WebSphere Portal Server and Web Services Whitepaper

Thomas Schaeck (schaeck@de.ibm.com)

IBM Software Group

Abstract

As web services will become the predominant method for making information and applications available programmatically via the Internet in the near future, portals need to allow for integration of web services as data sources and as remote application components. We see two important options for usage of web services in conjunction with portals:

- Portlets running on a portal server can access a web service to obtain information or invoke remote methods provided by the web service.
- Portals can publish portlets as remote portlet web services to make them available to other portals in a way that allows to easily find and integrate them.

IBM clearly sees the importance of web services for portals and will provide seamless support of web services in WebSphere Portal Server. In this paper, we describe how WebSphere Portal Server will be used to set up distributed enterprise portal systems allowing administrators to easily share portlets across portals and how content providers will be able to use WebSphere Portal Server to publish their content as remote portlet web services that can be integrated by portal administrators of other portals very easily through a web admin interface, without any programming effort.

Table of Contents

Introduction

Portals are focal points for users to access information and applications from many different sources. Typically, portals get information from local or remote data sources, e.g. from databases, transaction systems, syndicated content providers, or remote web sites. They render and aggregate this information into complex pages to provide information to users in a compact and easily consumable form. In addition to pure information, many portals also include applications like e-mail, calendar, organizers, banking, bill presentment, etc.

Different rendering and selection mechanisms are required for different kinds of information or applications, but all of them rely on the portal's infrastructure and operate on data or resources owned by the portal, like user profile information, persistent storage or access to managed content. Consequently, most of today's portal implementations provide a component model that allows plugging components referred to as *Portlets* into the portal infrastructure. Typically, portlets run on the portal server, processing input data and rendering output. Figure 1 shows a typical portal page with several portlets.

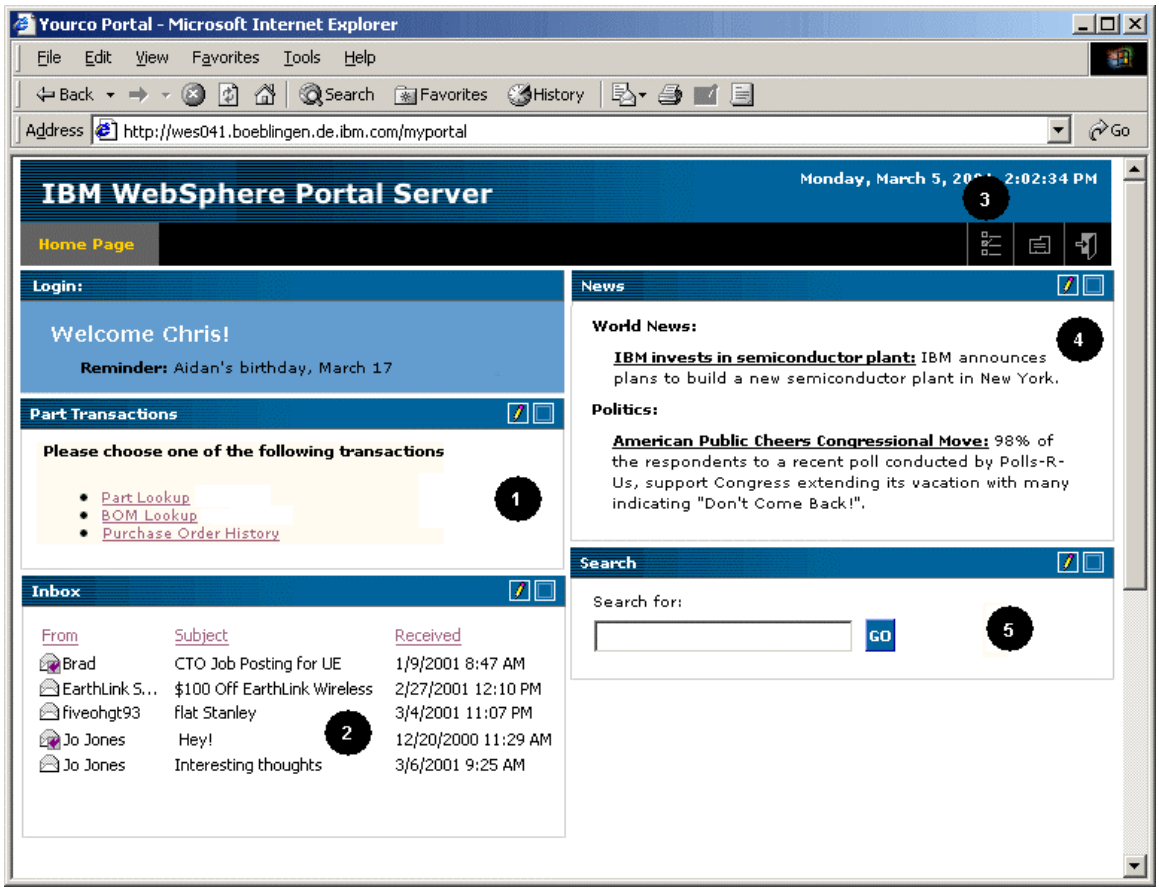

Figure 1: IBM WebSphere Portal Server

Often, the content for portlets displayed with a high frequency is cached locally to improve response times, performance and scalability of portal systems. Figure 2 shows an example where a weather portlet and a news portlet run on an Internet portal. The portal uses databases to cache weather info and news content locally so that the portlets can display them without delay.

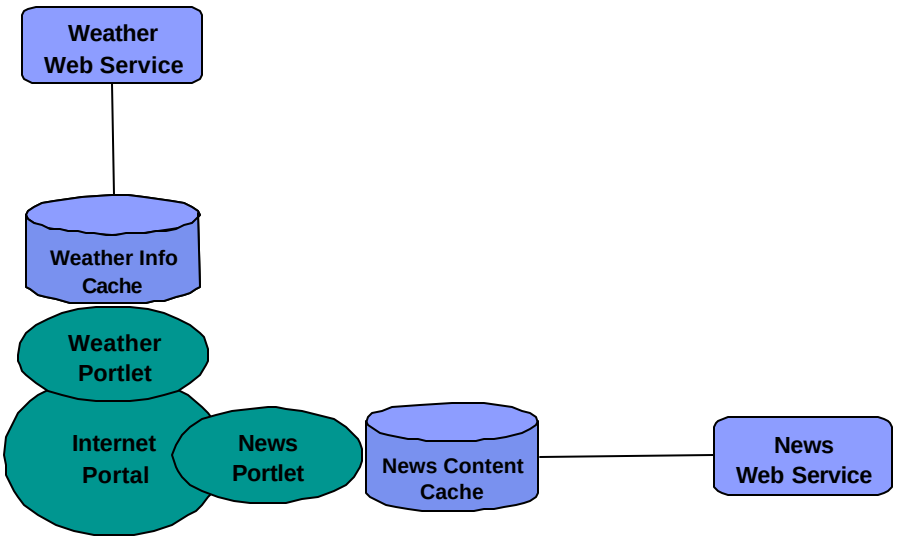

Figure 2: An Internet Portal displaying locally cached content

While local portlets in conjunction with appropriate content caching mechanisms provide very good response times, this approach is not well suited to enable *dynamic integration* of business applications and information sources into portals. Let us consider the following scenario: An employee portal manager wants to include a human resources service calculating variable pay for employees and an external weather service providing weather forecasts. One solution for this scenario is depicted in Figure 3 – a human resources portlet and a weather portlet run locally on the portal server and access remote web services to obtain the required information.

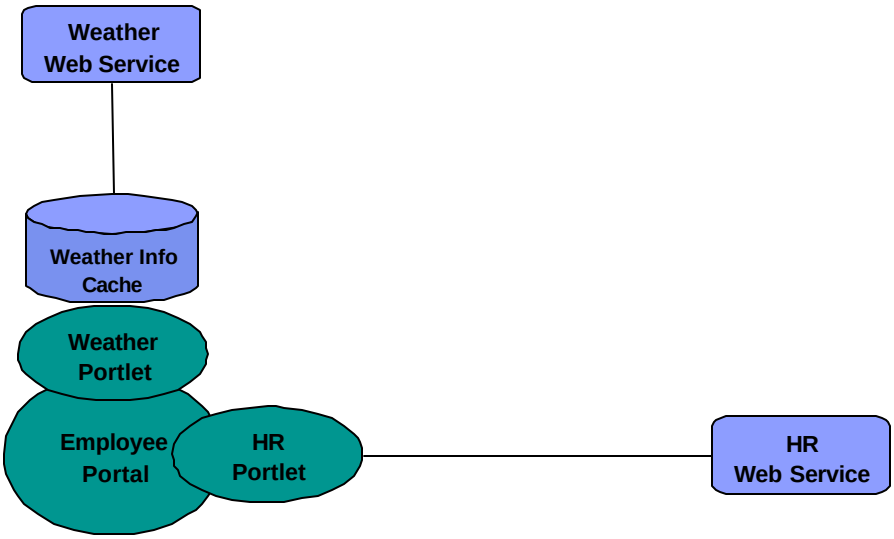

Figure 3: Example of local portlets using a web services

The HR portlet uses a HR web service to calculate the variable pay. By default, it displays a form to query the required input data, e.g. the employee's position. When the employee provides the data to the HR portlet, it invokes the remote web service to calculate the variable pay based on that data. It receives the result from the web service and displays it as a page fragment. The weather portlet by default displays weather forecasts for configurable locations and allows the user to select locations in an edit mode. When the weather portlet is invoked during page

aggregation, it gets the most recent forecasts for the selected locations and renders a page fragment that displays those forecasts.

This approach only works if all portlets are physically installed at the employee portal; the process of making new portlets available is tedious and expensive. To integrate HR information in the portal, either the HR department would implement the HR portlet and give it to one of the administrators of the employee portal to install it, or an employee portal developer would implement the HR portlet according to the interface description of the HR web service. For the weather, an employee portal developer would have to implement a special weather portlet according to the content cache interfaces and install a content cache instance that replicates data with the weather web service. In each case, significant effort is required to make the portlets available.

Obviously, it would be much more convenient if remote web services would appear as remote portlets including presentation and application logic as shown in Figure 4. Instead of just providing raw data or single business functions that still require special rendering on the portal side, *Remote Portlet Web Services* are visual web services including presentation. They are aggregatable web applications that can be invoked through a standard interface using generic portlet proxies on the portal side.

This means that no special portlet code at all needs to be installed on the portal. Use of generic portlet proxies eliminates the need to develop specific portlets for each web service to run on the portal. The task of the administrator is made much easier because portlets can be added dynamically to the environment, and users benefit by having more services made available to them in a timely manner. Additional remote portlets can be included into a portal just by finding them and binding to them by creating a new portlet proxy instance bound to the remote portlet web service. Through the use of portlet proxies, remote portlet web services appear to portals just like local portlets and can be selected by users as easily.

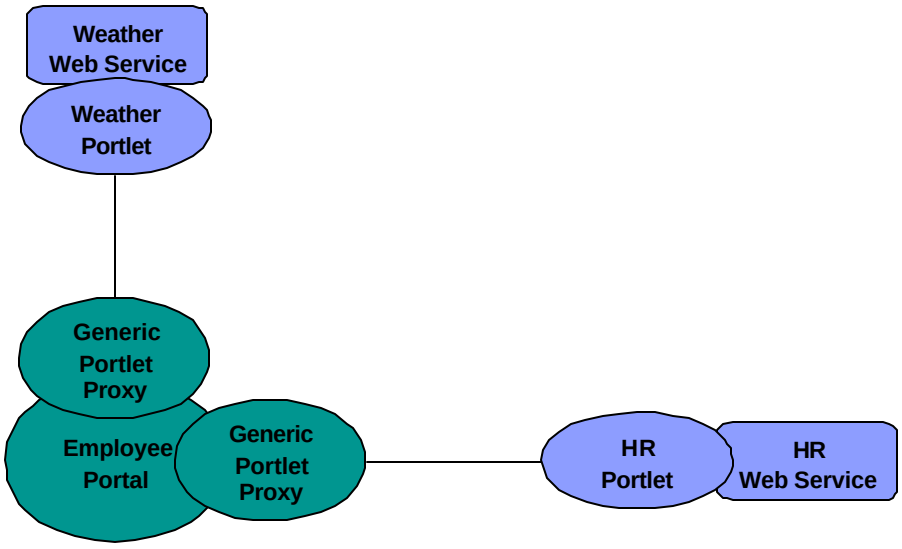

Figure 4: Example of a portal using remote portlets

In the near future, portals must not only be able to run local portlets, but also to include remote portlets and share local portlets by making them available to other portals as remote portlet web services. Figure 5 gives an example of a corporation that owns an employee portal, a supplier portal and a human resources portal. The employee portal has a weather portlet that runs on the local portlet container while the account, stock, search, variable pay and news portlets run remotely and are accessed by the employee portal through portlet proxies. The account and stock

portlet reside on a bank's portal while the news portlet runs on a content provider's portal. The employee portal itself has not shared any local portlets for remote use. The human resources department has a variable pay portlet running on their portal that has been made available for use by other portals by publishing it as a web service. The human resources portal in turn uses a calendar portlet that is provided by a central portlet server that front-ends several external web services.

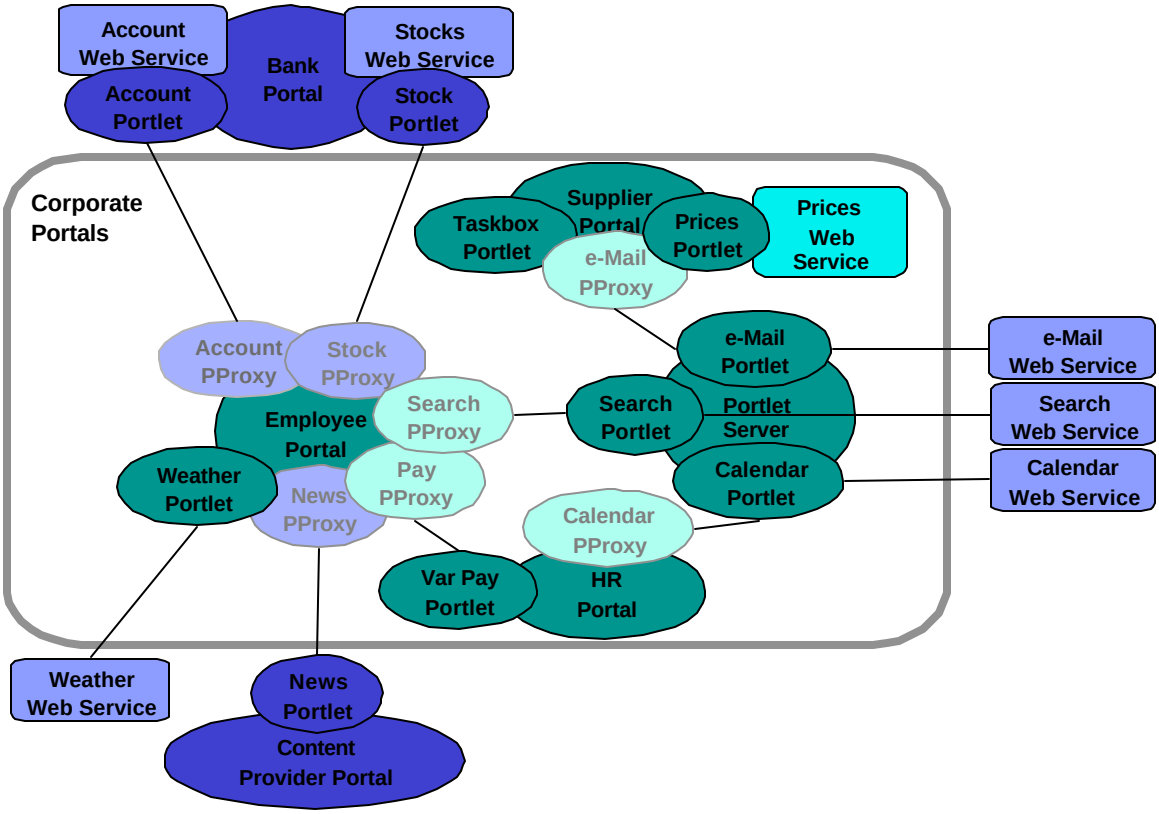

Figure 5: A distributed portal solution based on remote portlets and web services

WebSphere Portal Server Architecture

To implement systems as described in the previous section, a comprehensive and flexible architecture is required that defines the relevant building blocks as well as the interfaces and protocols between them. The architecture needs to cover all the way from client devices over gateways through portals and local or remote portlets to web services. Also, it must cover mechanisms and formats to deploy portlet code in portals locally or find and bind to remote portlet web services. Figure 6 shows WebSphere Portal Server's open portal architecture that addresses the areas mentioned above.

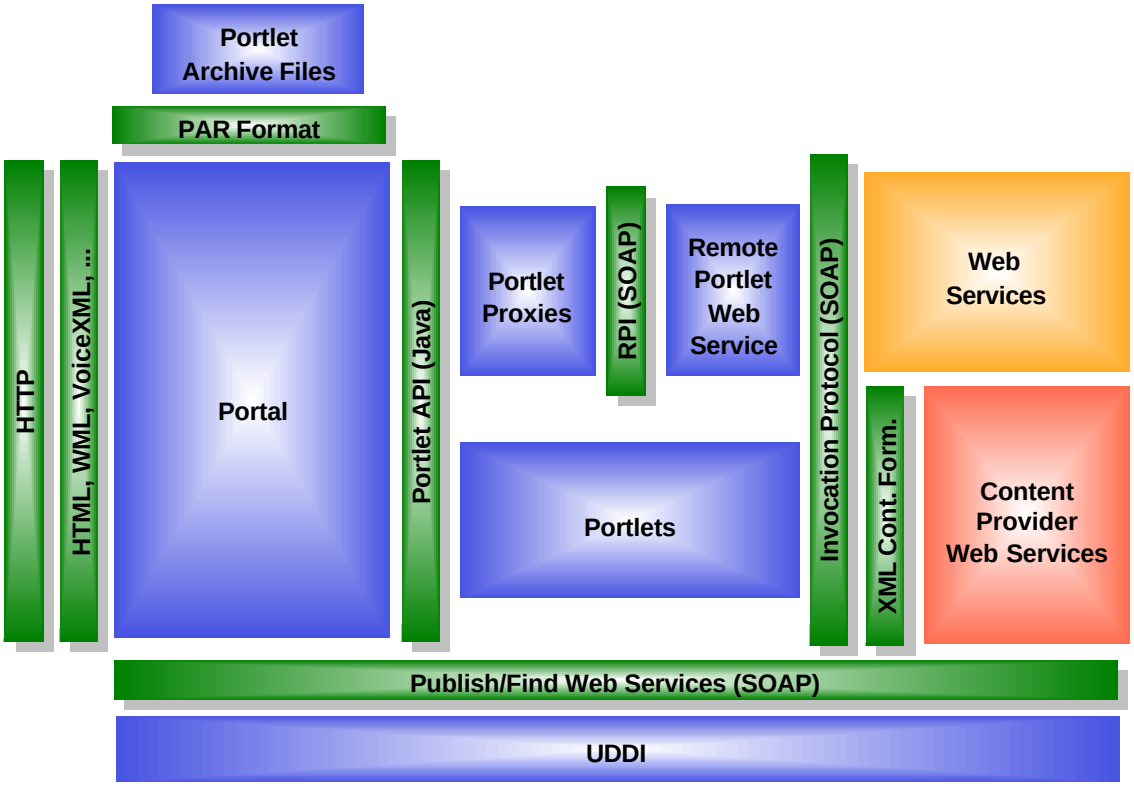

Figure 6: WebSphere Portal Server architecture including web services and remote portlets

Portal clients access portals via the HTTP protocol, either directly or through appropriate proxies or gateways like WAP gateways or voice gateways. The mark-up languages used by these devices may be very different. WAP phones typically use WML, iMode phones use cHTML, voice browsers mostly use VoiceXML while the well-known PC web browsers use HTML. To accommodate different devices, portals need to support different mark-up languages.

When aggregating pages for portal users, the portal invokes all portlets that belong to a user's page through the *Portlet API*. We differentiate two different kinds of portlets:

- **Local Portlets** run on the portal server itself. They are deployed by installing *Portlet Archive* files on portal servers and are invoked by the portal server directly through local method calls. As local portlets run on the portal server itself, they provide minimal latency times. However, installing portlets usually requires assurance that the portlets are not erroneous or even malicious.
- **Remote Portlets** run as web services on remote servers. They are published as web services in a Universal Description, Discovery and Integration (UDDI) directory to be easy to find and bind to. A remote portlet web service is bound by adding a *Portlet Proxy* to the portal's portlet registry when an administrator finds and selects the remote portlet web service in the UDDI directory. Portlet proxies are generic local placeholders that invoke portlets located on remote servers through a *Remote Portlet Invocation (RPI)* protocol based on the Simple Object Access Protocol (SOAP).

While local portlets can be expected to provide a large part of the base functionality for portals, the remote portlet concept allows dynamic binding of a large number of remote portlet services without any installation effort or code running locally on the portal server.

Portlets

Portlets are pluggable components runnung inside a portal's portlet container, similar to servlets in many aspects. Portlets are written to a portlet API similar to the servlet API. However, portlets run in a portal environment, while servlets run stand-alone in a servlet container. While servlets communicate directly with their clients, portlets are invoked indirectly via the portal application. In order to properly run in the context of a portal, portlets must produce content that is suited for aggregation in larger pages, i.e. portlets should produce markup-fragments adhering to guidelines that assure that the content generated by many different portlets can be aggregated.

When the portal receives a servlet request, it generates and dispatches events for any portlet affected by parameters in the request and then invokes all portlets that have to be displayed trough the portlet invocation interface in a second step (see Figure 7).

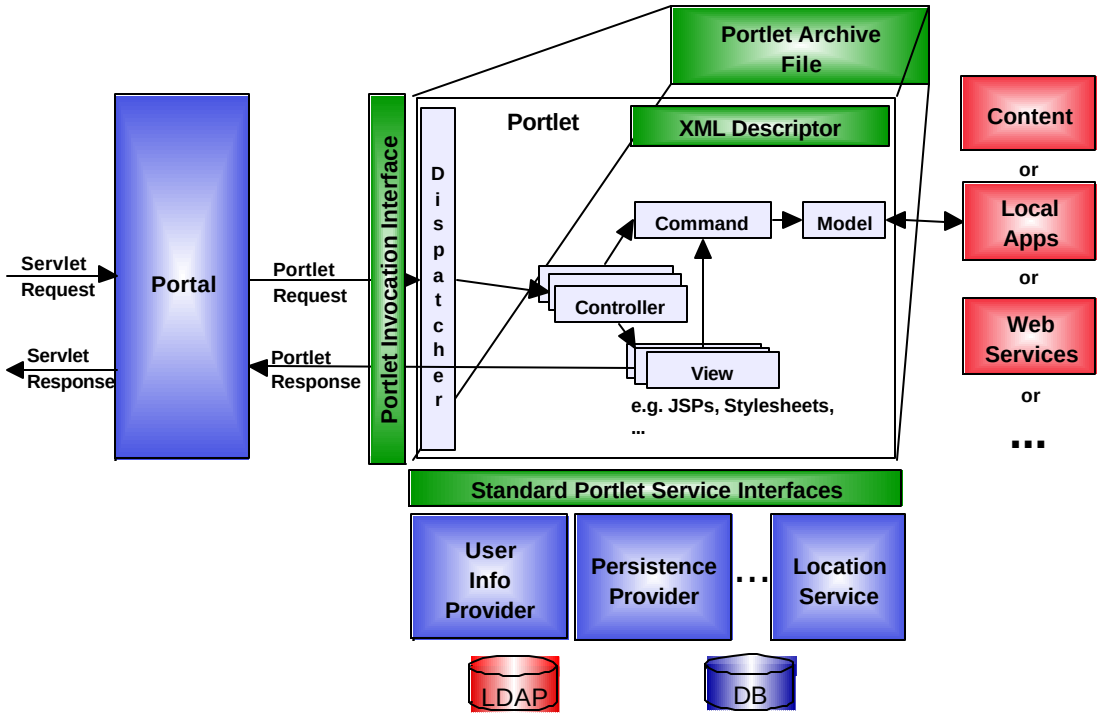

Figure 7: The Portlet Concept

While portlets must implement the invocation methods required by the Portlet API, internally they may be implemented differently. A pattern that has proven very suitable for portlet programming is the Model-View-Controller pattern. It separates the portlet functionality into a controller receiving incoming requests, invoking commands operating on a model that encapsulates application data and logic and finally calling views for presentation of the results.

Portlets have access to portal related functions and data through *Portlet Service Interfaces*. These interfaces provide portlets with functions including access to user profile information, persistent per-portlet instance data, action handling, etc. Apart from portal specific functions, portlets can use all the J2EE services that are available to servlets as well as vendor-provided connectors to access back-end data and applications or even services in the Internet.

For easier deployment, portlets can be grouped in *Portlet Applications* packaged into *Portlet Archive* files containing a deployment descriptor, Java classes, jar files, and resources.

Web Services

The concept of web services has been developed to allow business applications to communicate and cooperate over the Internet. Web services imply a paradigm shift compared to the way the Internet works today. While traditional applications interacting with services in the Internet need to know those services a-priori and need to be pointed to these peers manually, the web services concept allows applications to find web services in a standardized directory structure and bind to these services with minimal human interaction (see Figure 8).

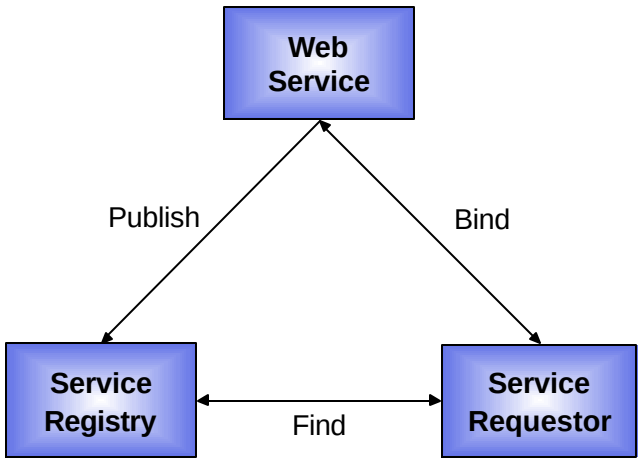

Figure 8: Web Services – publish, find, bind

Web services allow objects to be distributed across web sites where clients can access them via the Internet. Global service registries are used to promote and discover distributed services. A client that needs a particular kind of service can make a query to the global service registry to find services that suit its needs. The client can select one of the services, bind to that service, and use it for a certain period of time. As service discovery and selection in some cases can be performed without human interaction, services can be switched very quickly. Automated service discovery also allows establishing very robust networks of services. If multiple web services exist that provide identical functions, a client can easily switch to a back-up system when the currently used service fails.

The most important standards in this area are *Universal Description, Discovery, and Integration* (*UDDI*, see [3]) for registration and discovery of web services, the *Simple Object Access Protocol* (*SOAP*, see [4]) for communication between web services, and the associated *Web Services Description Language* (*WSDL*, see [5]) for formal description of web service interfaces.

As web services will become the predominant method for making information and applications programmatically available via the Internet, portals will need to allow for integration of web services as data sources and as remote application components very soon. A typical example is a news portlet that allows the user to configure the news categories to track and then gets the news for these categories live from a web service whenever it is displayed. In this case, the portlet code runs locally on the portal and uses the web service to access information. Rendering is done by the local portlet while the web service only provides the information to be rendered, for example as an XML document (see Figure 9, search portlet and news portlet).

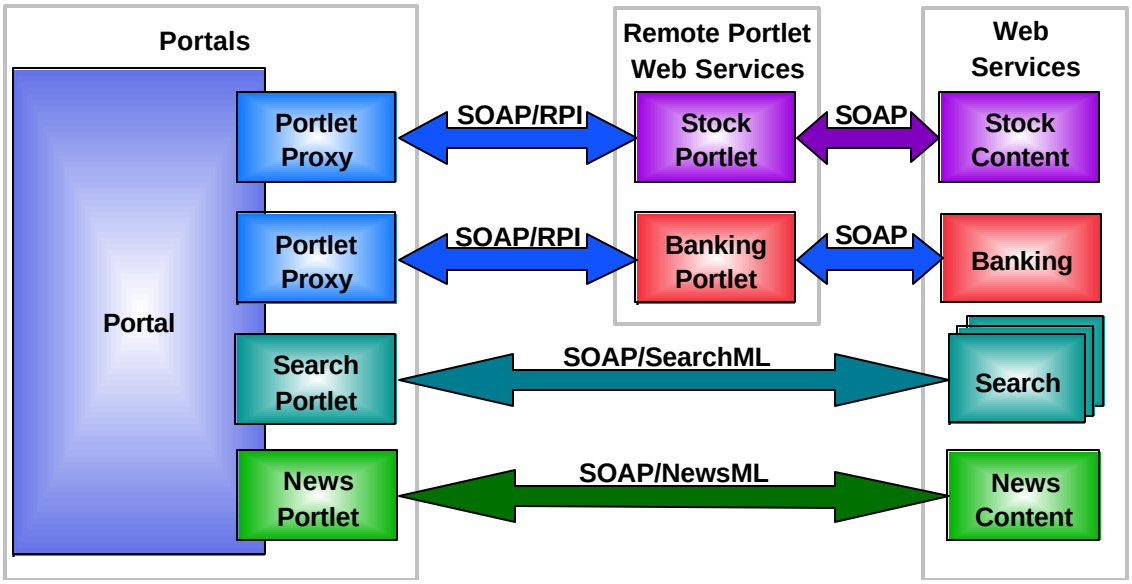

Figure 9: Portals and web services

Another scenario for use of web services by portals is sharing of portlets with other portals. In this scenario, a remote server, e.g. another portal publishes portlets as remote portlet web services in a UDDI directory. The portal can now find the remote portlet services in the directory and bind to them. As a result, the remote portlets become available for portal users without requiring local installation of portlet code on the portal itself (see Figure 9, portlet proxies for stocks and banking).

Web Services used by Portlets - Today

In the past, portlets could access applications and information in different ways. In the intranet this typically happens using database connections, LDAP connections, Java RMI, DCOM, CORBA, etc. Over the Internet, in most cases the HTTP protocol is used to send requests to remote applications and receive results. Within short time, SOAP will be the primary communication mechanism for invocation of remote services by portlets and will incrementally replace the mechanisms listed above.

With SOAP and UDDI, communication between web services and their clients and management of web services in global and corporate directories is unified. This allows programmatic finding, binding and usage of web services. Web services can be formally described using WSDL descriptions that can be used by appropriate tools to generate SOAP proxies for specific programming languages. Also, there are tools that can create web services and WSDL descriptions from existing code.

Figure 10 shows how a portlet that uses a web service. When a portlet receives a request that requires invocation of a remote service, the portlet makes calls on a SOAP proxy object. The proxy takes the parameters, marshals them into a programming language-independent SOAP request, and sends this request to the remote web service. The web service has a SOAP wrapper that receives the SOAP request, unmarshals the parameters and invokes the local service implementation with these parameters. When the service returns the result, the SOAP wrapper marshals the result data into a programming-language independent SOAP response and sends it back to the SOAP proxy. The SOAP proxy finally unmarshals the result data and returns it to the calling portet in the form of an appropriate object.

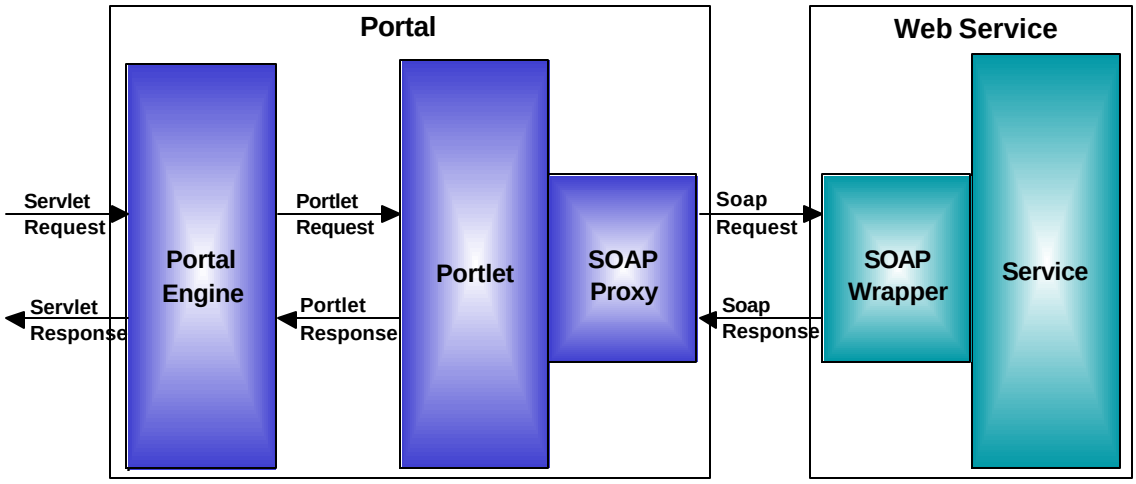

Figure 10: A portlet using a web service

To simplify writing portlets using web services, IBM provides a service proxy generator tool that automatically produces client code from a WSDL interface document, and optionally a service implementation document. If only a service interface document is used, the service proxy generator tool generates a generic service proxy which can be used to access any implementation of the given service interface. If both a service interface and a service implementation are used, the service proxy generator tool generates a service proxy that will only access the specified service implementation. The service proxy contains code that is specific to a binding within the service interface. For example, if the binding is a SOAP binding, then the service proxy will contain SOAP client code that is used to invoke the service.

Remote Portlet Web Services – Near Future

In order to allow for dynamic integration of portlets in portals without installing a portlet archive file with the entire portlet code locally, portlets themselves have to be provided as web services. This requires a Remote Portlet Web Service Interface description in WSDL.

The WSDL description defines a common set of methods for all remote portlets and the required parameters as well as the return values, corresponding to the Portlet API. This means that remote portlet services do not have to be implemented in Java, they could as well be implemented in other languages.

Web service providers who want to publish remote portlet web services must publish appropriate entries to a UDDI directory, referencing the Remote Portlet Web Services Interface WSDL description.

Once a remote portlet has been published, portal administrators can use their portal adminstration tools to search the UDDI directory for web services that implement the Remote Portlet Web Services Interface and pre-select some of the matching portlet web services for use in their portal by adding them to the portal's portlet registry (see Figure 11).

Once the portlets are in the registry, users can select them to be displayed on their personal pages. Alternatively, portals may be set up in a way that allows portal users themselves to browse the directory for portlet web services and add references to remote portlets to their personal pages.

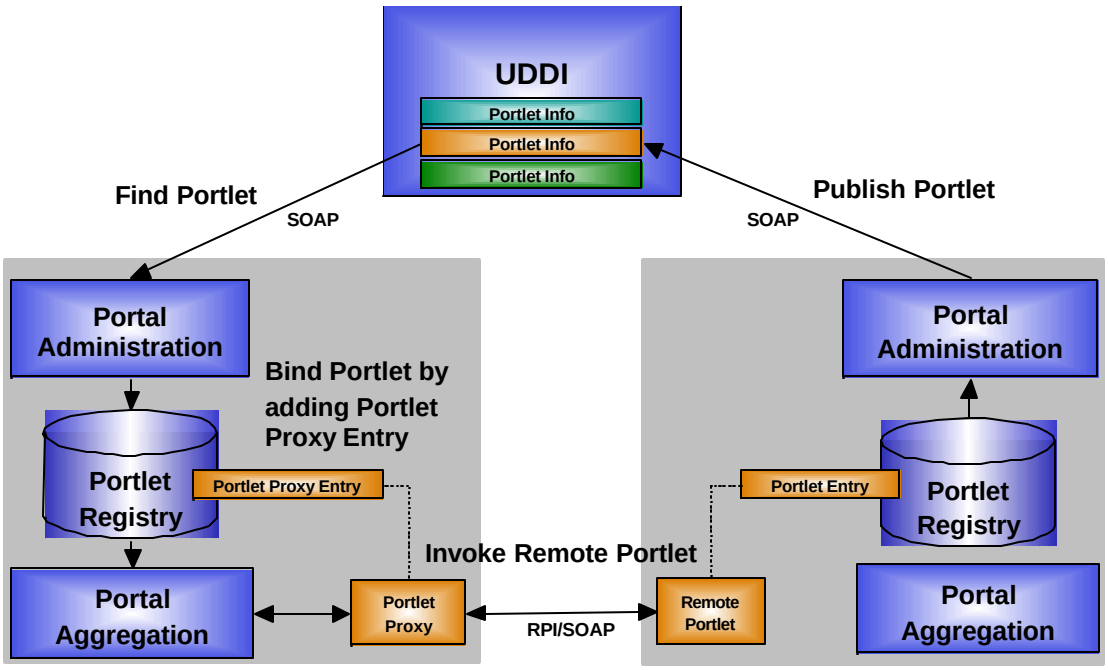

Figure 11: Finding and binding to remote portlets

When a page that references a remote portlet gets rendered, the portal uses a portlet proxy to invoke the remote portlet web service through the Remote Portlet Invocation (RPI) protocol (see Figure 12). The portlet invokes the portlet proxy exactly like it would invoke a local portlet, passing `PortletRequest` and `PortletResponse` objects. The portlet proxy internally invokes a SOAP proxy to marshals all parameters into a SOAP request and sends it to the remote server hosting the portlet web service. The SOAP wrapper on the web service side unmarshals all information in the incoming request and calls on the remote portlet.

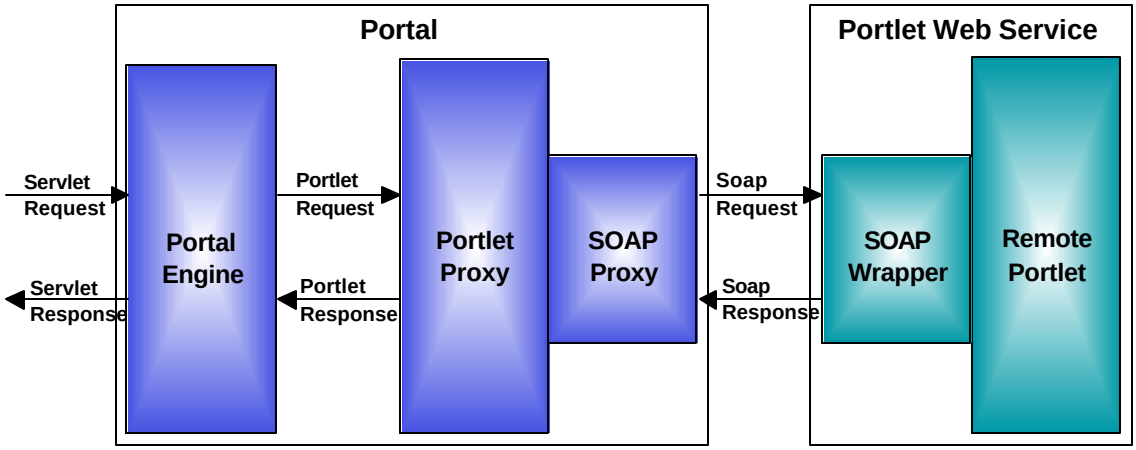

Figure 12: Remote Portlet Invocation (RPI)

For the remote portlet, it is transparent whether it is invoked directly by a portal engine or indirectly through the web service interface. In each case, it processes the input parameters and returns a `PortletResponse` object.

The SOAP wrapper marshals the response into a SOAP response and sends it back as the reply to the SOAP proxy that in turn unmarshals the reponse for the portlet proxy that finally returns a `PortletResponse` object to the portal engine that initiated the request.

Use of Web Services in IBM WebSphere Portal Server

In order to be easy to use, the mechanisms for publishing portlets as remote portlet web services, finding remote portlet web services, binding to them and using remote portlets must be integrated seamlessly into portal products. We can identify four different dialog flows that need to be provided (see Figure 13).

- **Publishing portlets:** Administrators can publish portlets to make them available for use by other portals as remote portlet web services.
- **Finding and binding portlets:** Administrators can find remote portlet web services and bind to these portlets.
- **Using remote portlets:** Users must be able to select and use remote portlets transparently, just as easily as local portlets.
- **Finding and using remote portlets:** "Power users" should be able to find remote portlet web services by browsing the directory themselves.

Publishing portlets may either include two or three steps, depending on whether the portal has already been associated with a UDDI business. If this is not the case, WebSphere Portal Server prompts the administrator to enter the required business descriptions and publishes a business entry to the UDDI directory and associates the portal with that entry. Once the portal is associated with a business entry in UDDI, publishing portlets only requires two steps – in the first step, the administrator selects the portlet to be published and in the second step he provides a description for the new UDDI service entry to be created for the portlet.

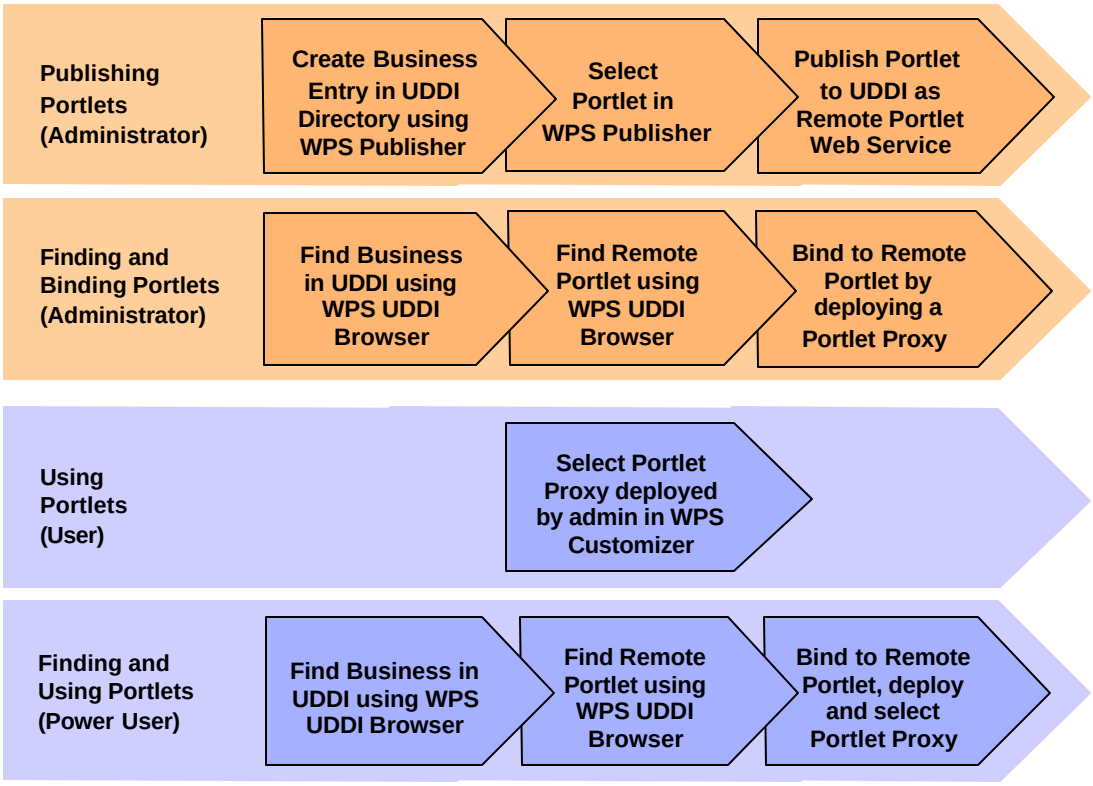

Figure 13: WPS dialog flows for publishing, finding, binding and using remote portlets

Finding portlets requires three steps. First, the administrator uses the built-in UDDI browser to find businesses that provide remote portlet web services and selects one of them. Second, he finds the desired portlet provided by the selected business and selects it. Finally, he lets WPS add the remote portlet to its portlet registry to make it available for portal users.

Using a remote portlet is as simple as using a locally installed portlet – users can select remote portlets in the customizer. To allow more sophisticated users to find and bind to remote portlets themselves, a portal may be configured to allow access to these functions for users. In this case, the dialog flow for the user would be identical to the workflow for administrators to find/bind portlets described above.

The following sections give a preview of how the flows mentioned above will be realized in future versions of WebSphere Portal Server. The screenshots shown in the figures are made from a prototype, the screens may look different in the product. All the screens shown are easily customizable my modification of appropriate stylesheets and JSPs.

Publishing Portlets as Remote Portlet Web Services in UDDI

Only portal administrators are allowed to publish portlets as remote portlet web services into a UDDI directory in order to make them available for dynamic integration in other portals. After logging in, the administrator can click on the *Publish* link. WebSphere Portal Server now displays a page that shows the list of available portlets. For each portlet, the portlet name and portlet description are displayed (see Figure 14).

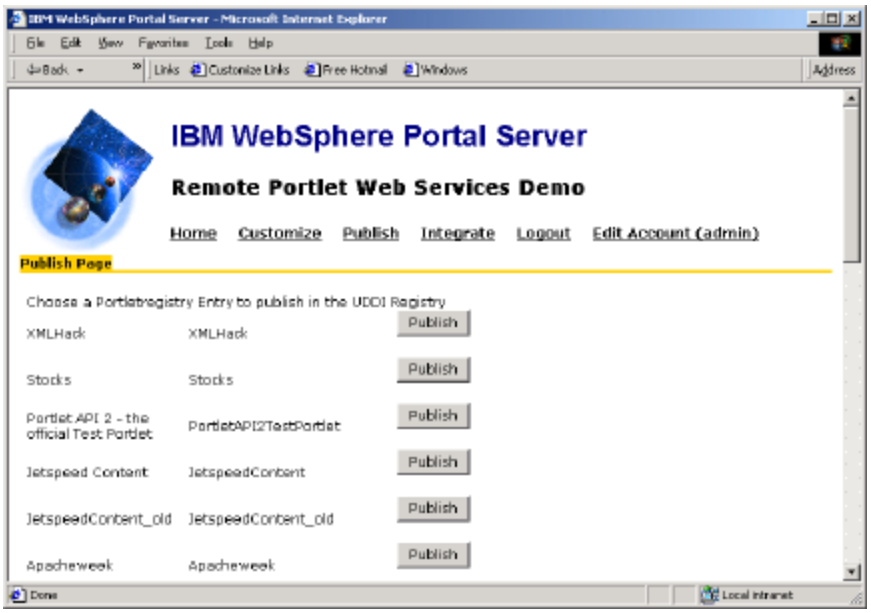

Figure 14: The administrator selects the portlet to be published

To publish a particular portlet to UDDI as a remote portlet web service, the administrator presses the *Publish* button beside that portlet. WebSphere Portal Server now prompts for a description for the portlet in a new screen (see Figure 15).

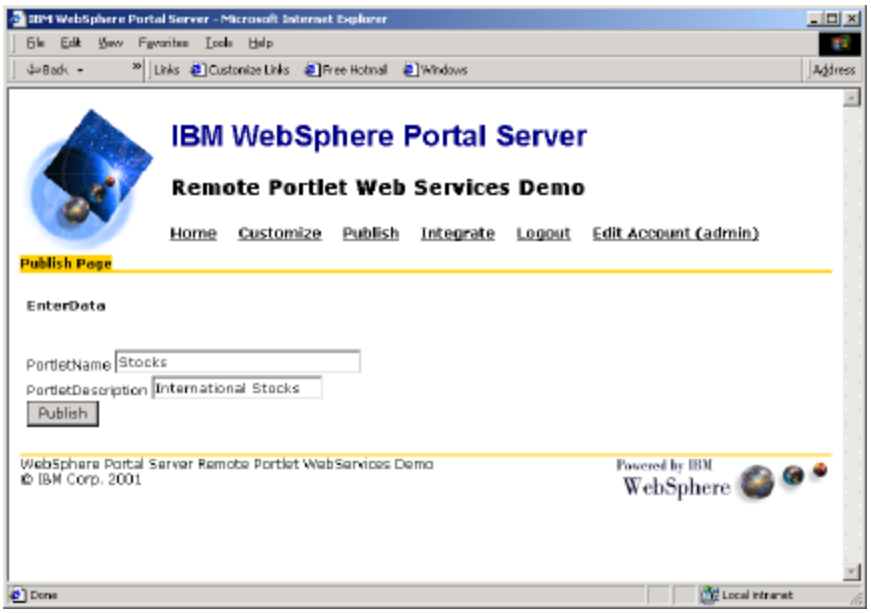

Figure 15: The administrator enters the remote portlet description

The administrator enters the description and presses the *Publish* button in the new screen. WebSphere Portal Server automatically creates a remote portlet web service description from the data entered by the administrator and the data stored in the WPS portlet registry and publishes it to the UDDI directory specified in the WPS settings. This may either be the global UDDI directory infrastructure or a corporate UDDI directory. When this action succeeds, WebSphere Portal Server displays a confirmation screen (see Figure 16).

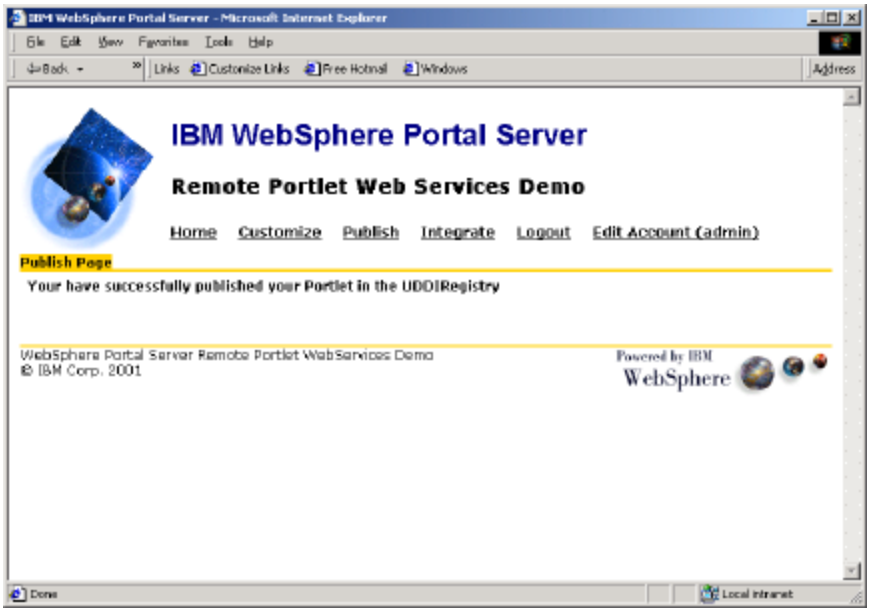

Figure 16: WebSphere Portal Server confirms successful publishing of a portlet

Finding and binding to Remote Portlet Web Services

Finding and binding to remote portlet web services will be possible for administrators. To find a remote portlet web service, the administrator clicks on the *Integrate* link. WebSphere Portal Server prompts for a business name prefix to search for. Alternatively, it can search for all businesses that provide remote portlet web services.

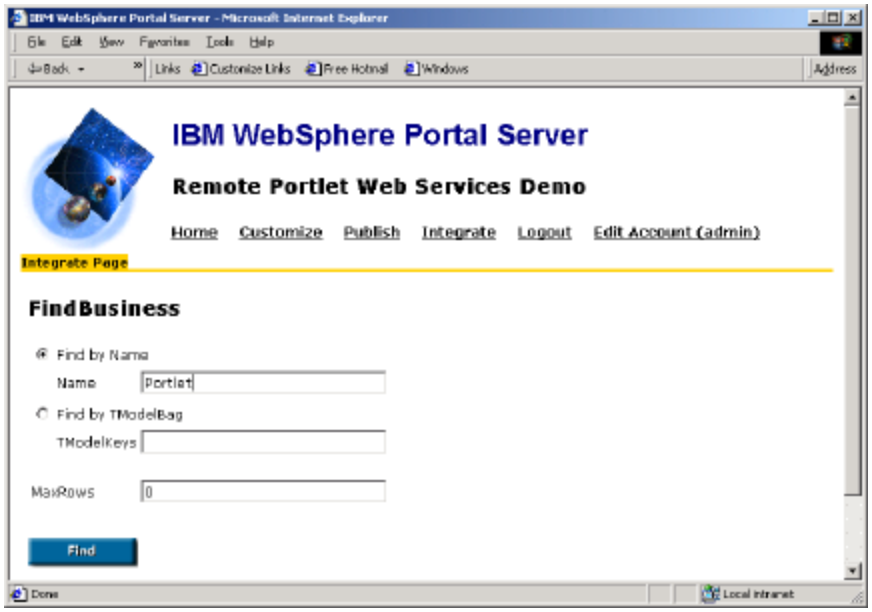

Figure 17: The administrator finds businesses providing remote portlet web services

When the administrator presses the *Find* button, WPS queries the UDDI directory for businesses with the given name or all businesses providing remote portlet web services (see Figure 18).

Figure 18: WebSphere Portal Server lists businesses providing remote portlet web services

The administrator can view the details of the UDDI business description or view the services offered by a business by pressing the *View Service* button. In the latter case, WPS queries the UDDI directory for the services offered by the particular business and displays them to the administrator (see Figure 19).

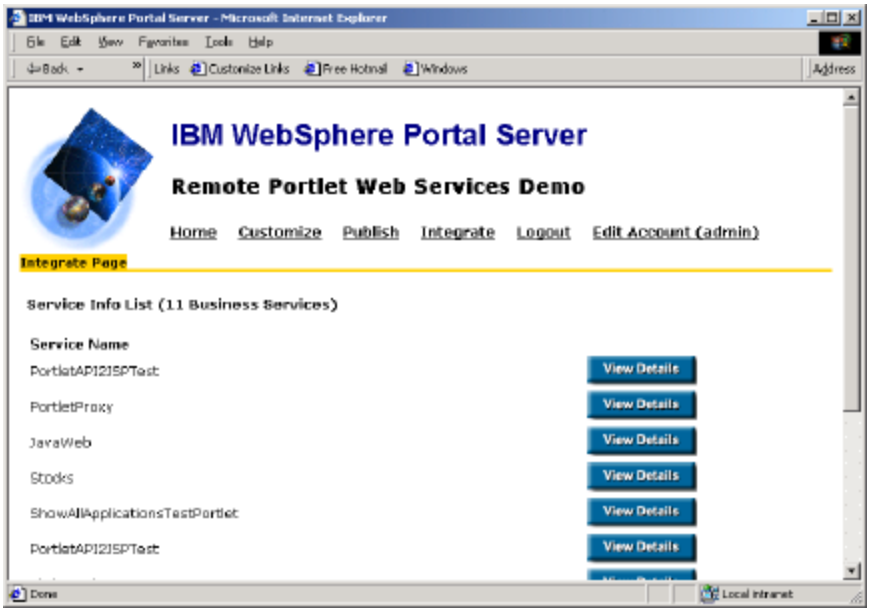

Figure 19: WebSphere Portal Server displays the remote portlet web services provided by a business

Each entry in the list shows a service name and has a button to view the service details. When the administrator presses the *View Details* button, WPS displays a page with the service details (see Figure 20).

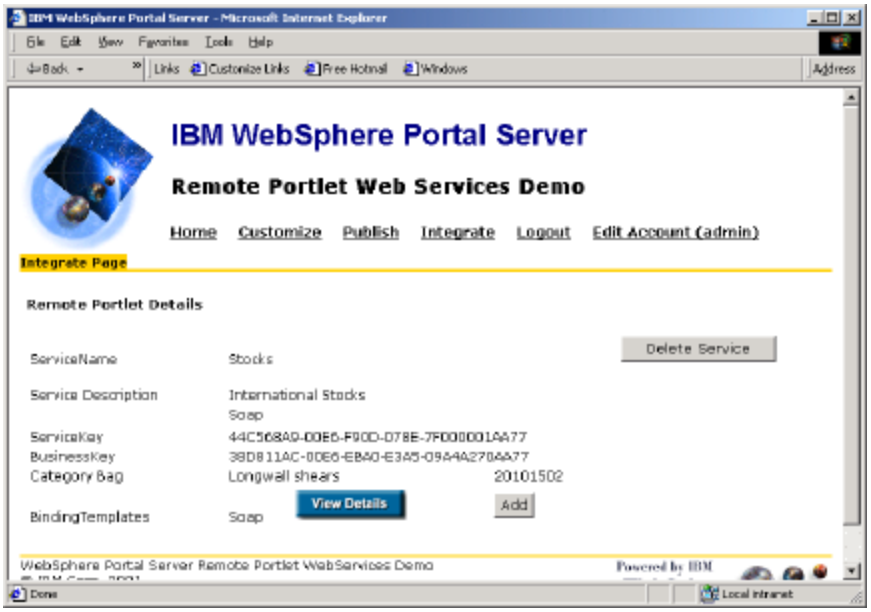

Figure 20: WebSphere Portal Server displays details for a particular remote portlet web service

This page shows the service name, description, keys, category bags and binding templates. It has a *ViewDetails* button to view further details, an *Add* button and a *Delete Service* button. Deleting the service will only work from the portal that published the remote portlet web service, which is ensured by the UDDI directory. To add the remote portlet to the WPS portlet registry in order to make it available to users, the administrator presses the *Add* button. As a result, WebSphere Portal Server gets the relevant information about the remote portlet web service and creates a new portlet proxy entry in its portlet registry to make the remote portlet available to users.

Using Remote Portlet Web Services

For users, usage of remote portlet web services is entirely transparent. After logging in, the user can click on the *Customize* link to navigate to the WebSphere Portal Server Customizer screen that shows all portlets that are available for the particular user (see Figure 21).

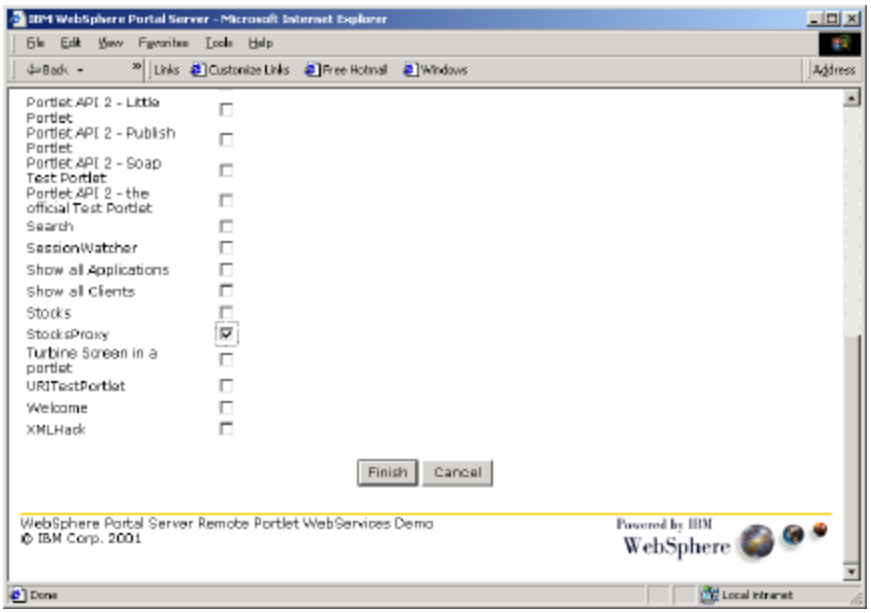

Figure 21: WebSphere Portal Server displays available local and remote portlets

The user can select a proxy for a remote portlet like any local portlet. After selecting the remote portlet in the customizer, it is displayed on the user's page (see Figure 22).

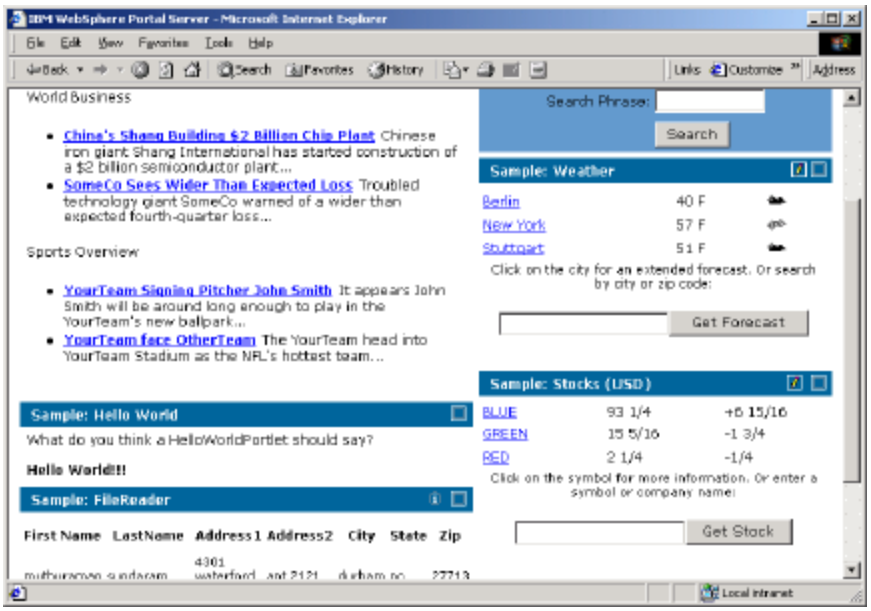

Figure 22: WebSphere Portal Server displays a page that contains a remote portlet

Application Examples

After explaining the basic concepts and presenting the capabilities of WebSphere Portal Server to publish, find and bind to remote portlet web services, in this section we give some application examples that show how these capabilities can be exploited.

Content Providers publishing Content through Remote Portlets

Today, most content providers publish their content live on the internet using HTTP or FTP servers or they provide client software that replicates and caches content via proprietary protocols. In each case, integrating content into a portal is a difficult task. While portals will provide some portlets supporting some content sources out of the box, it will be necessary to develop and install additional portlets for the remaining content sources, i.e. the party that sets up the portal needs to spend a lot of money and effort in order to aggregate a rich set of content from different sources. This is not only a bad situation for portal owners but also for content providers as the fact that it is relatively hard to include their content limits business growth of content providers to depend on services capacities.

In order to allow for integration of their content in portals without any programming or service effort, content providers can use WebSphere Portal Server to surface their content as portlets and publish these portlets as remote portlet web services in the public, global UDDI directory. Figure 23 gives an example where a content provider does not only provide raw content for rendering by portals but also easily integratable content portlets.

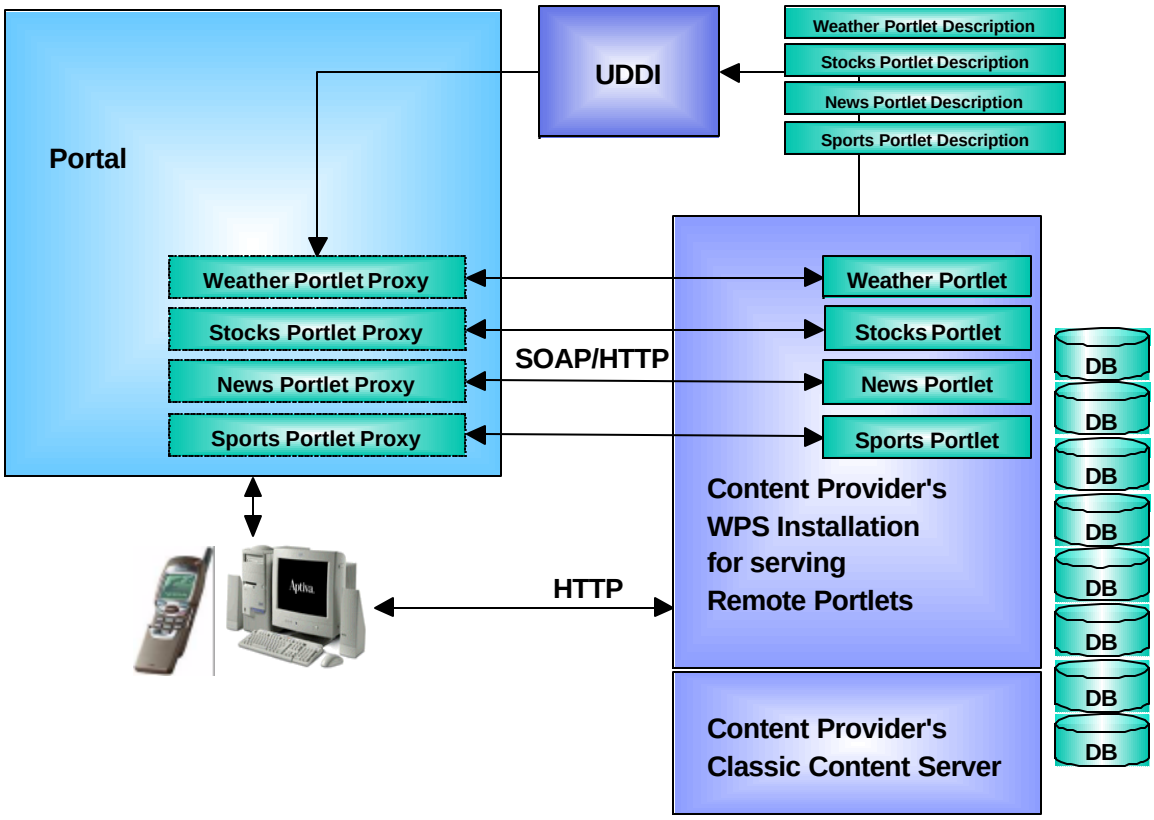

Figure 23: WebSphere Portal Server used to publish content through remote portlet web services

In order to provide this value add to customers, the content provider runs a WebSphere Portal Server installation serving remote portlets in addition to the classical content server. Once the content provider has used the publish function provided by WPS to advertise the remote portlet web services in UDDI, administrators of portals who wish to use content from the content provider can simply look up the content provider's business entry in the UDDI directory and bind to remote portlet web services that provide the desired content. The portlets on the content

provider's server become available immediately without any programming or installation effort and can be used by the portal users. At the same time, WPS provides the content provider itself with a portal, i.e. the content provider can also make content available to users directly if desired.

Portals publishing local Portlets for remote use

While portals initially have been operated in isolation from each other, now the demand for cooperation between portals starts to arise within big corporations. Very soon, corporate portals will also need to cooperate with supplier or customer portals, so ultimately portals will need to cooperate over the Internet as well as within intranets. In the introduction we have already described a scenario where an employee portal, a supplier portal, a HR portal and a remote portlet server within a corporation share portlets and also include portlets provided by external banking and content portals. Figure 24 shows the example of an HR portal that shares portlets with another corporate portal in more detail.

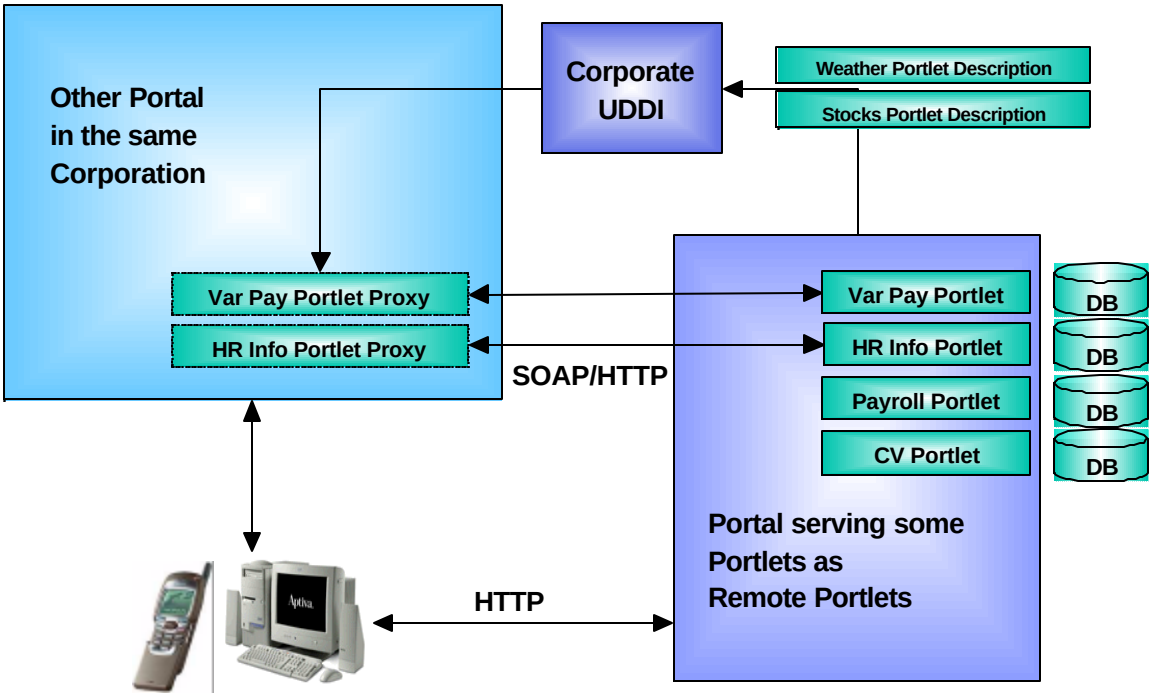

Figure 24: WebSphere Portal Server used by another portal

The HR portal provides various portlets. Some are only intended for use by HR staff like the Payroll Portlet or the CV Portlet. However, there are some portlets that are of interest to all employees, e.g. a Variable Pay Portlet that provides info on how big the variable pay will be based on current revenue and an HR Info Portlet providing HR related news.

Assuming that the corporation has its own corporate UDDI directory which is only accessible from the intranet, the a HR portal administrator would use WebSphere Portal Server's publish function to create remote portlet web service entries for both portlets in the corporate UDDI directory. Thus the Variable Pay Portlet and the HR Portlet become available for use by other portals in the corporation. An administrator of another portal inside the corporation can even find the remote HR portlets using WebSphere Portal Server's built-in UDDI browser and integrate them into his portal with a single click.

Conclusion

In this paper we have given an introduction to distributed portals and some important issues regarding cooperation between portals. We have explained the WebSphere Portal Server architecture in general and the web services aspects in particular. We described two ways of using web services in the context of portals: Use of web services by portlets and use of portlets as remote portlet web services.

Through proxies generated automatically using appropriate tools, portlets are able to use web services to obtain information or initiate transactions instead of the traditional mechanisms for remote procedure calls or data queries. IBM provides tools that make use of web services from portlets very easy already today. From a given service description, tools provided in the IBM Web Services Development Environment can create ready-to-use proxy classes in Java that can easily be used from portlets.

The concept of remote portlet web services will allow deploying distributed portals cooperating within an intranet or over the internet in the near future. Portal implementations provided by different vendors will be able to share portlets and cooperate across corporate boundaries if required. IBM WebSphere Portal Server will provide full support the remote portlet web service concept. Publishing of local portlets as remote portlet web services and integration of remote portlet web services in WebSphere Portal Server-based portals will be possible with just a few clicks by an administrator.

The ability to host portlets and publish them as remote portlet web services that can be integrated into portals easily will turn WebSphere Portal Server into a platform that allows content providers to provide content to their customers – portals – in the most easily consumable form. Also, it will allow application providers to embed their applications into portlets and publish them as remote portlet web services.

References

1. *WebServices Conceptual Architecture*, WebServices Architetcture Team, IBM Software Group, 2001

2. *WebSphere Portal Server 1.2 Technical Whitepaper*, IBM Software Group, 2001

3. *UDDI Technical Whitepaper*, Ariba, IBM, Microsoft, 2000 http://www.uddi.org

4. *Simple Object Access Protocol (SOAP) 1.1*, Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer, W3C Note, 2000 http://www.w3.org/TR/SOAP

5. *Web Services Description Language (WSDL) 1.1*, Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, 2000 http://www.w3.org/TR/wsdl