JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2007**

Vol. 6, No. 8, September-October 2007

A Component Model for Integrating Remote Applications and Services via Web Portals[1]

Rainer Weinreich, University of Linz, Austria
Andreas Wiesauer, University of Linz, Austria
Thomas Ziebermayr, SCCH Hagenberg, Austria

Abstract

Application integration via web portals is the most widely used and least expensive means for integrating enterprise applications and services. Component-based portals enable the composition of web pages from reusable portal components, where each component represents an independent application or service. This integration is often limited to components displayed on the same web page, to local deployed components, and to homogeneous environments. We describe a component model for enhanced integration of portal components in web portals. Our model supports not only the aggregation of components within one web page but also the composition of component navigation into a central navigation area, the communication between local and remote components, and the integration of heterogeneous environments. The approach is based on existing standards and uses XML for describing component navigation and communication capabilities. It is mostly declarative and may also be used for improving integration capabilities of already existing portal components.

1 INTRODUCTION

Enterprise application integration (EAI) aims at unifying the different information systems within an enterprise or between different enterprises (B2B integration). Applications can be integrated on the data level, on the business logic and process level, and at the presentation level. This categorization, which can be found in most books on EAI, corresponds to the architectural layers of a typical information system [Chari & Sheshadri, 2004]: presentation layer, business logic layer, and data layer. The data layer is responsible for storing and accessing data in different data stores. The business logic or application logic layer defines business processes and business rules. The presentation layer implements the user interaction logic and provides the user interface for presenting information to the user.

[1] This article is a significantly extended and revised version of a paper presented at the IEEE International Conference on Services Computing (SCC 2005), Orlando, Florida, July 11-15, 2005 (Weinreich & Ziebermayr, 2005).

Cite this article as follows: Rainer Weinreich, Andreas Wiesauer, Thomas Ziebermayr: "A Component Model for Integrating - Remote Applications and Services via Web Portals", in *Journal of Object Technology*, vol. 6, no. 8, September-October 2007, pp. 67-94 http://www.jot.fm/issues/issue_2007_09/article1

Data level integration views data stores as the primary point of integration [Linthicum, 2003] and involves moving data between different data stores in order to share relevant business data among applications [Linthicum, 1999]. Data stores are typically relational data base systems or mainframe database systems. With data level integration the application or business logic is completely bypassed and the use of the data by other applications is out of control of the original data owner [Gorton & Liu, 2004]. Data level integration is often the most flexible way of integration.

Business level integration aims at integrating application or business logic and at integrating business processes. Integration is achieved by means of middleware, typically message-oriented integration brokers and workflow engines [Alonso et. al., 2004]. Integration brokers implement routing logic and transformation rules, and provide adapters for accessing different applications and data sources. Workflow engines allow the definition of business processes.

Presentation level integration means integrating the user interface of applications. Please note that the term presentation level integration as used in this article is different from its use in many books on EAI (e.g., [Linthicum, 1999]; [Ruh et. al., 2001]), where presentation level integration often refers to integrating legacy applications into the business logic layer by automatically extracting data from (terminal-like) user interfaces (screen scraping). We use the term for integrating applications and services by integrating their user interfaces. A simple and common form of presentation level integration in the context of web applications is the usage of hyperlinks to the web front end of other applications. Linthicum [Linthicum, 2003] states that presentation level integration is currently the primary mechanism for application integration, because it is often the least invasive and thus the least expensive form of application integration. However, presentation level integration requires human interaction and leads to suboptimal solutions in cases where business processes need to be automated. In this case, integration can be achieved more efficient on the business logic level.

Enterprise application integration originally concentrated on integrating already existing and potentially monolithic legacy applications. However, to stay competitive, enterprises need a flexible application infrastructure that also permits changes and the quick deployment of new functionality with minimal integration effort [Chari & Sheshadri, 2004]. This means that new applications need already to be constructed for integration and composition. In order to increase flexibility and reusability, applications are further decomposed into smaller reusable units or, in cases where legacy applications need to be integrated, a component-oriented integration layer is introduced. To protect investments and reduce long-term costs of integration, organizations tend to follow generally accepted standards for component integration and composition.

The movement towards componentization and standardization is most obvious in the area of business logic integration. Application logic is decomposed into (web) services, which can be used on the basis of clearly defined and standardized APIs and protocols. Standards for describing and accessing web services are WSDL [W3C, 2004] and SOAP [W3C, 2003]. Integration of (web) services leads to service-oriented architectures. An

emerging standard for composing services and for describing business processes based on services is BPEL [BEA et. al., 2003], which is currently standardized by the OASIS consortium.

However, componentization and standardization is not limited to business level integration. Component-based web portals enable the integration of the user interfaces of different applications and services as components on the same web page. Companies are investigating presentation level integration using web portals for customizable integrated workplace solutions (enterprise portals) and for providing personalized information services to their customers (consumer portals).

In this article, we present a component model for enhanced presentation level application integration in component-based web portals. Presentation level components in web portals are typically called portlets. We provide enhancements to current standards in mainly two areas: Firstly, we support data exchange between presentation level components (portlets). We support not only data exchange between local portlets and between portlets of the same portal application but also between local and remote portlets and between portlets of different portal applications. Secondly, we support automatic integration of the navigational elements of different portlets into central navigation areas.

The approach has been implemented on the basis of the Portlet and WSRP specifications. This means, it can be used in any JSR-168 compatible portal server and works with any WSRP producer. In terms of portlet communication, it enables the exchange of arbitrarily structured XML-data, even if only part of the data is displayed at the user interface, and allows pre-configuration of data exchange connections (wires) between different portlets. For integration of navigational elements, each portlet may export an XML-based menu description, which may be used as building blocks for central navigation bars.

The remainder of this article is structured as follows: In the following section we give an overview of the basic concepts and standards of presentation-level integration with component-based portals. Then we define requirements for interportlet communication and navigation integration in our application domain and describe the basic concepts of our approach. A detailed description of the underlying component model and of the architecture and implementation of our approach is the main part of this article. In the final sections we reflect on deficiencies of existing standards, outline status and further work, include a detailed discussion of related approaches, and provide an outlook on upcoming standards and their implications for our approach. The article concludes with a summary of the main benefits of our approach.

2 BASIC CONCEPT AND STANDARDS

A well-known design principle for web based information system is the separation of application or business logic and presentation logic. This separation of concerns has many advantages like the possibility of reuse of the application logic for different applications, the possibility of offering different user interfaces for different end user

devices (multichanneling), and the possibility of assigning the different application parts to different, specialized development teams.

Web services are typically a means for application integration at the application or business logic layer. Application logic is partitioned into application parts or services and access is provided through clearly defined service interfaces specified in WSDL. Application integration at this level is most useful for automating and integrating business processes (e.g., using BPEL) and provides limited support for integrated workplace solutions and consumer portals. Integrating user-centric applications at the business-logic-level typically requires the development of a new presentation layer for the specific integration scenario, even if user interfaces for the individual applications and services exist.

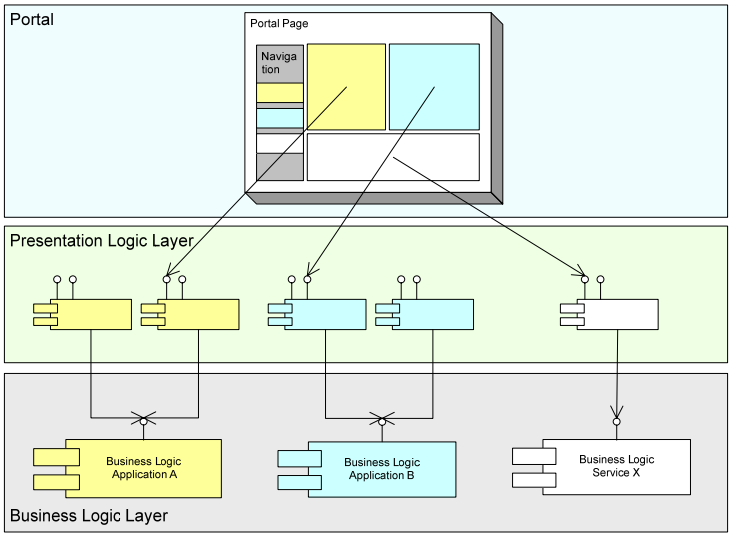
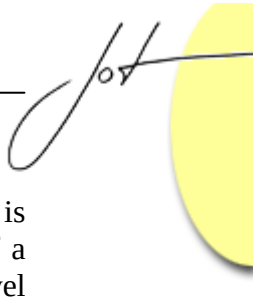


Figure 1: Presentation-level integration in web portals

Component-based web portals offer an approach for reusing applications and services at the presentation layer which exceeds the well known approach of linking web applications by means of hyperlinks. The main idea is to provide reusable components, called portlets, at the presentation level, each implementing the user interface for a logical application part or service. Portlets from different applications can be integrated within the same web page, as shown in Figure 1. For example, one portlet may display a list of customers; another portlet on the same page may display a detailed view of a specific customer; and a third portlet may show an application for calculating mortgage rates.

The composition of web pages from portlets is performed by portal administrators, who configure web portal pages by selecting the portlets to be displayed on the page using portal-server-specific configuration tools.

Composing applications from reusable presentation level components is important for personalization of portals and for creating integrated and adaptable workplace solutions. Presentation level integration may also be used as an alternative for business



level integration, especially if partitioning an application at the business logic level is very expensive and user interaction can be tolerated. In this case, the functionality of a potentially monolithic application can be reused by defining presentation-level components for certain functional aspects of the application. For example, a CRM application might offer a reusable customer search component at the presentation level.

Standards for integrating services at the presentation level are the Java Portlet Specification (JSR 168) [JCP, 2003] and the Web Services for Remote Portlets (WSRP) Specification [OASIS, 2003].

JSR-168 based web portals provide a component model for Java-based portlets. JSR-168 is restricted to local portlets. This means that portal pages can only be composed from portlets that have been deployed at the same portal server. This is a problem if portlets that are hosted by different organizational units have to be integrated into one application.

WSRP provides additional features that are not supported by JSR-168. WSRP is a platform-independent specification that allows the integration of remote, and also non-Java, portlets into portal servers and other applications acting as WSRP consumers. In addition, WSRP can be used for integrating Java-portlets in non-Java portal servers. In this case the Java portal server has to act as WSRP producer.

Currently, standards like JSR-168 and WSRP support no further integration of portlets from different applications than being displayed on the same page, especially if some of these portlets are hosted on a remote server. To stick with the previous example, in order to use the customer data (displayed by one portlet) in the mortgage calculation (displayed by the portlet of another application), a customer consultant has to copy the fields manually and complete the missing information by reentering it at the user interface of the destination portlet. In addition, each portlet has to maintain its own navigation and menu structure. This may lead to inhomogeneous and decentralized navigation structures, which is a significant problem for providing a consistent and intuitive user-interface to a human user.

3 REQUIREMENTS

The approach presented in this article has been motivated by integration requirements of related financial institutions in Austria, who need to integrate applications and services hosted by physically and organizationally separate computing centers with different hard- and software server infrastructure. Main requirements were adherence to standards due to heterogeneity of IT infrastructures and support for integrating *remote applications and services* at the presentation level in component-based web portals.

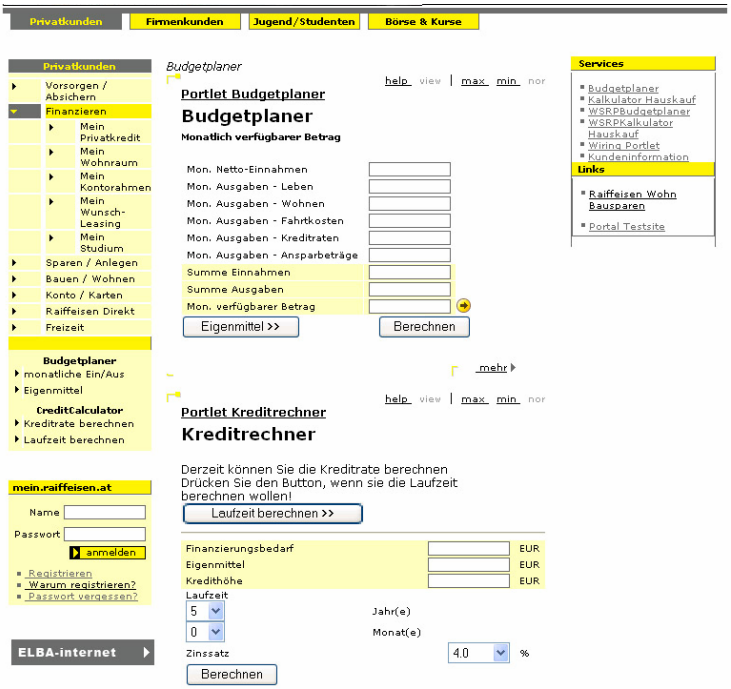


Figure 2: A typical portal page

Figure 2 shows a typical web page with aggregated content areas from different applications and services. Each area is generated by a portlet acting as presentation layer for the application or service. The figure shows portlets supporting bank customers in calculating special savings and mortgage rates. A portlet creates not only the user interface representation but is also responsible for handling user interaction in its area. Portlets may be running on one portal server, but may also be hosted and integrated from remote servers.

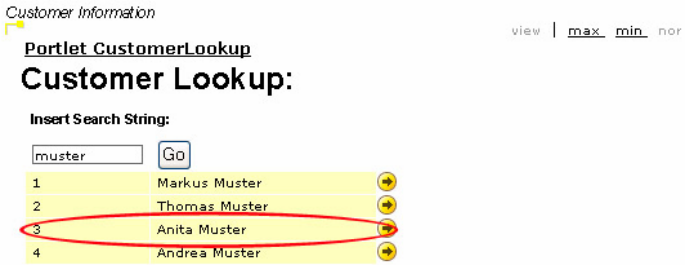
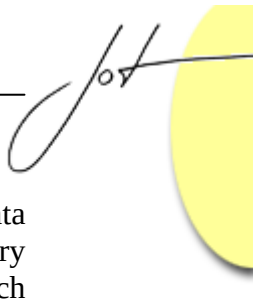


Figure 3: Customer search portlet

In order to support specific business processes, data exchange between portlets is necessary. For example, a Customer Relationship Management (CRM) application may provide a customer lookup portlet (see Figure 3), which can be used by a customer consultant in a financial institution for searching for a specific customer.



Since the customer asks for a credit, the consultant wants to transfer the customer data directly to the credit calculation application, instead of manually entering the necessary customer data. Ideally, he just sends the customer data provided by the customer search portlet to a portlet of the credit calculation application. Note that the portlet in Figure 3 only shows a fraction of the available customer data. The customer name may not suffice for further processing in other applications like the credit calculator. Thus the customer search portlet actually has to transfer much more information, which is transparent to the user (the consultant in this case).

This small example allows us to present the requirements on our approach in more detail. We start with the basic requirements on data exchange between portlets.

a) Intra- and inter-application communication.
   Since portlets of different applications may be part of one portal site and even of one portal page, we need to support data exchange between portlets within one application and between portlets of different applications.

b) Communication with remote portlets.
   Portlets located on remote hosts may be integrated using WSRP. We need to support data exchange between local and remote portlets.

c) Support for heterogeneity.
   The need for integrating remote portlets using WSRP implies that remote portlets may be implemented using different implementation technologies. This means that the data needs to be transmitted in a platform-neutral format.

d) Support for transmitting complex data structures.
   As described in our small example, supporting exchange of individual data fields visible on the user interface is not sufficient. We need to support the exchange of complex data structures transparently to the user.

e) Support for 1:n communication.
   Data from one portlet may be simultaneously sent to a number of destination portlets. A typical example is a portlet displaying a map of a geographical region. Upon selecting a particular location, the location data may be sent simultaneously to several portlets, one describing the location, one displaying weather information, and one showing current events at this location. This is a typical scenario in consumer portals.

f) Support for selecting a particular target.
   Sometimes data has to be sent to a particular target, even if several portlets are able to consume the data. For example, the consumer consultant in the financial example may want to consult a customer on savings and investments instead of credit offers. Thus he needs to send customer data from the CRM portlet to the appropriate target portlet according to the specific demands and business case.

g) Wiring particular portlets.
   Sometimes business processes for a particular workplace are clearly defined. For example, a particular consultant may always need to send consumer data to a

credit calculation portlet. In this case, the connection between particular portlets needs to be defined in advance as part of a workplace configuration.

Naturally Requirements a-c also apply to navigation integration. In addition to these basic (mostly functional) requirements we had to consider requirements regarding mostly non-functional system properties, aimed at protecting investments and long term viability of the approach. This includes issues like reusability, portability, and integration with existing components. For our approach, the following more specific requirements are important:

h) Portability of the developed services.
The communication and navigation integration services have to be deployed to Java-based portal servers. But both services need to be as independent from a particular portal server as possible.

i) Reuse of third-party components.
It should be possible to reuse components from independent third parties.

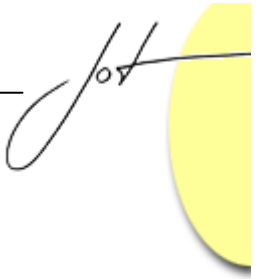
A well known strategy for ensuring these non-functional requirements is the adherence to existing standards like the Java Portlet Specification [JCP, 2003] and the Web Services for Remote Portlets (WSRP) specification [OASIS, 2003]. However, specifications like WSRP typically contain optional parts, which need not be implemented by compliant WSRP producers. This means that it is important to rely on core features of a particular standard, only.

The use of standards allows reuse of standard compliant portlets, potentially from third parties, and enables the integration of these standard portlets with enhanced portlets supporting interportlet communication and navigation integration. However, reusing third-party components is not enough. Ideally, it should be possible to easily enhance these components with data exchange and navigation integration capabilities. Further, enhanced portlets may also need to be integrated in standard portal servers without data exchange and navigation integration services. This leads to the following two final requirements:

j) Adaptation of existing portlets.
It should be possible to augment existing JSR-168 and WSRP compliant portlets with data exchange and navigation integration capabilities, ideally without changes to their implementation.

k) Use of data exchange enabled portlets without the data exchange service.
It should be possible integrate portlets with data exchange and navigation integration capabilities in standard JSR-168 and WSRP compatible portal servers without the data exchange and navigation integration services.

As stated by [Chari & Sheshadri 2004], standards are an important means for reducing the long-term costs of integration. However, the use of standards also has disadvantages, especially if they are immature and incomplete. We outline shortcomings of existing standards with respect to our extensions after presenting our approach in the following sections.



4 APPROACH

The communication service supports sending of structured XML data from a portlet acting as data source to compatible portlets acting as data sink. A portlet acting as data source may associate arbitrary data with information presented at the user interface and may send this data to compatible data sinks on user request. An example for a portlet acting as data source is shown in Figure 4.

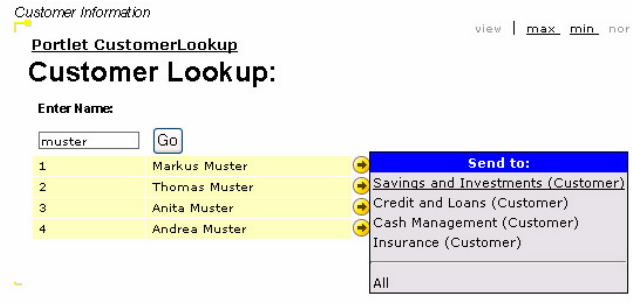


Figure 4: A portlet acting as data source

The figure shows a portlet displaying a list of customer names. Each customer name has associated customer data, which is indicated to the user by an arrow icon. An example for associated customer data is presented in Listing 1.

```
<person gender="male">
        <firstname>Markus</firstname>
        <lastname>Muster</lastname>
        <id>44564545</id>
        <address>
                <street>Musterstrasse 88</street>
                <city>Musterdorf</city>
                <zip>4135</zip>
        </address>
        <dayofbirth>
                <day>3</day>
                <month>5</month>
                <year>1973</year>
        </dayofbirth >
</person>
```

Listing 1: Customer data example

Data of a specific customer can be sent to other portlets, by selecting the arrow icon associated with the name of the customer. After selecting the data to be sent, the data exchange service determines all portlets, which are able to consume the selected data and displays a menu for selecting the desired target as shown in Figure 4. The data can be sent to one or to all compatible data sinks. An administrator is able to predefine connections between portlets and store them persistently. If a connection is already predefined, no menu is displayed and the data is directly sent to the target portlet.

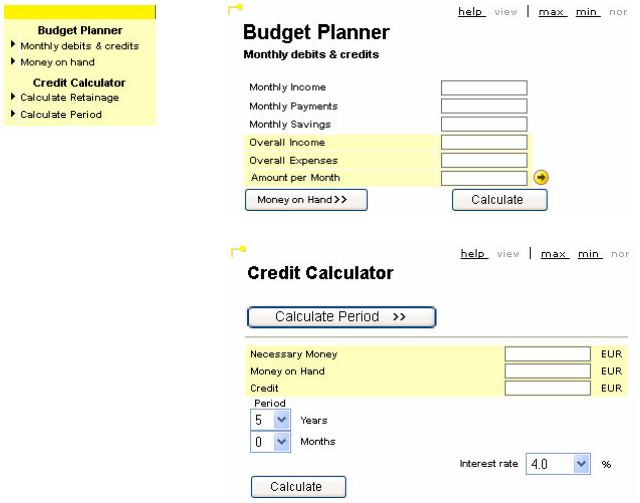


Figure 5: Navigation integration

The approach for navigation integration is shown in Figure 5. The figure shows two independent portlets that are displayed at the same portal page. Each portlet exports a menu structure (in XML) which is combined with the menus exported from other portlets into a central navigation area on the portal page (shown on the left side of Figure 5).

5 COMPONENT MODEL

Data exchange and navigation integration is supported by modeling portlets as presentation level components. Each component offers clearly defined input and output ports for communication and may export a menu for navigation integration. Input ports, output ports, the data to be transmitted, and the exported menu are described in XML to support integration of remote portlets and to enable platform independence as described in the Requirements Section (requirements b and c).

Each communication port has a name and a data type. The port name is used with the portlet name for creating durable connections between portlets. The data type is used for assuring compatible connections and for determining data sinks that are able to consume data offered at a specific output port.

The output ports of a data source and the exported navigation menu are embedded in the HTML-markup of the actual portlet page (fragment) presented at the user interface. This means that the output ports of a data source portlet and the portlet menu may change with each page being displayed. This is not surprising, since the output ports are associated with the data being displayed at the user interface.

```
…
<tr>
   <td>3</td>
   <td>Markus Muster</td>
   <td>
     <!-- <port
          name="personData"
          type="Person"
          schema=http://www.raiffeisen.at/schemata/types.xsd
          portlet="CustomerLookupPortlet">
          <person gender="male">
               person data (see Listing 1)
          </person>
     </port> -->
   </td>
</tr>
…
```

Listing 2: Output port description (data source)

Listing 2 shows a fraction of the HTML page fragment generated by the portlet displayed in Figure 4 including the embedded port description of an output port. The output port in Listing 2 is associated with one of the names displayed by the portlet in Figure 4. The attributes name and portlet are used for identifying a port when defining connections between portlets. The type and schema attribute defines the data type of the data available at a specific output port. The data to be submitted is also part of the port description. The listing shows that the available person data contains much more information than just the person's name.

The input ports of a data sink are equally described in XML and stored in an interface description file, which can be accessed using a URI. Listing 3 shows the interface description file of a sample data sink.

```
<datasink>
   <port>
      <name>Customer</name>
      <description>credit user</description>
      <type>person</type>
      <schema>http://www.raiffeisen.at/schemata/types.xsd</schema>
      <action>setCustomer</action>
   </port>
   <port>
      <name>Amount</name>
      <description>credit amount</description>
      <type>decimal</type>
      <schema>"http://www.w3.org/2001/XMLSchema</schema>
      <action>setAmount</action>
   </port>
</datasink>
```

Listing 3: Input port description (data sink)

In addition to port name and data type the definition of an input port includes an action identifier, which can be used to specify the action upon receiving data at this port. The action identifier is dependent on the dispatching mechanism used by the data sink

implementation. For example, it could denote a method to be called upon receiving data on this port.

Similar to output port descriptions an XML description of the navigation menu of a portlet is embedded in the HTML-markup of the actual portlet page. A sample menu description is shown in Listing 4.

```
<menu id="creditCalculator" title="CreditCalculator" xmlns="http://www.raiffeisen.at/portal/menu">
    <menuitem id="calculator_page1" href="….>" info="Calculates the Ratainage">
        <label>Calculate Retainage </label>
    </menuitem>
    <menuitem id="calculator_page2" href="…" info="Calculates Credit Period">
        <label>Calculate Period</label>
    </menuitem>
</menu>
```

Listing 4: XML-description of portlet menu

As shown in Listing 4, a menu may consist of menu items, each containing a URL link to the target portlet. The id attribute denotes the action to be performed by the portlet. The value of the label element is displayed at the user interface. The info attribute provides an additional help text, which may be useful for a more elaborate description of the menu element in certain situations.

6 ARCHITECTURE AND IMPLEMENTATION

The developed services are based on the Portlet (JSR-168) and WSRP specifications and can therefore be used with any JSR-168 and WSRP compliant portal server. We first describe the architecture of a typical JSR-168 compliant portal server and then illustrate the necessary extensions for the interportlet communication and navigation integration services.

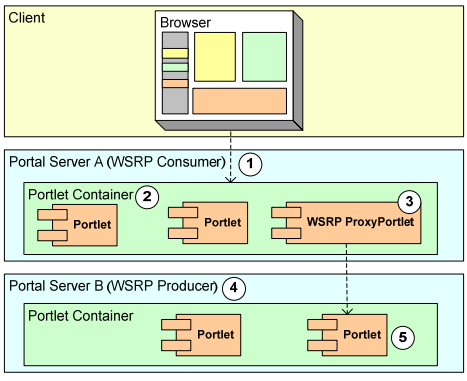


Figure 6: Portal server architecture (component perspective)

The Portlet Specification defines a component model for presentation components (portlets) in Java. A JSR-168 compliant portal server is typically implemented on the basis of a J2EE compliant web or application server. The architecture of a JSR-168 compliant portal server from a component perspective is shown in Figure 6.

The portal server (1) uses presentation components (portlets) for generating portal pages. Each portlet delivers a page fragment. All page fragments are aggregated to form the final portal page. Portlets are loaded in a portlet container (2). The portlet container is responsible for life-cycle management of the loaded portlets and defines the portlet context. Client requests are received by the portal server and are dispatched to a target portlet, depending on the configuration of the portal page.

Remote portlets are accessed via a proxy portlet (3), which acts as WSRP consumer. The proxy portlet forwards requests to the remote portlet using WSRP calls. The host of the remote proxy needs to act as WSRP producer (4) and delegates the requests to the appropriate remote portlet (5).

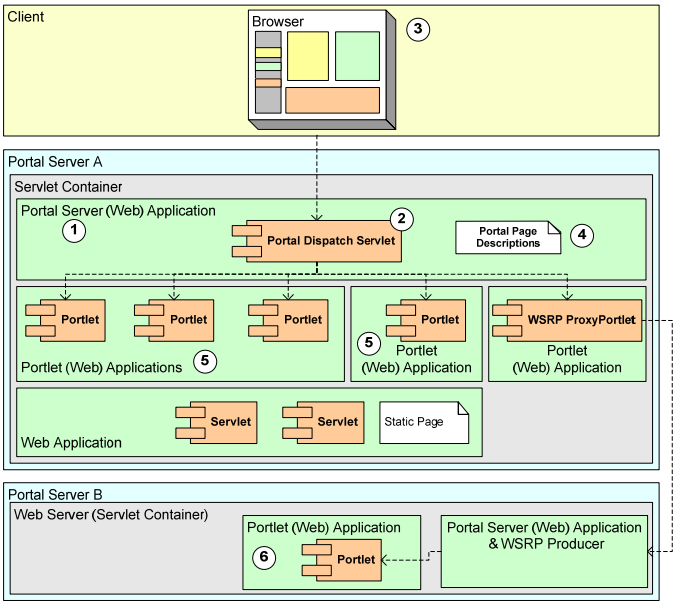


Figure 7: Portal server architecture (application perspective)

Figure 7 shows the portal server architecture from a web application perspective. Since a JSR-168 compliant portal server is typically based on a J2EE compliant web/application server it allows the deployment of web applications in a servlet container. A web application defines a run-time scope for the components (servlets) of the application. Components of an application may share user specific session state and global application state.

The portal server is usually implemented as special web application (1). It defines a central portal dispatch servlet (2), which delivers the requested portal pages to a client browser (3). The structure of a portal page and the portlets included in a page are defined in the portal configuration (4). The portlets on one page may be part of different (5), and even remote (6) applications. Portlets of different applications are not able not share any session or application state. This means that global session or application state cannot be used for portlet communication and navigation integration.

Figure 8 shows the structure of a portal server using the data exchange service. Portlets supporting data exchange are installed like any other portlet and are managed by the portlet container. The output ports of a data source portlet are embedded in the HTML page fragments generated by the portlet (4). As described in the Section "Component Model" an output port has a name, a data type and contains the data that is available at this port. A data sink portlet embeds URL references to the portlet itself and to the input port description file (7) into each generated page fragment.

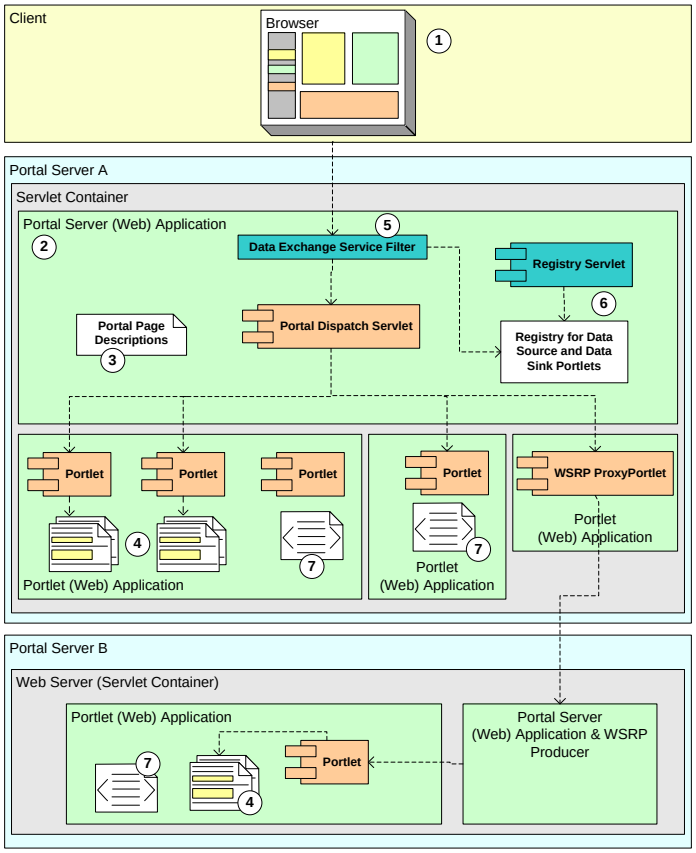


Figure 8:Data service architecture

The working of the data exchange service is best described by a typical interaction scenario. The numbers used in the scenario refer to the components shown in Figure 7.

The scenario starts with a browser (1) requesting a web page with information aggregated from several portlets from the portal server. The request will be processed by the portal server web application (2), which uses a pre-defined page description (3) for determining the portlets it needs to build the requested page. Each portlet returns an HTML-page fragment, which is aggregated with a general page frame and the fragments returned form the other portlets to form the final web page. The page fragments of data source portlets contain output port definitions with embedded XML-data structures (see Section "Component Model").

Before the aggregated page is returned to the web client, it is analyzed and transformed by a filter component (5) which is installed as part of the portal server web application. The filter extracts the output port description and replaces each output port with a hyperlink (illustrated by the arrow in Figure 4) and data exchange application logic. The hyperlink allows the user to send the data available at this output port to compatible data sink portlets. The inserted application logic determines all data sink portlets with compatible input ports and creates a menu for selecting the desired target. After selecting the target, the data is sent to the data sink portlet using an HTTP-POST request (more specifically the request is sent using an actionURL supplied by the data sink). Since data is sent like any other HTML-form data, this mechanism works transparently for remote portlets that are integrated via WSRP.

An important issue for sending data and connecting portlets is determining all data sink portlets that are compatible to a specific output port. Thus data sinks need to register at a central registry (see Figure 8). The registry (6) is located in the portal server web application. The registry also maintains information about data sources, which is used when defining durable connections between portlets. Each portlet is registered at the registry the first time it is displayed on the page. Registration is performed by the filter, which extracts the necessary information from the aggregated portal page.

The late registration of data sink portlets means that data can only be sent to portlets that have already been displayed to the user. This is no problem if data source and data sinks are displayed on the same page. It may lead to problem if data should be sent to an application, whose portlet is on another web page, which has not been opened, yet. The problem is due to the immaturity of current standards. We will comment on this problem in the next section. Currently, we can only solve this problem by defining durable connections between portlets located on different pages.

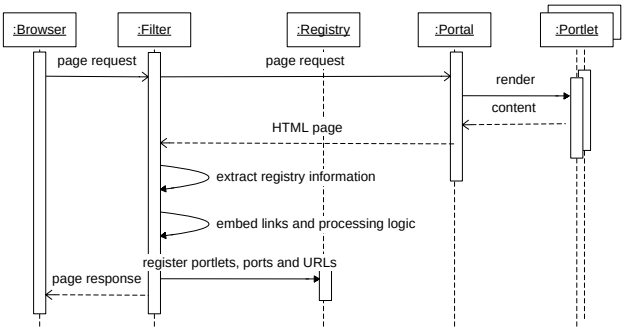


Figure 9: Registering data sources and data sinks

The process for supporting data exchange between portlets in our approach is summarized in Figure 9. The page request is sent from the browser to the portal server application. The portal server application aggregates a web page from the page fragments generated by the portlets and returns the page to the installed filter component. The filter extracts information about portlets and their input and output ports, and registers this information at the central registry. In addition output ports are replaced by hyperlinks and processing logic for determining compatible data sink portlets and for sending data.

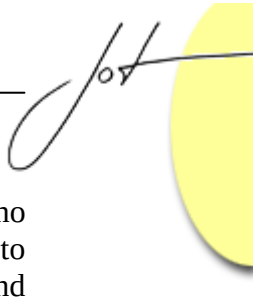
The mechanism for automatically aggregating navigation elements is implemented similar to the data exchange service. Each portlet embeds an XML based menu description in its HTML page fragment including hyperlinks for each navigational element. The page fragments are aggregated by the portal server and processed by a filter component for navigation integration. The navigation integration filter extracts the menu descriptions of all portlets within a portlet page and creates a central navigation area. After the page has been aggregated and processed by both data exchange service filter and navigation integration filter, it is sent to the client.

The described implementation allows a transparent integration of remote portlets, even if they are not implemented on the basis of J2EE and JSR-168. Remote portlets are integrated using a proxy portlet acting as WSRP consumer. They can be data sources as well as data sinks and they may export a menu description. The port description, the address of a remote portlet, and the menu description are embedded in the HTML-page fragment generated by the portlet and transmitted via getMarkup using the WSRP protocol. Sending data to a remote portlet is based on the same mechanism as sending HTML form data: Data is transmitted sending an HTTP-POST request to the portlet ActionURL which results in a call of the WSRP method *performBlockingInteraction*.

7 STATUS AND FUTURE WORK

The presented approach fulfills all requirements defined in the “Requirements” Section. Intra- and interapplication communication is supported as well as communication with remote portlets (requirements a-c). Heterogeneity and structured data are supported through the use of XML for describing interfaces and data (requirements c and d). Multicasts are as well supported as selecting a particular target by means of a menu and the definition of persistent connections between portlets (requirements e–g). We support global as well as user specific connections between portlets. Connections can be defined using a wiring tool.

Augmenting portlets with input and output ports and aggregating portlet menus requires no special APIs and no modification of the portlet code itself. This is because output ports of a data sink and menu descriptions are embedded in the HTML page fragment generated by the portlet. Typically a portlet generates its presentation from a view template based on technologies like Velocity or JSP. Embedding output port definitions and menu descriptions in HTML requires only a change of the view template. This allows augmenting third party portlets with data export capability (requirement j). Input ports are defined in a separate interface description file, which has to be supplied for data sink portlets. Of course, processing the data received by a data sink portlet requires implementation of the processing logic. Finally, since all data embedded in HTML is commented, portlets supporting data exchange and navigation integration can be used in portal servers with no data exchange and navigation integration services installed (requirement k).



Installation of the data service in a J2EE/JSR-168 compliant portal server requires no proprietary adaptations and code changes (requirement h). Only four components need to be added to the portal server web application: two servlet filters for analyzing and modifying portal pages, a servlet managing the registry and a servlet for transmitting data from a data source to one or multiple data sinks. None of these components is dependent on portal server specific code.

We are currently analyzing approaches for adapting remote portlets to different portal contexts. The adaptation of a portlets appearance is supported by existing standards. However, we would need an adaptation of a portlet's functionality depending on the actual integration context. This requires transmitting context and user profile information to remote portlets. We are also investigating the integration of portlets into rich clients acting as WSRP consumers based on our component model.

8 OPEN ISSUES

The main drawback of the current implementation is that communication between portlets on different web pages is only supported, if the page containing the target portlet has already been opened by the user. The problem is less obvious if a persistent connection between two portlets has been defined. In this case we cache the data sent and deliver it when the target portlet is activated. This restriction is due to missing elements in current standards, especially concerning portlet activation and addressing.

Portlets cannot be activated before they are displayed. While this feature is supported by some portal servers, it is not standardized and thus proprietary. This feature might allow us to register data sink portlets in advance.

Another problem is that the scheme for addressing portlets directly via a URL is not standardized. Currently the Portlet Specification only defines how a portlet may include a URL in its own HTML markup. The form of a portlet URL is left to the portal server. In addition, portal servers may encode state information in the URL. This means that the structure of a portlet URL is proprietary and may change each time a portlet is displayed. This makes it necessary, to update the current URL of a data sink portlet in the registry, each time the portlet is displayed. Also, a portlet URL is typically not available until the portlet is displayed. A standardized scheme for portlet URLs (e.g., portlet name relative to page URL) would enable to define a communication mechanism without the restrictions described above.

9 RELATED WORK

Table 1 gives an overview of important criteria for interportlet communication along several dimensions. The scope dimension in Table 1 determines the degree to which interportlet communication is possible, ranging from support of local homogeneous portlets of one and the same application to remote heterogeneous portlets of different

applications. Connections between portlets can be pre-configured or established on demand by means of data-type compatibility. In addition, it is interesting whether an approach supports only global connections or also user-specific connections. Other distinguishing features are data structure and format. If only simple data types are supported expressiveness and power are limited and the use of proprietary formats limits the integration capabilites of an approach. In terms of usage we distinguish between a declarative approach by means of a component model and a programmatic (API-based) approach. Interportlet communication based on the latter is harder to implement and typically reduces platform-independence. Finally we look at openness and support for heterogeneous environments since these issues are particularily important for application integration.

Dimension	Description
Scope	Communication between – local portlets of one and the same application – local portlets of different applications – heterogeneous portlets – remote portlets
Connections	– dynamic target selection – (pre-)defined portlet connections – global and user specific connections
Data	– elementary or structured data types – platform dependent or platform independent format
Usage	– programmatic (API-based) – declarative – component model
Openness & Portability	– works with third-party components – can be used in heterogeneous environments – components can be (re)used in other portal servers – services can be used in different portal servers

Table 1: Important criteria for interportlet communication

In the following we describe a number of approaches and try to relate them to the criteria listed in Table 1 and to our approach. We should note that we were not able to determine every aspect of the following approaches since most vendors are vague concerning the implementation of their products.

The eXo platform [eXo, 2005] provides an extension to JSR-168 for interportlet communication. The approach extends the portlet.xml file for defining event connections between portlets and the portlet API and life cycle for defining and calling Java-based event listeners. Consequently the approach works only for local eXo portlets. Communication with remote portlets via WSRP is not supported.

The Oracle Application Server Portal [Oracle, 2005] supports several portlet technologies. The most important in the context of our discussion are OmniPortlet and

standard Java portlets [Oracle, 2004]. The OmniPortlet technology is proprietary and supports portlet communication by means of events that can be mapped to portlet parameters with configuration tools. It is not clear whether this approach works for proprietary remote portlets as well. Standard portlets are portlets based on JSR-168 and WSRP [Oracle, 2004]. For JSR-168 portlets, Oracle offers an AJAX-based interportlet communication approach. This approach is based on JavaScript functions that can inject markup in the DOM of portlets. These JavaScript functions can be called from within portlets and used to transfer data to other portlets. The benefit of this approach is that portlets can communicate without the need of a page refresh. It seems that this approach only works with portlets on the same page, although the documentation does not make a definite statement on this [Oracle, 2006].

The BEA WebLogic Portal [BEA, 2005] supports event-based communication between portlets. Events are typically predefined, but user defined custom events are supported as well. Contrary to our approach BEA extends the WSRP protocol to support communication with remote portlets [Allamaraju, 2005]. This means that remote communication is only supported for homogeneous, BEA-based WSRP producers and consumers. In addition, remote communication between JSR-168 compliant portlets requires a complex producer. This means that the producer needs to implement WSRP management extensions, which is an optional feature of the WSRP specification. Interportlet communication is performed using so called backing files and special Java data types, which are BEA-specific and not JSR-168 compliant. Since the BEA approach is event- and not component-oriented, automatic detection of possible data consumers and interactive selection of a particular target like in our approach is not supported. Instead a portlet has to declare interest in events from other portlets using event handlers. This can be compared to defining predefined global connections in our approach. However, a connection in the BEA approach is created by defining an event-handler of a remote portlet as a listener to a portlet in the consumer portal. This is only possible if a global namespace for all distributed portlet providers exists, which seems to be provided by a portal domain in BEA. In our approach a global name space is not needed. Connections are defined on the basis of local portlet names. The name of the local portlet proxy is taken to qualify a remote portlet. The definition of connections is completely within the scope of the local portal server. In addition, user-defined connections are supported as well. Finally, BEA provides a proprietary approach for submitting custom data between a proxy portlet (WSRP consumer) and a remote portlet (WSRP producer). This can be used for transmitting context information.

The Portlet Messaging Library created by [Osmond, 2007] is an approach that supports interportlet communication of portlets which reside either in the same or in different portlet applications. Portlets can define a number of inputs and outputs, which can be of any Java object type. These inputs and outputs are mapped to uniquely identified message boxes: inputs are read from message boxes, outputs are written to them. Mapping is defined decleratively in portlet preferences in order to allow portal users changing interportlet communication behaviour at run-time. Message boxes reside in a messaging center which is stored in the session of the portlet application. For enabling cross-context messaging, a common message store is used, which is shared by

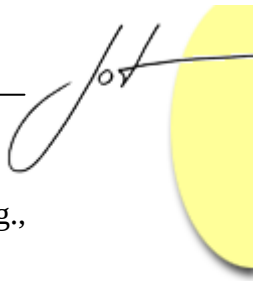
messaging centers of all involved portlet applications. The message store could be realized individually, e.g. as a database, an Enterprise Java Bean or a simple file. This approach supports flexible, user-configured interportlet-communication and is working with any JSR168-compatible portlet container. However, in contrast to our approach, it does not support interportlet communication between portlets residing on different portlet containers (remote portlets via WSRP).

In order to overcome the deficiencies of the portlet specification, JBoss developed a "Portal API" which allows portlets to interact with certain components of the portal, e.g. portal session, portal nodes (represents a tree structure of the portal) or the portal runtime context, which provides access to the current state associated with a user [Heute et al., 2007]. This API also provides an event mechanism that can be used for interportlet communication. The API defines several events, e.g., a PageEvent or a WindowsEvent. PageEvents can occur, for example, if a portal page is rendered or if a user decides to remove a portlet from a page. WindowEvents occur if something is happening to a portlet, e.g., if its window state changes or if a user performs an action on it. In order to react to such events, event listeners need to be defined. Interportlet communication could be implemented by defining a listener that catches events from a portlet A, queries its parameters and sets the parameters on a portlet B, i.e., parameters are copied from portlet A to portlet B. This approach also supports cross-context messaging, but it is proprietary and only works with the JBoss Portal. Additionally, it is not as flexible as our approach and the Portlet Messaging Library approach, because the communicating portlets have to be wired progammatically and communication cannot be changed at run-time.

An approach that is similar to our approach for interportlet communication is cooperative portlets [Roy-Chowdhury, 2003] in the IBM Websphere Portal. Cooperative portlets offers both a programmatic (API-based) and a declarative approach for interportlet communication. A service called property broker supports communication between portlets, which may be part of different applications. The approach supports dynamic target selection [Roy-Chowdhury & Wu, 2004], the definition of persistent connections (wires) and cross-page communication for proprietary portlets, but seems to be restricted to local portlets [Hiranniah & Konduru, 2003]. The functionality is limited for standard (JSR 168) portlets [Roy-Chowdhury & Wu, 2004]. For example, JSR-168 portlets can only communicate via persistent connections. Finally, the property broker is proprietary and can only be used within the IBM Websphere Portal.

To summarize, interportlet communication is an important feature for presentation level integration of applications and services. Communication between local portlets within the same application is supported by all portal servers. For JSR-168 compliant portlets, this form of communication can be realized programmatically using a portlet session with application scope and the portlet context.

Communication between local portlets of different applications is not supported by the JSR-168 specification. Some portal servers support inter-application data exchange by means of proprietary communication services. Often a vendor-specific API is required,



which limits reuse and openness, and data is submitted in a platform-specific format (e.g., Java objects), which limits heterogeneity.

Most portal servers support WSRP, which allows the integration of remote portlets. Since WSRP is platform-agnostic, heterogeneity is supported as well. However, WSRP does not support interportlet communication. Even if a portal server offers a communication service for local portlets and supports WSRP, this does not imply that communication with portlets integrated via WSRP is possible. A WSRP producer may not only be a different product but it may also use a different implementation platform than the WSRP consumer. This rules out the use of vendor-specific communication APIs and of a platform-specific data format.

To overcome the limitations of JSR-168 and WSRP 1.0, many portal vendors provide a proprietary mechanism for interportlet communication (albeit mostly for local communication within a portal server). Where standards like JSR-168 or WSRP are used, the provided functionality is either limited or the standard has been extended. However, this means that the additional functionality is still restricted to homogeneous environments, i.e., environments with portal servers based on one vendor or platform.

This is where our approach distinguishes itself from other approaches for remote portlet integration. We support interportlet communication and navigation integration between heterogeneous portal servers, as long as existing standards like JSR-168 and WSRP are used. To support heterogeneous platform integration, we use XML as platform-independent data format for remote portlet communication. Since an API can also not be used in a heterogeneous platform-independent scenario, we use again XML as the basis for a declarative approach for describing portlets and communication endpoints. The resulting component model allows connecting components, either dynamically at runtime or statically prior to component usage.

The approach is not intrusive and requires no changes and extensions of the WSRP protocol and the JSR-168 API. Extensions to the portlets are added in a descriptive way on the basis of a component model. This means that portlets with communication extensions can also be used in standard compliant portal servers without the communication service installed and that existing third-party portlets can be easily augmented with communication and navigation facilities.

As shown in the previous section our approach has some deficiencies due to standard limitations. When we developed our approach, we were confident that these and other limitations would be solved in future versions of JSR-168 and WSRP. We thought it likely that a future version of WSRP would specify interportlet communication itself. However, after a first look at drafts of the future portlet and WSRP specifications we cannot tell. Since both specifications are important for our approach, we take a look at them in the next section.

10 OUTLOOK - JSR-286 AND WSRP 2.0

Interportlet communication has been a feature requested heavily by portal developers. The absence of it in the current portlet specification (JSR-168) led to many proprietary developments by portal providers, as we showed in the last section. Therefore, interportlet communication is addressed in the second version of the portlet specification, which is currently in early draft review stage [JCP 2007].

JSR-286

The JSR-286 specification defines two new mechanisms that could be used for interportlet communication: event handling and shared render parameters.

The Portlet Specification 2.0 uses an event model in order to "allow portlets to react on actions or state changes not directly related to an interaction of the user with the portlet" [JCP 2007, section PLT 15.2]. It defines an additional request handling phase in which portlets can process events (see Figure 10).

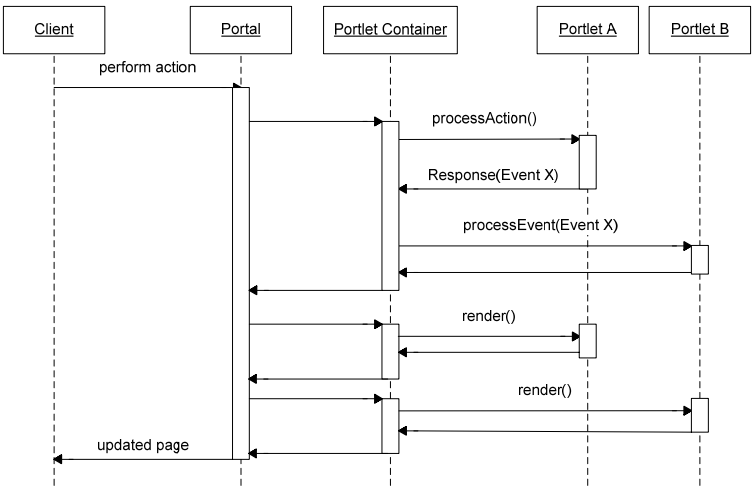


Figure 10: Portlet request handling, following [JCP 2007]

Portlets can declare events that they will fire and events that they will receive. Using these events, you can transmit data between portlets. If you look at the specification, it seems that interportlet communication via events has to be defined programmatically (namely in portlet's processEvent() method). In contrast, the specification states that "the portlet event model is a loosely coupled, brokered, model that allows creating portlets as stand-alone portlets that can be wired together with other portlets at runtime" [JCP 2007, section PLT 15.2]. The specification does not mention how portlets exactly can be wired together at runtime. It states that "it is not in the scope of this specification to define any means how portlets are wired together, nor how a set of portlets relate to each other or to

a portal page" [JCP 2007, section PLT 15]. Provision of run-time wiring portlets is left to developers of portal frameworks or portlet application programmers.

Shared render parameters allow portlets to share their view states. The portlet specification defines shared render parameters as a "shared storage of parameters that can be accessed and modified by different portlets" [JCP 2007, section PLT 11.1.1.4]. As the specification states, you could consider a weather portlet where you can select a zip code of a city for weather information. The portlet could encode the zip code as a shared render parameter making it available for other portlets that also use a zip code as a shared render parameter (e.g. a map portlet that displays the location of the city). Using this method, portal users can bookmark the zip code for several cities in their browsers, if the portal is URL-encoding the shared render parameter [JCP 2007, section PLT 15.1].

WSRP 2.0

The portlet specification only deals with portlets residing in the same portlet container. For the use of interportlet communication involving remote portlets, the new WSRP specification, which is currently in draft stage, basically provides a feature called "consumer mediated coordination". This feature is intended to enable interactions between portlets that are aggregated by a consumer [OASIS 2007]. It provides an event mechanism that allows propagating events between portlets, no matter if they are local or remote. This event mechanism extends the event mechanism defined in the second version of the portlet specification in that it is able to forward events from local portlets to remote portlets and vice versa (see Figure 11) [Thiagarajan 2007].

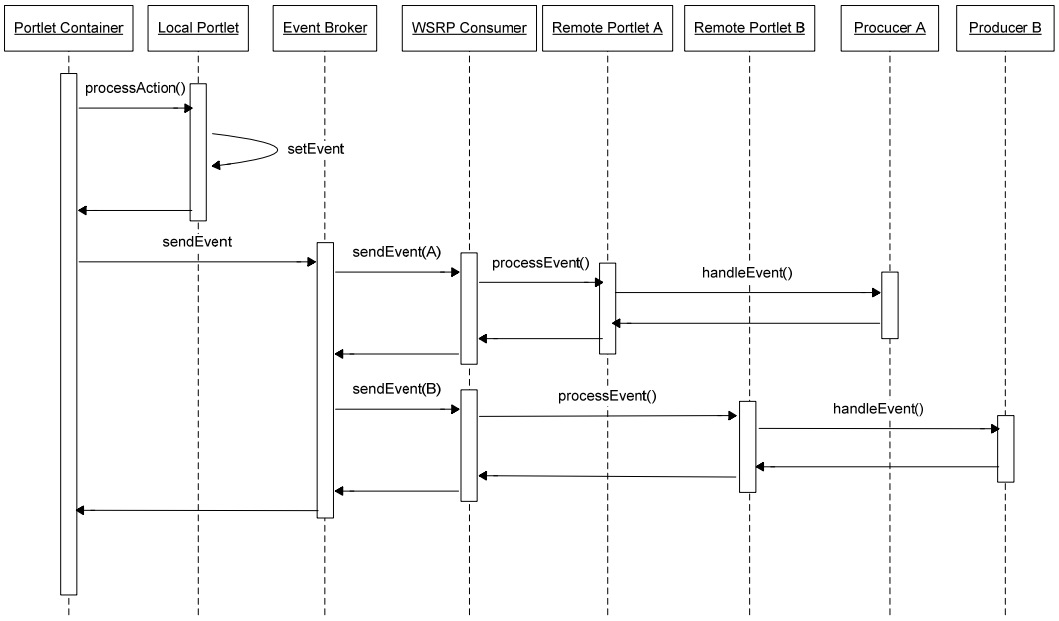


Figure 11: Event handling specified by WSRP 2.0, following [Thiagarajan 2007]

Additionally, it offers a feature called “state distribution”. State distribution is either done by “navigational parameters” or by “session properties”. Navigational parameters are an extension of what the portlet specification 2.0 called “shared render parameters”, in order for being used with remote portlets. Session properties are declared by a producer and can be altered by consumers and therefore be used for interportlet communication [Thiagarajan 2007].

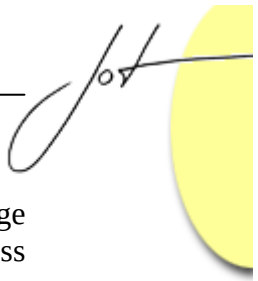
To summarize, both new specifications do not exactly define the way interportlet communication has to be done. They leave many aspects to portal framework developers and portlet application developers. In particular, they provide no component model but useful communication support like event mechanisms for enabling interportlet communication.

We tried to identify what the features of the new specifications mean to our approach. Will it be obsolete in the future or could it benefit from the new features? At a glance, there is no similar approach for interportlet communication included in the new versions of the specifications, so we think that our approach will remain useful. Instead, our approach could benefit from new features. For example, the input and output ports of portlets could be replaced by portlet events and listeners. This would reduce the complexity of the data exchange service filter, as most of its work would be done by the event model components of the portlet container (except embedding the navigation menu of a portlet in the HTML-markup of the portal page). An interesting question that will arise is how the event mechanism will work in conjunction with remote portlets and WSRP, especially if non-Java producers are involved. Because of the XML representation of inputs, outputs and data, this is no problem in our approach. When the specifications are completed and reference implementations are available, we will have a look if our approach could be better implemented using new features and if refactoring makes sense.

We also looked at the new specifications to answer the question whether they enable a solution of the open issues we mentioned in Section 8, namely communication with portlets on pages that have not been already opened by users, activation of portlets before displaying them, and a standardized scheme for portlet URLs. However, we could not find any sign that the new portlet specification will address these issues.

11 CONCLUSION

Presentation level application integration is a useful approach for many integration problems, especially if human interaction is involved, and may be less invasive and costly than integration at the business logic level. The approach presented in this article is intended to be used for enterprise and consumer portals in the banking and insurance domain. It supports presentation level data exchange and navigation integration for distributed portlets in heterogeneous environments.



The integration of remote and heterogeneous portlets is a necessity in large organizations with widely autonomic suborganizations, and in business to business collaborations, where components are to be reused across organizations. In such cases, components are typically hosted and maintained by computing centers at different locations with heterogeneous hard- and software infrastructure. Thus support for remote integration and communication as well as platform independence are a necessity in this context.

Finally, distribution and heterogeneity are core characteristics of the internet itself. Composition of new web pages at the presentation-level from globally distributed portal components might be an interesting approach for building future websites. It seems obvious that apart from remote communication and support for heterogeneity, such an approach would benefit from a component model and a descriptive approach for component composition as presented in this paper.

12 ACKNOWLEDGEMENT

The presented approach has been developed as part of the Enterprise Portal Architecture (ENIPA) project, which is a joint-project of the Software Competence Center Hagenberg (SCCH) and the GRZ-IT Center Linz. We wish to thank Hermann Lischka, Thomas Kriechbaum, and Franz Oberndorfer from the GRZ IT-Center Linz and Helmut Maier from the SCCH Hagenberg for many fruitful discussions and for implementing parts of the described services.

REFERENCES

[Allamaraju, 2005] Allamaraju, S. (2005): Inside WSRP. BEA dev2dev Portal (http://dev2dev.bea.com), March 7, 2005, Retrieved October 31, 2005, from http://dev2dev.bea.com/pub/a/2005/03/inside_wsrp.html.

[Alonso et. al.; 2004] Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004): Web Services – Concepts, Architectures and Applications, Berlin Heidelberg New York: Springer Verlag.

[BEA et. al., 2003] BEA, IBM, Microsoft, SAP, Siebel (2003): Business Process Execution Language Version 1.1. Retrieved June 28, 2005, from http://www-128.ibm.com/developerworks/library/ws-bpel/, currently being standardized as WSBPEL by the OASIS consortium, http://www.oasis-open.org.

[BEA, 2005] BEA (2005): BEA WebLogic Portal: Using WSRP with WebLogic Portal, Version 8.1, Service Pack 5, October 2005.

[Chari & Sheshadri, 2004] Chari, K., Sheshadri, S. (2004): Demystifying Integration. Communications of the ACM, Vol. 47, No.7., 58-63.

[eXo, 2005] Ruh, W.A., Magginnis, F.X., Brown, W.J. (2001): Portlet API extensions. Retrieved October 31, 2005, from http://www.exoplatform.com.

[Gorton & Liu 2004] Gorton, I., Liu, A. (2004): Architectures and Technologies for Enterprise Application Integration. Edinburgh: Proceedings of the 26th International Conference on Software Engineering (ICSE'04).

[Heute et al., 2007] Heute, T., Viet, J., Russo, R., Dawidowicz, B., Laprun, C. (2007): JBoss Portal 2.6.0-GA Reference Guide. Retrieved June 25, 2007, from http://cruisecontrol.jboss.com/cc/artifacts/jboss-portal-latest-doc/referenceGuide/html/index.html.

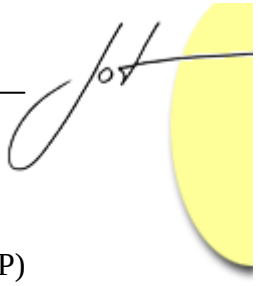
[Hiranniah & Konduru, 2003]Hiranniah, S., Konduru, S. (2003): Inter-Portlet Messaging with WebSphere Portal. WebSphere Advisor Magazine, Retrieved June 28, 2005, from http://websphereadvisor.com/doc/11928.

[JCP, 2003] Java Community Process (2003): The Java™ Portlet Specification (JSR 168) V.1.0. Retrieved June 28, 2005 from http://www.jcp.org/en/jsr/detail?id=168.

[JCP, 2007] Java Community Process (2007): The Java™ Portlet Specification (JSR 286) V.2.0. Retrieved June 25, 2007, from http://www.jcp.org/en/jsr/detail?id=286.

[Linthicum, 1999] Linthicum, D.S. (1999): Enterprise Application Integration. Boston: Addison Wesley.

[Linthicum, 2003] Linthicum, D.S. (2003): Next Generation Application Integration: From Simple Information to Web Services. Boston: Addison Wesley.



[OASIS, 2003] OASIS (2003): Web Services for Remote Portlets (WSRP) Specification V.1.0. Retrieved, June 28, 2005, from http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf.

[OASIS, 2006] OASIS (2006): Web Services for Remote Portlets (WSRP) Specification V.2.0. Retrieved June 27, 2007, from http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec.html.

[Oracle, 2004] Oracle (2004): Oracle® Application Server Portal Developer's Guide, 10g (9.0.4), Part No. B13922-01, May 2004.

[Oracle, 2005] Oracle (2005): Oracle® Application Server Portal User Guide, 10g (10.1.4), Part No. B13809-04, September 2005.

[Oracle, 2006] Oracle (2006): Oracle Application Server Portal Technical Note: Building Highly Interactive Portlets With AJAX. Retrieved July 19, 2007, from http://www.oracle.com/technology/products/ias/portal/pdf/portlets_building_with_ajax.pdf.

[Osmond, 2007] Osmond, M. (2007): A JSR168-compliant implementation of inter-portlet communication. Retrieved June 26, 2007, from http://www.doc.ic.ac.uk/~mo197/portlets/portlet_messaging.

[Roy-Chowdhury, 2003] Roy-Chowdhury, A. (2003): Using Cooperative Portlets in WebSphere Portal V5. IBM Raleigh Lab. Retrieved June 28, 2005, from ftp://ftp.software.ibm.com/software/dw/wes/pdf/CooperativePortlets.pdf

[Roy-Chowdhury & Wu, 2004] Roy-Chowdhury, A., Wu, Y.C. (2004): Developing JSR 168 compliant cooperative Portlets. IBM Websphere Technical Library, Retrieved June 28, 2005, from http://www-106.ibm.com/developerworks/views/websphere/libraryview.jsp.

[Ruh et. al., 2001] Ruh, W.A., Magginnis, F.X., Brown, W.J. (2001): Enterprise Application Integration. New York: John Wiley & Sons.

[Thiagarajan 2007] Thiagarajan, R. (2007): WSRP 2.0 - A Sneak Peek. Retrieved June 25, 2007, from http://blogs.sun.com/trajesh/entry/wsrp_2_0_a_sneak.

[W3C, 2003] W3C (2003): SOAP 1.2 Specification, W3C Recommendation. Retrieved June 28, 2005, from http://w3.org/2000/xp/Group/.

[W3C, 2004] W3C (2004): Web Service Description Language (WSDL), Version 2.0, W3C Working Draft. Retrieved June 28, 2005, from http://w3.org/2002/ws/desc/.

[Weinreich & Ziebermayr, 2005] Weinreich, R., Ziebermayr, T. (2005): Enhancing Presentation Level Integration of Remote Applications and Services in Web Portals, Florida: Proceedings of the IEEE International Conference on Services Computing (SCC 2005).

About the authors

Rainer Weinreich is an Assistant Professor at the Department of Business Informatics - Software Engineering, Johannes Kepler University of Linz, Austria. His current research interests include component-based, service-oriented, web-based and enterprise software architectures. He can be reached at rainer.weinreich@jku.at, see also http://www.se.jku.at/weinreich

Andreas Wiesauer is research and teaching assistant at the Department of Business Informatics - Software Engineering, Johannes Kepler University of Linz, Austria. His current research interests include enterprise portal technologies and web services. He can be reached at andreas.wiesauer@jku.at

Thomas Ziebermayr is software engineer at the Software Competence Center Hagenberg, Austria. He is currently working in the area of semantic web service composition. He can be reached at thomas.ziebermayr@scch.at