Web Services for Remote Portals (WSRP)

Note 21 January 2002

This version:

http://www.ibm.com/developerworks/library/ws-wsrp

Authors:

Angel Luis Diaz, IBM <aldiaz@de.ibm.com>

Peter Fischer, IBM <peter.fischer@de.ibm.com>

Carsten Leue, IBM <cleue@de.ibm.com>

Thomas Schaeck, IBM <schaeck@de.ibm.com>

Editor:

Thomas Schaeck, IBM <schaeck@de.ibm.com>

Abstract

Web Services for Remote Portals (WSRP) are visual, user-facing web services centric components that plug-n-play with portals or other intermediary web applications that aggregate content or applications from different sources. They are designed to enable businesses to provide content or applications in a form that does not require any manual content- or application-specific adaptation by consuming intermediary applications. As Web Services for Remote Portals include presentation, service providers determine how their content and applications are visualized for end-users and to which degree adaptation, transcoding, translation etc may be allowed.

WSRP services can be published into public or corporate service directories (UDDI) where they can easily be found by intermediary applications that want to display their content. Web application deployment vendors can wrap and adapt their middleware for use in WSRP-compliant services. Vendors of intermediary applicatios can enable their products for consuming Web Services for Remote Portals. Using WSRP, portals can easily integrate content and applications from many internal and external content providers. The portal administrator simply picks the desired services from a list and integrates them, no programmers are required to tie new content and applications into the portal.

To accomplish these goals, the WSRP standard defines a web services interface description using WSDL and all the semantics and behavior that web services and consuming applications must comply with in order to be pluggable as well as the meta-information that has to be provided when publishing WSRP services into UDDI directories. The standard allows WSRP services to be implemented in very different ways, be it as a Java/J2EE based web service, a web service implemented on Microsoft's .NET platform or a portlet published as a WSRP Service by a portal. The standard enables use of generic adapter code to plug in any WSRP service into intermediary applications rather than requiring specific proxy code.

WSRP services are WSIA component services built on standard technologies including SOAP, UDDI, and WSDL. WSRP adds several context elements including user profile, information about the client device, locale and desired markup language passed to them in SOAP requests. A set of operations and contracts are defined that enable WSRP plug-n-play.

Status of this Document

This specification is being submitted to the OASIS group for consideration. This is a proposal only.

Table of Contents

---

1. Introduction

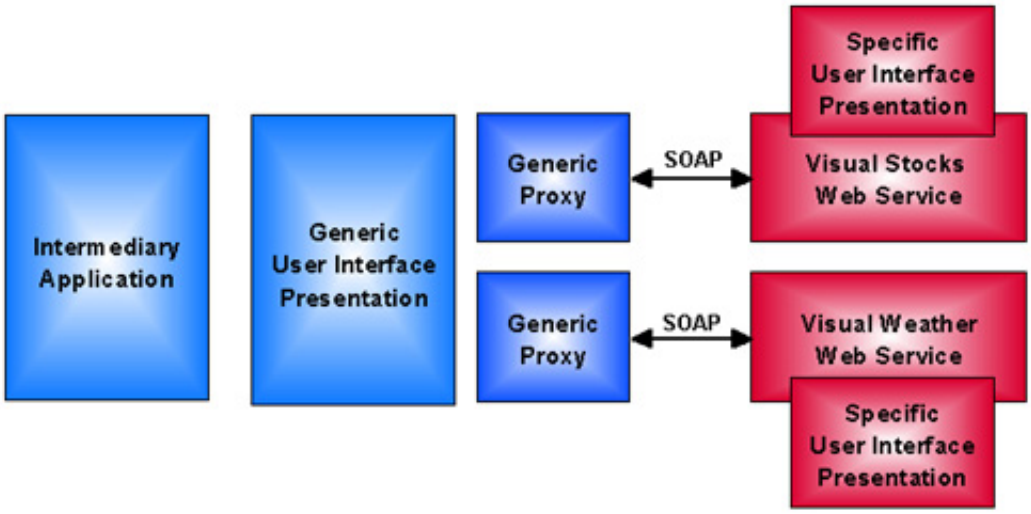
In this Note, we define pluggable, user-facing visual web services named Web Services for Remote Portals (WSRP) as a paradigm that complements the "traditional" form of purely data oriented web services. For the "traditional" web services, the typical usage pattern is that a client invokes a special operation of the web service, providing input parameters in a SOAP request and the web service processes the input parameters and creates a result that is sent back in a SOAP response. A good example for this kind of web services is a stock quote web service: A client sends the stock symbols for which it wants quotes to the stock quote web service in a SOAP request. The stock quote web service gets the symbols from the request, looks up the current price for each stock symbol and sends the list of prices back to the client in a SOAP response. It is important to note that the interfaces of data oriented web

services typically depend on the type of data or function they provide. For example, the stocks services may have functions like getAvailableStockSymbols and getCurrentStockPrices while the weather service may have functions such as getAvailableCities and getWeatherInformation.

Figure 1: Data-oriented web services require specific presentation components at the intermediary application

Data-oriented web services do not include any user interaction or presentation functionality; there always need to be intermediary applications that provide a user interface, using the data-oriented web service under the covers. To do so, it must know the interface of the web service, or in other words it must be written to conform to the web serviceâ€™s particular WSDL interface definition. This means that code is required that is specific to the particular kind of data-oriented web service - for example code for using a weather web service would be very different from code using a stock quote service (see Figure 1). The data-oriented web services approach does not allow providers to add the value of an end-user interface to their services. Considering the stock quote example, the stock quote provider might want to provide an additional, visual web service with an end user interface and presentation that can be plugged into intermediate applications like portals rather than just providing a simple, data-oriented web service that will require dedicated programming work for integration on the intermediate applicationâ€™s side. Doing so allows intermediary applications to simply aggregate the obtained presentation fragments into the larger context of their overall presentation rather than forcing them to create their own user interfaces and presentation code around the data provided by a purely data-oriented web service.

Figure 2: Data-oriented web services require specific presentation components at the intermediary application



When implementing visual, user-facing web services, there are two options - using custom sets of possible events and output operations or restricting the services to a concrete, well-defined set of interfaces and contracts. Web Services for Interactive Applications (WSIA) specification will provide a framework for creating visual web services of different kinds.

The WSRP services definition shares basic interfaces with WSIA, and defines a specific set of interfaces and contracts on top of the base component interface that are concrete and comprehensive enough to allow plugging any WSRP compliant service together with any WSRP compliant client. It will however be possible to use the Web Services for Remote Portals using standard WSIA operations only.

2. Portals and WSRP

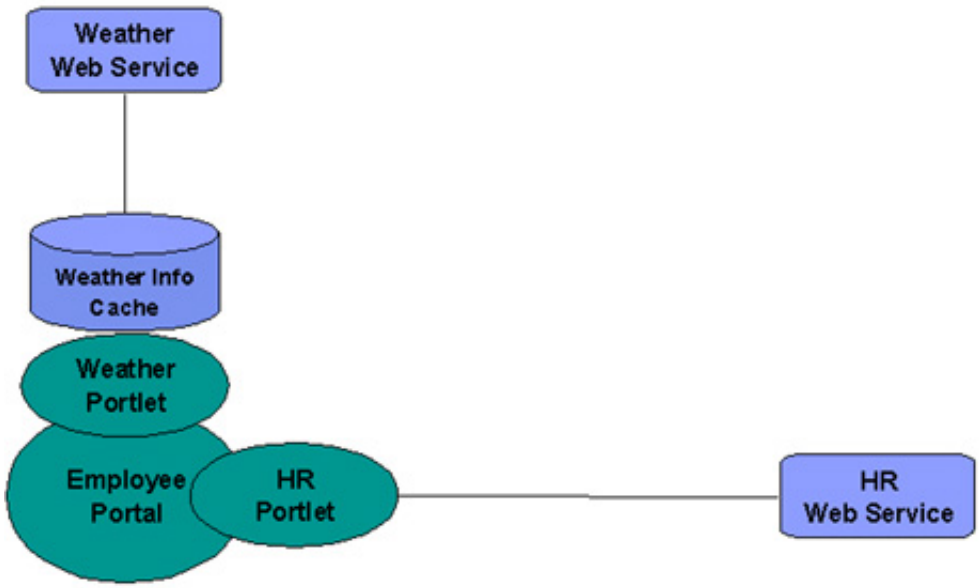
While WSRP services can be consumed by different kinds of intermediary applications, they are of particular importance for portals. For this reason, we describe their use in portals in some detail in this section. Portals are focal points for users to access information and applications from many different 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 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, etc.

Different rendering and selection mechanisms are required for different kinds of information or applications, but all of them rely on the portalâ€™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 todayâ€™s portal implementations provide a component model that allows plugging components referred to as Portlets into the portal infrastructure.

2.1. The WSRP Concept applied to Portals

While local web application components like portlets provide short response times, they are not well suited to enable dynamic integration of business applications and information sources. 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 data-oriented remote web services to obtain the required information.

Figure 3: Example of local portlets using web services



This approach only works if all portlets are physically installed at the employee portal and 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, similarly for the weather portlet. In each case, significant effort and local installation of additional code are required to make the portlets available.

Obviously, it would be much more convenient from a portal perspective if web services would appear as visual, user-facing services including presentation and application logic as shown in Figure 4. If a standardized web services interface exists, such visual, user-facing web services can be invoked through generic portlet proxies on the portal side.

Use of generic portlet proxies eliminates the need to develop specific portlets for each web service to plug into the portal. 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 visual, user-facing service. Through the use of portlet proxies, remote visual, user-facing web services appear to portals and can be selected by users just like local portlets.

Figure 4: Example of a portal using remote portlets

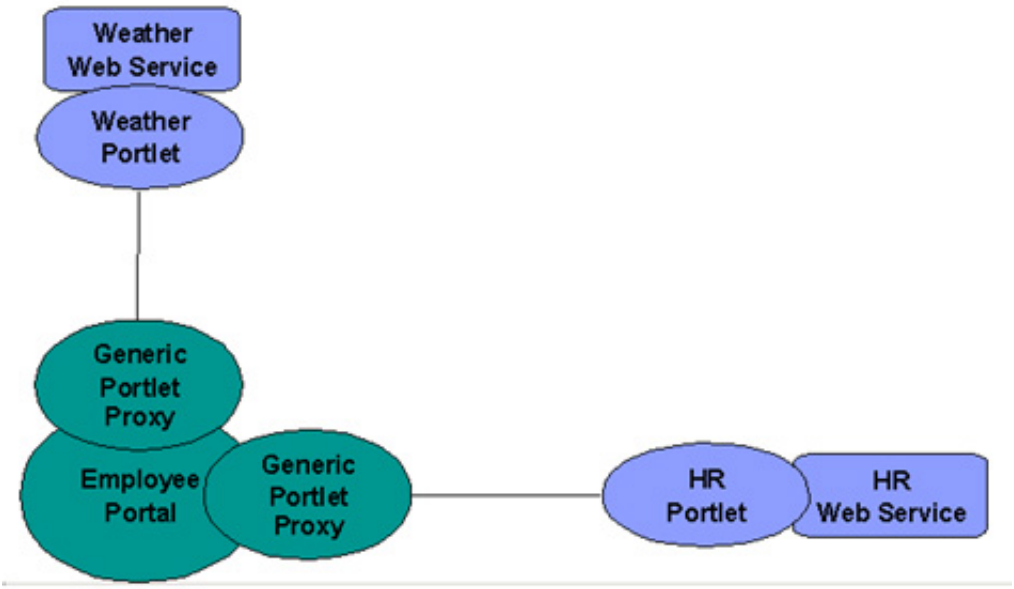
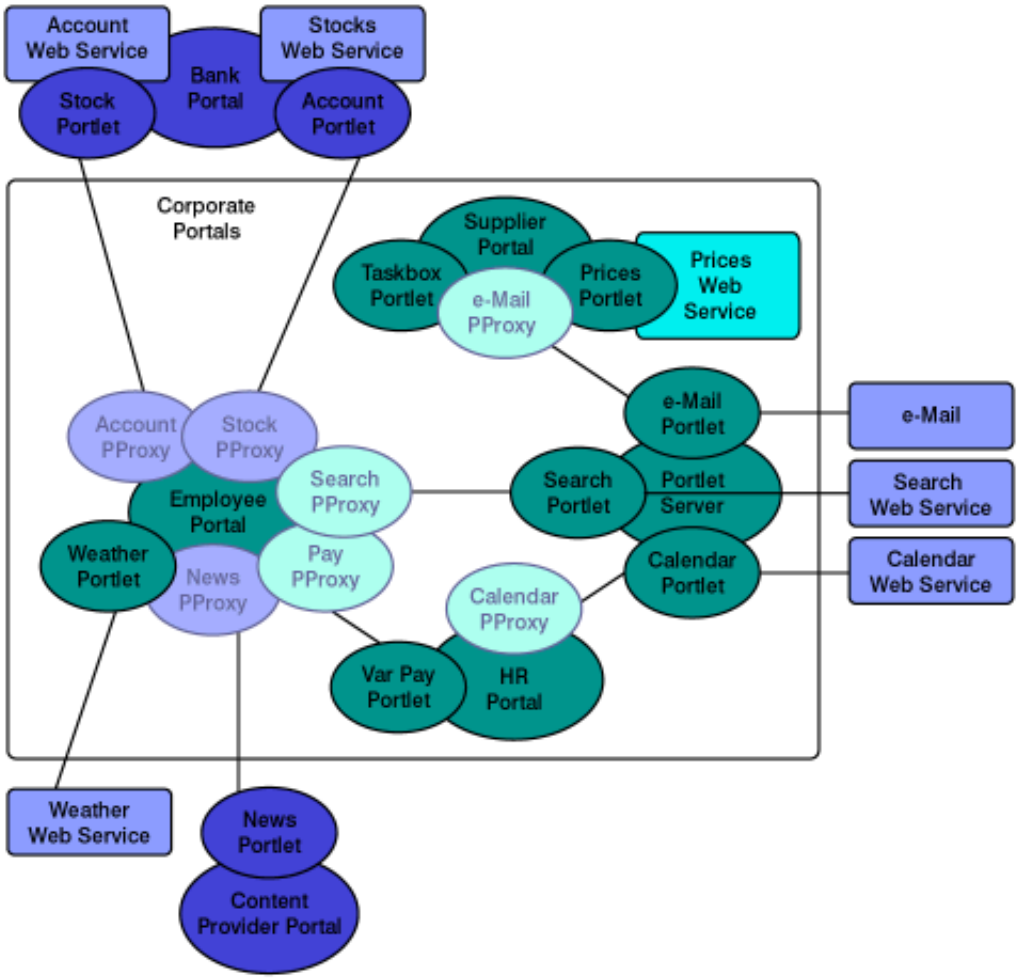


Figure 5 gives an example of a corporation that owns an employee portal, a supplier portal and a human resources portal. The employee portal has a weather portlet that runs on the local portlet container while the account, stock, search, variable pay and news portlets run remotely and are accessed by the employee portal through portlet proxies. The account and stock portlet reside on a bankâ€™s portal while the news portlet runs on a content providerâ€™s portal. 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.

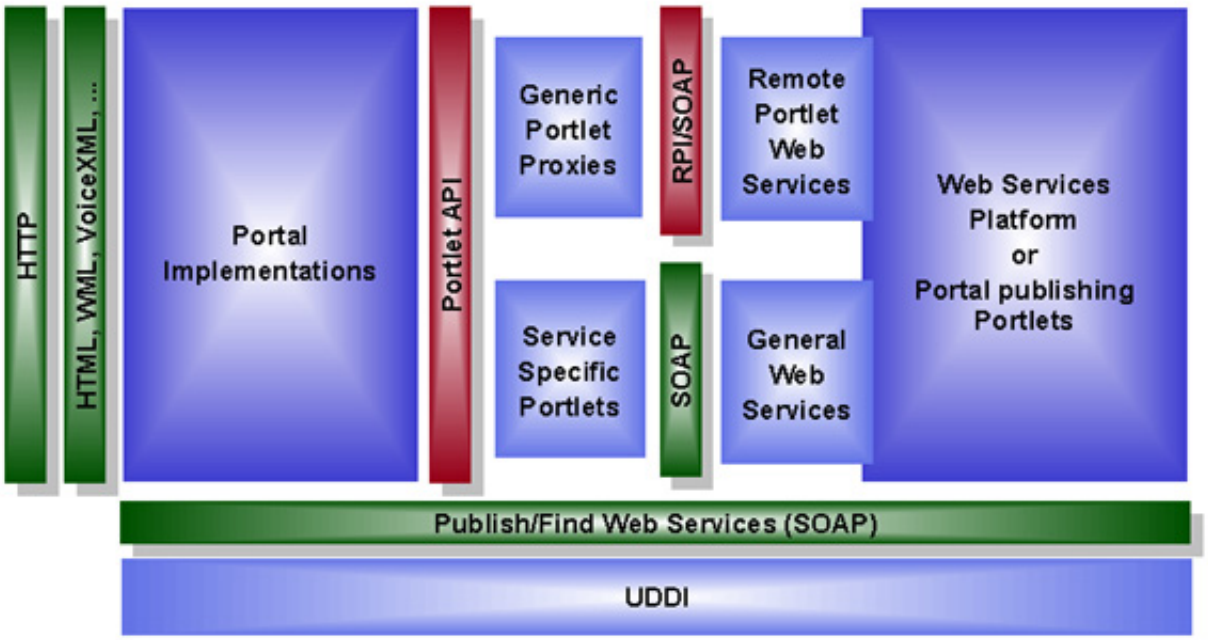
Figure 5: A distributed portal solution based on remote portlets and web services



As of today, the business problems described here are not addressed by appropriate standards. There is no standard in place that defines visual web services so that they can be found and plugged into intermediary applications like portals in a simple manner. The Web Services for Remote Portals (WSRP) standard aims to allow for interoperability between different kinds of intermediary applications and visual, user-facing web services.

To realize this goal, an architecture is required that defines the relevant building blocks and interfaces and protocols between them. The architecture needs to cover all the way from client devices over portals to visual web services to be included in portals. Figure 6 shows an open portal architecture that addresses the areas mentioned above and that lets us position WSRP services in relation to portals.

Figure 6: An open portal architecture



The architecture assumes that clients access portal implementations via the HTTP protocol, either directly or indirectly through appropriate gateways, e.g. WAP gateways or voice gateways. The mark-up languages used by the different devices may be different, for example 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 and likewise components that plug into portals need to be able to handle different device types.

When aggregating pages for portal users, portals typically invoke all portlets belonging to a userâ€™s page through a Portlet API for locally installed portlets. We differentiate two different kinds of portlets:

- Local Portlets run on the portal server itself. They may be deployed by installing portlet archive files on portal servers and are typically invoked by the portal server directly through local method calls.
- Remote Portlets run as web services on remote servers that are published in a UDDI directory to allow for easy finding and binding. Typically portlet proxies (generic local placeholders) will invoke WSRP services to which they are bound via the SOAP protocol.

While local portlets usually provide the base functionality for portals, remote portlets can provide a large number of additional functions without installation effort or need for third-party code to run locally on the portal server. Web Services for Remote Portals may be implemented as dedicated web services or by a portal with an adapter that exposes local portlets as WSRP services.

We can identify the areas that require standardization in order to ensure interoperability across all layers:

- Portlet API: Portlet APIs need to be standardized for different programming languages. The Java Portlet API will defined in the Java Community Process, versions for other languages may be defined similarly in other standards bodies.
- Web Services for Remote Portals: WSRP services can be defined and standardized in a programming language independent manner, based on the WSDL, UDDI, SOAP, and WSXL standards.
- Mark-up Fragment Definitions: Markup fragment definitions need to be defined for each particular markup language; the

document type definitions and rules for markup fragments need to be derived from the DTDs of the individual markup languages.

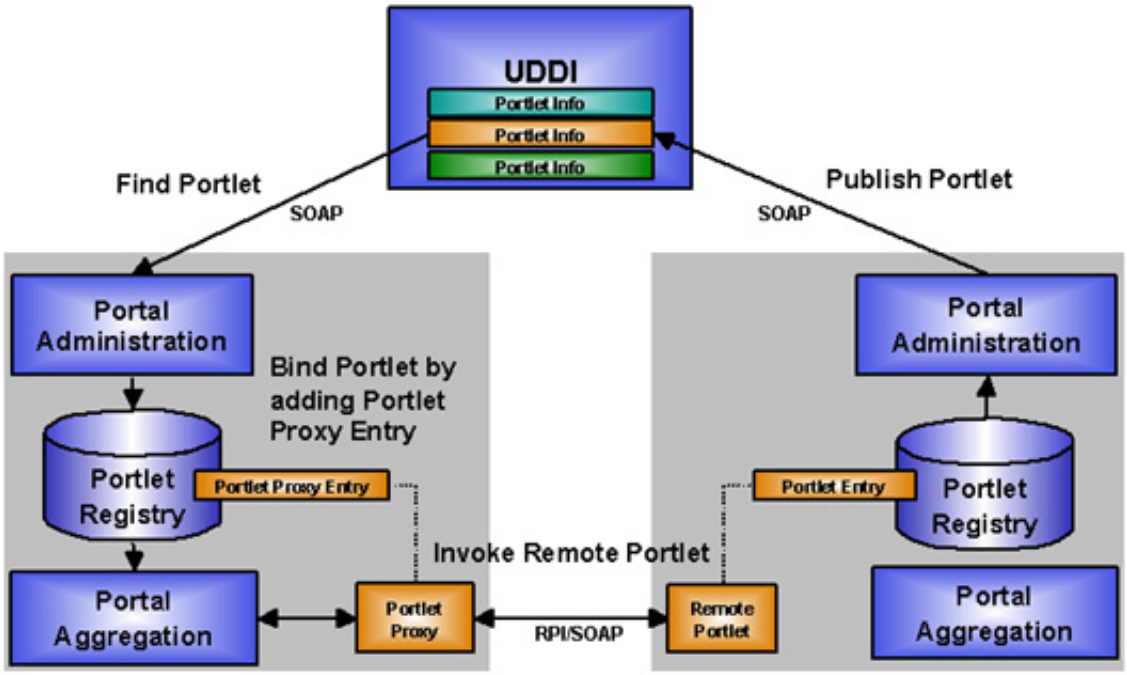
The Web Services for Remote Portals Specification will focus on the second aspect and define interfaces and contracts between portals or other applications and WSRP services.

2.2. How portals can use WSRP Services

In order to make a dynamic integration of WSRP services into portals as easy as possible, a find and bind function needs to be integrated. A UDDI registry acts as the central registry for publishing and finding WSRP services. The UDDI registry can be either a private UDDI that is limited to a corporate network or the public UDDI directory. Client applications that wish to provide or consume WSRP services can perform the following actions:

- Publishing a Portlet as a WSRP service: An administrator uses a publishing function to publish a portlet to the UDDI registry as a WSRP service. For example, the admin function of a portal may read the portlet registry of the portal and displays all available portlets so that the portlet administrator can choose the portlet to be published.
- Find&Bind a Remote Portlet Portlet Web Service: The find&bind administration function lets the administrator search a UDDI registry for WSRP services. For a selected service, the administration function can automatically generate a portlet proxy in the portlet registry that is bound to this service.
- Selecting a portlet proxy representing a WSRP service: After an administator has bound a portlet proxy to a WSRP service, users may be allowed to put the portlet proxy on one of their personal pages.

Figure 7: Example of portals sharing portlets



3. WSRP and UDDI

In order to make WSRP services discoverable for clients, they can be published to a UDDI directory. This section describes how Web Services for Remote Portals are published, how they are described in UDDI, and how they can be found and bound to.

3.1. Publishing Web Services for Remote Portals

In order to make WSRP services known, the provider must publish them to a UDDI directory. To allow for any WSRP Service to plug into any compliant client application, all WSRP services implement the same WSDL interface and behavioral contracts. Furthermore, they share a common set of attributes that can be analyzed by portals to find out what capabilities the service provides and how it has to be used by client applications.

The following attributes are used to describe Web Services for Remote Portals:

- **T-model ID** - The WSRP services T-Model ID to indicate that this is a WSRP service
- **Service Name** - The name of the WSRP Service, e.g. â€œStock Quote Serviceâ€•
- **Supported locales** - The list of locales supported by the service
- **Service Titles** - Titles of the service in all the locales that the service supports
- **Service Descriptions** - Descriptions of the service for all supported locales
- **Supported Modes** - The modes that are supported by the service, e.g. view, edot, config, help
- **Supported Markups** - The different markup languages supported by the service, e.g. HTML, XHTML, WML, VoiceXML, cHTML, â€¦
- **Cachability** - Information on how caching may be applied, including expiry times and indication on whether content is personal or shared.
- **Instance awareness** - Indicates whether the service is aware of portlet instances, i.e. has instance specific behavior
- **Keywords** - Key words describing the service which can be used for search
- **Supported View States** - Minimized, Normal, Maximized, â€¦
- **Allowed modifications** - Transcoding, Translation, Adaptation, â€¦

3.2. Finding WSRP Services

Web Services for Remote Portals can be found in UDDI directories by querying the directory for businesses providing services implementing the WSRP tModel and listing the WSRP services provided by particular businesses. As described in Section 3.1, WSRP Services have to be published with a defined set of attributes describing their capabilities and properties.

3.3. Binding WSRP Services

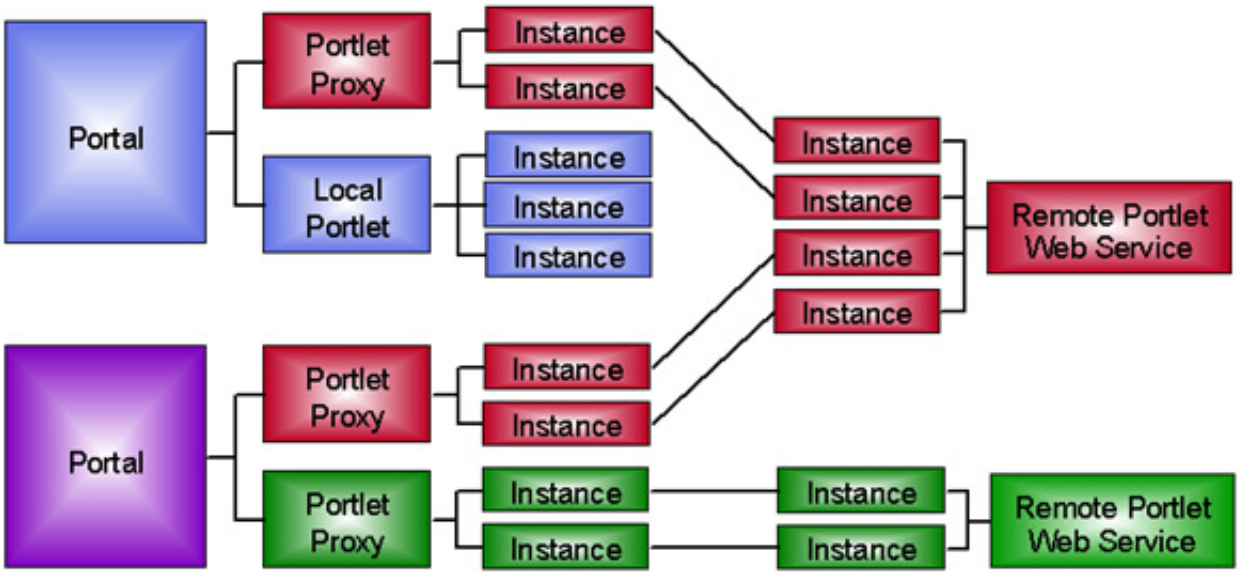
When client applications bind WSRP services, they will typically store the subset of attributes listed in Section 3.1 for administrative purposes and in order to keep information on how to invoke the service. For example, a portal may store the service titles and descriptions of the WSRP services to which it is bound and present this information to end users when they select what they want to see on their personal portal pages.

4. WSRP Life Cycle

Whenever client application binds to a WSRP service, it typically creates local proxy for the WSRP Service. The local proxy as well as the service can be seen as a type. The client application can create portlet instances within any WSRP Service and consequently create an associated, local proxy instance. In the example of a portal, this typically happens when a user selects a locally portlet proxy in the portal customizer and adds it to one of his pages. The portlet proxy instance and the WSRP Service instance must be associated persistently until both instances and their association are destroyed. A life cycle management interface is thus required to create and destroy instances of remote portlets.

Figure 8 shows an example of possible associations of portlet proxy instances and WSRP service instances between two portals and two WSRP services. Note that all boxes represent instances in the form of data entities, not programming language object instances. Whether or not object instances are used to represent the entities at runtime is up to the implementation, which e.g. may create objects per entity or apply the flyweight pattern to reuse objects.

Figure 8: Association of Portlet Proxy instances and WSRP Service instances



Note:

The figure above doesnâ€™t depict the relation between users and portlet proxy instances because this is irrelevant for remote portlet invocation. A portlet instance may be referenced from a single personal page, from multiple personal pages, or from a group page shared across many users. In other words, the same user may have multiple instances of the same WSRP Service and a single portlet instance may be shared amongst many users.

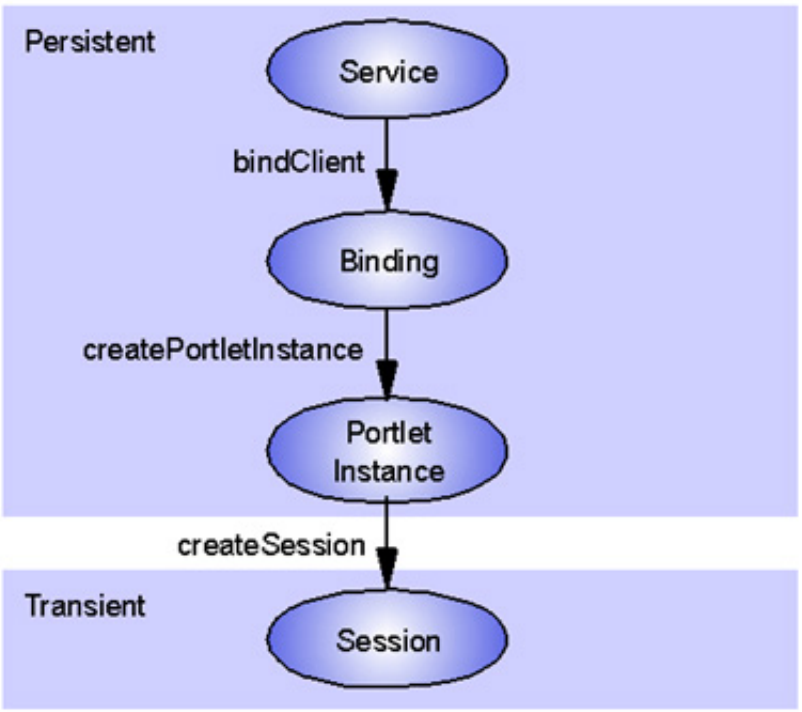
When handling a request, the portal passes the information about the user who triggered the request to the WSRP Service along with the portlet instance handle. While several sessions/users may share the same portlet instance, it may still look and behave differently based on user information.

4.1. Life cycle management operations

Initially, client applications must create a binding to the WSRP Service by calling the **bindClient** operation. This needs to be done only once by each client application that wants to use the service and creates a persistent relationship between the client and the service. The **bindClien**t operation returns a client handle that the client must store and use in all subsequent calls to the service. Once a binding exists, client applications can create WSRP service instances in the scope of the binding by calling the **createPortletInstance** operation, e.g. as a part of the process to instantiate portlet proxies.

Within existing bindings, clients may send events or request markup from the WSRP service for a given portlet instance. In the course of this interaction, sessions may be created by the service if it needs to maintain state across requests. If a WSRP service returns a session handle in a response, clients must pass that handle in all subsequent requests to the same portlet instance. It is not necessary for the client to store the session handle persistently. The client may force creation of a session by calling the **createSession** operation and may terminate a session by calling the destroyInstance operation. The WSRP service may implement a timeout mechanism for sessions. Both the **destroyInstance** operation and the optional timeout mechanism invalidate the session handle. Using an invalidated handle in subsequent operation invocations leads to a SOAP exception.

Figure 9: States and Life Cycle Methods



When a WSRP service instance is no longer needed, e.g. after destroying a portlet proxy instance referencing it, client applications must also destroy the associated remote portlet instance using the **destroyInstance** operation. WSRP services may implement a means to remove remote portlet instances without request from the client application that owns the associated portlet proxy.

When a client doesnâ€™t need the WSRP Service anymore, it must unbind from the service by calling the **destroyInstance** operation on the client handle, so that the WSRP services can release any resources allocated to and discard all data associated with that client.

4.1.1. Binding a client

New client applications perform a bind operation to establish a persistent relationship with the WSRP Service by calling the **bindClient** operation. It returns a client handle that has to be used for creating portlet instances in subsequent **createPortletInstance** calls.

```
bindClient

In:     -

Out:    Client Handle
```

4.1.2. Creation of portlet instances

Client applications can create persistent instances of WSRP services through the **createPortletInstance operation**.

The createPortletInstance operation takes as input the identifier of the remote portlet of which an instance shall be created. WSRP services which only expose one single remote portlet may safely ignore this parameter as in that case the access to the web service already uniquely identifies the remote portlet in question. The web service provider must persistently maintain the created remote portlet instance until the client makes a call to the corresponding destruction operation or unbinds. If a client unbinds, the service must discard all portlet instances associated with that client handle.

```
createPortletInstance

In:     Client handle
Remote Portlet ID

Out:    Portlet Instance Handle
```

As the return value after successful creation of a portlet instance, the create operation passes back a portlet instance handle that can be used to refer to the new portlet instance. If no new portlet instance could be created, an appropriate error message is returned. The client application must use the portlet instance handle in any subsequent calls to the WSRP service intended for that portlet instance. The portlet instance handle remains valid until a call to the destruction operation.

4.1.3. Destruction of Portet Instances, Sessions or Bindings

When an instance of a WSRP Service or a binding is not needed anymore by the client, the associated remote portlet instance or the binding respectively must be destroyed. The WSXL base component interface exposes a **destroyInstance** operation. For WSRP services, this operation can take as an input the portlet instance handle for the portlet instance to be destroyed or a client handle for the client binding to be destroyed.

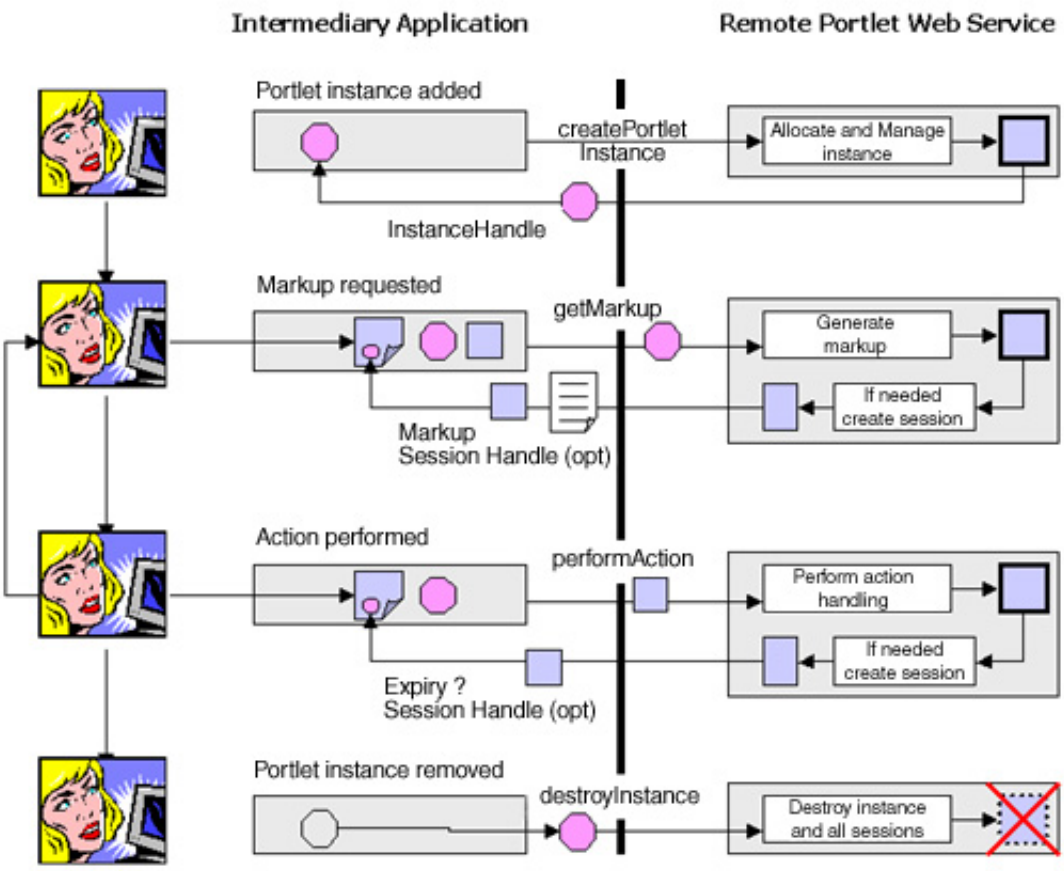
```
destroyInstance

In:     Portlet Instance Handle or Client Handle

Out:    -
```

5. WSRP Usage

Applications use WSRP services by creating remote portlet instances, sending events or requesting markup and finally destroying remote portlet instances when they are not needed anymore. Figure 10 shows an example of interaction between an intermediary application on the left (in this case a portal where WSRP services are embedded as portlets), and a WSRP service on the right.

When a user puts a portlet on one of the portal pages, the portal requests creation of a corresponding portlet instance at the WSRP serviceâ€™s side by calling the **createPortletInstance** operation, obtaining a portlet instance handle that it uses in subsequent requests. It can then obtain markup to embed from the WSRP service by calling the **getPortletMarkup** operation and display that markup within a portlet. If the obtained markup contains links or forms with associated actions, and the user clicks into the portlet, on one of these links or forms, the portlet triggers the corresponding action in the WSRP service by calling the **performAction** operation. When the portal doesnâ€™t need the portlet instance anymore because the portlet is removed from a userâ€™s page, it requests to discard the portlet instance on the WSRP Serviceâ€™s side by calling the **destroyInstance** operation.

Figure 10: Interaction between a portal and a WSRP Service



If the obtained markup contains links or forms with associated actions, and the user clicks on these links or forms, the intermediary application triggers the corresponding action in the WSRP Service by calling the **performAction** operation. When the intermediary application doesnâ€™t need a portlet instance anymore, it can request to discard the portlet instance by calling the **destroyInstance** operation. In the remainder of this section, we introduce the relevant data types and the operations to process events and to get markup.

5.1. Data Types

In the following sections, we will make use of the following complex type.

5.1.1. Request Data

The request data provides information that corresponds to the original request send from the device to the client application:

- Parameters from the original request sent by the client device.
- Session Identifier for the session between the client device and the service (null for the first event or request for markup).
- Mime headers indicating the mime types that can be accepted by the client device.
- Information about the userâ€™s client device.
- Information indicating whether communication between client and service via the portal is secure
- User profile of the client application user (optional, the intermediary application may or may not transmit user profile information depending on privacy settings)
- Locale preferred by the client application user (optional, if not specified the service should use a default locale)
- Locale preferred by the client portal userâ€™s device [fallback mechanism]
- Content type requested by the client application

- Character set requested by the client application
- Current portlet mode of the portlet proxy on the client application (e.g. VIEW, EDIT,CONFIG, HELP)
- Previous portlet mode of the portlet proxy on the client application
- Window state of the portlet proxy on the client application (e.g. NORMAL, MAXIMIZED)

Simple WSRP services may choose to not use all of the rich information that is provided to them; more sophisticated WSRP services may use this information to generate output that is tailored for the userâ€™s device type, translated to the userâ€™s preferred language and personalized based on the user profile.

5.2. Action Processing

WSRP service instances can react on actions that are triggered by users manipulating their UI representation. Users trigger action events by clicking on a link in markup generated by the WSRP service that is prefixed with the **ActionEventURIPrefix** and ends with an **Action Identifier**. Clicking the link in the browser results in a request being sent to the client application bound to the WSRP Service. The client application must convert the received request into an **ActionEvent** and pass it to the WSRP service by invoking the **invokePortletAction** operation. As parameters, this operation expects the destination handle, an action identifier, and request data. The destination handle may be either the portlet instance handle or a session handle retrieved by a call to **getPortletMarkup** or **invokePortletAction**.

```
invokePortletAction

In:     Destination Handle
        Action Identifier
        Request Data

Out:    Session Handle
        Expiry Indicator
```

The return values are the optional session handle for the session between the client device and the service and an expiry indicator. If present, the session identifier must be used in the request data of each subsequent **invokePortletAction** or **getPortletMarku**p call that shall be made within the same session. The expiry indicator must be set to true when the event results in expiry of the view that may be cached on the client application.

5.3. Getting Markup

The essential functionality of a WSRP service is to generate markup based on input parameters provided by client applications and its current state for aggregation within portals. To obtain markup, the client application must call the **getPortletMarkup** operation, providing the destination handle, client context information, client portlet state and request data as parameters:

```
getPortletMarkup

In:     Handle
                Request Data

Out:    Session Handle
        Markup
```

The **getPortletMarkup** operation returns the markup generated by the WSRP service based on the request and remote portlet state and the session handle for the session between the client device and the service. The markup may contain URIs that start with one of the Portal URI Prefixes, e.g. the **ActionEventURIPrefi**x and end with an **Action Identifier**. The session handle must be used in the request data of each subsequent **invokePortletAction** or **getPortletMarkup** call that shall be made within the same session. To allow the transfer of any type of markup the markup has to be transferred in Base64 encoding.

6. Markup Restrictions

As the markup provided by WSRP services must be aggregatable in the larger context of pages of a client application, it must adhere to certain restrictions and rules. The markup generated by a WSRP service may be adapted to the style of the calling client application in several ways:

- **Style sheets**: For some markups, e.g. HTML, style attributes like font face, size, colors, etc may be defined by a style sheet that is active in the client application. Remote portlets should only explicitly specify their own style attributes in exceptional cases. They must use to-be-defined standard style sheet attribute names that providers must use to allow clients to override the look and feel.
- **URLs**: URLs generated by the WSRP services must be adapted to the environment of the calling client application, for example in a portal server portlet proxies must intercept any interaction with the portlet. This implies the use of URL-rewriting which will be further detailed in Section 8.2.
- **Adaptation**: The markup of the remote web service may have adaption points that can be used to modify or augment WSRP servicesâ€™ output in a specific way. To support this, WSRP Service can make use of the â€œAdaption Description Languageâ€• as defined in the WSXL standard.

7. Security

Web Services for Remote Portals do not define special security mechanisms. The same security mechanisms as for other kinds of web services may be applied.

7.1. Authentication

Authentication lies in the responsibility of the providers of WSRP services. They may configure their servers to support SSL/TLS server authentication and if desired also for client authentication. In the latter case, only clients supporting SSL/TLS client authentication with the required cryptographic keys would be able to connect.

7.2. Authorization

Authorization lies in the responsibility of WSRP services. As Web Services for Remote Portals have access to client application identifiers and user profile information, they may implement authorization mechanisms based on portal identity, user identity, or both.

7.3. Confidentiality

For secure communication, Web Services for Remote Portals may allow for invocation via SOAP over HTTPS.

8. Contract between Portals and Web Services for Remote Portals

While implementing the interfaces defined in the previous sections is required for Web Services for Remote Portals to be pluggable into portals, it is not sufficient. This section defines the contracts that WSRP services must adhere to in addition in order to allow for aggregation within intermediary web applications like portals.

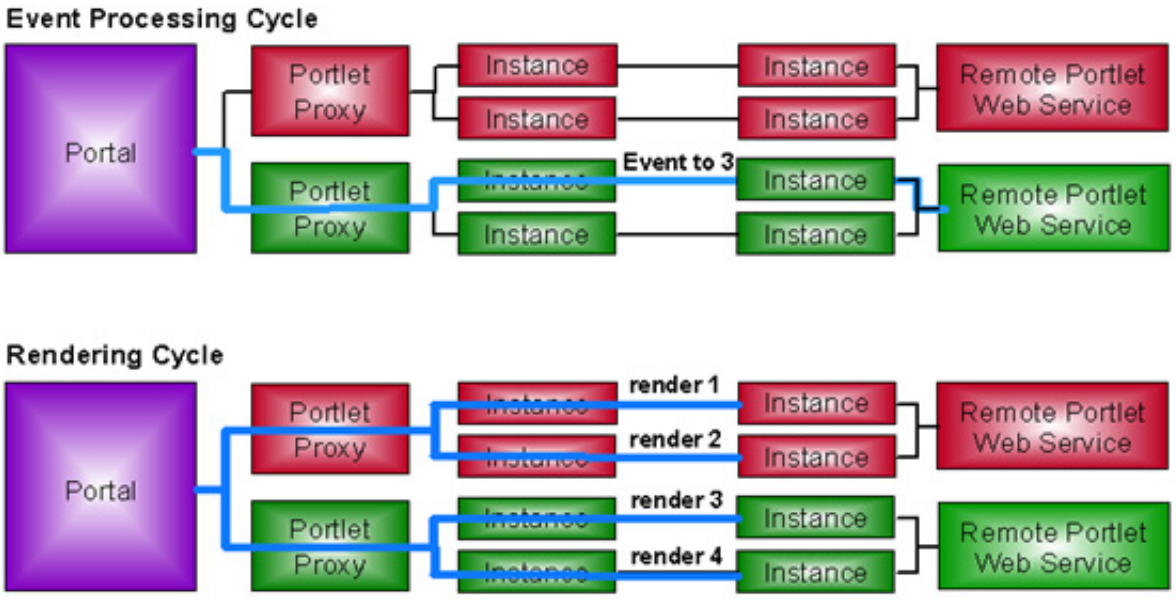
8.1. Two phase processing of events and rendering

To guarantee correct behavior of event handling and rendering of markup by multiple WSRP services bound to a portal, the portal must assure that event handling and rendering take place in two separate cycles.

- In the first cycle, event handling must be performed completely; any events raised as secondary events must be finished before the second cycle starts.
- In the second cycle, the portal requests markup from all components and aggregates it into a page.

Figure 11 shows an example showing why two separate cycles are required:

Figure 11: Example of a possible order of events



Let us assume the user clicks a link in Portlet 3, resulting in the **invokePortletAction** operation of Portlet 3 to be called. This operation in turn sends a message to Portlet 1, changing its state. Because of the layout of portlets in the page however, the portal must invoke the **getPortletMarkup** operation of Portlet 1 before that of Portlet 3. Only by handling events in an independent cycle before rendering, it can be assured that for Portlet 1 the current state is displayed. Ootherwise, the old state of Portlet 1 would be used to render it.

8.2. URL Rewriting

As WSRP services can be interactive, they often need to produce mark-up containing links pointing back to actions within the services themselves. To allow this while still keeping the user in the context of the intermediary web application, e.g. a portal that includes the WSRP service, a URL rewriting scheme is required. The service can only provide a client independent part that needs to be embedded into an appropriate tag that can be resolved by the intermediary web application before actually sending the servicesâ€™s markup to a client device.

URLs embedded in final markup for client devices must have a prefix that points to the client application and identifies the component, e.g. portlet that is bound to the WSRP Service and a second part that identifies the target **Action Identifier** within the WSRP service itself and may optionally contain parameters. If the client device has not enabled cookies, a third part might be required to carry a session ID between the intermediary web application and the client device. If a user clicks on such a link within mark-up created by a WSRP Service and rewritten by the intermediary application, the browser sends the resulting GET or POST request to the intermediary application that must convert the request into the appropriate action event and in turn invoke the **invokePortletAction** operation of the WSRP Service using the **Action Identifier** from the URL.

8.3. Restrictions on Markup Fragments

As the markup produced by many different WSRP service instances usually is aggregated and displayed by portals on common pages, WSRP services must generate markup fragments rather than complete documents. The structure of markup fragments depends on the particular markup languages of supported client devices.

8.3.1. HTML Markup Fragments

Markup fragments for HTML must be aggregatable in pages, e.g. it must be possible to encapsulate them within in <table> or <DIV> tags. This means that they may not contain tags such as <html>, <body>, <head>, etc. and they may not contain frames.

For font and color definitions, HTML markup fragments should only contain explcit definitions if required for example to preserve the corporate identity of the provider of the WSRP Service. In most cases portals including WSRP services will want them to adapt to the portal look&feel.

8.3.2. WML Markup Fragments

Tbd

8.3.3. VoiceXML Markup Fragments

Tbd

8.3.4. CHTML Markup Fragments

Tbd

8.4. Caching Considerations

Client applications using WSRP services will need to apply caching on results when possible to achieve short response times. For example, there may be a WSRP Service that provides the latest news, with an expiry of 5 minutes. For client applications that receive requests for the information they in turn acquire from the service, caching the markup obtained from the service and serving the requets from the cache reduces response times by orders of magnitude.

We can identify the following properties of WSRP services that are relevant to caching on the client application side:

- Expires - This attribute determines the minimal time before the view of a WSRP Service expires
- WSRP services should implement the WSXL property interfaces
- Instance awareness - This attribute determines whether the service is aware or agnostic of portlet instances

The expires-attribute may be used by client applications to determine how long views may be cached. The instance awareness-attribute may be used to determine whether cache entries have to be managed on a per-instance basis or may be shared for all portlet instances of a WSRP Service. Client applications must treat cache entries for WSRP Service instances as expired as soon as an action fires to the instance.

9. Relation of WSRP to WSIA

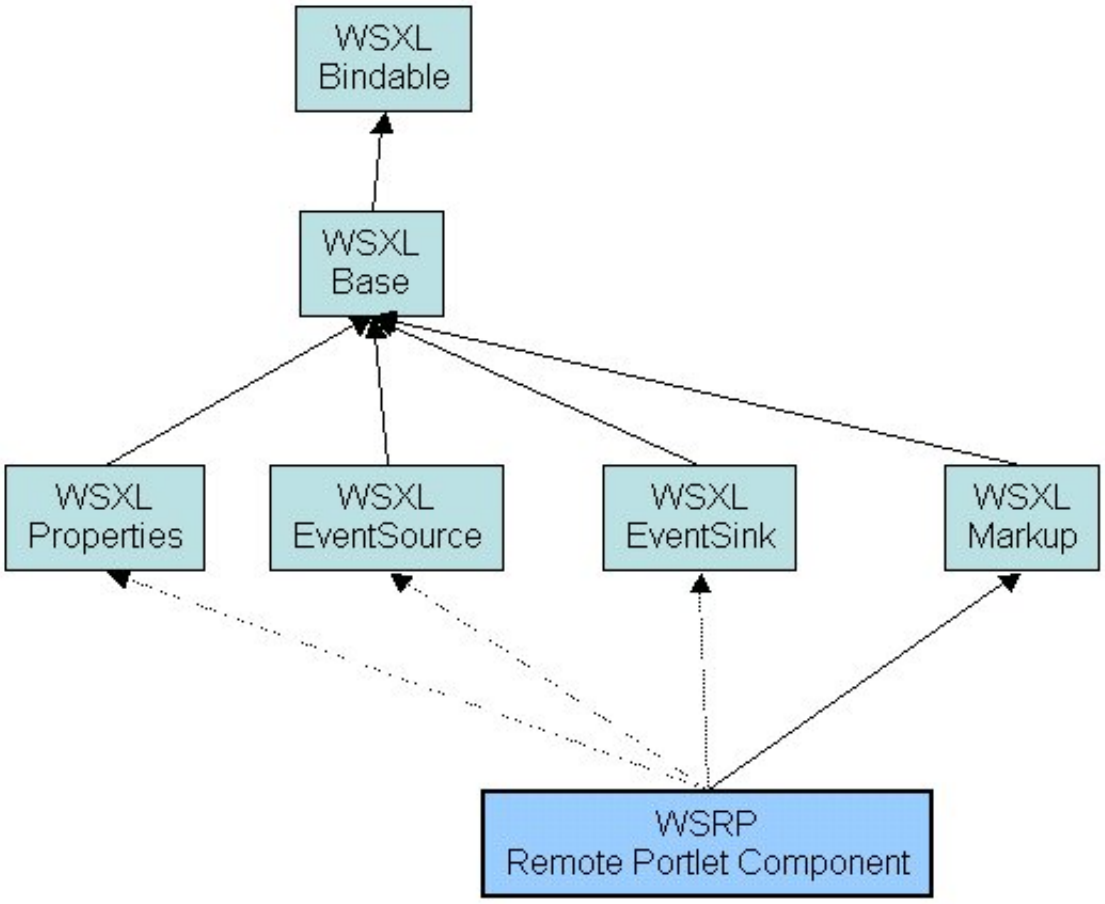
Web Services for Remote Portals are special WSIA components, sharing the basic interfaces and contracts defined in WSIA. This means that WSRP services behave like other WSIA components in terms of life cycle and provide operations for providing markup and for processing user actions. While WSIA as a horizontal standard defines a general programming model for web service-based application components and base interfaces, the WSRP standard can be seen as an vertical application of WSIA, defining a concrete interface and contract for components intended to be aggregated in and driven by client applications like for example portals. By standardizing the complete interface and behavior for compliant services and client applications in addition to what is standardized in WSXL, the WSRP specification enables â€œplug&playâ€• interoperability between any WSRP compliant client application and WSRP service, so that client applications can invoke WSRP services using generic invocation code.

WSIA provides a generic set of basic interfaces for life cycle, presentation and event handling. WSRP implements and extends selected WSIA interfaces and provides a specialized interface to its services. WSRP is designed so that implementing a compliant service is as easy as possible, services are extensible and interfaces can be efficiently accessed.

For interoperability, WSRP services not only provide specialized and efficient access through the WSRP interface for WSRP clients like portals but also provide generic access through the WSIA interfaces for usage by WSIA clients and components. We envision the following relation of WSRP and WSIA:

- WSRP services **must** implement the basic WSIA Bindable, WSIA Base, and WSIA Markup interfaces
- WSRP services **should** implement the WSXL property interfaces
- WSRP services **may** implement the WSXL event interface

Figure 12: Relation between Remote Portlet



10. References

1

SOAP Version 1.2, W3C Working Draft 9 July 2001, Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, Henrik Frystyk Nielsen http://www.w3.org/TR/soap12/

2

Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana http://www.w3.org/TR/wsdl

3

UDDI Specifications http://www.uddi.org/specification.html

4

Web Services Experience Language, Angel Diaz, John Lucassen, Charles F Wiecha
http://www-106.ibm.com/developerworks/library/ws-wsxl/index.html