Building and Managing Enterprise Wide Web Portals - Tutorial

Jana Polgar, Robert Bram, and Tony Polgar
Monash University, Melbourne, Australia

Jana.Polgar@infotech.monash.edu.au
Robert.Bram@colesmyer.com.au Tony.Polgar@colesmyer.com.au

Abstract

This tutorial aims at taking the participants through the concept and architecture of Web services, their components such as servlets, and JSPs; concept and usage of WSDL, and JAX-RPC-based web services. Furthermore, we introduce the concept of portals, portal page and portlets. A small part of the tutorial is devoted to the deployment practices of both – Web services and portlets. Relevant deployment descriptors – `portlet.xml, web.xml` - and the structure of the deployment directories are discussed. The closing remarks concentrate on the development issues from the perspective of the development manager and developer (programmer). The tutorial is based on the book "Building and Managing Enterprise wide Web services and portals" published by Idea Group. The additional material is available from http://neptune.netcomp.monash.edu.au/cpe4004/index.html .

Keywords: Web service, portal, portlet, servlet, service endpoint, WSDL, UDDI, SOA.

Introduction

The promise of reduced complexity of software, and software development in the third and fourth generation of computers has simply not eventuated. The catch phrase of the client server paradigm was a promise to reduce the complexity of the software by dividing electronic processing to two components- a smaller component– performed on a client computer, and the larger component, performed on a server. This way, the client specialized to front end processing while leaving the processing logic to the backend server. This was supposed to unravel the complexities of what we call today the separation of concerns. The idea of the Graphical User Interface (GUI) was born and its workplace was firmly allocated to the client. Nearly immediately, the user interface became much more sophisticated, filling the complexity space with GUI API's, event processing, and introducing user interface standards and objects.

On the servers' side, the void was also filled with additional functionality: generalized database connectors, capabilities to create lightweight processes called threads, which introduced additional complexity by the necessity to synchronize the execution of some of the parts of the threads, support for a large variety of devices and protocols, and ever-increasing complexity of logic.

The development process of the client – server programs did not become simpler either: the two components were often developed in two or more different programming systems. For example, it was very common to develop client software in Visual Basic

or Delphi, while the server software was developed in C or C++ (or both). The connection between client and server needed to be more robust, involving heterogeneous clients and servers – CORBA was suggested as a solution. The previously cohesive development team was broken into client and server specialists. The functional testing introduced a new, expensive, and difficult discipline: integration testing, where the client and server were seen in cooperation, for the first time. Finding errors, bugs, and functional irregularities became complex and very expensive as several specialists had to cooperate in resolving the problems.

The initial promise of reduced complexity was broken. The development managers were reeling with increased costs and were looking for a solution which would address the increased number of development personnel, testing and deployment costs. The developers were faced with a specialization dilemma – should I become a client or a server specialist? It should be remembered that the client specialist was also the user interface specialist, being responsible for the design of the user interface and its behavior.

World of Web: While the client server computing was brewing, the world of web was quietly assessing its strength. We talking about the period starting in 1889-1990: The following extract from Tim Berners Lee's original work (Berners-Lee, T. (1999) describes the birth of the World Wide Web concept:

> The first web browser - or browser-editor rather - was called WorldWideWeb as, after all, when it was written in 1990 it was the only way to see the web. Much later it was renamed Nexus in order to save confusion between the program and the abstract information space (which is now spelled World Wide Web with spaces).

The new web technology brought some simplification in the technology – but only for a brief period of time. The browser was thin – essentially displaying a HTML document and the server was rather primitive and very slow, the logic often implemented in a CGI (Common Gateway Interface) script. With the introduction of Java, the feeling of victory was also introduced: a single technology was used on both platforms, with a Java applet running in a browser and a Java application running on the server.

The legacy systems continued to be used and maintained and people wanted to use them in the "net", without the necessity to re-write them. A lightweight web application would do precisely that – provide access to a backend system. The back-end could be seen as a service – **Web service**. The service functionality and access points would be described in XML and this document would be accessible via some kind of registry (UDDI).

Now we approach a critical point in thinking – suddenly people wanted to access a *multiplicity* of applications in one easy to use window. The idea of a **portal** was born.

The necessity to provide *integrated* web services stems from the fact that modern businesses often grow by mergers and acquisitions. Although portals and Web services provide smooth and elegant access to organizations' data, support B2E, B2B and B2C models, development and implementation cost is quite high.

Portal and SOA

The look of portal has been used by media in TV and elsewhere for some time. TV programs, specifically, sport transmissions often used more than one window on one screen. The same idea is used in portals: it can display more than one window on a Web page. The trick was to get the windows to work so that the user can move from window to window, without losing the context, all within one Web page. Presumably, the windows somehow relate to the user needs. This leads to the requirement of configuration of the portal so that user can choose which windows to see on the relatively small real estate of the screen. If the user can choose what to see, then the system

should be able to provide security such that the user is authorized to access everything that has been configured, with a single sign-on.

Service Oriented Architecture can be defined as a special kind of software architecture which configures entities (services, registries, contracts, and proxies) to maximize loose coupling and reuse. SOA is set of principles and building blocks which utilizes the integration idea such that independent Web services applications can be re-used and connected by all participants with a minimum integration effort (Barry, D. K. (2003). The internal interfaces can be built and achieved by currently used Web services and the user interface integration can be achieved by the currently used portal technologies. So it is possible to make the next step in building loosely coupled collections of applications which perform seamlessly in an ever-changing business environment.

Implementation of SOA is possible with existing investment. It involves a change in the conceptual thinking and the use of existing Web services, portal services, and grid computing.

Advantages of Service Oriented Architecture are in providing a method to leverage existing assets to achieve faster time to market, reduced cost, and business improvement. It allows you to use grid computing as a virtual resource and on-demand deployment. SOA helps maximize resources in the organization. It is worthwhile remembering that the Web services are a set of enabling technologies for SOA.

Web Services

According to W3C (World Wide Web Consortium organization) a Web service is defined as follows:

“A Web service is a software system identified by a URI whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols.”

Closer look at the definition and a little bit of discovery of actual “plumbing” gives you a simpler view of the Web service that is a software application, accessible over the Internet or Intranet through a URL able to cooperate with another loosely coupled application. Web service clients use XML-based protocols, such as Simple Object Access Protocol (SOAP) to transfer data over HTTP. The access to this Web service application is obtained through the interfaces and bindings defined with XML based Web Services Description Language (WSDL) (The Java Web Services Tutorial).

Web services aim at providing application-to-application interoperability. The messages are exchanged between two parties called service provider and services requestor. The messages are described in an abstract way and then bound to a concrete network protocol and message format. The message exchange between provider and requester results in invocation of an operation. A collection of operations represents an interface to the service. This interface is then bound to a concrete protocol and message format via one or more bindings. The interface definition and operation implementation are responsibility of service providers.

The implementation of Web service endpoint does not require any specific platform (Christensen, E., Meredith, G., Curbera, F., & Weerawarana, S. (2001). In this tutorial, we focus on Sun’s Microsystems J2EE platform as the major contender in Web service endpoint implementation. The J2EE platform supports a multi-tier distributed application model (Figure 1). A typical structure defines a client tier, a middle tier, and a back-end tier.

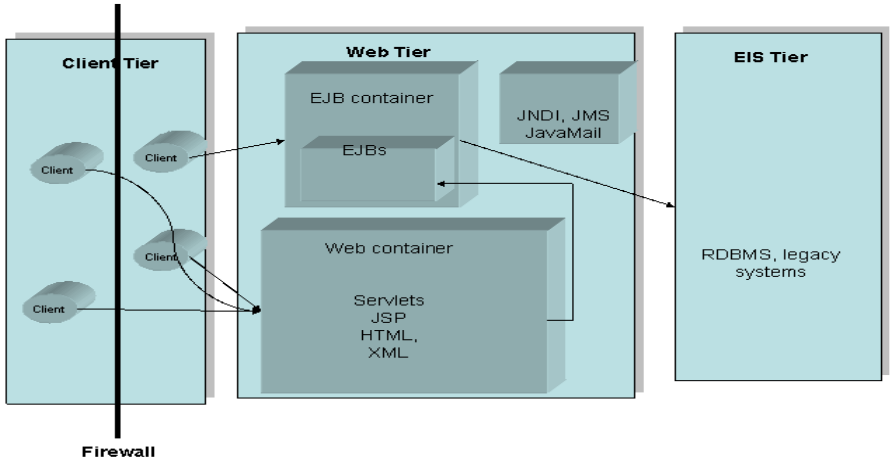

Figure 1 J2EE multi-tier architecture

The client tier is expected to support a variety of client types outside and inside of company firewalls. The middle tier is composed of one or more sub-tiers and interacts with clients. This tier utilizes Web containers and business logic implemented in components such as JavaBeans, and Enterprise Java Beans. The back end of these applications is formed by the enterprise information systems (EIS tier).

Web Service Description Language (WSDL)

WSDL is a language which attempts to describe communications between Web entities in a structured way. In a WSDL document, the services are defined as a collection of network endpoints called ports. This language is extensible and allows the definition of other binding mechanisms, such as Java and SOAP over Java Messaging Service (JMS).

The WSDL document contains the following main elements:

Types: a container for data type definitions using some type system for example XML schema (`xsd`).

Message: an abstract, typed definition of the data being communicated. A message can have one or more typed parts.

Port: a single endpoint, which is defined as an aggregation of a binding and a network address.

Port type: an abstract set of one or more operations supported by one or more ports.

Operation: an abstract description of an action supported by the service. The operation defines the input and output message and optionally a fault message.

Binding: a concrete protocol and data format specification for a particular port type. The binding information contains the protocol name, the invocation style, a service ID, and the encoding for each operation.

Service: a collection of related ports.

A Web service is built on XML operations where each XML operation defines a response to a particular client request. When an XML operation is executed, the Web service has to perform the following activities:

- It parses the XML document, extracts the document elements, and maps them to parameters of the methods called by the XML operation.

- It invokes the methods defined in the XML operation.
- It creates a response in the form of a XML document using return values from the methods.
- Finally, it returns the XML response to the client.

The support for developing Web services on the Sun's J2EE platform is primarily based on the implementation of Java APIs for XML and Web Services Complete list and documentation for J2EE platform can be found on http://java.sun.com/j2ee/ and http://java.sun.com/webservices/jwsdp/index.jsp

The following is a brief overview of selected API packages:

- JAXP (API for XML processing) provides support for XML documents using simple API for XML (SAX) and Document Object Model (DOM). A part of this processing is also XML Stylesheet engine (XSLT). These APIs give the developers XML platform independence and freedom to use different XML parsers (swapping from high performance parsers to memory conservative ones). This type of processing is suitable for fast enablement of e-commerce applications, application integration or dynamic Web publishing.
- JAX-RPC APIs support SOAP based interoperable and portable Web services. It is based on RPC (Remote Procedure Call) programming model and allows develop Web service clients and endpoints. We will provide more details in following sections.
- SAAJ stands for SOAP with Attachments API for Java. It enables the developers to construct and accepts messages conforming to SOAP specification. These APIs are derived from `java.xml.soap` package.
- JAXM (Java API for XML Messaging) defines messaging service. These APIs support sending and receiving document oriented XML messages.
- JAXB (Java API for XML binding to Java representation. These APIs provide support for using platform-neutral XML data in Java applications.

Web Service Discovery and Invocation, JAX-RPC platform

Web services and its clients can be built using J2EE components (for example JAX-RPC is very common for client-server models). The developer specifies the Web service as a collection of remote procedures and their interfaces. The remote procedures are invoked by remote Java clients. The Web service is then implemented as Web service endpoint which must adhere to the previously defined interface. The Web service endpoint is a set of methods which implement business logic. A client accesses the Web service using a Service endpoint interface.

The first step for the client is to locate the service using a registry. Once the Web service location is found, client discovers the options to access it and binds to the service. The client makes a request to the service as simple method invocation with parameters passed to it. There are three possible methods how to make the call: invoking methods on generated stub, using dynamic proxy, and using dynamic invocation interface (DII). JAX-RPC supports only limited subset of Java types to ensure interoperability with Web services implemented in other language. Basic rule is that all types must be serializable to and from an equivalent XML document representation form.

Client developers create a local object – proxy or stub - whose interface is the same as the remote service interface. The client simply makes a request and invokes the methods on the proxy. The

request is transferred to the server using HTTP and SOAP. There is no need to develop any parsers for parsing SOAP messages on either side. Parsing is done by the underlying runtime subsystem (in the case of JAX-RPC this is the JAX-RPC runtime system and JAX-RPC APIs). J2EE based Web services can be invoked by any Web service client.

In order to handle communication between a client and a service endpoint, the service endpoint must also define interfaces and proxy object (called tie or skeleton). The access to service endpoint implementation is invoked by the server passing the incoming message to the tie object. The stubs and skeletons on the client and server side respectively are commonly called artifacts. It is typical that the vendor who provides the runtime implementation of connection between clients and servers provides some tools to generate stubs and skeletons or ties. For example, the JAX-RPC specification does not require any specific tools to generate artifacts, but the majority of implementers of JAX-RPC provide the compile tool to ease the programming effort.

JAX-RPC overview

JAX-RPC provides support for two types of mapping: WSDL-to-Java and Java-to-WSDL. The mapping and provision of tools for mapping is part of the development environment for designing Web service clients and endpoints.

Web service runs on request-response mechanism. In the typical Web service scenario, a client request starts series of business processes within the company. The basic workflow starts with a client making a request to a particular Web service such as asking for example for a train reservation in a particular country or a weather report. The service processes the request and returns the response to the client. When both the client and the Web service reside in a Java environment, the request for service is represented by a Java method invocation, along with passing the required parameters. The response is the result of this method invocation. In, JAX-RPC client uses SOAP message to request services from J2EE service implementation.To make a Web service available to clients through JAX-RPC, developers must provide JAX-RPC service endpoint definition and implementation. This means that two classes for each endpoint must be defined: one class specifies the JAX-RPC service endpoint interface and the other implements this interface. There are three different models for a service endpoint invocation. These models are independent of any specific service implementation model.

Building Web Service endpoints - Servlets

Servlets are server side programs which respond to requests from browsers. They run in the Web environment. It should be remembered that the portal technology grew up from the servlets and each portal page ends up as servlet. The servlet technology is the foundation of Web application development using the Java programming language. Therefore, understanding the servlet technology and the Web server architecture is important.

Let’ have a look at some advantages of using servlets.

- Servlets show **better performance** compared to CGI. CGI enables the Web server to call an external program to process a HTTP request, accept the response, and pass it to the client. In this fashion, each HTTP request requires the server to spawn a new process which affects the performance. With servlets, each request is handled by the servlet container process (typically called an Application server or engine). When the servlet completes the processing, it stays resident in the memory, ready to process another request.
- Servlet applications use Java which guarantees their **portability** within Java world. They can be moved to any operating system which supports Java.

- Servlets have access to the rich Java library which helps to **speed up the development cycle**.
- The applications based on servlets can be built for **robustness.** The developers do not have to pay much attention to issues like memory leaks and garbage collection.
- Java Servlets are in version 2.4 and they are part of the J2EE (Java 2 Enterprise Edition) platform.

Similarly to many other components of the J2EE architecture, the servlets are living in containers. The life cycle of a servlet is controlled by the container in which the servlet is deployed. All servlets must implement `javax.servlet.Servlet` interface. This interface defines the following methods corresponding to the servlet lifecycle phases:

- *Initialization phase* through the `init()` method. When a container loads the servlet, its `init()` method is invoked before servicing any requests.
- *The service phase* (`service()` method) represents all interactions with requests until the servlet is destroyed. The `service()` method is invoked once per request and it is responsible for generating the response to the request. The `service()` method takes two parameters as objects:
  - *request object* `javax.Servlet.ServletRequest` which represents the client's request for dynamic resource (URL)
  - *response object*: `javax.Servlet.ServletResponse` is the object into which the servlet constructs the response (typically a textual document with HTML tags and dynamic content).
- *Destruction phase* is represented by the `destroy()` method. The container calls this method when the servlet is being removed from use and gracefully terminates (freeing up all the resources the servlet created).

When the request arrives and servlet container maps it to a servlet via its deployment descriptor (`web.xml`), the following steps are performed by the container:

- If the servlet does not exist, the servlet class is loaded, the instance of the servlet class is created and initialized by running the `init()` method.
- After the initialization is completed, the `service()` methods is invoked and request and response objects are passed to it.

The initialization process can be customized. Customization allows the servlet to read persistent configuration data, and initialize the resources. For example, this method can be used to initialize the variable pointing to the database object created by a context listener.

Listeners are used to monitor events in the servlet's life cycle. To listen to the servlet's life cycle, we define a listener receiving life cycle events. The listener can listen to the Web context, session, or request. The listeners are means for giving the developer (or rather the application) more global control over the available resources. Four kinds of event listeners respond to Web application life-cycle events:

- *Servlet context listeners*
- *Servlet context attribute listeners*
- *Session listeners*

- *Session attribute listeners*.

Filters are used to examine header or content of the request or response. Filters usually do not create response. They provide functionality that can be attached to the Web resource. This also means that the filter should not have any dependencies on the resource it is attached to. The filtering APIs are defined by several interfaces from `javax.servlet` package: `Filter`, `FilterChain`, and `FilterConfig`. The `Filter` interface can be implemented to define a specific filter. The following methods of `Filter` class are recommended for implementation: initialize the filter, provide custom processing and dispose of the filter (`init()`, `doFilter()`, and `destroy()`).

There are a variety of scenarios for the usage of filters. Typical ones are implementations of a hit counter or perusal of items in the shopping cart. The Web container uses filter mappings (`web.xml` file) to decide how to apply filters to Web resources. Matching can be done either by using the name of the servlet (`CountryServlet`) or by the URL pattern attached to the servlet (`alias`). One filter can be mapped to more than one servlets.

Java Server Pages

Good Web application design always tries to separate business objects from controls and the presentation. The benefit of using Java Server Pages (JSP) is the ability to separate Web page design from functionality and content dynamics (coding). The main features of the JSP technology are:

- Text-based documents written in a language which describes request processing and response construction.
- Extensibility to define the tags and associate them with functionality.
- Ability to access objects on the server.

A JSP page is a text-based document which contains both static data as HTML, WML, SVG or XML, and JSP elements which provide dynamic content. The servlets include Java classes that generate dynamic Web HTML content using `print()` statements. The JSP technology redefined servlets and it is now a key component of the J2EE specification. Many commercially available application servers support JSP (as an example: BEA WebLogic, IBM WebSphere, Sun Microsystems Application Server 8). A JSP page is a traditional Web page with the HTML markup and some “invisible” code hidden in the tag library. The prescribed file extension is .jsp and the server provides a special processing mechanism for this.

Deployment

Each Web application (composed of servlets, JSP, Java beans, and HTML) and its components (Web modules) must be deployed. Deployment ensures that the Web server, application server and/or portal server can resolve references to all objects. Application Web modules must be packaged into a WAR file. Several options are available:

- Using a deployment tool (for example, Sun Microsystems Application Server 8 comes with such tool – *deploytool* (or, on the command line, the *asant* utility in Sun Microsystems Application Server 8)
- Using the *jar* utility in a directory with prescribed layout
- Using the administrative console to deploy and launch new Web applications already packaged in a WAR file

It should be remembered that the war file is a hierarchical structure. The application war file must contain a **deployment descriptor** called `web.xml`. The `web.xml` file controls the application's life. All components (servlets, filters, applets, JSP) must be declared, and their mapping into URL patterns must be defined. The configuration information (initialisation values) consists of a set of string values. If the application is using listeners, they must be registered with the container.

Service EndPoint Design

The strategy for Web service design is characterized by the need to separate presentations from control and business data processing in Web service design. This strategy deals with the separation of concerns at the service level. As servlets and JSPs must be capable of supporting multiple views for multiple devices, the key design pattern is the well-known *model-view-controller* (MVC) design pattern[1]. In order to enable the interaction with clients, Web services must expose their interoperable interfaces. Web services can either be built from scratch, or can be added to the existing application thus make it look like a Web service. There are two separate issues in the design:

- Design of Web service capabilities, their interoperability, and efficiency of logic flow
- Design of the business logic.

Business logic is typically designed with no regard whether the application is used as a Web service (no consideration is given to the interoperable interface structure).

Design of Web service capabilities should start at a higher abstraction level. We start with splitting the responsibilities by assuming (similarly as with XML based Web service endpoint) that each Web service endpoint has service interaction layer (service interface) and service processing layer (implementation of business logic processing).

Service interaction layer

The service interaction layer must handle several major responsibilities relevant to fulfilling the service contractual obligations. The scope of this paper does not allow us to explore the details. There are two approaches to the service interaction layer development:

Approach 1 - Create Java interfaces first and then map them into WSDL. This approach seems to be very convenient. If there are vendor tools available for mapping Java to WSDL (for example *wscompile* from Sun Microsystems), no knowledge of WSDL is required, meaning that even less experienced developer can use this approach. However, there are some potential problems with this approach: Java interfaces are often subject to change and then it may be hard to adjust the Web service interface without making a change in the corresponding WSDL document. Alternatively, changing the WSDL might require rewriting the service's clients. The developer must be also aware that different tools may use different interpretations of Java data types.

Approach 2 - Develop WSDL description of the service and then build the corresponding Java interfaces. Similarly to CORBA and IDL, a detailed knowledge of WSDL and Web service interoperability requirements is necessary. Even a small mistake can lead to difficulties in communication with clients. This approach requires experienced (and more expensive) developers to de-

[1] A *design pattern* is a common strategy for a problem which occurs repeatedly. It is a generalised solution. MVC design pattern was first published in the classic book by Gamma, Helm, Johnson and Vlissides (1995). [Design Patterns – Elements of Reusable Software]

sign and write the application. Programmers who like to use so called method overload[2].meet with further difficulties as there are limitations to this approach. Each method call and its response in WSDL description are represented by unique SOAP messages. To represent overloaded methods, the WSDL description would have to support multiple SOAP messages with the same name. WSDL version 1.1 does not have this capability.

HTTP is a stateless request-response protocol. Consequently, Web-based enterprise applications need a mechanism for identifying a particular client and the state of any conversation between client and server. Clients view Web services as stateless and assume that a service endpoint retains no knowledge of previous client interactions. However, in some cases the client makes a sequence of calls to a service to accomplish a given operation. In this case, the client application has to maintain the state. There are several strategies, some of them have already been discussed in previous section, how to maintain the application state:

Use of `HttpSession` object associated with a servlet (The Java Web Services Tutorial. (2004). Sun Microsystems Inc. Retrieved from http://java.sun.com/webservices/tutorial.html

- J2EE 1.4 Java API specification. (2004)
- *Use of cookies*: A cookie is a small chunk of data the server stores on the client's platform. Each time the client sends a request to a server, it includes the headers for all the cookies it has received from that server. Browsers often provide options for the user to disable cookies in the application.

We like to think about cookies as a last resort since they always leave a footprint on the client's machine.

Integrating servlets with JSP

A controller in the MVC pattern is mostly implemented by a servlet. However, the controller can be implemented by a JSP page calling `RequestDispatcher.forward()` directly or using a custom tag,. The guidelines for using J2EE typically recommend that the controllers should strictly be implemented as servlets, because controllers are logical components, not presentation components. Code in Figure 2 demonstrates how a controller can be implemented as a JSP page.

```
<% String landscapeLocality = request.getParameter("landscapeLocality ");
if (landscapeLocality.equals("Sydney")) { %>
<jsp:forward page="/processSydney.jsp"/>
<% } else if (landscapeLocality.equals("Melbourne")) { %>
<jsp:forward page="/processMelbourne.jsp"/>
<% } %>
```

Figure 2 JSP as a controller

The code example shows the scriptlets and tags conditionally forwarding the request to another JSP page based on a request attribute. We have some reservation towards this approach since it clearly mixes the processing logic and presentation view. In this case, the J2EE coding guidelines suggest a better option that is to implement the controller as a servlet to avoid mixing conditional attribute tests with the HTML tags.

[2] Overloaded methods share the same method name but have different parameters and return values.

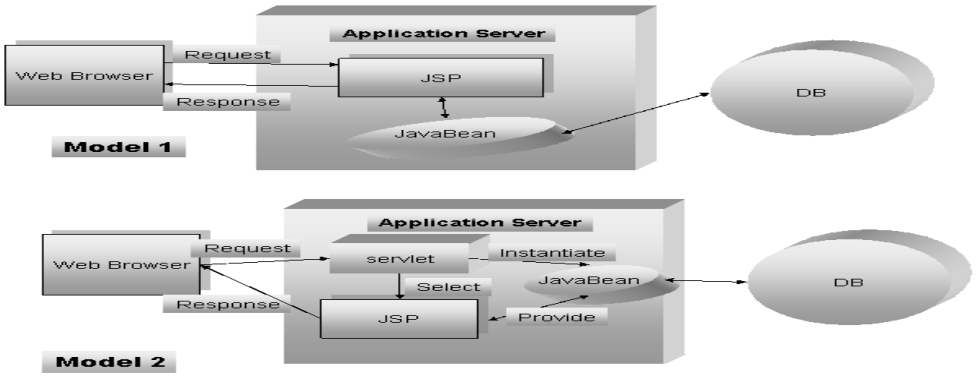

Figure 3 JSP implementation models

We believe that the problem of easily mixing code with HTML tags is arising from the JSP specification itself. The JSP specification allows two approaches for building Web applications (Figure 3): JSP model 1 and JSP model 2 based on MVC2[3]. These two models differ in location where the processing takes place. In model 1, the JSP is responsible for processing requests, extracting conditions on which to decide which activity is to be processed next, interacting with the data through the JavaBean component, and generating responses. Such implementation could result in breaking all the good design guidelines mentioned previously in this section: the control is bundled in JSP. In model 2, the control is performed by the servlet (proper Java code), JavaBean components are used as the model part of MVC and presentation in JSP avoids any control scriplets being concerned predominantly with HTML code.

Portals

In the following section the concept of portals and portlets, but not entire portlet specification, are explained. Let us start with some definitions of portal components.

Definition - 1: A *portal* is an integrated and personalized Web-based application that provides the end user with a single point of access to a wide variety of aggregated content (data, knowledge, and services) anytime and from anywhere using any Web-enabled client device.

Definition – 2: Portlet is a Java technology based component managed by a portlet container that processes requests and generates dynamic content.

Definition – 3: A portlet container provides run time environment for portlets and manages their lifecycle.

What are portals? Portals are containers for the next-generation desktop, delivering e-business applications over the Web to all kinds of client devices and integrating dispersed information from heterogeneous data sources. In addition, the portals provide site users with a *single point of access* to multiple types of information and applications. Regardless of where the information resides or the format it uses, a portal aggregates all of the information in a way that is relevant to the user. The purpose of portal solutions is to provide end user with convenient access to interact with enterprise applications, people, content, and processes. Users can personalize and organize their own view of the portal, manage their own profiles, publish and share documents.

A simplified version of the Definition 1 defines the necessary elements of portal applications:

[3] MVC2 pattern follows the original MVC patter with a few exceptions further explained in the tutorial notes.

- Portal combines several legacy back-end systems and applications at the request time into one page.
- It provides single sign-on facility.
- The content is highly dynamic, can be filtered, personalized, and secured.
- Portal is designed to service thousands of concurrent sessions in an implementation.
- Enterprise applications can run on load-balanced Web and application server clusters.

Web developers are facing problems introduced by increased complexity of Web applications. The problems are characterized by requirements for integration of multiple user interfaces and/or access to multiple data containers. Enterprise and corporate portals emerged as the strategy for developing enterprise applications that can provide the integration of diverse data sources, enable companies interact with external business partners as well their employees.

What do portals deliver?

Presentation services: portal Presentation Services provide customized and personalized pages for users through aggregation of content. Page content is aggregated from a variety of databases and applications. The portal presentation framework simplifies the development and maintenance of the portal by defining the page structure independently of the portlet definition. Portlets can be changed without impact to the overall portal page structure.

Single sign on (SSO): SSO is a popular feature enabling the end user to authenticate only once to obtain access to all relevant aggregated data. This feature makes portal different from Web Services where the end user may have different user name and password for each Web service[4]. It is important to realize that portlets are often shared between many portals and several pages of the same portal. They may communicate with back end enterprise applications which may have identity and authorization security constraints. This situation gets more complex when portlets communicate with other Web services and back end applications.

Directory services: Directories are often likened to databases. What sets them apart from general-purpose relational databases is the frequency of read requests and the defined database schema. Directories support high volume of read requests and they are optimized for fast read access.

Security: The security standards are playing an important role in the development of portals. They are based on encryption and decryption principles.

Content management: Portals often provide services to facilitate connections to content management sources.

Collaboration services: Collaboration services are provided by WebSphere portal through a set of predefined portlets. These portlets allow for team-cooperation, chat, e-mail, calendaring and many other collaborative technologies. In WebSphere 5, cooperative services also support page-to-page communication.

Search and taxonomy: Majority of portals offers search services. The search capability enables searches across distributed HTML, documents, and text data sources. The search may view documents in terms of categories, or taxonomies. The values of the taxonomy items are defined in metadata variables, associated with particular documents. The search can crawl a Web site and is

[4] It is of course possible to synchronize multiple user authentications across all Web sites. Furthermore, Web services may be distributed and ask for original user credentials. However, the synchronization is costly and requires customized solutions.

configured to follow several layers in a site, or to extend beyond several links in a site. For example, WebSphere provides simple search service called Portal Search. In addition, IBM Extended Search and Enterprise Information portal can be fully incorporated into the portal environment. These search engines provide federated searches across numerous data sources.

Support for mobile devices: Some vendors such as IBM provide transcoding technology which is integrated with WebSphere portal to transform the portal markup produced by the portal to mark up for small devices such as mobile phones and personal digital assistants (PDAs).

Site analysis: Sometimes it is practical to know who and how the portal is accessed. Some portal implementations provide means for gathering and analyzing the information about Web site visitor trends, usage, and content. For example with WebSphere, you can use the underlying WebSphere Application Server technology and Site Analyzer to collect such information and use it to improve the overall effectiveness of the site.

Personalization: Most portal implementations provide the capability for personalization. This is accomplished through either *edit* or *configure* mode of the portlet. In the *edit* mode, the changed values refer to the instance of the portlet which is being edited. In the *configure* mode, the changed values are shared in all instances of the portlet on every page it appears throughout the whole portal. Change to a different mode is achieved by the user pressing an icon on the portlet.

In both cases, the changed values are stored in the underlying database and survive server restart. It is important to realize that the modes are detected by the code and appropriate screen images are displayed in each case.

The edit and configure mode must be enabled by the portal administration function and its record is kept in the appropriate configuration xml file.

In summary, personalization is achieved in three steps:

- Appropriate mode is enabled by in the configuration file.
- A new JSP is created for each mode. Personalized values are set on the screen.
- The user performs the appropriate personalization, if authorized to use this portlet mode.

Please note that the above functions can be achieved through normal portlet functions. The above modes are designed to let the non-technical administrators to configure a portlet without any technical knowledge.

Web services integration: portals often provide some services for exposing and integrating portlets as remote portlets hosted on another portal platform. The cooperation and messaging is achieved by using the Simple Object Access Protocol (SOAP) protocol. The SOAP protocol supports middleware applications. It hides the process of packaging and responding to a SOAP request from the developer and the administrator.

Portlet concepts

A portlet is an application that displays some content in a portlet window (WebSphere Portal Fundamentals, 2004). They form the base building blocks of a portal. A portlet is developed, deployed, managed, and displayed independently of all other portlets. Portlets (Portlet Specification JSR 168 (2005); Servlets Specification. 2.4 (2004) may have multiple states and view modes. They also can communicate with other portlets by sending messages.

Portlets share some similarities with servlets:

- Portlets are Java technology based Web components. Both servlets and portlets are J2EE Web components,

- Portlets and servlets are managed by containers (portal server or application server respectively),
- Portlets and servlets generate Web dynamic content, and
- Portlets and servlets interact with Web client via a request/response paradigm.

The following aspects differentiate portlets from servlets:

- Portlets only generate markup fragments, not complete documents. Aggregation happens in the portal container
- Unlike servlets, portlets do not have URL, consequently they cannot be independently displayed in a browser
- Web clients interact with portlets through a portal container
- Portlets have predefined portlet modes (edit, view, normal, configure, help) and window states (maximize, minimize). These indicate the function the portlet is performing and its size of the real estate on the portal page
- Portlets can exist several times in one or more portal pages
- Portlets have a more refined request handling in terms of distinguishing between action requests and render requests. Action request is originated via URL by the client as opposed to render request which is called by the portlet container as a response to the portlet current state

The following points make portlets more sophisticated than servlets:

- Portlets have access to user profile information. This capability exceeds the basic user and role associations, and enables implementation of more advanced facilities for authentication, customization, and security.
- Portlets can perform portlet rewriting. They can create links which are independent of the portal server implementation, for example session information tracking.
- Portlets store objects in two different session scopes: application wide and portlet private.

However, portlets are lacking some of the servlet features:

- Portlets cannot change HTTP headers or set response encoding
- Portlets generate fragments which are placed on a servlet page. Therefore, some restrictions applicable to servlets will take place, such as access to the servlet's URL and page related tags.

Portlet lifecycle

Portlet container is responsible for management of the portlet lifecycle and invocation of its life cycle methods. The main events in the portlet's life are depicted in Figure 4.

- *Portlet is initialized* - Portlet `init()`; This method is called on abstract portlet[5]. The portal always instantiates only a single instance of the portlet. This instance is shared among all users the same way as a servlet is shared on an application server. Concrete

[5] The abstract `Portlet()` is used by the portlet container to invoke the portlet. Every portlet has to implement this abstract class, either by deriving directly from it, or by using one of the abstract portlet implementations.

portlet[6] is then initialized with other initialization method (`initConcrete()`); Concrete portlet is associated with `PortletSettings` object which contains configuration parameters.

- *User login*: When user starts the authentication process login method is invoked as the response to the user request (Portlet `login();`)
- *User page request* is processed with portlet's `service()` method.. This method is similar to servlet `service()` method and it is called when the portlet is required to render its content. During the lifecycle of the portlet this method is called many times which differentiates the portlet and servlet lifecycle.
- *Portal page return*: This is associated with the `PortalRequest` object. The following methods are involved: `login(), beginPage(), endPage() and service()` methods, and portlet markup is returned.
- *User logout*: When user indicates he/she wants to logout, the Portlet `logout();` is called.
- *Portal terminated*: This is the time when portlets are taken out of service and destroyed. Portlet `destroy();` and `destroyConcrete();` methods are called.

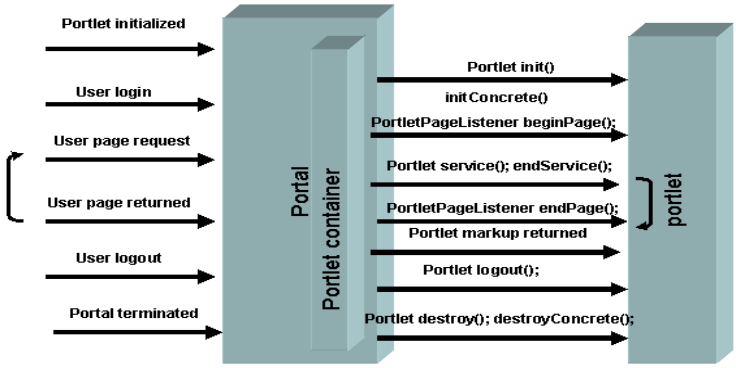

Figure 4 Portlet lifecycle

During its lifecycle, the portlet goes through some changes in its settings. When portlet is initialized and instance of the concrete portlet is created, portlet settings (`PortletSettings` object contains configuration parameters initially defined in the Web deployment descriptor – `portlet.xml`) are applied to the abstract portlet. This is done by the administrator when he/she uses their administrator console to deploy new application (fro example from the Web archive file `.war`) or when new portlet is loaded. Use of concrete portlets enables running of multiple configurations without the necessity to create new portlets. The same concrete portlet can be shared by multiple users.

[6] Concrete portlet is derived from the abstract portlet as an instance of the portlet.

When the concrete portlet is placed on the page, `ConcretePortletInstance` is created. The new instance is parameterized by the `PortletData` object. The `PortletData` object contains configuration data which can be manipulated via `Config` mode of the portlet (permissions to edit are required). The `PortletData` object stores persistent information for a portlet.

When user accesses the page, `UserPortletInstance` is created. User must login to the portals. The portal responds by creating `PortletSession` object for each of the user's portlets. The `ConcretePortletInstance` is now parameterized by `PortletSession` object resulting in user portlet instance. There can be many `UserPortletInstance` per one `ConcretePortletInstance`. Portlet applications group the related portlets (portlets with the same context) together. By context we mean all resources such as java classes, images, skins, etc. which belong to the same application.

Portlet modes

Portlet modes are properties of the portal presentation model. The portlet modes allow the portlet to display a different "face" depending on its usage. There are four modes (represented by icons in the portlet title bar:

View: The initial face of the portlet when created. It generates markup visualizing the portlet state and properties (`doViewOfgenericPortlet()`)

Help: It supports the help mode, a help page can be displayed for the user.

Edit: This mode produces markup to enable the user to configure the portlet for their personal use.

Configure: various vendors implementations may exist (Townsend, J. J. & Schaffer, D. R. (2004). If provided as in WebSphere portal, this mode displays a face that allows the portal administrator to configure the portlet for a group of users or a single user.

The portlet modes and window states are accessible from the portlet window title bar. Clicking on these icons can change portlet's mode. Portlet data is saved, retrieved, or deleted using the `PortletData` object. The `PortletData` object is linked to the request object (`PortletRequest`). Portlets can store values in the `PortletData` object only when the portlet is in edit mode. If the portlet is on a group page, then this information is available to all users of the portlet. The portlet retrieves a reference to an instance of `PortletData` using the `getData()` method from the `PortletRequest` object. The `store()` method saves the information in the `PortletData` object (only data of the Java `String` type are saved).

Portlet states

Portlet states determine how the portlet is displayed in the portal. The state of the portlet is stored in the `PortletWindow` object and can be queried for processing based on state. The three states of a portlet are:

Normal: The portlet is displayed in its initial state and size as defined when it was installed.

Maximized: The portlet view is maximized and takes over the entire body of the portal replacing all the other portal views. However, in some portals, maximized view only covers the original size of the portlet and there is no normal state.

Minimized: Only the portlet title bar is visible inside the portlet page.

Page aggregation

The aggregation process involves three basic steps:

- Collecting information about the user,
- Selecting the active applications, and
- Aggregating the output (creating the output stream and servlet).

The processing of request is different from request processing of a servlet does all processing in the `service()` method. In WebSphere, the processing is split into two separate phases: Event phase processing and content rendering. During Event processing, the portlet needs to listen to events. The Portlet APIs provide several listeners which extend the listening functionality of `ActionListener` or `WindowListener`. There can be only one action request and the target of this action can be only one portlet. If the user clicks on a link or button, similarly as with any Swing interface, `ActionEvent` object is created and passed to the `ActionListener`. This results in `actionPerformed`() event to be triggered on registered portlet followed by `doView()` – rendering method. All this activity happens before rendering phase. Remaining portlets on the page will process only rendering phase – `doView`() method. All portlets have to complete the exchange of messages (communicate) before the rendering phase starts.

Brief overview of core objects

There are four groups of core objects the portlets use: `PortletRequest`, `PortletResponse, PortletConfig` and `PortletSession.` In order to understand how a portlet lives and communicates with other portlets on a page you need to understand what are the basic building blocks which form a portlet and portal application.

`PortletRequest:` this is an object which is passed to the portlet through methods such as `login()`, `beginPage()`, `endPage()` and `service().` It contains the request specific data. This `PortletRequest` object enables access to the request data. Each request is associated with *attributes* (name – value pairs). *Parameter*s are also name - value pairs. They are sent by the client in the URI query string as part of a request. The parameters are often posted from a form. The portlet cannot set parameters from a request. Both, *attributes* and *parameters* are available only for the duration of this particular request

The `PortletRequest` object also contains the `Client` object. The `Client` object holds request-dependent information about the user agent (the actual device which initiated the request) or a specific request. Information from the `Client` includes the type of markup that the client supports. The `Client` is extracted using the `getClient()` method. The `Client` contains the **string** - sent by the user agent (user agent identification to the portlet), the **markup language** (for example, "wml"), the **MIME type** - supported by the client and **`Capability` object** - with more detailed information about the supported markup language.

The following objects, `PortletData` object, `PortletSettings` object, and `PortletSession` object, also arrive with the request. The information about either current or previous portlet mode is contained in `Portlet.Mode` attribute (from `PortletAPI tags`). For example, `mode="Edit"` is true if the user has placed the portlet in Edit mode. Another object, `PortletWindow` represents the state of current portlet window (MINIMIZED, MAXIMIZED, NORMAL). The `ModeModifier` can be used to change portlet mode before it is rendered.

`PortletResponse` object encapsulates the information to be returned from the server to the client. This content rendering phase is in principle very similar to servlet response processing. Output is generated using Java `PrintWriter` object.

The response object is passed via the `beginPage()`, `endPage()`, and `service()` methods. The response also includes methods for creating the `PortletURI` object or qualifying portlet markup with the portlet's namespace.

URI points to the portlet instance and can be further extended by adding portlet specific parameters and by attaching the actions. For example, portlet is in the ***`edit`*** mode, user is entering data and clicks the `Save button`. The portlet must process posted data before the next markup is generated. This can be arranged by adding the `Save` action to URI of the `Save button`.

`PortletSession` holds user specific data for the concrete portlet instance of the portlet, creating a portlet user instance. Concrete portlet instances differ from each other only by the data stored in their `PortletData`. Portlet user instances differ from each other only by the transient data stored in their `PortletSession`. Any persistent data must be stored using `PortletData`. Information stored in portlet's instance variables is shared between all concrete portlet instances and even between all concrete portlets[7] - with read and write access. Make sure you do not use instance attributes for user-specific data. On the other hand, you have to be cautious about what the portlet adds to the session, especially if the portlet ever runs in a cluster environment where the session is being serialized to a shared database. Everything being stored in the session must be serializable.

`PortletConfig` - provides non-concrete (abstract) portlet with its initial configuration. Information contained in the `PortletConfig` is shared by all concrete portlets based on the same abstract portlet. This object is typically used to access the initialization parameters set in the `web.xml` deployment descriptor of the servlet definition. These are read-only and cannot be dynamically changed.

`PortletData` holds data for the concrete portlet instance. The scope of `PortletData` object is determined by the type of page to which the portlet has been added:

- If the page is a group page, the `PortletData` object contains data for the group of user.
- If the page is a user page, the `PortletData object` contains data for this user.

`PortletSettings` object provides the concrete portlet with its dynamic data configuration. This object can be accessed with the `getPortletSettings`() method, available from the `PortletRequest` object.

`PortletApplicationSetting` object provides the concrete portlet application with its dynamic data configuration shared across all concrete portlets in the application

Portlet messaging overview

Portal server provides the environment for portlets to exchange messages and data. User can be presented with a portlet on a page which contains a list of choices e.g. a list of subjects offered in the semester. Selecting a subject causes another portlet on the same page to be updated with details of this selection (e.g. subject description retrieved from the online student handbook). This type of processing – propagating request through multiple portlets is achieved with portlet mes-

7 In other words, every new instance of the portlet will have access to this information.

saging and events. Portlets communicate using portlet actions and message listener. When the link is clicked, the action listener is called, which then sends a portlet message the receiving portlet.

- **Action events** are generated when a `HTTPRequest` is received by the portlet container that is associated with the action.
- **Message events** are generated when another portlet within the portlet application sends a message.
- **Window events** are generated when the user changes the state of the portlet window.

For inter-portlet communication, the portlet APIs provide a set of listeners to enable more functionality. The concept of listeners and specifically action listeners is the same as in Java language. The portlet container delivers all events to the respective event listeners (and therefore to the portlets) before generating any content to be returned to the portal page. Event conflicts such as when a listener while processing the event finds that another event needs to be generated are handled by the portlet container. The new event is queued by the portlet container and delivered at the time determined by the portlet container. There is no guarantee for event ordering. Container only guarantees that the event will be delivered and executed before the content generation phase.

Accessing remote systems from portals

Portal clients access portals via the HTTP protocol, either directly or through appropriate proxies or gateways like WAP gateways or voice gateways. To accommodate different devices, portals need to support different mark-up languages. During the aggregation phase, the portal invokes all portlets that belong to a user's page through the portlet APIs. Two different kinds of portlets are available:

- **Local portlets** that run on the portal server itself. They are deployed by installing *Portlet Archive* files on portal servers and are invoked by the portal server directly through local method calls.
- **Remote Portlets** run as Web services on remote servers. They are published as Web services in a UDDI directory to be easy to find and bind to. A remote portlet Web service is bound to a portlet proxy and registered in the portal's portlet registry. Portlet proxies are used to invoke portlets located on remote servers through a set of Remote Portlet Invocation APIs protocol which is based on based on SOAP.

IBM WebSphere Portal Server implements two important features to enhance option of inter-application cooperation: cooperative portlets (using the property broker programming model) and page-to-page communication.

Traditionally, Web services are data-oriented service. Data-oriented Web services do not contain any presentation or user interaction functionality. There must an application that provides required user interface using Web service data. Therefore, such application must know the Web service interface (typically WSDL description) to present data in specific format. On the other hand, WSRP services share basic interfaces with WSIA (Web Services for Interactive Applications) and define well-defined contracts and interfaces on top of the generic ones. The fact that the WSRP is intended for use with WSIA only imposes significant limitation on adoption of this technology in industry.

Universal Description, Discovery, Integration (UDDI)

UDDI is a registry and directory that attempts to support the business-to-business electronic co-operation over the Internet. Its programmable interface provides an industry-wide approach that allows companies to advertise both the business and technical aspects of their services. All information about UDDI and its usage can be found on uddi.com. UDDI defines an interoperability stack enabling publishing as well as search for Web services. UDDI relies on standards-based technologies such as TCP/IP, HTTP, XML and SOAP to create a uniform service.

White pages: White pages provide basic contact information (the business name, address and other contact details). White pages also provide unique business identifiers that are unique nine-digit sequences widely recognized for keeping track of businesses, such as Dun & Bradstreet's D-U-N-S numbers,. Standard business identification allows customers and partners to discover business services.

Yellow pages: Yellow pages describe a business service using various taxonomies. Through these taxonomies customers can discover business services (manufacturing or retail business).

Green pages: Green pages provide technical information about the behaviours and functionality of a business service. Green pages are not limited to describing XML-based Web services, but any business service type offered by a business entity. This may be services such as call centres, technical support for a product, fax-based services such as a fax to E-mail service, etc.

The scope of this paper does not allow us to provide detail explanation of the UDDI programming interface. With regard to the context of this tutorial, please note that the WSDL description of the service published in UDDI must contain the following six elements: `definitions, types, message, portType, binding and service`.

Conclusion - Technology Aware Managers

After looking at a small portion of J2EE and portals there is still an unanswered question: *Who is the winner? CORBA or J2EE, Web services and portals?*

Technology managers come in two varieties: those whose teams produce (or develop computer products) and those whose teams consume (or “use” the computer products). The tutorial’s aim is to contribute to the understanding of modern computer technologies (web, portal) from the point of view of both, the producers and consumers. The aspects of interest common to both groups include:

- The costs of development in terms of human resources and technology. When addressing this aspect, we need to take into account the technology and methods of development, using the technology
- The human experience. The human – machine interface needs to be understood in relation to the described technology
- Effectiveness of technology in terms of costs of development and maintenance per unit of time used by the consumers
- Future trends and expectations of both the developers and consumers.

The technology aware manager has her eyes on significant opportunities brought about by *service oriented architecture* This idea is based, but not solely, on web services which are available to all players – legacy assets as well as modern web applications. Technologies supporting Web services, such as Simple Object Access Protocol (SOAP), Web Services Description Language

(WSDL) and Universal Description, Discover and Integration (UDDI) allow building specific integration processes.

References

Barry, D. K. (2003). Service-oriented architecture (SOA) definition. Barry & Associates, Inc. http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html

Berners-Lee, T. (1999). Weaving the Web: *The original design and ultimate destiny of the world wide web by its inventor*. San Francisco: Harper.

Berners-Lee, T., Fielding, R., & Masinter, L. (1998). RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax. Network Working Group. http://www.ietf.org/rfc/rfc2396.txt

Christensen, E., Meredith, G., Curbera, F., & Weerawarana, S. (2001). Web Services Description Language (WSDL) 1.1. W3C. http://www.w3.org/TR/2001/NOTE-wsdl-20010315

Coward, D. (2003). JSR-000154 JavaTM Servlet 2.4 Specification (Final Release). Sun Microsystems Inc. http://www.jcp.org/aboutJava/communityprocess/final/jsr154/

Inderjeet Singh, S. B., Murray, G., Ramachandran, V., Violleau, T. & Stearns, B. (2004). Designing Web Services with the J2EE(TM) 1.4 Platform : JAX-RPC, SOAP, and XML Technologies. Addison-Wesley Professional.

The Java Web Services Tutorial. (2004). Sun Microsystems Inc. Retrieved from http://java.sun.com/webservices/tutorial.html

J2EE 1.4 Java API specification. (2004). Retrieved from http://java.sun.com/j2ee/1.4/docs/api/index.html

J2EE 1.4 Tutorial Update 4. (2004). Sun Microsystems Inc. Retrieved from http://java.sun.com/j2ee/1.4/download.html#tutorial

Orfali, R. & Harkey, D. (1998). Client/Server Programming with Java and CORBA, John Wiley & Sons.

Portlet Specification JSR 168 (2005). Retrieved from http://www.jcp.org/aboutJava/communityprocess/final/jsr168/

Roth, M. (2003). JavaServer Pages™ Specification. Sun Microsystems. http://jcp.org/aboutJava/communityprocess/final/jsr152/

Servlets Specification. 2.4 (2004). http://www.jcp.org/aboutJava/communityprocess/final/jsr154

Townsend, J. J. & Schaffer, D. R. (2004). *Building portals, intranets, and corporate web sites using Microsoft servers*. Addison-Wesley Professional.

Web Services Description Language (WSDL): An intuitive view. (n.d.). developers.sun.com. http://java.sun.com/dev/evangcentral/totallytech/wsdl.html

WebSphere Portal Fundamentals. (2004). International Business Machines Corporation (IBM) International Technical Support Organization. Retrieved from http://www-106.ibm.com/developerworks/websphere/registered/tutorials/dl/sw741/htmls/WebSphereportalFundamentals/

Biographies

Jana Polgar is a lecturer at Monash University. Her recent research interests include QoS in Grid applications and semantic grid issues. She holds PhD from RMIT (Melbourne).

Robert Bram is currently researching a PhD with Grid technology and multimedia. He also works with Portal technology and WebSphere. He lives in Australia and enjoys working with anything Java related.

Tony Polgar is a portal development manager leading the team of portal developers for Coles Myer Ltd. in Melbourne, Australia. He holds master degree in computing from Monash University.