

Aalto University  
School of Science and Technology  
Faculty of Information and Natural Sciences  
Degree programme of Computer Science and Engineering

Toni Sallanmaa

# Adapting existing web applications for use in portals

Master's Thesis  
Espoo, XX XX, 2010

**!FIXME Printing date: 18 oktober 2010 FIXME!**

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

School of Science and Technology  
Faculty of Information and Natural Sciences  
Degree Programme of Computer Science and Engineering

ABSTRACT OF  
MASTER'S THESIS

<b>Author:</b> Toni Sallanmaa	
<b>Title of thesis:</b> Adapting existing web applications for use in portals	
<b>Date:</b> XX XX, 2010	<b>Pages:</b> X + 42
<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.)	
<b>!FIXME Abstract text goes here (and this is an example how to use fixme) FIXME!</b>  The beginning of the table will be on top of the page when the abstract has a page of text.	
<b>Keywords:</b> keywords... .. more words ...	
<b>Language:</b> English	

Tekniska högskulan  
Fakulteten för informations- och naturvetenskaper  
Utbildningsprogrammet för datateknik

SAMMANDRAG AV  
DIPLOMARBETET

<b>Utfört av:</b>	Toni Sallanmaa	
<b>Arbetets namn:</b>	Adapting existing web applications for use in portals	
<b>Datum:</b>	Den XX XX 2010	<b>Sidantal:</b> X + 42
<b>Professur:</b>	Datakommunikationsprogram	<b>Kod:</b> T-110
<b>Övervakare:</b>	Professor Tuomas Aura	
<b>Handledare:</b>	Diplomingenjör Matti Hämäläinen	
ett abstrakt hit		
<b>Nyckelord:</b>	nyckelord... .. lite mera ...	
<b>Språk:</b>	Engelska	

Teknillinen korkeakoulu  
Informaatio- ja luonnontieteiden tiedekunta  
Tietotekniikan koulutusohjelma

DIPLOMITYÖN  
TIIVISTELMÄ

<b>Tekijä:</b> Toni Sallanmaa <b>Työn nimi:</b> Adapting existing web applications for use in portals
<b>Päiväys:</b> XX. Xkuuta 2010 <b>Sivumäärä:</b> X + 42
<b>Professori:</b> Tietoliikenneohjelmistot <b>Koodi:</b> T-110
<b>Työn valvoja:</b> Professori Tuomas Aura <b>Työn ohjaaja:</b> Diplomi-insinööri Matti Hämäläinen
Tiivistelmätekstiä tähän
<b>Avainsanat:</b> ... sanoja ... ... sanat jatkuu... <b>Kieli:</b> englanti

# Acknowledgements

TODO

Espoo XX Xth 2010

Toni Sallanmaa

# Abbreviations and Acronyms

FIXME	Correct this
TODO	Still needs to be done

# Contents

<b>Abbreviations and Acronyms</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
1.0.1 Goals . . . . .	2
1.1 Problem statement . . . . .	2
1.2 Scope of the Thesis . . . . .	3
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 . . . . .	7
<b>3 Services offered by web portals</b>	<b>12</b>
3.1 Services . . . . .	12
3.1.1 Single Sign On . . . . .	16
3.1.2 Role-based Access Control . . . . .	18
3.2 Major Enterprise Information Portals . . . . .	19
3.2.1 Microsoft Sharepoint . . . . .	19
3.2.2 IBM WebSphere . . . . .	19
3.2.3 Oracle WebLogic . . . . .	19
3.2.4 Apache Jetspeed-2 . . . . .	19
<b>4 Adapting existing applications</b>	<b>20</b>

4.1	Custom-built applications . . . . .	20
4.2	Legacy applications . . . . .	20
<b>5</b>	<b>Case study: Adapting ASP.Net application</b>	<b>21</b>
5.1	Introduction . . . . .	21
5.1.1	Analyse <sup>2</sup> Galaxy . . . . .	21
5.1.2	Security . . . . .	22
5.1.3	Motivation . . . . .	23
5.2	Portal software . . . . .	24
5.3	Outcome . . . . .	25
5.3.1	Embedding Galaxy within SharePoint . . . . .	25
5.3.2	Converting security subsystem . . . . .	27
5.3.3	Third party libraries . . . . .	32
5.3.4	Deploying . . . . .	33
5.3.5	Testing . . . . .	34
5.3.6	Resulting system . . . . .	34
5.4	Future work . . . . .	36
<b>6</b>	<b>Evaluation</b>	<b>38</b>
<b>7</b>	<b>Future work</b>	<b>39</b>
<b>8</b>	<b>Conclusions</b>	<b>40</b>

# List of Tables

# List of Figures

2.1	Portal page with portlets . . . . .	10
2.2	WSRP structure . . . . .	11
2.3	WSRP used in conjunction with JSR-106 . . . . .	11
3.1	Service-based portal . . . . .	15
3.2	PKI-based complex SSO system . . . . .	18
5.1	Architecture of Analyse <sup>2</sup> Galaxy . . . . .	22
5.2	Planned architecture of portal-ready Analyse <sup>2</sup> Galaxy . . . . .	25
5.3	C#-code to initialize Galaxy to use SharePoint Context. . . . .	27
5.4	Key parts of the user provider module . . . . .	29
5.5	Custom Code Access Security (CAS) constraint . . . . .	31
5.6	Utilizing Custom Code Access Security (CAS) constraint through attributes . . . . .	32
5.7	Architecture of portal-ready Analyse <sup>2</sup> Galaxy after modifications . . . . .	35

# 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. [18]

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. [18]

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 an uniform access point to a user.

### 1.0.1 Goals

This thesis is written alongside a project to adapt an existing web application for it work with a web portal framework. 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 find practices how to adapt existing software for use in a web portal and use the services provided, describe the adaptation of an existing application and evaluate the adapted system according to the practices described. The work should yield a working portal application ready to be deployed, improvements on where the process could be improved and how the integration to the portal could be made more seamless.*

## 1.1 Problem statement

The key question is how to adapt an existing application for the 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 to construct the portal is also an integral part of the question, but is often mandated by the technology used in the application to be integrated.

- What does the customer / end user benefit from the integration?
- How can an application benefit from being integrated into a web portal?
- How portal frameworks differ for the purposes of integrating an external application 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?
- How to make the whole process more streamlined?

## 1.2 Scope of the Thesis

The scope of this thesis includes the studying 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. 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 or 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 for streamlining the process. 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 [18]. 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 [16].

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 with the frontend of the applications being in dynamic HTML with a technique such as AJAX, the server backend can be developed in any language with sufficient bindings for network infrastructure. The languages are still further divided into scripting languages, popular examples being JavaScript and PHP, and compiled languages, popular examples including Java and C#.

Prechelt notes in his exploratory experiment of platform properties that web

development platform evaluation should also include the usable libraries and other code that is found on the Internet. Partly joined with the more social kind of programming that utilizes the Internet is the introduction of more 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 favoring variety in development possibilities. These factors combined make the selection of the development language a meaningful decision beyond the simple preference for a language. [16]

Pretchel's study showed that in an empirical experiment there were no significant differences in producing 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 people in the team and technological expertise are more important than the strictly technical platform properties. [16]

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 standardization. Both of these languages are suited for an extensive range of applications and are easily extensible by libraries that are also usable in portal environments.

Standardization does not offer a clear-cut solution to choosing a language to develop with. Java Portlet Specification (JSR106) is written for Java software and there are a lot of portal servers that support this standard. However, another portal standard exists, 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 runs inside the portal environment is usually called a portlet, but naming standards differ in different languages. A portlet can be a big monolithic project implementing everything including user interface, but it also can be a small reusable component. Building a monolithic portlet resembles a traditional software project, where third party

components or libraries are already included in the project itself. Software constructed as components will be integrated with other components when a designer creates a portal page from the already existing components, possibly integrating 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]

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. In the left side of the picture is the client machine, running client software. The client 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 needs to first 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, for example databases or other systems. 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 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 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 lessen development time. As a downside developing a project as components 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). [15]

## 2.3 Portal standards

There are two standards commonly used in portlet development for web portals [19]. These standards aim to make portlets interoperable between different portal servers, providing that the server being deployed to implements the corresponding 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 Java specification containing specification for portlets developed in the Java language. These portlets run inside a portlet container that is used by the portal server itself. Portal container is given multiple options by the specification on when to instantiate portlets and how to handle them. Portlets gain access to portal services through the container by the specification API and thus can utilize the services for example to identify users. [2]

Web Services for Remote Portlets is a specification aimed for standardizing the interactions between services producing some data of interest for the users and the presentation of this data. Traditionally the web service itself will produce data straight to the user and has to contain presentation logic to allow the data to be read in a clear manner. However, having to develop

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 similar way 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 basically allowing a single portal server to harvest portlets from multiple sources developed with multiple languages and platforms. [11]

WSRP works with three different parts displayed in Figure 2.2. Portlets (1) are the core component generating data and 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. Consumer and producer do not have to be on the same machine and when a consumer locates a portlet it needs to display a portlet handle is given out by the producer to the consumer for further communication with the portlet. 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. [11]

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 to 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. By utilizing architecture such as this it is easy to chain portlets and 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 servers 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 being generated somewhere else in a different system so security measures are in place. Specification suggests document level authentication to be handled with existing web security standards such as Security Assertion Markup Language (SAML) and authentication of producers to consumers and vice versa with certificates using 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 thanks to facilities like authenticated user identity. [11]

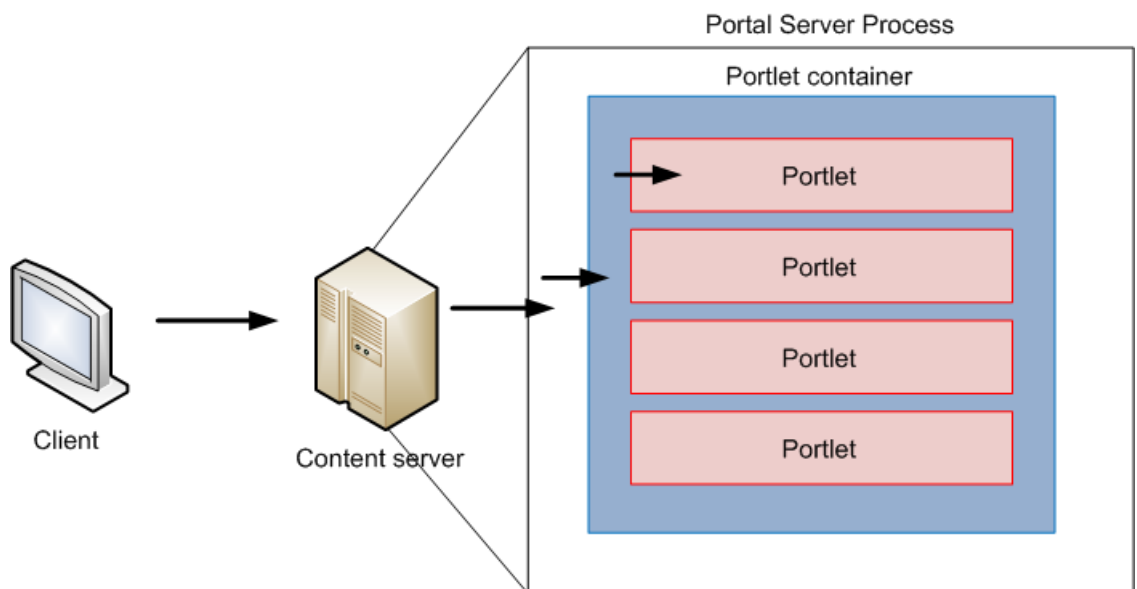


Figure 2.1: Example of a portal page with portlets

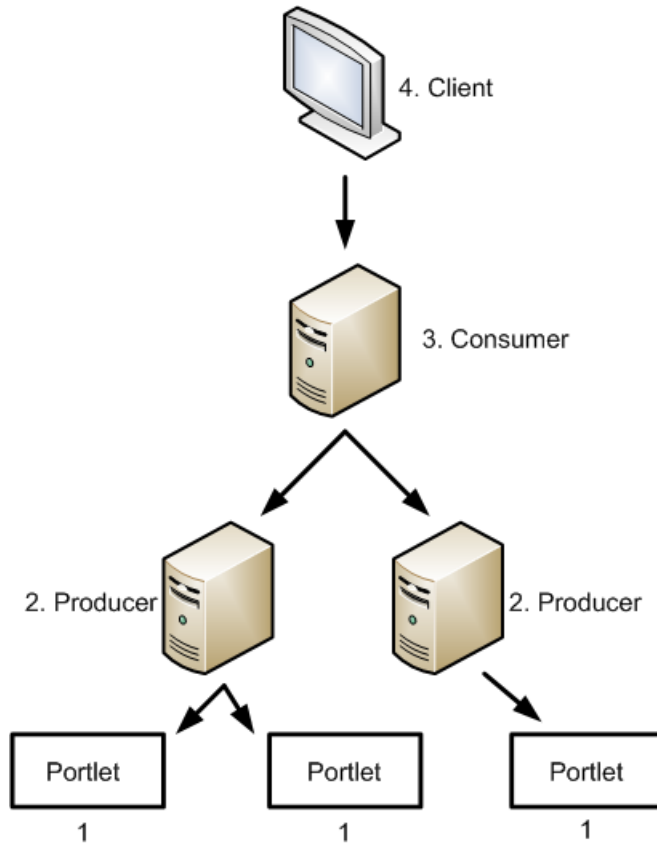


Figure 2.2: WSRP presentation structure

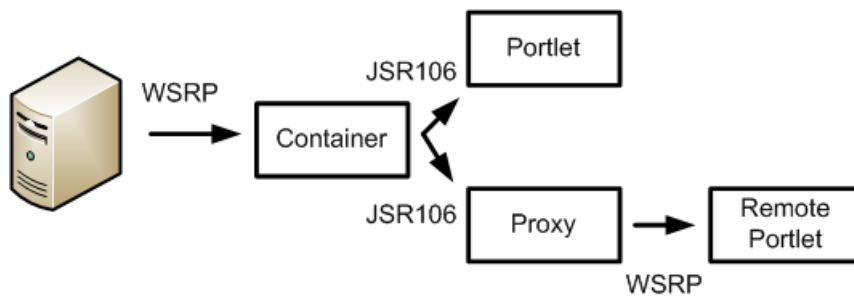


Figure 2.3: WSRP used in conjunction with JSR-106, adapted from Allan et al. [3]

# 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 **!FIXME aspects FIXME!** not related to the technology and software, such as support and community, which are important aspects 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. Portal platform is offering as a service to handle the logic relating to session handling and their validity. 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 binds 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 [4]. 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 handling of the 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 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 to perform tasks such as workflows can also use this system. The portal can handle the queueing and execution of these jobs and provide feedback status information 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 so 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 looked up from the registry with the service they provide and not by the actual implementation of the service.

### **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 to portlets and applications and schedule the accesses to maximize performance. In some instances also used to describe a service where users can 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 being generated at the portal system by other applications or processes. When an event is produced every listening process is notified of the event and is able perform tasks based on it. This is one of the core methods of interprocess communication and is usually implemented through message passing systems and as such is a feature provided by the portal platform. This service is also called notification service.

### **Portal-based collaboration and social 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. Allowing users to have their own personal pages or spaces in the portal is an example of social aspects.

### **Standardized outside content access**

Portal can implement some standard protocols for accessing outside content, such as Simple Object Access Protocol (SOAP) or eXtensible Markup Language (XML) based protocols. This eases the creation of portlets and applications that rely on the content so it is not necessary to implement content retrieval separately in each portlet.

Some of the services described above are much closer to the core of the portal platform than others. It is not feasible to implement certain services, such as messaging service, as portlets or addons to the portal platform while other services would be better served by excluding them from the platform core. Another way to select services and features to be moved outside the platform core is to measure their user base or how often is the feature used in installations of the portal platform. Less used services can be provided as addons 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 and portlets connected to it and it can offer more functionalities by utilizing remote portlets from other systems. The remote portlets are shown as the portlet proxies. User does not directly connect to the remote portlets. [3]

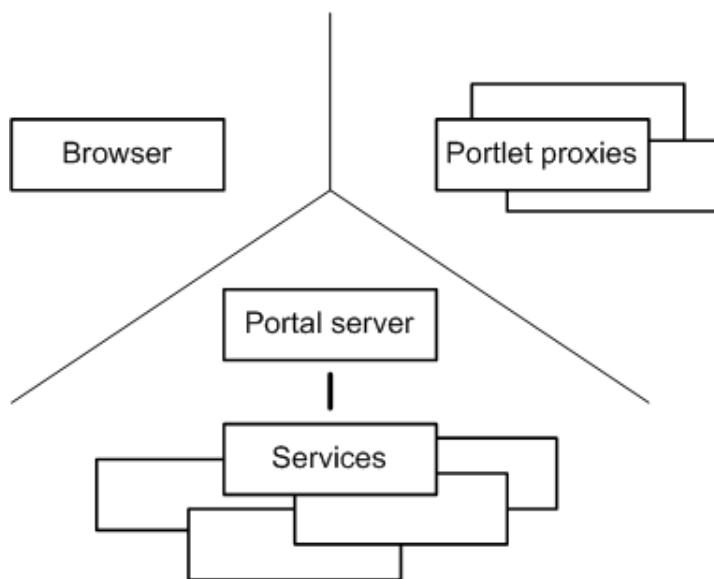


Figure 3.1: Service-based portal [3]

Allan et al. also provide a list of some criteria for selecting a suitable portal framework. A list adapted and expanded from their previous 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 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 consists of 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 in creating good quality portlets. Also the quality of Application Programming Interfaces and other developer facing tools is a factor.

#### **Adherence to standards**

Standards allow portlets to be transferred and deployed to many different portal platforms or 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, drive 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 side, WSRP already allows portlets to be created in any programming language as it provides a communication framework and not an Application Programming Interface (API).

### **3.1.1 Single Sign On**

Single Sign On (SSO) is one of the most prominent security features offered by the portal systems [19]. It 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 [10].

Single Sign On systems, both built-in and external, consist of multiple components. In general, SSO combines identification and authentication. Au-

thorization is handled by the application utilizing SSO. The party or system providing the Single Sign On service must be trusted from the application as the user only provides his or her credentials to that party. The SSO service will then provide identity for the user for trusted systems requesting user identity from it.[5]

Authentication server is the system providing the user identities for other systems and is the part commonly referred to as the SSO server. In essence this system 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. [5]

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 these systems tokens are obtained from the authentication server which are passed onto the servers providing services and these 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. [5]

Figure 3.2 shows a layout of an example 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 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. [5]

The concept of Identity Federation (IF) is related to Single Sign On. In IF the identity is managed at the client and transferred through a network to the Single Sign On system. An example is a portal to which users can log in and the identity is read by the portal security system from their own corporate network. This allows the clients to control the identities and passwords. 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 to use third-party

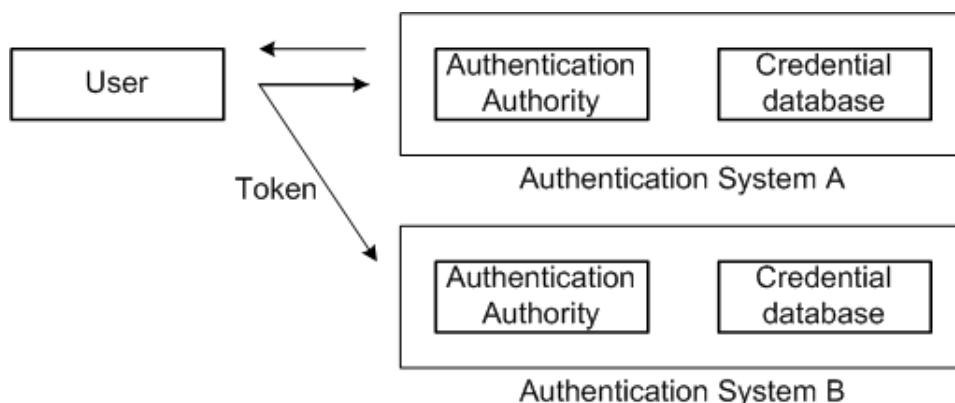


Figure 3.2: PKI-based complex SSO system [5]

authentication authorities. [14].

### 3.1.2 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 users with a role called 'administrator'. Privileges granted to users come in 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 an anonymous user or users who have not logged in. Additional roles granting additional privileges are added to that baseline. [6]

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 accountants have that role is modified instead modifying the permissions of the individual users. [7]

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 the accountant would gain the same privileges as the intern and add some more on top of that. [7]

Due to suitability to be used in a range of environments and widespread use [7] 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.

## 3.2 Major Enterprise Information Portals

The following Enterprise Information Portals (EIP) are described for services provided.

**!FIXME Different tools and methodologies compared by Sampaio et al [1]. Maltz lists some portals and features [13]. Allan et al. list services and portals. [3]. FIXME!**

### 3.2.1 Microsoft Sharepoint

### 3.2.2 IBM WebSphere

### 3.2.3 Oracle WebLogic

### 3.2.4 Apache Jetspeed-2

# Chapter 4

## Adapting existing applications

**!FIXME Write this chapter, these are only notes. FIXME!**

The commonalities from the Enterprise Information Portals described in the preceding chapter are gathered here and practicalities regarding the adaptation of software for use in a portal environment are discussed (such as APIs, injecting the application to portal runtime).

### 4.1 Custom-built applications

Custom-built applications, built in-house or by a third party, grant the ability to modify the source code and adapt the application in various ways.

### 4.2 Legacy applications

Legacy applications do not usually offer access to source code or documents detailing the inner workings of the application and are instead provided as-is. These applications are much tougher to integrate into a portal environment.

# Chapter 5

## Case study: Adapting ASP.Net application

This chapter describes a project to migrate an ASP.Net application to be run on an portal platform as a case study.

### 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 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 a 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.

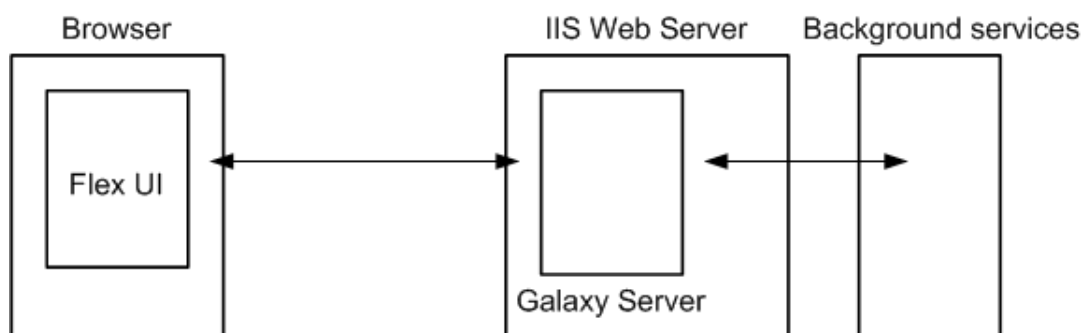


Figure 5.1: Architecture of Analyse<sup>2</sup> Galaxy

### 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, modifying where the user object is stored during usage is one such subsystem.

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 [8]. 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 for 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 a 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 assure 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 usage of the system or 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 being offered to the customer should be located in a single, customizable location and require only one sign-on process to access relevant information. The product portfolio should look coherent and unified and thus use a single style in user interface. 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 to match 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 by different developers so the whole portal may be customized and themed without any modification to the components or portlets.

Single sign-on enabled portal server will allow the user to log in to the portal once to simultaneously log in to all of the applications running 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, preferred operating language and preferred font size.

Finally, portal systems can incorporate new applications with relative ease which inherit all the items listed in this section. New applications will benefit from the sign-on and be themed appropriately and will provide a seamless experience for the customer. As an added benefit third-party applications can be also be used if they match the portal server APIs or standards and they 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 used directly in conjunction with SharePoint. Similarly, originating the same developer, SharePoint has good bindings for the SharePoint Application Programming Interface (SharePoint API) to be 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 usage in 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 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 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 environment. Login and logout functionalities are removed from the application itself and left to the portal to manage. User-specific properties have been migrated to the Sharepoint user profile as much as possible to promote interoperation with other services. Roles declared in the Galaxy database are migrated to be used from the SharePoint role system.

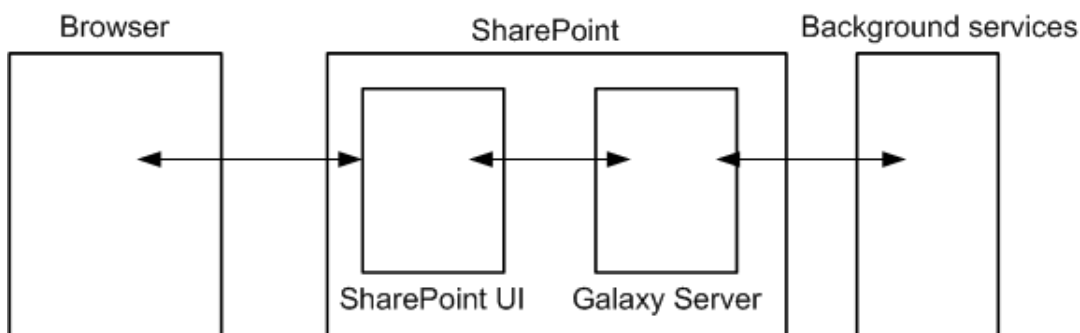


Figure 5.2: Planned architecture of portal-ready Analyze<sup>2</sup> Galaxy

## 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 [?] and others use proprietary mechanisms or WSRP. In this case retaining the original server code was a priority and utilizing WSRP would have required massive rewrites of interfaces and none of the original user interface could be 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 business logic or presentation layer were necessary. Additional benefit received 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 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 utilized by the SharePoint server as one of the possible languages to develop features in. 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). Bootstrapping 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 more lean solution of still using 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 information can still be utilized to make the resulting front end blend in more with the portal.

The Galaxy server process is ran within the same web server process as is 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 figure 5.3. Additionally the presence of SPWeb is also indicative of Galaxy being run inside a SharePoint Server context, so this is used to probe the environment for the presence of SharePoint without having to explicitly specify it in a configuration file or develop a specialized build for separate environments.

User interface had to be modified due to changes to log in functionality. Old log in window was used also to select 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 users preferred regional settings [12], which were used to select a default language for the user instead of 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 information relating to the originating application can be quickly and easily acquired.

---

```
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);
}
```

Figure 5.3: C#-code to initialize Galaxy to use SharePoint Context.

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 most of the basic controls such as session controls, access to personal preferences and others are in view despite being outside the Flex user interface. Other components can also be introduced to being shown constantly, 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 frames to normal Galaxy it blends in better to the portal while still not feeling out of place if used standalone. Ability to interact with other products through parameters while the communication is secured through the portal server further tightens coupling between 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 it's 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 users session expires or his or her privileges are revoked while using the application [12].

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

Figure 5.4 shows key parts of the SharePoint user provider module that is initialized in the index as seen in figure 5.3. 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`. 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 using Lightweight Directory Access Protocol (LDAP) as is used in Galaxy without SharePoint [10].

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 using predefined constraints, such as 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 given scenario. [17]

Galaxy utilized 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 principal permission for easy migration. The constraint also relies on the .NET security architecture and uses principal objects for the user privilege information but unlike

---

```
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.IndexOf('\\'));
    }

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

Figure 5.4: Key parts of the user provider module

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 figure 5.5. 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.

Figure 5.6 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.

An user performs login procedure to establish his or her identity to Galaxy which in turn stores the generated identity. Similarly, after utilizing 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 the interface for logging 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 hidden from the Flex interface when executed inside SharePoint.

To be able to bypass the log in procedure in Galaxy a system was needed where an existing user session could be used for the system 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 utilize user data requested from 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 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 figure ?? 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 publicly available 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");
    }
}
```

Figure 5.5: Custom Code Access Security (CAS) constraint

---

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

Figure 5.6: Utilizing 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 isolated functionality. 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 used. When using SharePoint one has to use the paths inside the deployment directory. However, libraries that are not as isolated may provide problems as is 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 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 with dynamic paths depending on the site being used make it hard to find an absolute path and as such modifications were needed to the open source library to tie the communications channels to correct paths.

Problems caused by the third party libraries not functioning correctly when migrated inside SharePoint were still minor and fixable compared to the benefits received 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 differing stages of testing [9]. After being tested the software is deployed to production environment or environments to be used by the endusers. For this reason it is critical that deploying software is easy, automated and minimizes the risk of errors as complex procedures can take a lot of developers time. The Galaxy project has multiple testing environments and multiple production environments with differing configurations and is done with a build script.

Deploying Galaxy is done by copying compiled binaries to a web server to be served and configuring both the Galaxy configuration file for the environment being deployed to 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 few static libraries as references. It is accessed through the portal via a site or sites but the actual application is ran 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 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 master style files. Deployment scripts required very few changes and this style of application can be deployed automatically very easily which is a very 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 Internet Information Services (IIS) web server in a different location so the C#-code can load them and this is better handled by an automatic system so developer does not 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 [9]. 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 SharePoint 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 with 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 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.

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

Figure 5.7 describes the modified architecture of the Galaxy version supporting SharePoint portal. Galaxy Server component resides fully within the portal server and has access to features and services provided by the portal platform. User interacts with the portal, which will provide the user interface for the Galaxy portlet. Client application 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.

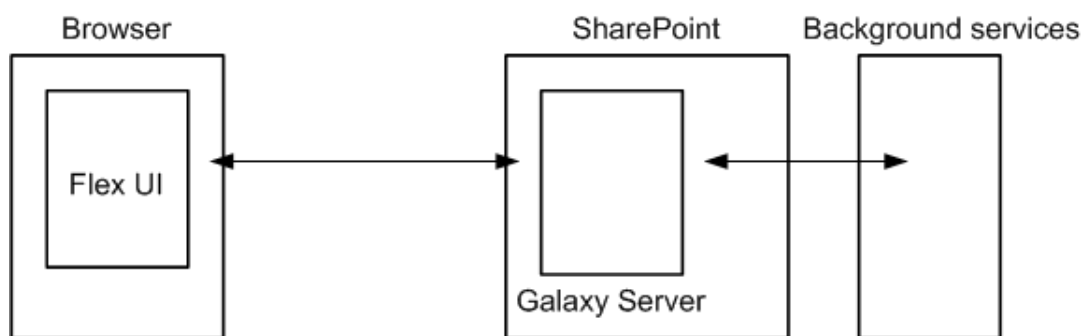


Figure 5.7: Architecture of portal-ready Analyze<sup>2</sup> Galaxy after modifications

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 to be provided through the portal as was in specified in the original plan. User preferences are also retrieved from the portal platform and user information is only needed 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 required a separate backend system for providing the business data is still required.

For developers the addition of a supported platform is abstracted and does not pose any additional requirements. Same tools and practices work with the

new platform as well as with the old. This is also of paramount importance as development will continue after migration and both versions share a common code base.

## 5.4 Future work

There are still a lot of 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 developing 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 to perform in a real environment. Automatic user interface testing based on browser is the only real possibility. Unit tests ran inside the SharePoint environment would benefit debugging efforts greatly as one cannot be sure that the application performs the same way 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 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 formats being produced. 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 to publish selected material from the applications to users own page or by allowing users to collaborate with their colleagues within the limits of the security features. Galaxy currently allows to share 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 that collects the important information of all the services he or she is entitled to and grants 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

Evaluation of the work performed in the case study project. Did the project reach its goals and if it did, how well is the application working in the portal. Furthermore, how integrated it is and how much of the potential of the portal platform could be added to the software and how much potential was wasted? How to further streamline the process and what to do differently?

# Chapter 7

## Future work

TODO

# Chapter 8

## Conclusions

TODO

# Bibliography

- [1] A. SAMPAIO AND A. RASHID. Report on Tools for Web Portal Construction, Apr 2005. version 1.0.
- [2] ABDELNUR, A., AND HEPPER, S. Java Portlet Specification 1.0, Oct 2003.
- [3] ALLAN, R., AWRE, C., BAKER, M., AND FISH, A. Portals and Portlets 2003, Apr 2004.
- [4] COOK, W. R. Policy-based authorization, 2003.
- [5] DE CLERCQ, J. Single sign-on architectures. In *Infrastructure Security*, G. Davida, Y. Frankel, and O. Rees, Eds., vol. 2437 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2002, pp. 40–58.
- [6] FERRAIOLO, 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.
- [7] FERRAIOLO, D. AND KUHN, R. Role-Based Access Controls. In *15th National Computer Security Conference* (Oct 1992), pp. 554–563.
- [8] 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* (New York, NY, USA, 2004), ACM, pp. 40–52.
- [9] JORGENSEN, P. C., AND ERICKSON, C. Object-oriented integration testing. *Commun. ACM* 37, 9 (1994), 30–38.
- [10] KRAUSE, J., LANGHIRT, C., STERFF, A., PEHLKE, B., AND DÖRING, M. *SharePoint 2010 as a Development Platform*, 1st, ed. Apress, 2010.

- [11] KROPP, A., LEUE, C., AND THOMPSON, R. Web Services for Remote Portlets Specification 1.0, Aug 2003.
- [12] MALIK, S. *Building Solutions for SharePoint 2010*, 1st, ed. Apress, 2010.
- [13] MALTZ, L. Portals: A personal door to the Information Enterprise, Aug 2005.
- [14] PFITZMANN, B. Privacy in enterprise identity federation - policies for liberty 2 single sign on. *Information Security Technical Report 9*, 1 (2004), 45 – 58.
- [15] POLBERGER, D. Component technology in an embedded system. Master's thesis, University of Lund, Sweden, 2009.
- [16] 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).
- [17] 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.
- [18] TATNALL, A. *Web portals: the new gateways to Internet information and services*, 1st ed. Idea Group Publishing, Hershey, PA, 2005. ISBN 1-59140-439-8.
- [19] 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.