

Aalto University  
School of Science  
Degree programme of Computer Science and Engineering

Toni Sallanmaa

# Adapting existing web applications for use in portals

Master's Thesis  
Espoo, January 19th, 2011  
Printing date: January 19th, 2011

Supervisor: Professor Tuomas Aura, Aalto University  
Instructor: Matti Hämäläinen M.Sc. (Tech.), Analyse Systems Finland Oy

<b>Author:</b>	Toni Sallanmaa	
<b>Title of thesis:</b>	Adapting existing web applications for use in portals	
<b>Date:</b>	January 19th, 2011	<b>Pages:</b> 13 + 65
<b>Professorship:</b>	Data Communications Software	<b>Code:</b> T-110
<b>Supervisor:</b>	Professor Tuomas Aura	
<b>Instructor(s):</b>	Matti Hämäläinen M.Sc. (Tech.)	
<p>It is of importance to be able to integrate every application in use to a portal system when deploying Enterprise Information Portals in a corporation or any other party. Integrating applications to a portal will offer added value for the user, for example in the form of user interface integration and security enhancements. This paper describes services offered by portal platforms and surveys various portal platforms from leading vendors. It was found that there are no major differences in services provided. Differences are minor between the portal platforms sharing the same technology base. Adherence to standards, feature emphasis and security features were found to be the most prominent differences.</p> <p>This paper examines multiple methods for integrating custom-built, commercial or legacy applications to a chosen portal platform. Method selection is found to rely heavily on the type of application, the architecture of the application and the portal platform in use. It was found that all solutions have drawbacks and no solution is universally applicable. For custom-built applications the solutions include utilizing portlet standards to convert applications to portlets and utilizing platform specific Application Programming Interfaces. For legacy applications the solutions include using custom wrapper software to translate communications.</p> <p>A custom .Net web application was integrated to Microsoft SharePoint as a case study. SharePoint was selected due to the technology base and feature set of the portal server. The application was integrated by using the APIs available in SharePoint after considering the options available. Ability to quickly integrate security features and to retain the original user interface were the most significant reasons for selecting this method.</p>		

It was asserted that the integration method selected worked well for this particular application. All integration goals were accomplished. It was also found that this method of integration allowed for more extensive testing that would have otherwise been possible. Another advantage of the modular design along with the integration method allows the application to function both inside the portal and as a standalone program. Disadvantages include additional complexity for the user and added dependencies to the software.

**Keywords:** portal, portlet, integration, EAI, SSO

**Language:** English

<b>Tekijä:</b>	Toni Sallanmaa	
<b>Työn nimi:</b>	Webohjelmistojen muuntaminen portaaliympäristöihin sopivaksi	
<b>Päiväys:</b>	19. tammikuuta 2011	<b>Sivumäärä:</b> 13 + 65
<b>Professuuri:</b>	Tietoliikenneohjelmistot	<b>Koodi:</b> T-110
<b>Työn valvoja:</b>	Professori Tuomas Aura	
<b>Työn ohjaaja:</b>	Diplomi-insinööri Matti Hämäläinen	
<p>Yrityksille ja muille yritysportaaleja käyttäville tahoille on tärkeää että kaikki yrityksen ohjelmistot saadaan käytettäväksi portaalisivuston kautta. Integraatio portaaliin tarjoaa käyttäjälle lisäarvoa, kuten esimerkiksi yhtenäistetyn ulkoasun muun portaalin ja muiden ohjelmistojen kanssa sekä portaalin tuomia parannuksia tietoturvaan. Tässä työssä tutkittiin millaisia palveluita portaalit tarjoavat ja verrattiin suurimpien portaali-toimittajien tuotteita. Tuloksena havaittiin että portaali-alustojen välillä ei ole merkittäviä eroja ja erot pienenevät entisestään samaan teknologiaan pohjautuvien portaalien välillä. Portaalien tukemat portlet-standardit, erityisesti korostetut palvelut sekä tietoturvaominaisuudet olivat suurimpia eroja eri portaali-alustojen välillä.</p> <p>Työssä tutkittiin myös eri tapoja integroida yksilöityjä, kaupallisia ja iäkkäitä ohjelmistoja portaaliin. Integraatiomenetelmän valinta projektiin on hyvin riippuvainen integroitavasta ohjelmistosta sekä portaali-alustasta jota käytetään. Havaittiin myös että jokaisella menetelmällä on omat hyvät ja huonot puolensa, eikä jokainen menetelmä sovi käytettäväksi jokaisessa tilanteessa. Yksilöityjen ohjelmistojen tapauksessa voidaan muuntaa ohjelmistot portaalin tukemaan portlet-muotoon tai käyttää portaali-alustan tarjoamia ohjelmointirajapintoja (API). Kaupallisten tai iäkkäiden ohjelmistojen ollessa kyseessä täytyy käyttää menetelmiä jotka eivät vaadi lähdekoodin muokkausta. Näihin menetelmiin lukeutuu muunmuassa wrapper-ohjelman käyttäminen ohjelmiston datavirtojen muokkaukseen.</p>		

Yritykselle yksillöllinen .Net ohjelmisto integroitiin toimimaan Microsoft SharePoint -portaaliympäristön sisällä osana työtä. SharePoint valittiin ympäristöksi yhteisen teknologian takia sekä ominaisuuksiensa takia. Työssä integroitiin ohjelmisto käyttäen hyväksi portaalin ohjelmointirajapintoja menetelmien vertailun jälkeen. Hyvinä puolina tälle menetelmälle havaittiin tietoturvaominaisuuksien hyvä integroituvuus sekä mahdollisuus säilyttää vanha käyttöliittymä muuttumattomana. Lisäksi säilytettiin mahdollisuus käyttää omia ohjelmistokirjastoja ohjelmistossa.

Havaittiin että valittu integraatiomenetelmä toimi hyvin tämän ohjelmiston sekä portaaliympäristön kanssa. Integraation tavoitteet saavutettiin täydellisesti käyttöliittymäintegraatiota lukuunottamatta. Huomattiin myös että mahdollisuus käyttää ohjelmistoa portaalin ulkopuolella integraation jälkeenkin mahdollisti myös ulkoisten testiohjelmistojen käyttämisen. Huonoina ominaisuuksina havaittiin että käyttäjän toiminta hidastui yhden uuden välivaiheen takia sekä ohjelmistoon tuli uusi riippuvuus portaaliohjelmistosta. Tietoturvaominaisuudet parantuivat merkittävästi portaalista saaduilla ominaisuuksilla, kuten Single Sign Onilla (SSO).

**Avainsanat:** Portaali, integraatio, portlet, EAI, SSO

**Kieli:** englanti

# Acknowledgements

This Master's Thesis was written for Analyse Systems Finland Oy.

First, I would like to thank my supervisor professor Tuomas Aura and my instructor Matti Hämäläinen for support, comments and help during the project. I received ample feedback for any issues I had with my thesis. I am especially thankful for all the grammatical suggestions and corrections that were given to me.

Additionally, I would like thank my family and friends for support and encouragement.

Espoo January 19th 2010

Toni Sallanmaa

# Abbreviations and Acronyms

AA	Authentication Authority
AD	Active Directory
AJAX	Asynchronous HTML and XML
AMF3	Adobe Messaging Format 3
API	Application Programming Interface
ASF	Apache Software Foundation
C#	Microsoft .Net Family programmin language
CAS	Code Access Security (.Net)
CBSE	Component-Based Software Engineering
CI	Continuous Integration
CMS	Content Management System
CSS	Cascading Style Sheets
EAI	Enterprise Application Integration
EIP	Enterprise Information Portal
GEMMLCA	Grid Execution Management for Legacy Code Architecture
HTML	HyperText Markup Language
IF	Identity Federation
IIS	Internet Information Services
IP	Internet Protocol
Java EE	Java Enterprise Edition
JDBC	Java Database Connectivity
JSR106	Java Portlet Specification 1.0
LDAP	Lightweight Directory Access Protocol
ODBC	Open Database Connectivity
ORM	Object-Relational Mapping
PHP	PHP HyperText Preprocessor
PKI	Public Key Infrastructure
RBAC	Role-Based Access Control
SaaS	Software as a Service

SAML	Security Assertion Markup Language
SWF	ShockWave Flash
SOAP	Simple Object Access Protocol
SSO	Single Sign On
TLS	Transport Layer Security
WSRP	Web Services for Remote Portlets 1.0
XML	eXtensible Markup Language

# Contents

<b>Abbreviations and Acronyms</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Goals . . . . .	2
1.2 Problem statement . . . . .	2
1.3 Structure of the Thesis . . . . .	3
<b>2 Background</b>	<b>4</b>
2.1 Web development languages . . . . .	4
2.2 Portlets . . . . .	5
2.3 Portal standards . . . . .	8
<b>3 Services offered by web portals</b>	<b>12</b>
3.1 Services . . . . .	12
3.2 Framework selection criteria . . . . .	15
3.2.1 Single Sign On . . . . .	17
3.2.2 Role-based Access Control . . . . .	19
3.3 Major Enterprise Information Portals . . . . .	20
3.3.1 Microsoft Sharepoint . . . . .	20
3.3.2 Oracle WebLogic . . . . .	21
3.3.3 IBM WebSphere . . . . .	22
3.3.4 Apache Jetspeed-2 . . . . .	23
3.3.5 Comparison . . . . .	24

<b>4</b>	<b>Adapting existing applications</b>	<b>26</b>
4.1	Enterprise Application Integration . . . . .	27
4.2	Integrating custom-built applications . . . . .	29
4.3	Integrating commercial or legacy applications . . . . .	31
4.3.1	Single Sign On with legacy applications . . . . .	34
<b>5</b>	<b>Case study: Adapting ASP.Net application</b>	<b>36</b>
5.1	Introduction . . . . .	36
5.1.1	Analyse <sup>2</sup> Galaxy . . . . .	36
5.1.2	Security . . . . .	37
5.1.3	Motivation . . . . .	38
5.2	Portal software . . . . .	39
5.3	Outcome . . . . .	40
5.3.1	Embedding Galaxy within SharePoint . . . . .	40
5.3.2	Converting security subsystem . . . . .	42
5.3.3	Third party libraries . . . . .	47
5.3.4	Deploying . . . . .	47
5.3.5	Testing . . . . .	48
5.3.6	Resulting system . . . . .	49
5.4	Further improvements . . . . .	50
<b>6</b>	<b>Evaluation</b>	<b>53</b>
6.1	User experience . . . . .	53
6.2	Technical features . . . . .	55
6.3	Development and administration . . . . .	55
6.4	Process . . . . .	56
<b>7</b>	<b>Future work</b>	<b>58</b>
<b>8</b>	<b>Conclusions</b>	<b>60</b>

# List of Tables

3.1	Comparison of leading vendor portal systems . . . . .	24
-----	---	----

# List of Figures

2.1	Portal page with portlets . . . . .	7
2.2	WSRP structure . . . . .	10
2.3	WSRP used in conjunction with JSR-106 . . . . .	11
3.1	Service-based portal . . . . .	16
3.2	PKI-based complex SSO system . . . . .	18
4.1	Enterprise Application Integration . . . . .	28
5.1	Architecture of Analyse <sup>2</sup> Galaxy . . . . .	37
5.2	Planned architecture of portal-ready Analyse <sup>2</sup> Galaxy . . . . .	40
5.3	Architecture of portal-ready Analyse <sup>2</sup> Galaxy after modifications . . . . .	50

# Listings

5.1	C#-code to initialize Galaxy with the SharePoint Context . . .	42
5.2	Key parts of the user provider module . . . . .	44
5.3	Custom Code Access Security (CAS) constraint . . . . .	46
5.4	using custom Code Access Security (CAS) constraint through attributes . . . . .	46

# Chapter 1

## Introduction

A web portal, in the broadest sense of the term, means a specialized web site (a collection of individual documents) that is used to access resources which are generally, but not necessarily, on the Internet. The nature of these resources is not strictly defined and different terms are used for different types of web portals. A general web portal is used to describe a gateway-type page from which the user can navigate to other sites not necessarily provided by the supplier of the portal. The type of portal this thesis focuses on is an Enterprise Information Portal (EIP). EIP can either be an internal portal or an external portal. In the latter case customers utilize the portal to access services and products offered by the company in a streamlined and integrated manner. [23]

Another way to define a portal is that a portal aggregates data from multiple sources and provides the aggregated information to user groups in specific ways. The function of the portal is to provide all the necessary resources and applications in an integrated manner and to offer only the information required by the user instead of unorganized source data. [23]

Companies providing software as a service (SaaS) use portals to combine their software portfolio behind a single, unique access point. These portals can be highly customized to cater to the needs of a specific customer. There are many technologies, tools and standards created to facilitate the maintenance and development of a web portal, each offering a differing set of features. [1] However, companies that already have web applications need to have their existing product portfolio seamlessly merged into a portal to provide a uniform access point to a user.

## 1.1 Goals

This thesis is written alongside a project to adapt an existing web application for it to work with a web portal framework. This project is done for Analyse Systems Finland Oy. The goal of the project is to make the application operable within the portal environment, while using the features and services provided by the portal framework to improve the application. These features include security features (including Single Sign On and Role Based Access Control) and user-specific features.

*The goal of this thesis is to describe the objectives and practices on adapting existing software for integration to a web portal, implement the adaptation of one existing application, and evaluate the adapted system according to the specified objectives. The work should yield a working portal application ready to be deployed, documentation of the integration process and ideas for how the process could be improved and the integration to the portal made more seamless.*

## 1.2 Problem statement

The key question is how to adapt an existing application for use in a web portal and what improvements can be achieved by integrating the application into a portal. The benefits a customer receives from the integration and the costs of the whole integration project are important questions as well. This question is raised when a company expands its product portfolio and wishes to present the whole range of products to a customer from a unique entry point in a uniform way. Selecting the tool for constructing the portal is also an important decision, but it is often mandated by the technology used in the applications.

- What is the customer or end user benefit from the integration?
- How can an application benefit from being integrated into a web portal?
- What are the different options for integrating an application to a web portal?
- How portal frameworks differ for the purposes of integrating external applications and what kind of services do they provide?
- What are the pitfalls that may prove difficult to overcome when adapting an application to a portal?

The scope of this thesis includes the study of different web portal implementations and their different feature sets focusing on how accessible they are for adapting an existing application to work on them and what limitations do they have. Services offered by different portal implementations are discussed as a tool to improve the application. Furthermore the thesis includes as a case study the adaptation of an application to a portal framework and evaluation of the resulting system.

The web portals surveyed for this study are limited to those offering more services than traditional content management systems (CMS), i.e. the ones that are capable of running custom applications.

### 1.3 Structure of the Thesis

This thesis is structured into eight main chapters. The first chapter is the introduction, which includes the problem statement and the scope of the work. The second chapter introduces background information on portals and web applications, and the following chapter describes the web portal development frameworks and tools and the services offered by them. The fourth chapter then describes the strategies used to adapt an existing application for use in a portal environment and is followed by a case study in chapter five. Chapter six focuses on evaluating the results from the case study and suggests improvements to both the integration process and to the technical solutions. Finally, chapter seven presents topics for future discussion and chapter eight summarizes the thesis.

# Chapter 2

## Background

Portals, as defined by Tatnall, are gateways to the Internet and aggregate data for the user while trying to protect from the general chaos present in the massive Internet [23]. However, websites and portals are not simple static pages as in the past but instead focus on techniques for adding interactivity and seamless transitions to distinguish them from the page based approach. This transition has resulted in a number of different web development platforms and languages [21].

This chapter introduces the web development environment and discusses the standards that are used in conjunction with Enterprise Information Portals (EIP) to achieve interoperability between different developers.

### 2.1 Web development languages

Web development has gone forward in strides and there is now a range of different languages and platforms that can be used to create web applications and web sites. The distinction between an application and a site is harder and harder to distinguish and especially with most applications working with dynamic Hypertext Markup Language (HTML) it is almost impossible. However, even when the frontend of the applications is written in dynamic HTML and techniques such as Asynchronous JavaScript And XML (AJAX), the server backend can be developed in any language that has sufficient bindings for the network infrastructure. The languages are still further divided into scripting languages, popular examples of which are JavaScript and PHP, and compiled languages including Java and C#.

Prechelt notes in his exploratory experiment of platform properties that web

development platform evaluation should also include the available libraries and open-source code that is found on the Internet. The more social style of programming that utilizes and encourages sharing of resources on the Internet has led to a variety of different global development cultures for different languages. A piece of code can be said to be in the style of some programming language, for example Perl. This presents the programmer with the problem of choosing one development culture in addition to the programming language. These factors make the selection of the development language an important decision beyond the simple preference for a language. [21]

Pretchel's study showed that, in an empirical experiment, there were no significant differences in productivity when writing the same piece of software in different languages. The study also found that the technical differences of the selected platform mattered the most. Some of the languages available are simply better for some tasks due to basic functionality available in the core platform libraries. In the end, it was observed that the people in the team and technological expertise are more important than the strictly technical platform properties. [21]

However, in the field of Enterprise Information Portals there are not many choices available for a developer. There are Enterprise Information Portals that support scripting languages, but most of the application development is in Java or C#, partly because of the portlet standards that support these languages. Both of these languages are suited for an extensive range of applications and are easily extensible by libraries that can also be used in portal environments.

Standardization does not offer a clear-cut solution to the choice of a language to develop with. The Java Portlet Specification (JSR106) is written for Java software and there are a lot of portal servers that support this standard. However, there is another portal standard, called Web Services for Remote Portlets (WSRP), which is designed to lean heavily on existing standards and to be language and platform independent. There are also portal servers supporting WSRP, some of which support JSR106 as well.

## 2.2 Portlets

Portal software can be created in different ways, for example as component-based development. Software that is executed inside the portal environment is usually called a portlet but the terminology differs between languages.

A portlet can be a big monolithic project that implements everything including user interface, or it can be a small reusable component. Building a monolithic portlet resembles a traditional software project, where third party components or libraries are tightly integrated into the software. On the other hand, software constructed as components will be integrated with other components only when a designer creates a portal page from them, and it can be integrated with other pre-existing or third party components. However, the term portlet usually refers to the component approach and the term for a monolithic project is web-based application hosted in a portal. [2]

The Java Portal Specification describes portlets as parts of a portal page where each portlet dynamically creates markup language to display the data generated by the portal. Portlets do not act independently but instead depend on the portlet container (also called a producer) to supply the portlet with the portal platform services and to instantiate it when required. One resulting portal page is then constructed from a number of portlets in a layout specified by the portlet container and sent to the user through the portal server. As such, portlets can be reused and offered in different scenarios and layouts for different users. [2]

Figure 2.1 shows an example of a portal page constructed with portlets. The left side of the picture depicts the client machine running the client software. The client machine hardware can be, for example, a desktop computer or a smart phone. The client software is usually a web browser. The client software requests a page from the portal server process running on a remote machine (Content server). The portal server will have identified the user in a previous step and proceeds to generate a customized portal page for the user. In order to do this, it first needs to find out which portlets are required in order to show the content and then ask the portlet container for the markup code generated from the portlets. The portlet container then invokes the necessary portlets after creating them if necessary. Portlets are independent from each other and can make use of further background services such as databases. Portlet generates the markup for the window it occupies and the portal container is responsible for combining the different portlet windows together to form a layout. Further interactions to the portlets will traverse the same path through the layers to the portlets and can be stateful in nature.

Web-based applications hosted in a portal act essentially the same way as portlets in a portal page. They can be thought of as huge portlets and have the same access to the portal platform as a portlet. Platform-specific features are added to the web application by launching it from inside a portlet. The user interface is not supplied from the portlet but is instead offered directly

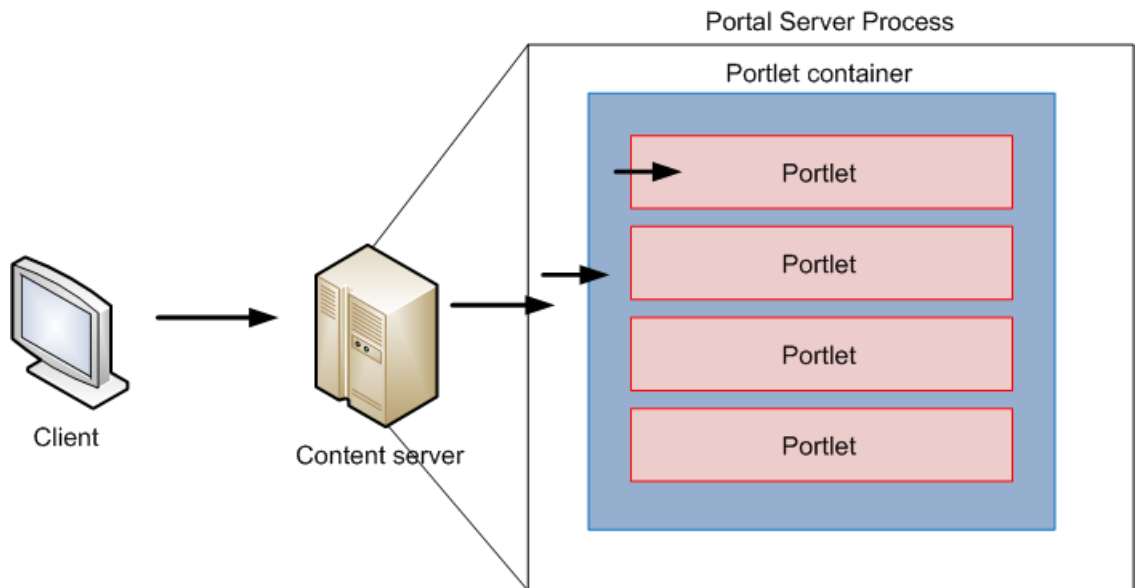


Figure 2.1: Example of a portal page with portlets

from the web application. If the portal part is not used, the web application is no different from an application running on independent server. Web-based application hosted in a portal is not to be confused with a portlet application, a Java term for a collection of portlets that together form an application and that are deployed together.

Developing portal software as small components responsible of only a small part of the whole functionality offers some additional benefits. First, the resulting software will be easier to develop due to a smaller code base and specification. Second, a designer can be used to create the user interface without the need for programming skills when the components, or at least placeholders, are ready. Third, common components, for example a search box, will be reusable for other projects when developed in a generic manner. Fourth, the more generic components can be bought from outside vendors and customized to fit the needs of the current project to shorten the development time. As a downside to developing a project as components is that it will bring additional overhead when integrating everything together to form the complete application. In software development, this style of development is called Component-Based Software Engineering (CBSE). [20]

## 2.3 Portal standards

There are two standards commonly used in portlet development for web portals [25]. These standards aim to make portlets interoperable between different portal servers, provided that the server implements the same standard. These standards can be safely omitted if only developing on a certain platform that provides a proprietary Application Programming Interface (API). One such portal server is Microsoft's SharePoint.

Java Portlet Specification (JSR106) is a specification for portlets developed in the Java language. These portlets run inside a portlet container that is used by the portal server itself. The portlet container is given multiple options by the specification for when to instantiate portlets and how to handle them. Portlets gain access to portal services through the container by the specification API and utilize the services for example to identify users. [2]

Web Services for Remote Portlets is a specification aimed for standardizing the interactions between services that produce some data of interest to the users and the presentation of this data. Traditionally, the web service itself would produce data straight to the user and has to contain the presentation logic to allow the data to be read in a clear manner. However, having to develop the presentation logic separately for different sources is inefficient and makes integrating the sources as a part of a portal much harder. WSRP solves this problem in a way similar to JSR106 by presenting an interface through which the data can be acquired in a standardized way. This means that WSRP and JSR106 are not competing standards and JSR106 Java portlets can potentially offer WSRP interfaces as well. WSRP separates the portal server and the hosting of portlets which allows a single portal server to harvest portlets from multiple sources developed with multiple languages and platforms. [15]

WSRP works with the three different parts displayed in figure 2.2. Portlets (1) are the core component generating data and the markup associated with it. Portlets are contained inside producers (2). A producer can contain multiple portlets and acts comparably to a portlet container. It also has interfaces for portlet management. Consumer (3) is a middle-tier service that queries producers to find out what services are offered. The consumer and the producer do not have to be on the same machine. When a consumer locates a necessary portlet it gets a portlet handle from the producer. The handle is used for communication between the portal and the consumer. There is no need to relay further communication through the producer. Finally, the consumer aggregates all the data in markup form it has received from

portlets and displays the aggregate to an end-user (4). The main difference with JSR106 is that one consumer can obtain portlets from many producers and that the producer can be omitted from the messaging chain after the correct portlet has been found. [15]

Figure 2.3 adapted from Allan et al. displays a scenario where JSR106 and WSRP are successfully used together and how both standards together can be used to construct complex portlet systems. WSRP is used by the portal server to gain access to the portal container. Portal container contains portlets and a portlet proxy. Container communicates with these entities with JSR106. Portlet proxy acts as a gateway to another remote system and acquires a reference to a portlet there using a producer and afterwards communicates directly with the remote portlet using WSRP as in the figure. With an architecture such as this, it is easy to utilize multiple portlets and portlet standards together. [3]

In both WSRP and JSR106, the portlets themselves generate a markup language that is in turn used by the portal server to display the resulting data. Utilizing technologies such as Cascading Style Sheets (CSS), the markup language does not contain styling information for the data and the layout and style of the displayed data is determined only at the portal server. This allows different users on different client platforms to receive a different representation of the original data.

Security is not a major concern in JSR106 as the portlets reside on the same server as the actual presentation logic, but in WSRP the data is generated somewhere else in a different system so that security measures are needed. The specification suggests document-level authentication with existing web security standards such as Security Assertion Markup Language (SAML) and authentication of data transfer between producers to consumers with certificates and Transport Layer Security (TLS). TLS can also be used to secure the transfer of markup and commands between the two parties. There are no defined methods for access control and the consumer can implement some access control scheme before requiring the consumer. Similarly, the producer can implement any access control scheme based on the authenticated user identity. [15]

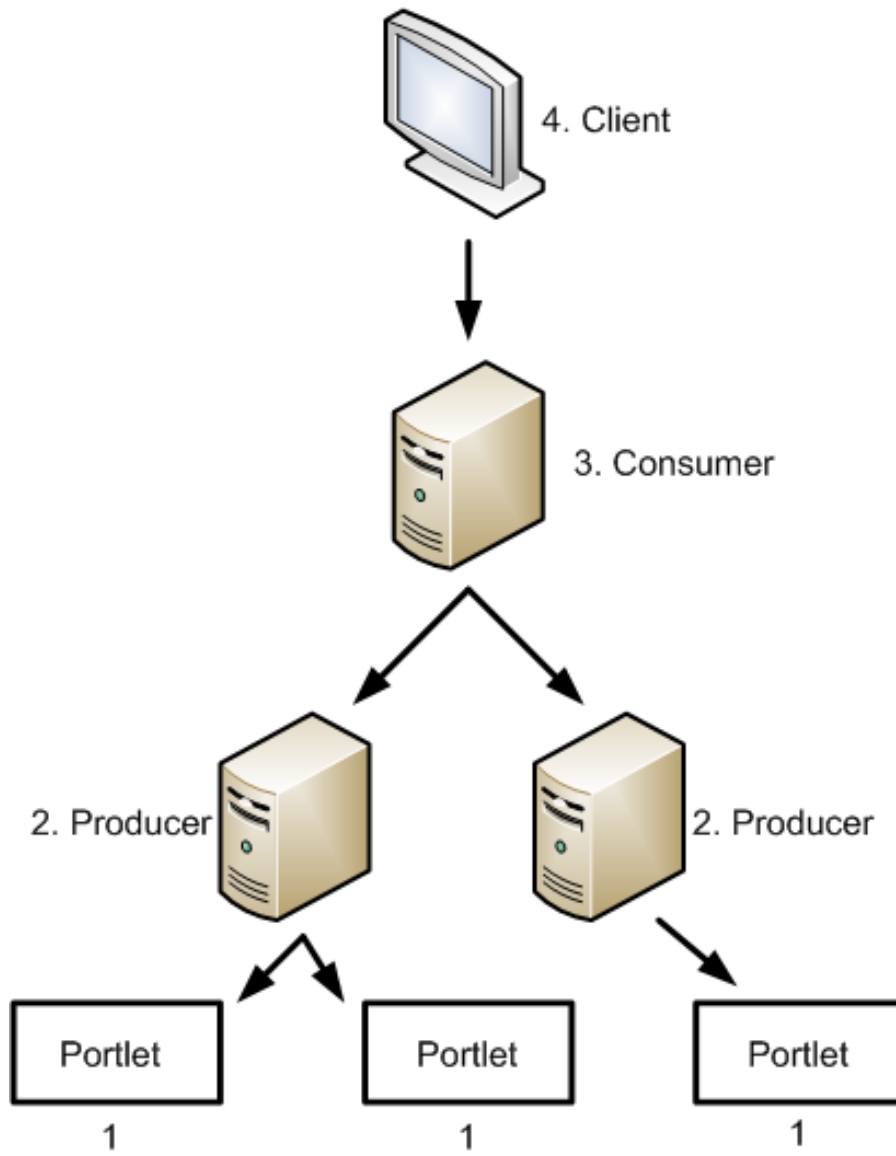


Figure 2.2: WSRP presentation structure

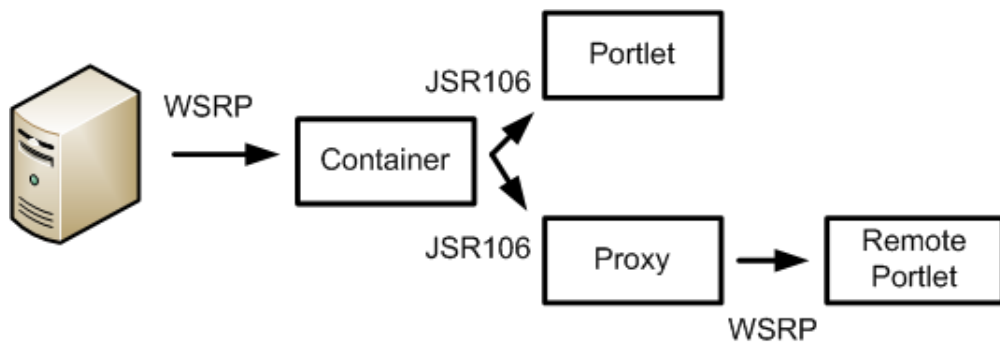


Figure 2.3: WSRP used in conjunction with JSR-106, adapted from Allan et al. [3]

# Chapter 3

## Services offered by web portals

The technology behind the portal server is an important consideration when choosing a suitable Enterprise Information Portal (EIP) or Application Server. Multiple vendors support the same standards but the range of additional services that the portal offers to users and to the portlets running inside them differ. There are differences in the services themselves, in their expandability and interoperability. Portals also differ in features not related to the technology and software, such as support and community, which are important features to consider as well. This chapter explores the services provided by different portal systems and introduces major vendors and compares the products. [3]

### 3.1 Services

Allan et al. present a list of example services that a portal framework might offer to the developer and to the user. The following is an adaptation of their list [3]. Some of the services that are described are not explicitly portal-related and have been implemented in other systems but they nevertheless complement the feature set of a typical portal environment.

#### **Session handling**

The timespan between the user arriving to the portal system and a predefined time after the last performed action is called a session. In some contexts a session is also the timespan between log in and log out. One of the services offered by the portal platforms is session handling and validation. Portal provides the session information to the portlets when requested.

**Login and authentication**

The portal has a user database or access to a user directory which it uses to authenticate users and to bind an identity to the user. This information is commonly stored during the session. It is created during log in and usually erased when the user logs out.

**Policy- or Role-Based Authorization**

The user identity commonly has roles attached to it to facilitate Role-Based Authorization. These roles are retrieved concurrently with the user identity at log in. If the user is not logged in, he is using a guest or anonymous identity. Policy-based Authorization is a system where actions have policies that deny access or grant it depending on the conditions set on the policy. It can use the roles, but there are other types as well [6]. Authorization is provided by the portal platform because it is commonly used in the portal itself.

**Shopping cart**

This is an example of a service that the portal can offer but is better suited for a portlet. Many portlets can use a shopping cart system to allow the user to store objects during the session. Interaction between portlets is required if the portal server does not offer a shopping cart service.

**Querying data sources**

The portal platform can offer the service of retrieving data to be displayed. The actual connections and queries as well as configuration options are all handled by the portal. Portlets can then request data from a specific source from the platform and display it to the user. This makes data-access configuration easier as everything is handled in one place.

**Job list and status**

Tasks that require heavy calculation or long-running processes can be queued up to be processed at a later stage. Also, tasks that require collaboration from many users, such as workflows, can also use this system. The portal can handle the queueing and execution of these jobs and provide status information as feedback to the portlets.

**Messaging service**

Messaging is a communication method between different parts of the portal platform and portlets present in the portal server. When using message passing as the communication method, processes leave messages to be routed to other processes. The messages are routed by the

messaging service which the portal supplies. Portals commonly use this kind of messaging internally and it is available as a service as well.

### **Service registry lookup service**

Service registry is a system for extending the system with new services that other portlets and applications can look up to provide additional features. These features are queried from the service registry with the actual type of service they provide instead of the concrete implementation. This allows multiple services to implement the same type of service, such as messaging.

### **Resource scheduling**

The portal can be configured to have access to outside resources such as databases or other data storage systems. The portal platform can provide access to these resources for portlets and applications and schedule the accesses to maximize performance. In some instances, this can also be used to reserve timeslices on reservable resources such as meeting rooms.

### **Event notification services**

Event notification is a service where applications or processes can register to listen to specific events generated at the portal system by other applications or processes. When an event is produced, every listening process is notified of the event and the processes are able to react to the event by performing various actions. This is one of the core methods of interprocess communication and it is usually implemented through message passing systems by the portal platform. This service is also called notification service.

### **Portal-based collaboration and social networking services**

A portal is a multi-user environment. The platform should allow for easy collaboration between different parties restricted only by the security features. This does not mean the portal should provide the features out-of-the-box but rather provide the foundations for building collaborative services. Workflows are an example of a collaborative service. Users' own personal pages or spaces in the portal are examples of social networking features.

### **Standardized outside content access**

A portal can implement some standard protocols for accessing outside content. Such standards include Simple Object Access Protocol

(SOAP) or eXtensible Markup Language (XML). This eases the creation of portlets and applications that rely on the content that it is not necessary to implement content retrieval separately in each portlet.

### **Multidevice support**

Portals are accessible from a wide range of different browsers that have different capabilities. Multidevice support aims to provide the best possible experience for any user regardless of the device that is used to browse the site. To achieve this goal, it may be necessary to leave out some content to optimize the site for smaller resolution displays. Interactive content may also be reduced to improve performance. The amount data to transfer to the client is also a concern when working with mobile devices. The network connections are not as fast as on dedicated devices and there may be a limit on the available memory.

Some of the services described above are much closer to the core of the portal platform than others. Certain services, such as a messaging service, need to be integrated into the portal platform while other services can be implemented as add-ons. One way to select services and features to be moved outside the platform core is to measure their user base or how often a feature is used in installations of the portal platform. Less used services can be provided as add-ons or additional installations if they are not critical to the operation of the base system. [3]

Figure 3.1 depicts a simple service-based portal on high level. Browser, the portal and the remote portlets are separated from each other by firewalls. The portal server has an array of services associated with it and shown as boxes within the portal system. The portal can offer additional services to the users by utilizing remote portlets residing in other systems. The remote portlets are shown as the portlet proxies. The user does not directly connect to the remote portlets. Instead, all communication goes through the portal server. [3]

## **3.2 Framework selection criteria**

Allan et al. also provide a list of some criteria for selecting a suitable portal framework. A list adapted and expanded from their work is presented here. [3]

### **Integration with existing functionality, portlets and applications**

It is preferable to use the portal platform that provides the best support

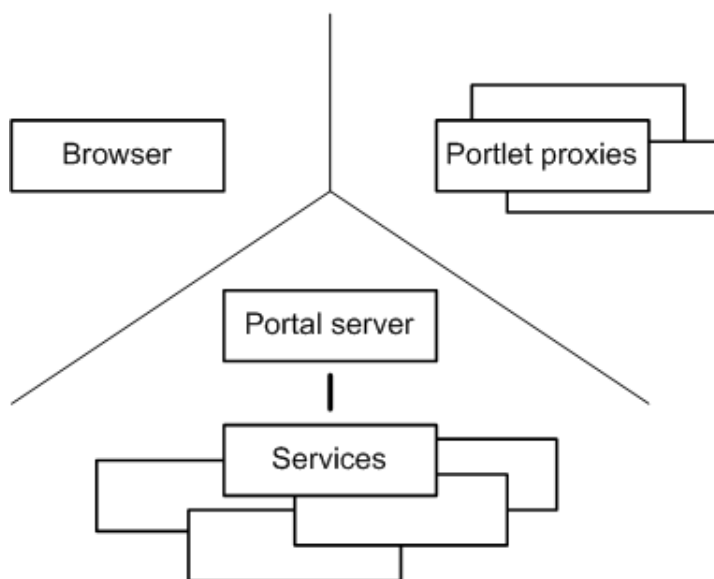


Figure 3.1: Service-based portal [3]

for existing applications or portlets. The technology behind the portal server is a factor but adherence to portlet standards helps to keep software compatible. Additionally, the services provided by the platform should work with the existing software without modification. Major code rewriting or refactoring is not necessary with a portal platform that integrates well with the existing code base.

### **Ease of development**

Ease of development depends on the documentation, support and user community surrounding the portal. These factors make developing new portal software easier as the community can provide libraries and offer support for problems. Documentation and support are provided by the portal server developers or publishers and good quality documentation is a key to creating good quality portlets. Also, the quality of Application Programming Interfaces and other developer-facing tools is another factor.

### **Adherence to standards**

Standards allow portlets to be transferred and deployed to many different portal platforms or to be used through portal proxies. Following JSR106 and WSRP standards will ensure that the portlet is usable in the future and, depending on the nature of the standard, drives developers to create software with good practices and architecture.

**Programming language independence**

A portal system that provides support for multiple languages can offer access to a wider variety of third party portlets and libraries. Integration with existing functionality is easier to implement with multiple languages. On the other hand, WSRP already allows portlets to be created in any programming language as it provides a communication framework and not an Application Programming Interface (API).

**Availability of source code**

Open source portals grant full access to their source code. This allows developers to make changes to the software to make it better suit their needs. Later it is possible to publish those changes and to integrate them into the community supported version. Poor documentation may be augmented by reading the required information straight out of the source code, assuming it is of good enough quality. Finally, adoption costs for open source products are often lower than for commercial products and as such they are good for prototyping efforts.

**3.2.1 Single Sign On**

Single Sign On (SSO) is one of the most prominent security features offered by the portal systems [25]. SSO is a mechanism which allows the user to log in to the system once, after which the authentication information is automatically propagated to every other system connected to the SSO server. Logging out happens in the same way. The most obvious benefit for the user is to have a single set of credentials to remember. This also increases security as the credentials are not replicated and the user can pick stronger passwords since there are fewer of them to remember.

SSO is commonly built in to the platform and is a core service as the portal administrative features are often secured with the same system that is provided for portlets. Plug-in systems have emerged for portals that allow them to use different authentication systems from different developers to make the platforms easier to integrate into existing systems [14].

Single Sign On systems, both built-in and external, consist of multiple components. In general, SSO combines identification and authentication. Authorization is handled by the application utilizing SSO. The party or system providing the Single Sign On service must be trusted by the application as the user only provides his or her credentials to that party. The SSO service will then provide the user identity to the requesting system.[7]

Authentication server is the service that provides the user identities for the other systems. It is commonly referred to as the SSO server. In essence, the SSO server is just a proxy for requesting the identity from another service that contains the actual user accounts, such as Microsoft Windows Active Directory. The identities and accounts come from the system called the Authentication Authority. One authentication server may use multiple authentication authorities and it is possible that the authority is built into the authentication server. [7]

More complex solutions have been developed for situations where the services utilizing SSO are not in such a close proximity (running on the same platform) to the authentication server. In that case, tokens are obtained from the authentication server and passed on to the application servers where they are used to grant access. The tokens are secured by cryptographic certificates to prevent token forgery but require a trust relationship between the servers as well. Multiple SSO systems can operate together or the systems may span multiple networks. [7]

Figure 3.2 shows an example layout of a complex SSO-system. The system is based on Public Key Infrastructure (PKI) and each party has a certificate containing their keys. User authenticates with Authentication System A. Services in system B are accessed by first asking system A for a token that grants access to a specified resource or service. Authentication System B validates this token, and because there is a trust relationship between the systems the access is granted. Other possible ways to accomplish the same result is to cache credentials on the user client and authenticate with system B without user intervention. This method also requires the credential database to be shared or replicated. [7]

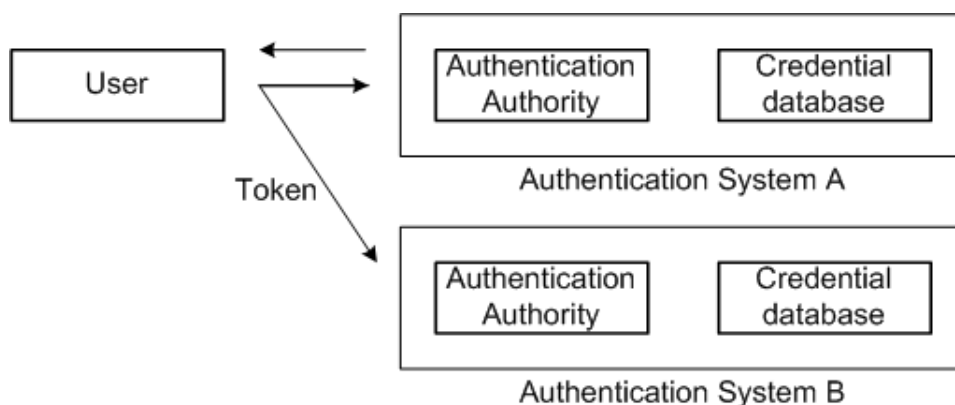


Figure 3.2: PKI-based complex SSO system [7]

Identity federation means that one organization or service accepts user identities defined in another organization or service. For example, in the HAKA federation, user accounts from one Finnish university can be used for access control to services at another university. In addition, any nonrelevant information does not need to be revealed to the portal. On the other hand, the party owning the portal has to trust the identities from the party providing the identities. Portals can also offer support for identity federation features, often by allowing the use of third-party authentication authorities. [19].

### 3.2.2 Role-based Access Control

Role-Based Access Control (RBAC) is an authorization method where access to resources is restricted to users with certain roles. As an example, RBAC could restrict administrative settings to be available only to the users with a role called 'administrator'. Privileges granted to users come in the form of different roles. When a user has a certain role he or she is said to be in that role. Even the most standard user has a special user role to distinguish him or her from anonymous users who have not logged in. Additional roles granting additional privileges are added to that baseline. [9]

The purpose of linking actions to the roles is to refrain from giving privileges to individual users. The addition of a new user is a simple task: create the user and assign proper roles to grant privileges pertaining to the user needs. Roles can grant privileges to do multiple tasks and actions can be allowed for multiple roles. This makes permission management a much easier task, especially when roles are assigned logical names. For example, a role given to certain users in a banking system might be called 'accountant' and, when you wish to modify permissions the accountants have, that role is modified instead of modifying the permissions of the individual users. [10]

Resources are guarded with policies that inform the system what roles are required to access a specific resource. These policies are often created by an administrator or they may come preconfigured. A hierarchical system is also possible, where policies form a hierarchy and policies are inherited from ancestors in the hierarchy. Roles can also have hierarchies with higher level roles gaining the same permissions as lower level roles in the same hierarchy. For example, the accountant role might have a lower level role called 'intern' in the same hierarchy, so that the accountant would gain the same privileges as the intern and add some more on top of that. [10]

Due to its suitability for a range of environments and its widespread use [10], Role-Based Access Control is a service that a portal platform can offer and

utilize as an internal security tool. An example of a portal platform using RBAC as an authorization method is the IBM WebSphere Server. RBAC together with a Single Sign On system can form the base of a portal platform security system.

### 3.3 Major Enterprise Information Portals

The following section examines the products developed by the lead vendors [1] of Enterprise Information Portals. The services offered by the platforms are briefly described and focus is also on adherence to the two prominent portal standards, JSR106 and WSRP. Another feature to note is the adherence to the newer version of JSR106 standard, the JSR226. The results are collected to a comparison table in Figure 3.1.

#### 3.3.1 Microsoft Sharepoint

SharePoint is a portal server and publishing system currently in version 2010. SharePoint has been widely adopted for different purpose portals and other projects. The portal itself is running on top of the Internet Information Services (IIS) web server. This is the only option for SharePoint as the technologies it relies on can not be used with any other web server. This also limits SharePoint to be usable only on the Windows family of operating systems as the the IIS web server is not available on any other server platform.

Unlike the rest of the portal platforms described here, SharePoint is not a Java-based product and instead uses .Net as the underlying technology. The technologies are remarkably similar and both offer basically similar services. The JSR106 portlet standard is not supported as it is a Java specific standard. SharePoint has similar Application Programming Interface (API) in place for creating self-contained portlets to be used with SharePoint portals. These portlets are called Web Parts in the SharePoint environment. The limitation on JSR106 portlets is not as absolute as it first seems. Remote portlet proxies provide portlets to other portals by utilizing the Web Services Remote Portlets (WSRP) standard for communication. The SharePoint portal supports WSRP and thus JSR106 portlets can be used in SharePoint by using a proxy in between. The remote portlets can even be physically on the same computer as the SharePoint server.

Most of the features of the .Net technology platform are also available for use within the SharePoint portal server. These include easy connectivity to a

number of persistence providers, such as databases, via generic connectivity systems like Open DataBase Connectivity (ODBC). The .Net security model is also the base for the SharePoint security system. The same components can thus be used with web applications and portlets. The security system of SharePoint also includes a Single Sign On (SSO) system, a related role-based access control system and an identity federation framework. A number of different authentication authorities can be used in SharePoint with replaceable security providers.

The SharePoint server ships with the portal server itself and a number of different preconfigured options for developing the portal. The portal consists of Sites that are independent of each other. Each site can be customized to suit a particular need from publishing portal to internal collaboration environment. Configurations to suit these needs are installed along with the portal and custom configurations can be created to extend the functionalities. Site design can be done without writing program code by using an interactive editor. Custom components for use in the sites are created by programming them.

Additional components for the server are also provided. Among these is a web-based control center which can be used to control almost every aspect of the portal server. Other components include a search, a blog system and a data manipulation suite designed to offer data in a convenient format. There are also other custom components available which can be used as standalone applications or integrated into any portal application. These components either come with other products or are separately available.

There is comprehensive support and documentation available for the technology, the platform and the portal itself. Premium support is also available as well as a library of books aimed at different kinds of developers and administrators. The portal system is created to be usable without deep technical knowledge.

### 3.3.2 Oracle WebLogic

WebLogic, currently in version 11, is a Java Enterprise Edition -based platform containing an application server, enterprise information portal and a set of other tools. The WebLogic platform is divided into three parts called the Portal Business Services, Portal Lifecycle Management and Unified Portal Framework. [1]

Portal Business Services consists of services, some of which were described earlier. Content management, search, collaboration, commerce and interac-

tion management are contained within Portal Business Services [1]. Content aggregation from multiple sources is one of the defined features of a portal, and content management services allows the WebLogic portal to gather information from multiple sources. Interaction management provides customization for the user and features to enhance the user experience. The platform supports JSR106. Commerce services contain services to enhance online commerce.

Portal Lifecycle Management makes the portal platform easily adaptable to a multitude of different users and user groups. Tools for controlling the portal through a browser-based interface aimed for non-technical staff are also provided. Developer tools used to create new features and portlets for the portal system are a part of the Lifecycle Services. WebLogic also has some extensions to the JSR106 standard, called portlet extensions, that allow newly developed features to use additional features from the portal platform. [1]

Finally, Unified Portal Framework allows for easier integration of other enterprise tools by utilizing Enterprise Application Integration (EAI) which standardizes the integration. The ability to use the portal platform to build portals to suit a variety of different needs (such as a publishing portal or an internal collaboration portal) is a feature under Unified Portal Framework. [1]

WebLogic portal can be used with any databases or other storage methods supporting the JDBC (Java DataBase Connectivity) interface. The purpose of JDBC is to allow access to relational databases from program code and to abstract the actual data source [4]. The portal itself uses a database to store internal information and user data. Configuration information is stored in an eXtensible Markup Language (XML) format. [1]

### 3.3.3 IBM WebSphere

WebSphere product family is also divided into three parts that together form the whole middleware platform. These parts are Foundation & Tools, Business Integration and Business Portals. The essential components for web services are all in Foundation & Tools. Among these is the Java EE Application Server used in WebSphere portals. Business Portals includes all the necessary tools for constructing portal services for various purposes. Additional WebSphere products in any of the mentioned categories can be used to build services to complement the portal. [1]

The portal server process runs on many platforms and is based on Java

technologies. The portal server is a continuation of an older Apache Software Foundation portal project, Apache Jetspeed [24]. It supports the basic portal features like administration, user handling, application connectivity and data presentation. This basic feature set is further extended by additional feature packages that are loaded into the portal server that bring more powerful features such as collaboration or customization options [1].

Websphere portal server also includes a Java Enterprise Edition (Java EE) application server and has support for web services and portlets. The portal server supports JSR106 and WSRP as portlet standards. It also supports the newer JSR226 portlet standard. Another notable feature is JDBC to allow a wide range of databases to be used. Java Enterprise Edition security features are also available to be used from the portal itself and from web services or portlets. [1]

Websphere Application Server is capable on running on various different platforms. Likewise it can utilize different web servers on different platforms. Extensions to the standards are available for applications. Due to crossplatform requirements, all extensions are available on every supported platform. The websphere application server has been built to scale, which is an important feature for high traffic portals and other services.

### 3.3.4 Apache Jetspeed-2

Apache Jetspeed-2 is an open-source alternative to the commercial enterprise portals from Apache Software Foundation, currently in version 2.2.1. It is built from the earlier Jetspeed portal project. It is well featured and actively maintained. Jetspeed-2 does not include an Java Enterprise Edition application server and requires one to be available. It supports many alternatives, including two of the mentioned commercial products, WebSphere and WebLogic. [5]

The portal server supports portal standards such as JSR106 and its later iteration JSR226 and WSRP. There is less focus on enterprise features than in commercial offerings but, instead, there is a heavy emphasis on easy extendability through portlets. To facilitate this, some extensions to the Java portlet standards are present in the portal server. Administration features are present in the form of administrative portlets and these do not require technical knowledge.[5]

The security system includes a Single Sign On functionality with federation support and is customizable and extendable. For example, the security system contains the possibility to use any Lightweight Directory Access Proto-

col (LDAP) compliant user management system as the user authority for the system. Like other Java portals, Jetspeed-2 can connect to JDBC-supported relational databases. The portal platform also supports Enterprise Application Integration (EAI) standards which can be used to integrate business applications to the portal using standardized communications channels.

One interesting new feature is the ability to customize the portal on the client side using dynamic web techniques such as AJAX [5]. This means the user does not have to use any of the portal designs the portal administrator has made available and can create a customized personal design. In some cases this is not a good idea, however. With full customization, the user could deface the appearance or cause unnecessary problems for regular use of the portal if workflow is disrupted. Finally, there is also a possibility to deploy client-side portlets.

Due to the portal being open source, primary support is available from the project web site and from the user community. Unlike commercial offerings, premium support is not available [18]. A fully functional portal environment can be built from solely open source products using Jetspeed-2 as a base.

### 3.3.5 Comparison

Portal	Vendor	Technology	Standards	Main features
SharePoint	Microsoft	.Net	WSRP	Security, administration, integration, ease of use and support
WebLogic	Oracle	Java	JSR106, WSRP	Content management, interactivity, extensibility
WebSphere	IBM	Java	JSR106, JSR226, WSRP	Extensibility, administration, security
Jetspeed-2	Apache Software Foundation	Java	JSR106, JSR226, WSRP	Security, administration, customization

Table 3.1: Comparison of leading vendor portal systems

Table 3.1 shows a brief comparison of the portal platforms from the leading vendors. As can be already seen from this generalized approach to features

the portal systems are not too far away from each other in terms of services and features. Java is the leading portlet language and portal technology at least in the surveyed portals. Consequently, it is also the only language to feature a dedicated portlet standard that is still being developed into new versions. All portals are also able to use remote portlets via WSRP and all have some support for providing an extensible feature set.

The selection of platform to use is mainly based on the costs of setting up the particular portal platform, the choice of the underlying technology and how well it fits the existing software products, and the feature set the platform provides. The three commercial platforms have premium support options available as well, while the free open-source platform relies on community support and documentation.

Finally, the feature sets are remarkably similar. Most platforms can be adapted for any use, but the important question is how easy it is to use the selected platform for the task at hand and would it be significantly easier on some other platform. The security features are also similar, with each platform offering at least some security features to the developer.

# Chapter 4

## Adapting existing applications

A corporation or any other organization may have existing web based software when it decides to deploy a portal environment. Depending on the software, the age of the software and the needs of the organization, the existing software may need to be adapted to be accessible from within the portal environment. Applications and other software can be adapted to work in the portal environment regardless of the portal software chosen, but differences in availability of Application Programming Interfaces (APIs), platform-dependent technologies, standard adherence and various other factors mean that the portal platform will influence the adaptation process.

Different concerns arise depending on whether the portal, or parts of it, is external or internal. External portals are available to the internet and meant to be accessed from the internet. Internal portals provide services and information only for the internal network of the party that controls the portal and are separated from the internet by firewalls or other methods. This means that security requirements are different for internal and external portals. Further consideration for external portals is whether the portal is anonymously accessible for browsing or only for users in possession of user accounts.

Furthermore, the process to adapt an application to the selected portal platform is dependent on the piece of software to be adapted, considerations made when building the software and the techniques used to build it. One of the more important considerations is whether the source code used to compile the application is available for the developers in charge of the adaptation process. Without the original source code, the possibilities to change or rewrite the software are limited and other means need to be found. The quality of the documentation in the original code also matters, especially if

the software was not built by the same developers.

The goals of the integration project can also vary and greatly affect the process to be selected. Considerations include the original UI of the application. Should it be retained? If only the data contents of the software system are kept, then solutions such as shared persistent storage are an option. If the whole application is to be preserved as is and the UI is to be moved inside the portal platform, then shared storage is not a viable option. Finally, the original application can be used only partially if, for example, it consists of multiple layers such as a service layer and a presentation layer. The resulting integrated application can retain the service layer while providing a new presentation mechanism.

## 4.1 Enterprise Application Integration

Enterprise Application Integration (EAI) is a framework to allow different systems to interact with each other by sharing data, connections and processes [11]. The need for the framework arises from the requirement to adapt to rapidly changing markets in corporations [16]. EAI fulfills this goal by allowing different systems to be interconnected in a rapidly changing manner [16]. Basically EAI is a middleware server to which EAI-enabled software may connect to share data. Applications can be added and removed from the middleware server at will [11].

Figure 4.1 shows a generalized view of an EAI system. The middleware server is represented by the box in the middle, and the other services and servers connect to it. The servers can be in different places and connected to the EAI server via various communication channels. The EAI server makes sure that all services can interact with each other within specified parameters such as security constraints.

EAI is no silver bullet to integration problems, however. Security issues are one of the obstacles as connecting systems may belong to different parties and security of sensitive data must be protected. Locating relevant data in response to queries in the rapidly changing interconnected network of services is another challenge. This is why Gable describes most EAI-projects as complex in nature and expensive to develop. [11]

There are multiple systems from multiple developers within the umbrella term of Enterprise Application Integration. A problem with the whole EAI field is the shortage of universal standards and each of the different EAI products implements the same things differently. This means that software

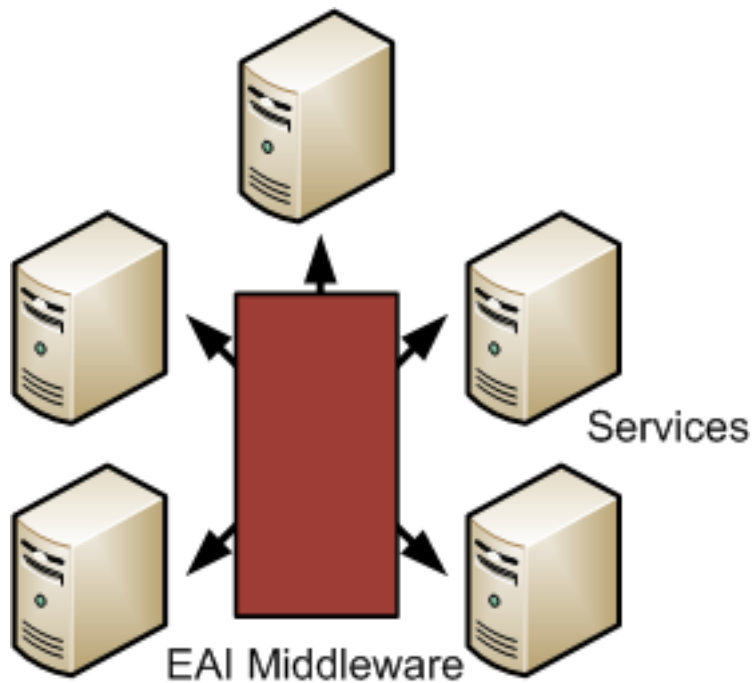


Figure 4.1: Enterprise Application Integration

is usually tied to a certain implementation of the middleware and conversion work is needed if the middleware is at a later date switched to a competing product.

A few of the portal platforms described in chapter 3, such as IBM WebSphere, are capable of working with Enterprise Application Integration that is a part of the same larger product suite. Unfortunately, the same drawbacks apply to these EAI features present in the portals as to the standalone EAI products. Furthermore, the features in these EAI products are more or less restricted to data transfer, and security features are not included in them [11].

Enterprise Application Integration is a viable alternative to the portlet standards or portal-platform-specific APIs especially if the application software supports a EAI without modifications. In most other cases, the added complexity, increased costs and omission of notable features such as user interface integration and security enhancements like single sign on (SSO) make EAI a lot less appealing option.

## 4.2 Integrating custom-built applications

Custom-built applications (or other software) are software products that are developed in-house in response to a specific need. The software is specific to the owner and no support or advice can be found specifically relating to that software. Custom software grants the developer team access to the original source code. Often the original developers are still working on the software or otherwise available so that full knowledge of the product, its features and capabilities and other design decisions is available to the developer team. This means that the software can be easily altered to aid the integration process in various ways.

One integration option is Enterprise Application Integration (EAI), but the drawbacks suggest that other means should be chosen unless the system already uses some EAI solution. In that case, it may be less painful to provide the data through EAI and possibly redo the front end to present it. This method uses the original application only as a persistent storage and validation system for data, which is available for multiple systems through EAI. The cost of integration with this method is high due to large number of new features required from the system.

Single Sign On (SSO) enables an approach where the application is left as is and authentication in the software is modified to support Single Sign On or Identity Federation. The portal will act as the Single Sign On server for the system or be a part of the same authentication realm as the application. The portal will only act as a front end and redirect the user to the correct server once authorization has been verified. This approach obviously has a problem with users being able to access the software directly. Thus, Session management and security measures must be implemented in both the portal and the application. The user interface is completely separate from the portal and any customization to suit different portal environments is much more expensive to develop. On the other hand, the application is not dependent on the portal system and is able to function if the portal is unavailable.

The remaining implementation options are based on moving the application to the portal server and running it as a service for the portal. This is a much tighter coupling with the portal platform but allows access to the portal features which can be used to augment the software. In many cases improvements to the application are already a good reason to rebase the application. These methods are also flexible because they allow developers to choose which features to include and which to leave out.

All portal platforms described in the previous chapter provide a way to add

functionality as portlets. One way to integrate an application to a portal server is to write it as a portlet, possibly usable in a wide variety of portal systems. This is especially convenient if the existing application or software has traditional HyperText Markup Language (HTML) or any extension of HTML as the output format. The presentation layer can then be used with relatively small changes as the portlet user interface. However, it is also possible to design a new interface with more portal-specific functionality such as localization and context awareness.

Adapting the application to function as a portlet will require it to conform to a portlet standard or to be specialized to a single portal platform. If the original application is built with Java programming language, the standard of choice is obviously JSR106. There is no directly applicable portlet standard in any other programming language but Web Services for Remote Portlets (WSRP) standardizes the portlet communication and is thus the best option in most cases where the original language is not Java. WSRP is also usable with Java-based applications. Platform-specific solutions use the Application Programming Interfaces (API) supplied by the portal platform and have additional freedoms and features at their disposal, but they are only usable with the particular platform.

JSR106 portlets have access to many of the portal features and they are executed inside the portal environment. The exact set of offered services differs from platform to platform but the access to these features is standardized. Security and localization features are explicitly standardized. WSRP portlets are executed in a remote system, or possibly in the same system in a different process, and all the data has to go through the communications channels. This offers a less direct access to the actual feature set of the portal platform. The security features in WSRP are also more complex to implement as the communication channel has to be secure and the authentication and authorization has to be done inside the channel by means chosen by the developers.

Portlets enjoy full user-interface integration regardless of the standard. The portlet applications only output HyperText Markup Language (HTML) or eXtensible Markup Language (XML) and the styling and final page composition is done at the portal server in the final stages of the page creation process. This allows the developers to leave the user interface customization to the user of the portlet. Portlets can still have the usual level of interactivity as requests can be transmitted via the communications channels. However if the desired user interface cannot be expressed with these tools or the cost of converting the existing user interface is too high, then portlets

may not be the best choice.

Platform-specific APIs can be used to adapt the software to work with only the specific portal platform. The process of converting software in this manner can potentially be faster than conforming to the standards if the drawback of being tied to a specific portal platform is acceptable. There is no single way to achieve a migration like this as the methods depend greatly on the software to be adapted and the portal platform. The level of freedom the portal platform offers in developing the application or portlet, the APIs present on the system, the features usable from the portal, and the way the portal was designed to be expanded by new software are all factors that influence the process. The freedoms may include things such as access to platform specific third-party libraries, having the output as something else than standard HTML, additional interactive components in the software, broader range of portal features, and better integration to existing features, as well as ability to utilize additional server processes or backend systems.

### 4.3 Integrating commercial or legacy applications

Corporations and other parties also commonly have other commercial software in daily use when starting to build a portal system. The aim is to integrate these other applications with the portal to provide seamless and streamlined experience for the user. The software to be integrated can be intended for public use or it may be an internal application. There are many obstacles to integrating a commercial or legacy application to a portal. For example, different usage scenarios have differing needs for security and different applications have different localization needs.

Commercial and legacy applications are similar in nature and legacy applications can be commercial or custom-built software. There is much less knowledge about the inner workings and processes of the application available to developers who are integrating them to the portal environment. There is usually no direct access to the source code in commercial applications and making extensive modifications to the functionality or opening new interfaces for communication is extremely hard. If there is access to the source code, it may be for viewing only and customizations are not allowed.

Legacy applications are old systems built with old technology. They may be custom software, but usually they do not support any portal platforms. Legacy software is usually treated as black box system providing specified

outputs with specified inputs. This is usually a result of not having the source code available (as with commercial systems), the original developers and their expertise not being available, or the code and program quality being seriously compromised. These reasons make it difficult to create any low-level modifications, including new interfaces to expose the functionalities required in integration process. [8]

The legacy software might be able to function without the user interface if it heavily relies on some form of persistent storage. The persistent storage, such as files on a hard disk or database system, could be directly accessed from a specially developed front end for the portal system. This approach foregoes all the business and feature logic present in the original application by discarding the original front end completely. Only the old data in the persistent storage is used in the new portal application. This means that this method is unusable if any part of the application is vital and needs to be preserved. In cases where there are multiple data sources or they are poorly documented, building a new front end may end up being quite costly. The security constraints on the original software will also be removed and have to be redesigned for the new portal based front end.

Wrapping the original application with a custom-built wrapper software can be used to integrate the application with the portal. Wrapping means developing a custom software to enclose the application and act as a new communications layer for the software. In this way the wrapper acts as a middleware, designed to interpret data streams to and from the application in a format that conforms to the portal expectations. This method has limitations based on the original features of the application, for example the ability to export data from the application. Applications that are meant to only be used on a desktop system usually do not lend themselves well to wrapping. Applications that work with character based inputs and outputs (such as eXtensible Markup Language) work well as these formats are much easier to translate than bitmap images. Depending on the original software the wrapper may create a new portal-based user interface that passes through the selections to the original software or it may just present the original user interface in cases where it is possible.

Security is another concern for the wrapper software. The actions the user can perform with the software need to rely on the user who is logged on the portal, but the original software may have its own authentication and authorization system. The wrapper needs to bridge these two in such a way that security is not compromised under any circumstances. Localization and support for multiple devices need to be handled in the wrapper if the original

software provides no such support. Integration with the portal user interface works well through a wrapper as the wrapper can provide the portal platform with a markup language that the platform can style according to the design.

It is also possible that the software being integrated is a web service or a web application to begin with. In this case the need for wrapping is reduced as the output from the software is already in a format the portal supports. Unfortunately the other drawbacks still apply. Security system needs to be bridged to the portal security system. In some cases the web application utilizes the web servers authentication methods which may be used from the portal as well. This requires the application and portal server to be situated in the same system. Interface can be customized by the portal server as the original application only provides the markup. The user interface can be superficially customized as long as the communication to the application is unaffected. As an additional benefit in this design the portal and application are not tightly coupled to each other. Other features from the portal are usually unavailable, because the data from the portal is unavailable to be used by the application.

Enterprise Application Integration is an option as well, but the cases where it can be utilized are severely limited. The original legacy or commercial application needs to support the EAI system in question to be integrated. In case of legacy systems, they most likely do not support any modern implementations of EAI and are unable to function with them. Commercial systems might support some EAI implementations and if the implementation supported happens to be the same one that is supported by the portal, then EAI is a good choice. Lack of access to source code makes implementing EAI directly in the application hard. Wrapper software might be used to get around the need for direct support in the application but the previously explained drawbacks still exist. Furthermore, if a wrapper is to be used, one could just as well integrate the software directly to the portal.

The last option is to leave the software as is and access it by a redirect from the portal. This only works with web applications as desktop software cannot be referenced from the portal page. This method is the last choice as it takes no advantage of any portal features and, in the worst case, the application will require a separate authentication process when it is accessed. The interface will be unchanged and it cannot take advantage of portal customizations or read localization information from the portal system. Some of these disadvantages may be remedied or mitigated but the other implementation options should be considered first.

### 4.3.1 Single Sign On with legacy applications

As discussed earlier in section 3.2.1, Single Sign On (SSO) is major feature in most portal systems and one of the motivations behind moving applications to a portal platform and developing new products for this platform. Unfortunately, applications need to support the particular SSO system for them to take advantage of the service. Legacy applications rarely have such technologies built into them and as such other approaches have to be used in order to utilize SSO. If the software is integrated into the portal as a portlet by using a wrapper or any other previously described method then SSO will work directly from the portal server. However, if the application is standalone and will be accessed by a redirect from the portal, or if it is a desktop application and the portal authentication needs to work with it, then an another method will have to be selected.

The existing application may use the web server as an authentication source if it is a web application or a Lightweight Directory Access Protocol (LDAP) based lookup if the application is a desktop application. Depending on the type of the web server and its capabilities it may be possible to authenticate to the other web application by authenticating to the web server at the same time as to the portal. This works only if the two services share the web server because this type of authentication cannot be done remotely. Desktop applications can use a similar method where logging into the portal also logs the user into the directory behind the LDAP query. Now when the application queries the directory the correct user credentials are returned and access is granted. This is a difficult task and specialized systems are required.

A wrapper can be used to interpret security queries to the Single Sign On server present in the portal if the legacy application has a security system that supports external user directories or authentication authorities (AA). The wrapper will take a query to the AA by the application and deliver it to the portal SSO server instead. The SSO server will check the logged in user and return credentials belonging to that user. The wrapper will interpret the response and deliver it to the application. Problems may arise if the two security systems are too different or poorly documented. Another problem is that the portal server needs to know who is trying to access the legacy application so it can return the correct credentials. There may be multiple users accessing in the application through the portal security at the same time and correct credentials need to go to each instance of the legacy application.

The last method is a low-level emulation of the log in process. Unfortunately this is only applicable to web services. It means that the portal server will

emulate a user logging into the legacy application by sending the relevant HyperText Transport Protocol (HTTP) messages and by storing the answers it receives. These answers will include a session key that identifies the user session that was created to the application. This key is passed on to the real user when he or she is redirected to the application. The session key will link the user to the existing session and no log in is needed. This method has several drawbacks that make it very difficult to implement in practice. The security system in the original application may check that the requests come from the same source during the session, which is not the case if the portal server does not act as proxy. Another problem is that to allow automated log in the portal server has to know the credentials to the application and this may be a problem if the server and application are controlled by different parties. Finally, the session key cannot be handed to the browser automatically in most modern browsers as the portal will not be able to set HTTP cookies for another site.

# Chapter 5

## Case study: Adapting ASP.Net application

This chapter describes a project to migrate an ASP.Net application to be run on a portal platform as a case study. The project is commissioned by Analyse Systems Finland Oy.

### 5.1 Introduction

#### 5.1.1 Analyse<sup>2</sup> Galaxy

Analyse<sup>2</sup> Galaxy is a web application powered by ASP.Net server backend. Adobe Flex has been chosen as the user interface library for the application. Flex is based on Adobe Flash and executed on the client computer. The application is executed standalone in an Internet Information Services (IIS) server. It relies on the web server and the web framework of ASP.Net for providing services, for example, web session handling and access rights to web server directories and files.

The server backend is composed of C# code along with some third-party libraries like the Object-Relational Mapping (ORM) library. The front end is Flash coded in ActionScript and MXML markup language that is compiled into Shockwave Flash (SWF) binary files during the build process.

Figure 5.1 describes the current architecture of the web application. The figure assumes that the browser on the client hardware has transferred the Flash bytecode to the client to be executed successfully. After this initiation

step, the Flex UI on the client machine communicates with the Galaxy server component running in the IIS Web Server using HTTP-connection (HyperText Transport Protocol) via specialized Adobe Messaging Format 3 (AMF3) messages. The server then provides information for the Flex UI by acquiring it from a number of background services, such as databases.

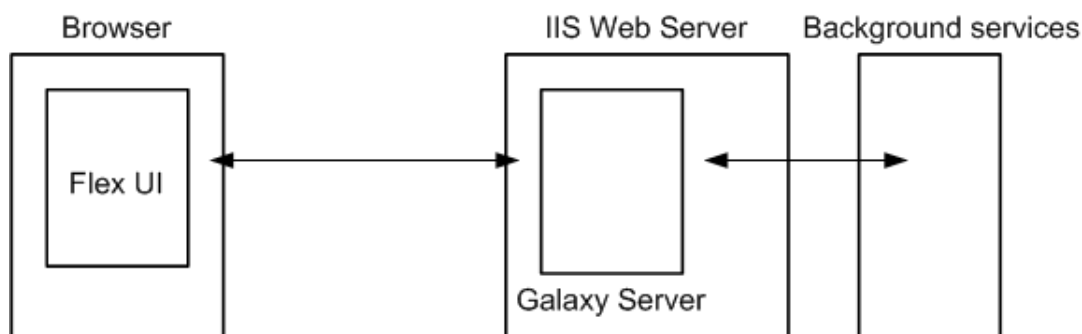


Figure 5.1: Architecture of Analyse<sup>2</sup> Galaxy

### 5.1.2 Security

Security in Galaxy is built on users and companies. User is an object representing one particular user of the software and a company is a collection of users. Users are identified by their unique usernames and the password connected to the user account. The architecture is built to be modular and subsystem implementations can be swapped easily to provide different or additional functionality. For example, the user management subsystem is easily replaceable.

Passwords are not stored in reversible or human-readable format anywhere in the system to prevent accidental disclosure or administrative misconduct. Care has also been taken to minimize most prominent web application security risks such as SQL injections [12]. After logging in, the credentials are stored in the HyperText Transport Protocol sessions and last until the user logs out or the session expires. Galaxy will log in the user automatically if valid user credentials are found in an active session when starting the application.

Galaxy has per-user settings stored in the relational database assigned to it, which also include access rights to specific functionalities in the product. Permissions, defined as roles and used in Role Based Access Control (RBAC), can be assigned and deassigned on the company or user level. Access to

software is granted by having an entry for a user in the database along with an entry in a separate directory that is queried with Lightweight Directory Access Protocol (LDAP). This presents an external dependency on another server in the security architecture.

Permissions are checked whenever a remoting call to the server is made from the client or when the client tries to download any Galaxy-relevant file. Coupled with the module-based design of the user interface, this feature assures that compiled bytecode cannot be downloaded for unauthorized modules. The same principle applies to localization files. A user should not be able to access any resource he or she has no authorization for and additionally the user should not be able to ascertain that the requested functionality even exists in the system.

Finally, logging is in place for various activities pertaining to security. Logins, logouts and incorrect login attempts are all logged with information regarding the source of the request. These can be used to measure the usage of the system or to track who has logged in and from which Internet Protocol (IP) address.

At the very least, the same precautions and functionalities regarding security should remain in place even in the migrated system, augmented by the services provided by the portal server where possible.

### 5.1.3 Motivation

Galaxy is a part of larger product portfolio and to ensure smoother customer experience all the services and products offered to the customer should be located in a single, customizable location and require only one sign-on process. The product portfolio should look coherent and unified and thus use a single style in its user interfaces. Finally, adding new services to the portfolio should be possible and each customer should have a different set of products available to them. To achieve all this, an enterprise information portal is ideal.

Customizing user interfaces to the needs of the user and aggregating data from different sources is one of the main features of any web portal. Customizing the visual appearance of the content and the user interface is also a feature found in many portal platforms. Often, the visual look of the interface is defined in a different location from the actual business logic and written by different developers. Thanks to this design the whole portal may be customized and themed without any modification to the components or portlets.

Single sign-on allows the user to log in to the portal once to access all of the applications on the portal platform. Third-party services offer functionality so that the user information can be retrieved from the client machine without having to provide any credentials manually.

Role-based Access Control allows the portal to display and hide relevant portlets or user interface components depending on the roles the user has in the portal environment. These roles are also centrally managed in the portal system so that every application does not have to contain a separate database for users and their credentials. Many other user properties can also be used to finetune the application to match the needs of the customer. Such settings include, for example, the preferred operating language and preferred font size.

Finally, portal systems can incorporate new applications with relative ease and the new applications inherit all the items listed in this section. They will benefit from the sign-on and can be themed appropriately and will provide a seamless experience for the customer. As an added benefit, third-party applications that match the portal server APIs or standards will receive the same benefits.

## 5.2 Portal software

The portal software selected for this project was Microsoft SharePoint. The most obvious reason for the selection is that the server component of Galaxy is written in a .NET language, which can be directly used in conjunction with SharePoint. The SharePoint API can be easily used from C# code. It was also seen that the portal server offers all the features required in chapter 5.1.3 while not supporting JSR-106.

The Galaxy server code could also have been ported to Java because it does not have many bindings to the ASP.Net web libraries and is mostly self-contained. This would have allowed its use in any Java-based Enterprise Information Portal. Also, Galaxy already has a message-passing interface for communication with the user interface, which could be leveraged to make the server a Web Service or to conform to the WSRP specification.

Figure 5.2 describes the planned architecture after the software has been successfully migrated to the portal. The Flex user interface component has been removed and there is no need to run Flash bytecode on the client machine. As a replacement, a Sharepoint user interface generates the markup for the content and queries the same Galaxy server process that in turn acquires the

necessary information by using various background services. Even though the server process is already in a compatible language, it also needs to be updated to take advantage of the server application programming interface (API).

All security features rely on the SharePoint server to provide user credentials and the application is not usable outside the SharePoint environment. Login and logout functionalities are removed from the application and left for the portal to manage. User-specific properties have been migrated to the SharePoint user profile as much as possible to promote interoperability with other services. Roles declared in the Galaxy database are migrated to be used from the SharePoint role system.

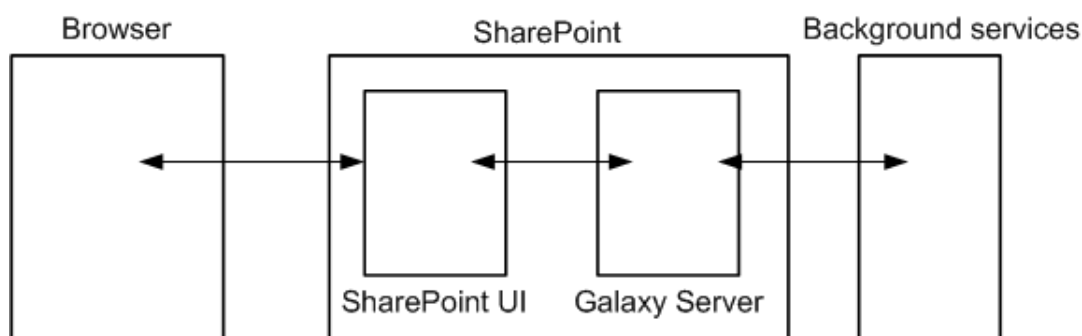


Figure 5.2: Planned architecture of portal-ready Analyse<sup>2</sup> Galaxy

## 5.3 Outcome

### 5.3.1 Embedding Galaxy within SharePoint

Gaining access to the features of the portal system from within the application being developed is done via a specialized Application Programming Interface (API) unique to the server platform. As seen previously, Java-based portals commonly use JSR-106 [1] and portals based on other technologies use proprietary mechanisms or WSRP. In this case, retaining the original server code was a priority to minimize changes to the business logic of the application. WSRP would have required massive rewrites of interfaces and none of the original user interface would have been preserved. Rewriting most of the application is not possible in most cases and the approach chosen to embed Galaxy in the SharePoint server was one where almost no rewrites of the business logic or presentation layer were necessary. An additional benefit

from keeping the user interface mostly intact is that Galaxy is not tied to SharePoint and can be deployed as a standalone application or as a part of a SharePoint Portal server.

Web pages used by Galaxy to transport the Flash bytecode to the client machine are in the ASP.Net programming language, which is also used as one the possible languages for developing features in the SharePoint server. The programming interface of the portal can be accessed from the web pages when they are served from inside the SharePoint server or, in other words running inside the SharePoint web context (SPWeb Context). Initializing the Galaxy server process with access to the SPWeb Context allows the C# server code to acquire information from the portal server and pass data back to it.

This solution also reduced the work of rewriting all of the user interface in SharePoint web parts and instead allowed a leaner solution that still uses the Flash bytecode as a front end to the software. The front end is not executed within the portal but instead on the client machine, as in the earlier architecture. However, due to the Flash object residing in a web page from the portal system, some of the styling is still possible to make the resulting front end blend in with the portal.

The Galaxy server process is executed within the same web server process as the SharePoint server but the Galaxy server process has no direct access to the SPWeb Context. The context is passed to the server process from the initialization of the web page that embeds the Flash object. Code describing this task can be seen in listing 5.1. Additionally, the presence of SPWeb is also indicative of Galaxy being run inside a SharePoint Server context. This is used to probe the environment for the presence of SharePoint without having to explicitly specify it in a configuration file or to develop a specialized build for separate environments.

The user interface had to be modified due to changes to log in functionality. Old log-in window was used also to select the language for the session but, due to the SharePoint integration removing the need for a log in window, the language selection had to be moved. SharePoint Server can be queried for the user's preferred regional settings [17], which were used to select a default language for the user instead of a fixed default selection.

Integration with other products in the same portal environment is also a desired feature and, to this end, a mechanism to partially control the behaviour of Galaxy when opening a new instance was added. Other applications can ask for a specific module to be opened for the user by default so that information relating to the originating application can be quickly and easily

```
using System;
using System.Collections.Generic;
using System.Web.UI;
using Microsoft.SharePoint;

// Construct the user provider for galaxy in OnLoad of the
//index page of the application. This page is used also when
//the application is hosted in the portal so this will be
//called before any access to the application.
protected override void OnLoad(EventArgs e)
{
    // SPContext may be null if Sharepoint Session doesn't exist.
    if (SPContext != null && SPContext.Current != null)
        SharePointUserProvider.ConstructSharepointProvider
            (SPContext.Current.Web);
}
```

Listing 5.1: C#-code to initialize Galaxy with the SharePoint Context

acquired.

The default user interface of Galaxy is mostly unchanged. The page that serves Galaxy to the user is however served from inside the portal environment and will style itself to the style of the site. This means that most of the basic controls, such as session controls, and access to personal preferences are in view despite being outside the Flex user interface. Other components can also be shown continuously, such as service messages for notifying the customers about possible downtime.

Overall, all the components of the integration are to provide a seamless experience for the user and tie in Galaxy to other products already running inside SharePoint Server. By bringing the SharePoint Portal frame to normal Galaxy, it blends in better with the portal while still not being out of place if used standalone without the portal. Ability to interact with other products through parameters while the communication is secured through the portal server further tightens the coupling between the products.

### 5.3.2 Converting security subsystem

SharePoint offers many security benefits for an application and those benefits were the main motivation for moving Galaxy to SharePoint, as was discussed

earlier. SharePoint Server provides Single Sign On (SSO)-functionality and provides its own user and group system. Furthermore, it is not necessary to store the user inside a HyperText Transport Protocol (HTTP) Session or any other place inside the application as it can be queried from SharePoint Server whenever needed.

Using the SharePoint context is in many ways similar to using the HTTP Session for storing the logged in user information. SharePoint abstracts away the need for the application to know where the user information is stored. It can simply be retrieved whenever needed and SharePoint will handle situations where the user session expires or his or her privileges are revoked while using the application [17].

Galaxy accesses the logged in user through a user provider, which can simply be swapped for another implementation relying on SharePoint. Each environment can choose a specific user provider that is selected when the program is executed.

Listing 5.2 shows key parts of the SharePoint user provider module that is initialized in the index as seen in listing 5.1. This code generates a principal entity containing the user identity, basically the user name without the domain identifier if one is present, and all groups in the site context he or she belongs to. This context provider is also responsible for determining whether SharePoint is available as seen in `IsSharePointAvailable`. The rest of the methods are supporting the creation of principals.

The real origin of the users and groups is masked from the application and it is possible to utilize the many different authentication methods available in the portal software. One of them is Lightweight Directory Access Protocol (LDAP) as it is used in Galaxy without SharePoint [14].

Code Access Security (CAS) is a .NET framework feature to control access to specific parts of the application. The developer tags methods with attributes that define the restrictions on entering that particular method. Alongside predefined constraints, such as the principal permission which requires the executing thread to have specified permissions, the developer can create customized constraint attributes if the predefined ones are not suitable for a given scenario. [22]

Galaxy used principal permissions before integration but, with SharePoint, the executing thread does not contain the required information and thus a customized constraint is needed. However, this constraint should be designed such that it works in both SharePoint and standalone deployments. This constraint, Galaxy permission, uses the same syntax as the principal permission

```
public class SharePointUserProvider : IContextUserProvider
{
    private static SharePointUserProvider _instance;
    private SPWeb _web;

    public static void ConstructSharepointProvider(SPWeb web)
    {
        Instance._web = web;
    }

    public static bool IsSharePointAvailable()
    {
        return (Instance._web != null);
    }

    public static SharePointUserProvider Instance
    {
        get { return _instance ?? (_instance =
new SharePointUserProvider()); }
    }

    public string GetUserName()
    {
        if (_web == null || _web.CurrentUser == null)
            return null;
        return _web.CurrentUser.LoginName.Substring(
_web.CurrentUser.LoginName.LastIndexOf('\\') + 1);
    }

    public IPrincipal GetPrincipal()
    {
        var gi = new GenericIdentity(GetUserName());
        var gp = new GenericPrincipal(gi, (from SPGroup
grp in _web.CurrentUser.Groups
select grp.Name).ToArray());
        return gp;
    }
}
```

Listing 5.2: Key parts of the user provider module

for easy migration. The constraint also relies on the .NET security architecture and uses principal objects for the user privilege information, but unlike principal permission, it uses the Galaxy-specific user provider system to retrieve the principal. As discussed earlier, the user providers all provide the principal in the same format and it can be then validated against the required roles.

Partial C#-class of the resulting constraint is shown in listing 5.3. The class implements the two interfaces required for a class to be a custom Code Access Security constraint. These constraints can be combined as can be seen from the Union and Intersect methods. The actual validation is performed in the Demand-method by comparing required role names to roles present in the principal. Raising a security exception will halt the execution.

Listing 5.4 shows how these constraints are used in practice to prevent users without certain privileges from accessing service methods in the server process. The attribute in the code snippet requires user to have administrator or user role to be able to access ServiceMethod.

A user performs the login procedure to establish his or her identity to Galaxy which in turn stores the generated identity. Similarly, after using the software the user logs out and the identity is removed from the server and it can no longer be accessed without logging in again. These functionalities are delegated to the SharePoint Server when running inside it. This means that the interface for logging in and out is not displayed to the user. Instead, the identity has been established to the portal server before accessing the application. Log out is also handled by the portal server and the controls for logging out are hidden from the Flex interface when executing inside SharePoint.

To bypass the log in procedure in Galaxy, a system was needed where an existing user session could be used if one existed before launching the application. This is useful without SharePoint present as it allows for faster log ins if the previous session is present. If SharePoint is available to Galaxy, it is necessary to request user data from the SharePoint Single Sign On system.

The sites in the web server that are used to access the Flash bytecode objects and modules relevant to it also contain user identity checks so that the user cannot download the bytecode modules or localizations he or she has no access to. These checks essentially perform the same check as the Code Access Security (CAS) constraint shown in listing 5.3 but are configured explicitly in the code instead of method attributes. These were also migrated to use SharePoint as the source for roles as the files inside SharePoint Server layout directory are public by default.

```

[Serializable]
public sealed class GalaxyPermission : IPermission, ICloneable
{
    private List<string> _roles;
    private readonly IContextUserProvider _provider =
        (SharePointUserProvider.IsSharePointAvailable()) ?
        (IContextUserProvider) SharePointUserProvider.Instance :
        new HttpContextUserProvider();

    public IPermission Intersect(IPermission target)
    ...

    public IPermission Union(IPermission target)
    ...

    public bool IsSubsetOf(IPermission target)
    ...

    [MethodImpl(MethodImplOptions.NoInlining)]
    public void Demand()
    {
        var principal = _provider.GetPrincipal();
        if (_roles.Any(principal.IsInRole))
        {
            return;
        }
        throw new SecurityException("Security_exception");
    }
}

```

Listing 5.3: Custom Code Access Security (CAS) constraint

```

[GalaxyPermission(SecurityAction.Demand, Role = Roles.Admin)]
[GalaxyPermission(SecurityAction.Demand, Role = Roles.User)]
public int ServiceMethod(int parameter)
{
    ...
}

```

Listing 5.4: using custom Code Access Security (CAS) constraint through attributes

### 5.3.3 Third party libraries

Galaxy uses some libraries developed by a third party to provide functionality not tied to any other feature. These include exporting data in a variety of formats, such as Microsoft Excel documents, and facilitating communication between the Adobe Flex user interface and the server code running inside Internet Information Services (IIS) web server.

Libraries that perform a specific task without needing outside resources such as the exporter library do not need any modifications to function after migration is done apart from file system paths. In SharePoint, one has to use the paths inside the deployment directory. However, libraries that are not as isolated may create problems as is the case with the FluorineFX library that provides remoting calls between Adobe Flex and C#-code. This is accomplished by a native module loaded into the web server and then by using specialized Gateway-webpages as communication channels. These communication channels are defined by the path where the Gateway resides in the served web site. SharePoint Server makes it hard to find an absolute path and, for this reason, modifications were needed to the open source library to tie the communications channels to correct paths. The final version of the portal application works on unmodified FluorineFX library.

The problems caused by the third party libraries not functioning correctly when migrated inside SharePoint were still minor and fixable compared to the benefits from using those libraries and not having to implement the features in the software itself. Furthermore, every other library on the project worked without modifications apart from the FluorineFX communications library.

### 5.3.4 Deploying

Deploying software is the act of publishing the current state of the software. Deploying is needed constantly during software development process. Usually software is deployed first for testing purposes, often many times and to many different locations to facilitate many different stages of testing [13]. After testing the software is deployed to the production environment or environments for endusers. For this reason, it is critical that deploying the software is easy, automated and minimizes the risk of errors. Complex procedures can take a lot of developer time. The Galaxy project has multiple testing environments and multiple production environments with differing configurations and deployment is done with a build script.

Deploying Galaxy is done by copying compiled binaries to a web server to

be served and configuring the Galaxy configuration files for the environment and the web server itself. The former can change over time and can be copied to the server along with the binaries. The web server configuration is static and can not be copied during deployment and must instead be done by hand at environment creation time.

Galaxy is only ran inside the SharePoint Server and does not have any dependencies to the actual SharePoint Portal aside from a few static libraries as references. It is accessed through the portal via a site or sites but the actual application runs inside the web server pool for SharePoint. This means deployment can be done independent of any sites or site collections in the SharePoint Portal and same tools can be utilized as when deploying standalone installations.

SharePoint deployment is done via a build script that compiles all the binaries related to Galaxy along with any dependencies. When a build is successfully completed without any errors, the script copies the new files to a specialized `_layouts` -directory under the SharePoint web root. This directory allows the resulting application to be run inside the Sharepoint Context and to be styled according to its master style files. Deployment scripts required few changes and this style of application can be deployed automatically very easily, which is a useful feature, for example, for automated testing.

Automated deployments have the obvious benefit of reducing errors that could otherwise occur in the deployment process. Deployment requires references to be available to the Internet Information Services (IIS) web server in a different location so that the C#-code can load them, and this is better handled by an automatic system in which the developer cannot forget to copy new references to the server.

### 5.3.5 Testing

Galaxy has an extensive array of unit tests designed to test all facets of the application. Unit tests are designed to test the smallest possible blocks of code, units, while replacing the inputs and outputs to that code by mock providers or stubs [13]. Unit tests in Galaxy also test the security features and especially they ascertain that access to methods is restricted according to the Code Access Security (CAS) constraints.

A test environment is set up for Galaxy with Continuous Integration (CI) server running tests against it constantly. This test environment is not located inside a SharePoint server. Automated user interface tests are performed on the Flex interface outside the SharePoint environment but Share-

Point specific features have to be tested by hand.

Unfortunately, testing the SharePoint features with automated systems like the Continuous Integration server is not possible at this point without building specialized tools for the purpose. Functional testing through a browser session to the Sharepoint system is possible and the results are obtained on what the remote browser receives. This method treats the system as a black box, only considering what the inputs generate as outputs. Testing methods that operate closer to the actual user interface can perform more specific assertions and provide more feedback on possible issues.

### 5.3.6 Resulting system

The resulting system is fully functional and resides within the SharePoint Portal infrastructure. Comparing the resulting system to the plan, it is apparent that it is smaller in scale and features. However, most of the features are present in the system and most of the desired advantages for the user were realized. User interface integration is the most apparent omission from the planned system. The resulting Galaxy achieves some parts of the original vision such as partial user interface integration in the form of having the regular controls present when using the system. This makes Galaxy a sort of hybrid-portlet since it still conforms to the definition of a portlet in the broadest sense. It resides in its own window in the portal page and can be shown along with other portlets if so desired.

Granting control for the outside resources to affect how Galaxy behaves also tightens integration with other products. This provides additional value for the user as the user experience is streamlined and also grants the ability to guide users to potentially interesting information in other products.

There was most to gain from the security system and all the planned features were realized. Authorization and authentication have been transferred to the portal in a generic manner and the system could be with other authentication systems with only a few modifications. Users benefit from the single sign on (SSO) functionality and have access to all of their products from one access point.

Figure 5.3 illustrates the modified architecture of the Galaxy version supporting SharePoint portal. Galaxy Server component resides fully within the portal server and have access to features and services provided by the portal platform. The user interacts with the portal, which provides the user interface for the Galaxy portlet. The client can still be anything capable of browsing the web and executing Flash bytecode. The new portal layer should

not provide any additional hindrance for the user as it replaces components from the old user interface.

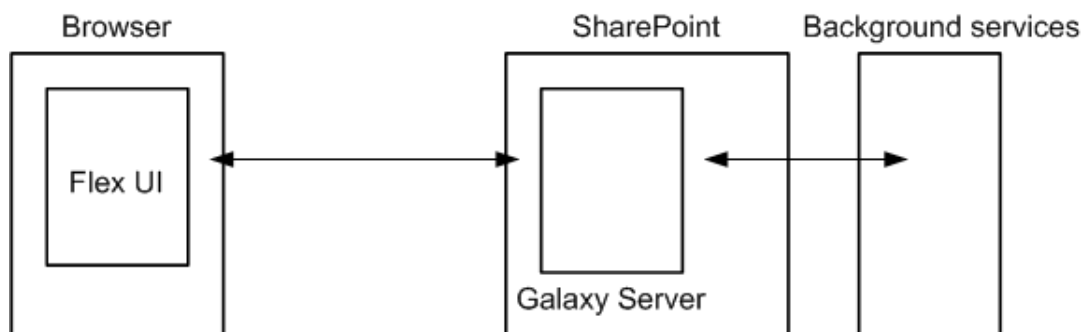


Figure 5.3: Architecture of portal-ready Analyse<sup>2</sup> Galaxy after modifications

User authenticates with the portal and receives the Flex interface only for interaction with the business logic part of the application. In essence, all the superfluous actions that are not tied to the business task at hand have been hidden in the portal as was in specified in the original plan. User preferences are also retrieved from the portal platform and users only need to keep their information up to date in one place.

The final system is reliant on more outside resources since the portal server brings in additional dependencies. Galaxy already required an outside directory server or another form of user management system which now resides within the portal system. Due to heavy calculations and massive amounts of data a separate back end system was required for providing the business data.

For developers, the addition of a supported platform is abstracted and does not pose any additional requirements. The same tools and practices work with the new platform as well as with the old. This is also of paramount importance as the application development will continue after migration and both versions share a common code base.

## 5.4 Further improvements

There are still many unsolved problems in the resulting system and some difficulties could have been resolved better in the course of the adaptation project. Some ideas for future improvements to the software, processes or architecture are presented here.

Galaxy incorporates a server backend that communicates to the Flex user interface via an Adobe AMF3 gateway. The server interface could, however, be used with standardized Sharepoint Visual Web Parts to provide an alternative user interface that has the native look and feel of the portal platform. This would allow developing a new portlet user interface along with the Flex interface for standalone installations, while only having to support one server backend.

Testing tools for an application such as this are practically nonexistent and many types of testing are unfeasible in a real portal environment. Automatic user-interface testing based on a browser is the only real possibility. Unit tests run inside the SharePoint environment would benefit debugging efforts greatly as one cannot be sure that the application performs the same way as outside of the environment.

Galaxy could be developed to use more features from the portal platform. Currently only parts of the portal system are being used while there are other features that could offer additional value for the customer. A search engine is built into the SharePoint platform but it can only search from internal SharePoint data and not from external applications. Addition of application data into the search database would greatly increase the potential of the search engine.

Galaxy exports data present in the system into many formats using a third party library of its own. These exported files are then presented to the user as downloads. This is a good solution but requires the user to have software on their computer compatible with the produced formats. Portals offer web services that can display data, for example Microsoft Excel documents, through the browser. Galaxy could utilise these services from the portal to provide a more fluid and less error-prone experience for the user.

Social networks are becoming more and more important and portals often include collaborative features or home pages for the users inside the system. These could be used by, for example, allowing publication of selected material from the applications to the user's own page or by allowing users to collaborate with their colleagues within the limits of the security features. Galaxy currently allows sharing of generated material with the users of the same company within the limits of the application. A general, portal-provided approach could be used to allow greater freedom in sharing data. Privacy and confidentiality concerns are a problem limiting the usefulness of this feature.

Finally, a dashboard front page could be offered for the user to collect the important information of all the services to give an overview in a concise manner. This would further increase integration with the portal and with

the rest of the product portfolio.

# Chapter 6

## Evaluation

Chapter 5 describes the whole process of adapting an ASP.Net application to work with the Microsoft SharePoint portal platform. This section focuses on evaluating the results of the adaptation process. Section 5.3 describes the system as it is after the process has been completed. The system is evaluated from multiple points of view, namely the user and user experience, technical features and development. These points of view are not separate as success in one area will also influence the others. For example, addition of some technical features will improve the user experience, the development cycle or both. Additionally, the integration process is briefly covered in the last section.

### 6.1 User experience

The goal of the integration process was to provide one entry point for the user via the portal and to provide a seamless user experience by integrating one of the existing web software applications to it. Another major feature to be included in the integration was the addition of the Single Sign On (SSO) system to allow the user to only have one set of credentials and to be logged into every application at the same time. The applications and the portal should also be allowed to collaborate to provide additional benefit for the user. It was also important to make it easy to add additional software to the portal system later. This allows the portal to provide every application and feature the user has access to.

One of the first things the user sees when arriving at the portal front page is the user interface of the portal. Before any actual content can be accessed

the portal has to identify the user. This happens with the user providing the credentials. The user logs into the portal and the SSO system at the same time and the whole process is invisible for the user. The log in interface is standardized and easy to use. The portal front page removes the need for the user to know the exact locators for the services he or she needs and shows the user exactly what services he or she has access to. Users get more value since they cannot forget the existence of rarely used services.

The goals called for unification of product interfaces to produce a seamless experience. Unfortunately this goal was not achieved and the user interface remains separate from the portal user interface. Additionally, some functionalities of the common portal system were added to the application as a frame for the original interface. Users will undoubtedly notice the shift in base technology from basic HyperText Markup Language to Adobe Flex and the differences in usability and conventions that it entails. Graphic designer can make the user interface blend in more in terms of graphic and design choices, but the base functionality cannot be changed to match the portal without rewriting the user interface.

From the user perspective, the integration has succeeded and the user does not have to log in again in order to access the application. In addition, the portal offers some customization features for the user such as personal preferences and multiple user interface themes. The user does have to go through the portal system to access the application, but the new improvements and additions as well as having all products in the same place more than counter the added complexity. In addition, the portal site can be used to distribute information to the users of the service. Maintenance breaks and other scheduled downtimes can be displayed more easily on the portal front page allowing the information to reach higher percent of the user base.

The goals in collaboration are also partially realized. Invisible for the user, the application will localize itself based on the preferences of the user in the portal. The application also supports incoming requests from other applications in the portal system. Other applications or the portal are allowed to request the application to open to a certain state so additional information can be offered to the user. Previously, the user had to do this manually by keeping in mind the relevant details.

Overall, most of the additions are invisible to the user and will grow in importance as the portal service is developed further and new applications and features are added.

## 6.2 Technical features

Technical goals of the integration project were fully realized. The goal was to have the application running within the Microsoft SharePoint portal server with access to the features and improvements the platform can offer. The security system available in the portal is listed as one of the prime objectives and it was integrated fully with all of the features present and usable. Furthermore, selected methods allowed the software to be executed both inside and outside the portal environment and the development of the product could continue while the integration was in progress.

Section 4.2 offers several alternatives for integrating a custom-built web application into a portal server. It was a necessity to integrate the software into the portal by executing it inside the portal context after the decision to keep the user interface developed in Flex intact. The software could have been hosted outside the portal and only the security system could have been integrated, but this would have left out all the other features the portal can offer. In this regard, the selected integration method was the only reasonable choice considering the goals set for the project and the available work hours.

The Single Sign On (SSO) system is utilized through the portal and not as a separate authentication server. This works well due to the application being inside the portal server context. Further, this ensures that the application is not available without authorization if the portal is under maintenance. The only viable alternative, retaining the old log in system built directly into the application, would forego all security improvements gained from the integration with the portal. Every security feature that was present in the software before the integration process was successfully transferred into the new application.

The software after the adaptation is ready for implementation of new features relying on the portal framework as soon as other applications or user needs require them. The SharePoint Application Programming Interface (API) allows direct access to every feature in the portal, which would not be possible in the same scale if built on portlet standards such as Web Services for Remote Portlets (WSRP).

## 6.3 Development and administration

The Application development process was slowed down as a result of the integration process. Most common development tasks are automated or scripted

but there are still some tasks that need to be performed by hand. This is a problem with the portal server in comparison to more traditional software development models. Notable problems include inability to commit portal sites into a version control repository and thus fragmenting the application.

Testing is also severely hampered in the portal system, as noted in subsection 5.3.5. There are no testing tools available which could run unit tests in a realistic environment. User interface testing tools are able to run tests via a browser session and unit tests can be performed in an environment outside the portal system. In this regard, it is also a blessing that the software functions without the portal platform present. There are some portlet testing tools available and this is a problem with the selected integration method rather than with portal systems in general.

Administration has been simplified and streamlined. All security features have been transferred to the portal security system and the user management contained within the application is not used. This allows administrators to perform security-related tasks, such as granting and removing privileges, directly in the portal itself without editing a database manually. Some of the administrator work could be handed on to non-technical staff.

The application used Role-Based Access Control (RBAC) previously with roles defined internally in the application database. The integrated application uses the RBAC features present in the portal platform to handle role assignment to users. This allows same roles to be used across different applications to control access to applications and features.

Overall, administration has become easier as user and privilege management is centralized within the portal for portal-enabled applications.

## 6.4 Process

The process of integrating the application had no problems and proceeded as planned. The selected integration method has proved successful in providing a working system in the timeframe allocated for the project. The revised goals for the project have been filled for the most part and the system has been tested as working. More time could have been invested to investigating the different portal platforms from the application integration point of view. Furthermore, there could have been a more thorough survey on which features to incorporate to the application. Prototype tests on other integration methods could have also been useful in analysing the possible advantages and disadvantages in relation to this particular application.

The integration process provided valuable experience on how to adapt applications to the portal system and the experience will aid future projects to bring more applications to the portal.

# Chapter 7

## Future work

There are various ways to integrate an existing application to a portal platform and this thesis explores one of the choices available in detail by means of a case study. There are still various approaches to be explored in detail, executed in practice and compared to earlier results to see if improvements can be found. Additionally, the source of the improvements could be examined and verified.

Another point of interest is integration of legacy applications and how they might be integrated in a seamless manner so that the user would not notice the legacy application. This would require incorporating many more of the portal features and unifying the interface to a much higher degree. Multiple options were presented on a theoretical level in section 4.3 but improving the methods presented and creating a proof of concept for the more difficult methods is an interesting continuation of this work.

Exploring the possibility of creating a universal framework for integrating custom built, commercial and legacy applications to various portal frameworks would also be an interesting topic for research. There are many challenges arising from the different types of applications and different portal systems. A potential integration framework would have to focus on a smaller subset of all the applications, for example web applications and services. There is a framework solution quite like this for legacy grid computing applications called GEMLCA [8], but there is a need for a framework with a broader and different focus.

There are many features present in the portals that cannot be easily used with integrated applications, such as the client-side Asynchronous HTML and XML (AJAX) customization present in Apache Jetspeed-2 [5]. Exploring a way to make the features more easily accessible from custom software while

still retaining standard compliance could lead to more robust applications and better user experience.

There were no actual user tests conducted on the prototype integration built in the case study and it would be interesting to examine the actual user benefits of integrating an application to the portal. Which features are most used in the portal system and what are the drawbacks from the integration? This research could be used to develop better integration methods.

# Chapter 8

## Conclusions

This thesis studies the various methods of adapting existing applications to function within a portal framework. First, the two prominent portal portlet standards, JSR106 and Web Services for Remote Portlets (WSRP), were described. JSR106 is only available for Java technology based portlets while WSRP is universally applicable. The standards can be used together to provide more complex portlet arrangements, potentially spanning multiple systems and domains.

The services offered by portals were described and the portals from leading vendors were examined next. It was shown that the major difference between portals is the technology on which the portal systems are built. The portal platforms based on the same technology have few differences with each other and the main differences are in the services emphasized and the availability of support. The Jetspeed-2 portal system from Apache Software Foundation was the only open-source portal that was examined and proved that a fully functioning portal system can be built from freely available components.

Security features were of particular interest. All the portal systems surveyed had user management features built in and had a web accessible security configuration. Most also offered Single Sign On (SSO) functionality to the portlets and applications within the portal system. Role-based Access Control (RBAC) was found to be another notable security feature.

Various integration methods were described for both custom-built, commercial and legacy applications. The application being integrated and the portal platform are major factors in deciding which integration method is the best and every method has some advantages and drawbacks. Enterprise Application Integration (EAI) was found to be an inadequate method for universal integration as all software connected to the EAI system has to support it

explicitly. Additionally, the portal platforms surveyed only supported their vendor specific EAI system.

Retaining only the data storage of the old application was found to be a good solution for cases where business logic and user interface could be discarded completely. Custom-built applications have the advantage of source code being available and, thus, additional customization to conform to the portal platform expectations is possible. Good solutions for custom applications include converting the application to a portlet either by utilizing the portal standards or by embedding the application within the portal by utilizing the portal Application Programming Interfaces (API). Legacy applications cannot be modified and one must resort to less intrusive methods such as developing a custom wrapper software to interpret communication.

A custom-built ASP.Net application was integrated into Microsoft SharePoint Portal as a case study. The goal was to develop a working prototype with SSO and other security features implemented and the user interface fully integrated to the portal server. SharePoint was selected as the portal platform mainly due to the application being written in the same technology as the portal server and because SharePoint had all the security features required of the new system. The integration process also had to deal with third-party libraries as well as testing.

The user interface integration did not succeed as planned. Full integration would have required full interface rewrite which was unfeasible and, instead, it was decided to keep the existing Flex user interface and integrate the application to SharePoint with the well-documented APIs provided with the system. Security features were integrated as planned. All existing security functionality was preserved and translated to work with SharePoint. New functionality was added to the system, such as SSO and RBAC. Both of these features were made to work with the standard tools provided with the portal platform.

It was observed that the integration method selected worked well for this particular application. All integration goals were accomplished after the user interface was accepted to be unmodified. Additionally, this method enabled the application to still function outside the portal environment as a standalone application. Furthermore, testing is possible in environments outside the system. Administration was simplified as the result of the integration. Now all relevant user details can be found inside the portal instead of across several different applications. Overall, it was found that the user of the system gained value from the features included in the integration process.

Disadvantages were found in the lack of testing tools that could test the

software while it was executing inside the portal system. Other disadvantages of the selected method include increased complexity for the user who has to navigate via the portal to reach the application. Inclusion of the portal platform added to the list of dependencies for the system as any flaws in the portal system or the portal site may block access to the application as well. Overall, the benefits from a successful portal integration were found sufficient to continue working with portal integration for other applications.

# Bibliography

- [1] A. SAMPAIO AND A. RASHID. Report on Tools for Web Portal Construction. Tech. rep., Lancaster University, Apr 2005. version 1.0.
- [2] ABDELNUR, A., AND HEPPEL, S. Java Portlet Specification 1.0. Tech. rep., Sun Microsystems, Java Community Process, Oct 2003.
- [3] ALLAN, R., AWRE, C., BAKER, M., AND FISH, A. Portals and Portlets 2003. Tech. rep., CCLRC e-Science Centre, Apr 2004.
- [4] ANDERSEN, L. JSR-000221 JDBC(tm) 4.0 API. Tech. rep., Sun Microsystems, Java Community Process, Nov 2006.
- [5] APACHE SOFTWARE FOUNDATION. Apache Jetspeed-2 Features, 2010. Retrieved from <http://portals.apache.org/jetspeed-2/features.html>, accessed Oct 2010.
- [6] COOK, W. R. Policy-based authorization. Tech. rep., University of Texas, 2003.
- [7] DE CLERCQ, J. Single sign-on architectures. In *Infrastructure Security*, vol. 2437 of *Lecture Notes in Computer Science*. Springer, 2002, pp. 40–58.
- [8] DELAITRE, T., KISS, T., GOYENECHÉ, A., TERSTYANSZKY, G., WINTER, S., AND KACSUK, P. "GEMLCA: Running Legacy Code Applications as Grid Services". *Journal of Grid Computing* 3 (Jun 2005), 75–90 (16).
- [9] FERRAILOLO, D., KUHN, R., AND SANDHU, R. RBAC Standard Rationale: Comments on "A Critique of the ANSI Standard on Role-Based Access Control". *Security Privacy, IEEE* 5, 6 (Nov 2007), 51–53.
- [10] FERRAILOLO, D. AND KUHN, R. Role-Based Access Controls. In *15th National Computer Security Conference* (Oct 1992), pp. 554–563.

- [11] GABLE, J. Enterprise application integration. *Information Management Journal* (Mar 2002).
- [12] HUANG, Y.-W., YU, F., HANG, C., TSAI, C.-H., LEE, D.-T., AND KUO, S.-Y. Securing web application code by static analysis and runtime protection. In *WWW '04: Proceedings of the 13th international conference on World Wide Web* (2004), ACM, pp. 40–52.
- [13] JORGENSEN, P. C., AND ERICKSON, C. Object-oriented integration testing. *Commun. ACM* 37, 9 (1994), 30–38.
- [14] KRAUSE, J., LANGHIRT, C., STERFF, A., PEHLKE, B., AND DÖRING, M. *SharePoint 2010 as a Development Platform*. Apress, 2010.
- [15] KROPP, A., LEUE, C., AND THOMPSON, R. Web Services for Remote Portlets Specification 1.0. Tech. rep., OASIS.
- [16] LINTHICUM, D. S. *Enterprise application integration*. Addison-Wesley Longman Ltd., 2000.
- [17] MALIK, S. *Building Solutions for SharePoint 2010*. Apress, 2010.
- [18] MALTZ, L. Portals: A personal door to the Information Enterprise. Tech. rep., Columbia University, Aug 2005.
- [19] PFITZMANN, B. "Privacy in enterprise identity federation - policies for Liberty 2 single sign on". *Information Security Technical Report* 9, 1 (2004), 45 – 58.
- [20] POLBERGER, D. Component technology in an embedded system. Master's thesis, University of Lund, Sweden, 2009.
- [21] PRECHELT, L. Platforms: A web development platform comparison by an exploratory experiment searching for emergent platform properties. *IEEE Transactions on Software Engineering* 99, PrePrints (2010).
- [22] SMANS, JAN AND JACOBS, BART AND PIESSENS, FRANK. Static Verification of Code Access Security Policy Compliance of .NET Applications. In *.NET Technologies 2005 conference proceedings* (May 2005), UNION Agency - Science press, pp. 1–13.
- [23] TATNALL, A. *Web portals: the new gateways to Internet information and services*, 1st ed. Idea Group Publishing, 2005.

- [24] WEGE, C. Portal server technology. *Internet Computing, IEEE* 6, 3 (may. 2002), 73 – 77.
- [25] X. YANG AND XD. WANG AND R. ALLAN. JSR 168 and WSRP 1.0 – How mature are portal standards? In *WEBIST 2006 Proc. Internet Technology and Web Interfaces and Applications* (Apr 2006), INSTICC Press, pp. 393–399.