applications that reside on the same server and reuse functionalities via a COM component of the actual application, but also via middleware that access the COM component remotely or via DCOM (*see 4.1.4 Component Object Model*). As a remainder, the technology of choice for implementing SOA-based interfaces is Web services, Web Services provide loose coupling between applications in comparison to similar technologies such as DCOM [Web Service vs. DCOM].

After the analysis on dependencies with external applications is done, the respective interfaces can be designed. As already mentioned the granularity of such functions is dependent on the requirement of external applications but also on the IT-policy that the company follows. Therefore the ideal design is a service interface, which can be reused by all dependent external applications and provides potential uses for future applications. In respect to the location, service interfaces should be provided via the business package

class. In regard to the functionalities reused by the business package methods, such as COM methods or SQL queries, service interfaces can be composed of multiple functionalities or just encapsulate a single function and expose them as a service interface. In the following illustration the design of SOA-based interfaces will be presented.

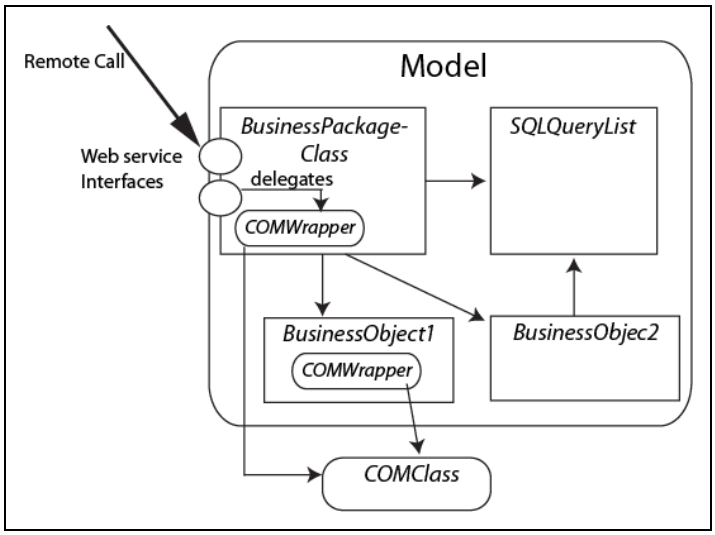

Fig. 6.6.4.a : Service Interface design

6.7 Software Implementation

In the software implementation stage, the chosen package will be migrated. Therefore, depending on the design and refinement made during the object refinement stage, components of the legacy application will be mapped to the new target design. The conduction of migration is done manually, by converting ASP code to Java, reusing SQL queries in Java classes and mapping ASP files to a MVC-based file structure. In the following the conduction of migration will be presented.

6.7.1 Mapping of ASP files to Structures

As analyzed in the software evaluation stage and designed in the object refinement stage, the ASP applications page structures must be mapped to Java classes, Java Servlets and JSP pages. This has to be conducted in this stage in accordance to the design from the object refinement stage. This means that the entire pageflow of the chosen package must be mapped manually to the target structures, using the same mapping pattern that was identified in the object refinement stage (*see 6.6.3 Refinement and Design of MVC Components*). This step has to be done in the beginning to create the package structures that will lead to migration. In the following an example of such pageflow mapping will be presented.

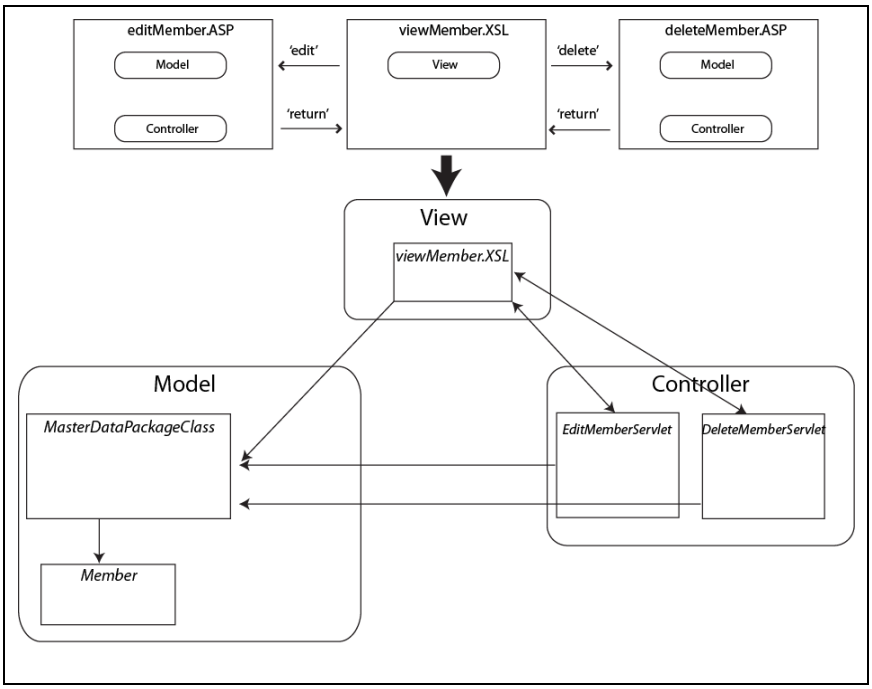

Fig. 6.7.1.a : Example of pageflow mapping from ASP to Java

6.7.2 Business Package Class & QueryList class

Independent from whether certain components will be reused, converted or redeveloped, a business package class must be created for the chosen migration package. This business package class acts as a central class that provides package related business functions to the application and other applications. Therefore the business package class will not only contain business functions but also interfaces to communicate with external applications. During the implementation step, the business package class will be filled with business functionality methods.

The second class which becomes necessary (when SQL queries are used) is the QueryList class, which encapsulates SQL queries of the legacy application. This class will contain all the SQL queries that are used by the legacy application, if it was decided to reuse all the SQL queries that are existent in the legacy application. Model, Control and View components can use this class and business objects can delegate their object oriented method calls to appropriate SQL queries when necessary. Therefore the work to do is to copy all SQL statements existent in the migration package to the QueryList class. Following for each SQL statement, a method will be generated which executes the SQL query on the database. In order to provide function calls from everywhere of the application, the QueryList class will be implemented as a Singleton [Singleton]. Since there may be a large number of SQL statements existent in the migration package, SQL queries and corresponding methods must be named in a structured way. The naming convention of those method will help to map SQL queries of the legacy application to function calls in the new application. Typically the QueryList class could be like this:

```
//Encapsulation of SQL Queries
//QueryList class as SingleTon
class QueryList{

   private static QueryList instance;
   private HashTable<String,String> querytable;

   private QueryList(){
      this.querytable = new HashTable<String,String>();
       ...
      querytable.put("viewMemberList_asp_sqlQuery1",SQLQUERY1);
      querytable.put("viewMemberList_asp_sqlQuery2",SQLQUERY2);
      querytable.put("viewMemberList_asp_sqlQuery3",SQLQUERY3);
       ...
   }

   public static QueryList getInstance(){
      if(instance == null) instance = new QueryList();

      return instance;
   }

   public List<Member> viewMemberList_asp_sqlQuery1()
   {
     ArrayList<Member> memberList = new ArrayList<Member>();
     String query =
      this.querytable.get("viewMemberList_asp_sqlQuery1");

     ResultSet rs = jdbc_conn.execute(query);
     ...
     Return memberList;
   }
}
```

Fig. 6.7.2.a : Example of QueryList class

Unfortunately this step can only be done manually, as an automatic tool has yet to be developed for this task. Therefore SQL files must one by one extracted from the ASP pages and copied to the QueryList class.

6.7.3 Mapping of Model code

The procedure how to migrate legacy Model codes to corresponding business objects, begins with the look up for annotated Model elements in the ASP pages. Following the code fragments identified as Model components will be migrated. However this depends on the migration methods and design of the Model components. In case the code fragment contains SQL queries, and SQL queries were already copied into the QueryList class, the code fragment can be mapped by calling the respective function of the QueryList class. In case the code fragment is annotated as part of a business object, such as a sequence of SQL queries to retrieve a certain data entity relative to the business

object, the code fragment will be mapped to the business object, which is implemented as a JavaBean class. The code fragment itself will be mapped as a method of the business object class, which calls the sequence of the SQL queries via the respective method of the QueryList class. On the other side when the model code is identified as a business function, the code fragment will be mapped to a method of the business package class, such as:

```
<% 'evaluate_cost.asp FunctionName
 'Business Functionality: Evaluates the Estimate Costs of a
 'Project at a given Date

 project_id = Request.Form("project_id")
 date = Request.Form("date")
 'mapped as query1
 query1 = "Select project.type FROM project" &_
         "WHERE project.id =" & project_id
 Set rs = ado_conn.execute(query)
 ...get project Type
 If project_type = "typeA" Then
    'Get Resources of the project
    query2 = "..."
    'calculate the estimated cost at given date
    'Write as XML structure
    Response.Write("<Evaluation><Materials><Cost>" & mat_c &_
      "</Cost></Materials></Evaluation>")
 EndIf
...
 'End of Business Function
```

```
//as SingleTon
class BusinessPackage1{

   ... SingleTom implementation ...

   public static Evaluation
     evaluate_cost_asp_FunctionName(Project p, Date atDate){
     String projectType = QueryList.evaluate_cost_asp_query1(p);
     if(projectType.equals("typeA")){
      List<Resources> rL = QueryList.evaluate_cost_asp_query2();
      //calculate the cost
      Evaluation ev = new Evaluation;
      ev.materials.cost = mat_c;
      return ev;
     }
    ...
}
```

Fig. 6.7.3.a : Mapping of Business Functions to Java

In a different case, when COM components are implemented instead of SQL queries, the business functions of the components should be mapped to the business package class,

and database access functions should be mapped to the business objects. There are two options how COM components can be migrated, which are conversion/redevelopment or encapsulation. In the first approach, the source code of the COM components has to be mapped to Java. This can be done manually or automatically. Manual and automatic conversion of COM codes to Java will not be discussed by this paper, however it must be done in that way that business functionalities should be converted to methods of the business package class, and data access objects in COM should be converted to business objects.

On the other hand, when COM components should be encapsulated in Java classes, encapsulation can either be done manually or automatically. In the manual approach the COM class must be wrapped by the Java Native Interface (JNI). By this solution, a Java class must be created, which calls the COM methods via native functions, the procedure is complex as a Java-to-COM bridge has to be created, which calls the native functions of the COM component. By this method the Java class must implement native functions that access stubs of the Java-to-COM bridge, which itself must be implemented in a native programming language [Ullenboom]. Fortunately a more efficient way to encapsulate COM components into a Java class exists. The COM wrapper tool, called “COM4J”, provides such functionalities by default and creates automatically Java-to-COM bridges, which can be used to call COM functions from a Java class (*see 17.14.2.2 COM4J*). The resulting Wrapper classes can then easily be accessed via regular Java classes and therefore should be encapsulated into the business objects and the business package class to make the reuse of COM components transparent.

6.7.4 Mapping of Controller code

The next step in the implementation involves the migration of Controller elements to Java Servlets. The procedure begins with the look up for Controller elements in ASP pages and the separation of Controller components to a dedicated Java Servlet. Therefore the ASP page will be mapped to a Java Servlet and JSP (only if the ASP page contains View components) structure. Controller components can only be converted, since ASP code must be mapped to Java explicitly, however data access within Controller components, such as with SQL queries and COM objects, can be migrated as described in the previous section. When 3rd party COM components can not be encapsulated, their functionalities must be replaced by equivalent Java services or redeveloped in Java. Thereby it must be kept in mind that the application logic and application structure shall not be broken by replacing COM components or redevelopment. Since redevelopment and replacement of components are too specific and dependent on different cases, they will be not discussed in this paper.

The mapping of ASP code to Java Servlet can either be done manually or automatically. In case of a manual mapping of ASP code to Java Servlet, the resulting code will be in a better quality than a code converted via an automatic tool, since certain information can be not processed by converters. As an example, the problem with typeless variables of ASP can be named. Since ASP code is interpreted, the variable types within the code will be processed during run-time. Therefore variable types at compile-time are processed as

“Variant” types in VBScript based ASP. As a result the resulting code, of converters such as J-ASP consists of Java objects of type “Variant”, which is a generated Java class in order to map variables from ASP to Java Servlet/JSP. As an additional reason against ASP to Java converters, the incompatibility of the generated code with the designed business object model can be named. As in the case of J-ASP, the tool does not really converts ASP code to Java, but simulates ASP functionalities via Java Servlet functionalities Java classes (*see 17.14.2.1 NetCoole J-ASP*). Therefore the codes converted by J-ASP is incompatible to the business object model and regular Java Servlet conventions, since variables converted are of type “Variant” and Java Servlet conventions are undermined by an ASP style of Java Servlet structure.

In comparison to conversion and mapping tools, a manual conversion of ASP elements to Java Servlet is more flexible. In such a case the annotated code will be mapped in such a way, that the corresponding Java Servlet maintains the original control flow of the application and is responsible for the validation of input parameters and the redirection of return values to a JSP View component. By such a structure the code will be converted instruction by instruction, by mapping SQL queries and COM objects to corresponding business objects and business package classes and built-in ASP objects such as “Server”, “Request”, “Response” etc to equivalent built-in Java Servlet objects. The following figure represents the table for the mapping between built-in objects of ASP and Java Servlet.

ASP Object	Java Servlet/JSP Object	Functionality	Scope
ASP Request ASP Respond	HttpServletRequest HttpServletRespond	• Object representation the HTTP request/respond • Contains input parameters (GET/POST)	HTTP request/respond
ASP Session	HttpSession	• Represents a stateful communication between client and server • Stores data appended through out the communication between client and server	A Complete Client/Server conversation, until time-out
ASP Application	ServletContext	• Represents the state of the whole application • Stores data that is used by the whole application	Globally for the whole web application
ASP ADO	Must be implemented specifically	• Database connector to access database management systems	Dynamic
ASP Server	Functionalities must be implemented specifically ServletContext	• Functionalities vary from COM object creation to explicit execution of ASP page	Server lifecycle
ASP FileSystem	Must be implemented specifically	• Offers functions on the file system	Dynamic
ASP Drive, Folder, File	Must be implemented specifically	• Offers functions on the file system	Dynamic
ASP TextStream	Must be implemented specifically	• Offers functions to read text files	Dynamic
ASP Dictionary	Provided by Java Dictionary class	• Associative array	Dynamic

Fig. 6.7.4.a : Mapping of built-in ASP objects to Java

The last step of mapping Controller components to Java, is the redirection of data to View components. Depending on the programming style of the ASP application, the migrated Java Servlet must either map the existing data redirection to View components or must create a new redirection to View components. The latter case occurs when Controller components and View components were previously mixed in the same ASP page (hence the redirection of data to View components was not necessary), therefore data objects must be redirected to View components via the “Session.setAttribute()” or “Response.setAttribute()” methods.

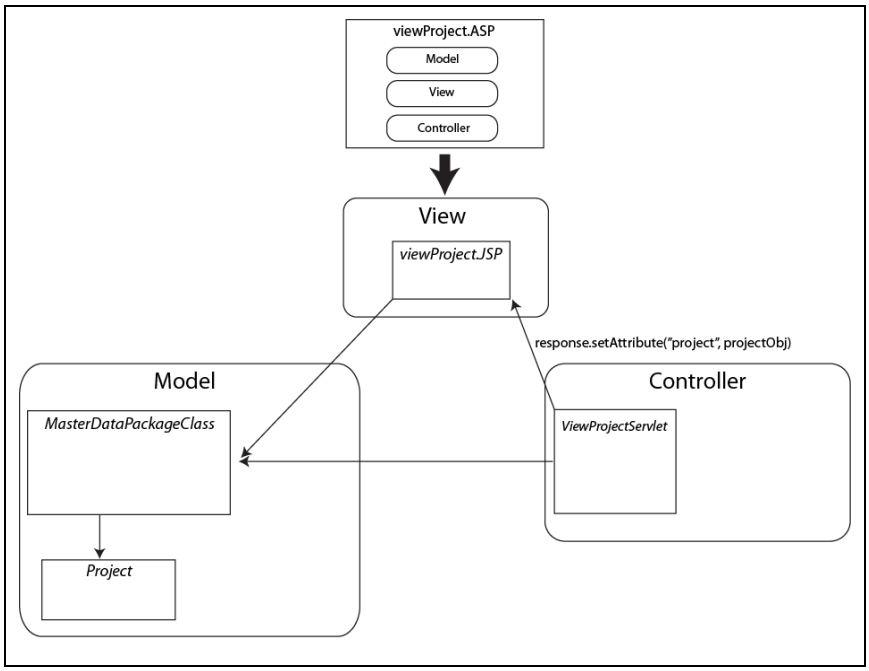

Fig. 6.7.4.b : Redirection of data to Views

6.7.5 Mapping of View code

The mapping of View components in an ASP page to JSP is the last step of the procedure to map legacy components to Java. As like with the other MVC components the mapping of the View component is dependent on the programming style of the legacy application. If the View component is mixed with Control components, the mapping procedure must map separate the View components to a JSP page. The mapping procedure begins with the look up of for annotated View components in the ASP application. If the View component is mixed with other components in one ASP page, a new JSP View must be created. Regarding the mapping of ASP codes there is again the alternative between manual mapping and automatic mapping. However, as already discussed automatic mapping may not have many advantages that outweight the flexibility of manual mapping, but this is dependent on the quality of the converter used. In comparison to the mapping process of ASP code to Java Servlet, the mapping of ASP to JSP does not differentiate much from it, therefore the mapping will be conducted line by line in the mapping process.

In the case View components are implemented via XSL Transformations (XSLT), the View components can just be reused as they are, as the View components remain independent from the Controller components, which are responsible for the redirection of the data in the correct XML structure. Therefore Controller components must just be mapped in that way that they return a correct structure of XML.

6.7.6 Implementation of Web Services

Following the mapping of MVC components to the target business package, the implementation of new service interfaces can be started. In this paper the implementation of service interfaces will be done by Web Services and via JAX-WS for reasons of simplicity. However in practice the choice is dependent on the technical requirements of the project. As in case of JAX-WS the implementation of Web Services is simple. Depending on the design and requirements, business functions can be exposed as Web Services by annotating the containing class, which is the business package class, with the "@WebService" annotation and the respective method with the "@WebMethod" annotation. By these 2 annotations the webserver will recognize the business package class as a Web Service with accessible web methods, during the deployment of the application. As an example an implementation of a Web Service containing a COM function call will be presented.

```
//Java – JAX-WS Web Service

@WebService(name="COMWebSerivceTest")
@SOAPBinding(style=SOAPBinding.Style.RPC)
public class BusinessPackage1{

   @WebMethod
   public String business_function1(int i)
   {

    COMClass com_obj = COMWrapper.create("COMClass");

    return com_obj.business_function1(i)
   }

}
```

Fig. 6.7.6.a : JAX-WS Web Service

6.7.7 Business Package Dependencies

Due to the dependencies between each other residing in business packages, one can not migrate business packages completely. These dependencies can occur in form of references to COM components or redirection of links. In the latter case, a Control component can reference to an ASP component which is not yet migrated. In such a case the referenced ASP page can be included to the migration cycle and be migrated along the business package, but this would lead to further references to other components. Therefore it is the best to invalidate the reference until the referenced component is migrated or to create a dummy stub which simulates the referenced component in a simple way (which means simulating the potential data that are passed through session objects and etc.). As far as COM components are concerned, wrapper classes can just be called as long as business objects or functions (of the to-be-migrated business package)

are still not implemented, but should be updated as the business objects and functions are implemented in the future. In case COM components are subject to be converted or re-developed, the alternative would be to create simple Java dummies which simulate the behavior of the functions. In the end all temporary stubs have to be replaced and integrated with the final implementations.

6.8 Software Testing

In this stage of the migration process the migrated package will be tested. Since the migration concept is based on an incremental approach, specifically on the vertical migration strategy, packages migrated from the legacy application can be run in parallel to the original system as the package is functionally separated. Therefore the migrated functionalities can be tested whether the interactions with other migrated business functionalities are working seamlessly. At every cycle all business packages have to be tested together in cooperation with each other, which means that references that have been updated in the implementation stage have to be error-free. At the end of this stage, depending on the results of the tests, the process returns to the implementation stage if any bugs were found or it progresses to the last step of the migration cycle.

6.9 Software Integration

In the software integration stage, the Web services defined in the migrated package will be registered to the enterprise service repository, to be retrievable for other applications in the company.

In order to register the Web services to the enterprise service repository, Web Services need to be published to a global directory service where they can be searched and found. This is often realized by an UDDI server, which is deployed in the company. Generally an UDDI server is a Web application which implements the UDDI specifications and provides an UDDI API for publishing and searching of Web Services. The most notable interfaces are called "*publish*" and "*inquiry*" in the UDDI Web application. With these two interfaces it is possible publish/inquire Web Services, either programmatically or via a WEB interface such as "*http://<servername>:<port>/publish*" or "*http://<servername>:<port>/inquiry*" (As in case of the open source Java UDDI server [JUDDI]). In order to publish a Web service interface in UDDI, the Web service interface must be mapped to a hierarchical structure which is the UDDI model specification. This model defines business publishers, services and the technical specifications of the service interface. For further reading a Web tutorial of [IBM UDDI] can be read. In the following the components of that model will be presented.

Data Structure	Description
businessEntity	- Specifies the Web Service provider - Contains information such as *company name*, *contact detail* and etc.
businessService	- Represents a group of min. one Web Service - Can contain descriptions that describe the set of Web Services - Contains meta data about the service such as classification names and descriptions
bindingTemplate	- Represents the Web Service - Contains technical information such as access points to invoke the Web Service
tModel	- Represents the technical specification of the Web Service - Contains the Reference to the WSDL file or the meta data of the Web Service specification

Fig. 6.9.a : UDDI Model for Web Services

When publishing a Web Service to a UDDI registry (programmatically or via the Web interface), the UDDI server maps the service interface defined by the WSDL file to the UDDI model, which includes the descriptions of the business provider, service contents, and service names.

The migration process can end with the software integration stage, or can be reiterated from the object refinement stage to migrate further business packages or to improve existing packages.

7 Use Case: Migration of TICOS

As a “best practice” migration project, the migration of TICOS was conducted at Robert Bosch Japan. In this project the TICOS application was migrated to Java according to the primary requirements (section 1.4) and secondary requirements (section 2.2), which were defined by the company. In this chapter the experiences resulting from this migration project will be presented.

Fig. 7.a : The legacy TICOS application

7.1 Migration Process

Software Evaluation

In the software evaluation stage, the TICOS application’s software state was analyzed. Beginning with basic metrics the application was checked for its complexity and structure. Thereby the application’s size, consisting of about 300 ASP files with about 70 lines of ASP codes per page, and the application’s structure , consisting of MVC layers which are structured in dedicated ASP pages named after a certain convention (*see Fig. 7.1.a : TICOS application structure*) was captured. In the next step an overview over how MVC components are implemented was taken. Thereby it turned out that View components are separated in XSL files and Controller and Model components are mixed together in ASP pages and implemented exclusively via SQL queries and ASP codes. The structure of TICOS can be therefore described as mixed in an averaged degree. However the file structure is structured quite well (files are named by their functionality and action

such as “all_project.asp” and “all_project_check.asp”) which helps to understand the code better.

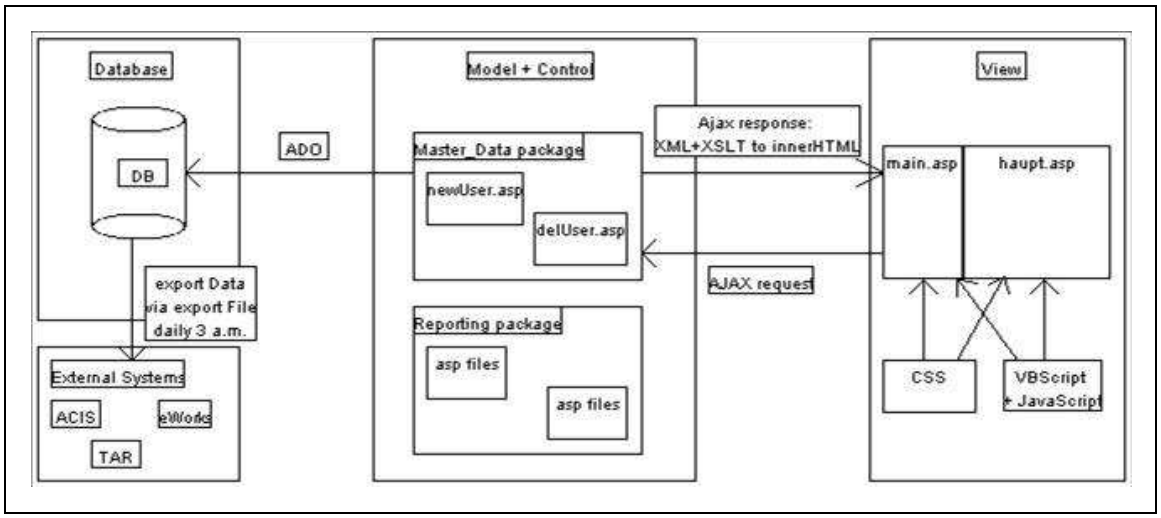

Fig. 7.1.a : TICOS application structure

After the brief analysis on the size and structure of TICOS was conducted, the comprehension on functionalities was attempted to be established. Therefore documentations of TICOS were studied, however it turned out that documentations were lacking in details and extensiveness. As for example documentations of source-code, package structures and data model were missing. Therefore manual reverse-engineering approaches were done, in which the ASP files were examined for their linking structures and functionalities. From the gathered information simple pageflow graphs and use-case diagrams were developed.

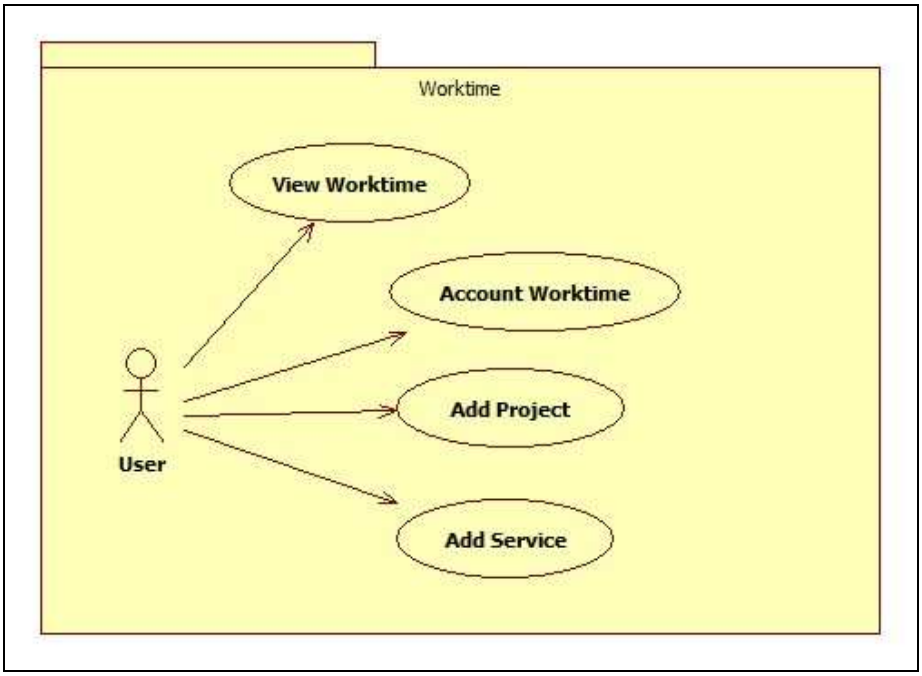

Fig. 7.1.b : TICOS Worktime management use-case diagram

In order to understand the data model of TICOS, the database model of TICOS was analyzed. Thereby Entity-Relationship diagrams were developed.

Problems and Experiences

Since the migration is conducted manually, an adequate understanding over the legacy application must be developed. This is however heavily dependent on the documentations available and the complexity of the application. Through experience it can be said that in companies, documentations for *Custom Business Web Applications* can rarely be found, thus it takes a significant amount of time to conduct the software evaluation stage.

In regard to analysis and comprehension of TICOS, the manual reverse-engineering approach taken is much more time-consuming (it took 3 person-weeks) as compared to reverse-engineering done in automatic migration approaches. However the result of manual reverse-engineering is that codes can be better understood and overall application comprehension is better. The result of the gained comprehension helps to decide which components and application parts can be reused, converted or redeveloped in consensus with project requirements. Furthermore a better understanding of the program leads to a better quality when it comes to the transformation of legacy code to new code.

In the special case of TICOS the problems of reverse-engineering occurred in the comprehension and generation of pageflow models. Since in ASP it is allowed to embed nested ASP documents it was difficult to understand the dataflow (of session objects) in the application. On the other side reverse-engineering of database models and general functionalities were rather simple.

Object Modeling

In the object modeling stage, it was attempted to create an object oriented business object model out of the procedure oriented structures of TICOS. Therefore the database model of TICOS and the application components were analyzed. Since TICOS is completely implemented in a procedural manner as each ASP page implements a business functionality by itself, the question arises if certain application functions should be objectified. However this was considered not necessary at the beginning stage of the migration, since enhancements on the object structures could still be done after the whole legacy application is migrated to the new system. On the other side business entities have to be objectified in order to expose them to Web Services. Therefore the database model of TICOS was reverse-engineered and database tables were objectified. The issue is that database entities can be related to each other represented by different Entity-Relationship Normal Forms, such as in the case of a N:M relation, between 2 tables “Person” and “Project”, an additional table “Person_to_Project” can be generated to extend the relationship of A and B by mapping the primary keys of A and B to each other. In such a Normal Form it is possible to extend the relationship of “Person” and “Project” with an additional “Cost Accounting” entity. However, in regard to the actual implementation, references to the supporting table “Person_to_Project” is only used within SQL

subqueries, in a manner that entities of “Person_to_Project” are never returned. Therefore it is the goal to identify such data entities in SQL queries, so that only necessary data entities are objectified.

In parallel to the objectification of business entities, business packages were identified and created. In case of TICOS the business packages were already well defined so that the original packages could be taken over to the new system. Therefore TICOS namely consists of the following packages: *Master Data, Project Accounting, Project Reporting, Charging and Business Planning, Worktime Management and System.*

Problems and Experiences

In the case of TICOS it was rather simple to identify and objectify business objects for Model components, which is due to the decision to reuse SQL queries and reuse SQL queries in a single “QueryList” class. Thereby queries can be invoked via a dedicated method of the “QueryList” class, which returns the result of the queries in an objectified form. However this was not always possible in case for Controller components, since SQL queries there can return combined data entries such as described in *Fig. 6.6.3.c : SQL queries in Controller component.* In summary this stage took less than a 1 person-week to be completed.

Post-Documentation

In the post-documentation stage of the TICOS migration the analysis of functionalities and structures were put down on paper. Among typical documents as specification document, use-case or class diagrams, pageflow and dataflow graphs were created to document the relationships between ASP pages, which are important in the software implementation stage of the migration process. At source-code level annotations in form of comments were made to identify MVC components.

Problems and Experiences

Since TICOS consists of about 300 ASP files, the time spent on this stage was significant. Thereby for most of the pages MVC components were identified and annotated. While this is a repetitive task, it is quite problematic to identify MVC components in the code, even though it is done manually. The problem lies in the distinction of Model and Controller code, which often times can be mixed up. As an example, it is very difficult to distinguish validation code that manages the control flow of a function, from business logic code in which validation is done. Therefore this task is error-prone, especially when a large number of ASP files are given. Due to the amount of files this task is not only error-prone but also very time-consuming. In the case of TICOS it took about 4 person-weeks to conduct the whole post-documentation of the system.

Object Refinement

After the analysis and the documentation stages, TICOS was migrated in incremental steps. Beginning with the "Master Data" package, in which master data such as personal, project or customer data can be managed, the package was designed fully for the first time. At this stage, the business object classes such as "Person", "Project" and etc. were refined with attributes and methods that represent the relationships of the business objects to each other. Furthermore the business package class "Master Data" was designed, in this class Model components that were identified as business functionalities were designed. Thereby a naming convention relating to the ASP filenames were used, in order to identify the functions from which ASP page they come from.

After the new Model consisting of business package class and the business objects are created, the communication with Controller and Views have been designed. Thereby Servlets were created for Controller components of the package and JSP pages were created for Views. An important aspect in this task is how to design the communication between JavaBean (Model), Servlet (Controller) and JSP (Views), although they originated from only one ASP file. Therefore new codes have to be generated in order to pass data from Servlet to JSP. As a result the communication and data flow between Servlets and JSP had to be designed.

After the "Master Data" package was migrated, the remaining business packages were migrated step by step. In summary the time required to migrate a package was about 2 to 3 person-weeks.

Problems and Experiences

The problems in the Object Refinement stage lies in the selection of a fitting business package. In the case of TICOS the first business package chosen was the "Master Data" package, which consists of almost every important business entity of the application. Therefore it was simpler for the later business packages to be migrated, since almost every reference to the master data of the application was already migrated. By experience there can be said, that for the start of the migration, it is a good decision to migrate packages, which consist of many master data so that future business packages can be migrated easier.

A major problem in the decision process, is whether functional enhancements should be made on-the-fly or not. As the migration project progress, one may be attempted to not just migrate the old components to the new system, but also implement new functionalities or enhance existing functionalities in one migration cycle. As an example from TICOS, the SSO functionality for the portal integration was implemented along with the migration of the Login mechanism. Thereby enhancements in the data model of TICOS and business logic had to be changed a little. On the positive side is that TICOS can be just modernized and enhanced in one single migration cycle. On the other side, however this has caused changes in the migration of all other business packages, since business logic was changed. Therefore documentations done in the previous stages/cycles

may be become obsolete. Thus enhancements or modernizations of existing components may be feasible, but should be really questioned if they are not contra-productive in regard to further migrations. As a result it should be considered whether additional functionalities should be better implemented when the whole legacy application was migrated completely.

Software Implementation

Following the design phase, the packages were migrated in an incremental manner. Thereby SQL queries of each ASP file were copied to a central static Data Access Object ("QueryList" class) and data access calls were mapped to methods of that class. The process of the implementation began with the implementation of the "QueryList" class and the BusinessPackage class such as the "MasterDataPackage" class, followed by the implementation of business object classes if not already implemented. Afterwards the pageflows of TICOS were mapped to Java structures, such as Java classes, Servlets and XSL files (since the View components could be reused). Following the logic of each ASP file were mapped manually to Servlets, thereby calls to Model components were replaced by object oriented calls from business objects, which delegated the data access calls to the "QueryList" class. In case the ASP code contained business functionalities and business logic, appropriate business functions were implemented in the business package class as already designed in the object refinement stage. As the last step to map ASP pages to JSP files, data structures were mapped to XML structures and redirected to the XSL file.

According to the requirements of the company the application has to fit the corporate design, which was provided by the company, in a form of a JavaScript library and CSS files. Therefore View components just as the XSL file and the JavaScript files had to be adjusted in order to fit the corporate design and the required structure of the corporate design libraries. As the last step of the mapping, the AJAX functions in the JavaScript files had to be adjusted to call the newly defined Servlets instead of former ASP pages.

In order to serve the requirements of Single Sign On a new log in mechanism based on Browser Cookies were implemented. This mechanism was provided by the SAP NetWeaver Portal system via the two dll libraries: "*SapSecuLib.dll*" and "*SAPSSOEXT.dll*" and a Java bridge to access the functions. Via these two libraries the application had decrypted the encoded cookie which was sent by the SAP NetWeaver Portal once the user has logged in. From the information retrieved the users Portal ID was mapped to the local TICOS User ID, and authorized according to the roles defined to the user.

Problems and Experiences

In the software implementation stage problems occurred mostly during the code-to-code transformation process. During such a process program bugs may occur frequently. In the case of TICOS difficulties happened when codes of nesting ASP pages have to be implemented. Although codes of nesting ASP pages may have been already transformed

and could be reused (by calling the associated business package method), it is a difficulty to keep the overview over the data flow of nested ASP pages, especially when the ASP pages were not documented properly.

Testing

Following the implementation phase, the implemented business package was deployed in a test environment. Thereby the functionalities could be tested against the data model an be run in parallel to the legacy application. However, although legacy components and migrated components run in parallel together they could not be integrated with each other easily, since simple HTTP links couldn't pass data from Java context to ASP context. Therefore functionalities could only be tested against the shared database and in combination with other migrated components. In the case software bugs were found, the migration process switched back to the implementation phase, where bugs could be removed. This procedure was repeated until all known bugs were removed and the business package was tested successfully.

Software Integration

After the testing phase validated the correctness of the business package and its business functionalities, Web Services were created on top of the business functionalities, in order to expose them in the Service Oriented Architecture. Additionally data services that were used by other systems were modernized and published as Web Services. In the case of TICOS, data were exported as textfiles through a service which extracted the data from the TICOS database periodically. Since this kind of data transfer violates the goal of a homogenous environment, an appropriate Web Service was created which retrieves those data from the database. The Web Services relied on the objectification of the returned data, therefore the same business object classes as from the migration process were returned. However the created Web Services could not be tested in practice since the cooperating system was not migrated yet.

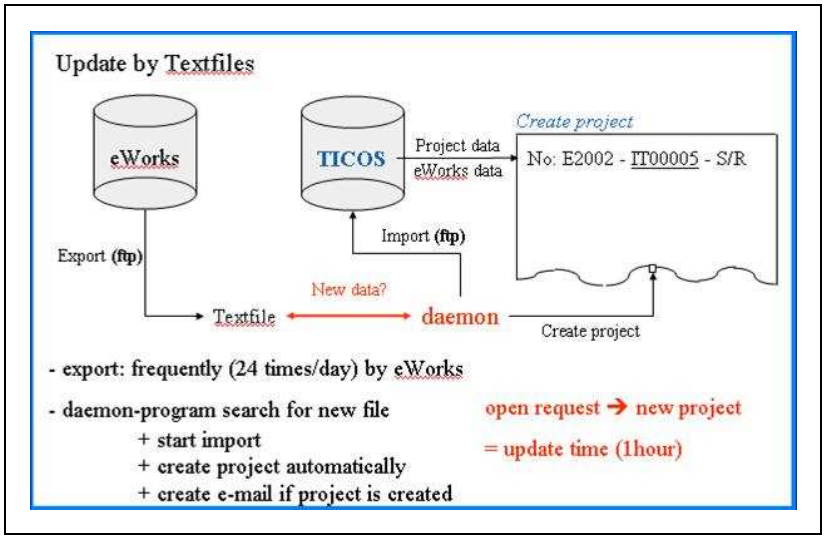

Fig. 7.1.c : TICOS data export via files

7.2 Problems and Advantages

Problems

The migration concept provides a systematic way to migrate legacy ASP Web applications to Java, however due to the manual conduction, the concept inherits the problem when it comes to the size of the application. As experienced in the migration process of TICOS, it was a problem to analyze and document components of the legacy application, when hundreds of ASP files are involved. As a consequence the mapping process at the implementation phase could not be conducted as intuitive as it could have been, because application components were not analyzed and documented well enough. Beside this the cost of time can become a significant aspect. Additionally the way some components, such as SQL statements, were migrated was still inefficient, since each statement had to be filtered and mapped to Java manually. Depending on the convention used when it comes to the mapping of SQL queries to object functions, it could have been a hassle for the conversion of ASP code to Java, when SQL queries had to be looked up for their mapped method names. As a whole the process of extracting SQL queries and mapping of MVC components was extensive although the conduction was systematic and repetitive.

Another problem which was inherent by the choice of the migration method was the introduction of secondary requirements into the migration project. As experienced in the migration of TICOS, secondary requirements that involve an extension of existing functionalities, such as the introduction of Single Sign On to TICOS, can lead to complexities when conducted along the migration process, because of the interference with the design of the legacy system. This is especially the case when changes to the data model are necessary. Depending on the scale of the changes, new functionalities can have impact just on the business object level, for example when just a new table has to be introduced in the data model, or have to be adapted more extensively, for example when the structures and relationships were altered by the extension. In such a case, the adaptation involves the SQL queries that were previously encapsulated into DAO objects. Therefore it should be considered extensively if it is not better to implement secondary requirements at the end, after the legacy application is migrated 1:1 to the new environment. In this case the migration concept offers the opportunity to implement secondary requirements in additional incremental steps.

Advantages

One of the most important advantages that come with this migration concept is the systematic approach, with which ASP applications are analyzed and migrated. Although there can be numerous ways how to transform a legacy ASP application into a Java application, it is necessary to have a structured schema how to migrate from one technology to another. This became apparent, when migrating more complex and bigger ASP applications to Java, as experienced with TICOS, where application components could be migrated to Java in a straight forward fashion.

The other benefit that comes with the migration process is the transformation from legacy structures to a modern MVC design in Java, through the migration methods applied in the migration concept. Therefore applications are provided with more extensibility and flexibility. Additionally the migrated code is of better quality since it is readable for humans. This is the result of the advantages of a manual migration approach when compared to automatic or semi-automatic migration approaches, where generated code is most of the time not readable and unstructured.

Additionally through the incremental migration process (vertical migration), complex applications could be migrated step by step and be tested incrementally. Furthermore, application packages that are well tested can be operated in parallel to the legacy application in order to ease the familiarization process for users. In the following the TICOS system is shown in the new portal environment of Robert Bosch Corporation.

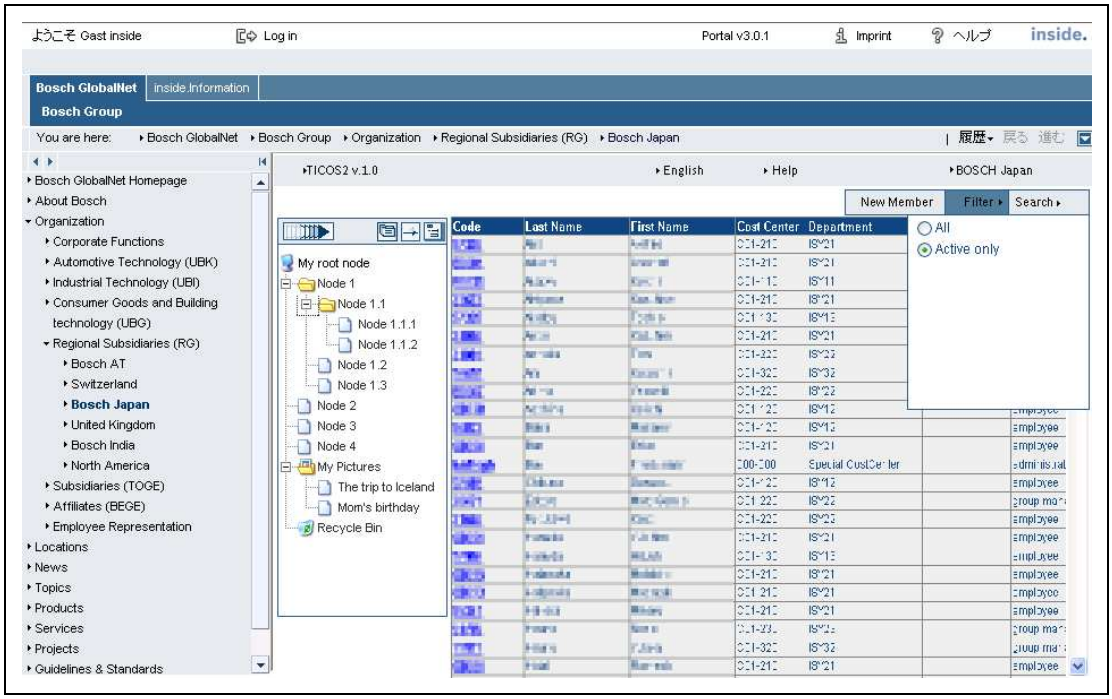

Fig. 7.2.a : Deployment of a migrated business package on the SAP Portal

8 Outlook

The migration concept provides a guideline, which allows companies to conduct their migration projects in a systematic way. As described at the beginning of this paper, the class of Custom Business Web Applications is existent in many local business departments and their applications are managed locally. Therefore it is desirable from the perspective of a company to provide local branches with a migration concept which allow their employees to migrate legacy applications after a given schema. For that reason this migration concept can be seen as an example, how migration tasks can be approached by a company from an IT-government perspective and be designed and adjusted from case to case specifically in order to propagate migration instructions to local departments.

As a basis for ASP-to-Java migration, this migration concept provides a manual migration process for companies, which want to migrate small-sized to mid-sized ASP applications. As a manual migration approach, the migration process can be extensive for big-sized applications and be too repetitive and inefficient in some tasks, namely documentation and code transformation. Although the manual migration brings advantages such as a more flexible ASP-to-Java code transformation and mapping of MVC components in general, it could be enhanced or supported by a semi-automatic migration process. In case of a semi-automatic migration process, it was attempted to incorporate migration tools such as the ASP-to-JSP tool "J-ASP" in the migration process, however this failed due to the design of that tool. Nevertheless it is possible to design a tool which migrates ASP applications based on the analysis and documentation done in the early migration stages (Software Evaluation, Object Modeling, Post-Documentation), via Metadata tags in the ASP application. In such a tool, the migration team would analyze the legacy application for its functionalities and relevant business data, and annotate each MVC component of the ASP application, afterwards the migration tool would automatically transform the annotated ASP files with the inherent information to a Java-based Web application, which is structured after the MVC pattern. With such a tool the extensive task of mapping ASP codes and extracting components could be automated, which would be beneficial for any company with a large number of legacy ASP applications.

In order to design such a tool, one can refer to the work of [Ping et al], which described a migration tool that analyses the structure of IBM .Net Data applications and migrates such applications to Java-based MVC Web applications. In the named work, the authors proposed a migration tool which extracts SQL statements and View components from the original application and migrates them to a Java Web application based on JavaBeans and JSP pages. Thereby the complete code had to be parsed and an Abstract Syntax Tree (AST) had to be constructed, in order to extract SQL statements from the code and to transform the original code to Java/JSP constructs. In case of SQL statements, dynamic variables had to be stored into a property file which allows the tool to link the dynamic variables with input parameters coming from different JSP pages. In relation to ASP applications, the same mapping would be necessary, since mixed components (MVC) have to be separated from each other, and be linked with each other properly afterwards. The problem that was not described by [Ping et al], was how to distinguish MVC

components automatically in the original code, in order to separate MVC components on-the-fly along the migration process. Therefore the proposed methods of [Ping et al] could be used in combination with a manual analysis and annotation of MVC components in order to provide a semi-automatic migration tool, which can be used to migrated ASP Web applications to Java-based MVC Web applications.

As a bottom line it can be said that since the presented migration concept is based on systematic analyses and design, it can be enhanced with semi-automatic migration mechanisms, which would be beneficial for companies to migrate large numbers of ASP-based Web applications more quickly and more efficiently.

References

Bibliography

[Brodie/Stonebreaker]

- Brodie M. L., Stonebreaker M.; *DARWIN: On the Incremental Migration of Legacy Information Systems;* 1993; p. 5

[Cimitile et al.]

- Cimitile A., Carlini U., Lucia A.; *Incremental Migration Strategies: Data Flow Analysis for Wrapping;* 1998

[Gimnich/Winter]

- Gimnich R., Winter A.; *Workflows der Software-Migration;*

[Jeenicke]

- Jeenicke M.; *Architecture-Centric Software Migration of Web-based Information Systems*

[Ullenboom]

- Ullenboom C.; *Java ist auch eine Insel; 5. Auflage; Kapitel 24; Galileo Computing; 2005*

[Martin/Müller]

- Martin J., Müller H. A.; *Strategies for Migration from C to Java;*

[Nicolescu/Klappert/Krcmar 2007]

- Nicolescu V., Klappert K., Krcmar H.; *SAP NetWeaver Portal*; SAP Press; 2007

[Ping et al.]

- Ping Y., Lu J., Lau T. C., Kontogiannis K., Tong T., Yi. B.; *Migration of Legacy Web Applications to Enterprise Java™ Environments – Net.Data® to JSP™ Transformation, 2003*

[Sneed]

- Sneed H. M.; *Migration prozeduraler Anwendungssysteme in eine objektorientierte Architektur*

[Sneed 1999]

- Sneed H. M.; *Objekorientierte Softwaremigration;* Addison-Wesley; 1999

[Vlachakis/Kirchhof/Gurzki 2005]

- Vlachakis J., Kirchhof A., Gurzki T.; *Marktübersicht Portalsoftware 2005*, Fraunhofer IRB Verlag; 2005

Web References

[COM]

- COM: Component Object Model Technologies; http://www.microsoft.com/com/default.mspx

[COM4J]

- COM4J Project Website; https://com4j.dev.java.net/

[IBM UDDI]

- Understanding WSDL in a UDDI registry; http://www.ibm.com/developerworks/webservices/library/ws-wsdl/

[IBM SOA]

- Develop a migration strategy from a legacy enterprise IT infrastructure to an SOA-based enterprise architecture; http://www.ibm.com/developerworks/webservices/library/ws-migrate2soa/

[J-ASP]

- ASP to JSP/Servlet Migration Tool from Netcoole; http://www.netcoole.com/jasp.htm

[JSF]

- JavaServer Faces Technology; http://java.sun.com/javaee/javaserverfaces/

[JUDDI]

- Open Source Java UDDI Server; http://ws.apache.org/juddi/

[MSDN Migration]

- Microsoft .Net/Com Migration; Migration Strategies; http://msdn.microsoft.com/en-us/library/ms978506.aspx#cominterop_topic4

[Seam]

- Seam Framework Website; http://seamframework.org/

[Singleton]

- Singleton Design Pattern; http://www.dofactory.com/patterns/PatternSingleton.aspx

[SUN JEE 5 Tutorial]

- SUN JEE 5 Tutorial;
  http://java.sun.com/javaee/5/docs/tutorial/doc/index.html

[Waterfall model]

- The Waterfall model explained;
  http://www.buzzle.com/editorials/1-5-2005-63768.asp

[Web Service vs. DCOM]

- Using Web Services instead of DCOM;
  http://msdn.microsoft.com/en-us/library/aa302336.aspx