Different Approaches of Integrating Applications to a Portal Engine

K.M.L. Sureshika, A.P. Pathirage, S. Nirathan, S.P. Mendis and Shahani Weerawarana
Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka

***Abstract*—In a profit oriented business environment professional skills and time are highly scarce and costly resources. Even though a company's top management has the best professional skills, talent and time, if the correct information do not flow to them at the right time, there is a very little chance for the organization to survive in the long run. This is the time where Enterprise Application Integration made a revolutionary change across the business world. It has given the business intelligence for its management to consolidate relevant data for decision making. With the proper identification of this critical problem, many solutions were popped up from around the world. Portal-portlet solution is one such approach .While integrating portlets to portals, there are many different approaches one can follow. Approaches we have discussed in this paper include, using web services as portlets, creating portlets from scratch, displaying a live web interface through a portlet, using off-the-shelf portlets, and integrating complete applications as portlets. Along with the discussion of how to use these five methods, we have also analyzed the pros and cons of the five methods along with working examples taken from the open source project, the Jefe -Software Development Workflow Management Solution (can be found in http://www.jefesolutions.com) which is built according to these different approaches.**

***Index Terms*—Enterprise Application Integration (EAI), Portlets and Portals, Portlet Lifecycle, Portlet Development, Off-the-shelf Portlets, iFrames.**

I. Introduction

In profit oriented large business organizational environments, having business critical data dispersed in many isolated applications, creates a huge competitive disadvantage. It reduces the ability of the top management to foresee the opportunities and threats in them on time to maximize their profits. This was the key reason for the Enterprise Application Integration boom in the recent past. There were many EAI solutions popping up from around the world. Portal technology was one such solution largely adopted by many giant organizations and recognized as a success throughout the world.

According to the JSR 168 portlet specification, portal is a web based application that hosts the presentation layer of Information Systems [1]. It consists of many portlets and according to the same specification portlet is defined as a web component that processes requests and generates dynamic content [1].

There are many open source and proprietary portal solutions readily available for use. Liferay, uPortal, WebSphere, eXo are only some of them. EAI is achieved through integrating many isolated information systems to a single portal by using portlets. Integration here represents both integration between those applications with the portal itself and integration between applications. While open standards like JSR 168 provide integration between applications and portal, other open standards like JSR 286 make sure integration between portlets possible. This provided the ability to introduce integration among applications without modifying their code base. Hence portal solution actually provides more attractive means of integrating the legacy systems with newly blossoming sophisticated business critical applications.

Before analyzing different approaches of integrating applications to a portal as portlets, it is better to first understand what exactly are they, and how they operate. The next chapter of this research paper is dedicated for this topic.

II. Portals and Portlets

A. Portal

As mentioned in the Introduction chapter, portal is defined as a web based application that hosts the presentation layer of the Information Systems. The value of this portal solution lies on the attractive features it provides, while fulfilling the above mentioned key role in EAI. Some of those features are,

--personalization
--single sign on security
--content aggregation
--layout management
--user management
--internationalization

It is the responsibility of the portal to aggregate the fragments generated by portlets which are hosted in its portlet container and provide a complete markup page.

B. Portlet

Even though JSR 168 portlet specification has defined a portlet as a Java based web component that process requests and generate dynamic content, it need not to be essentially restricted to Java technology. There are many portlets developed using PHP and other technologies as well. Portlets are used by the portal as pluggable user interfaces to create dynamic content.

A portlet window has few key components that are been aggregated to create a portlet. Those are,

--Title
--Decorators and Controls
--Portlet Fragment

Portlet fragment carries the markup (Eg. HTML, XHTML, WML) but should not carry any sort of header tags with it. Fragments generated by a portlet aggregate with other portlet's fragments to form a portal page. Fragment generated by the same portlet can differ depending on the user according to differently set user configurations.

Portlet lifecycle is managed by the portlet container of the portal engine. Web clients interact with the portlets using the request response paradigm implemented by the portal engine [1].

C. *Portlet Container*

Portlet container is an extension of a servelet container. Portlet container provides the run time for the portlets on the portal. It maintains the portlet lifecycle by,

--Creation of a Portlet

--Processing of user requests

--Removal and Garbage collection of the portlets

Creation of a portlet includes loading, instantiation and initialization. Loading of classes that are required by the portlet will be accompanied by invoking the constructor that results in instantiation. Initialization will be done by calling the init() method of the portlet, by the container. Portlet container will only call the init() method once and until it is called the portlet will not be considered as active [2].

Once portlet creation is over, portlets wait to interact with its users by using request handling. Portlet container receives requests from the portal to execute requests on the portlets hosted by it. The portlet container is also responsible for passing the responses back to the portal.

Removal of the portlets will be carried out by the portlet container by invoking the destroy() method. The portlet container will not invoke this method until all pending, initializing or processing threads are completed. It will release any resources held by the portlet and de-reference the portlet allowing it to be fetched by the portlet container.

In addition to all above key roles performed by the portlet container, it also caches the portlets and provides storage for portlet preferences.

D. *Interaction*

We have already discussed the key elements in a portal solution and their responsibilities. To get an idea on how they interact at runtime, let's take the example provided by the JSR 168 specification and discuss with more information.

--A web client (Eg. a web browser, a web enabled phone, etc.) makes an HTTP request to the portal. It can be a click on a link, or a submission of a form, or a page refresh, or a selection in controls, etc.

--Portal first receives this request.

--Then the portal is required to determine whether the received request, has an action targeting any of its portlets.

--For this the portal must first identify what triggered the user request. It can be triggered either by an actionURL or renderURL. If it has been triggered by an actionURL, it is usually translated into one action request and many render requests, one per each portlet on the portal page. Else if the user request has been triggered by a renderURL, it is usually translated to many render requests one per each portlet on the portal page. In both cases, if there were cache enabled portlets render requests are not required to be invoked.

--After translating the user request into a set of action and render requests, the portal through the portlet container invokes the hosted portlets, to process these translated requests targeted to each of them.

--First, portlet container needs to invoke all the action requests. Then only the render requests can be processed. These render requests can be carried out sequentially or in parallel without having a specific order of execution.

--These action requests results in changing the state of targeted portlets while render requests generate portlet fragments according to the current state of corresponding portlets.

--When all the portlet have rendered their portlet fragments the portal will once again aggregate the different fragments sent by the portlet container and reconstruct the portal page.

--This resultant completed markup page, including header information, will be then sent back to the web client, as the HTTP response to the earlier HTTP request.

This can be graphically shown as follows using a sequence diagram.

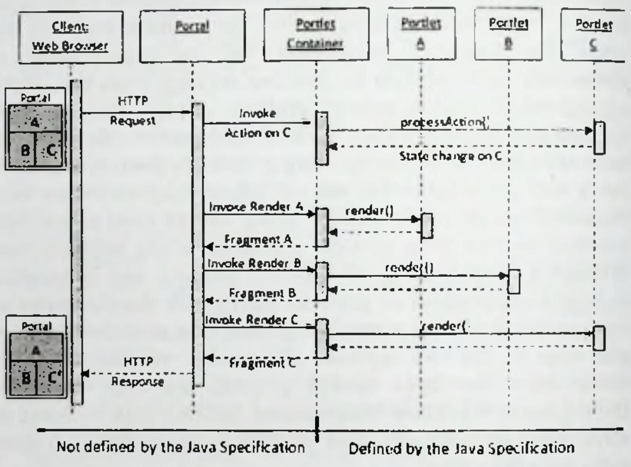

Fig. 1. Sequence Diagram of portal request handling procedure. This simplifies the description given in subsection *D. Interaction* under Portals and Portlets.

III. Problem

If we summarize the problem that we are addressing by this research paper, it is one of the most critical need immerging from the business world to consolidate their most valuable assert, the information in profit making. With the new trend of expanding business over multiple geological locations and product diversification to minimize the risk, it had been difficult to manage the information assert generating at an exponential rate, to filter the required figures to make the right decision at the right time to maximize the profit. It is the development of business intelligence and minimizing (of) redundant work that we address here.

IV. Solution

With respect to the above mentioned problem there are many forms of solutions, developed by computer professionals all over the world in past few decades. Among all those different alternative ways of addressing the problem, we chose the portal solution to discuss in this paper. The beauty of the portal solution lies in its magnificent features that enable maximum personalization while requiring minimal changes to the existing enterprise applications. This contains valuable piece of information.

Portal solution is an excellent means of providing interoperability between isolated applications including legacy systems, reducing redundancy and increasing business intelligence level without compromising the user freedom.

Even if we narrow down our scope to portals and portlets there are many other different ways of integrating applications to the portal framework. From here onwards different techniques used in application integration with portal engines are discussed with working examples.

For this we are using Liferay 5.2.3 as our reference portal engine with different examples that are actually implemented in **the Jefe, Software Development Workflow Management Solution** project (**http://www.jefesolutions.com**) to integrate different open source enterprise applications to the Liferay portal.

A. Usage of Web Services in portlets

Developing and modifying portlets to integrate to a portal is time consuming and costly. Either the portal administrator should recruit a portlet developer or buy off-the-shelf portlets. But Web Services for Remote Portlets (WSRP) standard has simplified the effort required to connect to remote application to a degree, that the portal administrator can select from a rich stack of available web services and integrate them to the portal simply with few mouse clicks [3].

WSRP specification was first introduced by the WSRP OASIS Technical Committee as the WSRP v1.0 in 2003. The main intention of introducing this standard was to define a set of interfaces for accessing and interacting with interactive presentation-oriented web services [4]. Here the presentation-
oriented web services means that the web service provider provides the markup for the presentation layer instead of just providing raw data.

To understand this concept better, let's look at how local portlets and web services are integrated to a portal engine.

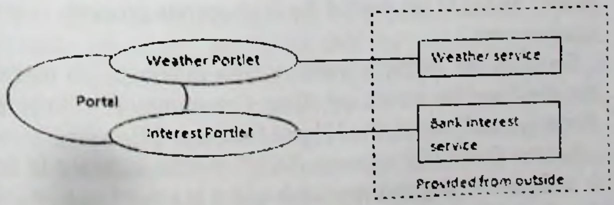

Fig. 2: This figure shows two separate local portlets integrated to a portal. They generate content according to the data they receive from outside

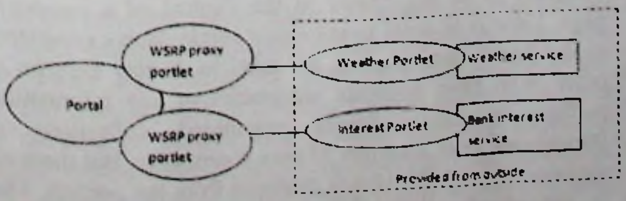

Fig.3: This figure shows the same two portlets but now implemented using web services

There are four defined roles to carry out different responsibilities of the WSRP operation according to the specification [4]. Those are.

--Portlet
--Producer
--Consumer
--End-User

Portlet is hosted in the web server of the producer. It is responsible for generating the markup. Producer provides some web service interfaces including Service Description, Markup, Registration and Portlet Management. Consumer on the other hand is the intermediate communicator who is responsible for gathering information from the web service (i.e. portlet-producer pair) and for aggregating the content to present it to the end user and vice versa.

Interactions of these four entities are given below in summary.

-Consumer discovers the Producer.

– Establish connection between the Producer and the Consumer.

--Consumer learns a complete list of capabilities of the Producer based on the created relationship.

--Connection between the Consumer and the End-User is established.

--Portlet fragments are aggregated to form a portal page.

--User request is generated on the Consumer URL.

--Consumer processes to determine the targeted Portlet/Producer and interact with it to generate the response.

--At the end, destruction of the relationship between the Consumer and the Producer occurs.

Since most of the proprietary and open source portals provide the facility to expose their portlet services as producers and also provide a common consumer with WSRP proxy portlets, integrating portlets through WSRP is only a matter of few mouse clicks to feed some configuration data about the producer.

So this approach becomes a better solution to consider before moving to any other application integration approaches for quick, rich and less expensive solutions.

B. Creating a portlet from scratch

As mentioned in the previous chapter, portlet life cycle has mainly three phases which are instantiation, initialization and request handling. Portlet Container instantiates appropriate portlet when user requests for a service. After the portlet service is instantiated, there are four portlet interface specific methods used to initialize the portlet. These methods can be overriden in the portlet's java class file that extends the GenericPortlet class and they are [1]:

--init() method : This method is called as the first method to instantiate the portlet. Normally this doesn't have any portlet specific implementation.

--Render methods : There are three default render methods namely doView(), doEdit() and doHelp() defined for three portlet modes. Most of the portlet implementation details are contained in these methods.

--processAction() method : It also does have some portlet specific implementation details. This method is called when a client triggered for action request by ActionURL.

--destroy( ) method : This method is called when the portlet gets out of service.

The Portlet API defines three portlet modes in addition to the above four methods. Those portlet modes are view, edit and help modes. The portlet implementation must decide what method to render for each of these modes. Web Flow exposes the string value of the portlet mode under portletMode via the request map on the external context. The process action method is responsible for reading the information sent from the edit configuration form and storing it in a database or in portlet preferences. When a user request comes with action request, it executes processAction method and from there the portlet mode can be changed to move to any render method as required for the action. But when user requests just for rendering, the portlet container can simply executes one of the render methods according to the request. And for each of the portlet modes there are separate JSP pages where the user interfacing is implemented according to the user requirements.

Using unique CSS files for different portlet fragments other than the one specified by the JSR specification is also not recommended [1]. The intention of this convention is to preserve the same look and feel among all the portlets originated by different vendors in the same portal page.

In the above mentioned Jefe project most of the portlets including Hudson portlet, JIRA portlets, Bugzilla portlets and Unit Testing portlet has been developed using this approach since there were not any open source compatible off-the-shelf solutions for any of these applications.

C. *Displaying existing web interface of an application*

Creating a portlet with a more granular level without relying on JSR 168 portlet API [5] is done by embedding the web interface of an application to an IFrame (Inline Frame) portlet.

An IFrame portlet invokes an URL inside an IFrame within the portlet window. It gets content referenced by an URL and displays it in a portlet. IFrame portlets can be downloaded freely or else they can be found already integrated in many web portals. Thus a user can use a portlet integrated to a portal or either create a custom IFrame portlet. Basically these portlets contain following JSP pages:

--configuration.jsp (for edit mode) -This is for adding configuration values such as source URL of directing web page, look and feel attributes and authenticating attributes.

--view.jsp (for view mode) - This makes possible to view the embedded web page inside the current portal page with the help of the functions in the portlet namespace.

--proxy.jsp- This facilitates passing (the) timeout values and the authentication values.

Following custom configurations can be done to a default IFrame portlet.

--It is possible to configure the embedded URL by limiting the 'iframeSrc' portlet preferences attribute to a custom URL instead of adding the URL of embedded web page several times at each time the IFrame portlet is added to a portal page.

--The portlet name can be changed by changing the parameter 'portlet-name.value' in the portlet.xml file.

--It is possible to manage session timeouts of an IFrame with Ajax using a XMLHttpRequest call to refresh the content inside the portlet without having to refresh the entire portal page [6].

--It can be configured to retrieve the user name and password of the current user, who is viewing the page using the 'getUser' method of the 'PortalUtil' class and that will be used to authenticate against the embedded web page.

D. *Configuring third party portlet already available*

Without re-inventing the wheel by developing a portlet from scratch, some specific portlets can be downloaded as off- the- shelf solutions from a range of vendors under the free and open source license. Though this statement sounds like these portlets are able to easily incorporate into web portals, most of the time they don't.

Most of these portlets are (been) developed and (been) tested on one or few portal engines and are expected to work on others. Even though they were built according to open standards, there is a high possibility that the portlets may fail or throw exceptions in new environments. Not only that, when portal engines evolve, sometimes later versions lose the backward compatibility to accommodate portlets that were built for earlier versions. This was a major issue we faced while trying to integrate a Hudson portlet which was built for early Liferay bundle.

If all these conditions are satisfied, installing a new portlet can be carried out either by selecting 'install or upload new plug-ins' link from the control panel or by directly copying the plug-in war file to the deploy folder of the portal engine.

After portal engine log mentions that the new plug-in is successfully deployed and ready to use, the portlet should appear under 'add new applications' area of the control panel. Then it is possible to add it to a portal page as desired, and configure it to suit the new environment by feeding correct configuration data in the portlets edit mode. For this to be done successfully, proper documentation either in help mode of the portlet or as a separate document should be provided from the vendor.

If proper and adequate documentation support is not there for the selected portlet, integration becomes very complex and difficult task. It will require lot of time to study, understand the code and debug (the) errors or find what should be fed to the portlet for it to operate properly in your environment.

Some of the example vendors who distribute (to the) off the shelf portlets stacks are JBoss Community [7], Weblogic Portal partner companies [8] and EmForge [9].

Source Code Management (SCM) portlet included in Jefe is integrated using this approach and it is a third party portlet developed by EmForge.

E. *Integrating entire application as it is*

As mentioned in the chapter two, portlets can be considered as reusable web components which are specially designed to be aggregated in the context of a composite page. Usually in order to provide a single-source experience to the portal user, portlets are used to display excerpts of other Web sites, generate summaries of key information, perform searches, access assembled collections of information from a variety of data sources etc. But there are situations where this idea is deviated from the portlets. That is instead of extracting data from different applications through the portlets, portlets itself can be entire applications.

When implementing the entire application as a JSR 168 compatible portlet, Java Server Faces (JSF) technology comes in to (the) play. JSF portlets are created using the JSF Portlet Bridge and it allows JSF applications to run within a portal environment. It exposes the existing web applications as JSR 168 portlets and these portlets are appropriate mostly when it is necessary to display contents from a JSF application as a portlet without hosting the entire application, or without separately building a portlet for the same [10]. Due to its event-based architecture, standardized component model and the rich tool support, JSF approach is well matched with this scenario.

There are several such off-the-shelf portlets available, and Alfresco Document Repository [11] portlet is one good example. Using the JSF architecture it provides a web-based client that would run as a portlet within a JSR 168 compatible Portal environment.

V. Discussion

During the research and implementation phases of our project Jefe, it was possible to identify various comparative advantages and issues associated with each of the above five approaches of integrating portlets to a portal. Since all the approaches were practically implemented in the Jefe, this chapter is dedicated to address the appropriateness of each approach.

According to the first approach, when using the web services as portlets, it is possible to acquire lots of advantages such as interoperability and portability through the web services. Also for this approach it requires little or no programming experience and it decouples the deployment from hosting [12]. But at the same time this approach has the problem of lack of web services support for most of the existing enterprise applications even though there is a trend of converting enterprise applications to web services.

The second approach of creating portlets for existing applications gives the full flexibility for the developer to come up with the most suitable portlet according to the requirements. But the problem with this is developing a portlet from scratch and integrating it needs lots of professional and development skills as well as time and effort.

The approach of configuring third party portlets already available gives the ability to use the existing resources without duplicating the effort. But that might consume lot of effort and time because there may be lot of configuration issues associated with existing portlets. Also extendibility and understandability of the source code reduces, if proper development methodologies were not followed.

The approach of displaying an existing web interface of an application simplifies portlet development by enabling to provide the portlet functionality without relying on the JSR 168 portlet APIs. Also it enables the application to run simultaneously as a regular web application and as a portlet from the same installation while eliminating the requirement to store, maintain and deploy portlets separately from web application [5]. But at the same time this approach has several disadvantages. It does not support portlet messaging or cooperative portlets and timeouts could be happened easier. IFrame-based applications which rely on JavaScript are failed due to cross-Site-Scripting (CSS) restrictions. Also the links from an IFrame, can take the user outside the IFrame and even outside the portal instead of having it scoped inside the IFrame. Another major problem with IFrame is, it does not maintain the state. Therefore, when users interact with the contents in the IFrame, they might experience unexpected behavior and difficulty to control and provide a consistent look-and-feel in the portal [13].

According to the final approach, by integrating a set of entire applications as a portlets, the main concept behind the EAI technology can be achieved easily. That is because when the applications are developed as portlets, it can gain the advantage of having generic features of portals such as content aggregation, single sign on, etc. This approach provides a common platform between applications and the portal while providing several operations such as user management, content management are automatically synchronized between both the platforms. But this approach can cause severe performance degradations, if the application itself is large and it can reduce the whole portal performance. Also the application and the portal are tightly coupled in this scenario and this can make some unexpected runtime anomalies within the system.

Acknowledgment

It is our privilege to thank our supervisors Dr. Shahani Weerawarana and Ms. Sulochana Sooriyaarachchi from the Department of Computer Science and Engineering, University of Moratuwa who encouraged us to research and carry out our Software Development Workflow Management project - Jefe, using different approaches while integrating enterprise applications to the portal engine which spurred this research paper.

It's our pleasure to thank our external project supervisor Mr. Ishan De Silva from MillenniumIT, for his kind guidance and advises.

We would also like to make this an opportunity to show our gratitude also to our project coordinator Dr. Shantha Fenando from the Department of Computer Science and Engineering, University of Moratuwa, for his kind advises, guidance and for always motivating us to perform our best.

References

[1] A. Abdelnur, S. Hepper, "Java Portlet Specification," version 1.0, 2003, PLT.2, [Accessed: Apr. 02, 2010].

[2] J. Linwood, D. Minter, "Building Portals with Java Portlet API," Apress, 2004, ch.1-3, [Accessed: Apr 03, 2010]

[3] T.Schaeck, "Web Services for Remote Portals (WSRP) Whitepaper," 2002, [Online] Available on: http://www.oasis-open.org/committees/wsrp/documents/wsrp_wp_09_22_2002.pdf, [Accessed: Apr. 10, 2010].

[4] R. Thompson, "Web Services for Remote Portlets Specification," version 2.0, 2008 [Accessed: Apr. 10, 2010].

[5] Oracle (2007).Oracle® Fusion Middleware Developer's Guide for Oracle WebCenter.(1st edition) .[Online]. Available: http://download.oracle.com/docs/cd/E15523_01/webcenter.1111/e10148/jpsdg_bridge.htm [Accessed Apr. 05, 2010].

[6] Liferay Inc "iframe portlet" Internet:http://www.liferay.com/community/wiki/-/wiki/Main/Iframe+Portlet;jsessionid=19B9B802B18BDEF3E9814DDD36CAAEAA node-1,[Accessed Apr. 04, 2010]

[7] JBoss Community " Miscellaneous Portlets." Internet http://www.ibm.com/developerworks/websphere/library/techarticles/0607_boezeman/0607_boezeman.html,[Accessed Apr. 10, 2010]

[8] Oracle Corporation(2008).Portlet Development Guide. [Online].Available:http://download-

llnw.oracle.com/docs/cd/E13155_01/wlp/docs103/portlets/thirdparty.html[Accessed Apr. 11, 2010].

[9] EmForge Portal." EmForge Download." Internet: http://www.emforge.org/wiki/Download,[Accessed Apr. 11, 2010]

[10] Oracle® Fusion Middleware, "Developer's Guide for Oracle Portal." May 2009.[Online].Available: http://download.oracle.com/docs/cd/E12839_01/portal.1111/e10238.pdf [Accessed: Mar. 31, 2010].

[11] Alfresco Software, Inc, "Open Source Document Management System by Alfresco,"2010. [Online].Available: http://www.alfresco.com/products/dm/ [Accessed: Apr. 08, 2010].

[12] X. D. Wang, R. Allan , "Portlet, WSRP and Application," GridSphere and Portlet Workshop, NESC, 2005. [Online]. Available: http://www.nesc.ac.uk/talks/549/Day_1_1545_Wang.pdf [Accessed: Apr.10, 2010]

[13] Richard Gornitsky and Richard Gornitsky." Options for rapid integration of Web applications into WebSphere Portal." Internet : http://www.ibm.com/developerworks/websphere/library/techarticles/0607_boezeman/0607_boezeman.html,July.12,2006[Accessed : Apr. 04, 2010]