Web Services for Remote Portals (WSRP) Whitepaper

22 September 2002

Author: Thomas Schaeck <schaeck@de.ibm.com>

Executive Summary

Integration of content and application into portals has been a task requiring significant custom programming effort. Typically, portal vendors or organizations running portals had to write special adapters to allow portals to communicate with applications and content providers to accommodate the variety of different interfaces and protocols those providers used. The OASIS Web Services for Remote Portals (WSRP) standard simplifies integration of remote applications and content into portals to the degree were portal administrators can pick from a rich choice of remote content and applications and integrate it in their portal simply with a few mouse clicks, without programming effort. As a result, WSRP becomes the means for content and application providers to provide their services to organizations running portals in a very easily consumable form.

To achieve this, the WSRP standard defines pluggable, user-facing, interactive web services with a common, well-defined interface and protocol for processing user interactions and providing presentation fragments suited for mediation and aggregation by portals as well as conventions for publishing, finding and binding such services. By virtue of the common, well-defined WSRP interfaces, all web services that implement WSRP plug into all WSRP compliant portals without requiring any service specific adapters – a single, generic adapter on the portal side is sufficient to integrate any WSRP service.

WSRP standardizes web services at the presentation layer on top of the existing web services stack, builds on the existing web services standards and will leverage additional web services standards efforts, such as security efforts now underway, as they become available. The WSRP interfaces are defined in the Web Services Description Language (WSDL). In addition, WSRP defines metadata for self-description for publishing and finding WSRP services in registries. All WSRP services are required to implement a SOAP binding and optionally may support additional bindings.

The OASIS WSRP standard will enable thousands of portals to aggregate content from tens of thousands of content and application providers offering hundreds of thousands of user-facing, pluggable web services for millions of end users/devices.

Table of Contents

Introduction 3
Vision..4
Portlets 5
Web Services..7
    Data-oriented Web Services used by Portlets 8
    Web Services for Remote Portals 9
WSRP Usage Scenarios 10
    Content Providers publishing WSRP Services 10
    Portals publishing local Portlets for remote use..11
The Web Services for Remote Portals Standard 11
    Related Standards 12
    WSRP Services – From simple to complex...12
        Simple WSRP Service – View only 12
        Interactive WSRP Service with persistent Entity State............................13
        Interactive WSRP Service with Entity State and Session State 14
        WSRP Service with Registration / Deregistration 15
    WSRP Life Cycles 16
    Invocation Context Information 16
    Markup Fragment Definitions..17
    Publishing, Finding, and Binding WSRP Services 17
Conclusion 18
References..18

Introduction

Portals provide personalized access to information, applications, processes and people. 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 composite 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, host integration, 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. Portlets are user-facing, interactive web application components rendering markup fragments to be aggregated and displayed by the portal.

Often, content is provided by external services and displayed by specific local portlets running on the portal. While this approach is feasible for establishing the base functionality of a portal, it is not well suited to enable *dynamic integration* of business applications and information sources into portals. As an example, 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 1 – a human resources portlet and a weather portlet run locally on the portal server and access remote web services to obtain the required information, i.e. the web services provide data and the presentation layer is implemented in specific portlets.

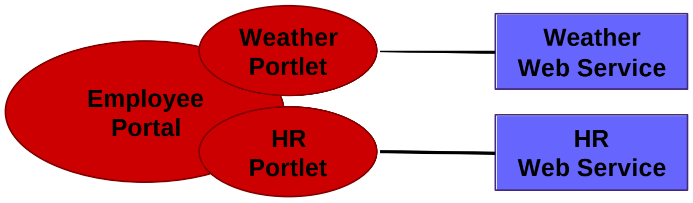


Figure 1: Local portlets using a remote web services

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 id. When the employee provides the data to the HR portlet, it invokes the HR 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 requests the most recent forecasts for the selected locations from the weather web service 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; the presentation layer has to be re-developed for each portal. To integrate HR information in the portal using local portlets as described above, 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, similar effort would be required for the weather portlet. In each case, there is significant cost and loss of time.

Obviously, it would be much more convenient if the HR and weather web services would include application *and presentation* logic and themselves produce markup fragments that are easy to aggregate at the consuming portal, as shown in

Figure 2.

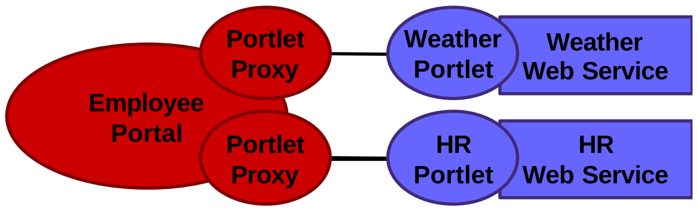


Figure 2: A Portal using Remote Portlet Web Services

Instead of just providing raw data or single business functions that still require special rendering on the portal side, ***Web Services for Remote Portals*** are user-facing, interactive web services including presentation. They are easy to aggregate and can be invoked through a common interface using generic portlet proxy code that is built into the portal. No special portlet code needs to be installed on the portal at all, re-implementation of the presentation layer on each portal is avoided. The use of generic portlet proxies consuming all remote portlet web services conforming to the common interface eliminates the need to develop service-specific portlets 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. Administrators can integrate remote portlet web services into a portal by using their portal's administration user interface to find and bind them with just a few mouse clicks. As a result, the portal creates new, generic portlet proxies bound to the remote portlet web services so that they are available to users in the same manner as local portlets. The remainder of this white paper explains WSRP vision, the notion of portlets and remote portlets, web services and the basic WSRP use-cases and concepts.

Vision

After the introduction of web services as the means for integration of business logic via the Internet, the web services-based WSRP standard will become the means for integration of web services based presentation components. WSRP will provide a standard that enables all content and application providers to provide their services so that they can easily be discovered and plug into all compliant portals without any programming effort on the portal's side.

Portal administrators can find and integrate the WSRP services they need with just a few mouse clicks, typically by using their portal's admin UI to browse a registry for WSRP services and selecting some for automatic integration into the portal. Portals act as intermediaries between end users and WSRP services and aggregate services from many different content providers. They often offload significant traffic from content providers by caching content, thereby enabling content providers to serve a huge number of users with little IT infrastructure.

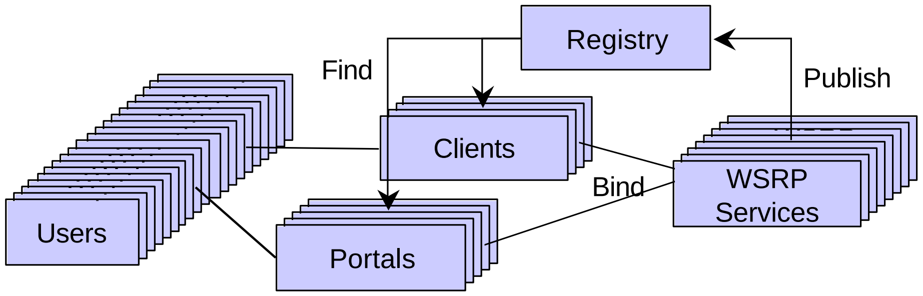


Figure 3: A Portal using Remote Portlet Web Services

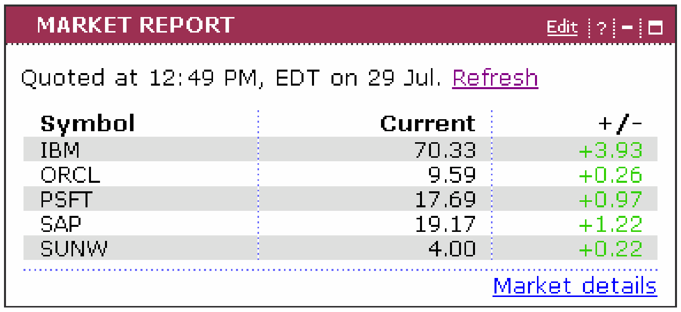
By providing WSRP services, content and application providers can leverage portals as multiplying intermediaries to reach a number of end users that they never could have reached otherwise.

Portlets

Portlets are user-facing, interactive web application components designed to be aggregated by portals and communicate with users through the hosting portal server. Typically, portal servers maintain a catalog of available portlets from which end users can select portlets for placement on portal pages.

All Portlets support a ***View Mode*** in which they provide their actual functionality that may range between rendering some static markup to a highly interactive application with a sophisticated screen flow. In addition, portlets may support an ***Edit Mode*** that provides a portlet specific UI to edit their instance data, a ***Help Mode*** that explains how the portlet should be used, a ***Design Mode*** that provides a portlet specific UI for changing the appearance of the portlet, and a ***Preview Mode*** that renders a preview of the portlet's appearance. The view, edit, help, and preview mode are typically exposed to end users of portals while the config and design modes are typically available to administrators only. In addition to the different modes, portlets can typically be displayed in different sizes, namely ***Minimized***, ***Normal***, and ***Maximized***.

A typical example for a portlet with multiple views that can be rendered in varying sizes is the market report portlet depicted in Figure 4. In the view mode in normal size, it displays stock quotes for a list of stock symbols; if the user wants to view details, she clicks on the "Market Details" link which directs the portlet to display an expanded set of information, but remain in the view mode. In the edit mode accessed by clicking on the Edit button, it allows users to customize the list of stock quotes displayed. In the help mode accessed by clicking on the question mark, the portlet informs the user how it works. By clicking on the minimize or maximize buttons, the user can switch the portlet to minimized or maximized mode, respectively.



Quoted at 12:49 PM, EDT on 29 Jul. Refresh

Symbol	Current	+/-
IBM	70.33	+3.93
ORCL	9.59	+0.26
PSFT	17.69	+0.97
SAP	19.17	+1.22
SUNW	4.00	+0.22

Market details

Figure 4: Example of a Portlet

Portlets can maintain state at different levels: Transient ***Session State*** typically reflects conversational state with a particular end user, persistent ***Configuration State*** that affects all instances of a portlet, and persistent ***Instance State*** that affects a single portlet instance. In the example above, the portlet uses transient session state to remember whether it is in the stock quote list screen or the details screen for each end user, persistent instance state to store the list of stock symbols a user is interested in, and configuration state to store the stock quotes feed to use.

Portlets may be local or remote to a portal server. We envision that typically, portals will use a mix of local and remote portlets as depicted in Figure 5 – depending on individual tradeoffs between proximity to the portal server and ease if integration.

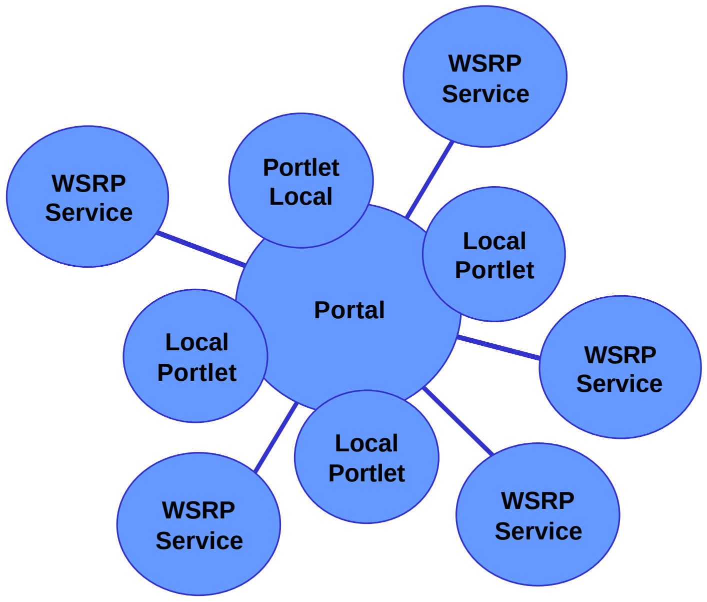


Figure 5: Local and remote Portlets

Local portlets are usually tightly integrated with portal servers and typically run on the same physical server or cluster of servers. Remote portlet web services run on remote servers at other places in the intranet or the Internet and are loosely coupled to the portal server.

Figure 6 shows an example of a high-level portal architecture that may be employed to allow for combined use of local and remote portlets as well as making local portlets available for other portals.

Most portal clients access the portal via the HTTP protocol, either directly or through appropriate 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.

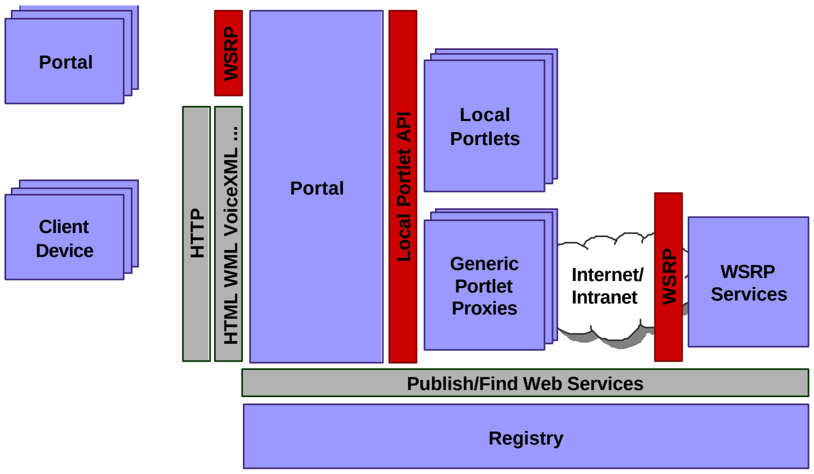


Figure 6: Example of a Portal Architecture exploiting WSRP

When aggregating pages for end users, the portal invokes all portlets that belong to a user's page through a local ***Portlet API***. This may be a portal vendor specific API or in the near future a standard API, like the Portlet API defined in JSR 168 (see [4]) for the Java programming language. 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 variety of remote portlet web services without any installation effort or code running locally on the portal server.

Also, portals may wrap local portlets and publish them as remote portlet web services for integration by other portals. Conversely, remote portlet web services can be integrated into portals by wrapping them in a proxy written to the local portlet API.

Web Services

The concept of web services has been developed to allow business applications to communicate and cooperate over the Internet. Registry standards enhance this by defining how the web services may be published, found and bound with minimal human interaction (see

Figure 7).

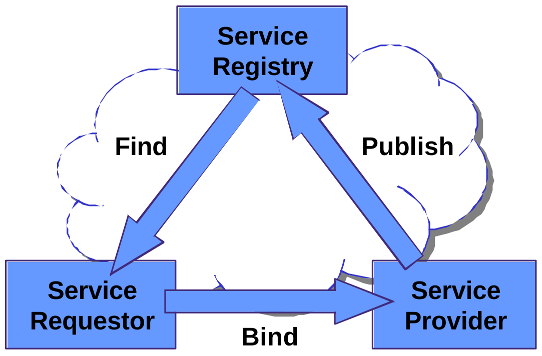


Figure 7: Publish, Find, Bind

Web services allow objects to be distributed across web sites where clients can access them via the Internet. Global or corporate service registries are used to promote and discover distributed services. A consumer that needs a particular kind of service can make a query to the global service registry to find services that match their requirements. The consumer can select one of the services, bind to that service, and use it for a certain period of time.

The most important standards in this area are the *Simple Object Access Protocol* (SOAP, see [1]) for communication between web services, and the associated *Web Services Description Language* (WSDL, see [2]) for formal description of web service interfaces and bindings. Additional standards are currently emerging in the areas of security, identity, reliability, and transactions.

From a portal perspective, we can differentiate between two different kinds of web services – the "traditional" data oriented web services and presentation oriented, user-facing, interactive web services (see Figure 8).

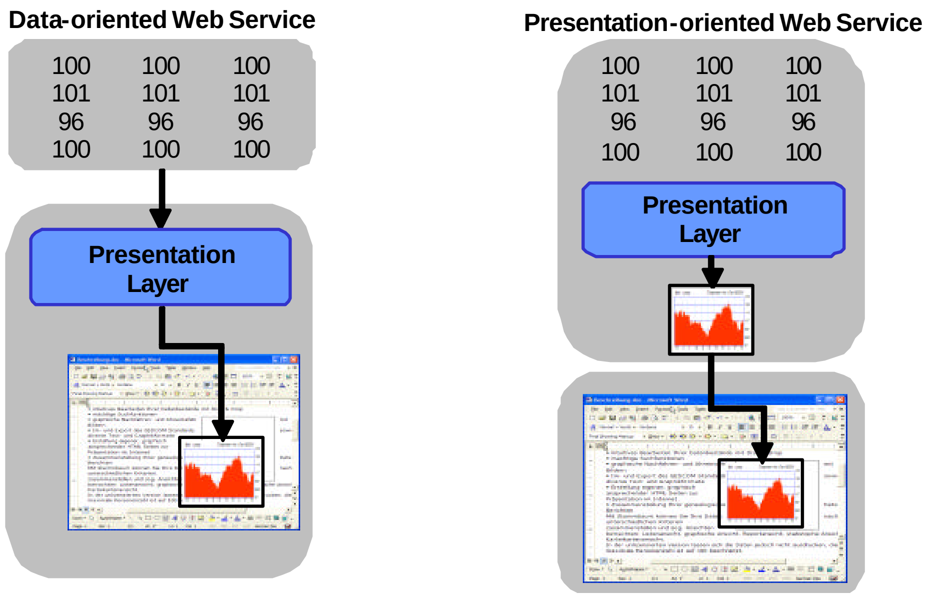


Figure 8: Data-oriented Web Services compared to Presentation-oriented Web Services

Data-oriented web services are web services that receive requests and return data objects encoded in XML documents in the response. The signatures of their operations as well as the structure and semantics of the returned data are typically service type-specific. It is the responsibility of the service consumer to process the received data in a service-specific manner and generate any required presentation. While this is a good approach for applications that require specific data and know how to consume and process this data, it is not appropriate for portals that need to be able to quickly integrate content and applications form various sources.

User-facing web services include presentation and optionally interaction as a part of the service itself. They don't just provide raw data to be processed and turned into a presentation by the consumer, but instead produce markup fragments that can easily be aggregated by their consumers, e.g. portals.

Data-oriented Web Services used by Portlets

Portals usually allow portlets to access data-oriented web services to obtain data from remote systems or drive transactions. When using data-oriented web services with a given interface, the portlets need to contain service interface-specific code that matches the web services' operations and their particular signatures.

When a portlet receives a request that requires invocation of a remote service, the portlet makes calls on a service-specific proxy. The proxy takes the parameters, marshals them into a programming language-independent request, and sends this request to the remote web service. The web service receives the request, unmarshals the parameters and invokes the web service implementation with these parameters. When the service implementation returns the result, the web service marshals the result into a programming-language independent response and sends it back to the proxy on the consumer's side. The proxy unmarshals the result data and finally returns it to the portet as the appropriate object (see Figure 9).

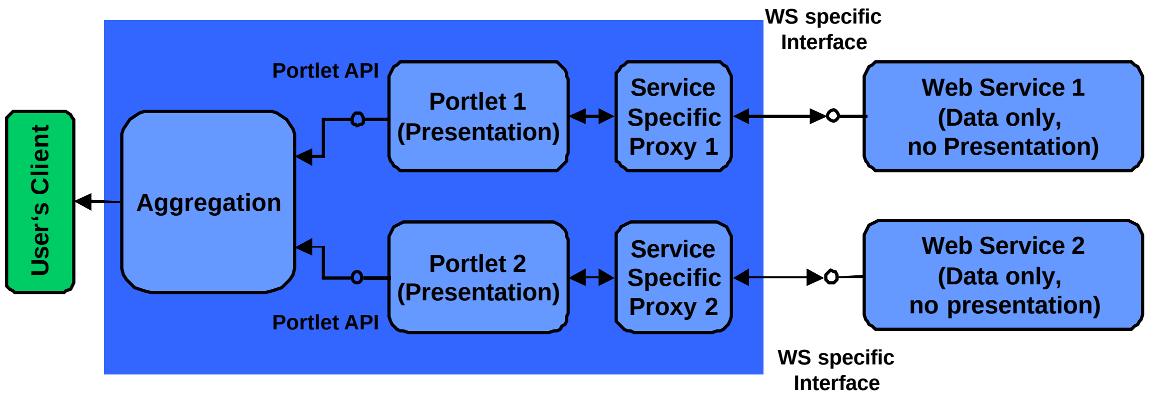


Figure 9: Portal consuming Data-oriented Web Services using service-specific Portlet Code

Web services can be formally described using WSDL interface descriptions that can be used by appropriate tools to generate service interface-specific proxies for different programming languages. This way, a part of the service specific portlet code can be generated automatically. To simplify writing portlets using data-oriented web services, tools can automatically produce the appropriate proxy code for particular programming languages from a web services WSDL interface description.

Portlets that consume data-oriented web services via a proxy use the service and language specific proxy and thus indirectly depend on the particular operations of the data-oriented web service that determine the proxy's methods. They need to have knowledge of how to make calls to the proxy and how to process and render the results. This means that in addition to the service-specific proxies, the individual portlets contain significant amounts of code tailored to the particular web service's interface.

Web Services for Remote Portals

In order to allow for dynamic integration in portals, remote portlets can be provided as user-facing, interactive web services conforming to a well-defined, common interface description defined in WSDL. Such remote portlets may be implemented in any programming languages as long as they adhere to the common interface description.

When invoking a remote portlet, portals typically use a *generic* portlet proxy to invoke the user-facing, interactive web service. The portal invokes the generic portlet proxy exactly like it would invoke a local portlet. The generic portlet proxy marshals all parameters into a request and sends it to the remote portlet. Since all remote portlet web services adhere to the same service interface definition, the same proxy can be used for all of them. The remote portlet web service unmarshals all information in the request and calls on the remote portlet's implementation.

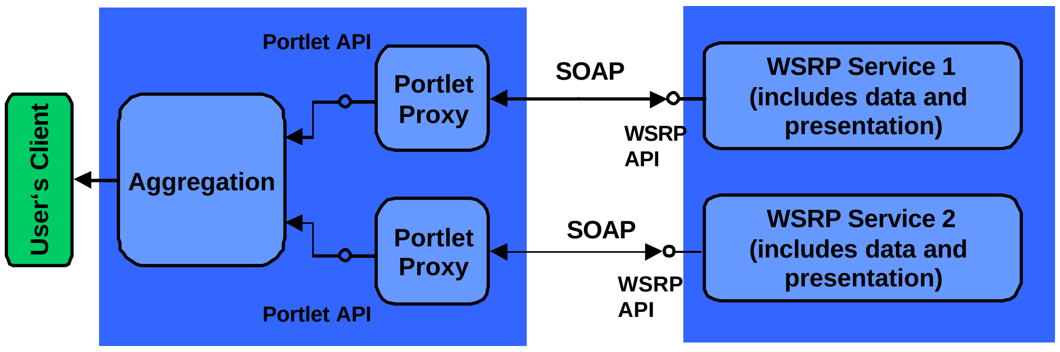


Figure 10: Portal consuming WSRP Services using Generic Portlet Proxies

The result is then marshaled into a response and sent back as the reply to the proxy that in turn unmarshals the response for the portlet proxy that finally returns a result object to the portal engine that initiated the request, just like a local portlet would.

The big difference between WSRP services and data-oriented web services is that they are user-facing, interactive services that include presentation and all have one common interface. This means that they simple plug into portal servers and do not require any service specific code to run on the consuming portal.

WSRP Usage Scenarios

In this section we present two major usage scenarios that show how the capabilities of WSRP can be exploited by portals and content providers.

Content Providers publishing WSRP Services

Today, many 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 may provide some portlets supporting particular content providers out of the box, it is typically necessary to develop and install additional portlets for the remaining content providers, i.e. the party that runs the portal spends a lot of money and effort in order to integrate a rich set of content from different providers. 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. It also limits their capability to excert some control over the way their content is rendered by the subscriber's portal.

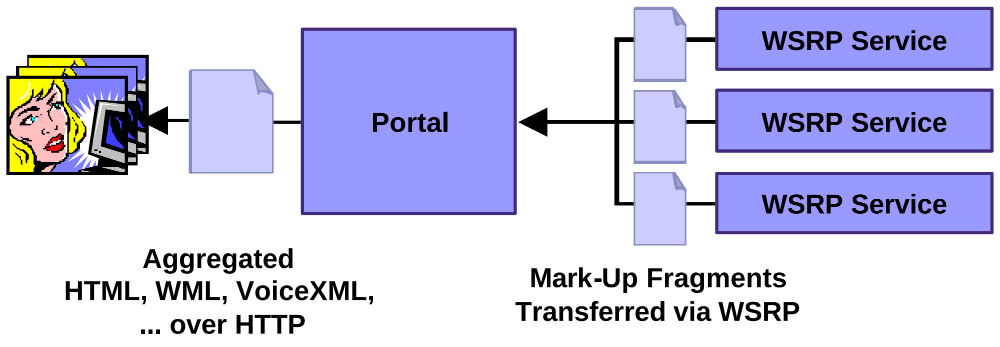

Portal
WSRP Service
WSRP Service
WSRP Service
Aggregated HTML, WML, VoiceXML, ... over HTTP
Mark-Up Fragments Transferred via WSRP

Figure 11: Content/Application Providers providing WSRP services

In order to allow for easy integration of their content in portals, content providers can use WSRP to surface their content as remote portlets and publish them as WSRP services in the public, global directory.

In order to provide this value add to subscribers, the content provider serves remote portlets via the desired bindings in addition to the classical content server. Once the content provider has published a WSRP service in registry, administrators of portals who wish to use content from the content provider can simply look up the content provider's business entry in the registry and bind to the WSRP service that provides 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.

Portals publishing local Portlets for remote use

While portals traditionally have operated in isolation from each other, large corporations are now demanding cooperation between their portal installations. 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 consumes a service provided by the human resources function within a corporation.

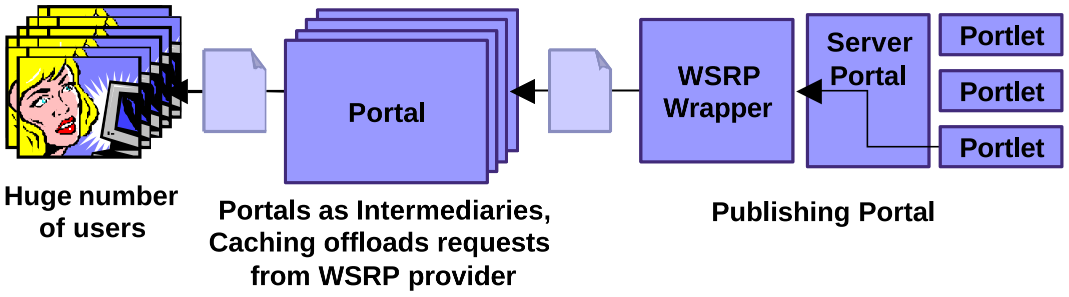


Figure 12: Portal sharing portlets as WSRP services

Now let us assume the human resources department runs a portal that provides various HR related portlets. Some are only intended for use by HR staff like a Payroll Portlet or a Staff Record 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 for example has a corporate registry only accessible from the intranet, an HR portal administrator would use a portal server's publish function to create remote portlet web service entries for the Variable Pay Portlet and the HR Info Portlet in the corporate registry. Thus these portlets become available for integration in other portals in the corporation – e.g. the administrator of the corporation's employee portal can find the remote portlets web services published by the HR portal using his portal's built-in registry browser and integrate them into his portal with a single click.

The Web Services for Remote Portals Standard

In this section we provide an overview of the WSRP standard. We first position WSRP in relation to other standards and then explain the WSRP concepts based on examples starting with a very basic WSRP services and going towards more complex WSRP services.

Related Standards

WSRP fits into the greater context of the web services standards stack. It on WSDL to formally describe the WSRP service interfaces, SOAP can be used for invocations of WSRP services. Furthermore, WSRP has overlap with WSIA (Web Services for Interactive Applications) with which it shares a common base of interface and protocol definitions.

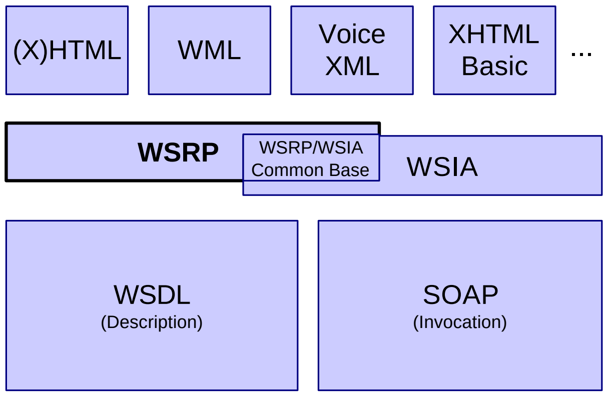


Figure 13: WSRP and related Standards

WSRP defines the notion of valid fragments of markup based on the existing markup languages such as HTML, XHTML, VoiceXML, cHTML, etc. For markup languages that support style definitions, WSRP also defines a set of standard style names to allow portlets to generate markup using styles that are provided by WSRP compliant portals so that their markup fits nicely into the look and feel of the consuming portal.

WSRP Services – From simple to complex

A goal of the WSRP standard is to make it very easy to implement very simple services that just provide markup fragments but also allow for more complex services that require consumer registration, support complex user interaction and operate based on transient and persistent state. In this section we give an overview of the different levels of functionality that WSRP enables and explain the relevant parts of the WSRP interfaces and protocol.

Simple WSRP Service – View only

The simplest possible WSRP service provides a single view, without any user interaction. An example is a schedule of flights leaving from an airport. For such a simple WSRP service, implementing a `getMarkup` operation that returns a ***WSRP Markup Fragment*** for the current flight schedule is sufficient.

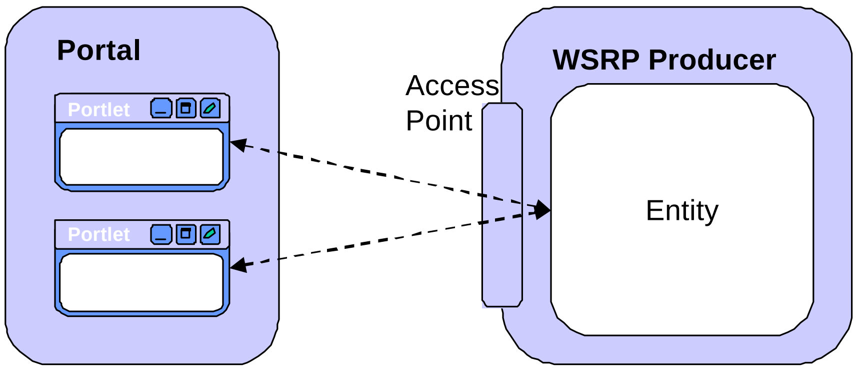


Figure 14: Simplest WSRP Service

Interactive WSRP Service with transient conversational State

A slightly more complex case is a WSRP service that supports user interaction and maintains conversational state reflecting user interaction. An example is a news service that provides an overview of headlines of the different news articles and allows users to click on the headlines to navigate to the individual articles and on a back-link. Such a service may want to track conversational state within a ***WSRP Session*** to always display the correct view for a particular user and return an ID for an internally managed session in each response of the `getMarkup` operation. The markup returned in responses may contain action links that will trigger subsequent invocations of the `performInteraction` operation.

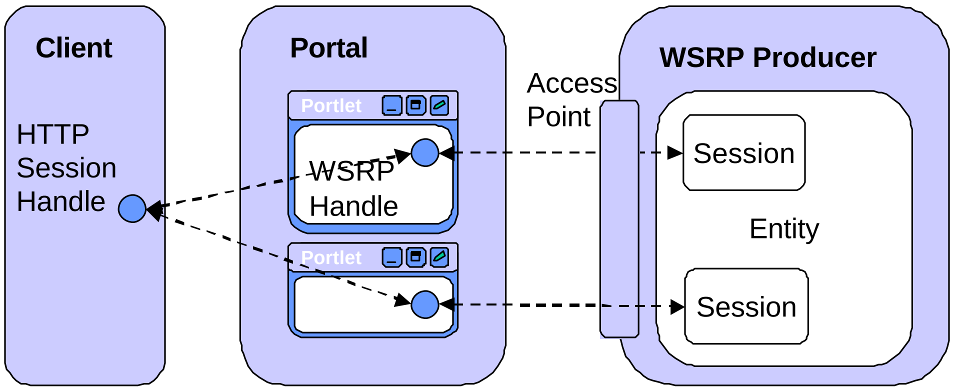


Figure 15: WSRP Entities and their relation to Portlet Instances in a Portal

Interactive WSRP Service with persistent Entity State

At about the same level of complexity of the last example, let us consider a WSRP service that maintains persistent state that can be associated with individual portlet instances available from the WSRP producer. An example for such a service is a stock quotes service that allows individual users to define their own personal portfolios. This use case requires the concept of multiple persistent ***Entities.***

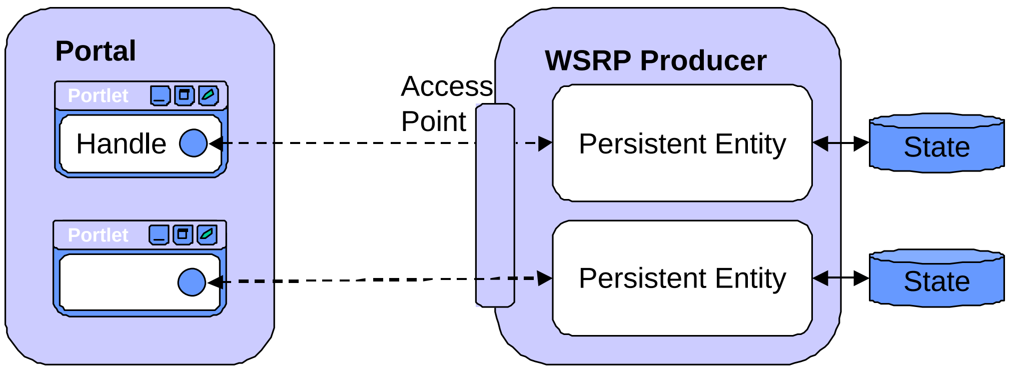


Figure 16: WSRP Entities and their relation to Portlet Windows in a Portal

Consumers create new persistent entities by invoking `cloneEntity()`, specifying an existing entity – either a producer offered entity or one previously created by the consumer. The new entity will be initialized with the same (persistent) state as the existing entity. As a result, the consumer obtains an entity handle for referring to the entity when calling the producer. When an entity is no longer needed, it can be discarded by calling `releaseHandles()`, passing the entity handle.

Interactive WSRP Service with Entity State and Session State

A more complex WSRP producer may employ both persistent entity state as well as transient session state. Zero of more WSRP sessions may be associated with a persistent entity at a given time. For example, many WSRP sessions to the same entity may exist for a consumer that is a portal with shared pages referencing it and being used concurrently by multiple end users (see Figure 17).

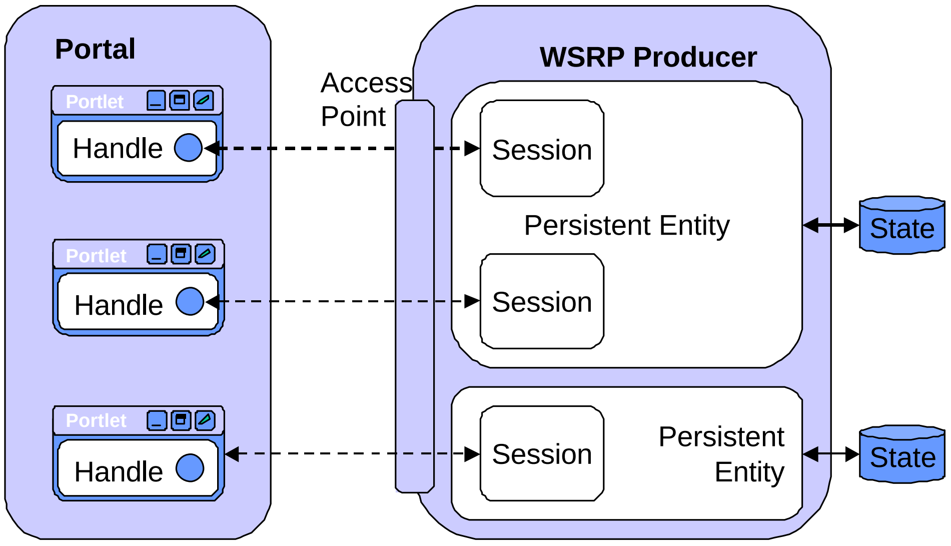


Figure 17: WSRP Producer employing persistent Entity State as well as Session State

A typical usage pattern for interactive WSRP Services with entity and session state is shown in Figure 18; in this example the consumer is a portal using a WSRP producer as a remote portlet.

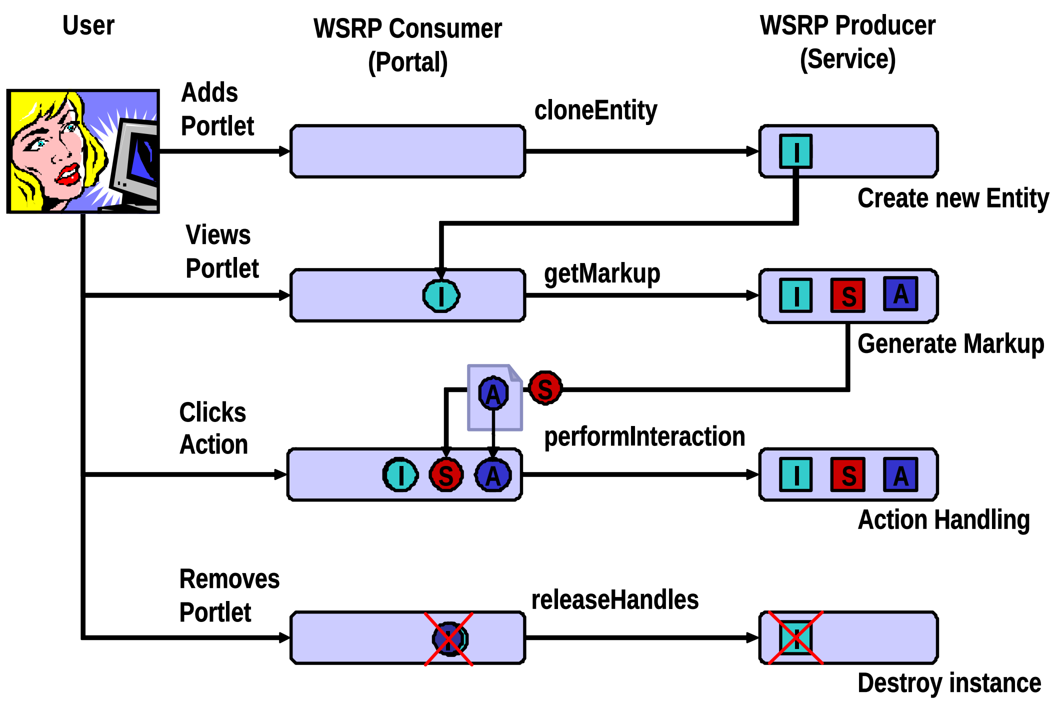


Figure 18: Interaction Diagram for an interactive WSRP Service with Entity State

When an end user adds a portlet to a page in the portal, the portal invokes `cloneEntity` operation on the WSRP service specifying the producer offered entity corresponding to the portlet to obtain a new entity handle ***I*** that it stores associated with a newly created portlet instance on the portal side.

When the user views the page containing the new portlet instance, the portal determines the entity handle and uses it to make a call to the `getMarkup` operation of the WSRP service to obtain the markup fragment to be displayed. The returned markup may contain action links ***A*** and/or a session handle ***S*** if the WSRP service wants to maintain session state. The portal may need to rewrite any action links to make them work in the final markup sent to the browser and must store any returned session handle to provide it with each subsequent request.

When the user clicks on a link in the markup a request is sent from the browser to the portal, the portal intercepts the request and maps it to an invocation of `performInteraction` operation of the WSRP service, passing the session handle to allow the WSRP service to look up associated session state. Upon a `performInteraction` call, the WSRP service typically changes state. When the `performInteraction` operation returns, the portal refreshes the page which results in a call to `getMarkup` to starts a new user-interaction cycle.

When an end user does not need a portlet instance anymore and discards it from a portal page, the portal determines the handle of the entity which is no longer needed and invokes `releaseHandles` on the WSRP service. The WSRP service must discard the entity and may release any related resources.

WSRP Service with Registration / Deregistration

WSRP providers that do not just support access by anonymous consumers need to implement appropriate operations for registration and deregistration of consumers. To register with such a service, the consumer calls the `registerConsumer` operation. To deregister, the consumer calls the `releaseHandles` operation.

WSRP Life Cycles

WSRP services may maintain state in the scope of consumer registrations, entities, and sessions which have nested life cycles. A consumer can create a consumer registration by calling the `registerConsumer` operation of the WSRP service. In the scope of a consumer registration, a consumer can create entities by calling the `cloneEntity` operation. In the scope of an entity, the WSRP service may create sessions, typically per user, that expire after a duration of inactivity.

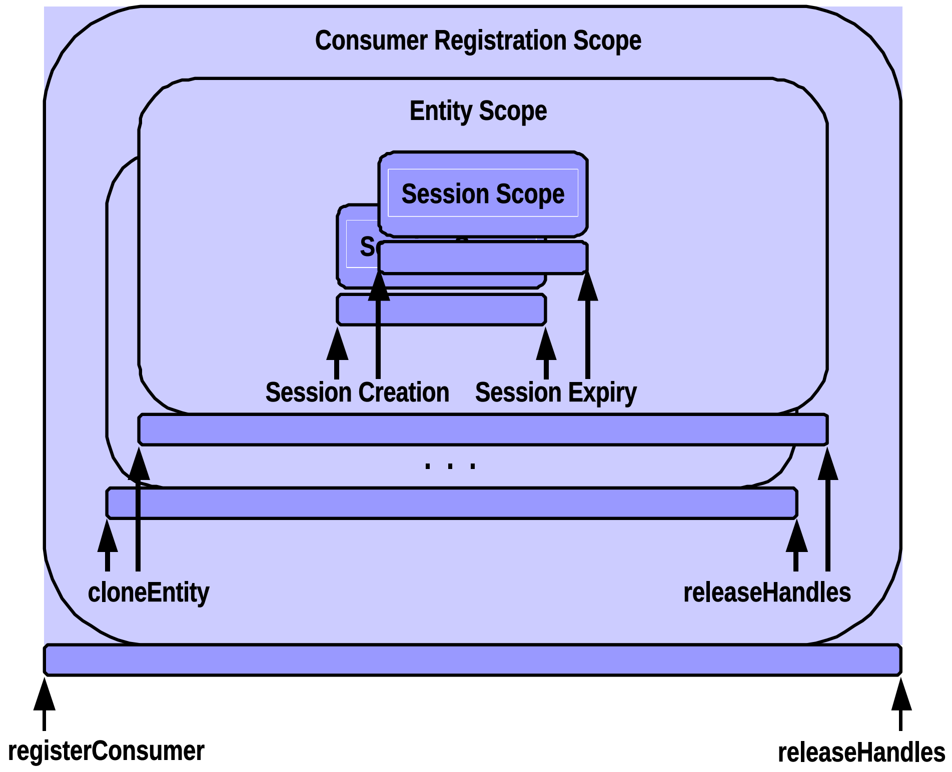


Figure 19: WSRP Life Cycles

When a consumer does not need an entity anymore, it calls the `releaseHandles` operation, passing the entity handle, to allow the WSRP service to release all associated resources and all sessions within the entity scope. When a consumer does not need the WSRP service anymore, it calls the `releaseHandles` operation, passing the consumer registration handle.

Invocation Context Information

When a consumer invokes a WSRP producer, it passes context information with the invocation, including information about the consumer as well as the end user on whose behalf the request is sent. Examples include the type of device the end user has and its required markup language, the preferred language, a user identifier and optionally user profile information. For the user profile information, WSRP defines attribute names for the commonly required information like name, address, affiliation, phone, FAX and e-mail.

Markup Fragment Definitions

In addition to a web services interface and protocol, WSRP standardizes markup fragments returned by WSRP services for the relevant markup languages including XHTML, HTML, WML, VoiceXML, cHTML. WSRP specifies the tags that may be used for the individual markup languages, defines a general URL and name spacing scheme and defines common styles for the markup languages supporting stylesheets.

Publishing, Finding, and Binding WSRP Services

Content or application providers who want to offer remote portlet web services can publish their service entries registry, referencing the WSRP interface description. The information that is published typically includes:

- Name and Description
- Supported markup types, locales, and modes
- Cachability information
- Key words and parameters

Once a remote portlet has been published, portal administrators can use their portal administration tools to search the registry for web services that implement the WSRP interface and make some of the matching portlet web services available for their users by adding them to their portal's portlet registry (see

Figure 20).

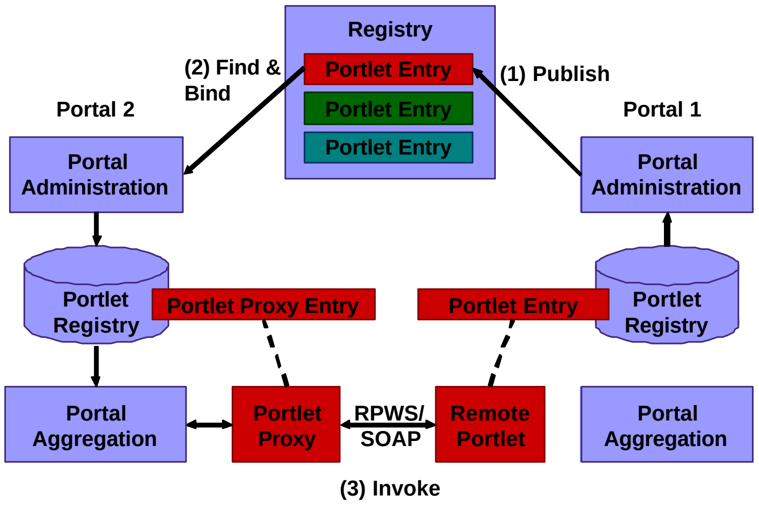


Figure 20: Publishing, finding and binding Remote Portlets

Once the portlets are in the registry, users can select them to be displayed on their personal pages just like local portlets. In portals, the mechanisms for publishing portlets as remote portlet web services, finding remote portlet web services in a registry, binding to them and using remote portlets can be seamlessly integrated in the portal's administration user interfaces. There are four different dialog flows that are typically relevant to use of WSRP services:

- **Managing Registries:** Administrators manage a list of registries they wish to use for querying and publishing WSRP services.
- **Publishing Portlets:** Administrators can publish portlets to make them available for use by other portals as WSRP services.

- **Finding and binding Remote Portlets:** Administrators find WSRP services in a registry and bind to these services to make them available for portal users.
- **Discovering Capabilities:** While many of the capabilities of a WSRP service may have been discovered through the find/bind step involving the registry, the `getDescription()` operation on the service returns the full capabilities of the service.
- **Using Remote Portlets:** Users select and transparently use WSRP services that have been integrated by administrators, just as easily as local portlets.

In addition to advertising WSRP services in a registry, there are alternative ways of finding and binding WSRP services. For example, within a corporation, a portal administrator may obtain the URL of a WSRP service directly from a department that wishes to have their WSRP service integrated into the corporate employee portal. For such a case, the portal may allow the administrator to enter the URL manually instead of getting it by browsing a registry.

Conclusion

By defining a standard for user-facing, interactive web services that plug into portal servers without any programming effort, Web Services for Remote Portals (WSRP) lays the foundation for a large variety of remote portlets to be offered by providers in the Internet as well as intranets. WSRP will enable interoperability of portals by allowing them to consume remote portlets provided by other portals or content / application providers as well as sharing local portlets by providing remote access to them to other portals in a standardized manner.

References

1. *SOAP Version 1.2, W3C 2002*
   http://www.w3.org/TR/soap12-part1/

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

3. *OASIS Web Services for Remote Portals Web Site*
   http://oasis-open.org/committees/wsrp

4. *Portlet API (JSR 168)*
   http://www.jcp.org/jsr/detail/168.jsp