Master Thesis

Migration Methodologies for Web Applications

A Case Study with the IVY to JSF Presentation Frameworks

Author:	Wei Sun
Student Number:	27409
Subject:	Information & Media Technologies
Date:	16.03.2006
Supervisor:	Prof. Dr. Joachim W. Schmidt
	Prof. Dr. Helmut Weberpals
	Mr. Rainer Marrone

Table of Contents

Statement

Hereby I do state that this work has been undertaken by me. All literally or content related quotations from other sources are clearly pointed out and no other sources rather than the ones declared are used.

Wei Sun

Hamburg, March 2006

Acknowledgement

I have received great many supports to accomplish this master thesis.

I would like to thank Prof. Dr. Joachim W. Schmidt, Prof. Dr. Helmut Weberpals, and Mr. Rainer Marrone for giving me the opportunity to work on this thesis project under their supervision and for their constructive suggestions and instructions to help me improve this master thesis.

I would like to thank Dr. Christian Pütter, Dr. Holger Schimanski, Mr. Ronald Brill, Mr. Andre Klaassen, and Mr. Frank Danek, who were always willing to provide me with useful advices, technical supports and timely feedbacks. With them together I had a pleasant and fruitful working experience at Bayer Business Services GmbH to conduct this master thesis.

Beside me there are also my family and dear friends, who are continuously encouraging me to conquer the hardship with optimism and to meet new challenges with perseverance. I would like to thank them, too.

Abstract

Support for business processes in the general domain of science, technology and environmental protection is a high-ranking goal for many companies engaged in the production and distribution industries. For example, the Bayer Business Services GmbH (BBS) provides by its Science & Technology Department (S&T) Bayer-wide software development support with a particular focus on information systems based on web applications. With the specific aim to support the development of chemistry-oriented web applications, BBS has developed the so-called IVY Framework which is mainly based on J2EE and Oracle technology. So far, BBS-S&T has developed more than sixty web applications of various scales using IVY Framework, and they will be kept in operation for years.

Due to the decision made by BBS-S&T to adopt the JSF Framework, all the IVY web applications are planned to be migrated within the next two or three years. The migration from one legacy web application framework to the standard is a fairly new case, and researches and practical experience are required. This paper presents the optimized migration methodologies developed for BBS-S&T with the objectives to make its migration projects executed in a smooth and successful approach. Nevertheless, these migration methodologies can also be applied by other organizations with the similar migration projects of web applications.

List of Figures

List of Tables

List of Abbreviations

JSF	JavaServer Faces
JCP	Java Community Process
QoS	Quality of Service
TCO	Total Cost of Ownership
JSP	JavaServer Pages
COTS	Common Off-The-Shelf
CVS	Concurrent Versions System
RI	Reference Implementation
MVC	Model-View-Controller
MLC	Method Lines of Code
DIT	Depth of Inheritance Tree
WMC	Weighted Methods per Class
LCM	Lack of Cohesion of Methods
Ca	Afferent Coupling
Ce	Efferent Coupling

1 Introduction

Before the release of JavaServer Faces (JSF) by the Java Community Process (JCP)[1], it was a rather complicated and time-consuming task to choose the right framework for Java web development. The reason[2] was that there were too many choices, for instance, Apache Struts, the most well-known one, and other alternatives. All of these frameworks are built upon the J2EE technology[3] to provide robust functionalities for form handling, layer separation, navigation handling, templates, internationalization, etc. However, none of them was regarded as the optimal choice on the market. It was driven by such a demand for a new standard framework of web applications that JSF got its prosperous development and the initial version of its specification was published by Sun and other famous vendors in 2004.

JSF offers several favorable features, such as a solid component framework, support for multiple client devices, the user interface event-oriented development, robust tool supports, and an extensible architecture, to name a few. With JSF growing to be a mature technology, organizations, which may have standardized on other web frameworks, are evaluating their options for future development. They are interested in the key benefits that the migration to JSF and the JSF tool support will bring about. Meanwhile, they are examining the possibilities for leveraging their existing code base and skill sets.

The Science & Technology Department (S&T) of Bayer Business Services GmbH (BBS) is one of those organizations. BBS-S&T provides Bayer-wide supports for business process in the domain of science, technology and environmental protection. One of its focuses is to develop the information systems on the basis of web applications. With the specific aim to support the development of chemistry-oriented web applications, it has developed the so-called IVY Framework mainly with the J2EE and Oracle technology. With JSF being the new standard of Java web development framework, BBS-S&T has determined to migrate all its web applications into JSF in the near future.

The migration projects, if not well organized, are likely destined to performance failures, behind schedule and financial loss. This paper presents optimized migration methodologies developed for BBS-S&T with the objectives to enable its migration projects to be carried out in a smooth and successful approach. Nevertheless, these migration methodologies can also be useful for other organizations with similar migration projects of web applications.

[1] JSR 127: JavaServer Faces
[2] See Kito D. Mann (2005)
[3] See J2EE 1.4 Turorial

1.1 Task Description

The IVY Framework, developed by BBS-S&T to facilitate the development of chemistry-oriented web applications, is composed of four models, namely IVY Foundation, IVY Chemistry, IVY Presentation, and IVY Collaboration. Figure 1-1 outlines its architecture and the relationship among the four models. IVY Foundation is the base library to provide services for database access, exception handling, logging, multilingual support, security, utilities, to name a few. IVY Chemistry is developed to simplify the implementation of web applications with chemical features. IVY Presentation is mainly applied to develop web applications complying with the Bayer Standard Layout Guideline. Detailed information about IVY Presentation is introduced in Chapter 3. IVY Collaboration is a collection of web services, which are utilized to integrate varieties of web applications for the BBS portal. So far BBS-S&T has developed about sixty web applications of various scales using IVY Framework. These web applications will be still kept in operation for years. Since BBS-S&T has decided to adopt the JavaServer Faces Framework, all the IVY web applications are going to be migrated within the next two or three years.

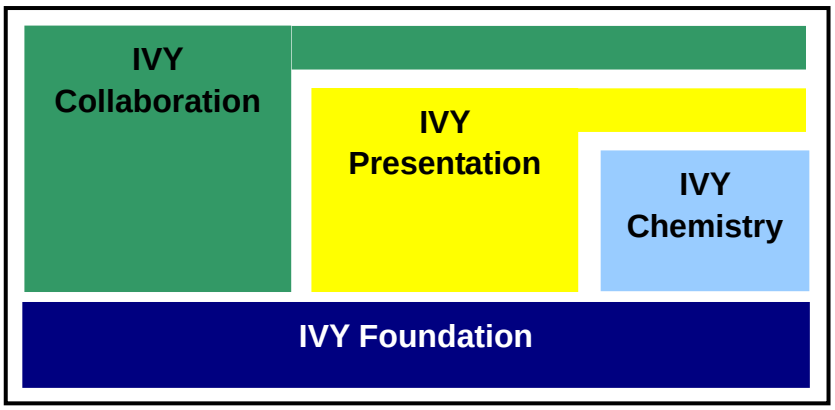


Figure 1-1: Architecture of IVY Framework

Nowadays the technology is continuously replaced or upgraded by new and advanced ones. Under such a circumstance, migration is no longer a brand-new topic in the IT industry. However, the migration of web applications to a different web development framework is still a fairly new case. Researches and practical experience are in highly demand. Therefore, the mission of this master project is to work out suitable migration methodologies for web applications and the migration of web applications from IVY Presentation to JSF for BBS-S&T is taken as a case study.

To be specific, the task assigned to this master project consists of four parts. The first task is to conduct a research on the existing migration technology with the particular aim to learn from the proven migration techniques and to design effective and

actionable migration methodologies for BBS-S&T. This goal can only be achieved with a thorough comprehension of the source IVY Presentation framework and the target JSF framework. So the second task is to make an in-depth comparison of the two frameworks in terms of the architecture and concepts and to identify their commonality and difference. The objective is to formulate mapping rules with regard to how the migration can be performed in practice. The third task is to develop some useful tools, templates and methods, which must conform to the migration rules and enable the migration task to be done effectively and efficiently. Since the theory must be appraised and improved by practice, the last but not the least task is to verify the usefulness and feasibility of the developed migration methodologies with concrete migration projects.

1.2 Vision and Guidelines

A clear vision and working guidelines are helpful to guarantee the quality of the project execution. The vision of this master project is to develop effective methodologies for the migration of web applications. Through the interview with several migration engineers of BBS-S&T, the following guidelines are drawn up to lead the progress of this master project.

First of all, migration engineers are the end-users of the developed migration methodologies, so their expectations and requirements must be given sufficient attention. Besides, it should be pointed out that the migration methodologies are evolving processes, the effectiveness of which needs to be improved continuously through the experience gained from previous migration projects. The aim of the project is to develop the fundamental migration methodologies that are applicable to all migration projects, and not to explore every single detail that may change itself over time.

As to the migration tools, the objective is to accelerate the migration process and to simplify the migration tasks, so they should fulfill end-users' requirements for the simple usage and the effective reduction of tedious workload. Because the tools to be developed are only required for the migration of the existing web applications, the implementation of these tools must be cost-effective.

Last but not the least, the migration guideline is an important outcome of the project. It needs to be organized in a clear and logic structure. The focus is to provide migration engineers with adequate information to apply the migration methodologies.

Table 1-1 lists the main contents for each chapter. Readers can select information according to their interests and needs.

Chapter	Outline
Chapter 2	• Describe the influence of migration on enterprise • Introduce existing migration strategies • Select optimal migration strategies for BBS-S&T • Describe the model-driven migration approach • Illustrate the formulation of migration plan • Propose migration plan for BBS-S&T
Chapter 3	• Introduce the IVY web application framework and portal application framework • Introduce the JavaServer Faces framework • Comparison between IVY and JSF frameworks
Chapter 4	• Describe the mapping rules
Chapter 5	• Introduce the developed migration tools • Illustrate the migration project template • Outline the migration guideline
Chapter 6	• Verify the migration methodologies with practical projects • Demonstrate the performance test results • Exhibit the code-evaluation test results
Chapter 7	• Conclusion • Outlook

Table 1-1: Chapter Outlines of Master Thesis

2 Overview of Migration Technology

Migration, referring to the definition given by Sun Microsystems, Inc. is “the transition of an environment's people, processes, or technologies from one implementation to another”. The value achieved by migration is derived from improved quality of service (QoS) and reduced total cost of ownership (TCO) as a significant characteristic of the new platform, environment, and overall IT infrastructure. In the IT industry the frequently occurring migration contexts cover, for instance, the upgrade of common off-the-shelf (COTS) software to its up-to-date version, the migration of applications from one operating environment to another, the migration of data from one database to another, and the migration of a legacy application from one programming language to another. With the rapid development of web technology, the migration contexts are certainly enriched by the migration of web applications from one legacy framework to the standard. It is this wide scope that leads to the various interpretations of migration.

2.1 Impact of Migration on Enterprise

In order to understand the typical impact that the migration has on an enterprise, it is essential to have the knowledge of the enterprise infrastructure. Figure 2-1 illustrates a model of the infrastructure that an enterprise normally has.

The enterprise infrastructure is divided into three levels, namely the decision level, the execution level and the management level. On the decision level, issues with regard to business strategy, people and process are primarily determined by the executive team of the enterprise. It is these crucial decisions that distinguish the vision and development strategy of each enterprise from the other. These decisions must be in turn executed and supported by the lower levels. Moreover, these decisions set the priorities of how resources are allocated to the involved business functions.

On the execution level the business application locates right under the business process. Generally speaking, changes of business process are most likely to trigger the corresponding changes in the business application, so a rapidly evolving enterprise must cultivate the capacity to implement the application changes in a timely and organized manner. Business applications are supported by application infrastructure including web server, application server, middleware and database technology, which build upon the multilayer architecture. The application infrastructure is operating on the computing and storage platforms, which are composed of a heterogeneous mix of hardware from different vendors. Nowadays the communication through network technology is widely employed, so the network infrastructure is an indispensable component of the enterprise infrastructure. The facilities infrastructure is equally

important, because if it doesn't work, the whole system is unable to operate at all. This execution level is featured with some system properties, namely availability, scalability, measurability and security, which are critical to every enterprise.

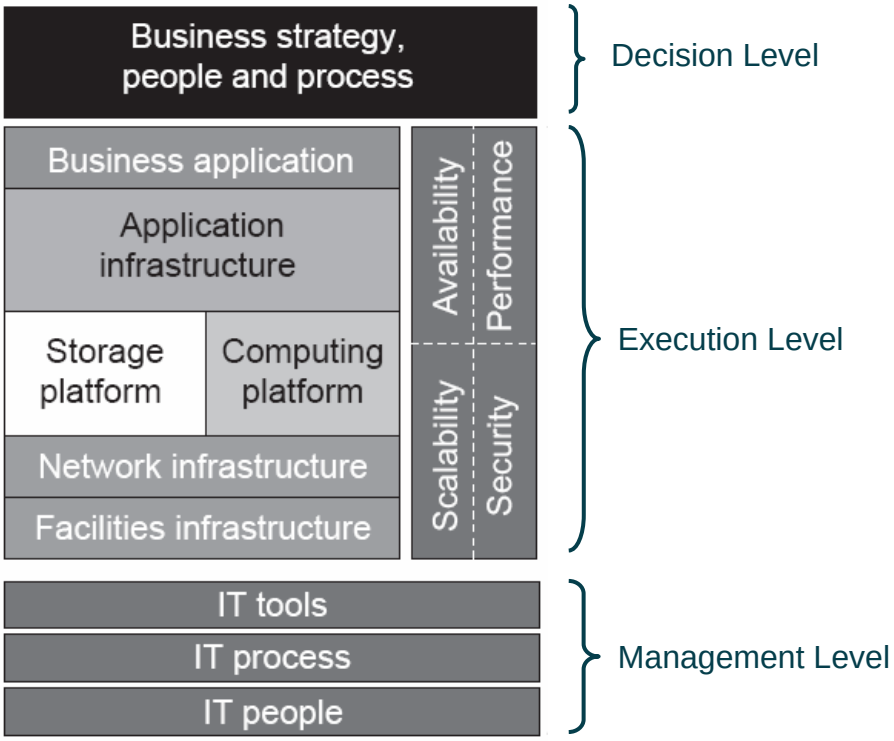


Figure 2-1: Model of Enterprise Infrastructure[4]

The management level consists of the tools, processes and people, which are combined to control, measure and manage the execution level. The tools and processes are applied to support changes, services, deployment and maintenance. The IT people take the full responsibility to develop and maintain these tools and processes. Therefore, to adapt to changes responsively, enterprise should encourage its IT people for innovation and equip them with required knowledge and skills, so that they can confront changes with more willingness and ease.

In conclusion, the migration affects the enterprise as a whole, but to varied extents. As a team work, the migration requires harmonious cooperation among the decision, execution and management levels. Therefore, it is critical for the execution team of BBS-S&T to be aware of the potential impacts, to provide sufficient supports to IT people, and to work out feasible solutions to minimize possible risks.

[4] See: Migration Strategies

2.2 Overview of Migration Strategies

There exist various strategies [5] for migration, namely refronting, replacement, rehosting, rearchitecture, interoperation, and retirement. The introduction to these strategies aims to set the foundation for the selection of the optimal migration strategy for BBS-S&T.

Refronting means to add or improve the graphical user interface, instead of rewriting the entire application. This strategy is suitable for applications, which have good functionalities, but aren't user friendly.

Replacing means to substitute part of a complex and often customer-written legacy application with a COTS application, When the legacy application is decomposed into functional building blocks.

Rehosting refers to moving a complete application from a legacy environment without any change in functionality. It can be realized in several ways, namely recompilation, emulation and technology porting. Recompilation, as its name reveals, needs to recompile the source codes, which can be done in two approaches. The first approach is to develop or acquire a compatibility library that provides identical functionality of the legacy environment. The second is to alter the source codes in order to call the APIs of the new environment. Emulation is to remain the source codes intact and to introduce an additional layer to emulate the instruction set used in the source binary. Technology porting is to supplement the new environment with the capability to execute the code that runs natively in the original environment.

Rearchitecture stands for a tailored approach, which enables the whole application architecture to be migrated to the new environment, probably using new programming paradigms and languages.

Interoperation is to leave the legacy application as it is and surround it with new technology if it is required by an enterprise.

Retirement is utilized, when changes in technology obviate the need for specific functionality of an application or an overall solution. In this case, legacy utilities or applications are retired because they are no long required or applied in the solution.

In order to determine which migration strategy be chosen, both the IT environment and the application itself need to be taken into consideration. First of all, the selected migration strategy of an application must fit into its overall IT environment. As is

[5] See: Migration Strategies

indicated by Figure 2-2, the existing environment can be evaluated from two aspects, namely how the application meets business needs and how technically effective it is.

The X-axis of the diagram indicates the adequacy of meeting business needs, which an be evaluated by the time required to introduce new features, the ease of use, the ability to support the functional requirements of the enterprise and the ability to support the future growth of the enterprise. The Y-axis reflects its IT effectiveness in terms of total cost of ownership (*TCO*), technological stability, functional separation, service level issues and implementation technologies.

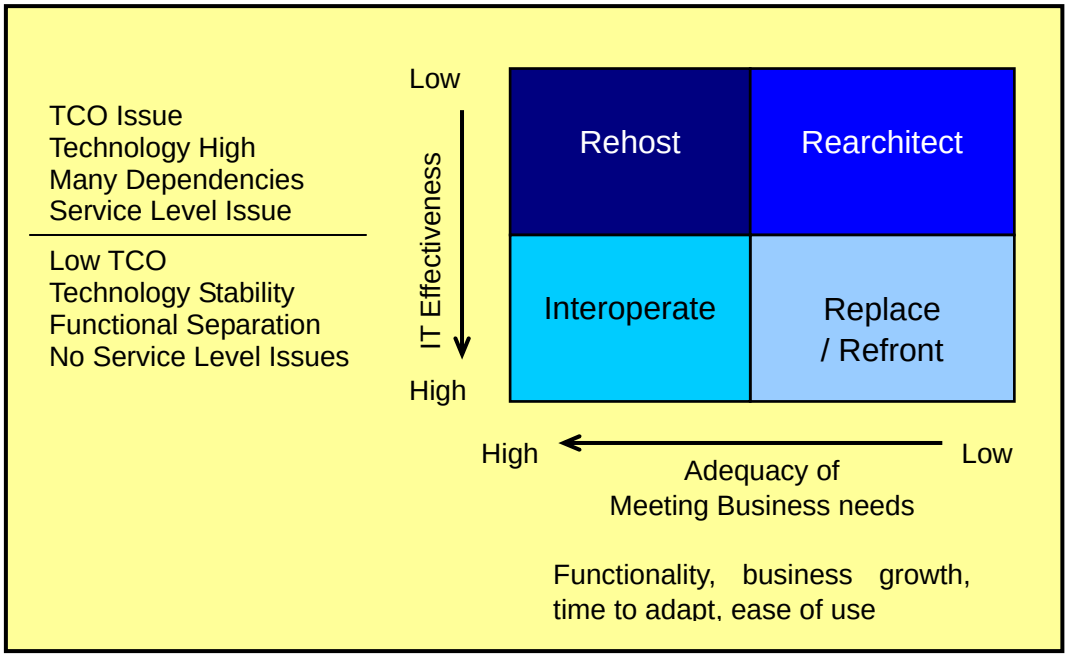


Figure 2-2: Effectiveness and Business Needs[6]

After the comprehensive evaluation of applications according to the two criteria, those falling in the lower-left quadrant are meeting business needs well and are highly technically effective. They should be maintained as they are, and interoperation strategy can be applied to make them compatible with the new environment. Those in the lower-right quadrant have high IT effectiveness, but fail to meet business requirements. They need to be enhanced, instead of being migrated, so refront or replace is the right choice. For those in the upper-left quadrant, which meet functional business requirements, but show lower IT effectiveness, rehost is the right solution. For those in the upper-right quadrant, which are low in IT effectiveness and lack of adequate business functionality, rearchitect is a better choice.

In addition to the overall IT environment, the selection of migration strategy is also influenced by the application architecture. Nowadays most modern applications are

[6] See Migration Strategies

built upon the multi-tier architecture, and its decomposition allows for the different migration strategies to be applied to each tier separately. Table 2-1 on the next page lists migration strategies appropriate for each tier.

Tier	Purpose	Common Approaches
Presentation	Hosts the processing that adapts the display and interaction as appropriate for the accessing client device, be it a desktop computer, a cell phone, a PDA, or any other device.	Refronting, rehosting, interoperating, and replacing
Application or Business Logic	Hosts the logic that embodies the rules of the enterprise, irrespective of access device or resource implementation.	Rehosting, interoperating, and replacing
Integration	Allows for the connection of disparate applications and data sources.	Rehosting, interoperating, and replacing
Resource or Database	Consists of legacy systems, relational databases, data warehouses, or any other back-end or external processing system that accesses and organizes data.	Rehosting and replacing
Persistence	Holds the permanent data for the enterprise. In the past, this was considered part of the Resources tier, but with the growth of intelligent storage, it has become a tier in itself.	Rehosting and replacing

Table 2-1: Migration Strategy Decided by Application Architecture[7]

It is well-known that migration strategies require different amount of effort and bring about varied values as well. Generally speaking, the achieved value is proportional to the effort committed to the migration project.

Interoperation requires the least amount of effort, and provides the least amount of benefit. The architecture and infrastructure remain unchanged and simple connector technology is deployed to support the interaction with new applications or hardware. Since no new functionality is added, this task requires minimal time and expense.

Rearchitecture occupies the other extreme. It supports tailored functionality, modular, tiered design, and a modern implementation language. So it leads to great benefits. But on the other hand, the effort and associated cost can be significant. Moreover, this solution is error-prone, so it requires a rigorous validation and verification effort.

[7] See Migration Strategies

Refronting or replacement enhances the application by adding a presentation layer, which will add new functionality. But given that the application is considered to be somewhat unacceptable, this enhancement adds minimal overall benefit compared with the amount of effort it requires.

Rehosting is the solution that provides the most value for the least effort. It typically involves modifying the source code and building the environment for an application, so that it compiles and runs on the new target system.

2.3 Strategy Selection for BBS-S&T Migration Projects

After the overview of migration and its actable strategies, this sub-chapter presents a detailed analysis and selection of the migration strategy suitable for BBS-S&T.

It is essential to have a good knowledge of the influence that the migration has on BBS-S&T. Since the executive team of BBS-S&T made the decision to migrate the web applications from IVY Presentation to JavaServer Faces, they should attach great importance to the projects and provide migration engineers with sufficient supports. Web applications and their operating environment belong to the business application layer and the application infrastructure layer in the enterprise infrastructure, where most of the changes take place. The rest layers of the execution level can remain to a great extend the same. Besides, the migration will have great impact on the manage level as well. The tools and processes that people are used to utilize may change, and some new tools or processes need to be introduced for the development, deployment and maintenance of web applications with the new framework. Most important is that IT people must have the knowledge to manage the changes and the willingness to learn new techniques. Therefore, good supports to prepare them ready for the migration is of highly necessity. Therefore, for BBS-S&T the migration of web applications will have major influence on the decision level, the business application layer and the application infrastructure layer of the execution layer and the management level. This paper is focused specially on the development of migration methodologies to deal with the changes to the business application layer.

The influence of migration on the business application layer reveals itself on both IVY Framework and web applications, so migration strategies should be selected for the most appropriate usage. Let’s discuss the strategy for the migration of IVY Framework first. From the perspective of IT environment, IVY Framework shows a high standard in IT effectiveness. However, its IVY Presentation model cannot fulfill the business needs, especially in terms of its capacity to support the future growth of the enterprise. JSF has grown to be a standard and mature technology for web application development and it has powerful vendor and industry supports. Therefore,

for the migration of IVY Presentation, replacement is the most suitable solution. To integrate JSF into IVY Framework seamlessly, some extra work needs to be done and this task is in the charge of the JSF research project currently conducted in BBS-S&T. The so-called IVY Faces model has been implemented to provide JSF web applications with convenient access to services of IVY Framework. Besides, services of IVY Faces are utilized in this project, and that will be introduced in Chapter 4.

Now let's focus on the strategy for the migration of web applications, which is the main task of this master project. From the aspect of IT environment, web applications built upon IVY Presentation are able to fulfill the business needs with satisfactory functionalities and ease of use. From technical point of view, it is stable in technology. However, since BBS decides not to apply IVY Presentation for the web application development any more, the total cost of ownership to maintain these web applications will be high and in the future there will be service issues involved. Through the analysis, rehosting proves to be the optimal strategy. Besides, from the perspective of application architecture, web applications fall into the application and business logic category, and rehosting is again considered to be a better choice. As is introduced Chapter 2.2, rehosting can be realized using different approaches. Due to the service issue, it is suggested that BBS-S&T adopt the source code porting approach, which modifies the source codes to call APIs of the JSF Framework directly.

Normally rehosting constrains its scope by adding no new features and functionality. However, many companies want to take the chance to add new features or functionality during the migration, and the migration of IVY web applications will involve improvement as well. Therefore, BBS-S&T is also interested to know the appropriate approach to deal with improvement and migration. The improvement can be classified to three levels, namely great improvement, small improvement and no improvement. If great improvement is required, it indicates that the web application doesn't meet the business requirements. In this case, rehosting isn't a proper migration strategy, instead rearchitecture should be used. The web application can be redesigned by using reverse engineering or redeveloped from scratch, and the general software development process can be applied.

Due to the time constraint, this project pays more attention to the migration of IVY web applications with small or no modification in terms of its presentation and business logic, which are also the cases in BBS-S&T. IVY web applications have been put in operation for years. End-users are used to the outlook and the functionality of the existing web applications. Any change in the application especially on the outlook will result in some cost for training the end-users and in consequence increase the productivity loss to some extend. Due to concern of potential risks, careful consideration should be taken for the decision of changes. However, in some cases

changes are necessary and some new features need to be added, two approaches can be chosen to handle the improvement. One is to implement the improvement after the completion of the migration, while the other is the carry out the improvement during the migration process. The criteria used to select the proper approach for one specific scenario are the complexity of the IVY web application, the knowledge that migration engineers have about the IVY web application, and the code reuseness of the IVY web application.

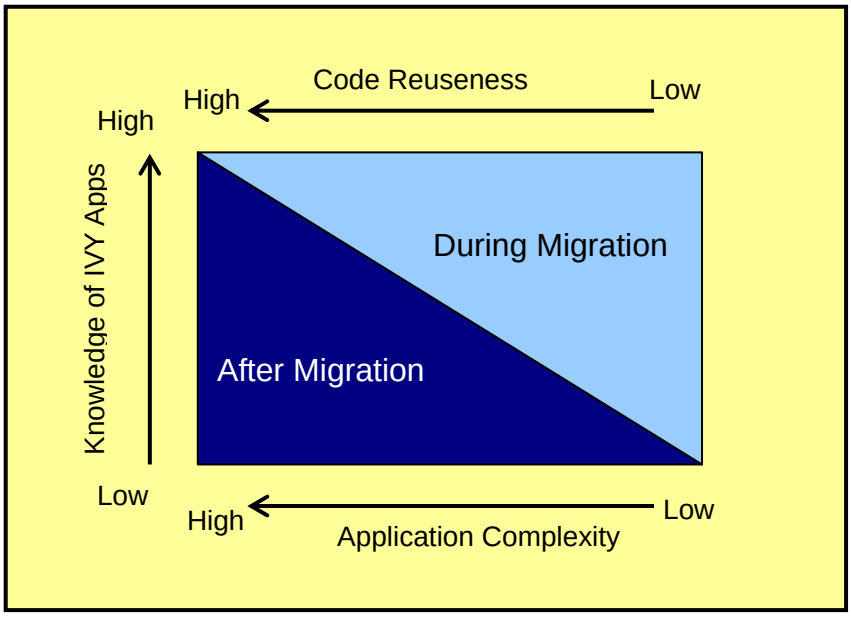


Figure 2-3: Handle Improvement for Migration Project

To determine what and how improvement to be made, migration engineers need to know the architecture and functionality of the IVY web applications very well. When the application complexity is high, it is difficult for them to manage changes when they are lack in the knowledge of the web application. It is recommended to do the migration first with the purpose to acquire the comprehension of the web application. However, if changes are made afterwards, it may cause the loss of migration efforts and time cost. To solve this problem some standards and good practices are introduced in Chapter 4, so that migration can be done as efficiently as possible. Figure 2-3 can be used by migration engineers to decide when the right time for improvement is. When the application complexity is high and the knowledge of the application is low, it is suggested to make the improvement after the migration. When the application complexity is low and the knowledge of the application is sufficient, improvement can be done during the migration. As to the other two cases, improvement can be carried out either after or during the migration. When codes of the IVY web applications, such as codes for test, can be reused, it is better to do the improvement afterwards.

In conclusion, rehosting is selected to be the migration strategy for BBS-S&T to migrate web applications, and the rehosting will be realized with the source code porting approach. Because fewer changes are involved, the migration projects are expected to be completed rather quickly and less expensive.

2.4 Model-Driven Migration Approach

Through the investigation of the existing migration techniques, the model-driven migration approach illustrated in Figure 2-4 is adopted to lead the master project. This model divides the whole migration process into a succession of sub-models, so that the migration task is accomplished in an organized manner.

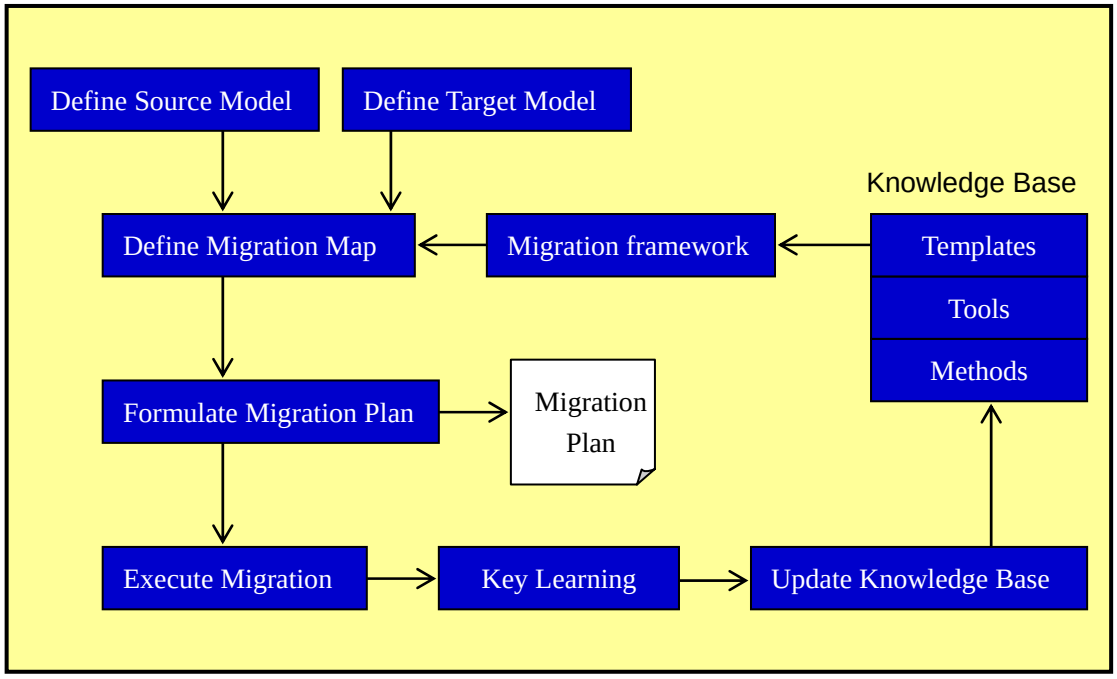


Figure 2-4: Model Driven Migration Approach[8]

The first activity of the model-driven migration approach is to set up the source and target environment models. A comprehensive knowledge of the two environments is highly demanded, and a comparison of their architectures and concepts is carried out in comparable terms, so that the development of the migration map is simplified.

The migration map, which is the second task, defines what elements in the source system need to be migrated to the target system, and how the migration is actually implemented. The migration map is stored in the migration knowledge base, and is enriched by experiences obtained from each migration project.

[8] See Patrick DJ Kulandaisamy(2004)

Through the experience obtained from practical migration projects, it is found out that migration project rather complicated and error-prone due to the numerous details and the lack of documentation of the source model. Only if the migration is well-organized, will it end up with success. In this case, a well-prepared migration plan with detailed list of required tasks is a great help.

The execute migration phase is where individual migration project is conducted using the migration map and being managed by the migration plan. However, the migration process doesn't end here. Migration is a dynamically self-evolving process. The last but also the most valuable activity is to summarize knowledge acquired from the previous projects and placed it into the knowledgebase, so that it is shared by others and the benefit is maximized.

This master project has followed the model-driven migration approach. The study of the source and target systems, i.e. IVY Presentation and JSF is presented in Chapter 3. The mapping rules are described in Chapter 4. The design of the migration plan is introduced immediately. And the tools, templates and methods developed to facilitate the migration process are presented in Chapter 5.

2.5 Formulation of Migration Plan

The formulation of migration plan is by no means a simple task. To ensure the feasibility of the migration plan developed for BBS-S&T, some experience is learned from the research on software migration planning conducted by the Department of Defense, U.S. in 2002. They have worked out a set of criteria[9] that a good software migration plan should meet and the criteria are listed below.

- clearly describe the approach for migrating users and operations from the legacy system to the new system
- contain sufficient detail to verify that the approach is complete, coherent, and consistent, and to validate that the approach is on target
- provide a basis for communication and understanding among stakeholders, managing resources and staff, managing the project, establishing the appropriate relationships with and commitments from stakeholders, establishing a framework under which any related contract efforts can be managed
- provide a means for executive management to monitor the effort
- reduce risk and increase the likelihood of a successful migration effort

[9] see Bergey, O'Brien, Smith(2002)

As a high-level overview, the migration plan normally identifies the principal factors, such as the key tasks, roles and responsibilities, the current environment, the new environment, migration timing and priorities, migration policies and risk factors. However, it needs an additional level of detail to make migration plan actionable and a mechanism to make it a live document that evolves throughout the duration of migration. Therefore, they put forward six critical focus areas for general migration projects and worked out detailed activities related to each focus area. These six focus areas are[10]:

1. migration planning and management
2. deployment and transition assistance
3. database conversion
4. customer relationship management
5. management of the legacy system interface
6. customer training

For each focus area, a responsible person can be designated to develop a mini-plan of actions down to additional levels of detail. The mini-plans need to answer the questions, like what needs to be done, who is going to do it, how it will be done and how to make sure that it is done satisfactorily.

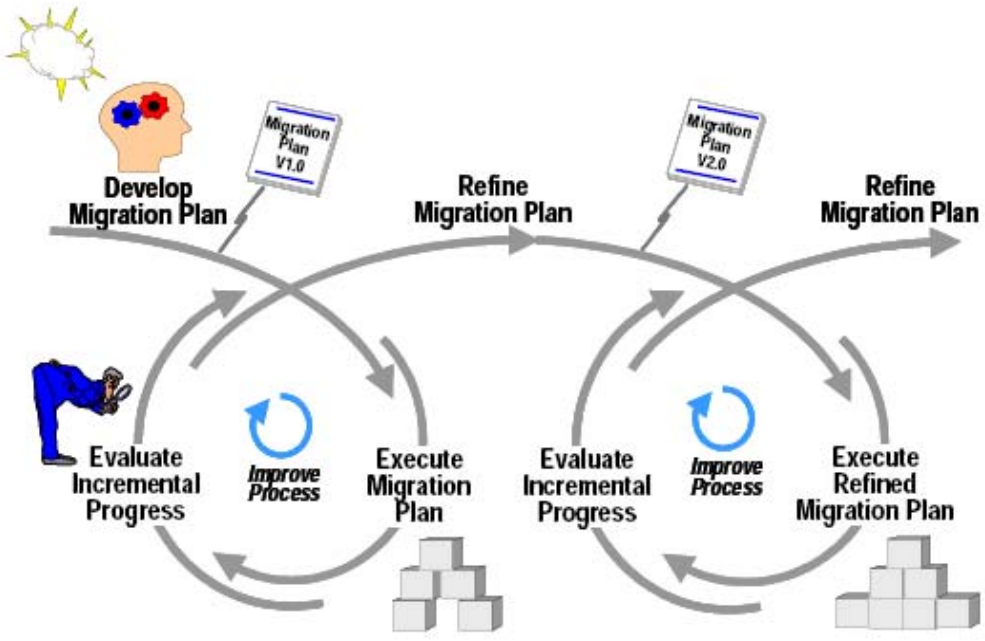


Figure 2-5: Dynamic of Migration Planning[11]

[10] see Bergey, O'Brien, Smith(2002)
[11] see Bergey, O'Brien, Smith(2002)

Figure 2-5 demonstrates vividly an active and dynamic view of migration planning, which is a continuous effort that begins with the first increment of the migration plan. The first increment of the plan identifies the primary issues and concerns. Successive iterations add more substance and make the plan actionable and manageable. The iteration provides more elaboration in the form of detailed mini-plans of action for a set of focus areas that are most relevant to the project. At any point in time, the migration plan can be viewed as the sum of the mini-plans of action.

The Department of Defense focused the analysis of the migration planning in the horizontal manner, while the other companies like Oracle design their migration planning more in the vertical manner. The Oracle Relational Migration Maps[12] is intended to provide an overview of the recommended process for the migration of an existing third-party database to Oracle. This map consists of six phases, and they are:

1. Definition: The Definition phase describes how to gather sufficient information about the source system to create migration estimates.
2. Analysis: The Analysis phase marks the true start of the project. During this phase the findings of the Definition phase are confirmed and more details about the source system are retrieved, so that more thorough project planning can be undertaken.
3. Design: During the Design phase the Migration Engineers use the information from the Definition and Analysis phases to investigate and design solutions for any migration issues that have been identified.
4. Migration: It is the phase where the migration task is implemented.
5. Transition: The Transition phase includes instantiation of the newly migrated system at the customer site, customer testing, and go-live.
6. Production: During the Production phase, it may be necessary to provide post-production support to the customer depending on what was agreed in the contract.

In face, Oracle has also utilized some concept identical to the Focus Areas. Therefore, it is concluded that to make a good migration planning it is critical to identify the main tasks and to define a reasonable road map. Chapter 2-6 is going to present the migration plan developed for BBS-S&T.

[12] see Oracle Relational Migration Map

2.6 Migration Planning for BBS-S&T

Based on the theoretical research and the practical experience from migration projects, the migration plan is designed for BBS-S&T with consideration in both vertical and horizontal aspects. Figure 2-6 outlines the vertical representation of the migration plan, which consists of seven phases, including definition, analysis/mapping, design, migration/restructure, deployment on test server, production and knowledge transfer.

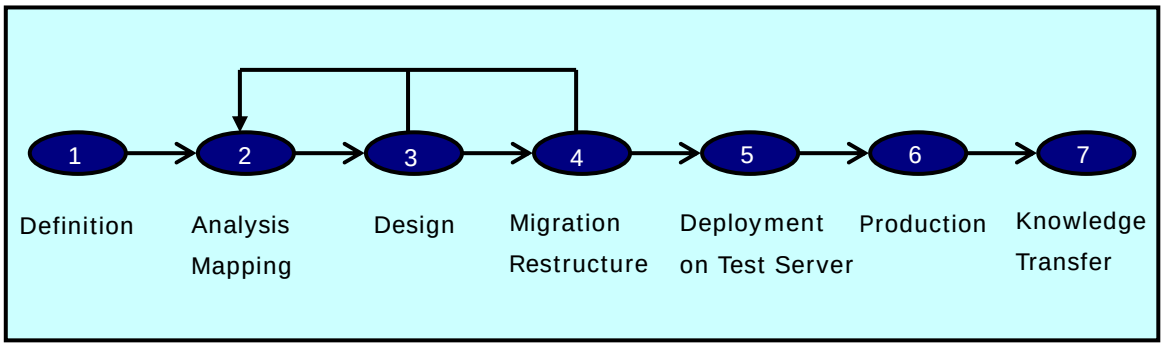


Figure 2-6: Migration Plan for BBS

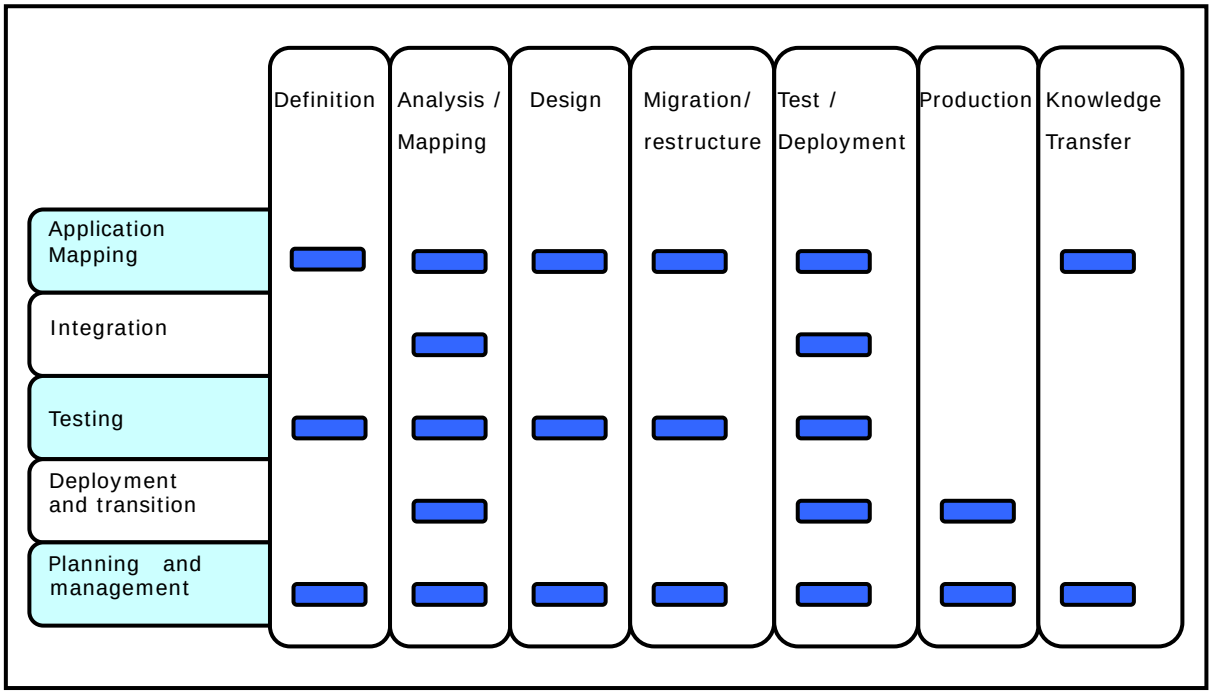


Figure 2-7: Distribution of Tasks during Migration Phases

When analyzed horizontally, the BBS-S&T migration project should pay attention to five focus areas, namely, migration planning and management, integration to the overall IT environment, testing issues and deployment and transition issues. Figure

2-7 provides a visual representation of how the five focus areas are distributed across the seven phases. In each of the seven migration phases, there are specific tasks of the above mentioned focus areas should be accomplished.

Task Distribution in Definition Phase

The definition phase is the kick-off period for the complete migration team to be formally built up and to set the common goals they want to achieve. Team members should know clearly how to obtain information about the IVY web applications to be migrated and make preparations for the real start of the migration projects. To complete the definition phase successfully, it needs to complete the following tasks.

First make sure that the development environment is well integrated to the overall IT environment. In BBS-S&T the migration projects are performed using Eclipse with Exadel plugin[13]. Due to the rapid upgrade of Java Development Toolkit, Tomcat Servlet Engine and the base libraries developed by BBS-S&T internally, the project manager should make sure to set up the local development environment identical to the productive environment, in order to minimize the risk caused by incompatibility afterwards. When the local development environment is established, the IVY application can be checked out of CVS for the analysis phase.

The project manager should start to compose the project plan. In order to make a rough estimation of the project duration and to get an impression of the project complexity, the project manager can use the migration tool developed with this project, and the details can be seen in chapter 5. This initial knowledge allows the project manager to find out the resources required to complete this project. If any external resources are needed, the contact person should be identified. Besides, the project manager can assign the tasks to team members. It is important to make sure that all the team members have the sufficient knowledge for tasks and they are informed of the available supporting resources. If necessary, team members should work out their own mini-plans, so that the project manager can better supervise the progress of the project.

No test is to be made in this early phase, but it is necessary to regulate the testing policies in terms of test methods, test priority and related reports. Ideally, test cases should be designed by team members who understand the function or technology to be tested, and each test case should be submitted for peer review. It is obvious that it is not feasible to test everything. Instead, tests should be prioritized to perform the most important ones. It is ideal to have automatic tests, which can be repeated easily. However, for test cases which can’t be done automatically, documentation of test

[13] See Exadel Tutorials & Demos

cases is a helpful supplement. Normally documentation of test cases can be done as detailed and recipe-like steps or as general description. The detailed test cases are ideal in situations where testers do not have the intimate knowledge of the original application. In descriptive test cases, the tester who has an intimate knowledge of the legacy application decides at the time of the test how to perform the test and what data to use. Besides, project manager should make sure that every team member understands and agrees with the testing policies..

Task Distribution in Analysis/Mapping Phase

The analysis and mapping phase is the second phase, but actually it signals the real start of the migration project. The main task of this phase is to acquire a good comprehension of the IVY web application in terms of its architecture and functionality. However, because the migration of web application is primarily about the mapping of web controls and its underlying business objects and business logics into the new framework, some mapping can be done in this phase with the purpose to improve the migration efficiency.

In the focus area of application mapping, three tasks need to done. One is to create the JSF web application project in Eclipse. To simplify the task a project template can be used, which helps to speed up the migration process by delivering some standard java codes. The new project created with the template needs to be adjusted to get running properly. The detailed instruction of this project template and guideline can be seen in Chapter 5. The running JSF web application project should be checked in CVS. The second task is to examine the IVY web application carefully. Meanwhile, some mapping can be performed, when appropriate. To do the work effectively, the mapping done in this phase should follow standards and good practices introduced in Chapter 5. Thirdly, problems without solutions in the knowledge should be identified, which will be dealt with in the design phase.

In the focus area of integration, the main task is to find out if there is any dependency that other web applications may have on this one, and vice versa. It needs to be clarified how the JSF web application can better fit into the overall system.

In the focus area of test, it needs to find out if the IVY project provides any test codes insides the project packages. If yes, how these codes can be reused in the JSF project. Besides, it is also the task to design task cases and decide the test methods. Since some mapping is done in this phase, tests on the mapped controls can be made. It is suggested to do the test on the page basis.

In the focus area of deployment and transition, the main responsibility is to clarify if any hardware or software needs to be deployed before that of the new JSF web

application. Information should be collected as to how the preparation can be made. The project manager can set the date roughly for deployment and find out what preparation needs to be done for the deployment and transition

In the focus area of project management, since more accurate information of the project has been obtained, the migration plan needs to be updated in terms of project duration and cost in order to reflect the current situation. Team member should report to project manager regularly with regard to project progress, identified problems and potential risk. Project manager is responsible to provide prompt feedback. Moreover, decision needs be made if any improvement is necessary. As to when the improvement should be made, the approach introduced in Figure 2-3 can assist in the decision making.

Task Distribution in Design Phase

The design phase is the time specially allocated to solve the open issues and to design how the application architecture can be optimized.

In the focus area of application mapping, the main task is to work out solutions for open issues identified in the analysis phase. The open issues may cover the development of composite components, customized validator or converter, to name a few. The new solutions need to be tested and entered into the knowledge base, so they can be shared by others. Besides, in this phase migration engineers have obtained better knowledge of the IVY web application and are able to work out optimized design for the application architecture. In case it is determined in the analysis phase that improvement should be done during the migration, migration engineer could consider how to accomplish the improvement.

In the focus area of test, the new solutions can be applied to the migration, and the migrated pages should be tested.

In the focus area of project management, project manager needs to control if the migration project is proceeding on schedule, if the allocated time and resources are sufficient and if the team members have sufficient knowledge to handle the open issues. Any external resources, which are required, should be identified in time to make arrangement in advance. Project manager should take corrective measures to deal with or avoid risks.

Task Distribution in Migration/Restructure Phase

The migration/restructure phase is when the focus is fully placed on the implementation using solutions available in the knowledge base and developed the design phase.

In the focus area of application migration, the task is to complete the migration task as a whole. The mapping done in the analysis phase needs to be restructured in order to match the design optimization made in the design phase. In case some new open issues are identified, go back to the analysis or design phase.

In the focus area of test, make sure that the functionality of each web page is tested and in the end an integrated test needs to be performed. Besides, if there is any dependency between this web application and the other, test should be done to see if they can work properly. For those test cases which can't be tested automatically, descriptive test cases are considered necessary, so that migration engineers have a good knowledge of what should be tested when the web application is deployed either on the test server or go productive. Prepare the report of the testing results.

In the focus area of Project Management, project manager should update the project plan. The date for the web application to go productive can be fixed.

Task Distribution in Deployment on Test Server Phase

In the deployment on test server phase, the main task is to deploy the JSF web application on the test server in order to verify its performance. Besides, migration engineers should collect experience of deployment, so that the web application can be deployed on the productive server smoothly.

In the focus area of application migration, in case any error takes place, the JSF web application needs to be evaluated and modified.

In the focus area of integration, the focus is to check if the web application can work compatibly with others.

In the focus area of test, the web application needs to be test carefully in terms of functionality and performance. Project manager should inform the end-users of the web application to test jointly.

In the focus area of project management, project manager needs to update the project plan and start to prepare for the documentation. Project manager should work intensively to analyze the potential risks, which may occur during the production phase. It is highly important that project manager has figured out measures to handle any possible emergency.

Task Distribution in Production Phase

In the production phase, the JSF web application is deployed on the productive server and the IVY web application is cut off.

In the focus area of deployment and transition, the IVY application is switched off and removed from the web server and the JSF application is deployed for operation.

In the focus area of test, the tests of the JSF web application should be conducted again carefully on the productive server by following the descriptive testing scenario. Migration engineers should get prepared to handle any unexpected situation.

In the focus area of project management, the project manager should make a good summary of the project and get all required documentation ready.

Task Distribution in Knowledge Transfer Phase

As indicated by Figure 2-5, migration plan should be a dynamic process. Before the official end of the project, it is of great importance to put all the new migration patterns and methods and the experience of project management into the knowledge base. If any existing migration patterns and methods need to be updated, they should be done, too.

In conclusion, this chapter presents the results of the researches in term of the migration strategies and the techniques to manage migration project and to formulate migration plans. Besides, it introduces the migration strategy and migration plan specially selected and designed for BBS-S&T. The purpose is to put emphasize on some fundamental issues that migration engineers should know in advance. In the practice, migration engineers are welcomed to develop their own migration strategy and plans to fit for their expertise and the real migration situation.

3 Comparison between IVY Presentation and JavaServer Faces

3.1 Overview of IVY Presentation

IVY Presentation[14] is designed and implemented with the objective to relieve developers from the complexity of underlying technologies. It provides standard services shared by all the web applications, so developers can concentrate on the presentation and business logics, which vary web applications from one another. On the client side, IVY Presentation supports a number of browsers including Netscape 4.7 and above, Internet Explorer 4.0 and above, Mozilla, and Opera, while on the server side, the minimum system requirements are JDK 1.3, Servlet 2.2 and JSP 1.1.[15]

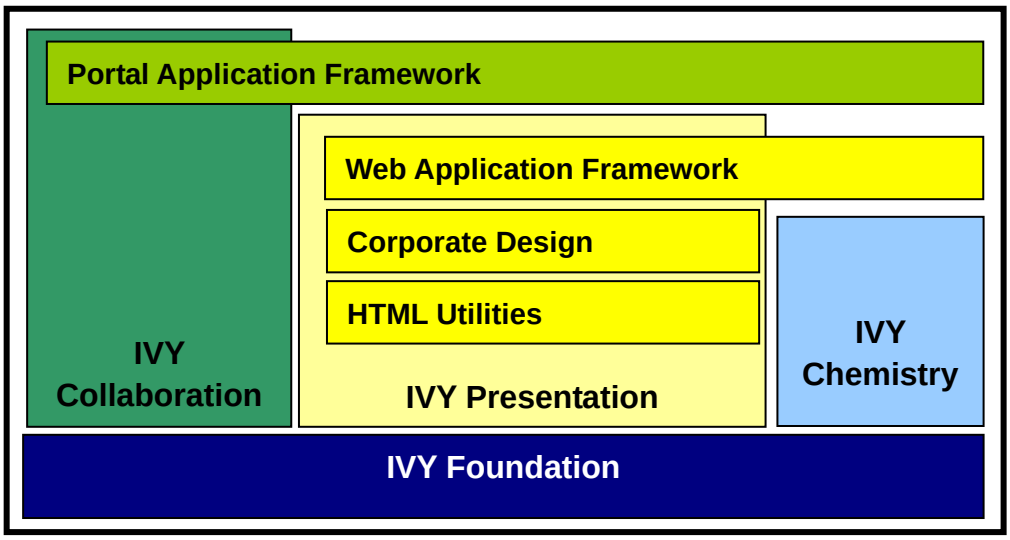


Figure 3-1: Architecture of IVY Framework

Inside the IVY Framework, IVY Presentation is the core library for the development of web applications. Figure 3-1 illustrates the relationship that it has with IVY Foundation, IVY Chemistry and IVY Collaboration. IVY Foundation[16] is the base library of IVY Presentation, IVY Chemistry and IVY Collaboration. IVY Chemistry[17] provides chemistry-oriented services. To make the use of IVY Chemistry convenient, IVY Presentation implements a web control, which wraps the services of IVY Chemistry. IVY Collaboration offers the infrastructure to develop portal applications. Its Portal Application Framework, which is built upon the Web Application Framework of IVY Presentation, is applied specially for the development of portal

[14] See IVY Presentation Documentation
[15] See Ronald Brill (2004)
[16] See IVY Foundation Documentation
[17] See IVY Chemistry Documentation

applications. Therefore, in BBS-S&T the web applications can be categorized to two types, namely the stand-alone web application and the portal application.

After the brief introduction to the relationship among the four models in IVY Framework, the following discussion is to be focused on the IVY Presentation, which consists of three components. On the bottom are the HTML utilities, which are collections of utility classes for the rendering of web controls in HTML[18]. On top of the HTML utilities is the Corporate Design (CD), which is „The official design of the logo and name of a company or institution used on letterheads, envelopes, forms, folders, brochures, etc. The house style is created in such a way that all the elements are arranged in a distinguished design and pattern.[19]. It automates the creation of web pages with the layout in accordance with the Bayer Standard Layout guideline, so that all the Bayer web pages have the identical outlook to emphasize the corporate identity of Bayer. The Web Application Framework is established upon the Corporate Design and the HTML utilities, and is composed of two parts, namely the base classes constructing the Web Application Framework and a collection of basic and complex web controls. The web control called BayerNavigation wraps the services provided by the Corporate Design for convenient use.

3.1.1 Architecture and Concepts of Web Application Framework

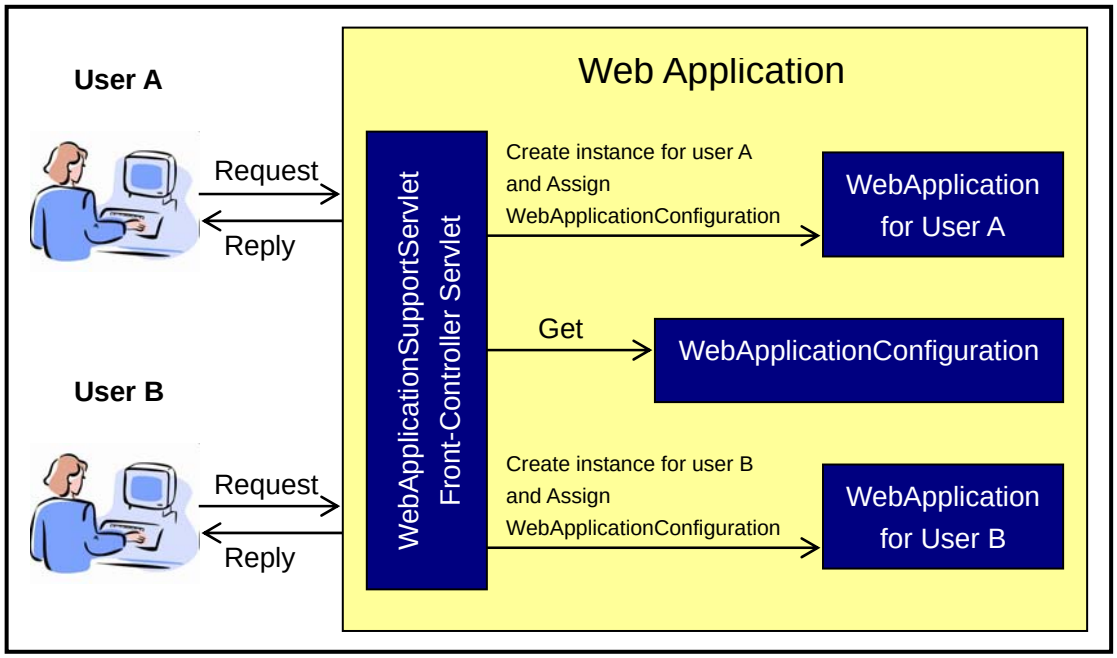


Figure 3-2: Front-Controller Pattern in IVY Presentation

[18] See HTML Tutorial
[19] See Wikipedia – Corporate Design

Since the Web Application Framework is the key component of IVY Presentation, its architecture and concepts are to be explicated in detail. IVY Presentation adopts the Front-controller[20] pattern in combination with the Model-View-Controller[21] pattern. As is shown in Figure 3-2, the class *WebApplicationSupportServlet* is the controller servlet, which is the central place to receive client requests for the desired web application. Besides the controller servlet, migration engineers should have the knowledge about other essential classes of the Web Application Framework, namely *WebApplication, WebApplicationConfiguration, WebForm and WebControl*. Their relationship is illustrated in Figure 3-3. *WebApplication* is functionally equivalent to *HTTPSession* of the *HTTPServletRequest*, so an instance of *WebApplication* is created for each user. *WebApplicationConfiguration* is utilized to set and get the configuration data of the web application. Because it is shared by all the users, the singleton pattern is applied and a copy of its instance is assigned by *WebApplicationSupportServlet* to each instance of *WebApplication*. *WebForm* offers the fundamental services to declare controls contained in one web page and to define the control behavior, so that they can handle user requests in the desired manner. *WebControl* is the super-class of web components provided by IVY Presentation.

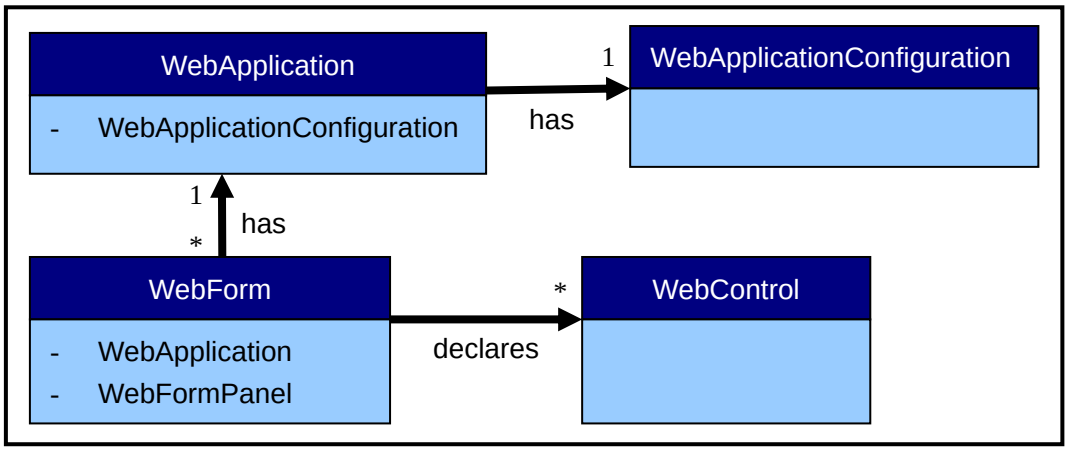


Figure 3-3: Architecture of Web Application Framework

In IVY Presentation, the role of controller is jointly played by *WebApplicationSupportServlet, WebApplication* and *Webform*. *WebApplication* determines the lifecycle in general and delegates the control of the lifecycles to *WebForm*, which has the concrete knowledge of the declared web controls, involved action listeners and the dynamic navigation rules. The role of view is played together by *WebForm* and corresponding JSP files. The web controls and their behavior are defined in *WebForm*, while JSP file is only a layout machine to determine the outlook of the whole web page. As to the role of model, IVY Presentation has no explicit

[20] See Core J2EE Patterns – Front Controller
[21] See Model-View-Controller

specification. To develop a web application using IVY Presentation, developers need to create a subclass for *WebApplicaiton*, a subclass for *WebApplicationConfiguration*, several subclasses for *WebForm* and their JSP files as is shown in Figure 3-4. To make the following explanation clear, those subclasses are referred to as *IVYWebApplication*, *IVYWebApplicationConfiguration* and *IVYWebForm*. Besides, the IVY web application needs to have a servlet configuration file web.xml, where the names of *IVYWebApplication* and *IVYWebApplicationConfiguration* are given as init-parameters.

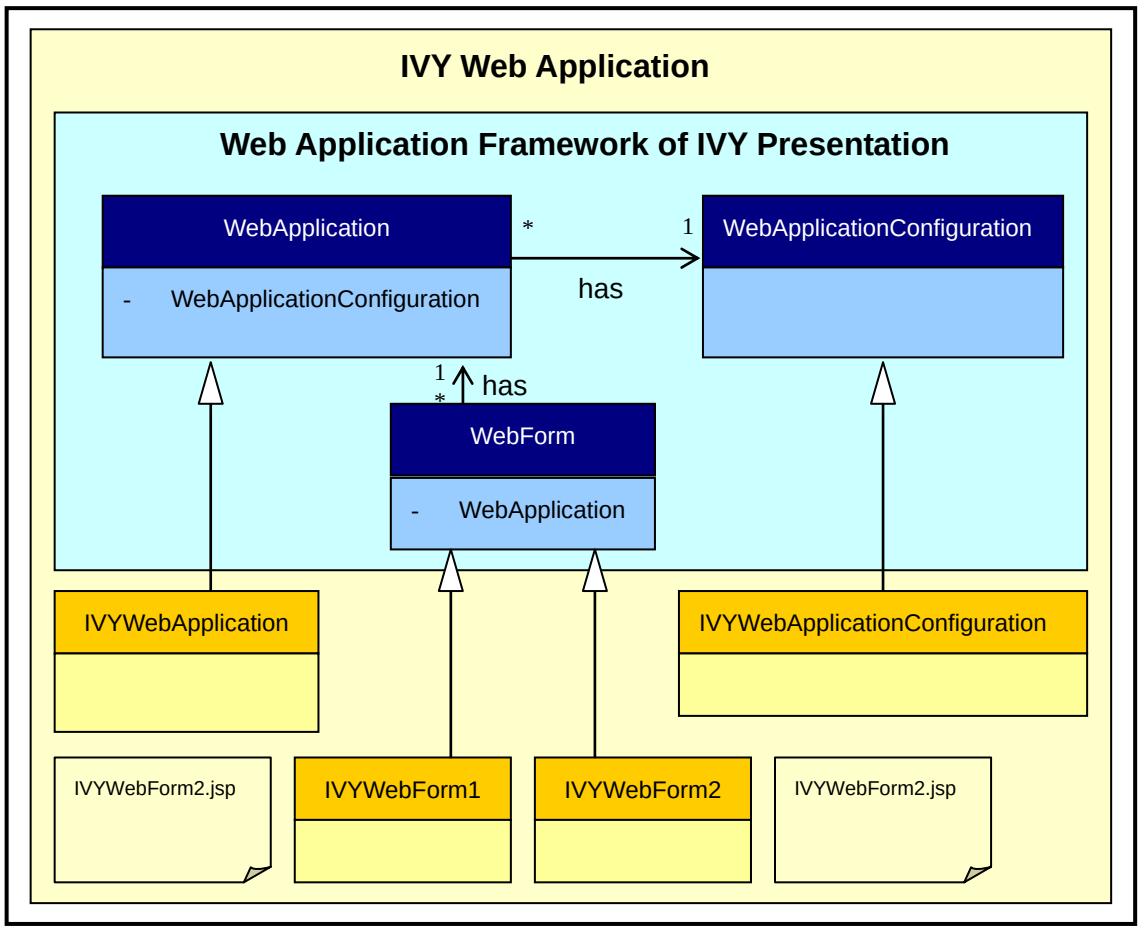


Figure 3-4: Architecture of IVY Web Application

After the general introduction to the architecture of the Web Application Framework, let's take a look at its life-cycle, which is outlined in Figure 3-5 on the next page. The life-cycle consists of five phases, namely initiate *IVYWebApplication*, determine *IVYWebForm*, process request, create successor and render *IVYWebForm*.

The initiate *IVYWebApplication* starts, when a user sends out a request for the web application. *WebApplicationSupportServlet* retrieves from its web.xml the class names of *IVYWebApplication* and *IVYWebApplicationConfiguration*. An instance of *IVYWebApplication* is created for the user on the server. Then an instance of

IVYWebApplicationConfigutaion is generated and assigned to *IVYWebApplication*. Then *WebApplicationSupportServlet* creates an instance of *WebPageContext* with *ServletContext*, *HttpServletRequest* and *HttpServletResponse* using the facade pattern and assigns the instance to *IVYWebApplication* to provide the unified access for all request information.

Next the life-cycle enters into the second phase of determine *IVYWebForm*. *IVYWebApplication* checks first if the *WebPageContext* contains target *IVYWebForm* name. If the name of *IVYWebForm* isn't available, it searches for the requested servlet path and looks up the mathcing *IVYWebForm* in *IVYWebApplicationConfiguration*. If the web form name is still null, it calls for the default authorized IVYWebForm defined in *IVYWebApplicationConfiguraton* or the default *IVYWebForm* when the default authorized *IVYWebForm* is undefined. Otherwise, it throws an exception and an error page is to be rendered. Therefore, in each IVY web application two JSP files called error.jsp and error404.jsp are used to display error messages.

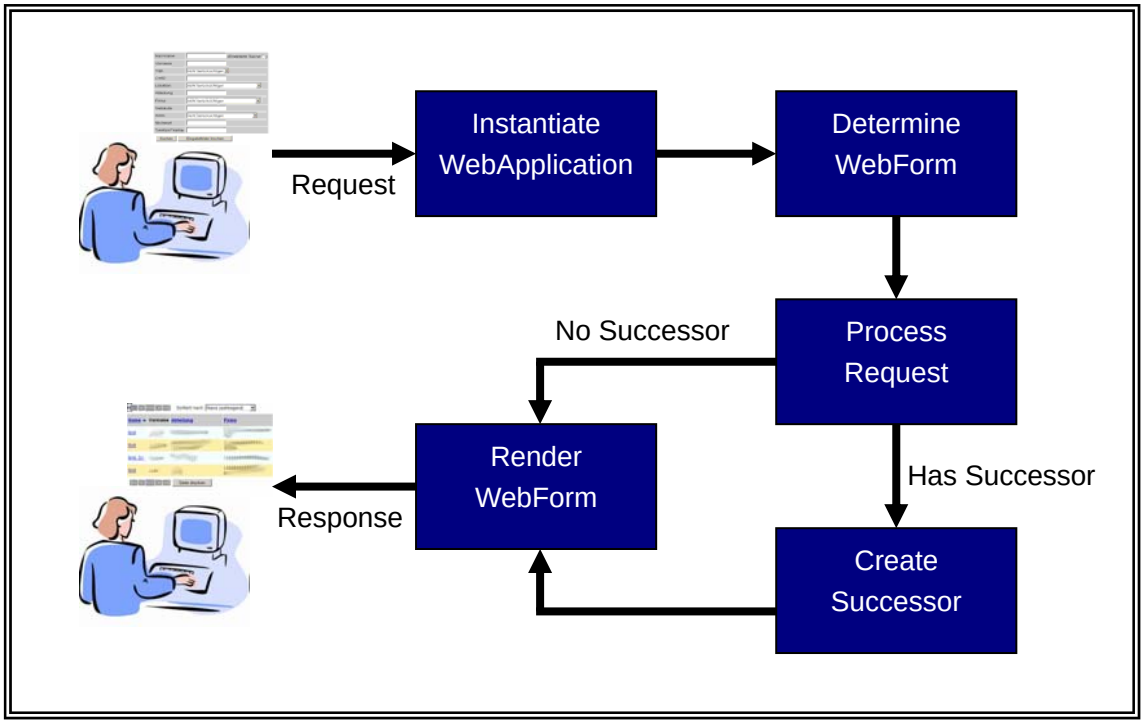


Figure 3-5: Life-Cycle of IVY Web Application

In the third phase, *IVYWebForm* updates its state and history of binding *IVYWebApplication*, creates instances of *WebControl* contained in its inner class *WebPanel*. Each *WebControl* is given a chance to check if there is any parameter in *WebPageContext* for it. Then it checks if the user has triggered any event, and the corresponding action listener, which contains the business logics, is activated. In the forth phase, *IVYWebApplication* checks if a successor of *IVYWebForm* exists. For example, when *IVYWebForm* is called for the first time, then it has no successor. In

the fifth phase, the reply *IVYWebForm* is created and rendered. The whole life-cycle will start to process the next request.

After the introduction to the life-cycle of the Web Application Framework, let's dive deeper into the above-mentioned classes for a better comprehension of their functionalities. *WebApplication* is mainly used to keep an instance of the *WebApplicationConfiguration* in order to handle the locale that users may change during the session, and to provide the multilingual support since it has the knowledge of the current locale. *WebApplication* is designed specially to substitute the *HTTPSession*, where developers must set the properties explicitly for values that they want to maintain in the session.

WebApplicationConfiguration is used to store the configuration data, which normally includes the context root, the log level, and the name and path of the properties file for multilingual support and parameters used to initiate required web services and databases. The configuration data are given in the web.xml as init-parameters and assigned to *WebApplicationConfiguration* by the control servlet.

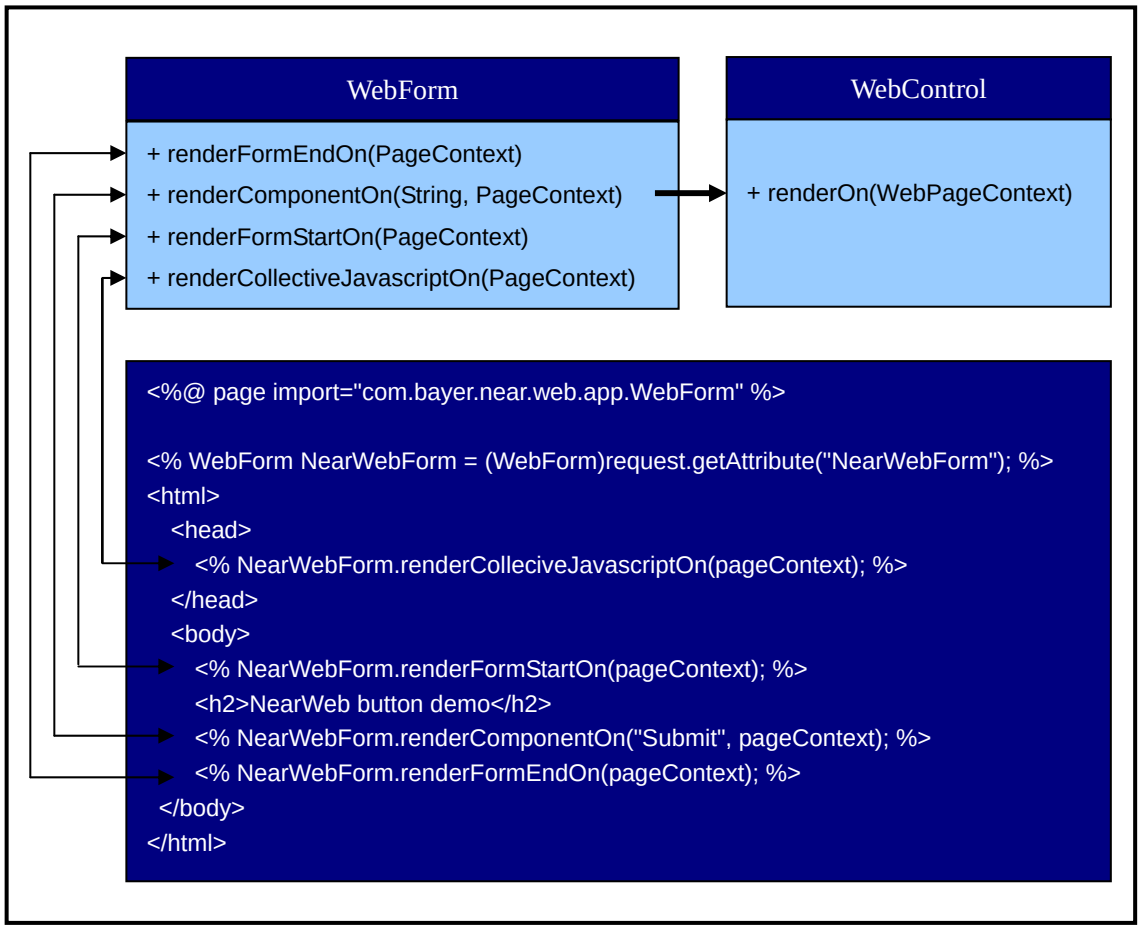


Figure 3-6: Relationship among JSP, WebForm and WebControl

WebForm provides the fundamental methods to declare web controls, to initialize them statically, to process requests, to set successor and to get the URL of the corresponding JSP and render the successor in HTML. The JSP is only a layout machine for each web form and it calls the methods of *WebForm* using JavaScript. Therefore, each *WebForm* must have a matching JSP.

WebControl is the super-class of the various web controls provided by the Web Application Framework. All the controls must implement a rendering method, which is used to render itself in HTML by *WebForm* at run time. The Figure 3-6 illustrates a standard format of JSP and how it works with its aligning *WebForm* and *WebControl*.

The web controls provided by the Web Application Framework include the basic control types for navigation, input, output, selection, command, and a number of composite controls like Calendar, Chimepro, Fileupload, etc. The detailed documentation of web controls can be seen in IVY Presentation Documentation.

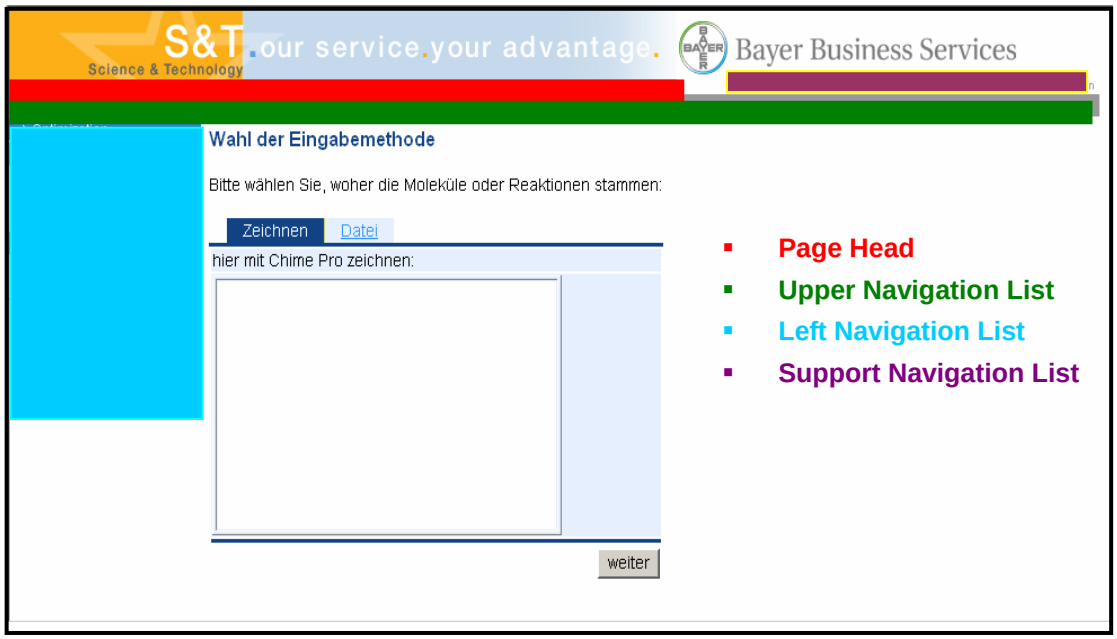


Figure 3-7: Corporate Design of Bayer

The BayerNavigation is the web control, which wraps the services of the Corporate Design mentioned earlier. It defines the webpage in terms of the page head, the upper navigation list, the left navigation list, and the support navigation list as shown in Figure 3-7. Moreover, it also specifies the colors, fonts, and images etc. With this control, developers can have the page head and the three navigation lists inside their JSP files as conveniently as using basic web controls. All the requirements stated in the Bayer Web Layout Guide and the navigation items are realized by this navigation control using a set of configuration data. The configuration data can be defined in various formats, such as XML file or Java classes. To retrieve the configuration data

flexibly, the reader concept is applied, and the *ConfiguraitonReaderManager* manages the readers for various formats, for example, the *XMLConfigurationReader* for XML file. The normal practice is to use the navigation.xml file. The automatic generation of the navigation items has two advantages. Firstly when a web page is added, renamed or removed, it is unnecessary to modify the navigation item for each related webpage. Instead, changes only need to be made in the central configuration file. Secondly when one of the navigation items is selected, the web page updates its presentation automatically.

In conclusion, the Web Application Framework is mainly used to develop stand-alone application. Due to the demand to build BBS portal, the Portal Application Framework is implemented with the basis on the Web Application Framework.

3.1.2 Architecture and Concepts of Portal Application Framework

IVY Collaboration model is developed for the construction of BBS portal.

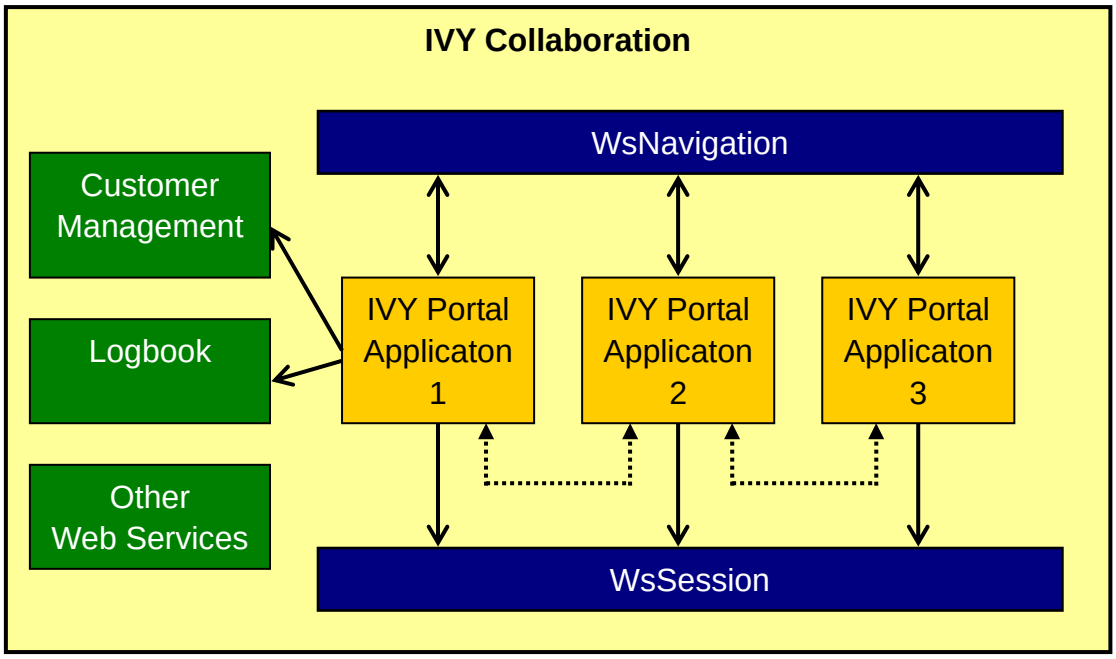


Figure 3-8: Architecture of IVY Collaboration Model

Figure 3-8 shows the architecture of IVY Collaboration. The web services WsNavigation and WsSession set up the Portal Application Framework, and must be employed by all portal applications. WsNavigation is used to generate a unified outlook of the portal. The communication between WsNavigation and portal application is a two-way connection, which is managed by the servlet *PortalListenerServlet*. When a new portal application joins the portal, it needs to register itself with the WsNavigation, which will in turn inform the other portal

applications to update their navigation configuration file dynamically. Meanwhile, WsNavigation traces the status of all the registered web applications. When it detects that one web application has been removed, it will notice the other applications. Some data need to be passed among portal applications, for example, the user authentication data and selected locale. These data are communicated using WsSession. There are some other web services, such as customer management and logbook, which can be used by portal applications, when appropriate. WsCustomerManagement is mainly used for user authentication and authorization, and WsLogbook is to log information required for business logics.

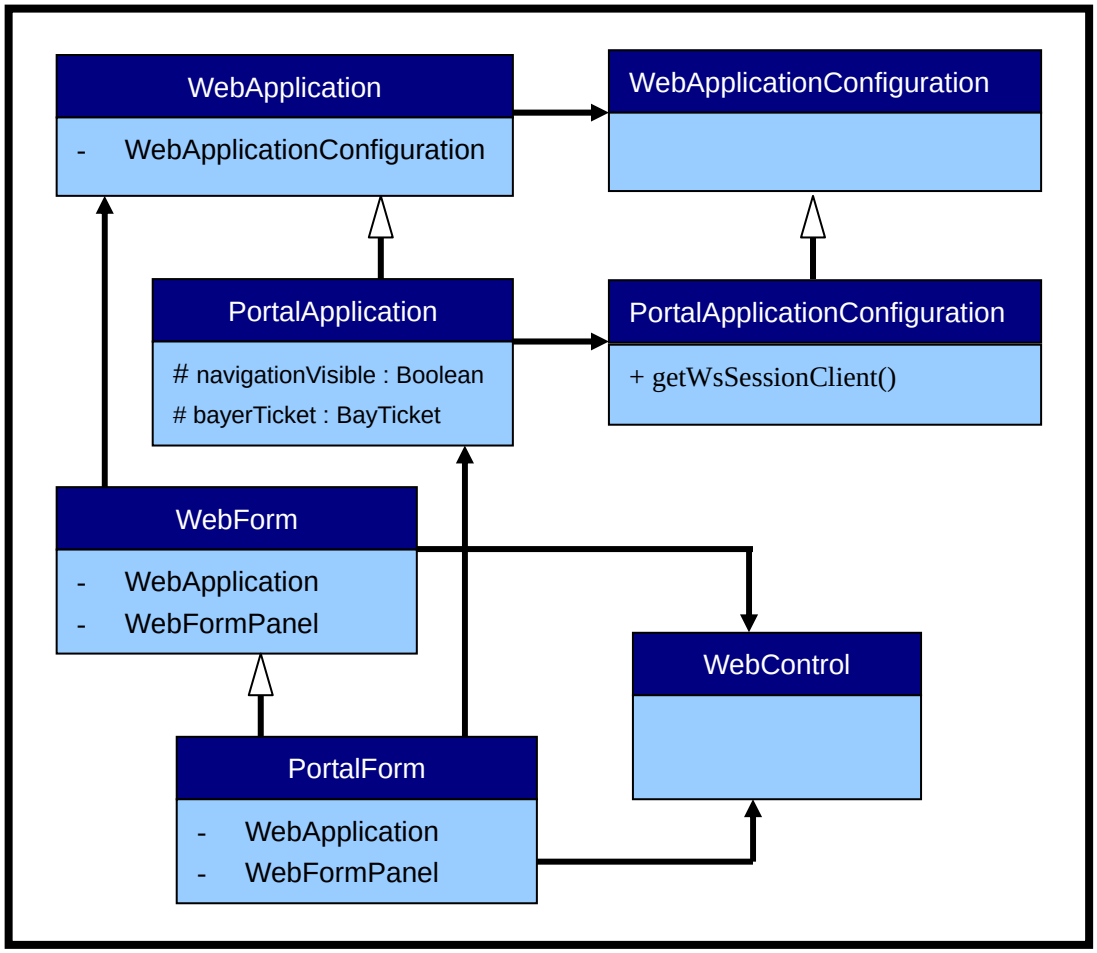


Figure 3-9: Architecture of Portal Application Framework

The portal application framework is an extension to the Web Application Framework of IVY Presentation as is illustrated in Figure 3-9. *PortalApplicationConfiguration* is a subclass of *WebApplicationConfiguration*. It allows developers to supply the configuration data of applied web services through an XML file and it provides methods to retrieve the information automatically. Because all the portal applications need to possess a BayerNavigation control, and to provide the user authentication, these two features are built into *PortalForm*, the subclass of *WebForm*. The visibility

of the BayerNavigation control, the BayTicket used to control user session and authentication by WsSession are added to *PortalApplication*, the subclass of *WebApplication*.

In conclusion, IVY Presentation utilizes the Front-Controller pattern and the Model-View-Control pattern for its architecture. Its Web Application Framework provides a set of basic and complex web controls and supports for action listener, navigation, authentication and internationalization. The Portal Application Framework is en extension of the Web Application Framework with some portal features. To distinguish the web applications developed with the two frameworks, those using the Web Application Framework are still referred to as web applications, while the others using the Portal Application Framework are called portal applications.

3.2 Overview of JavaServe Faces

JavaServer Faces (JSF) is by definition the server-side user interface component framework for java technology-based web application. JSF is based on the Servlet 2.3 and JSP 1.2. The JSF framework is responsible for interacting with client devices, and it provides tools for tying together the visual presentation, application logic and business logic of the web application. However, the scope of JSF is restricted to the presentation tier. Database persistence, web services and other back-end connections are outside of the scope of JSF, as is highlighted by Figure 3-10.

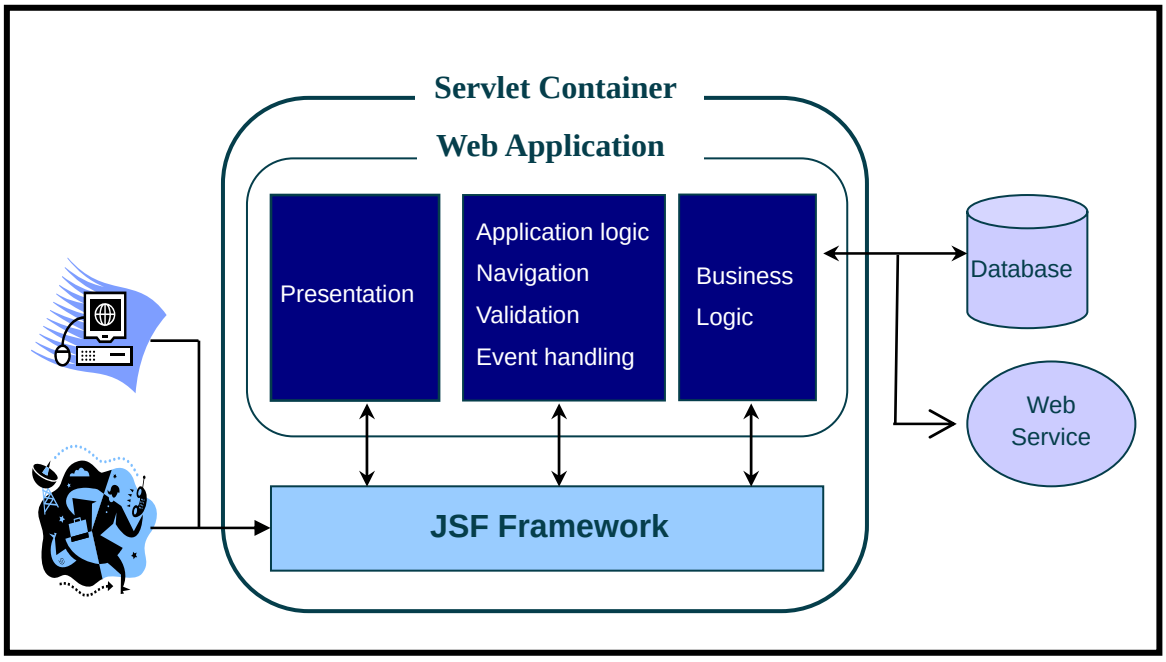


Figure 3-10: High-level Overview of JSF Framework[22]

[22] See David Geary, Cay Horstmann (2004)

As a new standard for web application development, JSF has made improvement in the following areas. It allows the creation of user interfaces from a set of standard, reusable server-side components using JSF tag libraries in JSP. It provides extendable component model and rendering architecture. It transparently saves the state information and repopulates the forms when they redisplay. Moreover, the JSF specification enables tool vendors to develop Integrated Development Environment for a more standard Web application framework.

There are several implementations of JSF available, among which the most well-known are the Reference Implementation (RI) by Sun and MyFaces by Apache. In BBS-S&T, MyFaces is selected, because it has fewer bugs than RI. Furthermore, MyFaces offers better supports for component models.

3.2.1 Architecture of JavaServer Faces

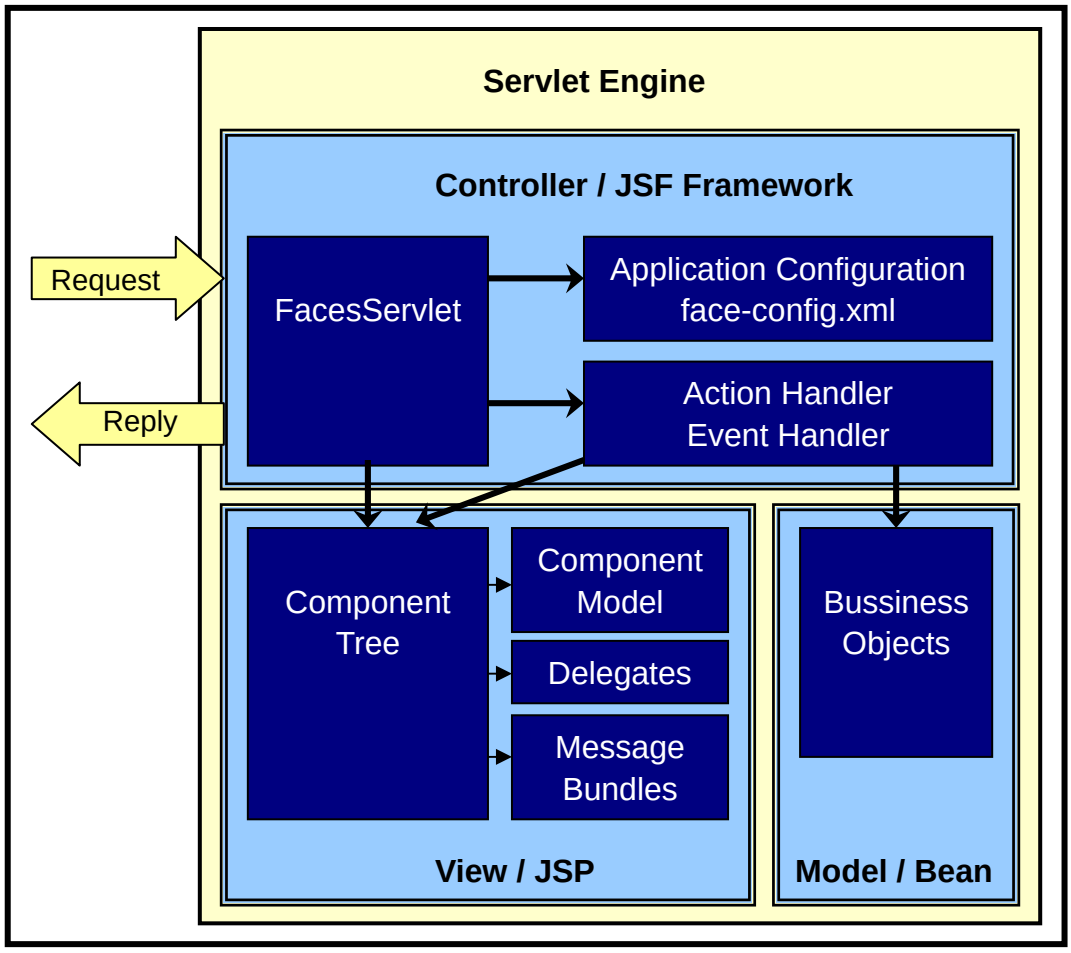


Figure 3-11: MVC implemented in JavaServer Faces[23]

[23] See Bill Dudney (2004)

The architecture of JSF is also built up the Front-Controller pattern combined with the MVC pattern. The controller servlet in MyFaces is implemented by the class *javax.faces.webapp.FacesServlet*. Every JSF web application needs to supply an application configuration file, which is by default called faces-config.xml. Through this configuration file, *FacesServlet* retrieves the knowledge of the bean resources, the navigation rules, the supported locales and other important application-related information. The business logic and application logic are dealt with by action handler and event handler respectively.

As is illustrated in Figure 3-11, *FacesServlet* together with faces-config.xml, the action handler and the event handler plays the role as controller. JSP, which contains web components declared by JSF tag libraries, serves as view, and bean object as the model. *FacesServlet* synchronizes the view and the model by wiring a view component to a bean property of a model object. Besides, it responds to the user requests through action and event handlers and then routes the requests to codes that update the model and the view. Figure 3-12 shows the operation workflow of *FaceServlet*. When it receives a user request, it creates the *FacesContext* and then passes the control to the Lifecycle, which will in turn process *FacesContext* in various phases.

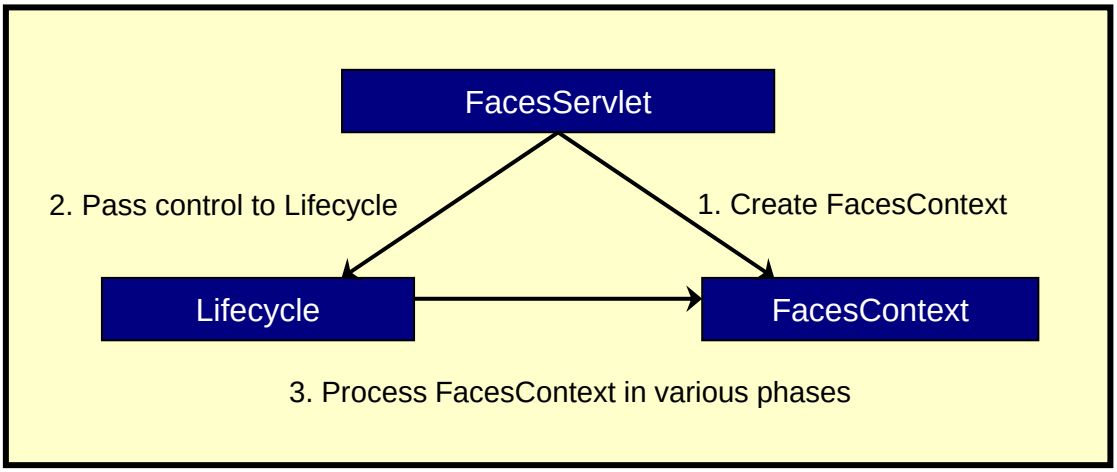


Figure 3-12: Request Processing Flow[24]

The JSF specification defines six distinct phases[25] as is outlined in Figure 3-13. The restore view phase retrieves the component tree for the requested page if it was displayed previously, or constructs a new component tree if it is displayed for the first time. When the page is called for the first time, FacesServlet initializes the application codes and reads the matching JSP, which contains JSF tags. Each tag has an associated tag handler class and the tag handlers collaborate with each other to build a

[24] See Deepak Goyal, Vikas Varma
[25] See David Geary, Cay Horstmann (2004)

component tree. The components tree is a data structure that contains java objects for all the user interface elements on the requested page. Then the web application goes directly to its last phase render response as is shown by the red dash line in Figure 3-13. In the JSP texts that are not JSF tags are simply passed through. Each JSP tag associates with a component, has a default renderer that produces HTML output. This process to convert JSF tags into HTML is called encoding. The encoded page is then sent to the browser on the client side. After the page is displayed in the browser, the user fills in the form fields and clicks the submit button. The browser sends the form data back to the web server, formatted as a post request. As part of the normal servlet processing, the form data are placed in a hash table that all components can access.

Afterwards, the life-cycle enters into the second phase Apply Request Values. The JSF framework iterates over the component objects in the component tree, and gives each component a chance to inspect that hash table. This process to retrieve user's input parameters is called decoding. Each component decides on its own how to interpret the form data.

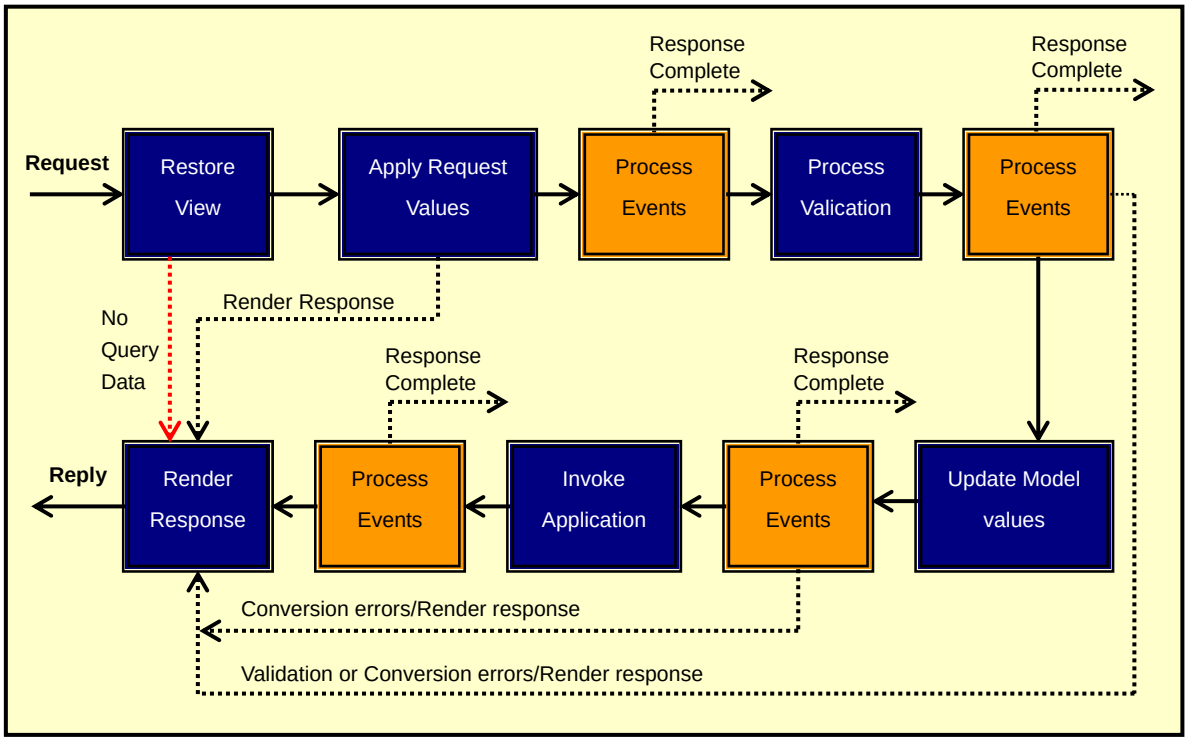


Figure 3-13: Life-cycle of JavaServer Faces[26]

In the process validation phase, the submitted string values are first converted to local values, which can be objects of any type. Validators can be attached to a JSF to perform correctness checks on the local values. If validation passes, the JSF life cycle

[26] See David Geary, Cay Horstmann (2004)

proceeds normally. However, when conversion or validation errors occur, the JSF Framework invokes the render response phase directly and redisplays the current page, so that the user has another chance to give correct inputs. After the converters and validators have done their work, it is assumed that it is safe to update the model data.

During the update Model phase, the local values are used to update the beans that are wired to the components. In the invoke application phase, the action method of the button or link component that caused the form submission is executed. That method can carry out arbitrary application processing. It returns an outcome string that is passed to the navigation handler. The navigation handler looks up the next page. Finally, the render response phase encodes the response and sends it to the browser. When a user submits a form, clicks on a link, or otherwise generates a new request, the cycle starts anew.

This is a brief overview of the architecture and life-cycle of JavaServer Faces framework. In the next sub-chapter, the context will be focused on the introduction to its concepts.

3.2.2 Key Concepts of JavaServer Faces

The key concepts of JSF includes User Interface (UI) Components, renderers, model objects, validators and converters, events and listeners, expression language, navigation and internationalization support.

Components and Renderers

In JSF UIComponent and UIComponentBase are the base classes for all user interface components on the server side. JSF provides standard UIComponent subclasses including UICommand, UIForm, UIOutput, UIInput, UIGraphic, UISelectBoolean, to name a few. UI Components can render themselves using the default HTML 4.01 render kit shipped with JSF or delegate the display task to a renderer, which is responsible for encoding and decoding components.

Each UI Component has a corresponding tag handler, which declares the component in JSP. The JSF tag libraries are composed of core tags and HTML tags. The core tags are independent of the rendering technology. Most of the core tags represent objects added to components, for example, attributes, listeners, converters, validators, facets, parameters, select items. In addition, they also contain tags for defining views and subviews, loading resource bundles and adding arbitrary text to a page. The HTML tags generate HTML specific markup for components of type inputs, outputs, commands, selections and others. The Login.jsp in Figure 3-17 is a simple example of using JSF tags. The detailed usage of JSF tag libraries can be seen in the book Core

JavaServer Faces. The declared components in one JSP are stored in a tree called view on the server. JSF retains the state of components transparently on the server side or on the client side. Developers only need to set the mode of state management in the servlet configuration file web.xml. If the server side is selected, the component hierarchy is stored in session, while on the client side components are serialized and stored in a hidden field. MyFaces doesn't support the server-side state management very well, so the client-side state management is recommended.

Model Objects

Model objects in JSF are regular JavaBeans with read and write properties. The UserBean.java in Figure 3-16 shows a simple example. They are used by JSF as the conduits between the user interface and the backend of the application. JSF introduces two new terms: managed bean and backing bean. JSF provides a strong managed-bean facility. JavaBean object managed by a JSF framework is called managed bean, which describes how a bean is created and managed. It has nothing to do with the bean's functionalities. Managed bean need to be configured in faces-config.xml with a proper scope as is shown in Figure 3-16.

JSF supports four bean scopes, namely none, request, session and application. The request scope is short-lived. It starts when an HTTP request is submitted and ends when the response is sent back to the client. The session scope persists from the time that a session is established until session termination. The application scope persists for the entire duration of the web application. A bean has scope none if it is never requested from a JSP page. Managed beans can be wired together as is shown in Table 3-1, it is important to make sure that their scopes are compatible.

A bean of this scope	Can use beans of these scopes
none	none
application	none, application
session	none, application, session
request	none, application, session, request

Table 3-1: Scope of JSF Managed Beans

The backing bean defines properties and handling-logics associated with the UI components used on the page. Each backing-bean property is bound to either a component instance using JSF API or its value. A backing bean also defines a set of methods that perform functions for the component, such as validating the component's

data, handling events that the component activates, and performing processing associated with navigation when the component activates.

A typical JSF application couples a backing bean with each page in the application. However, sometimes in the real world, forcing a one-to-one relationship between a backing bean and a page is not the ideal solution. It can cause problems like code duplications. Therefore, several pages may share one backing bean behind the scenes.

Validator and Converter

In order to guarantee that invalid inputs will never end up in the business logic, JSF carries out the conversion and validation in the Processing Validation phase of the life-cycle. Because the web user interface deals exclusively with strings and the web application stores data of various types, the converter is fully responsible for converting user inputs into the component values, and vice versa. Besides, JSF supplies standard converters for the conversion from String to a primitive type or BigInteger/BigDecimal. JSF also provides standard converters for numbers and dates and these converters needs to be explicitely specified with <f:convertNumber> and <f:convertDateTime> and their attributes as is shown in Figure 3-14. Besides, it also supports the development of custom converters. Details can reference the book Core JavaServer Faces.

Convert Date:	**Convert Number:**
<h:inputText value="#{payment.date}"> <f:convertDateTime pattern="MM/yyy" /> </h:inputText>	<h:outputText value="#{payment.amount"> <f:convertNumber type="currency" /> </h:outputText>

Figure 3-14: Example of JSF Default Converter

The converted values are not immediately transmitted to the beans. Instead, they are first stored inside the component objects as local values. Then validator performs the correctness check on local values.

Validate String Length:	**Validate Long Range:**
<h:inputText value="#{payment.card }" required="true" > <f:validateLength minimum="13" /> </h:inputText>	<h:outputText value="#{payment.amount" required="true" > <f:validateLongRange maximum="1000" /> </h:outputText>

Figure 3-15: Example of JSF Default Validator

JSF provides standard validators to check string lengths and numeric ranges using <f:validateLength>, <f:validageLongRange> and <f:validateDoubleRange> and attributes as is exemplified in Figure 3-15. It also supports to develop custom validators. Details can reference the book Core JavaServer Faces.

Both converter and validator are combined with message. When a conversion error or a validation error occurs, the component, which failed, posts a message and declares itself to be invalid. The JSF redisplays the current page immediately after the process validations phase has complete.

Expression Languages

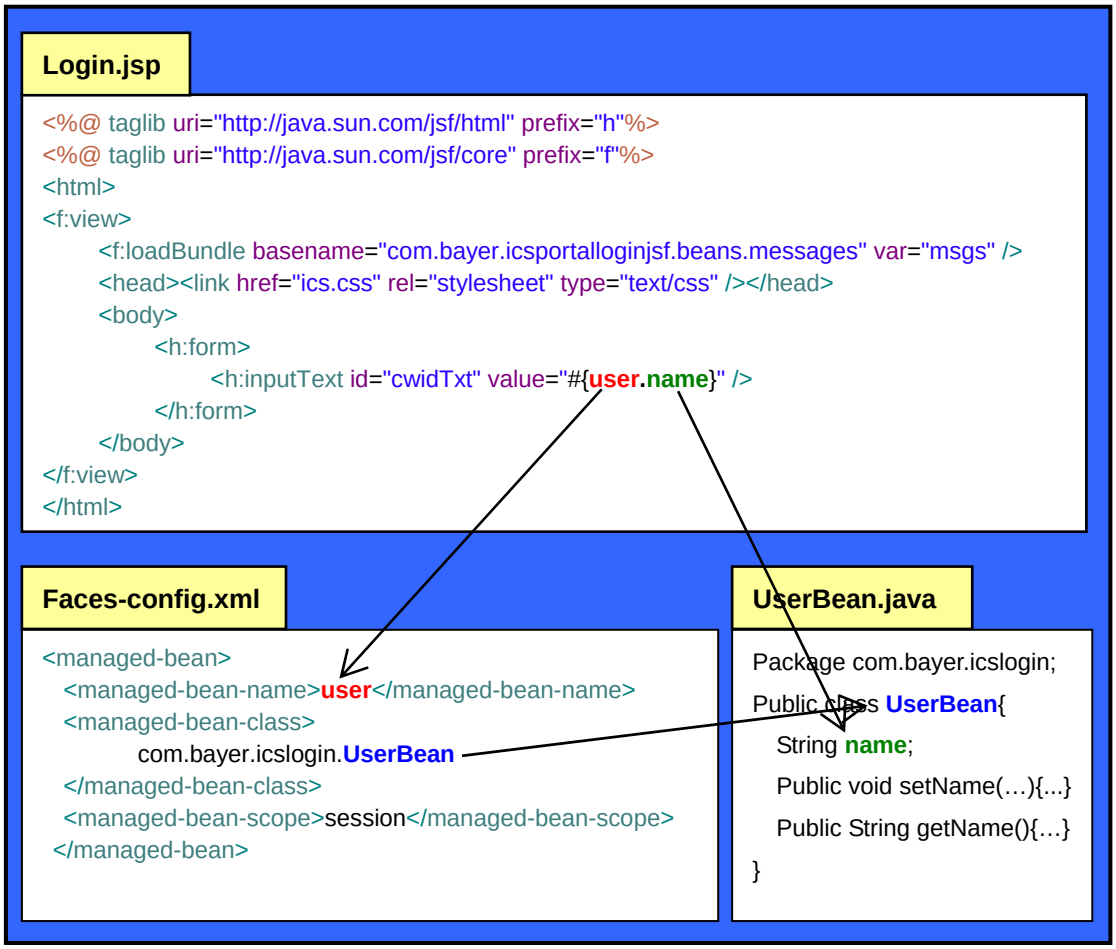


Figure 3-16: Binding UI to Managed Bean

Expression languages are used to assocaite UI components with backing beans and model objects. Properties of beans are referenced with value binding using the expression #{myBean.myProperty}, while methods are referenced with method binding using the expression #{myBean.myMethod}. Besides, JSF supports the

mixture of literal values and implicit variables. Figure 3-16 illustrates how faces-config.xml, bean and JSP coordinate with each other.

Events and Listener

JSF supports three kinds of events, namely value change events, action events and phase events. Value change events are fired by input components, such as h:inputText, h:selectOneRadio, and h:selectManyMenu, when the components's value changes and the enclosing form is submitted. The value change listeners are notified after the Processing Validations phase of the life-cycle.

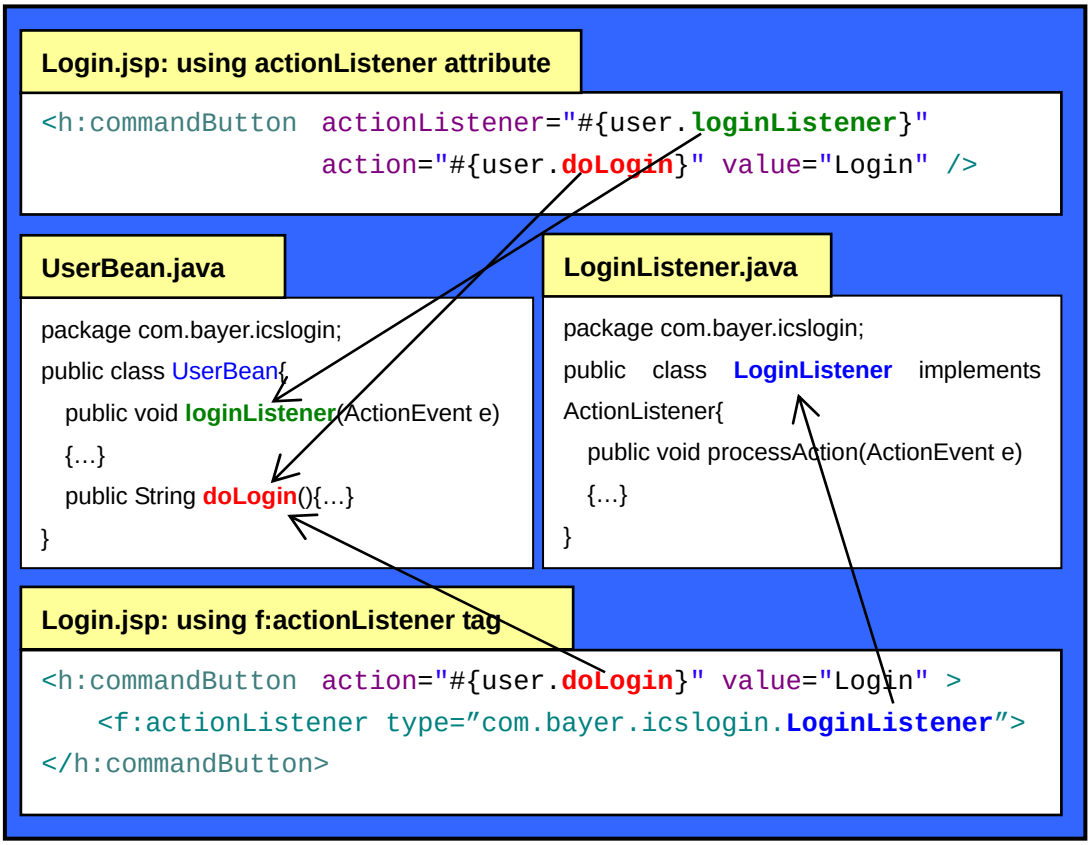


Figure 3-17: Usage of Action Listener Attribute and Tag

Action events are fired by command components, when the component is activated. To handle action events, JSF introduces both action listener and action to separate user interface logic and business logic. Action is designed for business logic and participates in navigation handling and it has no access to the event and component that fire them. The action listener typically performs user interface logic and doesn't participate in navigation handling. Action listeners and actions are notified after the Invoke Appliation phase of the lifecycle and action listeners are executed prior to actions. Listeners can be implemented either by a backing bean method or a separate class. Value change listeners must implement the ValueChangeListener interface,

while the action listeners must implement the ActionListener interface. The Figure 3-17 demonstrates the usage of action listener attribute and tag respectively.

Phase events are triggered before and after each life-cycle phase. Unlike value change and action listeners which are attached to individual components, phase listeners need to be specified in faces-config.xml. Phase listeners can be created as many as required and they are invoked in the order in which they are declared in the faces-config.xml. Phase listeners need to implement the PhaseListener interface.

Navigation

JSF provides full support for declarative navigation, which depends on the outcome of action methods to select the next page. Figure 3-18 demonstrates what takes place when users click a command button whose action attributes is a method binding. The specified bean is retrieved and the referenced method is called. The returned string is passed to the navigation handler, which in turn look up the next page. In case an action returns null, it indicates that the same page is to be redisplayed.

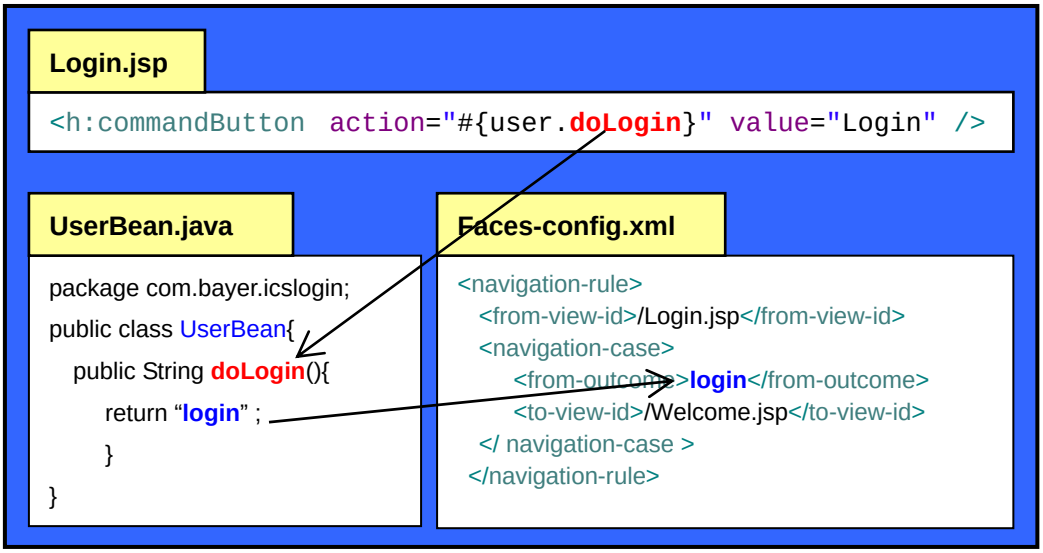


Figure 3-18: Dynamic Navigation

Internationalization

JSF provides internationalization supports using the Java properties file, which is called message bundle in JSF. The message bundles contain message string in key-value pairs. The bundle file can be localized by adding a locale suffix to the file name, for example, messages_de.properties.

As Figure 3-19 shows, the message bundle must be declared in JSP and faces-config.xml. With the default and supported locales in faces-config.xml, JSF can

find the best matching locale according to the request header sent by the client browser.

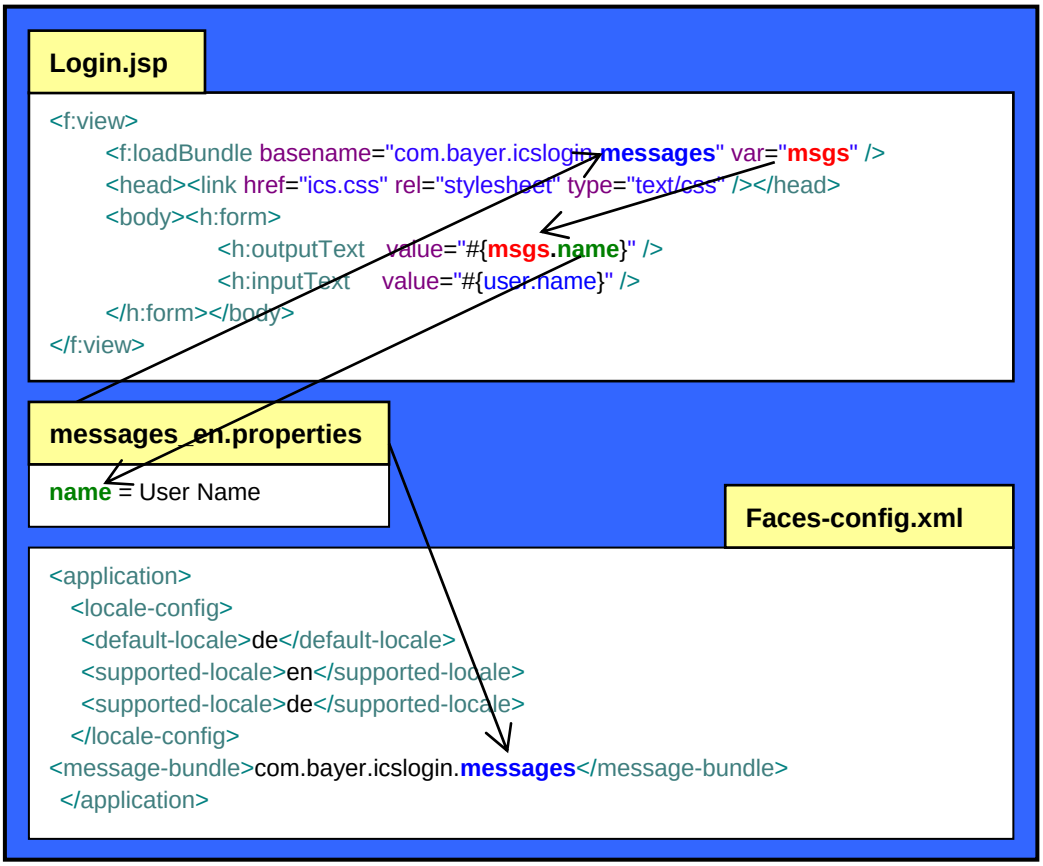


Figure 3-19: Internationalization and Localization

In conclusion, this sub-chapter introduced the architecture, life-cycle and key concepts of the JavaServer Faces Framework. Some examples have the provided to exemplify the concepts of User Interface (UI) Components, renderers, model objects, validators and converters, events and listeners, expression language, navigation and internationalization support. With the knowledge of both IVY Presentation and JSF, the comparison of the two web framework is made in the next sub-chapter.

3.3 Comparison between IVY Presentation and JavaServer Faces

To make preparation for the development of mapping rules, the comparison between IVY Presentation and JSF is carried out and the comparison results are highlighted in Table 3-2 on the next page.

JSF demands higher system settings than IVY Presentation does. JSF needs to run with JDK 1.4, Servlet 2.3, JSP 1.2, WebSphere 5.2 and Tomcat 4.1, while IVY Presentation requires lower settings of JDK 1.3, Servlet 2.2, JSP 1.1, WebSphere 3.5 and Tomcat 3.x and iPlanet.

JavaServer Faces	IVY Presentation
Minimum System Requirements	
JDK 1.4, Servlet 2.3, JSP 1.2 WebSphere 5.2, Tomcat 4.1	JDK 1.3, Servlet 2.2, JSP 1.1 WebSphere 3.5, Tomcat 3.x, iPlanet
Architecture	
Front-Controller pattern Model-View-Controller model Mix with other servlets possible	Front-Controller pattern Model-View-Controller model Mix with other servlets possible
Central Components	
FaceContext Faces-config.xml	WebPageContext WebApplication WebApplicationConfiguration WebForm
LifeCycle	
Defined lifecycle Connection to lifecycle through Faces-config.xml and event listeners	Define lifecycle Connection to lifecycle through implementation of the strategy pattern and event listener
State Management	
Service of the framework to provide server and client state management through configuration	State is stored on server. Provides API for client state, but must be implemented
UI Components	
Provides both basic and complex components	Provides both basic and complex components
Events	
Action event, value-change event and phase event	Action event
Validation and Conversion	
Framework provides default validators and converters and Interfaces for customized validators and converters	No direct support, must be implemented
Navigation	
Defined in Faces-config.xml and dependent on the event listener	Implemented in the action listeners
Other Supports	
Internationalization API for complex components Value-binding Method-binding	Internationalization API for complex components Authentication Logging

Table 3-2: Comparison between IVY Presentation and JSF

Both frameworks utilize the Front-Controller pattern. In IVY Presentation the controller servlet is *WebApplicationSupportServlet,* and the *WebPageContext* maintains the data of *ServletContext, HttpServeletRequest* and *HTTPServletResponse*, while in JSF the controller servlet is *FacesServle,t* and the *FaceContext* keeps the information. So they both utilize the servlet configuration file web.xml and they support to work with other servlets as well.

The Model-View-Controller pattern is the common architecture adopted by both IVY Presentation and JSF, but the implementation by the two frameworks is varied. In IVY Presentation, the role of controller is jointly played by the *WebApplicationSupportServlet, WebAppliation* and the actions attached in *WebForm*. There is no explicit specification and support for model. Data objects used to hold values of controls are stateless. In case the state of the data objects needs to be retained, they are either stored in *WebApplication* as member variables, or passed through *WebForm*s as hidden value or through the constructors of *WebForm*s. In IVY web applications, subclasses of *WebForm* are views where web controls are created. JSP plays only a minor role as the layout machine referring to web controls in *WebForm*s through JavaScript. In JSF there is clearer separation among model, view and controller. *FacesServlet* together with faces-config.xml, action handler and event handler operates as controller. JSP serves as view and JavaBean as model. Besides, JSF provides good facility to manage beans.

Both IVY Presentation and JSF have defined life-Cycle. In IVY Presentation, web application is connected to the framework by extending the abstract classes of *WebApplication*, *WebApplicationConfiguration* and *WebForm* using strategy pattern. In JSF, web application is attached to the framework by declaring managed beans, navigation flow and other important information through the application configuration file faces-config.xml.

In IVY Presentation the state is stored on the server, and support for client state must be self-programmed using API. JSF provides the state management on both server and client sides, and developer only needs to set the mode in web.xml.

IVY Presentation and JSF support validator and converter in different approaches. In IVY Presentation strategy pattern is applied. To implement validator and converter some methods in *IVYWebApplication* need to be overridden to get desired performance. However, in JSF the validator and converter are declared as listener. The lifecycle of JSF leaves specific interface for the registration of validator and converter. Besides, JSF offers standard converters and validators combined with messages.

JSF supports three kinds of events, namely value change event, action event and phase event and it motivates clear separation between user interface logic and business logic. IVY Presentation supports only action events, which are attached to the command controls in *IVYWebForm*.

JSF applies the declarative navigation, which is controlled by navigation handler according to the outcome of actions. The navigation flow is specified in faces-config.xml. In IVY Presentation the navigation flow is dynamically determined by the setSuccesser method of the action listener bund to command controls.

Last but not the least, JSF doesn't provide supports for security and logging utilities, and some additional efforts are needed to make JSF compatible with IVY Framework.

The practical experience obtained through the migration projects has proven that compared to IVY Presentation JSF is easier to master and convenient for use. JSF achieves a better separation between the web development framework and the web application. Web developers don't need to know much about the inner architecture of the framework, so they can better concentrated on the development of the web application using the services provided by the framework. Besides, JSF has a better implementation of view than IVY Presentation. The JSP and JSF tag library combination is much easier for use than the JSF and WebForm combination in IVY Presentation. The connection to the life-cycle of JSF using configuration file is more convenient and flexible than that of IVY Presentation using strategy pattern. Moreover, JSF provides more favorable services, such as value-binding, method-binding, validation and conversion, which has to a great extend simplify the web development task. The source codes of JSF applications have much clearer structure and less dependency than those of IVY application. Therefore, the JSF applications are more readable and easier to comprehend, so the maintenance of JSF applications is then less expensive. The performance test and the quality test presented in chapter 6 will also confirm that JSF web applications produce more throughputs and possess better code quality. BBS-S&T has developed the so-called IVY Faces model, so JSF can be implemented more conveniently with IVY Framework for services of Corporate Design, chemical-oriented supports, security and logging support.

In conclusion, it is a wise decision to migrate web applications from IVY Presentation into JSF. After BBS-S&T finally standardizes on JSF, its web development projects can be carried out with higher quality and greater efficiency.

4 Development of Mapping Rules

In accordance with the model-driven migration approach, the next step is to development mapping rules with the basis on the comprehension of both IVY Presentation and JavaServer Faces frameworks. This chapter presents the detailed description of the mapping rules designed for both web and portal applications.

For any web application, it is necessary to have the following building items.

- Framework Libraries
- Servlet Configuration File
- Application Configuration Class
- Navigation Web Control
- Message Resources
- Business Objects and Business Logics
- Web Forms
- Validation and Conversion
- Composite Controls
- Security
- Logging
- Build Scripts
- Tests

Therefore, the mapping rules for these items are developed and introduced below.

4.1 Mapping of Framework Libraries

The framework is included in a web application in the form of libraries, for example, IVY Foundation as nearweb.jar, IVY Presentation as nearweb.jar, IVY Chemistry as ivychemistry.jar, the Portal Application Framework as nearportal.jar, and the IVY Collaboration as various jars corresponding to web services. To have JSF framework in the migrated web application, the Apache implementation myfaces-impl.jar needs to be added to the web application libraries. Besides, to use Apache Myfaces some third party libraries need to be applied. Further information can be found in MyFaces library dependencies documentation[27]. Moreover, when composite web components developed by Apache MyFaces are used, the library tomahawk.jar needs to be included. With the purpose to integrate the JSF into the IVY Framework, IVY Faces ivyfaces.jar is required as well. For ICS portal application, the library icdev.jar must be added, too.

[27] See Library Dependencies

Therefore, to map the framework, MyFaces-related libraries need to be added to the web application library path, which is by default set to /WebContent/WEB-INF/lib. Since IVY Faces is built upon IVY Presentation and the migrated JSF application still requires the services offered by IVY Framework, IVY-related libraries should be added to the library path as well. However, remember to use the latest version of IVY libraries. For IVY portal applications, nearportal.jar and icdev.jar are additionally demanded. In the end, the migrated web applications or portal applications should have the libraries as is shown in Figure 4-1.

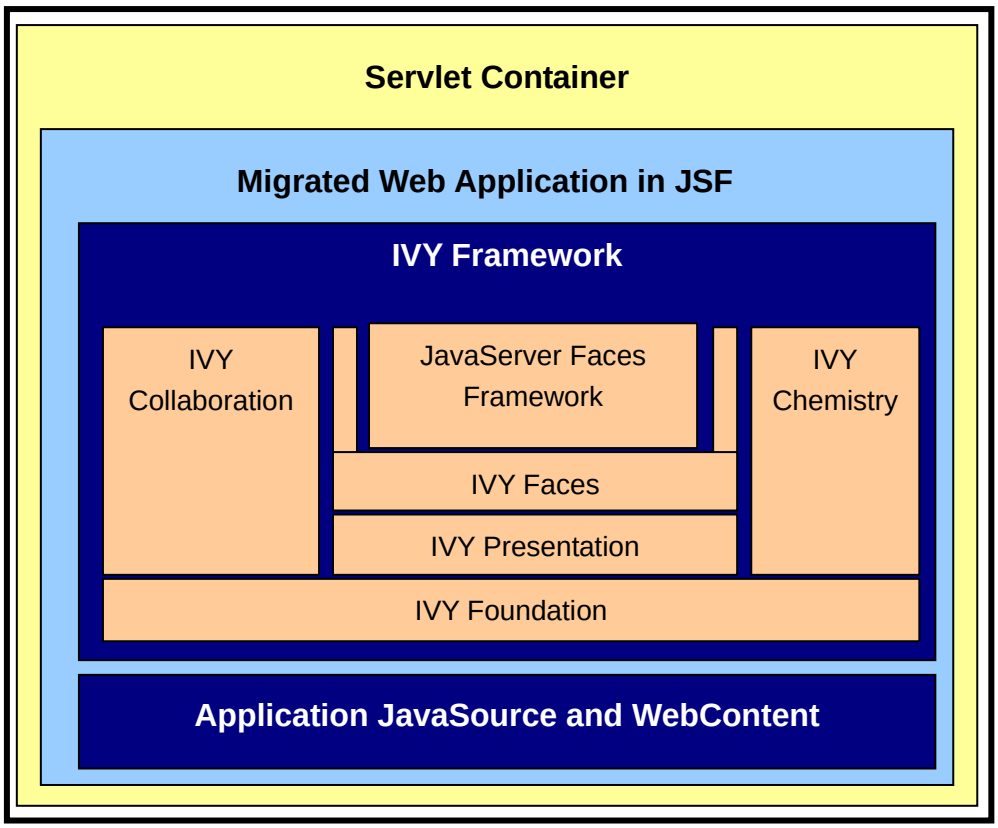


Figure 4-1: Add JSF Framework into Web Application

4.2 Mapping of Servlet Configuration File

Both IVY and JSF web applications require the servlet configuration file web.xml. The servlet declaration should be changed to *FacesServlet* together with the servlet-mapping element. Migration engineers should pay attention to three context parameters. The first parameter is javax.faces.CONFIG_FILES, which specifies the relative path and name of the JSF application configuration file. By default, /WEF-INF/faces-config.xml is the value of this parameter. The second one is javax.faces.STATE_SAVING_METHOD, which sets the mode of state management. As is mentioned in Chapter 3, it is recommended to use the client mode for Myfaces. The third one is org.apache.myfaces.AUTO_SCROLL, which value must be set to false for

ExcelTest. Besides, org.apache.myfaces.webapp.StartupServletContextListener needs to be specified as listener. For both migrated web and portal applications, the declaration of servlets GraphicProvider and FrameProvider and their servlet mappings are required, so that BayerNavigation control can be used to handle frame and graphical objects on the web pages. The index.html defined as welcome file simply redirects the request to the entry page of the JSF web application.

For portal application exclusively, the IVYConfiguration, which is to be explained in chapter 4.3, needs to be specified as listener. Besides, to build up the communication with WsNavigation web service, the servlet PortalListenerServlet and its servlet mapping must be declared.

This is a basic configuration of the web.xml. For advanced purpose, for example, the use of file upload component, please refer to the corresponding documentation.

4.3 Mapping of Application Configuration Class

As is illustrated in Figure 3-4, in both IVY web and portal applications a subclass of *WebApplicationConfiguration* or *PortalApplicationConfiguration* is generated to retrieve application-specific configuration data and to initialize required web services. To keep the application logic and business logic of the source IVY application intact, this configuration class must be mapped to the target JSF application. BBS-S&T has supplemented the IVY Presentation, IVY Faces and icdev.jar classes as shown in Figure 4-2 to simplify the task.

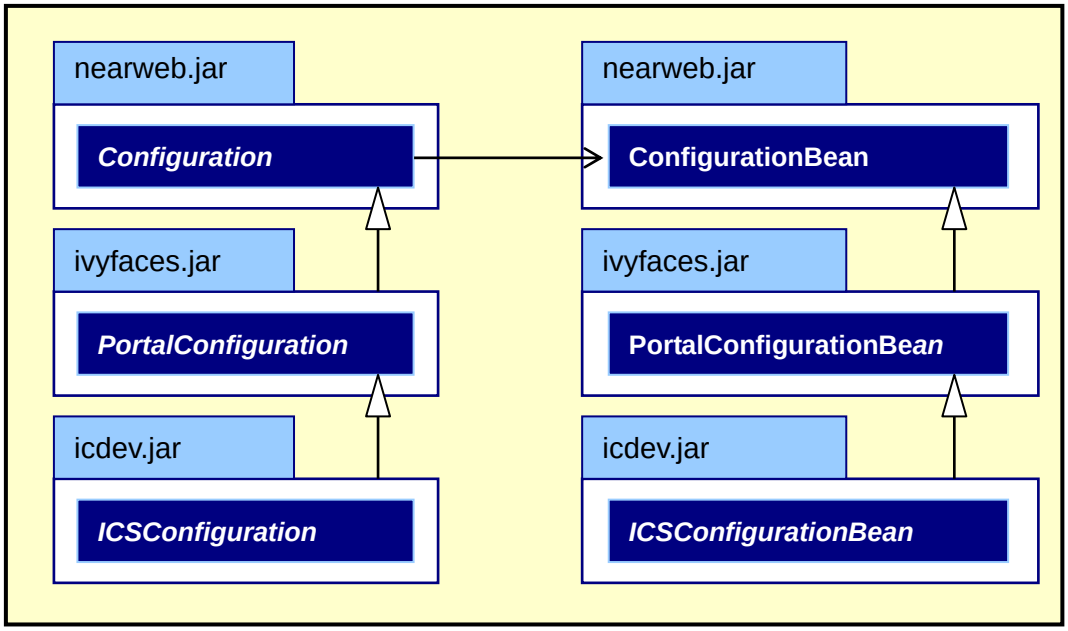


Figure 4-2: New Architecture of Configuration Class for JSF

Configuration is now equivalent to *WebApplicationConfiguration* for IVY web applications, and *PortalConfiguration*, which is a sub-class of *Configuration*,

equivalent to *PortalApplicationConfiguration* for IVY portal application. ICSConfiguration, a sub-class of PortalConfiguration, is specially implemented for ICS portal applications where the SwitchBoard service is used. What this new architecture is different from that of IVY application is the adoption of beans to store the configuration data read from the configuration.xml. As is shown in Figure 4-2 each configuration class has a matching bean class.

Therefore, as the first step to migrate the IVY configuration class, a proper sub-class of *Configuration* should be created according to the type of the source IVY application. Besides, some methods of configuration need to be overridden for the desired performance. Next, the configuration data must be mapped to the configuration.xml of the migrated JSF application by following certain format. In IVY web applications these configuration data are stored in web.xml. In IVY portal application they can be found in the configuration.template under /WEB-INF. In the last step, if the migrated application has configuration data not defined in the super bean classes, then migration engineers should generate a sub-class of the super bean class for those additional configuration data. The detailed information regarding these configuration and bean classes and the format of the configuration.xml can be seen in the document "Struktur der Configuration"[28].

4.4 Mapping of Navigation Web Control

To comply with the Bayer Standard Layout guideline, both IVY web and portal applications utilize the BayerNavigation web control. In IVY web applications, BayerNavigation is declared and initialized in *IVYWebForm* as is shown in Figure 4-3. An action listener is attached to it with the method handleNavigation, which is responsible to toggle languages and sometimes to manage the navigation flow. The portal application framework makes some improvements. It declares and initializes the BayerNavigation in *PortalForm* with the functionality to handle toggle language and show/hide navigation, so that *IVYPortalForm* can use the service directly and only overrides the handleNavigation method, when necessary. Migration engineers should have good knowledge of the usage in both frameworks, because the HandleNavigation method may define the navigation flow, which is important information for the migration to JSF. Because JSF applications must comply to the Bayer Standard Layout guideline as well, the counterparts of BayerNavigation are generated in IVY Faces with Navigation for JSF web applications and PortalNavigation for JSF portal Applications. Like BayerNavigation, these two components also use navigation.xml for configuration. Moreover, a backing bean for these two web components is created as the class *NavigationBean* in ivyfaces.jar.

[28] See Frank Danek, Andre Klaaßen (2006)

Therefore, as the first step to migrate BayerNavigation, a managed bean of *NavigationBean* must be registered in faces-config.xml with the fixed name “navigation” and the scope of request. Secondly, migration engineers should replace BayerNavigation with Navigation or PortalNavigation in each JSP of the migrated JSF application. Figure 4-3 exemplifies the migration of BayerNavigation to Navigation. To inform the JSP of the customized tags for Navigation, the URI of the tag library “http://by-near.bayer-ag.com/” must be declared. The Navigation is configured using attributes, and actually it has more attributes than shown in Figure 4-3. The actionListener attribute refers to the handleNavigation method of the managed bean called navigation. The usage of PortalNavigation control is rather similar, but with one more attribute called applicationUrl, which refers to the URL of the migrated JSF application and is used to integrate this application with others. Besides, to support the show/hide navigation functionality, another managed bean of class PortalUserBean in ivyfaces.jar must be declared in faces-config.xml with the name “user” and the scope of session.

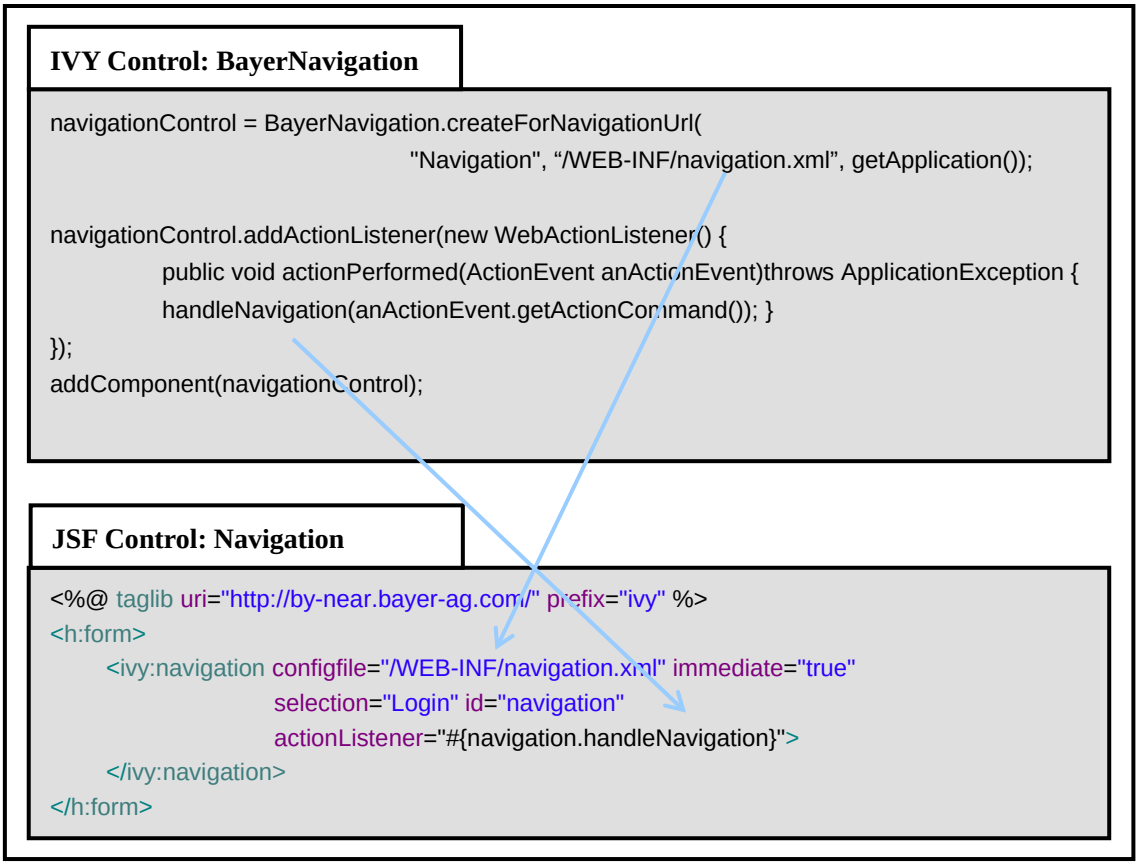


Figure 4-3: Migration of BayerNavigation to Navigation

In the last step, the navigation.xml is to be mapped to the JSF application. The navigation.xml takes the same format as in IVY application. However, some attribute

values should be changed. The detailed information can be seen in the document "Einbinden der Navigation"[29].

4.5 Mapping of Message Resources

The IVY Presentation supports internationalization by using the Java properties file, which is a file keeping key-value pairs delimited either by equal sign "=" or colon ":". Frequently the properties file contains the texts for output labels, button labels or the error messages. Inside the IVY web or portal applications the properties file has by default the identical name and location as *IVYWebApplicationConfiguration*. The functionality to retrieve the translated text according to the current locale is provided by the method getTranslatedText() of *WebApplication* with an object of *Translationkey*. The *TranslationKey* object is constructed using the key in the properties file as the input parameter. In the JSF Framework, the properties file is also applied to support internationalization.

```
<f:view locale="#{localeUtil.userLocale}">
<f:loadBundle basename="com.bayer.myapp.MyMessages" var="messages" />

Case 1: normal key
some_text = Here is a sample text in English
<h:outputText value="#{messages.some_text}" />

Case 2: key delimited by dot
some.more.text = Here is another sample text in english.
<h:outputText value="#{messages['some.more.text']}" />

Case 3: text with parameters
text_with_parameters = Hello {0}! You have {1} new mails.
<h:outputText value="#{messages. text_with_parameters}" >
<f:param value="#{user.firstName}"/>
<f:param value="#{mailbox.countNewMails}"/>
</h:outputText>

Case 4: text with html-tag
text_with_html = Please contact our <A
      HREF="mailto:STServiceDesk@BayerBBS.com">ServiceDesk</A>.
<h:outputText value="#{messages.text_with_html}" escape="false"/>
```

Figure 4-4: Multilingual Support of JSF

Therefore, as the first step to migrate the message resources, the properties files can be directly moved to the migrated JSF web applications. The recommended practice is

[29] See Frank Danek, Andre Klaaßen (2006)

to create for each desired locale a messages_locale.properties file. Migration engineers shouldn't use messages.properties for English locale, instead use messages_en.properties. The reason is that if JSF searches for the English locale, but fails to find it, it will use JVM default locale depending on the locale of the server.

Secondly, the properties file needs to be declared in the faces-config.xml. Thirdly, to use the build-in services of JSF in JSP, the properties file must be declared with the f:loadBundle tag. If the key is in different formats, the expression to use them varies as is demonstrated in Figure 4-4. To make the use of message resources convenient in the java codes, IVY Faces provides a utility class called LocaleUtil, which offers methods to retrieve the current locale and to get the translated text of the currently-applied locale.

4.6 Mapping of Business Beans and Logics

Even though there is no explicit definition in IVY Presentation, there are classes containing objects and business logics in IVY web and portal applications. Migration engineers need to identify these classes and map them into beans in JSF. There are various ways to arrange the beans in JSF web and portal applications. One approach is to create beans for each JSP to hold the values of the contained controls. The advantage is the clear one-to-one relationship between JSP and bean, while the disadvantage is the code redundancy and the dependency between the web application framework and the business logic. Although JSF is an advanced technology nowadays, it may become out-of-date in the future. In case it is to be replaced by another more advanced web application framework, the web application needs to be migrated again and the codes can't be reused. The other approach is to generate beans according to the design of business logics. The advantage is the code efficiency and the clear separation between presentation and business logics and high code reuseness. The disadvantage is that this approach requires that migration engineers have good knowledge in both software design and the business logics of the web application. However, in BBS-S&T lots of work is done by external and students of various proficiencies, so it is rather difficult to control the quality of the design if this second approach is taken. Besides, this approach isn't effective and efficient in migration projects, because business logics are distributed in *IVYWebForm* classes, and it takes migration engineers great time to get a complete view of the functionality as a whole. After the careful study of the situation in BBS-S&T and the special features of migration project, the bean architecture is designed as shown in Figure 4-5, and it is recommended to be applied as a standard.

This design separates the codes for presentation and those for business logic. In the presentation section, beans are created for each JSP and they are declared in faces-config.xml as managed bean of proper scopes. The manage beans should only have the properties to match the values of controls contained in the corresponding JSPs. The actions attached to command controls should be placed in backing beans, which can be used for single JSP or shared by several JSP to reduce code redundancy. The main responsibility of backing beans is to map the managed beans to business beans, to delegate the action to business logic and the map the return values to managed beans. Because these backing beans are combined with messages, they should be declared in the faces-config.xml with the scope request. The business section should contain well designed business object and logics.

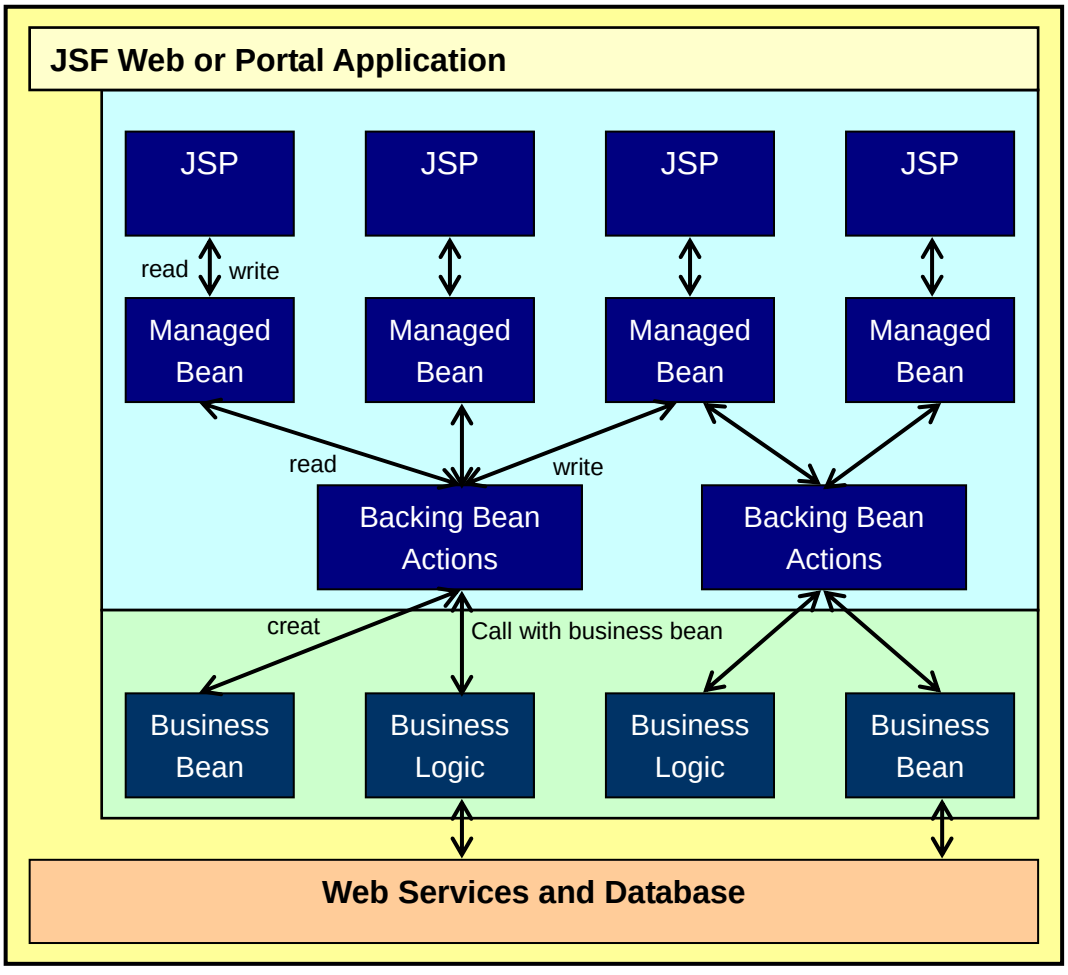


Figure 4-5: Design of Model Architecture in JSF

This approach shows many advantages. First of all, the separation between presentation and business improves the code reuses, and it is easier to learn. Besides, it allows BBS-S&T to better control the application quality by focusing on the business section. Most important is that it works well with the migration plan designed in Chapter 2. The mapping done in the analysis phase will be more effective,

because the work is mainly done for the presentation section. After migration engineers have collected sufficient knowledge, they can work on the design of business section in the design and migration phase. Therefore, the combined efficiency and effectiveness are improved.

4.7 Mapping of Web Forms

During the migration process, the mapping of web forms is the most complicated and error-prone task, and that requires great attention and carefulness. The migration of web forms are divided into three subtasks, namely the migration of their layouts, the mapping of contained web controls and the declaration of navigation rules in faces-config.xml. Through the practical projects, the procedure for the migration of web forms has been developed and can be applied in migration projects. To help the explanation Figure 4-6 takes the button control as an example to illustrate the mapping procedure.

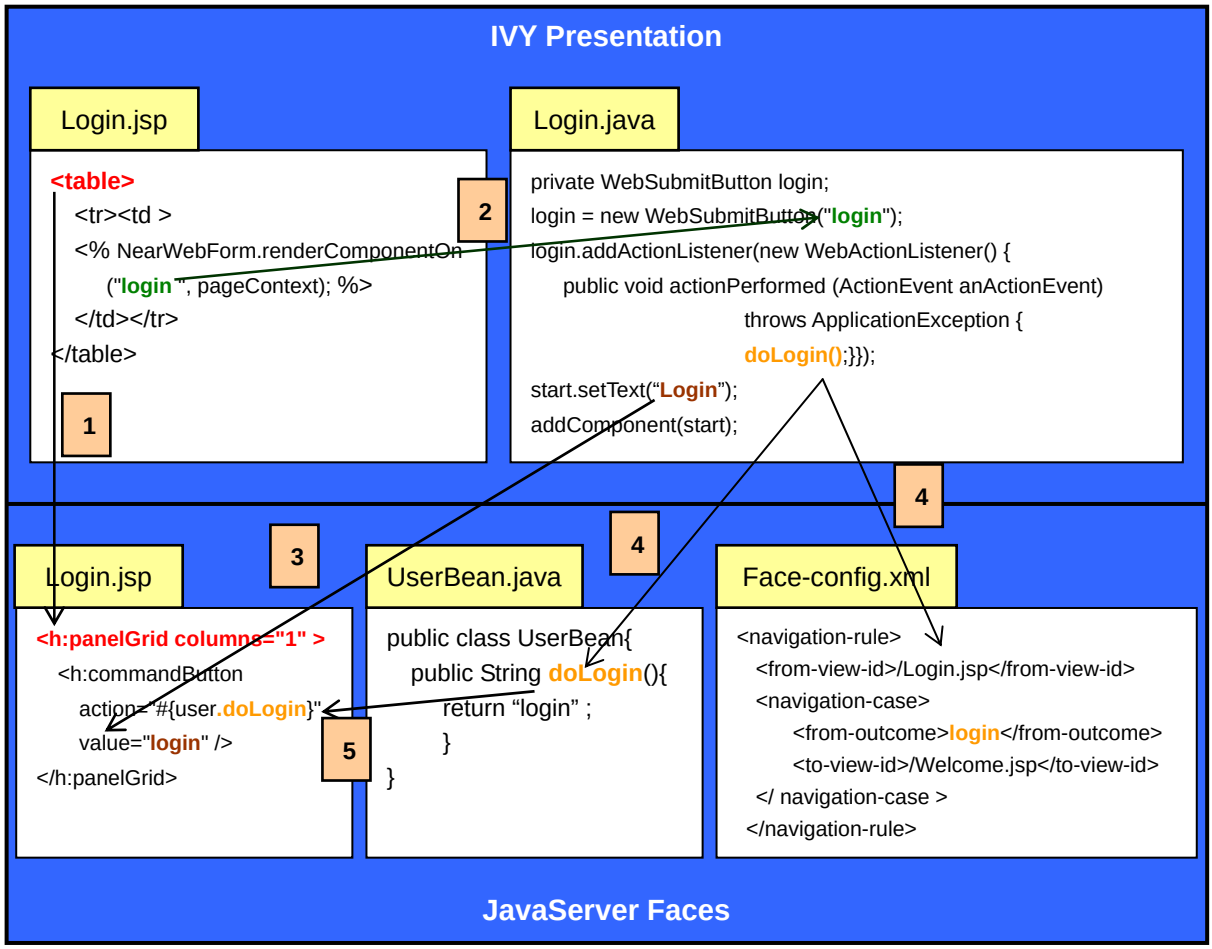


Figure 4-6: Mapping Procedures of Web Form

The mapping of layout is the first step. Even though the JSP in JavaServer Faces supports HTML tags, it is still recommended to use the JSF HTML tag library. Afterwards, the web controls contained in the JSP are mapped, so the second step is to search for the detailed information about the controls in its corresponding web form class of the IVY project. Normally web form class needs to create its own constructor and override the methods of createStaticItems() and prepareResponse(). The declaration of web controls can be made either in the constructor or the method createStaticItems(). The method prepareResponse() is used to update the values of web controls before they are rendered. For a button control the basic information needed to know is its text and action listener. Having found its text, the button control can be mapped to the JSP of the JSF project using JSF HTML tags. According to the design of model architecture in JSF as is shown in Figure 4-5, each web form in JSF has or shares one backing bean. Therefore, in step four the action listener method is moved into the backing bean and the navigation rule is declared in faces-config.xml. In the last step, the button control is bund to its backing bean by assigning the action method to its action attribute. By following this procedure, the mapping of web forms can be done in a rather organized manner.

Comparatively speaking, the migration of layout and the declaration of navigation rules can be done with ease, while the mapping of web controls is the most difficult task.

4.8 Mapping of other Items

To complete the migration project, mapping rules for validation/conversion, composite controls, security, logging, build scripts and tests are developed as well. Because the mapping rules for these items are relatively simple, they are put together in this sub-chapter.

Validation and Conversion

Because there is no direct support for validation and conversion in IVY Presentation, they are applied not so often in IVY web and portal applications. Most likely they are performed in the action listeners of web controls. Therefore, migration engineers need to identify these codes and map them to the validation and conversion services provided by JSF. They need to decide where validation and conversion are appropriate.

Composite Controls

In IVY web and portal applications there are some composite controls, which must be migrated to JSF applications. JSF provides the API to develop customized web

components. Since it is a rather complicated topic, the detailed information can be seen in the book Core JavaServer Faces[30].

Security

Especially for IVY portal applications, user authentication and authorization are required. The portal application ICSLoginJSF is used by all others for user authentication. With the purpose to simplify the security task, BBS-S&T has developed in IVY Faces the LoginListener that is an implementation of PhaseListener and the PortalService to perform authentication and authorization respectively. The detailed information for usage can be seen in the documentation "Einbinden der Login-Funktionalität"[31].

Logging

Since JSF applications also applies the logging services of IVY Foundation. The source codes for logging can be mapped directly. Besides, migration engineers need to override the getLoggerIdentifier method of Configuration in order to return the actual application name.

Build Scripts

BBS-S&T provides build scripts to facilitate the task of deployment. They can be directly applied in JSF web applications.

Tests

To test the functionality of IVY web and portal applications automatically, BBS-S&T has developed the so-called ICS ExcelTester. The test case can be written down in Microsoft Excel sheet, HTML or XML. The ExcelTester can be configured using java properties. The ExcelTest has been extended to be able to test JSF applications. Since the migrated JSF applications are expected to have the identical behaviors as the IVY applications. The ExcelTests, if available, are worthwhile to be mapped to test the JSF application.

Therefore, for the mapping purpose the interpreter of the ExcelTester must be changed to com.bayer.ics.autotester.interpreter.jsf.JSFDBWebInterpreterGermanEnglish. Besides, ExcelTester uses the name attribute to identify the IVY web controls. But for JSF web components it uses the id attribute. So in ExcelTests the values used to identify web components must be actualized. Moreover, migration engineers should make sure that the any change to the workflow of JSF application is updated in ExcelTests.

[30] David Geary, Cay Horstmann (2004)
[31] See Frank Danek, Andre Klaaßen (2006)

5 Migration Tools, Templates and Methods

In accordance with the model-driven migration approach, this project also developed some migration tools, templates and methods to facilitate the migration process. To make sure that they are useful, interviews were conducted with staff of BBS-S&T who had some experience with migration projects in order to find out their expectations and requirements. A brief summary of their requirements is presented below.

Firstly, all the interviewees prefer a well-structured and detailed migration guideline to an automatic migration tool. The reasons are the following. IVY web and portal applications vary from each other in application architecture and implementation, so it is difficult to develop an automatic tool that can deal with such variety intelligently. Besides, it is hard to identify mistakes in the automatically-generated codes. In case something is wrong, migration engineers have to check the correctness of all the codes, because there is no knowledge where the mistake can possibly be. In this way, the automatic tool doesn't simplify the task. Instead it makes migration engineer feel frustrated and increases the time and effort cost. Therefore, a clear migration guideline is more helpful, because it provides instructions, but migration engineers have a full control of the migration process.

Secondly, all the interviewees think, that a tool, which can help estimate the time duration of migration projects, is useful. With this tool, project leaders can manage migration project with more accuracy.

Thirdly, both IVY and JSF projects use lot of xml files for configuration, for example, web.xml, configuration.xml and navigation.xml. The mapping of these files is mainly labor work. Therefore, some interviewees would like to have a tool that can automatically map these files.

Last but not the least, some interviewees want a tool that can generate the navigation flow diagrams of the IVY web and portal applications automatically. The navigation flow diagram can help migration engineers to understand the business logic of IVY web and portal applications. Besides, in JSF the navigation rule needs to be declared in faces-config.xml.

To fulfill their requirements, tools that can be used to estimate project duration and to facilitate the generation of navigation rules by mapping JSP are designed and implemented. Method to create a JSF project template are introduced. A migration guideline with instruction to carry out a complete migration project is composed. They are to be introduced in this chapter in detail.

5.1 Migration Tool for Project Duration Estimation

The migration of web applications from IVY Presentation to JSF is conducted using Eclipse. For the sake of convenience, the tool used for the estimation of migration duration is designed and implemented as an Eclipse plugin[32], which is an extension to the popup menu as is shown in Figure 5-1.

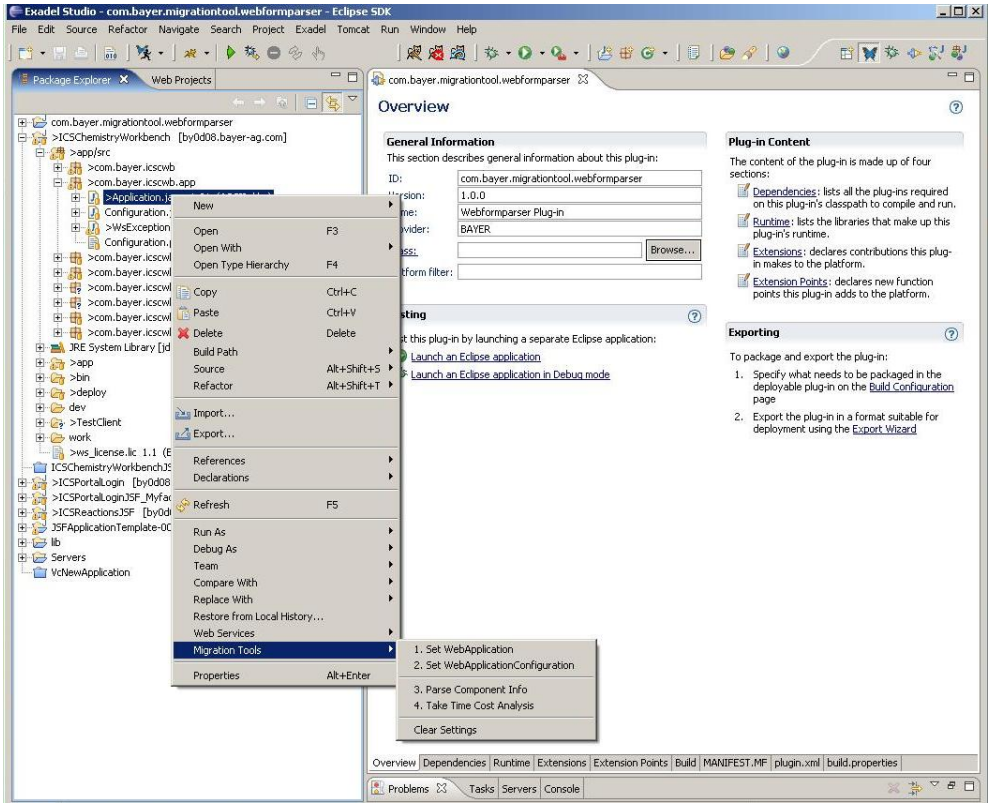


Figure 5-1: Tool for Project Duration Estimation

The mechanism to calculate the migration duration is based on the total number of web controls contained in the IVY web or portal application. If migration engineers utilize the JSF project template, which is to be introduced afterwards, the time spent on the fundamental configuration can be effectively reduced. Therefore, most of the migration time is consumed by the migration of web controls and their underlying business logics. On the other hand, the number of web controls is a value that can be quantized, and the time spent on the migration business logics can be distributed to the number of web controls. Furthermore, the class *WebForm* provides the services to retrieve the number and types of web controls contained in each web form using the method getComponents. Attributed to the above-mentioned reasons, the duration of

[32] See John Arthorne, Chris Laffra

migration projects can be derived from the number of web controls in the IVY projects.

This tool utilizes HSSF[33], which is POI project's pure java implementation of the Excel '97(-2002) file format, to output the analysis results in an Excel file. This Excel file is created under the root of the IVY web application, and consists of three sheets. Sheet one contains the table of time unit used to calculate the migration time and the analyzed result. Sheet two lists all the web controls in terms of their affiliated *WebForm* class name, the name, the class, the category, time for the phases analysis, design, migration and test respectively. With this Excel file project leaders can manipulate the data freely for the desired information. IVY projects normally have abstract subclasses of *WebForm*, so sheet three records these abstract classes.

It needs to be pointed out, that this tool needs to know the name of *IVYWebApplication* and *IVYWebApplicationConfiguration* in order to process *IVYWebForm* classes in the IVY web Applications. For IVY portal applications, a mocker class of PortalApplicationConfiguration needs to be created by simply copying a code sample. To simplify the development, the tool is designed to be available whenever a file with suffix .java is selected. Migration engineers must set *IVYWebApplication* and *IVYWebApplicationConfiguration* or the mocker class manually. Below is the instruction to use this tool.

1. Select the java file of the class *IVYWebApplication*, click on the right mouse, and then click Migration Tools → 1. Set WebApplication.
2. Only for IVY portal application, create the mocker class of *PortalApplicationConfiguration*.
3. Select the java file of the class *IVYWebApplicationConfiguration* or the mocker class, click on the right mouse, and then click Migration Tools → 2. Set WebApplicationConfiguration.
4. Select the java files of *IVYWebForm* to be processed, click Migration Tools → 3. Parse Component Info
5. When all the *IVYWebForm*s are processed, click Migration Tools → 4. Take Time cost Analysis.

According to the design, when the Time Cost Analysis is completed, all the cache is cleared. By clicking Migration Tools → Clear Settings, the cache can be cleared any time.

Table 5-1 exhibits the sheet one of the project duration estimation result. The values given in the Migration Time Cost Estimation Unit should be the approximate time

[33] See HSSF

spent on various control types during the analysis, design, implementation and test phase. Attributed to the utilization of HSSF, the cells of the Analysis of Migration Time Estimation are assigned with formula referring to the Migration Time Cost Estimation Unit. Therefore, they are updated when the values in the Migration Time Cost Estimation Unit changes.

Migration Time Cost Estimation Unit (Min):

Control Type	Analysis	Design	Implementation	Test
IVY Bayernavigation and WebBorder	0	0	3	2
IVY Output Control Types	10	3	3	10
IVY Input Control Types	10	3	3	10
IVY Button Control Types	15	3	60	3
IVY Select Control Types	10	3	10	10
IVY Composite Control Types	5	5	30	15
IVY Composite Control Types not in IVY Presentation	5	25	30	15
IVY Controls without need for test	5	0	3	0

Analysis of Migration time Estimation:

Number of Processed WebForm Classes:	83				
Type	Analysis	Design	Implementation	Test	Total
Per Project (hours)	137.67	143.78	375.67	116.27	773.4
per WebForm Class (hours)	1.66	1.73	4.53	1.4	9.32

Table 5-1: Sample of Project Duration Estimation File

The values of Migration Time Cost Estimation unit in Table 5-1 are derived from practical experience of migration projects, which are valid for migration engineers who have rather good knowledge of IVY presentation and JSF. However, there are some potential risks, which are:

- The IVY project is very complicated and migration engineers are unfamiliar with its functionality. It may take them more time to understand the navigation, business logics and class relations.
- The development of customized JSF component may take longer time.
- The development environment may be slow and takes extra time.

- If several migration engineers are doing the project together, it also takes time for communication.
- Migration engineers may not be able to work eight hours with high efficiency, some additional time is required.

On the other hand, if migration engineers are familiar with IVY projects and the development of customized JSF components is not as difficult as expected, less time is needed. Based on the above analysis, the time variation percentage due to risks can be set to 15%. In conclusion, project leader should update the time cost units to better reflect the actual situation and to improve the accuracy.

5.2 Migration Tool to Facilitate the JSP Mapping

Normally there is no document about the navigation rules available in IVY web or portal application. For the migration task, it is helpful to generate such a navigation flow diagram in order to analyze the web forms systematically. The Navigation Diagram Editor provided by Exadel can be applied for such a purpose. Migration engineers need to analyze the navigation rules of the IVY web or portal application by themselves. However, this task can be easily done by looking for the SetSuccessor method of the action listener attached to web controls such as buttons, or the handleNavigation method of the BayerNavigation control. Then they can use the Navigation Diagram Editor of Exadel to draw the diagram, which will automatically add the navigation rules into faces-config.xml. To make better use of the feature, it is recommended to use the name of next web form as the navigation flag, which is self-explaining and easier for migration engineer to remember. The only inconvenience to use this editor is that JSP must be created before they are referenced in the editor. To simplify the task, another tool shown in Figure5-2 is developed.

To use this tool, the following instruction needs to be applied.

1. Prepare one JSP in the JSF application as the template. The template needs to have a comment exactly as <!-- Content --> to indicate the tool where the content of the copied JSP from the IVY application is placed.
2. Select that JSP, click the right mouse and then click Migration Tools → Set JSP Template.
3. Select the JSP of the IVY web or portal Application, right click the mouse and then click Migration Tools → Copy IVY JavaSErver Pages.
4. Select the folder where the JSPs should be pasted, right click the mouse and then click Migration Tools → Paste IVY JavaSErver Pages.

By default, after the paste all the cache is cleared. But the command Clear Template and Copied JSPs can be used any time to clear up the cache.

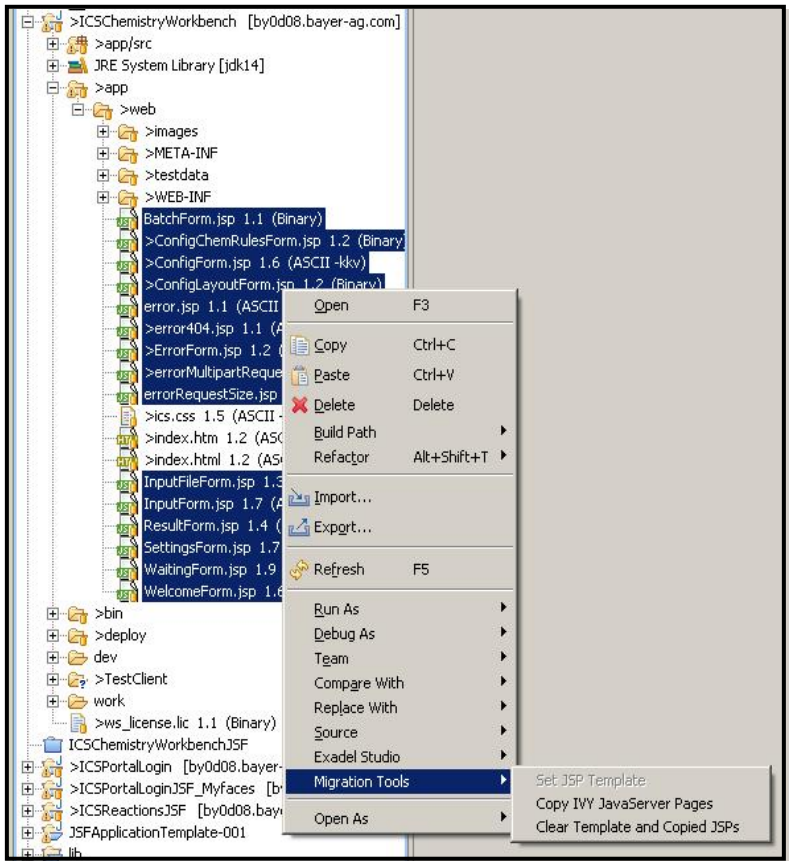


Figure 5-2: Migration Tool to Map JSP

5.3 JavaServer Faces Project Templates

The practical experience obtained from the migration project of the portal application called ICSChemistryWorkbench has proven that the migration process can be accelerated by the use of a JSF project template. Migration engineers can generate the project template on their own, so that they can decide with great flexibility the functionality, which the project template should provide. To create the project template, migration engineers need to have some knowledge of the standard project structure and the recommended package structure of the source codes. The knowledge is also useful for them to create the JSF project from scratch, if they decide not to use project template. The standard project structure is outlined in Figure 5-3, where the JSFSampleApp is used for an example.

The folders of the project are introduced in the order that they are listed in Figure 5-3. The source codes are placed in the JavaSource, and its package structure is illustrated in Figure 5-4. To examine the quality of the migrated application, BBS-S&T requires that the following test be performed, namely JUnit test for business logics, ExcelTest for the correct performance, Emma for the code coverage, FindBugs for the quality of codes, CheckStyles for the quality of code style and Metrics for the evaluation of the

codes. These test files are stored in the Test folder. The Ant folder contains the build scripts and the required libraries. The Docs folder keeps the JavaDoc, the project documentations and the presentation PowerPoint. The WebContent folder is the root of the web application, which contains the images subfolder, the WEB-INF subfolder and JSP files. The WEB-INF subfolder has a subfolder called lib to store the libraries referenced by the source codes, and various configuration files, namely configuration.xml, faces-config.xml, navigation.xml and web.xml. The work folder is the working directory.

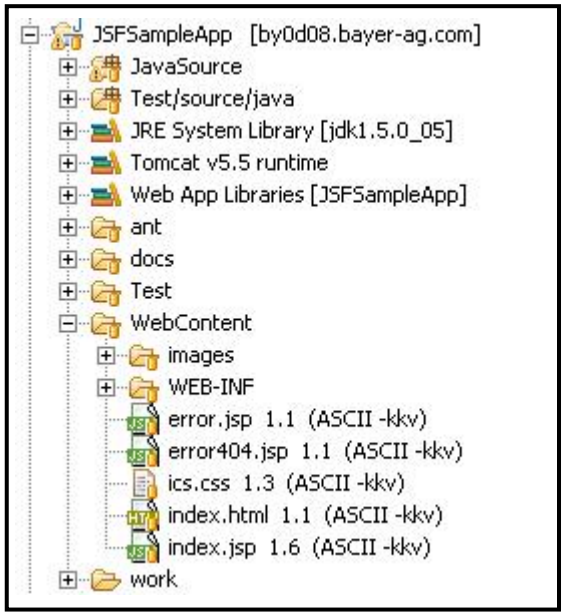


Figure 5-3: Project Structure of JSF Application

With the purpose to achieve a clean separation of JSF-related codes and business logic codes, the package structure inside the JavaSource folder is recommended to follow the standard illustrated in Figure 5-4. The JSF-oriented source codes should be placed in the package of “com.bayer.applicationname.jsf” and the source codes related to business logics in the package of “com.bayer.applicationname.logic”. Besides, for JSF-related codes it is suggested to categorize the codes for application configuration, beans, action listeners, value change listeners, phase listener, message bundles, validators and convertors, and put them into separate sub-packages. By following this practice, the project is ensured to have a clear structure, and it is easier to identify the functionality of each package. This practice will also bring benefits for the maintenance in the future.

Since all the migrated web and portal applications should comply with the standard project structure, it is a good motivation to utilize project template. Besides, web and portal applications share quite a lot of other commonalities. For example, in the JavaSource folder they all need the code for version, application configuration,

message bundles, and navigation listener. For portal applications, they need in addition codes for the authentication. In the Test folder, they should all have the libraries for Emma and Exceltester. In the ant folder, they need to have the libraries and the build scripts. In the docs folder, they need to have the standard project document templates. In the WebContent folder, the images stored in the images subfolder are standard. In the lib subfolder they all need the basic libraries for IVY Framework and MyFaces. Since the configuration files take standard formats, migration engineers can create templates for them. Last but not the least, they should all have the JSP files for error pages and the stylesheet file ics.css is also standard.

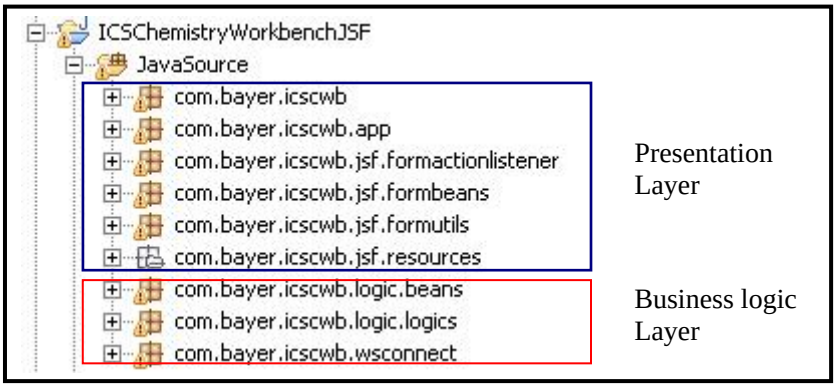


Figure 5-4: Package Structure of JavaSource

In conclusion, it is worthwhile for migration engineers to invest some time to create their own project template to speed up the migration process. They can be flexible to design the project template in their own preferred way with desired functionality. This project can be registered with Exadel of the Eclipse and applied for future projects. However, it should be pointed out that normally some values in the project templates needs to be updated to match the specific project. It is recommended for the migration engineers to make a checklist for their project templates, so that the fine tuning task can be performed correctly and systematically.

5.4 Migration Guideline

After the study of IVY Presentation and JSF, three migration projects of various scales have been carried with the purpose to accumulate practical experience of the migration process and to complement the comprehension of the two frameworks and the generation of the mapping rules. First of all, four migration principles have been derived from the practical experience. Migration engineers can apply them flexibly and develop their own principles.

Principle 1: Do the migration in a systematic approach

Migration project is complicated and error-prone, because the migration engineers may lack the sufficient knowledge of the source application, and on the other hand, the numerous details of the migration project can be easily overlooked by migration engineers. Therefore, it is of great importance for migration engineers to handle the migration project in a systematic approach. Only in this way will the potential mistakes be avoided and will the migration process proceed efficiently. An optimized order to perform the migration projects will be introduced afterwards. Migration engineers can try it out and make improvement.

Principle 2: Cope with the complexity through subdivision

This is a common approach to deal with complexity and hence is applicable to the migration projects as well, especially for large migration projects. The complicated migration project is difficult to be managed as a whole. In consequence, it must be divided into several small actionable migration projects. The navigation rules can be good criteria for migration engineers to divide the migration tasks and for project manager to allocate migration tasks. Therefore, it is helpful to generate the navigation rules before the execution of the migration projects, so that migration engineers can master an accurate knowledge of the project complexity.

Principle 3: Make small progress

It is optimal to set the migration of each web form as a task unit. The web application is a coherent work flow of several web forms, which is heavily dependent on the output of the previous web form. Therefore, the aim should set to make small progress and make each web form run correctly before moving to work on the next web form.

Principle 4: Do test when appropriate

With each web form as one task unit, it is recommended to perform tests for each migrated web form, so that any missing information required for tests can be added

immediately. ExcelTest should be performed for functionalities that can be tested automatically. Otherwise, descriptive test cases need to be documented, so that migration engineers can perform the test manually and systematically.

Generally speaking, the above-mentioned principles can be applied to manage all migration projects. To execute individual migration project, the following instructions are derived from the practical experience, and are supplied as reference to migration engineers. The list stated below shows the suggested order to lead the migration process and it corresponds to the migration plan presented in Chapter 2.

1. Get the new JSF web application running with the HelloWorld web form
2. Map the JSP files and generate the navigation rules
3. Map the message bundles and the source codes for business logics
4. Map the web forms and create corresponding managed bean and action listeners.
5. Redesign the application architecture to separate the presentation and logic layer.
6. Complete the migration of each web form and perform test
7. Test the migrated web application as a whole and deployment

In the Analysis/Mapping phase, the target JSF project is to be first generated in Eclipse either using the project template or creating a blank project. The task is to get this new JSF application running in the local development environment with configuration, navigation and authentication, if appropriate. In case the project template is applied, migration engineers need to update the template according to their checklist with data retrieved from the source IVY project. If the migration engineers decide to start with a blank project, they need to create the project with the structure as shown in Figure 5-3 and add all the required source codes and configuration files. BBS-S&T has now detailed documentation available to show the standard way of how the configuration, user authentication, and the navigation are implemented. At last, the JSF project must be able to run on the local Tomcat with a simple JSP to show “HelloWorld” or similar. Moreover, the show navigation, the toggle language and login/logout buttons on the support navigation list of the web page should function correctly.

The HelloWorld JSP can be used as a template to map all the JSP files from the source IVY project to the target JSF project. For this purpose the migration tool for the JSP mapping introduced in Chapter 5.2 can be utilized. Then the next task is to generate the navigation rules using the Navigation Diagram Editor of Exadel in the target JSF project. To help migration engineers remember of the navigation rules, it is suggested to use the name of the JSP files as the outcome of actions. The navigation

rules can be derived from the source IVY project either through the setSuccessor method of the action listeners linked to the command controls, or the handleNavigation method of the action listener linked to the BayerNavigation control.

Afterwards, the massage bundles can be mapped directly to the target JSF project. Because BBS-S&T doesn't recommend to use dot "." in the key name, any violation in the properties files needs to be modified to meet the requirement. The business beans and logics in the source IVY project can also be mapped to the target project for the time being. In the design phase, the application architecture needs to be redesigned in order to achieve a clean separation of the presentation layer and the business logic layer. Therefore, some changes to the business beans and logics are necessary to be made.

Next migration engineers can concentrate on the comprehension of the subclasses of WebForm in the source IVY project and, meanwhile, map the web controls into the JSP of the target JSF project using the JSF core and HTML tags. Besides, the corresponding JavaBeans for component values and backing beans for action listeners should be created for each JSP as described in the mapping rule of business beans and logics. These beans need to be registered in faces-config.xml. Normally the JavaBeans should have the scope of session, while the beans for action listeners have the scope of request.

With the purpose to facilitate the migration of web controls, a mapping method has been designed, which performs the migration in three steps. The first step is to determine the type of the web control. The second step is to find out the related information of the web control. The third step is to map the web control using JSF tags. The web controls are categorized to five types, namely output, input, select, command, and composite controls. The mapping of these five kinds of web controls is explained below.

Mapping of the Type Output

The functionality of this kind of web controls is to show their values on the web page as strings. They can be mapped using JSF tags <h:outputText> or <h:outputArea>. Their values can be set either by the message bundles or the binding managed beans. If the value source is message bundles, the translation keys must be identified and set to the value attribute of the JSF tag. If the value source is managed beans, the value binding needs to be used for the value attribute. Besides, it is necessary to find out if any converter should be applied to this output control and in which condition the web control is rendered.

Mapping of the Type Input

The functionality of this kind of web controls is to get user inputs, store them in the corresponding managed bean. Normally their values are used as input parameters for the action listeners triggered by web controls of type command. They can be mapped using JSF tags <h:inputText>, <h:inputArea>, <h:inputSecret> for secret inputs and <h:inputHidden> for hidden values. Therefore, they must be bund to the matching managed bean. Besides, migration engineers need to find out if any validator or converter is required for these web controls, before their corresponding managed beans need to be updated. At this point a good knowledge of the JSF life-cycle is helpful. They should also look for information about the condition of rendering. Besides, the input web controls can be bund to the value change listener as well, in case the web form needs to be updated when the input value changes.

Mapping of the Type Select

The functionality of this kind of web controls is also to get user inputs, which are used as input parameters for the action listeners triggered by web controls of type command. Therefore, migration engineers need to identify the available select items and their values must be bund to the matching managed beans as well. Besides, they need to find out their default values and in which condition they should be rendered. Besides, the select web controls can be bund to the value change listener, in case the web form needs to be updated when the selected value changes.

Mapping of the Type Command

The functionality of this kind of web controls is to trigger the attached action listeners, which contains the business logics to process users' inputs and determine the navigation flow dynamically. The action listeners of the command web controls in the IVY project need to be moved into the backing beans of the JSF project. These methods are attached to the action listener attributes of the JSF command web controls using the method binding. The method setSuccessor of the IVY action listeners must be removed, since in the JSF project the return values of the actions are used to determine the navigation flow dynamically. In the IVY projects, the action listeners may also contain codes to perform the validation and conversion. Therefore, in mapping the command web controls, another task is to extract these codes and place them into the proper validator or converter classes, so that users' inputs can be processed earlier before the action listeners are activated. Besides, in IVY projects, some business logics may be contained in the constructor of the WebForm. They should be moved into the corresponding actions of the backing beans in the JSF projects.

Mapping of the Type Composite

IVY Presentation provides a set of composite web controls. To simplify the mapping of composite controls, BBS-S&T has developed some JSF customized controls for navigation, chemipro, etc. Besides, Apache MyFaces has also provided good supports for composite web components and some customized validators. The detailed information can be found in http://myfaces.apache.org/tomahawk/overview.html. If migration engineers need to develop customized controls to meet their specific requirements, detailed instruction to build customized components can be seen in the book Core JavaServer Faces.

Having completed the mapping of the web forms, migration engineers should have acquired good understanding of the source IVY project and hence are able to redesign the application architecture in the design phase, so that the migrated JSF application has a clean separation of the presentation layer and the business logic layer and shows less dependency on the JSF framework. The business logics extracted from the action listeners need to be integrated to the originally existing business logics. Besides, migration engineers need to redesign the business beans, when appropriate. Moreover, the JUnit[34] tests can be performed to examine the functionality of the business logic layer. In the end the application architecture of the migrated JSF project should comply with the design illustrated in Figure 4-5.

In the migration phase, migration engineers can concentrate on the migration of each web form based on the well-designed application architecture. It is important to perform tests for each migrated web form.

After the whole migration project is completed, the new application should be tested as a whole. And at this stage, the other tests, such as CheckStyle, FindBugs and Metrics can be performed as well. When the application is evaluated to have met the requirements, it can be deployed on the test server. End-users should be informed to test the functionality of the web application. If there is any requirement for improvements, the requests should be fulfilled. If everything is fine, the application can be deployed on the production server and the IVY application will be retired.

In conclusion, the migration tools implemented to facilitate the migration process have been presented first in this chapter with instructions for usage. Because the project template can really accelerate the migration process, migration engineer can create their own project template with flexibility. In the end, the migration guideline, which is derived from experience of practical migration projects, has been introduced. Migration engineers can try it out and make further modification or improvement.

[34] See JUnit Cookbook

6 Evaluation of Migration Methodologies

6.1 Verification with Migration Projects

The migration plan, mapping rules and the migration tools, templates and methods, which are introduced in previous chapters, are derived not only from the comprehension of IVY Presentation and JSF, but also from the first-hand practical experience acquired from three migration projects.

At the beginning of the master project, a portal application called ICSPortalLogin and a web application called CDRom were migrated into JSF. ICSPortalLogin is a rather simple application with only one web form. However, it is a very important one, because it is used by all other IVY portal applications for authentication. Compared to ICSPortalLogin, CDRom is a little bit more complicated with four web forms. These two migration projects are mainly carried out to facilitate the comprehension of IVY Presentation and JSF, to develop the mapping rules and to find out what kinds of tools, templates and methods are helpful. Based on the experience from these two projects and the research on the good practices from the internet resources, the migration plan, mapping rules and the migration tools are developed. To evaluate their feasibility, another portal application called ICSChemistryWorkbench is migrated to JSF. It is composed of eight web forms and is more complicated than the previous two projects. Besides, as its name indicates, it deals with chemical objects, so this project is a good supplement to the comprehension of IVY Chemistry model. Through the migration of the three applications, most features of IVY Framework have been practiced.

Attributed to the developed migration plan, the mapping rules and the migration tools, templates and methods, the migration of ICSChemistryWorkbench turns out to be much easier and more successful. First of all, the migration tool enables the estimation of the project duration more accurately. Guided by the migration plan, migration engineer is equipped with good knowledge of the task description and the execution order. The mapping rules provide migration engineer with a clear instruction of how the migration is realized. The migration templates relieve the migration engineer from the tedious common tasks, so that the working efficiency is improved. The migration guideline provides migration engineer with integrated instruction to make good use of the migration plan, mapping rues and the migration tools and template.

In conclusion, the practical experience has proven the feasibility of the migration plan, the mapping rules, the migration tools and the migration project template. With these supports migration tasks can be carried out with better organization and finished on time and with more satisfactory quality and higher efficiency.

6.2 Performance Test on IVY and JSF Applications

In order to compare the performance of web applications implemented with IVY Presentation and JSF, the tests have been performed using Apache JMeter[35] on both IVYChemistryWorkbench and IVYChemistryWorkbenchJSF. Two groups of tests have been carried out with ExcelTest files of various complexities. Both tests are conducted under the same conditions of 10 threads and 100 repeating times. The interval between threads is set to 300 ms in order to simulate the real situation. The first group of test utilizes an ExcelTest files, which calls for the first two web pages of the portal application. The test results are illustrated in Figure 6-1.

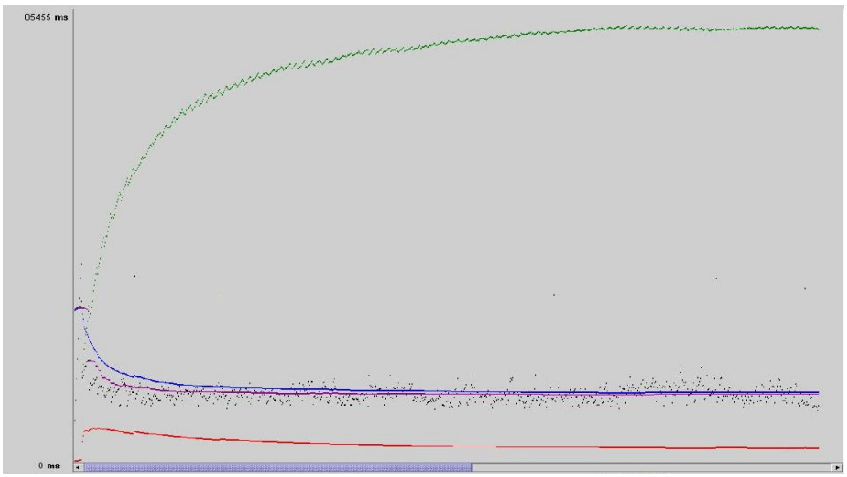
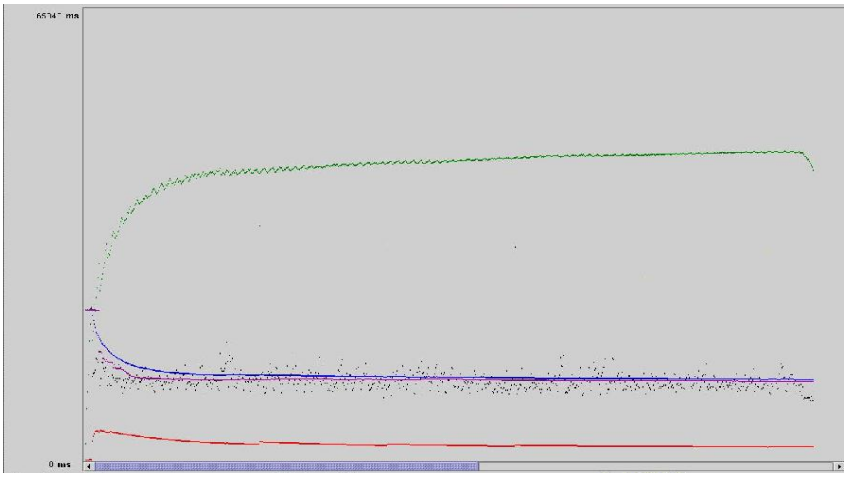
ICSChemistryWorkbench	
	Throughput: 43.136494/minute Average: 1310 ms Median: 1274 ms Deviation: 258 ms
ICSChemistryWorkbenchJSF	
	Throughput: 46.33437/minute Average: 1173 ms Median: 1146 ms Deviation: 204 ms

Figure 6-1: Result Comparison of Performance Test A

[35] See Apache JMeter

In the result view, the green line stands for the throughput, the blue line for average processing time, the purple line for median processing time and the read line for standard deviation. The second group of test utilizes a more complicated ExcelTest file, which calls for the first three web pages of application. The test results are illustrated in Figure 6-2.

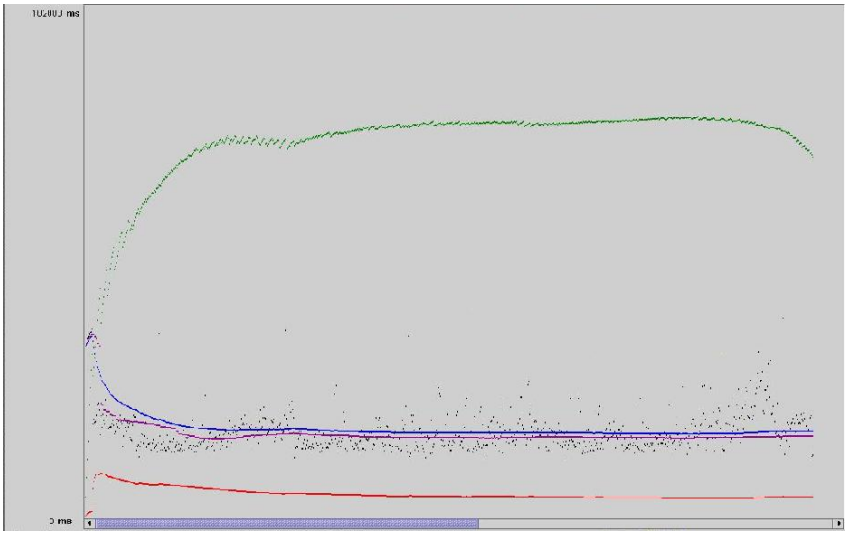
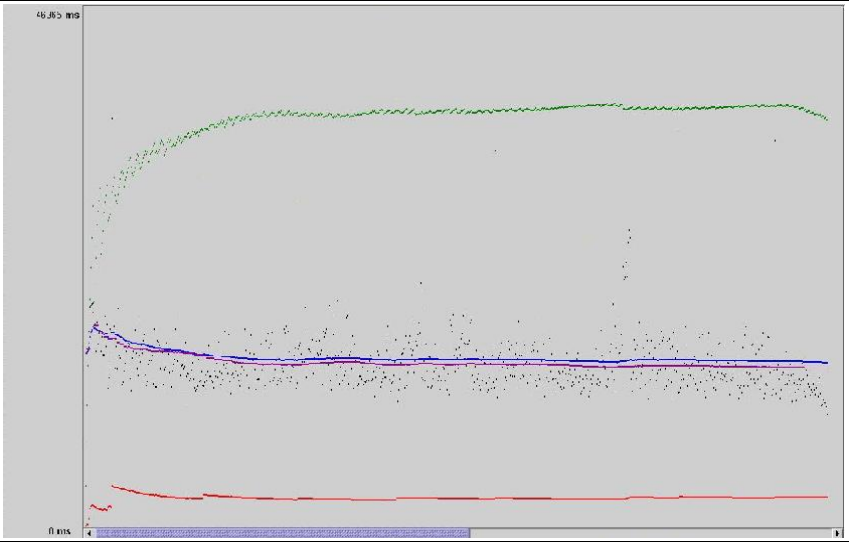
ICSChemistryWorkbench	
	Throughput: 31.539656/minute Average: 1709 ms Median: 1614 ms Deviation: 401 ms
ICSChemistryWorkbenchJSF	
	Throughput: 37.597637/minute Average: 1485 ms Median: 1437 ms Deviation: 267 ms

Figure 6-2: Result Comparison of Performance Test B

It can be observed that in the first group of test, ICSChemistryWorkbenchJSF has a higher throughput than ICSChemistryWorkbench by 7.41%, while in the second group of test by 19.21%. Besides, ICSChemistryWorkbenchJSF has a smaller

standard deviation compared to ICSChemistryWorkbench, which indicates that JSF application has a more stable performance than the IVY application. The trends of the throughput lines can also confirm this conclusion, since the throughput of ICSChemistryWorkbenchJSF increases quickly to high values and remains stable. Through the comparison of the two pairs of test results, it can be concluded that the application built upon JSF demonstrates a better performance than the one upon IVY Presentation.

6.3 Code-Evaluation Test on IVY and JSF Applications

Besides the performance test, the two applications have been also examined in terms of the code quality using Metrics[36]. The comparison result is listed in Table 6-1.

Metrics	IVY	JSF
Method Lines of Code	1830	1391
Number of Packages	7	8
Number of Classes	29	26
Number of Attributes	118	67
Number of Methods	203	212
Number of Interfaces	0	0
Number of Children	14	0
Number of Static Attributes	48	46
Number of Static Methods	12	11
Number of Overridden Methods	7	2
Depth of Inheritance Tree	3.067	1.385
McCab Cyclomatic Complexity	2.228	2.076
Weighted Methods per Class	479	463
Instability	0.45	0.442
Lack of Cohesion of Methods	0.387	0.306
Abstractness	0.008	0
Afferent Coupling	5.571	3.25
Efferent Coupling	4	2.75

Table 6-1: Comparison of Metrics Test Results

The analysis of the test results is focus on the following indexes, Method Lines of Code (MLC), Depth of Inheritance Tree (DIT), McCab Cyclomatic Complexity,

[36] See Metrics 1.3.6 – Getting Started

Weighted Methods per Class (WMC), Lack of Cohesion of Methods (LCM), Afferent Coupling (Ca), Efferent Coupling (Ce), and Instability[37].

Method Lines of Codes shows the total number of method lines without including the comments and blank lines. A small value for MLC is preferred, since the large value indicates that the development and maintenance are more expensive and that the application is more complicated and error-prone. MLC of the JSF implementation is 1391, a reduction of 24% compared to the IVY implementation with 1830.

Depth of Inheritance Tree shows the distance from class Object in the inheritance hierarchy. A small value is preferred for less dependency. DIT of the JSF implementation is 1.385, a reduction of 55% compared to the IVY implementation of 3.067. This value truly reflects that the IVY application adopts the inheritance more than the JSF application.

McCab Cyclomatic Complexity counts the number of flows through a piece of code. Each time a branch occurs, such as if, for, while, do, case, catch, the ?: ternary operator, as well as the && and || conditional logic operators in expressions, this metric is incremented by one. Weighted Methods per Class is sum of the McCabe Cyclomatic Complexity for all methods in a class. Small values for these two indexes are preferred, because large values indicate that the work flow is inefficient and time-consuming and that the application is more complicated and error-prone. McCab Cyclomatic Complexity and WMC of the JSF implementation have a slightly better performance of 2076 and 463, an improvement of 7% and 3.3% compared to the IVY implementation of 2228 and 479.

Lack of Cohesion of Methods measures the Cohesiveness of a class. A low value is preferred and a value close to 1 indicates the lack of cohesion and suggests the class should better be split into a number of subclasses. LCM of the JSF implementation is 0.306, an improvement of 21% compared to IVY implementation of 0.387.

Afferent Coupling shows the number of classes outside a package that depend on classes inside the package, while Efferent Coupling shows the number of classes inside a package that depend on classes outside the package. The lower values are preferred and higher values indicate greater dependency. The Ca and Ce of JSF implementation are 3.25 and 2.75, an improvement of 42% and 71% compared to those of IVY implementation of 5.571 and 4.Instability (I) is calculated using the formula Ce / (Ca + Ce). I of JSF implementation is 0.442, an improvement of 18% compared to that of IVY implementation of 0.45.

In conclusion, the JSF implementation demonstrates a better performance and quality.

[37] See Metric 1.3.6 – Getting Started

7 Conclusion and Outlook

7.1 Conclusion

The vision of the master project to develop migration methodologies for web application from IVY Presentation to JSF is accomplished with a complete package of migration strategy, migration plan, mapping rules, migration tool, migration template and migration guidelines.

The four tasks have been completed in accordance with the task description of Chapter 1.1. The research on the existing migration strategies and technology is conducted. Based on the research, the influence of migration on BBS-S&T is analyzed and the source code porting method of rehosting is selected as the migration strategy. The model-driven migration approach is applied to guide the master project. Through the experience learned from the Department of Defence, U.S. and Oracle, the migration plan is formulated to manage the migration projects of BBS-S&T. To ensure the feasibility of the migration methodologies developed by this master project, an in-depth comparison between IVY Presentation and JSF has been made and two migration projects are carried out to facilitate the comprehension of the two framework. Based on both theoretical knowledge and practical experience, the mapping rules, migration tools and templates are developed. The migration guideline provides an integrated instruction to combine the migration plan, mapping rules and migration tools and templates for the implementation of one migration project. At last the migration methodologies are tested with the migration project of ICSChemistryWorkbench. This evaluation project proves the feasibility and usefulness of the migration methodologies, since ICSChemistryWorkbench is migrated with better quality and higher efficiency. Besides, the proposed migration methodologies have been acknowledged by other migration engineers at BBS-S&T and begin to be utilized in migration projects.

7.2 Outlook

The tests results presented in Chapter 6 have confirmed that JSF implementation of ICSChemistryWorkbench excels the IVY implementation in both application performance and code quality. Therefore, it is a wise decision to replace IVY Presentation with JSF and to migrate the existing web applications into JSF. The migration of web application from IVY Presentation to JSF is a rather complicated topic with the involvement of tremendous details and varieties. Due to the time constrain, this master project attaches the focus mainly on the most fundamental and major issues, while some details may be left out of the scope of this paper. However, these details are also critical for the efficiency and quality of the migration projects. Migration engineers need to accumulate these crucial details and store them into the knowledge base, so that they can be accessed by other people as well.

As is mentioned in Chapter 2, the migration is a dynamically evolving process, which needs to be improved and updated with continuous efforts. Therefore, the migration methodologies put forward in this master thesis are serving as a starting point of the iterative process, and as a basis for the future development. Therefore, they are open for further improvement and enhancement. The migration methodologies which are derived from the practical projects are more suitable for small and mid-scale migration projects. So they need to be improved to be able to manage more complicated migration projects. Besides, Migration engineers, who have their own style for working, can make good use of these migration methodologies and develop their own methodologies.

Furthermore, JSF is growing to be a more mature and powerful web development framework, and more vendor and industrial supports will be available in the market. Besides, BBS-S&T is also taking great effort to develop IVY Faces. Therefore, it can be foreseen that the migration projects can be accomplished with greater ease in the future. Consequently the migration methodologies presented in this master thesis will need to be changed and updated.

References

A: Books and Articles

[1] Migration Strategies:
Migrating to Solaris Operating System
Sun Microsystems, Inc.

[2] Patrick DJ Kulandaisamy(2004): Patrick DJ Kulandaisamy
Model Driven Legacy Migration
Infosys Technologies Limited, Bangalore, India

[3] Bergey, O'Brien, Smith(2002): John Bergey, Liam O'Brien, Dennis Smith
An Application of an Interative Approach to DoD Software Migration Planning
CMU/SEI-2002-TN-027

[4] David Geary, Cay Horstmann (2004): David Geary, Cay Horstmann
Core JavaServer Faces
ISBN 0-13-146305-5, Copyright 2004 Sun Microsystems, Inc.

[5] Ed Burns: Ed Burns
About Faces: The JavaServer™ Faces API and how it relates to Struts
Sun Microsystems, Inc.

[6] Jim Keogh, James Edward (2002)
J2EE – The Complete Reference
New York, NY [u.a.] : McGraw-Hill/Osborne, c2002, ISBN: 0-07-222472-X

[7] Bill Dudney (2004): Bill Dudney
Building JSF Applications – Architecting JSF Based Web Applications for the Real World
Object System Group

[8] O'Hara, Kausmeyer, Zhang: Jim O'Hara, Ed Kausmeyer, Jingming Zhang
JavaServer Faces
MyFaces

[9] Jeff Swisher: Jeff Swisher
Introduction to JavaServer Faces
Dunn Solutions Group

[10] Erich Gamma, Kent Beck: Erich Gamma, Kent Beck
Contributing to Eclipse: Principles, Patterns, and Plug-Ins
ISBN 0-321-20575-8, Copyright © 2004 by Pearson Education, Inc.

[11] Ludwin Poertzgen: Ludwin Poertzgen
Legacy Application Modernization

[12] John Arthorne, Chris Laffra: John Arthorne, Chris Laffra
Official Eclipse 3.0 Faqs
ISBN: 0321268385 Addison-Wesley Professional; (July 2, 2004)

[13] Legacy Systems Migration: Bing Wu, Deirdre Lawless, Jesus Bisbal, Jane Grimson, Vincent Wade D O'Sullivan[1], Ray Richardson
Legacy Systems Migration - A Method and its Tool-kit Framework
Trinity College, Broadcom Éireann Research, Dublin, Ireland.

[14] Deepak Goyal, Vikas Varma: Deepak Goyal, Vikas Varma
Introduction to Java Server Faces(JSF)
Sun Microsystems

[15] Ronald Brill (2004): Ronald Brill
NearWeb – Bayer Corporate Design Support
BBS-IM-RDS

[16] Ronald Brill (2004): Ronald Brill
NearWeb Overview
BBS-S&T, 2004-02-10

[17] Ronald Brill, Christian Zander (2005): Ronald Brill, Christian Zander
JavaServer Faces vs. IVY Presentation – Ein Vergleich
BBS-S&T

[18] Kito D. Mann (2005): Kito D. Mann
From Struts to JavaServer Faces - Evolving Your Web Applications to Support the New Standard
Kito D. Mann, editor-in-chief, JSF Central May, 2005

[19] Deepak Goyal, Vikas Varma: Deepak Goyal, Vikas Varma
Introduction to Java Server Faces(JSF)
Sun Microsystems, Inc.

[20] Frank Danek, Andre Klaaßen(2006)
Sturktur der Configuration
BBS-S&T

[21] Frank Danek, Andre Klaaßen(2006)
Einbinden der Navigation
BBS-S&T

[22] Frank Danek, Andre Klaaßen(2006)
Binbinden der Login-Funktionalitaet
BBS-S&T

B: Web Resources

[23] Library Dependencies
http://wiki.apache.org/myfaces/Library_dependencies

[24] J2EE 1.4 Tutorial: The J2EE 1.4 Tutorial
http://java.sun.com/j2ee/1.4/docs/tutorial/doc/
Sun Microsystems

[25] HSSF: POI-HSSF - Java API To Access Microsoft Excel Format Files
http://jakarta.apache.org/poi/hssf/index.html , 09.03.2006

[26] Oracle Relational Migration Map
http://www.oracle.com/technology/tech/migration/maps/index.html ,
09.03.2006

[27] EMMA: a free java code coverage tool
http://emma.sourceforge.net/index.html, 09.03.2006

[28] Core J2EE Patterns – Front Controller,
http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html
08.03.2006

[29] JUnit Cookbook,
http://junit.sourceforge.net/doc/cookbook/cookbook.htm, 25.08.2005

[30] IVY Presentation Documentation
http://by-near.bayer-ag.com/near/products/Products_javaLib_web_de.jsp
09.03.2006

[31] IVY Foundation Documentation
http://by-near.bayer-ag.com/near/products/Products_javaLib_base.jsp
09.03.2006

[32] IVY Chemistry Documentation
http://by-near.bayer-ag.com/near/products/Products_baycof_de.jsp
09.03.2006

[33] HTML Tutorial
http://www.w3schools.com/html/

[34] Bayer Corporate Design Guidelines
http://www.corporatedesign.bayer.com/ , 09.03.2006

[35] Model-View-Control
http://java.sun.com/blueprints/patterns/MVC.html, 09.03.2006

[36] JSR 127: JavaServer Faces
http://www.jcp.org/en/jsr/detail?id=127, 09.03.2006
Java Community Process

[37] Metrics 1.3.6 – Getting Started
http://metrics.sourceforge.net/ Stand: 09.03.2006

[38] Apache JMeter
http://jakarta.apache.org/jmeter/, 09.03.2006

[39] Exadel Tutorials and Demos
http://www.exadel.com/web/portal/products/Tutorials, 09.03.2006