# BUILDING AND MANAGING Enterprise-Wide Portals

Jana Polgar
Robert Mark Bram
Anton Polgar

# Building and Managing Enterprise-Wide Portals

Jana Polgar
Monash University in Melbourne, Australia

Robert Mark Bram
Monash University in Melbourne, Australia

Anton Polgar
Monash University in Melbourne, Australia




**IDEA GROUP PUBLISHING**
Hershey • London • Melbourne • Singapore

Acquisitions Editor: Michelle Potter
Development Editor: Kristin Roth
Senior Managing Editor: Amanda Appicello
Managing Editor: Jennifer Neidig
Copy Editor: Kim Barger
Typesetter: Diane Huskinson
Cover Design: Lisa Tosheff
Printed at: Yurchak Printing Inc.

Published in the United States of America by
Idea Group Publishing (an imprint of Idea Group Inc.)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@idea-group.com
Web site: http://www.idea-group.com

and in the United Kingdom by
Idea Group Publishing (an imprint of Idea Group Inc.)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 0609
Web site: http://www.eurospanonline.com

Copyright © 2006 by Idea Group Inc. All rights reserved. No part of this book may be reproduced, stored or distributed in any form or by any means, electronic or mechanical, including photocopying, without written permission from the publisher.

Product or company names used in this book are for identification purposes only. Inclusion of the names of the products or companies does not indicate a claim of ownership by IGI of the trademark or registered trademark.

Library of Congress Cataloging-in-Publication Data

Polgar, Jana, 1945-
Building and managing enterprise-wide portals / Jana Polgar, Robert Mark Bram, and Anton Polgar.
p. cm.
Includes bibliographical references and index.
Summary: "This book discusses the technology behind portals and grid networks, including topics such as management issues, deployment considerations, facilities and tools for creating and managing grid and portal applications"--Provided by publisher.
ISBN 1-59140-661-7 (hbk.) -- ISBN 1-59140-662-5 (pbk.) -- ISBN 1-59140-663-3 (ebook)
1. Web portals. 2. Web services. 3. Computational grids (Computer systems) I. Bram, Robert Mark, 1974- II. Polgar, Anton, 1946- III. Title.
TK5105.88813.P65 2005
025.04--dc22
2005023875

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

All work contributed to this book is new, previously-unpublished material. The views expressed in this book are those of the authors, but not necessarily of the publisher.

# Building and Managing Enterprise-Wide Portals

## Table of Contents

**Preface** ..... **vii**

**Section I: The Toolset**

**Chapter I. Introduction** ..... **1**
- The Idea of a Portal ..... 1
- Technology Aware Managers ..... 3
- Service-Oriented Architecture ..... 4

**Chapter II. The Foundations of XML and WSDL** ..... **7**
- What is XML? ..... 8
- XML Document Processing and Development ..... 10
- XML Authoring and Java Parsers ..... 15
- Web Services Description Language: WSDL ..... 22
- Document Object Model ..... 28

**Chapter III. Protocols** ..... **32**
- UDDI (Universal Description, Discovery, and Integration) ..... 32
- Simple Object Access Protocol (SOAP) ..... 48
- What is URI, URL, URN? ..... 52

## Section II: Web Services as Shared Resources

**Chapter IV. Web Services ..... 55**
Multi-Tier Component-Based Architecture ..... 56
Web Service Architecture ..... 58
Web Services with JAXB ..... 59
Web Services with JAX-RPC ..... 63
Web Service Discovery and Invocation ..... 72
XML Documents as Part of Web Services ..... 76
XML-Based Web Service Endpoint Considerations ..... 79

**Chapter V. Servlets ..... 82**
The Servlet Application Architecture ..... 83

**Chapter VI. Java Server Pages (JSP) ..... 94**
JSP Development Notes ..... 96
Integrating Servlets and JSP ..... 98
Web Module Deployment ..... 100

**Chapter VII. Web Service Design Issues ..... 104**
Service Endpoint Design ..... 105
Web Service Client Design ..... 110
Available Web Application Frameworks ..... 115

## Section III: Putting Portals on the Web

**Chapter VIII. Portals ..... 119**
Historical Background ..... 121
What Do Portal Services Deliver ..... 127

**Chapter IX. Portal Development Framework ..... 134**
MVC Model and Portlets ..... 136
Overview of Portal Concepts ..... 138
Portlet Lifecycle ..... 145
Environment for Portals ..... 162
Portal Deployment ..... 164
Performance and Implementation ..... 167

**Chapter X. Accessing Remote Locations ..... 173**
Cooperative Portlets ..... 177
Page-to-Page Communication Scenarios ..... 182

**Chapter XI. Portlet Authentication and SSO ..... 186**
Authentication Process ..... 186
Single Sign On (SSO) ..... 190

**Chapter XII. Struts Framework ..... 192**
Struts in Web Service Endpoint ..... 192
Struts in Portal Development ..... 195

**Chapter XIII. Writing Portlets: A Quick Lesson for IT Managers ..... 196**

**Chapter XIV. Brief Overview Portal Application Platforms ..... 206**
WebSphere Portal for Multiplatforms 5 ..... 206
Apache Portal Tools and Server ..... 207
Sun ONE Portal Server ..... 208
Portal Development with Struts ..... 211

**Chapter XV. Installation and Maintenance Costs ..... 216**

**Chapter XVI. Standards Overview ..... 219**
Portlet Specification JSR 168 ..... 220
Portlet Relationship with Servlet Specification ..... 222
Web Services for Remote Portlets (WSRP) ..... 226

**Chapter XVII. Managing Portals ..... 237**
FAQ ..... 237
Future Trends ..... 244

**Section IV: Grids as Virtual Organizations**

**Chapter XVIII. Grids ..... 247**
Grid Goals ..... 247
Grid Resources ..... 251
Grid Architectures: Physical and Logical ..... 254
What is a Grid Network? ..... 255

OGSA: Web Services as a Grid Platform ..... 261
OGSA, OGSI, and WSRF: From GT3 to GT4 ..... 268
Building Grid Solutions ..... 301
Feasibility ..... 306

**References ..... 309**

**Glossary ..... 317**

**Acronyms ..... 324**

**About the Authors ..... 328**

**Index ..... 330**

# Preface

An enormous amount of material dealing with the Web and Web Services is available in books, online, and in various publications. Anybody interested in learning can find the material but suffers from the overload of information. Which information is important, up-to-date, and correct? How can I find information that constitutes the background for the problem facing me? Is the information relevant, up to date, and reliable? Can I simply apply what the published references recommended?

Inspired by similar questions, we decided to put together a systematic review of current Web technology and trends to meet two goals:

1. Provide information to people who need general, rather than in depth, technical knowledge such as information technology (IT) development mangers, software designers, architects, IT students, and project and program managers. Our ideal reader is technically inclined, with broad interests and management responsibilities.
2. Describe the logical development of business applications technology, from client-server to Web, Web Services, Portal, and computer Grids.

The book is practically oriented; it gives a large amount of industry focused advice and down-to-earth observations. The reader will probably need to seek more detailed information in specialized books, but he or she will be able to maneuver in the sea of available information.

It is not necessary to read the book sequentially, but there are advantages in doing so as the topics build on previously discussed topics.

Most importantly, acronyms and glossary terms are explained at the end of the text. Terms that we consider important are listed in the Index.

The reader should be familiar with the basics of current information technology, the fundamentals of Web architecture, and introductory Java. It is possible to understand the ideas without understanding the Java examples that appear in the text. We rely heavily on the XML concepts, and we envisage that the reader will consult the Web for additional information when reading this book.

We found many definitions and principles on the Web, and, where possible, we quote the source for the reader. Published Request for Information (RFC) material is used extensively throughout the book.

Some information will become obsolete with time, as many suggestions and design principles refer to the current industrial strength implementations. Many examples and practical observations refer to IBM WebSphere as the authors have experience with the product. We encourage readers to critically evaluate some of our more controversial advice.

## Promises and Expectations of Client-Server Computing

The promise of reduced complexity of software and of software development in the third and fourth generation of computers has simply not eventuated. The catch phrase of the client-server paradigm was the promise to reduce the complexity of the software by dividing electronic processing into a smaller portion performed on a client computer and a larger portion performed on a server. This way, the client (the computer, not the person) specializes in front end processing, leaving the logic to the backend server processing. This separation of duties was supposed to achieve what we call today the separation of concerns. The client interacted with the user and provided data validation, while the server processed the data and returned the result, often by connecting to a database. The idea of the Graphical User Interface (GUI) was born, and it was firmly allocated to the client. Nearly immediately, the user interface became much more sophisticated, filling the vacated space with GUI API's, event processing, and user interface standards. Thus, GUI processing grew into a specialized and complex area, demanding high skills and development costs.

Programming languages such as Visual Basic were able to produce functionally rich user interfaces with graphics, which the user more easily interpreted.

Previously unseen visual objects, such as tabs, pop up windows, buttons with graphical depiction or symbols of their function, and colorful navigational prompts filled the user's view.

On the server side, additional functionality also filled the void: generalized database connectors; lightweight processes called threads (which introduced additional complexity for synchronization); support for a large variety of devices and protocols; and ever-increasingly complex logic.

The development process for client-server applications did not become simpler either. The two subsystems were often developed in two or more different programming environments. 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 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 to resolve the problems.

The initial promise of reduced complexity was broken. The development managers were dealing with increased costs and were looking for a solution that would address the increased number of development personnel, ballooning 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. At the same time, the end users became accustomed to better usability and demanded ever-improving quality of the user interface, intuitive behavior, and sophisticated error management, where it should not matter whether the validation errors originated from the server or client processing.

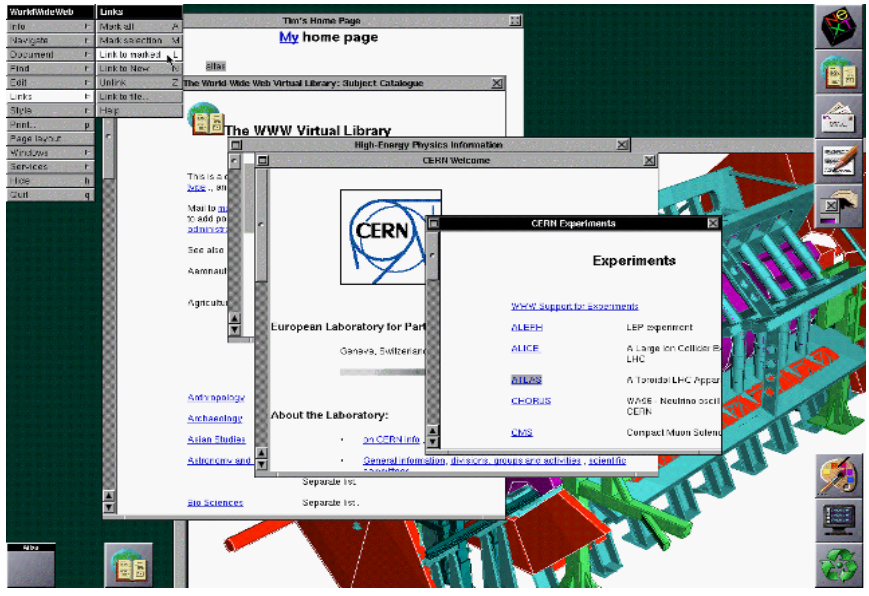
## The World of the Web

While client-server computing was evolving, the world of the Web was quietly assessing its strength. This occurred in 1989-1990. An interested reader can see one of the early Web sites on http://www.w3.org/History.html or http://www.w3.org/People/Berners-Lee/WorldWideWeb.html, or read about it in Weaving the Web by Tim Berners-Lee with Mark Fischetti (Berners-Lee & Fischetti, 1999).

Let us look at one of the first browsers, shown in Figure 1. It bears an unmistakable influence of what we call today a "thick" client — rich graphics, creative use of space, various visual clues, and many windows. The hyperlinks are the new elements, previously not used. Ted Nelson first coined the term hyperlink in 1965 in project Xanadu (Nelson, 1965)! Looking at the visual components, which became very common later, we notice that the Navigate menu (third from the top on the left side) has "back", "next", and "previous" options. The Style menu option (halfway down the first menu list) has a style sheet option where one could load different style sheets. The "X" box to close the window was copied from the NeXT toolbox and was later universally adopted by Windows. The "edit" option was also new. The browser was written in a dialect of C, called Objective-C. Note that the "Help" menu is also present as the last option on the second menu from the left. Many of these components are re-used in the current Web and Portal interfaces.

Very shortly afterward, the style of the interface became more restrained, less graphical, with limited expressiveness. This was caused by the amount of information that had to be transferred through the relatively slow communication lines. It is important to remember that by then, client-server computing had undergone a substantial change in that the connections between the Web client and the Web server were much slower than point-to-point, dedicated client-server connections.

*Figure 1. Berners-Lee's screen shot of the browser (1994) (http://www.w3.org/History/1994/WWW/Journals/CACM/screensnap2_24c.gif)*



The new Web technology brought some simplification, but only for a brief period. 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[1] (Common Gateway Interface) script. Since the plain HTML/CGI combination provided services that were regarded as a step backwards (compared to more sophisticated thick clients), a method was found which brought the browser-based client closer to a thick client. Thus, client-side processing was introduced, constituting a full circle in the development. Java applets, Java scripts, and various other means were used to achieve functionally rich, dynamic user interfaces.

From the point of view of a development manager, the situation was not much better than before. A variety of skills were necessary in a development team, including a GUI designer, an HTML developer, a CGI script developer, and later a Java developer. However, a substantial gain was achieved — the whole system could be developed on a single machine, with a local Web server.

With the introduction of Java (and later .Net technology), 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. It looked as if the technology was finally making life simpler, although the quality of the interface was still not as good as in the thick client-server processing. The net effect of the latest developments was that the enterprise's client-server systems were being re-implemented in the "Intranet" — applications designed for the Internet to be only accessible within the company network. Legacy systems continued to be used and maintained, and people wanted to use them in the "net" — here meaning Intranet — without the necessity to re-write them. A lightweight Web application would do precisely that — provide access to a backend system. This represented 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.

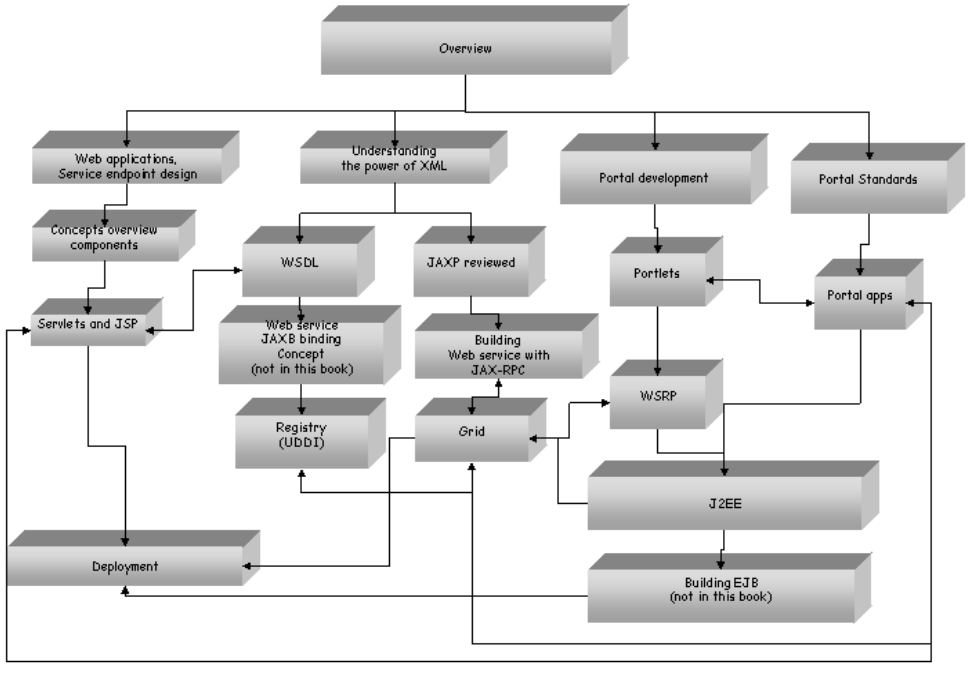
The necessity to provide integrated Web Services stems from the fact that modern businesses often evolve through mergers and acquisitions. These businesses have applications that are either redundant or reusable. However, integration of these applications is not trivial, mainly from the point of view of the number of connections. Consider n applications that are to be integrated; it will be necessary to build n*(n-1) interfaces. With every new application (n+1), one must test and maintain 2n (where n is the number of original applications) new interfaces. Obviously, the architect tries to find a solution where a new application requires only one new interface to the system. However, there is more to it; the integration must be done not only on the level of interconnectivity,

but also on the user interface level. For example, two business applications that are related should be accessible from one screen, if possible. If there is some legal information or other textual information that is useful, such as business procedures, it should be also readily available. The industry answer to the previously-mentioned problem is the service-oriented architecture, which is described in the section called Service-Oriented Architecture. We will explain how the necessity of simplifying the development process contributed to the idea of Portal, Web Services, and computer Grids.

## Where to Start?

In Figure 2, we list the major subjects discussed in the book and their relationships. The reader can see the necessary prerequisites for the desired subject. The diagram shows that modern Web-oriented computing is reaching a stage where many components cooperate in the final product. It is important to realize this idea when designing, building, and managing the development of these systems. Understanding historical trends helps to keep current and future trends in perspective. We encourage the reader to look back before looking to today and onto the future.

*Figure 2. Roadmap of this book*



This book arose from university courses taught by the authors at Monash University in Melbourne, Australia and from the practical managerial experience of the portal development manager. Most courses and textbooks in this field are targeted at either the low programming level of APIs for portals, J2EE components and Web Services, or high-level aspects of services and portal deployment in the enterprise information systems. We also realized that the majority of the textbooks focus on material around Web Services and ignore the fact that portals and Grid applications are in some ways the inevitable extension of Web Services.

This book uses Java, J2EE platform, Globus Toolkit, and IBM's WebSphere for Multiplatforms 5 to demonstrate the programming and deployment aspects of development of Web Services, portals, and Grid applications.

## Organization of the Book

At this point, the reader should be ready for the main body of the book, which is divided into four parts:

1. **In Section I:** The Toolset (Chapters I-III), we explain technicalities related to XML and Web Services Description Language (WSDL). The purpose of Chapter II is to give the reader basic knowledge necessary to read the rest of the chapters.

   Chapter III deals with Web oriented protocols, UDDI (Universal Description, Discovery and Integration), and SOAP (Simple Object Access Protocol). We explain relationship between UDDI version 2 and WSDL. In addition, we provide a simplified example of accessing UDDI Registry using JAXR (Java APIs for Registries). SOAP is described in terms of messaging — a widely used technology in Web Services.
2. **In Section II:** Web Services as Shared Resources (Chapters IV-VII), we describe Web Services and look at some implementation issues. Chapters V and VI discuss servlets and JSP (Java Server Pages). At this point, the reader will have the necessary knowledge for understanding the Web service design issues discussed in Chapter VII.
3. **In Section III:** Putting Portals on the Web (Chapters VIII-XVII), we utilize the knowledge of Web Services and develop the concepts of portals. We briefly touch on portal solutions for business processes, and then continue with portal development framework and the close relationship between portlets and servlets. Large portions of Chapters IX-

XIII are devoted to understanding portlets and portal concepts, their lifecycle, and messaging. Chapter XVI is dedicated to discussion about relevant portlet standards and issues related to accessing presentation-oriented Web Services. Chapter XVII is based on practicing manager's experience with installation and delivering portal solutions in large organizations. These issues are particularly relevant to the B2E and B2B solutions.

4. **In Section IV:** Grids as Virtual Organizations (Chapter XVIII), we introduce the concept of computer Grids, building on the previous two parts. As recently discussed in various Web communities, Web Services are seen as an enabling technology for Adaptive Enterprises where business and IT are synchronized to accommodate fast and efficient changes in real time. With the new incoming WS-Resource framework (authored by The Globus Alliance, HP and IBM), we feel that thorough understanding of Grid concepts would be beneficial to our readers. This seemingly long section will provide the reader with sufficient knowledge to allow him or her to appreciate new trends in creating Grid-based communities and applications running on the Grids.

## Endnote

[1] The Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers, such as HTTP or Web servers. See also the following Web site for more information: http://hoohoo.ncsa.uiuc.edu/cgi/intro.html

# Acknowledgments

*Our thanks go to Professor Elizabeth Kendall, who has been so supportive and encouraging throughout our research and who kindly revised the manuscript. This book benefited from her experience in many ways.*

# Section I:

# The Toolset

## Chapter I

# Introduction

## The Idea of a Portal

To create an initial impression of what a *portal* is, we can compare a portal page to the colonial window pictured on Figure 3. This window can represent a Web page, with several smaller windowpanes. We can now imagine that the user is looking at a different application through each windowpane. To extend this metaphor, we can imagine that the application behind each pane belongs to the builder and that each member of the household uses his or her personalized window, with only some panes visible to him or her. The *portal* is the frame that holds the window together (indeed, we will see later that it is a container); the window itself is the Web page, and the windowpanes are called portlets.

The look and feel of a *portal* has been used by media in TV and elsewhere for some time. TV programs, specifically sports and news transmissions, often use 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 is 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's needs. This leads to the requirement of configuration for the Portal so that the user can choose which windows to view on the screen. If the user needs to be able choose what to see, then the system must provide security protection so that the user is authorized to access everything that has been configured, with a single sign-on.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 3. Colonial window as a metaphor for portal*



Three very important aspects of portals have been developed:

1. Ability to display more than one Web-like window on a page;
2. Ability to configure the page so that only windows selected by the user (or administrator) are displayed on the page; and
3. Provision of a single sign-on so that the user perceives the system as homogenous.

The requirements for multiple windows led to the development of the concept of a *portal* as a container. This container holds portlets, which produce one or more windows. The other two requirements — configuration and single sign-on — caused the development of infrastructure to support these and other requirements.

From the technical perspective, a portal is a *portlet container* that is built on top of the servlet container, reusing functions.

Let us look at portal technology from the point of view of the development manager. To develop a portal "window", the developer must produce a portlet. This does not necessarily mean that each portlet can have only one view port, or what we previously called a "window" (a windowpane in our initial example) within a browser window. However, two consequences of this approach emerged.

The portlet produces only a small part of the overall window, called a *fragment*. However, the fragment behaves very much like an independent window. The fragment consists of HTML code, displayed in the portal Web page.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The fragment cannot be addressed by a URL[1] (see http://www.w3.org/Addressing/); only the whole page can be addressable by a URL.

Obviously, to produce a portal and to meet the previously-mentioned requirements of multiple portlets, security, and configuration, the technology must provide new tools and concepts. This book describes some aspects of the technology. However, the main purpose is to provide a technical explanation of the consequences of the new requirements.

The development process of the portal is in many respects similar to the development of Web material. The windows must be designed to meet the users' requirements, and the programming techniques are similar to Web programming. However, the total cost is lower with portals, as the supporting technology reduces the development effort with regard to security and assists in page navigation.

The main contributor to cost reduction is the fact that a lot of the functionality that had to be repeatedly programmed is now provided by development kits, in particular security and configurability. The development can be performed on one platform, and the developed artifacts can run on another platform without change. However, one of the basic problems of the Web environment – the necessity to keep the client pages lightweight because of the limited bandwidth of the communication channel to the server — has not been solved. However, with improved communication facilities, this disadvantage becomes less and less important. Consequently, the look and feel of the portlet fragments is similar to current Web pages.

## Technology Aware Managers

Computer 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 book's aim is to contribute to the understanding of three modern computer technologies (Web, Portal, and Grid) from the point of view of both the producers and consumers. The aspects of interest common to both groups are addressed in the book, and they include:

- The costs of development in terms of human resources and technology. When addressing this aspect, we explain the main features of the technology and the methods of development that use the technology.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- The user experience. The human-machine interface will be explained in relation to the described technology.
- The effectiveness of the technology in terms of costs of development and maintenance.
- The future trends and expectations for both the developers and consumers.

## Service-Oriented Architecture

Referring to Kishore Channabasavaiah and Kerrie Holley (2004), we describe service-oriented architecture (SOA) as a set of principles and building blocks that utilizes the idea that independent Web Services applications can be re-used and connected by all participants with a minimum integration effort. The internal interfaces can be built and achieved by current Web Services, and user interface integration can be achieved by current portal technologies. With this approach, it is possible to make the next step in building loosely coupled collections of applications that perform seamlessly in an ever-changing business environment.

Imagine an organization that has been around for 10 or more years and has had an active IT department that developed a number of applications in a number of technological platforms. Let us define these applications as services and make them safely available on the Intranet or Internet. Each of these applications or groups of applications operates in its own environment, often on different platforms, using different and incompatible programming systems, databases, and protocols. How can we integrate these applications to interoperate with each other and with any additional, new applications built in the future?

If we defined all functions as independent services and provided an invokable interface, we have made a big step toward our goal. The invokable service can be coupled to the application, or it can be totally separated, built for the sole purpose of invoking the function as a Web service. With this technique, our legacy function became an independent, invokable, and manageable service.

The management of the service includes:

- Security of invocation and data transmission,
- Deployment of the service,

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- Load balancing and fail-over capabilities, and
- Version management.

The conversation with the service typically takes place in a source-neutral way, such as via XML.

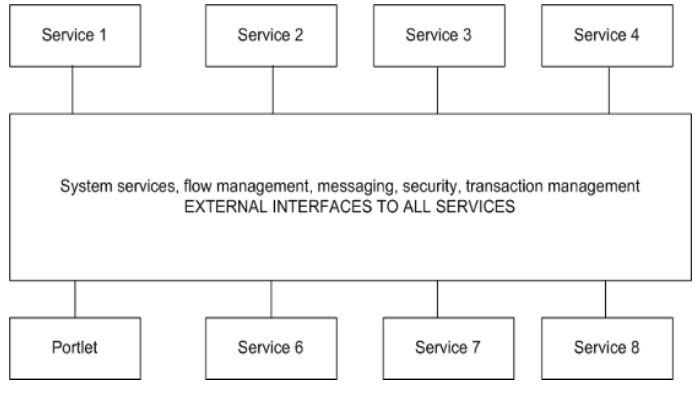
Figure 4 shows the idea of SOA in a simplified way. The framework allows any service to talk to any other service. Note that a portlet can also talk to any service, an idea described in more detail later in the book when we talk about end user connectivity and the use of the portal as an integrating device.

Part of the SOA is information integration. It provides consistent access to all data in the enterprise, by all applications that need it, with no restrictions. The location, source, format, and method of accessing the data are very transparent to the application because the data are provided by a common service. The use of portlets provides a good way of making the data available to the end user.

Implementation of SOA is possible with existing investment and infrastructure. It involves a change in the conceptual thinking and the use of existing Web Services, portal services and Grid computing, all of which are discussed in this book.

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 as a means to on-demand deployment. SOA helps maximize resources in the organization.

*Figure 4. Service-oriented architecture overview*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Endnote

[1] URL is a Universal Resource Locator, a unique name of a Web page. URL belongs to a broader group of URI — Universal Resource Identifiers, which identify resources in the Web: documents, images, downloadable files, services, electronic mailboxes, and others.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter II

# The Foundations of XML and WSDL

XML stands for Extensible Markup Language (http://www.w3.org/XML/), and it has been adopted by industry for exchanging data in a platform, language, and protocol independent fashion. While XML has many benefits during the development stage, it has some performance disadvantages. This chapter provides a quick look at the following topics:

1. Overview of the standard and basic concepts;
2. Basic XML document structure;
3. Information about usage of Document Type Definition (DTD);
4. Structure and usage of XML Schema; and
5. Discussion about the design and performance issues when using XML documents with Web service.

XML plays an important role in the implementation of Web Services. Because XML is not in binary notation, it can be created and edited in any text editor (provided the character set complies with the Unicode[1] in which XML documents are encoded). XML is also useful for storing small amounts of data. XML and stylesheet standards allow developers to dictate how the information will be displayed.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Where can XML be used? The following list states several options relevant to this book:

- XML can contain data for a program to process with traditional data processing. XML is the answer for the exchange of all sorts of transactions as long as both the client and the provider agree on the XML markup.
- XML documents can be used as template containers that help build interfaces and applications from existing components. This approach is called document-driven programming.
- XML, in combination with DTD or one of the schema standards, defines the structure of documents. This approach allows code to be easily generated automatically. During this process, known as binding, classes are created according to the given schema specifications. Input data arriving in the XML document can be processed, and the response document is created also in XML.

For those readers who are familiar with XML concepts, the following section will not reveal anything new. We have included this section in order to provide consistent technical background to other chapters of this book, in particular the discussions relevant to Web Services, portals, and deployment issues.

## What is XML?

XML is a markup language that enables hierarchical data content to be represented as a marked-up text document. The language uses tags to mark pieces of datum. Each tag assigns meaning to the data associated with it, thus transforming the data into meaningful information. XML aims to separate data from processing and presentation.

Although XML was not meant to be read by end users, the XML specification states that "XML documents should be human-legible and reasonably clear." You can also view XML as a meta-language with appropriate grammar and vocabulary (called schema), which can be used to define other markup languages.

We start our quick tour with the essential background and properties of XML:

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

1. An XML document must be well-formed in order to be successfully processed or parsed. Well-formed means that the document has exactly one root element; each tag has a start and a proper end; all attribute values are in quotes; and elements do not overlap (however, they can be nested).
2. Each document can be associated with a DTD syntax, which is part of the core XML specification. This allows for the definition of domain-specific schemas and makes XML extensible. Furthermore, the capability to reference DTD or other schema provides the framework for document driven programming and supports the binding process.
3. A document is valid if it is well-formed, refers to, and conforms to an accessible DTD, and contains the declaration <!DOCTYPE> as shown in the following example:

```
<!DOCTYPE internet_lectures SYSTEM "Course01.dtd">
```

In 2001, the W3C defined the XML Schema Definition language (xsd). The XML schema is a document used to describe the structure and semantics of an XML document. There is an interesting perspective on XML schema in Singh, Brydon, Murrary, Ramachandran, Violleau, and Stearns (2004). The authors divide the schema into vertical and horizontal domains as a broad taxonomy of XML representations.

- *Horizontal domains* are cross-industry domains that address common situations in a particular domain. Examples are Mathematical Markup Language (MathML), Synchronized Multimedia Integration Language (SMIL), and Resource Description Framework (RDF).
- *Vertical domains* are specific to certain types of industries. There are XML schemas for e-commerce such as Electronic Business with XML (ebXML), Commerce XML (CXML), Common Business Language (CBL), and Universal Business Language (UBL).

Developers in large enterprise organizations often define their own custom schemas. These schemas can be proprietary or shared with business partners to enable the exchange of data between enterprise applications. The organiza-

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

tion may decide to expose their private schema to enable potential business partners to use it.

## XML Document Processing and Development

Online XML document processing consists of the following steps:

1. Parse and validate the incoming document against the schema to which the document should conform;
2. Transform the document from its conforming schema to the internally supported schema;
3. Disassemble a document;
4. Search the incoming document for relevant information (relevant information either contains the embedded tags or it is found in expected locations);
5. Extract the information; and
6. Map the information to domain-specific objects.

Developers who implement the document-driven applications where business logic manipulates the documents typically have trouble separating document handling from business logic. This makes document processing complex, as it will become apparent in later chapters.

**Core XML tags:** Although many books exist about XML, the authors wish to provide a brief revision of basic syntactic rules. The XML document structure should follow the following outline:

- Documents are made up of a prolog and a body. The document prolog contains the XML Declaration, and the document body contains the actual marked up document. According to the official XML specifications, every well-formed XML document must contain a prolog.
- The prolog must appear **before** the root element (first element in the document). The prolog consists of two parts: the XML declaration and the Document Type Declaration (DTD). Depending on your needs, you can choose to include both, or either of these items in your XML document. The DTD is most often used.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The example shown in Figure 5 shows the basic structure of an XML document.

Note that in this example, all tags are closed, so the document is well-formed. The "!DOCTYPE internet_lectures" indicates that all the tags inside the XML file will have the root called "internet_lectures." Each XML file can have only one !DOCTYPE. The XML fragment in Figure defines several elements: <lectures>, <lecture>, <title>, and <item>.

Each lecture has a title and one or more items (item can also be empty). A namespace is a collection of names, identified by a URI reference (*xmlns* tag in Figure 5). Namespace is used to define the context within which the names of elements are unique. For example, the students' material for the Web subjects in the undergraduate course is in the namespace:

<internet_lectures

xmlns="http://walkabout.netcomp.monash.edu.au"

*Figure 5. Example of the basic structure of an XML document (internet_lectures)*

```
<?xml version='1.0' encoding="ISO-8859-1" standalone="yes" ?>
<!DOCTYPE internet_lectures SYSTEM "Course01.dtd">
<internet_lectures
   xmlns="http://walkabout.netcomp.monash.edu.au" >
  <lectures title="Sample lectures in CPE5014"
     date="today" author="Jana Rob Tony">
  <!-- This is example of lecture organisation -->

            <lecture type="all">
                        <title content="Introduction to XML"
subtitle="List of topics">Lecture 1: XML parsers</title>
            </lecture>

<!—Parsing XML lecture overview -->
-
            <lecture type="all">
                  <title content="Parsing XML" subtitle="List of topics">Lecture
2: In-troduction to SAX parser</title>
                    <item>Topic one SAX parser introduced

                    </item>
                    <item>
                          Basic approach to non-validating parser
                    </item>
              </lecture>
          </lectures>
</internet_lectures>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

This namespace enables avoidance of the name clashes with the XML schema names (tags) used in other subjects.

## Using Data Type Definition (DTD)

**Creating a DTD:** As mentioned previously, a DTD is a structured collection of information that defines the structure and characteristics of an XML document. It contains required tags, relationships among tags, valid attribute values, and named entities. They are needed for checking the structural correctness of the XML document. The example in Figure 5 shows an excerpt of an XML document that uses an external DTD file called "course01.dtd" for simple lecture presentations.

Development of a DTD can be complicated. DTDs are stored as ASCII text with the extension .dtd. To make the development of XML documents easier, there are published DTDs that define tags for commonly used elements. An example DTD is in Figure 6.

The DTD tag ELEMENT (<!ELEMENT internet_lectures (lectures+)) specifies the name of the element and one or more items (sign +) indicating valid content for this element. The DTD tag !ATTLIST begins the series of attribute definitions. Each attribute is defined by a series of three space-separated values. Commas and other separators are not allowed. The first value is the name of the attribute; the second value indicates the type of data (CDATA means unparsed character data). The last entry specifies the default value and if it is required value.

## Other W3C Standards Worth Noting

There are a few more specifications that should be noted: The Extensible Stylesheet language will be discussed later in more detail, and the XML Link Language (XLL), which is a protocol consisting of two specifications to handle links between XML documents: XLink and XPointer.

**XLink:** The XLink protocol allows elements to be inserted into XML documents in order to create and describe links between resources. XLink provides a framework for creating both basic unidirectional links and more complex linking structures. It allows XML documents to:

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 6. Example of the corresponding Course01.dtd file*

```
?xml version='1.0' encoding="ISO-8859-1"?>
<!--
   DTD for a simple internet_lectures organisation.
-->

<!ELEMENT lectures (lecture+)>
<!ATTLIST lecture
        title   CDATA   #REQUIRED
        date    CDATA   #IMPLIED
        author  CDATA   "unknown"
>
<!ELEMENT lecture (image?, item-title?, item*)>
<!ATTLIST item
        type   (tech | exec | all) #IMPLIED
>

<!-- Defines the %inline; declaration -->
<!ENTITY % xhtml SYSTEM "xhtml.dtd">

<!ELEMENT item-title (title+)>
<!ELEMENT item ((title+) | item)* >
<!ELEMENT image EMPTY>
<!ATTLIST image
        alt    CDATA    #IMPLIED
        src    CDATA    #REQUIRED
        type   CDATA    "image/gif"
```

- Assert linking relationships among more than two resources,
- Associate metadata with a link, and
- Express links that reside in a location separate from the linked resources.

An important application of XLink is in hypermedia systems that have hyperlinks. A simple case of a hyperlink is an HTML element, which has these characteristics:

- The hyperlink uses URIs as its locator technology.
- The hyperlink is expressed at one of its two ends.
- The hyperlink identifies the other end (although a server may have great freedom in finding or dynamically creating that destination).
- Users can initiate traversal only from the end where the hyperlink is expressed to the other end.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- The hyperlink's effect on windows, frames, go-back lists, style sheets in use, and so on is determined by user agents, not by the hyperlink itself. For example, traversal of links normally replaces the current view, perhaps with a user option to open a new window.

For more information on the XLink specification, see also http://www.w3.org/TR/WD-xml-link.

**XPointer:** This specification targets documents that use its ID. XPointer supports addressing into the internal structures of XML documents. It allows for examination of a document's hierarchical structure and choice of its internal parts based on various properties, such as element types, attribute values, character content, and relative position. In particular, it provides for specific references to elements, character strings, and other parts of XML documents, whether or not they bear an explicit ID attribute. For additional information, see http://www.w3.org/TR/WD-xptr.

The **XHTML** specification is a way of making XML documents that look and act like HTML documents. The result of this specification is a document that can be displayed in browsers and treated as XML data. The XHTML specification is a reformulation of HTML 4.0 into XML. The latest information is at http://www.w3.org/TR/WD-html-in-xml/.

XML Schema (xsd) defines the types of elements a document can contain, their relationships, and the data they can contain. It provides options well beyond what the current DTD specification provides. For more information on XML, see the W3C specifications XML Schema (Structures) and XML Schema (Datatypes).

There are 13 kinds of components in three groups in XML schema: primary, secondary, and helpers. Their names are in the following list.

The primary components:

- Simple type definitions
- Complex type definitions
- Attribute declarations
- Element declarations

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The secondary components:

- Attribute group definitions
- Identity-constraint definitions
- Model group definitions
- Notation declarations

The helper components:

- Annotations
- Model groups
- Particles
- Wildcards
- Attribute uses

The excerpt from Course01.xsd is in Figure 7. You can see some similarities with definitions in DTD. Similar to DTD definitions, elements are defined in terms of their attributes (name and type). The ComplexType is a composition or sequence of other elements that serve to provide more accurate descriptions of the element. In later sections on Web Services and the Java binding compiler (Web Services with JAXB), the authors provide more information about the importance of XML schema in the process of marshalling and unmarshalling XML documents.

## XML Authoring and Java Parsers

Several authoring tools are currently available for creating XML documents, as well as browsers supporting XML[2]. Our list of references contains some pointers to Web sites with XML tools for writing XML documents, creating DTDs, and other documentation.

There are two kinds of parsers: validating and non-validating. According to the W3C organization (http://www.w3.org/TR), the definition of a validating parser is as follows:

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 7. Example of XML schema*

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://neptune.netcomp.monash.edu.au "
      xmlns="http://neptune.netcomp.monash.edu.au/Course01.xsd ">

<xsd:element name="lectures">
    <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="lecture"
                type="InternetLectureType" >
         <xsd:complexType>
           <xsd:sequence>
             <xsd:element name="name" type="xsd:string"/>
             <xsd:element name="title" type="xsd:string"/>
…
             </xsd:sequence>
         </xsd:complexType>
       </xsd:element>
     </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

*Validating processors must, at user option, report violations of the constraints expressed by the declarations in the DTD, and failures to fulfill the validity constraints given in this specification. To accomplish this, validating XML Processors must read and process the entire DTD and all external parsed entities referenced in the document.*

Non-validating processors are required to check only the document entity and the internal DTD subset for well-formedness. They do not check the document for validity. However, they are required to process all the declarations they read in the internal DTD subset and in any parameter entity within the document. Non-validating parsers generate three types of errors: a fatal error, an error, and warnings. Non-fatal errors typically occur when a XML document fails a validity constraint (invalid tag, tag is not allowed).

In client-server transactions, XML data is exchanged between parties; the parties are free to create their own tags to describe the data, set up schemas to specify which tags can be used in a particular kind of XML document, and use XML stylesheets to manage the display and handling of the data. Each XML document must be processed to be understood by a computer (server or client). Typical processing relies on document parsing. The parser, with the help of the XML schema document (either .xsd or .dtd file), extracts the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

relevant information necessary for the parser to create entities the computer understands.

There are a number of XML parsers available for XML document processing. The most popular seems to be JavaSofts's XML Parser and Apache's Xerces-J Parser.

We will briefly look at three XML parsers that are relevant to the objectives of this book (J2EE platform):

- SAX (Simple API for XML) Parser,
- DOM (Document Object Model) Parser, and
- XSLT (XML Style Sheet) Parsers.

**Using and setting up the SAXParser:** The SAXParser is flexible and can be used with any XML-compliant parser. It has a so-called "pluggability" layer, which enables it to "plug in" an implementation of the SAX, DOM, and XLS processor (stylesheets), thus letting the developer control how XML data is displayed.

J2EE provides JAXP APIs in the package javax.xml.parsers. SAX is an event-driven model (this means that you have to provide the callback methods, and the parser invokes them as it reads the XML data). The setup of SAX parser requires a bit of work. Detailed instructions about how to setup the SAX parser can be found in J2EE Tutorial (J2EE 1.4 Tutorial Update 4, 2004). The authors provide the reader with only limited information on SAX parser usage, which hopefully is enough to set up and use the SAX parser for simple document exchange and processing.

Let's start with the composition of an XML document. Each XML document is composed of elements, attributes, and entities. Element boundaries are defined by the start tag, such as <ElementName> and an end tag, </ElementName>. Attributes are name-value pairs associated with elements, and they serve as descriptors of the information about the element. For example:

```
<lecture lecture_type="video_recorded">
   <title> PORTALS </title>
</lecture>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

where the <lecture_type> is the attribute indicating the style in which the lecture will be conducted, and <title> tag holds the name of this particular lecture.

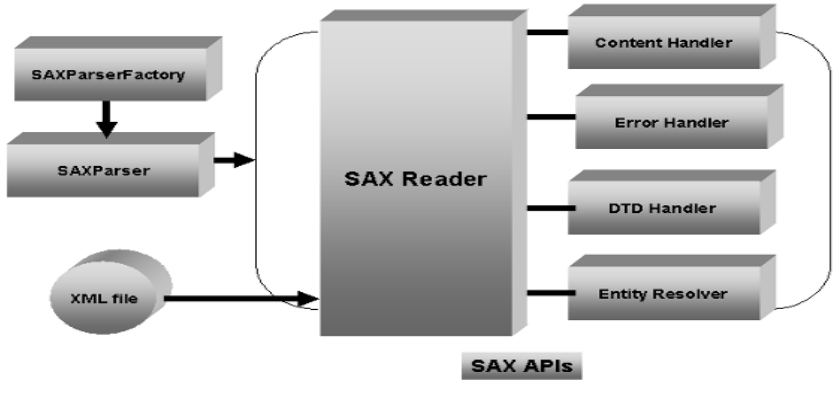
The basic structure of SAX APIs is in Figure 8. The SAXParser class is a wrapper, which defines a number of convenience methods. The SAXParserFactory is used to create the actual parser. The SAXParser interface contains several kinds of parse() methods. The XML document is passed to the parser for processing. If any errors are detected, the appropriate error handling methods are invoked on the ErrorHandler objects.

The ContentHandler is invoked when compulsory tags, for example start document, start element, or end element are missing. The parser uses the EntityResolver object to detect data identified by a URI. DTDHandler processes DTD schema.

Entity references, marked by special identifiers, will be expanded by the parser. A user-defined entity is an XML mechanism, which associates the identifier with the user-defined data to be expanded. For example, we can assume that a set of lectures to be presented in a postgraduate course is defined as title entities in DTD (e.g., lecture on PORTALS, WEB_SERVICES, etc.). We can write the following code:

```
<lecture lecture_type=”video_recorded”>
<title> &PORTALS </title>
</lecture>
```

*Figure 8. SAXP API structure*



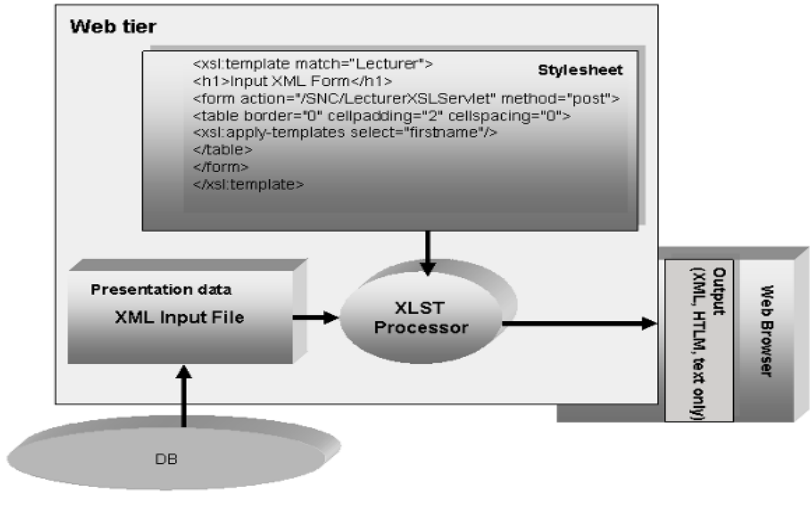

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The variable &PORTALS will be replaced with the DTD definition of this particular lecture.

- **Using stylesheets (XLST):** An XML document can be transformed to another type of document using an XLST (a language to specify how to transform an XML document of one type to another document type). The transformer can be used to produce HTML, a different XML structure, or even a text file. The J2EE platform provides JAXP XSLT to support the transformation. The central component of this is the javax.xml.transform package, which defines and supports the mechanics and overall process for any transformation that is performed.
- **Stylesheets compilation:** Before a stylesheet can be processed (Figure 9), it must be converted into an internal, machine readable format. Typically, the stylesheet is read into memory using an XML parser, translated into machine format and then preserved in memory for repeated use. Many different XSLT processors implement stylesheet compilation. JAXP includes the javax.xml.transform.Templates interface to provide consistency.

The newTransformer() method of this interface allows the stylesheet compiler to obtain a new instance of a class that implements the Transformer interface. This Transformer object supports the performance of XSLT transformations.

*Figure 9. XLST processing*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Transformations are generally static and can be used multiple times over different XML documents. Java classes that offer translation capabilities are called translets. Translets are expected to be small, in the 100 Kilobytes range size. Translets can significantly improve the speed of server-side transformations and the client-side transformation of XML into other formats. The Apache XML project (xml.apache.org) provides XSLTC, which combines a compiler and a runtime processor. The compiler compiles an XSL stylesheet into a set of Java classes (translets). The classes are ready to be deployed in a similar fashion to any other Java classes, along with the xsltc.jar and the runtime.jar.

XPath recommendations can be found on the W3C Web site (http://www.w3.org/TR/xpath). XPath is now a part of IBM's Application Developer Support Tools package. In a sense, it is an XSL sub-language that is designed to be used with XSLT. It is used for identifying or addressing parts of a source XML document. Every node within an XML document can be uniquely identified and located using the syntax defined for XPath.

## How to Use SAX Parser

To demonstrate how to set up the SAXParser, we have modified the example from a J2EE Tutorial (J2EE 1.4 Tutorial Update 4, 2004). In this example, the program echoes the parsed data derived by the SAX parser class from the XML input file. The first step is to create a Java class that incorporates the parser. Let us start with necessary imports (Figure 10).

*Figure 10. Imports and main() method*

```
/*
 *Echo the input with SAX parser
 */


import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;


public class Echo01 extends DefaultHandler {
    static private Writer out;
    StringBuffer textBuffer;
…..
}
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

We need to import DefaultHandler, SAXParser, and SAXParserFactory (at least) classes. We also need to create main() method in our code sample. Note that we are running a standalone program invoked from the command line with one argument — the XML file we want to echo.

Figure 11 shows the code needed to create a new instance of the parser and invokes parse() methods to start parsing. The parser uses ContentHandler classes (in our case, the DefaultHandler was extended to Echo01) to provide empty methods for handling XML document tags such as startElement, endElement, and so forth. When any of these tags are encountered, they are sent to the appropriate method together with all attributes for processing. We need to code these methods to perform the intended logic (Figure 12).

All methods use SAXException instance to report errors. SAXException could be constructed using a message and another nested exception in order to be more specific regarding the type and location of the error.

SAX is fast and efficient in sequential data filtering. The SAX event model calls one method when the start of the element is encountered and another method at the end of element tag. On the other hand, if the program needs to do some processing for Element 1 and something different for Element 2, then the *pull parser* such as the Streaming API for XML (StAX) would be a better choice. Pull parsing is suited for applications that need to transform input XML to other formats. Such a transformation is often complex; therefore, the pull parsers

*Figure 11. Set up the parser and default handler*

```
// Use an instance of our class as the SAX event handler
        DefaultHandler handler = new Echo01();

        // the default is a (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();

        try {
// Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

// Parse the input file provided on the command line
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse(new File(argv[0]), handler);
        } catch (Throwable t) {
            t.printStackTrace();
        }

        System.exit(0);
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 12. startElement() processing example*

```
//note: simple name = simple name, qName = qualified name
//eName = element name

public void startElement(String namespaceURI, String sName,       String
qName, Attributes attrs) throws SAXException
{
      echoText();

      String eName = sName; //make simple name = element name
      if ("".equals(eName)) {
         eName = qName; // element name becomes qualified and not
namespace aware
      }

      emit("<" + eName);

      if (attrs != null) {
         for (int i = 0; i < attrs.getLength(); i++) {
            String aName = attrs.getLocalName(i); // Attr name

            if ("".equals(aName)) {
               aName = attrs.getQName(i);
            }

            emit(" ");  //output method
            emit(aName + "=\"" + attrs.getValue(i) + "\"");
         }
      }

      emit(">");
   }
```

must maintain the state during parsing. Using SAX would require maintaining the state between call-backs to determine the correct response to a SAX event. In pull parsing, the application can access the information from XML when needed and pull the next event when it is ready to process it.

## Web Services Description Language: WSDL

WSDL (Web Services Activity, 2004) is a language that attempts to describe communications between Web entities in a structured way. In WSDL document, the services are defined as a collection of network endpoints called ports.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

This language is extensible and enables the definition of other binding mechanisms, such as Java and SOAP over Java Messaging Service (JMS).

Each WSDL document must contain the following main elements:

- **Type:** A container for data type definitions, using some type definition 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 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 fault messages.
- **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.
- **Port:** A single endpoint, which is defined as an aggregation of a binding and a network address.

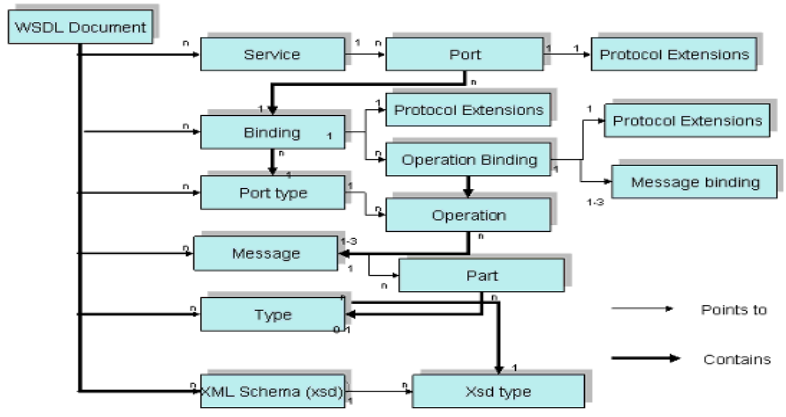
As an extensible language, WSDL 1.1 and WSDL 2.0 (draft) introduce specific binding extensions for various protocols and message formats:

- WSDL SOAP binding, which is capable of describing SOAP messaging over HTTP,
- Direct HTTP interface (for example any servlet), and
- MIME binding.

WSDL does not define any mapping to programming languages such as Java; the language binding capabilities deal with transport protocols only. This represents a major difference from interface description languages, such as CORBA's Interface Definition Language (IDL), which contains information about type mappings and language bindings as well.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 13. WSDL document relationships*




The WSDL document has a defined structure in terms of its comprising elements and predefined relationships between these elements.

Figure 13 shows the infrastructure, element relationships, and multiplicity within the WSDL file. The summary that follows indicates how to interpret the diagram:

- One WSDL document contains several services. A service can contain several port definitions (service endpoints); however, each port definition may contain only one specific protocol extension.
- One WSDL document (the same document) contains several bindings. A binding can be pointed to by several ports. The binding contains one protocol extension with defined style and transport, and many operation bindings. Each of these operation bindings is composed of one protocol extension with defined action and style and one to three message bindings with defined encoding.
- The same WSDL document contains many port types. A port type can point to many bindings and contain many operations, which are referenced by zero or more operation bindings.
- The same WSDL document contains many messages. One to three messages are required to define an operation. The message is composed of several parts.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- The same WSDL document contains several types. A type points to zero or more parts.
- The same WSDL document points to many XML schemas. An XML schema contains several XSD types that define the different data types.

In practice, the WSDL definition is typically split into several physical files (in such case we need to use *import* statements to bind the physical files together). There are also some important considerations with regard to service documentation and existing deployment descriptors. More extensive HTML documentation of a service covering parameters required by the service, return values, and exceptions, helps developers design better interfaces and interact with the service. This information typically goes beyond what is included in the WSDL document.

In order to deploy a Web service and appropriate processing structures, several deployment descriptors are required:

- web.xml deployment descriptor controls Web application behavior and informs the J2EE Web container how the J2EE components are going to be deployed, and
- webservices.xml deployment descriptor informs the servers and clients that the component acts as a Web service.

Similar to other types of middleware, Web service interoperability depends on mapping of different components and types. First, we need to review notational conventions and specify the relationship between namespace prefixs and namespace definition locations (see Table 1).

*Table 1. Namespace prefix and locations*

| Namespace prefix | Namespace location |
|---|---|
| xsd | http://www.w3.org/2001/XMLSchema |
| xsi | http://www.w3.org/2001/XMLSchema-instance |
| soapenc | http://schemas.xmlsoap.org/soap/encoding/ |
| wsdl | http://schemas.xmlsoap.org/wsdl/ |
| wsdlsoap | http://schemas.xmlsoap.org/wsdl/soap/ |
| ns | user-defined namespace |
| apache | http://xml.apache.org/xml-soap |

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The WSDL document defines the following mappings:

- **java-to-wsdl mapping:** This is the mapping descriptor containing the minimum of information provided by the developer. There is a distinct similarity with CORBA concepts and IDL (for more details refer to Harkey & Orfali, 1998).
- **wsdl-to-java mapping:** This mapping descriptor contains information about wsdl types and their relationship to generated Java types. Some vendors also provide a processor that implements java-to-WSDL mapping rules. The example is Sun Microsystesm's wscompile tool, which generates the WSDL file for the service for use with JAX-RPC clients and services (some usage notes are on this Web site: http://java.sun.com/webservices/docs/1.1/tutorial/doc/JAXRPC6.html). The other example of such processor is IBM's *Java2WSDL* command for *bottom-up processing*. In bottom-up processing, an existing Java service implementation is used to create a WSDL file defining the Web service. In this case, it may happen that not all Java classes and constructs have mappings to WSDL files. For example, Java classes that do not comply with the Java bean specification rules might not map to a WSDL construct. Some Java classes and constructs have multiple mappings to a WSDL file. For example, a java.lang.String class can map to either an xsd:string or soapenc:string construct. Typically, the processors select one of these mappings, and the developer must edit the WSDL file if a different mapping is required.
- **Mapping between WSDL and SOAP messages:** The WSDL file defines the format of the SOAP messages that are transmitted between the provider and the requestor. This information is used at runtime to ensure that the SOAP message is properly serialized and deserialized.

Additional mapping specifications relevant to all the earlier mappings and applicable for packages, XML types, variables, exceptions, and messages are described by additional mapping descriptors:

- package-mapping between the Java package name and the WSDL XML namespace. The XML namespaces provides a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- Java-XML-type-mapping between Java class (Java type) and associated WSDL type.
- variable-mapping descriptor defines the mapping between the variable in a Java class and a corresponding WSDL element.
- exception-mapping descriptor defines a mapping between a Java exception type and a WSDL type representing the exception/message.
- service-interface-mapping defines a mapping between a Java type (class) for a Web service and a corresponding WSDL type. It uses some subtypes such as service-interface, wsdl-service-name, port-mapping, and port-name.
- service-end-point-interface-mapping descriptor maps a Java type for a Web service endpoint and WSDL port.
- service-endpoint-method-mapping descriptor represents the mapping between Java methods for a Web service endpoint interface and WSDL port operations.
- method-param-parts-mapping describes the mapping for Java method input parameters for a Web service endpoint interface method and the WSDL port operation input messages.
- WSDL-message-mapping is the specification for the WSDL message to which a Java input parameter is mapped.
- WSDL-return-value-mapping is the mapping between a Java method output parameter for a Web service endpoint interface method and a WSDL port operation output message.

Note that when the service is implemented as a JAX-RPC based service (the service utilizes the Remote Procedure Call [RPC] paradigm), the webservices.xml document always contains the following pair of elements:

<webservice-desciption> and <jaxrpc-mapping-file>.

The <jaxrpc-mapping-file> defines the mapping between Java types used by this Web service and the corresponding WSDL types in the WSDL file associated with the Web service. The java-wsdl-mapping.xml is further defined by the XML schema that is referenced at the top of the java-wsdl-mapping.xml document.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

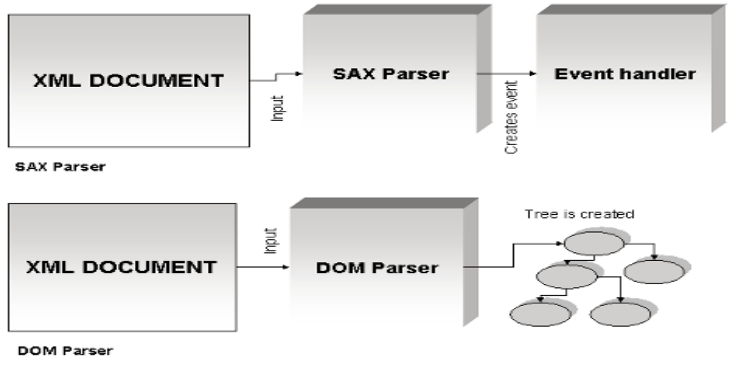
In summary, any cooperating J2EE collection of Web components may be deployed as Web Services. The web.xml descriptor is used to control the Web service behavior (servlets, Java Enterprise Beans, and JSP deployment structures). The webservices.xml and java-wsdl-mapping.xml deployment descriptors are used as configuration files to configure the J2EE component as a Web service and enable the container to map Java types to WSDL types.

## Document Object Model

According to W3C, "The Document Object Model is a platform — and language-neutral interface — that will allow programs and scripts to dynamically access and update the content, structure, and style of documents. The document can be further processed, and the results of that processing can be incorporated back into the presented page."

The objective for the XML DOM is to provide a standard programming interface to XML and HTML. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense; increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and DOM may be used to manage this data. The XML DOM is designed to be used with any programming language and any operating system. A programmer can create an XML or HTML document, navigate its structure, and add, modify, or delete its elements.

*Figure 14. SAX and DOM parsers processing*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Figure 14 shows the difference between DOM and SAX parsers. SAX processes documents serially. It converts the elements of an XML document into a series of events. Each particular element generates one event, with unique events representing various parts of the document. A developer provides appropriate event handlers to handle the events and take appropriate actions. SAX processing is fast because of its sequential event model and small memory storage requirements.

DOM processing creates a tree from the elements in the XML document. The processing requires more memory to store the tree. On the other hand, this feature allows random access to document content and enables splitting documents into fragments, which makes it easier to code DOM processing.

DOM represents a tree view of the XML document. The documentElement is the top level of the tree. This element has one or many child nodes that represent the branches of the tree. A Node Interface Model is used to access the individual elements in the node tree. Some XML parsers support the necessary functions to traverse the node tree (pull parsers), access the nodes and their attribute values, insert and delete nodes, and convert the node tree back to XML. For more information, visit http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/introduction.html.

Every specific document structure is represented in DOM by one of the following specialized interfaces and extended interfaces:

Specialized interfaces:

- Document
- Document fragment
- Attribute
- Element
- Text
- Comment
- Character data
- DOM exception
- DOM implementation
- Node

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- Node list
- Named node list

Extended interfaces:

- CDATASection
- DocumentType
- Notation
- Entity
- EntityReference
- ProcessingInstruction

*Document* represents the whole document, and the interface defines methods for creating elements, attributes, comments, and so on. Attributes of a node are manipulated using the methods of the element interface.

The example of an XML document in Figure 15 will produce the DOM tree shown in Figure 16. Note that the end nodes of the tree structure do not allow any "children" nodes; for example, Text is an end node with no children nodes.

The DOM application reads an XML document and forms an object representation. However, this representation remains only in memory. Changing a DOM object in memory does not automatically modify the original file. The application programmer has to make the changes in the original file.

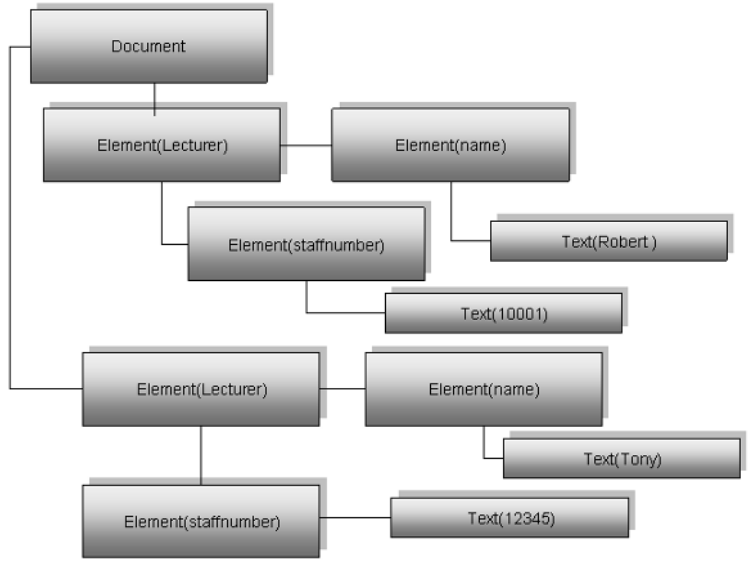
The previously-mentioned interfaces are language neutral; their implementation is language specific. The interfaces allow the programmer to analyze, manipu-

*Figure 15. Example of a DOM document*

```
<?xml version="1.0" encoding="UTF-8"?>
<Lecturer>
   <name>Robert</name>
   <staffnumber>10001</staffnumber>
</Lecturer>
<Lecturer>
    <name>Tony</name>
    <staffnumber >12345</ staffnumber >
</Lecturer>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 16. Example of DOM documentstructure*




late, and produce XML or HTML documents, using programming tools. This way, human- and machine-readable documents are processed and produced by computer processes. This is important for various dynamic interfaces between systems, as we will see in the next chapter.

## Endnotes

1 Unicode (currently in version 4.0.1) provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. XML documents can, however, declare a different encoding to reduce the size of the document.

2 A list of browsers supporting XML and stylesheets is on this Web site: http://www.ucc.ie/xml/#browser

.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter III

# Protocols

Today's enterprises increasingly rely on the Web to support their operations, global business, and integration of their business processes with those of their suppliers, partners, and customers. This requires that the architecture and design of business systems and applications should allow collaboration among distributed software components. The attempts to support collaboration and integration of heterogeneous systems led to the creation of a number of protocols and middleware. To mention a few, Common Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), and Java-based Remote Method Invocation (RMI) have been developed and widely used. In addition, supporting pipelines (communication protocols) have emerged over the years.

There are many protocols used in building large Web service applications or portals. This section introduces two main protocols — Universal Description, Discovery and Integration (UDDI) and Simple Object Access Protocol (SOAP), and explains the properties of URI, URL, and URN.

## UDDI (Universal Description, Discovery, and Integration)

UDDI is a registry and directory that attempts to support the business-to-business electronic cooperation over the Internet. Its programmable interface

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

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.

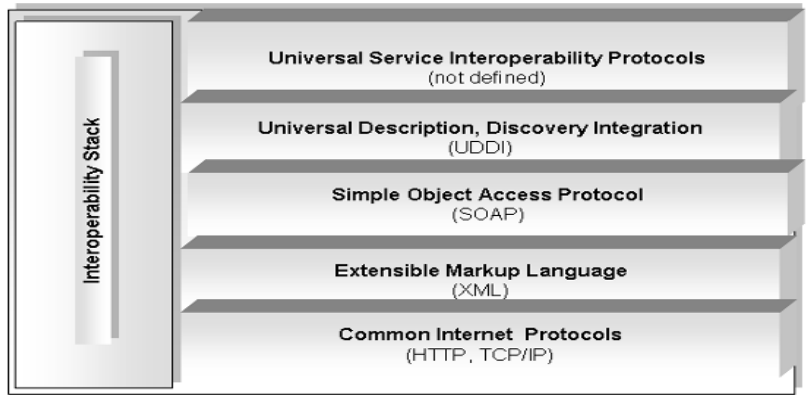
UDDI defines an interoperability stack shown in Figure 17, which enables publishing, as well as searching for Web Services. UDDI relies on standards-based technologies such as TCP/IP, HTTP, XML, and SOAP to create a uniform service description format and service discovery protocols.

The UDDI specification describes a conceptual underpinning of Web Services and associated programmatic interface that defines a simple framework for describing and accessing Web Services. The specification consists of several related documents and an XML schema that defines a SOAP-based programming protocol for registering and discovering Web Services. Currently, several industry groups are working on the definition of Web Services interface standards.

A business can register three types of information in the UDDI registry:

- **White Pages:** White pages provide basic contact information (the business name, address and other contact details). White pages also provide unique business identifiers that consist of a nine-digit sequence. This is 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.

*Figure 17. UDDI interoperability stack and related protocols*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- **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 behaviors 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 centers, technical support for a product, fax-based services such as fax to e-mail conversion service, and so forth.

UDDI usage is based on the perspective of the user of the registry (Figure 18). We observe three types of users:

- Business analyst, whose role is to match the services with required business processes and propose the suitable business solution;
- Business user, who is searching for business services; and
- Developers, who publish services or execute a look-up to discover a service to which they want to bind.

From a *business analyst's perspective*, UDDI is similar to an Internet search engine for business processes. A standard search engine organizes and indexes Web sites by their URLs. A business exposing a Web service needs to reveal more information than a URL. A business analyst can browse one or more UDDI registries to view different businesses and their exposed Web Services, and evaluate suitability based on their specifications. There are many portals and marketplaces attached directly to a particular UDDI server or to a more general search portlet that supports UDDI registry and search. Even though the APIs support random search for business partners, it is not feasible to select new business partners just by running Registry query programs. A more realistic case is that the business analyst uses his or her domain knowledge of the problem to discover potential business partners or developers using the exposed interfaces to write programs and interfaces to use the discovered services.

*Business users* need a registry to search for services. UDDI is not suitable to be browsed directly. The information stored within it contains XML and other documents that are not human reader friendly. Over the years of UDDI

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

existence, a series of marketplaces and business search portals has been formed to provide a more user-oriented approach to browsing the business services exposed in a UDDI registry.

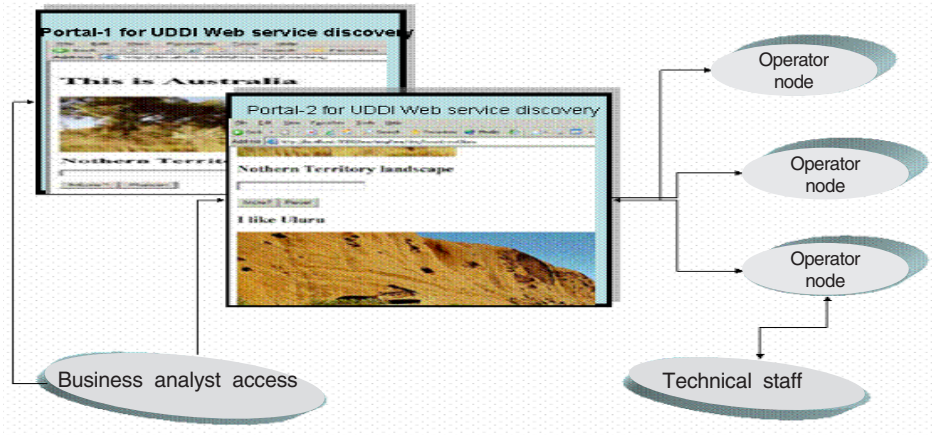
*Software developers* use the UDDI programming API to publish services, query the registry to discover services, and bind to the exposed services. It is expected that in the future, any service could be discovered dynamically using programming interface and used without human interaction or need to create additional UIs.

Structurally, UDDI uses four types of topologies:

- The *UDDI Business Registry* (UBR) is the network of public UDDI registries on the Internet. It is a Grid formed by operator nodes;
- The *UDDI E-marketplace* is the UDDI Registry dedicated to a given industry sector;
- The *UDDI Private node* is the UDDI Registry on the intranet utilized by corporate users; and
- The *UDDI Extranet node* is the UDDI Registry on the extranet to expose services to business partners.

The UBR provides inquiry services to all interested entities. The facility to publish services is restricted to authenticated entities only. Any business can

*Figure 18. UDDI usage perspectives*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

create an operator node and make it available over the Internet as a part of UDDI. Private operator nodes can define their own access rules for other business entities.

Figure 18 shows the process of working with the UDDI registry. The UBR, also known as the Public Cloud, is conceptually a single system built from multiple nodes, whose data are synchronized through replication. A set of operator nodes hosts a copy of the content (replication function). The same information and quality of service is provided by all operator nodes. Content is uploaded into a single operator node, and that operator node becomes the master owner of that content. Any subsequent update or delete operations to change or remove the data must occur at the operator node where the data were originally uploaded.

Some large corporations such as IBM, Microsoft, SAP, and NTT Communications are running public UDDI sites. For example, the public UDDI test registry maintained by IBM is located on the Internet at http://uddi.ibm.com/testregistry/registry.html. Any company can register their services in these UDDI instances, or look for services provided by other business entities.

The UDDI project (UDDI specification) defines a set of XML schema definitions that describe the data formats used by the various APIs. These documents are all available at http://www.uddi.org. The current version of all specification groups is version 3.0. The specification defines important processes and interfaces of the UDDI registry. These definitions include:

- **UDDI Replication:** This document describes the data replication processes and interfaces to which a registry operator must conform to achieve data replication between sites. This specification is not a programming interface, but the replication mechanism used among UBR nodes.
- **UDDI Operators:** This document specifies the behavior and operational parameters for node operators. It defines data management requirements the operators must follow. The specification includes information relevant to node operators' responsibilities, for example recording, backup and data integrity for all data, also ensuring that each business registration has a valid e-mail address. This document describes the process and mechanism for maintaining the UDDI nodes. It is not a programmer's API specification; private registries do not have to support UDDI operators.
- **UDDI Programmer's API:** The UDDI hosts a set of services including inquiry and publishing services. This specification, together with the UDDI

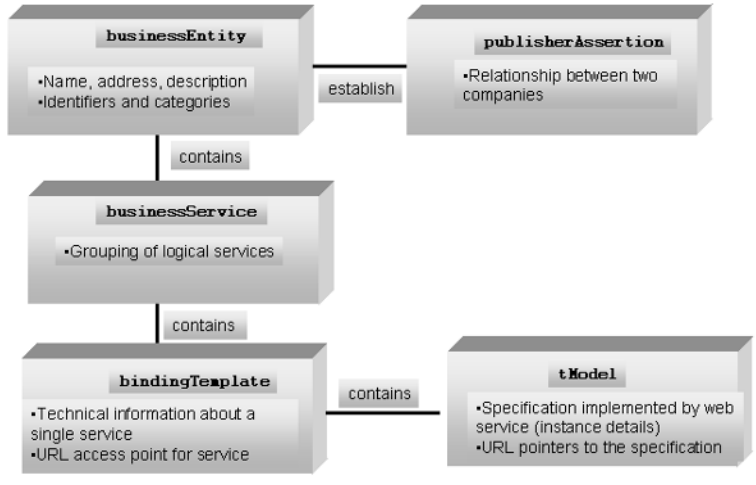
Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

XML API schema and the Data Structure specification forms, is a complete programming interface to the UDDI registry. The specification defines a set of interfaces to hosted services that all UDDI registries support, and a series of SOAP messages containing XML documents that a UDDI registry accepts, parses, and responds to.

- **UDDI Data Structures:** This specification contains the details of the XML structures contained within the SOAP messages and defined by the UDDI Programmer's API. The specification defines five core data structures and their mutual relationships. The main elements of the UDDI data model are listed in the following:
  - businessEntity represents the physical company that registered the services with UDDI.
  - businessService represents a specific service offered by a company.
  - bindingTemplate contains instructions for service invocation.
  - tModel: Briefly, it is a structure similar to a database table. It contains the following information about an entity: the name, description, URL, and the unique key. Figure 19 shows the relationship between the four main elements in the UDDI structure.

The businessEntity (a company or other business entity) has a descriptive URL that holds the information about the company and a list of contacts. In

*Figure 19. Relationship of primary UDDI structures*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

addition, the businessEntity also contains a list of business services provided by the company. Each business service can expose multiple interfaces through which the service can be invoked. Each invocation is described by a bindingtemplate, which describes the details of the exposed interface for this service. A bindingtemplate and necessary documentation details are linked together through a data structure called tModel. Each bindingTemplate can be composed of a collection of tModels.

The publisherAssertion structure is used to establish public relationships between two companies represented by corresponding businessEntity structures. Two companies have a valid relationship only if both companies created a similar publisherAssertion structure.

A businessEntity may operate one or more services encapsulated in the corresponding businessService structures. Each businessService element describes a set of related services provided by the business. These services can be Web Services or other services, even non-electronic services. Each businessService document can be used by several businessEntity elements.

Each businessService may contain one or more bindingTemplate structures. The bindingTemplate provides pointers to service technical descriptions and the access point URL. The bindingTemplate does not contain any service details or specifications, but it may contain an optional Web service documentation in textual form, the access point URL, and a reference to one or more related tModel structures.

A tModel is a digital description of how to interact with this Web service. It is an abstraction of Web service behavior. This structure contains only pointers to the locations of the actual specifications, not Web service direct specifications. Companies can use the information in a tModel to see if the Web service is compatible with their business requirements and usage characteristics.

In order to use tModels in UDDI, you need to comply with the requirements listed in the following:

- All businessEntities need to be identified and categorized for search purposes;
- Each businessService also must have an identity and belong to one or more categories;
- tModels have an identity, and they must be categorized; and
- The businessServices must provide aliasing and binding information pointing to their abstract tModel definitions.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

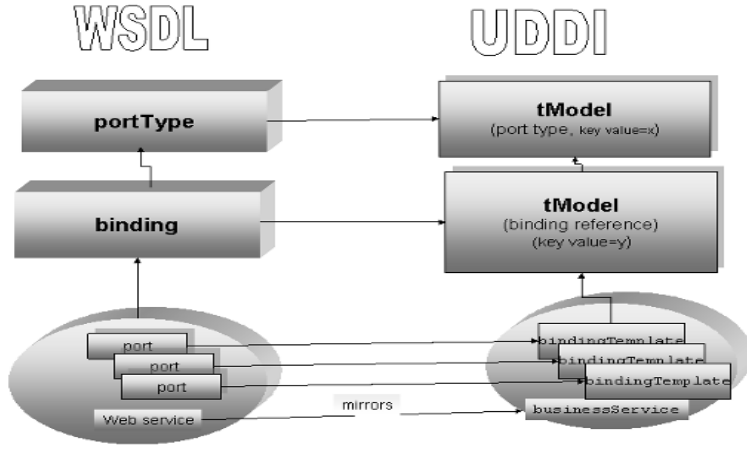
If you wish to publish a Web service in UDDI, you need to map WSDL service descriptions to the UDDI model. This approach is detailed in the Technical Note[1]. The technical note, apart from some other goals, also introduces new canonical models, specifically nine new tModels and six new Category Systems. The newly introduced Category Systems are listed in the following:

- WSDL Entity Type Category System
- XML Namespace Category System
- XML Local Name Category System
- WSDL portType Reference Category System
- Protocol Category System
- Transport Category System

Let's use WSDL Entity Type Category System for explanations of some necessary concepts used in publishing and finding services. The WSDL Entity Type Category System is used to identify UDDI entities that correspond to WSDL entities. It is currently used with the tModels and businessServices structures.

Each WSDL description of the service published in UDDI must contain the following six elements: definitions, types, message, portType, binding, and service. The relationships are outlined in Figure 20. The portType is repre-

*Figure 20. Mapping from WSDL to UDDI*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

sented by a UDDI tModel. This tModel is categorized, using the WSDL Entity Type. The relevant Category System is known as a WSDL portType tModel category and distinguishes it from other types of tModels with which the service might be associated.

A WSDL binding is also represented by a UDDI tModel. This tModel is categorized as a WSDL binding tModel. This categorization uses the same Category System as the portType tModel, but with a different key value to differentiate a binding tModel from a portType tModel.

The WSDL for Web service may represent a Web service interface for an existing service. However, sometimes this may not be the case. In this case, there may be an existing UDDI businessService that is suitable, and WSDL information can be added to that existing service. If there is no suitable existing service found in the UDDI registry, a new businessService is created. Finally, the WSDL port is represented by UDDI bindingTemplate. A WSDL service may contain multiple ports. These ports are exactly mirrored by the containment relationship in a UDDI businessService and its bindingTemplates.

The UDDI registry is accessible using Java API for registries (JAXR API). In a sense, the unique keys of tModels represent the concept similar to a "namespace." For example, the company registers several numbers (switchboard phone number: 61 3 99045555, company fax number: 61 3 99046666, and help desk Web page). In order to learn the semantic meaning of these numbers you need to know the context (or namespace). From the context, you can establish that the number 61 3 99045555 is the switchboard phone number, the number 61 3 99046666 is the fax number, and www.mymonash.edu.au is the main university Web page. This company can also declare other entries such as a help line phone number and help class URL (www.mymonash.edu.au) and store them in the tModel database[2] (Figure 21). The tModel descriptions are

*Figure 21. tModel database information*

```
company name: mymonash
Service name: helpdesk
tModel: key=15, name=help-phone, description=helpdesk phone, url:
www.mymonash.helpdesk
//here you need to specify the binding in the form of access point and the key:
binding:
    accesspoint: 61-3-monash_helpdesk
    tModelInstanceInfo: 15
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

identified by a unique key. Our helpdesk's phone has the key=15 (this is typically a much longer number).

In Figure 20, we have shown the relationship between a bindingTemplate and tModels. A bindingTemplate can point to a technical specification identified by a tModel. This technical specification has two parts:

- The type of the specification (e.g., e-mail, fax, WSDL, SOAP service, etc.), and
- Documents identifying inputs and outputs (in the case of a SOAP service, these documents could be XML input/output message formats).

To create description of the company service in UDDI, the following steps are required:

- Define a tModel for the service. This requires a name for the service, which is assigned to one of the types of services available (some of the categories to which a tModel can belong are pre-defined by the UDDI WSDL, SOAP service).
- Define a business service (businessService) under the company identification, pointing to the tModel created previously.
- In the business service definition, specify a URL pointing to the actual WSDL document defining inputs/outputs to the service.

The UDDI specification does not define Java API for accessing a UDDI registry. The specification for use by developers (Programmer's API) defines only a series of SOAP messages that a UDDI registry accepts. Several approaches are defined for accessing the registry:

- **Java-Based SOAP API:** The developer can use API to create SOAP messages containing the XML document. Such XML document must be created manually and inserted into the body of a SOAP message. The developer must be familiar with the SOAP structure and message ordering in the SOAP that the UDDI registry accepts.
- **Developing Java-Based Client:** Some companies created a specific set of API — client — for accessing a UDDI registry (for example Systinet). The API has classes that represent data and message structures

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 22. Service interface in Java*

```
public interface CampusService {
        public String campusLocation();
}
```

*Figure 23. WSDL definitions for CampusService service*

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CampusService"
targetNamespace="urn:helpdesk">
    xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:tns="http://neptune.monash.edu.au/helpdesk"
xmlns:xsd1="http://neptune.monash.edu.au/campusservice/schema/"

<documentation>
        Standard WSDL service interface definition for a Campus
    location service.
</documentation>
```

supported by the UDDI registry. The interaction with UDDI and SOAP or XML message construction are also part of this API.

- **JAXR API:** JAXR is a Java XML API for working with service registries. All elements of a UDDI are described using XML documents. We can send an XML message containing a service definition, and the UDDI registry should be able to work with it (update, register, etc.).

Working with JAXR may be quite complex. Let's assume that we need to provide a service application that lists University campus locations. The simple interface consists of only a single method that produces a list of campuses that the University currently operates. The definition of the Java interface is in Figure 22.

We need to convert this interface definition into the service description in WSDL. We have already discussed the structure of a WSDL document. It must contain the following elements: definitions, types, message, portType, binding, and service. Let's define the WSDL definitions first (Figure 23).

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The definitions include all standard namespaces for SOAP, SOAP encoding (soapenc), and XML schema definitions. Furthermore, the location of the CampusService (targetNamespace) and associated xsd[1] schema must be specified.

The only type defined here is the string array for the campusLocation() method return and the return type (Figure 24).

There are two messages necessary to be defined (Figure 25): a *response* message, which refers to the type with the name ArrayOf_soap_enc_string declared previously in types. The other message is the *request*. Since we are not passing any parameters, this request message is empty.

We need to define the portType for the service called CampusService (Figure 26).

The name of the portType corresponds to the name of the service: CampusService. It operates on input and output messages specified earlier.

*Figure 24. WSDL types definition for campusService*

```
<types>
        <schema targetNamespace="urn: helpdesk"
        xmlns:xs="http://www.w3.org/2001/XMLSchema" >
          <complexType name="ArayOf_soapenc_string">
                <complexContent>
                        <restriction base="soapenc:array">
                                <attribute ref="soapenc:arrayType"
                                        wsdl:arrayType="soapenc string[]"/>
                        </restriction>
                </complexContent>
          </complexType>
          <element name="ArrayOf_soap_enc_string" nillable="true"
                type="ArrayOf_soapenc_string"/>
        </schema>
</types>
```

*Figure 25. Input and output message definitions*

```
<message name="campusLocationResponse">
        <part name="return" type="ArrayOf_soapenc_string"/>
</message>

<message name="campusLocationRequest">
</message>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 26. portType specification for CampusService*

```
<portType name="CampusService">
        <operation name="campusLocation">
                <input message="tns:campusLocationRequest"
                        name="campusLocationRequest"/>

                <output message="tns:campusLocationResponse"
                   name="campusLocationResponse"/>
        </operation>
</portType>
```

We define only one service binding. This binding allows access to the service via SOAP. The SOAP operation corresponds to the method public String campusLocation().

This section of the WSDL document defines where the service's port bindings are actually accessed. Although, the WSDL document may define bindings for SOAP, HTTP GET, HTTP POST, and MIME, you can use any binding mechanism, for example RMI/IIOP. The example in Figure 28 uses CampusServiceBinding as port binding in service definition.

Using JAXR to access registries means that your program calls JAXR's client API. Those calls invoke objects on a JAXR provider. A JAXR implementation then maps registry information models to a specific registry's requirements and communicates with a registry provider. Using JAXR and programmatic interface, we can construct a registry client to register our *CampusService* service as suggested in the following example.

*Figure 27. Service binding*

```
<binding name="CampusServiceBinding"
                type="tns:CampusService">
  <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="campusLocation">
                <soap:operation soapAction=""/>
        <input name="campusLocationRequest">
        </input>
        <output name="campusLocationResponse">
        </output>
    </operation>
</binding>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 28. Service specification*

```
<service name="CampusService">
  <port binding="tns:CampusServiceBinding"
name="CampusServicePort">
  </port>
 </service>
</definitions>
```

*Establish the connection to the registry, and set the properties* (Figure 29)*:* JAXR connections are created via the ConnectionFactory class. You can use the javax.xml.registry.ConnectionFactory class property to specify what factory class to use when creating a registry connection. The JAXR specification suggests that you obtain your registry ConnectionFactory via a naming and directory service lookup; alternatively, you might create a new factory via a static creational method.

You have to associate your credentials with the connection, as well as specify credentials for lifecycle management-related activities (Figure 30). Furthermore, you also need to obtain the registry service object and managers. If you decide to query the registry, you will obtain a BusinessQueryManager or a DeclarativeQueryManager from RegistryService, depending on how you

*Figure 29. Set properties and create connection to the registry*

```
Properties props = new Properties();
//load proxy host and proxy port properties
properties.load(new FileInputStream("./proxyHost.properties"));
set URL properties
props.setProperty("javax.xml.registry.queryManagerURL", queryUrl);
props.setProperty("javax.xml.registry.lifeCycleManagerURL", publishUrl);
//create connection
try {
 // Create the connection and pass it the configuration properties
   ConnectionFactory factory = ConnectionFactory.newInstance();
    factory.setProperties(props);
    connection = factory.createConnection();
    System.out.println("connection to registry is created");
       } catch (Exception e) {
          e.printStackTrace();

          if (connection != null) {
             try {
                connection.close();
             } catch (JAXRException je) {
             }
.....
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 30. Get access authorization and set credentials*

```
try {
    // Get registry service and managers
    rs = connection.getRegistryService();
    bqm = rs.getBusinessQueryManager();
    blcm = rs.getBusinessLifeCycleManager();
    System.out.println("Obtained registry service, query " + "manager, and life cycle manager");

// Get access authorization from the registry and set credentials
   PasswordAuthentication passwdAuth = new  PasswordAuthentication(username,
password.toCharArray());

   Set creds = new HashSet();
   creds.add(passwdAuth);
   connection.setCredentials(creds);
   System.out.println("Established security credentials");
```

wish to search. For lifecycle management tasks, you need to have the registry's LifeCycleManager.

**Start working with the registry (query or update the registry).** The registry's response to both queries and updates is transmitted to the client in the form of JAXRResponse objects. BulkResponse is a JAXRResponse type and contains a Collection object, whose elements correspond to query replies.

The example in Figure 31 shows how to create a new organization, categorize it, and include it in the registry using appropriate managers' methods.

**Using tModel to register the service.** A business entity — for instance the University — may advertise a set of business services. Each service defines the access points and technical specifications (bindings). In order to register the service, we need to categorize this service. The category in turn identifies the appropriate tModel. We may then use the special classification for tModels that points to WSDL documents called *wsdlSpec* tModels (category name is *uddi-org:types* with the value *wsdlSpec*) and the registry JAXR APIs (Java API for XML Registries) to register a new tModel that points to a URL of the CampusService WSDL interface programmatically.

The client can also find organizations by name using the findOrganisations() method. This method uses the collection of qualifiers and name patterns as parameters and returns the array list of organizations matching the request.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 31. Set details of a new organization in the registry*

```
// create organization name and description
 Collection orgs = new ArrayList();
TelephoneNumber telephoneNumber = 61-3-9995555
Organization org = blm.createOrganization(blm.createInternationalString("School of Networking"));

            org.setDescription(blm.createInternationalString("teaching and research service"));

Service service = blm.createService(blm.createInternationalString("The Government Service"));
service.setDescription(blm.createInternationalString("Services of the State Government"));

User user = blm.createUser();
PersonName personName = blm.createPersonName("Great Teacher");

 org.setPrimaryContact(user);

 TelephoneNumber telephoneNumber = blm.createTelephoneNumber();
 telephoneNumber.setNumber("61-3-555 4444);
 telephoneNumber.setType(null);

PostalAddress address = blm.createPostalAddress("9999", "McStreat Road", "Frankston", "Victoria",
"Australia", 9999", "");
Collection postalAddresses = new ArrayList();
postalAddresses.add(address);

Collection emailAddresses = new ArrayList();
EmailAddress emailAddress = blm.createEmailAddress("mycampuss@monash.edu.au");
emailAddresses.add(emailAddress);

Collection numbers = new ArrayList();
numbers.add(telephoneNumber);
user.setPersonName(personName);
user.setPostalAddresses(postalAddresses);
user.setEmailAddresses(emailAddresses);
user.setTelephoneNumbers(numbers);
```

The JAXR provides support for selected classification schemas:

- The North American Industry Classification System (NAICS) (http://www.census.gov/epcd/www/naics.html)
- The Universal Standard Products and Services Classification (UNSPSC) (http://www.eccma.org/unspsc/)
- The ISO 3166 country codes classification system maintained by the International Organization for Standardization (ISO) (http://www.iso.org/iso/en/prods-services/iso3166ma/index.html).

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 32. Adding classifications to the Concept*

```
ClassificationScheme scheme = null;
Collection schemes = new ArrayList();
  while(nameIter.hasNext()) {
    try {
          schemeName = (String) nameIter.next();
          scheme = busQueryManager.findClassificationSchemeByName(null,
schemeName);

Collection classifications = new ArrayList();
Classification.add(wsdlSpecificationClassification);
Classification.add(industryClassification);
Concept.addClassifications(classifications);
```

The task is to create classification for our service (*wsdlSpec*); find industry classification and add your type to it; create an external link for the WSDL document location; and add industry classification and service classification to the Concept (Figure 32).

Sun Microsystems has recently released Web Services Developers Pack 1.6 (WSDP 1.6), which includes implementation of UDDI version 2 to provide a private registry environment for testing and application development. The registry is composed of two parts: a Web application and a servlet that implement UDDI 2 functionality and a database based on the native XML database Xindice[3]. The WSDP 1.6 also provides scripts for adding and deleting users. More information is provided in the UDDI project site http://www.uddi.org/, JAXR home page http://java.sun.com/xml/jaxr/, J2EE 1.4 Tutorial http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html, and Java WSDP http://java.sun.com/webservices/webservicespack.html. The Java WSDP also provides a working example of a registry browser for finding organizations by name or classification and adding organizations and services to organizations.

## Simple Object Access Protocol (SOAP)

SOAP is a communication protocol and stands for Simple Object Access Protocol. SOAP allows Java objects and COM objects to talk to each other in a distributed, decentralized, Web-based environment.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Many applications communicate using Remote Procedure Calls (RPC) between objects within the middleware layers such as DCOM and CORBA. CORBA and RMI communication systems are based on messaging protocols GIOP/IIOP and RMI/IIOP. HTTP was not designed to support complex communication protocols. Firewalls and proxy servers will normally block HTTP traffic. HTTP is supported by all Internet browsers and servers, and SOAP was created to provide a way to communicate between applications running on different operating systems with different technologies and programming languages.

The main characteristics of SOAP are:

- It supports communication between applications communicating over the Internet;
- It defines a format for sending messages;
- It is based on XML, and it is platform independent and language independent;
- It allows the messages to get through firewalls; and
- It has been developed as a W3C standard.

A SOAP message is an XML document exchanged between a SOAP client and a SOAP server. The SOAP message contains four elements (some of them are optional):

- *Envelope* element that identifies the XML document as a SOAP message (required element);
- *Header* element that contains header information (optional element);
- *Body* element that contains call and response information (required element); and
- *Fault* element that provides information about errors that occurred while processing the message (optional element).

All the elements in the previous list are declared in the default namespace for the SOAP envelope defined at the W3C Web site, http://www.w3.org/2001/12/soap-envelope. The default namespace for SOAP encoding and data types are also available from W3C at http://www.w3.org/2001/12/soap-encoding.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 33. Skeleton SOAP message*

```
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>
  ...
  ...
  </soap:Header>
    <soap:Body>
  ...
  ...
      <soap:Fault>
    ...
    ...
      </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

The SOAP message must not contain any reference to DTD or XML processing instructions. A skeleton SOAP message encoded in XML and containing envelope, header, and fault elements is shown in Figure 33.

Web Services rely on a three-tier protocol stack: TCP/IP, HTTP/HTML, and XML. The first tier, the TCP/IP protocol, is concerned with passing data across the wire in packets. TCP/IP protocol guarantees reliable transport transmission and physical connectivity across public networks. It is a backbone protocol of the Web on which higher-level protocols such as HTTP rely. The second tier, HTML over HTTP, is the presentation tier supporting browser-based processing and sharing of information. The tier focuses on the manipulation and presentation-based navigation. HTML lacks both extensibility and programming power.

SOAP allows building applications by remotely invoking methods on objects. Being a platform independent protocol, SOAP removes the need to invoke methods through proprietary protocols, the requirement that two applications must run on the same platform, and the requirement that the applications are written in the same programming language. SOAP provides XML, a text-based syntax for invoking method calls.

To execute the exchange of SOAP messages between two applications we need two components: a SOAP client and SOAP server. The SOAP client is a program that creates an XML document containing the information needed

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

to invoke a method - remotely on a server. SOAP clients can be traditional clients (thick clients), a Web server, or a server-based application.

Messages and requests from SOAP clients are sent over HTTP. A SOAP server is a program that listens for SOAP messages, interprets them, and distributes them to respective Web Services. External Web Services may interact with application servers based on J2EE technology, which processes SOAP requests from a variety of clients.

SOAP servers ensure that documents received over an HTTP connection are converted to a language that the objects at the other end understand. Because all communications are made in the form of XML, objects in one language (for example Java) may communicate through SOAP with objects in any other language (for example C++).

We will now look at the role of the SOAP client and SOAP server in a simple request/response transaction over HTTP:

- **Locating the Service:** A SOAP client uses the UDDI registry to locate a Web service. Typically, a SOAP application has prior knowledge of a particular type of port and style of binding, and it dynamically configures the address of the service to be invoked to match the ones discovered through UDDI. This approach is considered more practical than one that manipulates WSDL directly.
- **Application Client Builds the Request:** The client application builds a SOAP message as an XML document capable of performing the desired request operation.
- **Web Server and SOAP Server:** Web server is listening to SOAP requests. The client sends the SOAP message to a JSP or ASP page on the Web server. The SOAP server parses the SOAP documents, invokes the appropriate methods of the called object, and passes the parameters included in the SOAP document to the appropriate methods. Optionally, SOAP headers may indicate to intermediate nodes which functions should be performed prior to the SOAP server receiving the message.
- **Forming Response:** The request object performs the indicated function, formulates the response, and returns data to the SOAP server. The SOAP server packages the response in a SOAP envelope and then wraps the SOAP envelope in a response object, such as a servlet or a COM object, which is sent back to the requesting client.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- **Response Processing by the Application Client:** The SOAP client receives the object, strips off the SOAP envelope, and sends the response document to the program that requested it. The request/response cycle is now completed.

The developers should consider the role of SOAP services to see if they ease server-side programming. The advocates of SOAP services argue that an enterprise application server should be seen as a collection of services (or transactions). These services are conveniently listed in registries for browsing, searching, and access. Services as single units of transactions can be relocated, load-balanced, replaced, security-applied, and so forth. The assumption is that designing an application server as a collection of services that are easily located and accessed will encourage developers to use better design practices and focus more closely on service-based development methodology.

The opposition developer community sees SOAP services as another attempt to bring CORBA and COM back for wider use. They argue that SOAP messaging requires a lot of work. SOAP invocation style is seemingly less complex than CORBA calls. Many already running Java applications use polymorphism (method overload and overwrite). Such applications are not suitable for SOAP messaging without removing polymorphic methods. The other argument is that the benefits of UDDI are overstated as far as a business differentiation and sales tools.

## What is URI, URL, URN?

JPA URI can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URIs that, in addition to identifying the resource, provides a means of locating the resource by describing its primary access mechanism (for example, its network "location"). The term "Uniform Resource Name" (URN) refers to the subset of URIs that is required to remain globally unique and persistent, even when the resource ceases to exist or becomes unavailable.

An individual scheme does not need to be cast into one of a discrete set of URI types such as "URL," "URN," "URC," and so forth. Any given URI scheme may define subspaces that have the characteristics of a name, a locator, or both, often depending on the persistence and care in the assignment of identifiers by

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

the naming authority, rather than on any quality of the URI scheme. For that reason, the specification deprecates the use of the terms URL or URN to distinguish between schemes, and instead uses the term URI throughout.

The definition from Webopedia (http://www.Webopedia.com/TERM/U/URI.html) states:

*"Short for Uniform Resource Identifier, the generic term for all types of names and addresses that refers to objects on the World Wide Web. A URL is one kind of URI."*

Although many URI schemes are named after protocols, this does not imply that use of such a URI will result in access to the resource via the named protocol. URIs are often used in contexts that are used purely for identification, just like any other identifier. Even when a URI is used to obtain a representation of a resource, that access might eventuate through other components such as gateways, proxies, and caches. Furthermore, it is possible that the name resolution services independent of the protocol of the resource origin are used, and the resolution of some URIs may require the use of more than one protocol (e.g., both DNS and HTTP are typically used to access an "http" URI's resource when it cannot be found in a local cache).

A parser of the generic URI syntax is capable of parsing any URI reference into its major components; once the scheme is determined, further scheme-specific parsing can be performed on the components. In other words, the URI generic syntax is a superset of the syntax of all URI schemes.

## Endnotes

1 Technical Note is a document published by the OASIS UDDI Specifications Technical Committee that provides guidance on how to use UDDI registries.

2 Please note that we have omitted the concept of <identifierBag> and <categoryBag>.

3 Xindice is part of Apache XML project.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

# Section II:

# Web Services as Shared Resources

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter IV

# Web Services

Web services aim to provide application-to-application interoperability. Messages are exchanged between two parties called service provider and service 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 the 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 the responsibility of the service providers.

According to W3C, 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.*

A closer look at the definition and some investigation of the actual "plumbing" gives you a simpler view of a Web service; it is a software application, accessible over the Internet or Intranet through a URL. Web service clients use XML-based protocols, such as the SOAP to transfer data over HTTP. The

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

access to this Web service application is obtained through the interfaces and bindings defined with the XML based Web Services Description Language (WSDL).

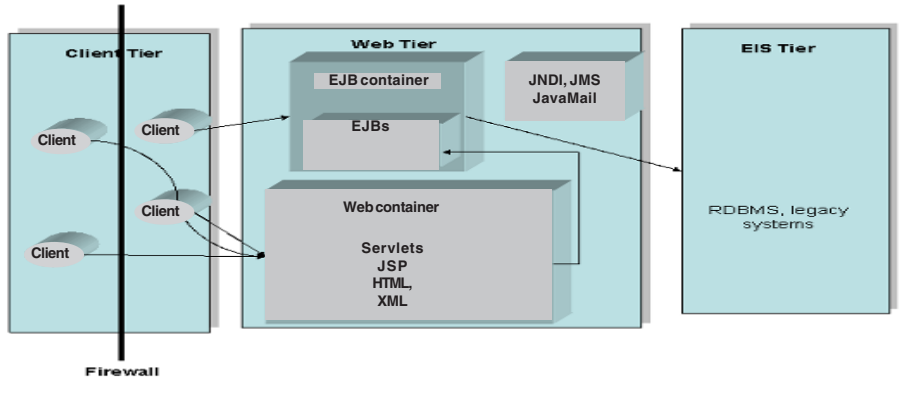
Traditionally, Web services are data oriented. Data is extracted from a service provider in a raw format, and then moved to the requestor in a message. On the other hand, with the invention of portals, there is a requirement for the provision of presentation oriented Web services. Presentation oriented services contain user interfaces, and the data is presented to the end user in the form of a remote portlet (Web Services for Remote Portlets [WSRP], 2003).

## Multi-Tier Component-Based Architecture

The J2EE platform supports a multi-tier distributed application model. A typical structure defines a client tier, a middle tier, and a back-end tier (Figure 34). The client tier is expected to support a variety of client types inside and outside 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).

In order to understand Web service concepts as well as the structural composition of Web service endpoint implementations with J2EE, we need to explain briefly J2EE as a component-based development model. This model

*Figure 34. J2EE multi-tier architecture*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

relies on the notion of containers. Containers are runtime environments that provide specific services to components. These services must be available to any J2EE component, on any platform, from any vendor. All containers support the lifecycle of the components living in these containers, such as the initialize, invoke service, and destroy methods. Web containers provide runtime processing of client requests and perform functions such as request-time processing, invocation of JSP pages, or servlets. They return responses to the clients. EJB (Enterprise Java Beans) containers support transactions and persistency frameworks, bean lookup and other services. Containers also provide access to the enterprise information systems, for example, access to relational databases through the JDBC APIs.

The configuration of containers is not confined to a single platform. Web containers can live on one platform and EJB containers on another, or a J2EE platform can be made up of multiple containers on multiple platforms.

Web service behavior and configuration is given by the WSDL descriptors. These descriptors expose the interface and represent the behavior of the resource on the Web. There are multiple ways of binding the Web service to resources. The Web service becomes a collection of endpoints bound to the same resource.

The Web Services can use two messaging models:

- **Synchronous Model:** This model represents a simple request-response operation. The client accesses a Web service (request) and receives a SOAP message in response. The client has to wait for the response. The communication is typically RPC-oriented Java APIs processing XML documents.
- **Asynchronous Model:** The client invokes the Web service, but it does not wait for the response. The Web service receives the request and then uses a messaging provider (server) to send the results to the client. The asynchronous sending of messages requires a messaging provider. Asynchronous receiving also requires an application server. An example of asynchronous receive support is *Java API for XML messaging* (JAXM).

The synchronous model is suitable for transactional processing. Furthermore, the synchronous model is more suitable when an application performs a synchronous transactional access to one or more back-end systems (EIS, legacy systems). Although there is some overhead (in particular for secure

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

transactions) and tighter coupling between the back-end system and the client, the synchronous model provides a higher quality of processing and guarantees transaction data integrity.

Asynchronous messaging is typically chosen for better throughput and quality of services because it buffers messages, supports message routing, and guarantees message delivery. From the programming point of view, an asynchronous model is more complex than the synchronous request-response model. The asynchronous model provides more services, however at a cost of application complexity and more work on the part of developers.

We should mention that the asynchronous model can be successfully simulated by using an on-demand protocol, where the request is sent to the service provider, and, later, another request is sent to obtain results of the first request's processing. The practicality of this approach is very much dependent on the nature of processing. For example, you can update multiple single rows in a remote database, and later, at the right moment, check the status of all updates in one transaction.

## Web Service Architecture

A Web service relies on the exchange of XML documents between client and server and 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 an XML document using return values from the methods; and
- Finally, it returns the XML response to the client.

The support for developing Web Services on the Sun platform is primarily based on the implementation of Java APIs for XML and Web services. The following is a brief overview of some API packages available on the J2EE platform.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

We have already discussed JAXP (API for XML processing) in Chapter II (XML Authoring and Java Parsers). JAXP 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 implementation 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 the RPC (Remote Procedure Call) programming model and supports the development of Web service clients and endpoints. We will provide more details in following sections.

SAAJ stands for SOAP with Attachments API for Java. It enables developers to construct and accept messages conforming to the SOAP specification. These APIs are derived from the java.xml.soap package.

JAXB is a Java technology that enables generating Java classes from XML schemas by means of a *JAXB binding compiler*. The JAXB binding compiler takes XML schemas as input, and then generates a package of Java classes and interfaces that reflect the rules defined in the source schema. These generated classes and interfaces are compiled and combined with a set of JAXB utilities to provide a *JAXB binding framework*.

JAXM (Java API for XML Messaging) defines messaging service. These APIs support sending and receiving document oriented XML messages.

## Web Services with JAXB

JAXB is a set of APIs representing the binding compiler, which generates Java classes that map to constraints in the source XML schema (marshaling and unmarshaling).

The process of creating XML documents for Web services and binding them to the underlying runtime infrastructure is depicted in Figure 35. The first step in creating XML documents is to define the appropriate XML schema. The purpose of the XML schema is to define a class of XML documents, their structural rules, and constraints. The documents that adhere to the given XML schema can be processed by the application. The J2EE platform provides the JAXB compiler, which binds Java classes and packages to the XML source

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

documents. The developers can follow the set of default binding rules, which can be customized (Binding customization) if required.

The next step is to run the *JAXB binding compiler* (Figure 35) using the XML schema as input. The binding compiler generates Java classes that map to constraints in the source XML schema.

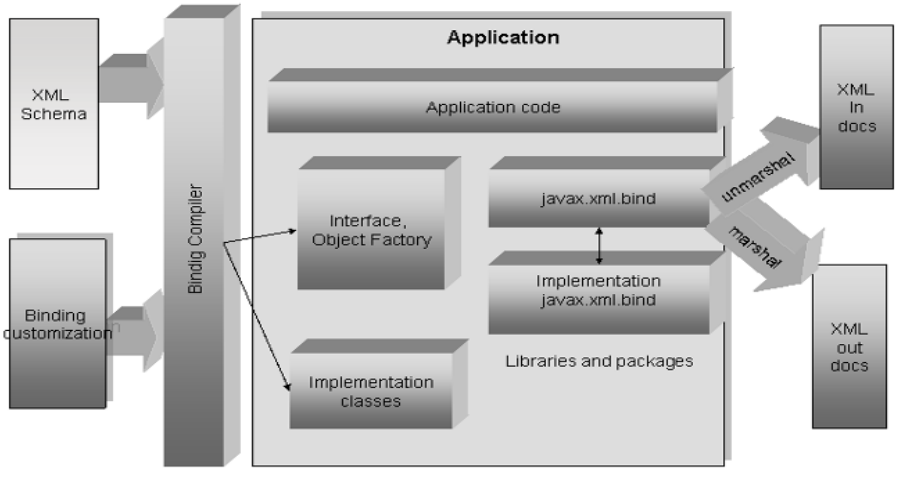
Implementation of javax.xml.bind package is a runtime API that provides interfaces for unmarshalling, marshalling, and validating XML content in a Java application. *Application code* is a client Java application that uses the JAXB binding framework to unmarshall XML data, validate and modify Java content objects, and marshal Java content back to XML data.

*XML input documents* (XML in docs) are instances of an XML document, from which a Java representation in the form of a content tree is generated. The unmarshalling process supports *validation* of the XML input document against the constraints defined in the source schema.

*XML output documents* (XML out docs) refer to the marshalling process of an instance of an XML document. This involves parsing the XML content object tree and writing out an XML document that is an accurate representation of the original XML document and is valid with respect to the source schema.

*Interface Object factory:* A client application may also generate Java content trees explicitly rather than unmarshalling existing XML data. The application must have the access to each of the schema-derived ObjectFactory classes. For each schema derived Java class, there will be a static factory method that produces objects of this particular type.

*Figure 35. JAXB components and cooperation*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Typically, the JAXB binding framework is wrapped in a larger Java application that may provide UI features, XML transformation functions, data processing, or whatever else is desired.

The following is an example. A Web service is provided to obtain, store, and retrieve a person's first name and surname. Incoming (unmarshalled) and outgoing (marshalled) documents are given a formal specification. There are two methods available: XML schema (.xsd file) or DTD schema (.dtd file). The DTD method has been used for some time. The DTD format is good at specifying document structure. However, there are some shortcomings when you need to specify data types. On the other hand, XML Schema is good at specifying data types, but lacking in structural specifications. The first step in the example is to create an XML schema (Request.xsd).

This schema contains one top-level element and a complexType declaration composed of a sequence of elements: name and surname. We have already indicated that there are a number of ways of handling XML parsing in Java: JAXP, JDOM, JAXM, JAXR, and JAXB. In our example, we will be creating Java objects from an XML document called Person.xml (Figure 37). Person.xml (XML request) describes the process called unmarshalling. The binding compiler is JAXB/_xjc.

*Figure 36. Request.xsd (XML schema)*

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="Personrequest" type="PersonRequestType"/>

<xsd:complexType name="PersonRequestType">
   <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="surname" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

*Figure 37. Person.xml (XML request)*

```
<?xml version="1.0"?>
<Personrequest>
<name>Jane</name>
<surname>Smith</surname>

</Personrequest>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 38. Example of build.xml file*

```
<?xml version="1.0" standalone="yes"?>
<!--
  build.xml for Personrequest.xsd.
-->
<project basedir="." default="run">
  <!--  -->
  <property name="jwsdp.home" value="../../.." />
  <path id="classpath">
    <pathelement path="src" />
    <pathelement path="classes" />
    <!--for use with bundled ant-->
    <fileset dir="${jwsdp.home}" includes="jaxb/lib/*.jar" />
    <fileset dir="${jwsdp.home}" includes="jwsdp-shared/lib/*.jar" />
    <fileset dir="${jwsdp.home}" includes="jaxp/lib/**/*.jar" />
  </path>
  <taskdef name="xjc" classname="com.sun.tools.xjc.XJCTask">
    <classpath refid="classpath" />
  </taskdef>

  <!--compile Java source files-->
  <target name="compile" description="Compile all Java source files">
    <echo message="Compiling the schema..." />
    <mkdir dir="gen-src" />
    <xjc schema="Request.xsd" package="person" target="gen-src">
      <produces dir="gen-src/person" includes="**/*.java" />
    </xjc>
    <echo message="Compiling the java source files..." />
    <mkdir dir="classes" />
    <javac destdir="classes" debug="on">
      <src path="src" />
      <src path="gen-src" />
      <classpath refid="classpath" />
    </javac>
    <copy todir="classes">
      <fileset dir="gen-src">
        <include name="**/*.properties" />
        <include name="**/bgm.ser" />
      </fileset>
    </copy>
  </target>

  <target name="run" depends="compile" description="Run the sample app">
    <echo message="Running the sample application..." />
    <java classname="TestRequest" fork="true">
      <classpath refid="classpath" />
    </java>
  </target>
```

To simplify the use of the JAXB binding compiler from the command line, we have also created the build.xml file and used services of the apache ant language together with the predefined projects (refer to http://java.sun.com/xml/jaxb/) (Figure 38).

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 38. Example of build.xml file (cont.)*

```
  <target name="javadoc" description="Generates javadoc" depends="compile">
    <echo message="Generating javadoc..." />
    <mkdir dir="docs/api" />
    <javadoc sourcepath="gen-src" destdir="docs/api" windowtitle="Using
unmarshaller (formerly SampleApp1)" useexternalfile="yes">
      <fileset dir="." includes="gen-src/**/*.java" excludes="**/impl/**/*.java" />
    </javadoc>
  </target>

  <target name="clean" description="Deletes all the generated artifacts.">
    <delete dir="docs/api" />
    <delete dir="gen-src" />
    <delete dir="classes" />
  </target>
</project>
```

As the last step, you also need to design your test program (Figure 39).

The output is in Figure 40.

The xjc binding compiler is run on the Person.xml schema to bind the schema to Java classes. Figure 43 shows generated classes including interface (Personrequest.java), and Objectfactory (ObjectFactory.java). Some of the XML constructs are not supported by JAXB. For example, a schema element such as xs:any, xs:key, xs:unique and schema attributes such as complexType.abstract, complexType.final, and others, are not supported.

## Web Services with JAX-RPC

JAX-RPC (http://java.sun.com/xml/jaxrpc/overview.html) is used for Web services interoperability across heterogeneous platforms and languages. This makes JAX-RPC a suitable technology for Web Services integration.

You can use the standard JAX-RPC programming model to develop Web service clients and endpoints based on SOAP. A Web service endpoint is described using a Web Services Description Language (WSDL) document. JAX-RPC enables JAX-RPC clients to invoke Web services developed across heterogeneous platforms. In a similar manner, heterogeneous clients can invoke JAX-RPC Web service endpoints. JAX-RPC requires SOAP and WSDL standards for this cross-platform interoperability.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 39. TestRequest.java to test the example*

```
/**
 * TestRequest.java
 */

import java.io.*;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;

import person.*;  //here are the types required by the compiler

public class TestRequest{

   public static void main(String[] args){
         new TestRequest();
   }

   public TestRequest() {

         InputStream in = null;
         try {
            in = new FileInputStream(new File("Person.xml"));
         } catch(java.io.FileNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
         }

         Personrequest request = null;
         try {
         // person is the package name used in xjc compiler
         JAXBContext jc = JAXBContext.newInstance("person");

         Unmarshaller u = jc.createUnmarshaller();

         request = (Personrequest) u.unmarshal(in);

         } catch(JAXBException e) {
            e.printStackTrace();
            System.exit(1);
         }

         System.out.println("Name is " + request.getName());
         System.out.println("Surname is " + request.getSurname());

         System.exit(0);
   }
} // TestRequest
```

*Figure 40. Unmarshalled documents*

```
 run:
Running the sample application...
Name is Jana
Surname is bundasek1

BUILD SUCCESSFUL
Total time: 1 second
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 41. JAXB — Generated classes*

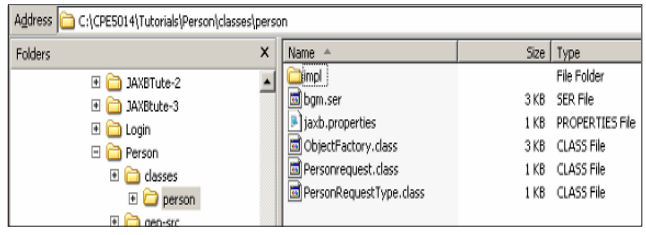



*Figure 42. JAXB Source classes (.java)*

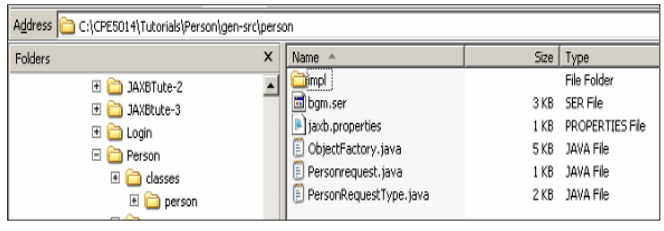
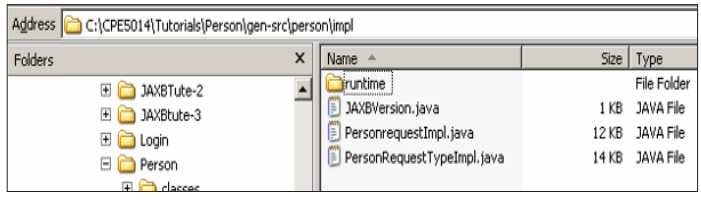



*Figure 43. Generated implementation classes*

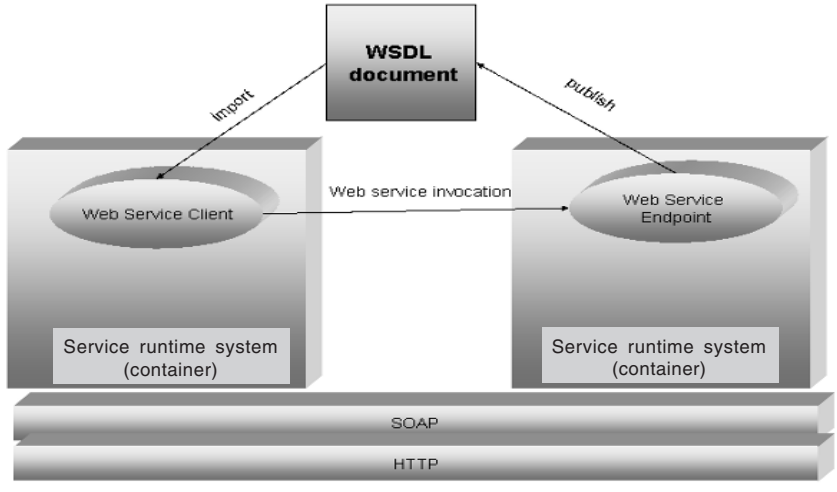



JAX-RPC provides support for two types of mapping: WSDL-to-Java and Java-to-WSDL. The mapping and provision of tools for mapping is part of the development environment for designing Web service clients and endpoints.

As previously mentioned, Web Services run on request-response mechanisms. In the typical Web service scenario, a client request starts a 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

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 44. JAX-RPC publish, import, and invoke principle*




by a Java method invocation and transmission of the required parameters. The response is the result of this method invocation. In Figure 45, a JAX-RPC client uses SOAP message to request services from a J2EE service implementation.

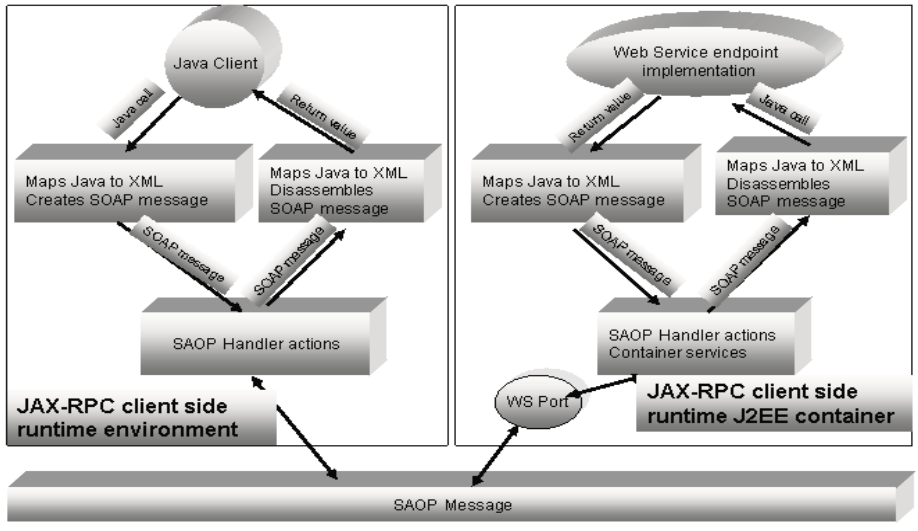
The first step for the client is to locate the service using a registry[1]. Once the Web service location is found, the client discovers the options to access it and binds to the service. The client makes a request to the service as a simple method invocation with parameters. There are three possible methods for how to make the call: invoking methods on generated stub, using dynamic proxy, and using dynamic invocation interface (DII). JAX-RPC supports only a limited subset of Java types[2] to ensure interoperability with Web Services implemented in other languages. The basic rule is that all types must be serializable to and from an equivalent XML document representation form.

*JAX-RPC runtime environment* maps (Figure 45) the call to a XML document and creates a SOAP message that encapsulates the method call and its parameters. The SOAP message is passed to the SOAP handler and transferred over a standard TCP/IP connection to the server side (*WS port* in Figure 45 means Web service port).

*The JAX-RPC framework* (JSR 101) does not specify what particular means for Java object-XML mapping are to be used for serializing and de-serializing. When the SOAP message reaches the Web service port, the J2EE container applies its declarative service to the SOAP message. On the server side, the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 45. Structure of Web Services with J2EE platform*




SOAP handler processes the request. The SOAP handlers must be written by developers, and they are free to apply application-specific processing logic for this particular service endpoint.

The service endpoint extracts the method call, performs any *XML-to-Java* object mapping, and passes the method to the Web service interface implementation for processing. Similar steps are performed when the Web service returns a response to the client.

JAX-RPC Web service clients can be designed as thick clients in standalone J2SE application fashion or as J2EE (server) components. The following steps detail the implementation with generated stubs that was discussed earlier in this section (see Figure 46).

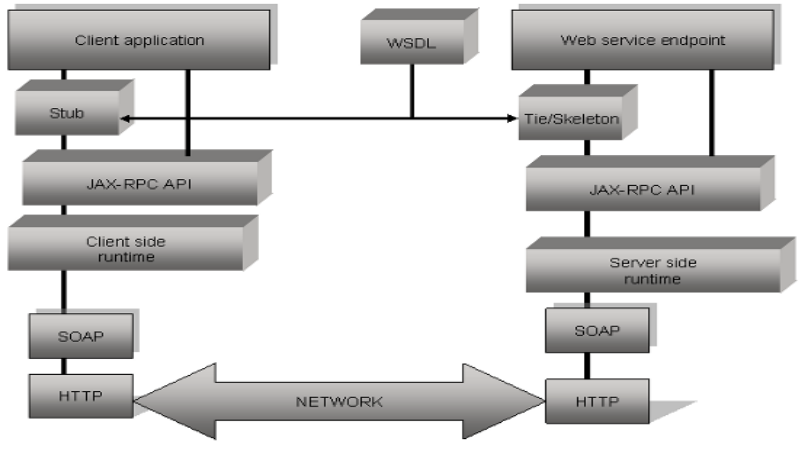
The client makes the request with the following RPC calls:

A client application makes an RPC call on a local object called a ***stub***.

- The stub converts the RPC call to JAX-RPC runtime system requests using JAX-RPC API.
- The JAX-RPC runtime system (client side runtime) maps the requests to a SOAP message and transmits it as part of an HTTP request.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 46. JAX-RPC client-server components*




The request is received on the server side as a SOAP message with RPC calls:

- The JAX-RPC runtime system extracts the SOAP message from the HTTP request and maps it to a method call on a local object called a tie or skeleton.
- The tie invokes the method call on the Web service.
- The Web service processes the request.

Note that, in the J2EE platform, the business logic for the Web service could be in a Web tier component such as a servlet, or in an Enterprise JavaBeans component in the EJB tier. Obviously, it can be part of a portlet, requesting data from the remote Web service.

The service formulates the response with RPC response processing on the server:

- The Web service processes the request and returns the response.
- The tie converts the response to JAX-RPC runtime system response.
- The JAX-RPC runtime system maps the response to a SOAP message and transmits it as part of an HTTP request.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The response is received on the client side with RPC response processing:

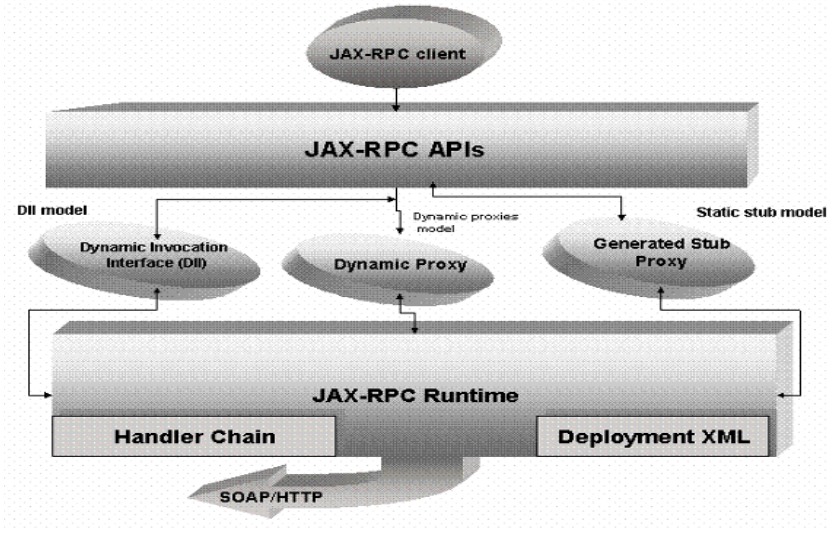
- The JAX-RPC runtime system extracts the SOAP message and maps it to a response on the stub.
- The stub returns the response to the client application.
- The client application receives and processes the response (displays the response in a browser or any other GUI).

## JAX-RPC Invocation Models

To make a Web service available to clients through JAX-RPC, developers must provide a JAX-RPC service endpoint definition and an 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 (see Figure 47). These models are independent of any specific service implementation model.

- **Stub-base model (static stubs):** Service stubs can be created either from the Java service endpoint interfaces or from the WSDL description of the service (the stub class can be generated using either the WSDL2Java

*Figure 47. JAX-RPC invocation models*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

tool from IBM WebSphere application server tools or the wscompile tool from Sun Microsystems Application Server 8, or any other mapping tool provided by the server container vendor). The stub interface provides APIs to configure stubs by setting properties such as the endpoint address, session, user name, and password.

- **Dynamic proxies:** The client at runtime creates dynamic proxy stubs (javax.xml.rpc.Service interface). In this case, the client must have prior knowledge of the WSDL service description and service details such as parameters and service name, and so forth. Using the ServiceFactory class (a known pattern), the client creates the service and obtains the relevant proxy.
- **Dynamic Invocation Interface (DII):** A DII client discovers service details (name and parameters) at runtime using either the service broker or lookup services, depending on the actual application server implementation. Service brokers can use varying service discovery mechanism such as JINI (http://www.jini.org), ebXML, UDDI, etc. In this case, the binding with the Web service is dynamic.

## JAX-RPC Implementation Considerations

Service endpoint development with JAX-RPC needs some further considerations. Coding is based on the RPC paradigm, but even if you are familiar with client server programming with CORBA, the implementation is not straightforward. Some important notes are in subsequent paragraphs.

*Packaging* and removing requires some programming effort: The client and server are tightly coupled entities. The programming effort to develop the client could be extensive. Therefore, we believe that client developers and service endpoint providers should not operate independently. When it is expected that the target clients operate on JAX-RPC, the service endpoint developers should provide prepackaged libraries and server-side documentation to ease the programming burden of the client's development. The provided libraries should contain all files, such as the client-side representation of the service endpoint interface and its dependent classes that are necessary for accessing the service. The library should also include additional presentation interface classes to simplify development of the client's view of the service and shield the client from maintenance changes to the service.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

**Programming rules.** To expose a Java-based service as a Web service, the service endpoint interface is typically defined first. The developers must be reminded that the following coding rules apply:

- All methods must throw RemoteException; omission may lead to difficulties in debugging, mostly on the client side;
- No *public static final* modifiers can be defined on service endpoint interface; and
- Developers should use only those method parameters and data types supported by the JAX-RPC types.

**Dealing with the state.** Web Services are stateless and therefore the service endpoint should be designed as stateless to allow maximum interoperability. This concept is similar to HTTP Servlets (see details in Chapter V: Servlets). That means that a Web service receives the request, generates the response, and then is free to service another client's request. There is no guarantee for the state to be preserved on the client side. However, often the business process represented by the Web service is not a simple request-response type, but a collection of request-response tuples with some dependencies. These situations should be resolved as a stateful Web service.

To design a stateful Web service, we need to utilize features of the underlying communication transport, in particular HTTP headers (see Chapter VII Web Service Design). The session-related information can be embedded in the HTTP header and the Web service client runtime must make sure that this information is passed with the HTTP request object (this is also called URL rewriting). The Web server runtime reads the information and associates it with the state maintained on the server side for the particular client session.

The Web Services implemented as JAX-RPC are means for invoking some processing on a remote Web server, with data returned to the requester, using the remote procedure call paradigm (RPC). They are not dissimilar to the older CORBA model. The development of a smoothly running client with Web service requires a cooperative effort by developers on both sides — client and server. It is possible to find the interface definition and the implementation details published in well-known locations and in well-known formats. The vehicle that allows this freedom is the XML definition (XML schema and XML documents). Managers embarking on the development of a client or server

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

should be aware of technologies used to implement this solution and enforce proper coupling of the technology to generate the required solution.

## Web Service Discovery and Invocation

Web Services and their clients can be built using J2EE components (for example, JAX-RPC and JAXB are very common solutions for client-server models. With JAX-RPC, the developers specify 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 a Web service endpoint, which must adhere to the previously defined interface. The Web service endpoint is a set of methods that implement the business logic. A client accesses the Web service using a Service endpoint interface.

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.

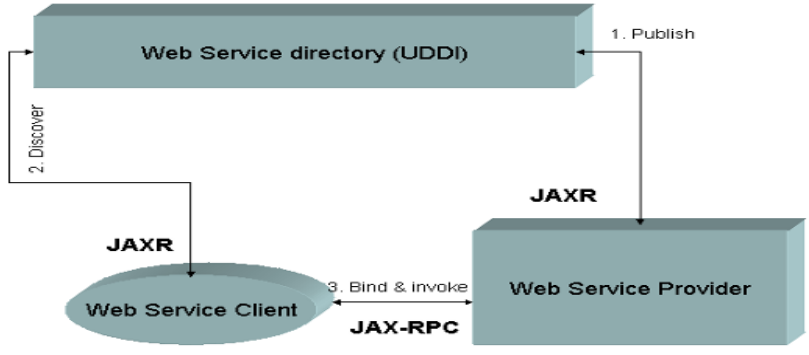
In order to handle communication between a client and a service endpoint, the service endpoint must also define interfaces and a proxy object (called tie or skeleton). The access to the service endpoint implementation is invoked by the server passing the incoming message to the tie object. The stubs and skeletons on the client and server side are commonly called artifacts. It is typical that the vendor who provides the runtime implementation of the connections 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.

The J2EE platform provides tools to quickly build, deploy, and test Web Services; it also supports the development and deployment of clients to Web Services that interoperate with other Web Services and clients. The basic mechanism for publishing and discovering Web Services is in Figure 48.

Regardless of whether the Web service will be accessible to a single enterprise or to other companies (public access), the details about the service (its

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 48. J2EE publish-discover-invoke mechanism with JAX-RPC*




interface, parameters, location, and so forth) must be made available to interested clients or other Web services. This is accomplished with a WSDL description of the Web service and a Web service directory, where the details of the Web service are published.

**Publishing Web service (step 1).** In order to be accessible to interested parties (Web service providers), the Web service is published in a Registry or Web service directory. There are several choices regarding where to publish a Web service:

- If the Web service is intended for the general public, then a well-known registry is recommended (for example UDDI). Such publishing makes the service available to a wide audience. The consequence is that not only the WSDL description is made public, but also any XML schemas referenced by this description are now accessible to everybody.
- The Web service intended for enterprise use over an intranet should be published in a corporate registry only. No access from outside the firewall is required.
- Finally, providing all clients are dedicated partners in business and there is an existing agreement on usage of this service, the Web service can be published on a well-known location on a company server with proper security access protection. Such a server would be placed on the public side of the company firewall, but it would allow limited access, similar to a B2B Web server.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Registries (Web Services directories) are made up of a repository and the taxonomies (classification of registered entities for easier search) associated with them. There are no restrictions on publishing the Web service in multiple registries, even in multiple categories. Some restrictions, therefore, apply for usage of a registry:

- A registry, in particular a public one, cannot be used as a data repository. It contains only details of what services are available and how to access them.
- All registries typically use some taxonomy to classify the registered services. It is important to register the Web service in an appropriate category to make it easier for clients to find and use the service.

**Discovery of service (step 2).** Registry implementations can differ, but there are some common steps that the client must perform before it can discover and bind (step 3) to the service:

- The client must determine how to access the service's methods, such as determining the service method parameters, return values, and so forth. This is referred to as *discovering the service definition interface*.
- The client must locate the actual Web service (find its address). This is referred to as *discovering the service implementation*.

**Bind to the Web service and invoke it (step 3).** The client must be able to bind to the service's specific location. The following types of binding may occur:

- Static binding during client development or at the deployment time, and
- Dynamic binding (at runtime).

From the client's point of view, the binding type and time play important roles in possible scenarios relevant to the client's usage of the Web service. The following situations are typical:

- A Web service (WSDL and XML schemas) is published in well-known locations. The developers of the application that use the service know the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

service, its location, and the interface. The client, which is a process running on a host, can bypass the registry and use the service interfaces directly. Alternatively, the client knows the location and can statically bind to the service at the deployment time.

- The Web service expects its clients to be able to easily find the interface at build time. These clients are often generic clients[3]. The Web service must be registered in a registry. Such clients can dynamically find the specific implementation at runtime using the registry. Dynamic runtime binding is required.

Development of Web service clients requires some rules to be applied and design decisions to be made regarding which binding type is more appropriate for the given situation (static or dynamic binding). Three possible cases are discussed here (details relevant to the JAX-RPC style of Web service use are in Figure 47):

- **Discovering the Service Interface Definition:** If we are dealing with a known service interface, and the service implementation is known (no registry is required), the actual binding should be static.
- **Discovering the Service Implementation:** In this case, static binding is also appropriate because we know the interface. We need to discover the service implementation only at build time.
- **The client does not know the service interface and needs to discover** the service interface dynamically at build time. The service implementation is *discovered dynamically at runtime*. This type of invocation is called Dynamic Invocation Interface (DII). In this case, the binding must be dynamic.

There are still some additional concerns about dynamic service discovery that are not discussed in this book. Web Services Dynamic Discovery defines a multicast discovery protocol to locate services. This latest development by Microsoft and BEA uses probes that are sent to a multicast group, and targets services that match return a response directly to the requesting client. To scale to a large number of endpoints, the protocol defines multicast suppression if a discovery proxy is available on the network. To minimize the need for polling, target services that wish to be discovered send an announcement when they join and leave the network.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

One of the problems is dealing with all of the different XML dialects used by the services. The service developers often use their own XML format in order to deal with their particular case, which the accepted standard does not address.

## XML Documents as Part of Web Services

Many Web service implementations expect an XML document containing client information relevant to the request. The service uses this document to start a process to handle the request. Sometimes the XML documents may be exchanged as part of the Web service. These documents arrive as parameters in the method calls. In the J2EE platform, there are three options for exchange of the XML documents: *Java-MIME mapping*, *String type documents*, and *SOAP fragments*.

**Java-MIME mapping.** The Web service endpoint receives documents as javax.xml.transform.Source objects. The package (javax.xml.transform.*) defines the generic APIs for processing transformation instructions, and performs the transformation. The interfaces in the package do not have dependencies on SAX or the DOM standards, and try to make as few assumptions as possible about the details of the source and result of the transformation. An object that implements this interface contains the information needed to act as the source input (XML source or transformation instructions). The application defines two interfaces: Source and Result interfaces. The XML document is passed as a Source object, and the container handles the document as an attachment.

Documents can be sent as *String types*: In this case, the WSDL is used to map the XML document. The following example shows how the XML is mapped into xsd:string type:

```
<part name="StudentEnrolmentXMLDoc" type="xsd:string"/>
```

The corresponding Java interface for the WSDL where the XML document (enrolDoc) is passed to the method submitEnrollmentForm() as a string is in Figure 49.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 49. Java to String type interface*

```
public interface StudentEnrolmentService extends Remote {
        public String submitEnrollmentForm(String enrolDoc)
                        throws RemoteException, InvalidFormException;
}
```

Passing a large document as a string could result in some performance penalties. You may expect that such a document can grow in size, resulting in the payload size of the message becoming too large. The other problem is the XML format; the XML document loses its original format after being converted into a string.

**XML document is a SOAP document fragment.** XML schema have a built in data type hierarchy. This hierarchy (Figure 50) starts with the top complex type: anyType. We may utilize this type hierarchy and map the XML document to xsd:anyType in the WSDL description of the Web service. Passing XML documents as SOAP document fragments provides portability across J2EE implementations and interoperability with all platforms. This solution requires the Web service to be implemented using the WSDL-to-Java approach. The WSDL description has the following format:

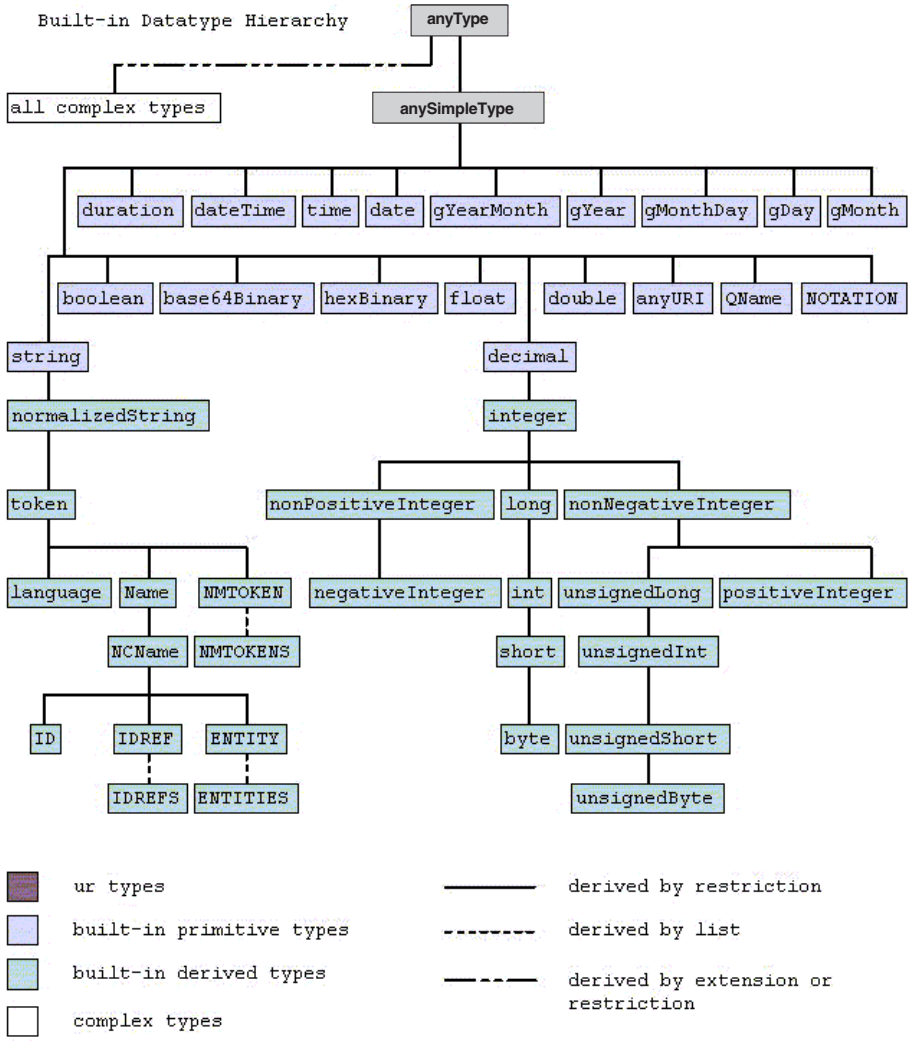
```
<part name="StudentEnrolmentXMLDoc" type="xsd:anyType"/>
```

In order to map the XML document type into xsd:anyType, the document parameter must be defined as a SOAPElement object. The service parses the SOAP document fragment using relevant APIs from the javax.xml.soap.SOAPElement package.

The WSDL-to-Java approach allows direct mapping of the exchanged document to its description (schema) in the WSDL. The corresponding Java interface represents the document as an instance of a Java Object. The service endpoint never sees the original document form. This approach results in the endpoint being tightly coupled with the document schema. Tight coupling typically results in some maintenance difficulties; any change in the document schema must be reflected in corresponding change to the endpoint. Therefore, using xsd:anyType to map the document as a loose coupling option may save some maintenance expenses.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 50. XML data types hierarchy (http://www.w3.org/TR/xmlschema-2/#date)*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

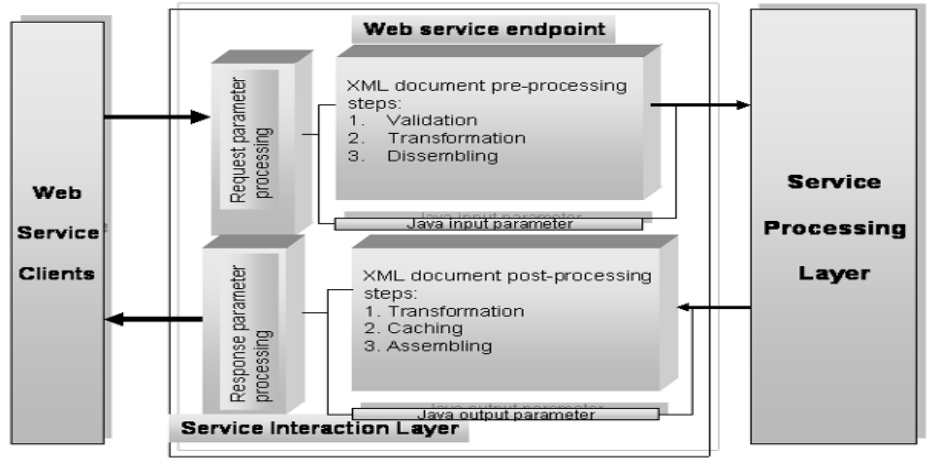
## XML-Based Web Service Endpoint Considerations

Let's assume that the XML document that is sent by the requestor and received at the provider contains some business logic. As a minimum, the Web service must "read" and/or "modify" the XML document. The separation of concerns in terms of separating document manipulation logic from the actual processing enables the developer to change or apply various document manipulation techniques. The Web service interaction layer serves as the first point of contact. Its purpose is to handle tasks such as accepting and handling incoming requests, invoking a security service, processing authentication, validating and transforming incoming XML documents into domain objects, deciding on business logic processing, and handling errors.

Figure 51 shows a layered architecture of a Web service endpoint, taking into account the separation of concerns at a Web service endpoint, responsibilities, and business logic processing.

**Request receiving considerations.** The service interaction layer receives the client's request in the endpoint layer. The requests are in the form of SOAP messages. The incoming SOAP message is mapped into method calls in the service interaction layer; the parameters are validated and, depending on the mapping method, transformed into Source objects or String types. The request is disassembled and further processing is delegated to the layer with the

*Figure 51. Web service request processing at the endpoint*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

business processing logic. The validation deals mostly with checking the incoming XML document against its schema.

Often, the business logic layer may have its internal XML schema. In these situations, the XML documents should be transformed into the internal schema before they are delegated to the business logic processing layer. Sometimes the service endpoint interface receives only XML documents, but the implementation requires objects to process the business requirements. In this case, the interaction layer must convert the incoming XML documents into appropriate objects before submitting the request to the business logic layer for processing.

**Service processing layer considerations.** As mentioned before, service processing can happen in a synchronous (the client blocks and waits for the response) or an asynchronous way (the client does not block but expects some notification about the request completion). When we know that the service processing is very short, we can opt for the synchronous processing model and let the client block. However, some requests require long processing, and the client response time would become unacceptably long. In the asynchronous model, the client can perform some other processing while waiting for the response.

The asynchronous client on the J2EE platform must use the Java Message Service (JMS) to receive notification that the response is ready to be delivered. There are several steps the asynchronous client must complete:

- The asynchronous client sends the request to the service interaction layer. The service interaction layer validates the request and ensures its completeness and transformation.
- The service endpoint returns a unique ID (sometimes called a *correlation ID*), and the transformed message is then passed to the service processing layer.
- The client can poll for the status of the message (periodically "enquire" about the status of the message).
- Alternatively, if the client itself is a Web service capable of receiving callbacks with results, the client implements the callback routine and waits to be notified by some callback method.
- Business logic is often implemented using Enterprise Java Beans. In this case, the asynchronous client can use message-driven beans to read the request, initiate processing, and formulate the response.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

**Assembling responses.** Responses should be assembled and cached in the interaction layer. A common challenge is how to approach the server's handling of a variety of client types, ranging from rich clients, through a variety of browsers, to hand-held devices. The service interaction layer must implement several types of transformations of the results to comply with the client's rendering capabilities.

**The service processing layer design.** In the J2EE platform, the business logic can be implemented by Session beans and Entity Beans (commonly known as EJB). The EJBs can reside either in the same JVM as the remaining subsystems (that is, servlets or JSP) or on a different machine and a different JVM. The designer makes decisions about his/her interface model by exposing either EJB local or remote interfaces. Further decisions must be made on the usage of EJB container-managed persistency or bean-managed persistency. More details on these issues can be found on the relevant Sun Microsystems Web site (http://java.sun.blueprints).

**XML document size considerations.** Fragmenting XML documents helps in the logical organization of the documents. Instead of passing an entire document to all entities in the processing layer, the logical fragmentation of a large XML document allows splitting the document into smaller segments and passing only segments for processing.

## Endnotes

1 Note that this scenario changes for non-Java clients.

2 Java types are explained in any Java programming guide.

3 It should be noted that these clients are not designed solely to use this particular Web service.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter V

# Servlets

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

The following examines some advantages of using servlets:

- Servlets show better performance compared to CGI. CGI enables the Web server to call an external program to process an 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 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 the Java world. They can be moved to any operating system that supports Java.
- Servlets have access to the rich Java library, which helps to speed up the development cycle.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- Because of Java's capabilities, robust applications based on servlets can be built, protected from memory leaks, and with support for garbage collection.

Java Servlets are in version 2.4, and they are part of the J2EE (Java 2 Enterprise Edition) platform. More information about Java Servlets and J2EE SDK can be found on Sun Microsystems' Web site (http://java.sun.com.products/servlets).

## The Servlet Application Architecture

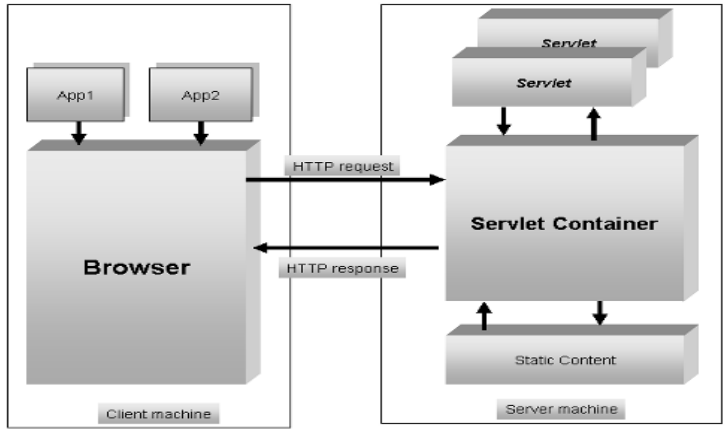
A servlet container manages servlets on a per Web application basis. Web applications and associated configuration files are specified by the Servlet Specification (Servlets Specification, 2004). The Servlet specification describes the following components and their usage:

- *Servlets and HTTP Servlets* define mechanisms for sending responses and receiving requests in a client-server model. They are mostly used for generating dynamic content for Web clients using native support for the HTTP protocol.
- *Filters* provide abstract methods for manipulating a client's request before it reaches the endpoint implementation. They can also be applied to responses from a server to a client. A filter is an object that can transform a request or modify a response. It is a preprocessor of the request before it reaches a servlet and/or a postprocessor of the response leaving a servlet.
- *Servlet security* uses the security features of the JVM. However, servlet security also defines an additional mechanism for controlling access to resources in Web applications.
- *Internationalization* allows the content to be developed and presented in many languages.

The requests for static content are directed to the HTTP server (for faster processing). A servlet-aware Web server is called a servlet container (Figure 52), and it forms a layer on the top of the HTTP server. Servlets interact with

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 52. Servlet application architecture*




clients via the HTTP communication model, also called the request-response model. The requests for dynamic content are examined by the server using configuration files (for example, web.xml file) and passed to the servlet that is servicing the application requests. A servlet container supports HTTP protocol for client requests and server responses. Secure transactions could use HTTPS, which is also supported by the servlet container.

Similar to many other components of the J2EE architecture, the servlets living in containers have a life cycle. All servlets must implement the javax.servlet.Servlet interface. This interface defines the following methods corresponding to the servlet lifecycle phases:

- The initialization phase is addressed through the init() method. When a container loads the servlet, its init() method is invoked before it services 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 represents the client's request for a 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).

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- The 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).

Before we continue to discuss the servlet lifecycle methods further, we need to explain two important principles: the relationship of the servlet to its container and the principles of communication between the client (browser) and the server. This leads us to the explanation of basic HTTP.

## Working with HTTP

An HTTP transaction starts with the HTTP client creating and sending a request to the server. This HTTP request is formatted as a line of text containing three parts:

1. Action type (method): for example GET, POST,
2. The resource requested from the server: typically a Web page identified by a URL (appp_root/CountryImages/Landscape.html), and
3. A version of HTTP protocol such as HTTP/1.0.

For example, GET/landscape/CountryImages HTTP/1.0 is a valid HTTP request that displays the Web page shown in Figure 53a. In the field More, we type the landscape we wish to retrieve (*Uluru*). The content of this field is passed to the servlet as a parameter. The first word - GET - is a method known to HTTP protocol. When this method reaches the server, it will ask for the specific resource (/landscape/CountryImages). HTTP/1.0 is the version of HTTP to be used in this transaction.

The content of the /CountryImages/Landscape resource is returned from the application called Country that is registered on our server[1]. The server in Figure 53b returns, in response to action More, a view of Uluru from the base of the rock. All the HTTP requests are similar. The server returns not only the response containing the content of the resource requested, but also the version of the protocol (1.1) and a response code (viewed from the command line the response looks like this: HTTP/1.1 200 OK). Table 2 contains the response code groups that the server returns.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 53. Example of HTTP GET and response to GET*

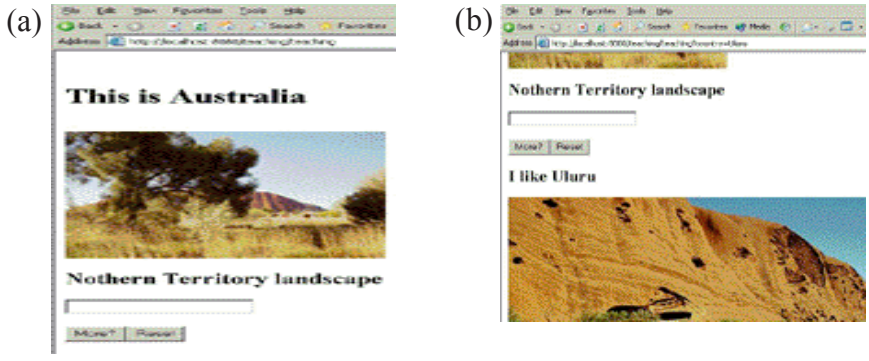



*Table 2. HTTP return codes*

| | |
|---|---|
| 1xx | International: request received, continuing process |
| 2xx | All went well. |
| 3xx | Further action must be taken to complete the request |
| 4xx | User-Agent error (typically bad syntax) |
| 5xx | Server error: server failed to fulfill the valid request |

The status code that is most often returned is 200. This means that everything succeeded and the response is valid. Other codes may appear: 404 (cannot find the requested URL) and 401 (authorization failed).

When going back in the history of the HTTP protocol, the widely used version of HTTP 0.9 supported only the GET method. The GET request provides information in the form of a query string. The length of this string has been limited to 256 characters, which is only enough to accommodate the length of a complete URL. To overcome this limitation, the later version of HTTP 1.0 also included the POST method. The POST method uses a different transfer mechanism — a socket connection — and can therefore transfer an unlimited amount of information. This extra information does not appear as part of a URL, and it is sent only once. The current version of HTTP 1.1 contains seven methods: GET, PUT, POST, TRACE, DELETE, OPTIONS, and HEAD. The HEAD method requests only headers of the response; PUT is used to place

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

documents directly to a server; the DELETE removes documents on a server; and the TRACE method is used for debugging. The OPTIONS is used to obtain a list of methods supported by a server.

## Customizing the Servlet Initialization Phase

In practice, a servlet contains more than the three life cycle methods of init(), service(), and destroy(). The content of the Web is accessed via the HTTP protocol. Only a subclass of a generic servlet — javax.servlet.HttpServlet — knows how to deal with the HTTP request.

The life cycle of a servlet is controlled by the container in which the servlet is deployed. When a request arrives and is mapped to a servlet via its deployment descriptor, the following steps are performed by the container:

- If the servlet does not exist, the servlet class is loaded; an instance of the servlet class is created and initialized by running the init() method.
- After initialization is completed, the service() method 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, as shown in the code listing in Figure 54.

In the following example we are referring to the Web application named Country. The database of images (countryDB) contains many images of the

*Figure 54. Enabling database access*

```
//Find the Database holding the landscape images
//and open the connection using lookup service and JDBC

public CountryLocationDBAO() throws Exception {
  try {
     InitialContext ic = new InitialContext();
     Context envCtx = (Context) ic.lookup("java:comp/env");
DataSource ds = (DataSource) envCtx.lookup("jdbc/CountryLocationDBAO ");
    con = ds.getConnection();
  } catch (Exception ex) {
      throw new Exception("Couldn't open connection to database: " + ex.getMessage());
      }
  }
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Australian landscape. A customer can browse the database, create a list of images (selection list) he/she would like to download, and delete images from the selection list. When the selection is completed, selected images are bundled into a zip file ready to download.

The first step is to find the database with the country images and to connect to it. The access to the database is enabled via JDBC (Figure 54).

The next step is to create the context listener and associate it with servlet events. The listener listens to the servlet lifecycle events, creates a Java Bean instance of type CountryLocationDBAO, and associates its name with the context attribute countryDB (Figure 55).

When the servlet is loaded and the container starts its initialization phase by calling the init() method, the context attribute object countryDB is obtained and a connection is established (Figure 56).

*Figure 55. Create context listener*

```
//Initialize the context listener to listen to servlet lifecycle methods.
import database.CountryLocationDBAO;

public final class ContextListener implements ServletContextListener {
private ServletContext context = null;

public void contextInitialized(ServletContextEvent event) {
      context = event.getServletContext();

try {
   CountryLocationDBAO countryDB = new CountryLocationDBAO ();
   context.setAttribute("countryDB ", countryDB);
      } catch (Exception ex) {
      System.out.println("Couldn't create images database bean: " + ex.getMessage());
      }
   }
```

*Figure 56. Servlet initialization phase*

```
//Initialize the database pointer in the servlet:

private CountryLocationDBAO countryDB;

public void init() throws ServletException {
    countryDB = (CountryLocationDBAO)
getServletContext().getAttribute("countryDB");

      if (countryDB == null) {
         throw new UnavailableException("Couldn't locate database.");
      }}
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

HttpServlet is derived (subclassed) from GenericServlet and contains a few modifications that are necessary for working with the HTTP protocol. Its service() method implementation calls one of the seven methods: doGet(), doPut(), doPost(), doHead(), doOptions(), doDelete(), which correspond to seven methods supported by the HTTP protocol. Coding of HttpServlet is straightforward.

We will now step through the code in Figure 57. Two objects are passed to the service() method: HttpServletRequest .request, and HttpServletResponse. response. The javax.servlet.http.HttpServlet class service() method handles redundant parts of the HTTP request and response unless the developer overrides the methods provided with HttpServletRequest or HTTPServletResponse objects to customize processing. Sending the information back to the client is accomplished via the HttpServletResponse object. This object itself produces only an empty response. The required HTML markup is achieved by using either getWriter() or getOutputStream() methods for the java.io.* package. All markups are written in the response object, and when the out.close() method is called, the response stream is put on the HTTP connection. We used the following code sequence to create the HTML code (Figure 58).

The HTTP request object arrives with parameters corresponding to the input fields on the HTML page (see Figure 53, the field called More). To obtain the content of this field, we need to access the request object parameters by their names. For example, using something similar to the listing:

```
String country = request.getParameter("country").
```

All parameters passed within the request objects can be obtained by their names. There are APIs to retrieve enumerated parameter names. In order to generate a response, the servlet "prints" the HTML markup into the PrinterWriter type object. Sometimes we need to break up the larger servlets into smaller parts and redirect the processing to other servlets. For this purpose we use request delegation. The servlet first obtains the RequestDispatcher object as shown in Figure 59.

The RequestDispatcher object's include() method dispatches the request to the "response" servlet path (/response). The dispatched servlets typically contain common banners and other resources that need to be passed to the other servlets.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 57. Show Country Images application*

```
package servlets;

import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
/**
 * This is a simple example of an HTTP Servlet. It responds to the GET method of the HTTP protocol.
 */
public class CountryServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    response.setBufferSize(8192);

    PrintWriter out = response.getWriter();

    // then write the data of the response
    out.println("<html>" + "<head><title>Country</title></head>");

    // then write the data of the response as the HTML string
    out.println("<body  bgcolor=\"#ffffff\">" + "<br>"+ "<br>" + "<h1>This is Australia</h1>" + "<br>" + "<img
src=\"Uluru.gif\" alt=\"NTUluru.gif\">" + "<h2>Nothern Territory landscape</h2>" + "<form method=\"get\">" +
"<input type=\"text\" name=\"country\" size=\"25\">" + "<p></p>" + "<input type=\"submit\" value=\"More?\">"
+"<input type=\"reset\" value=\"Reset\">" + "</form>");

    String country = request.getParameter("country");

    if ((country != null) && (country.length() > 0)) {
//pass handling to the other tresource - /response
       RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/response");

    if (dispatcher != null) {
       dispatcher.include(request, response);
           }
        }

       out.println("</body></html>");
        out.close();
  }

    public String getServletInfo() {
    return "The teaching servlet";
    }
}
```

*Figure 58. Creation of HTML markup*

```
PrintWriter out = response.getWriter();
out.println("<html>" + "<head><title>Country</title></head>");
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 59. Using RequestDispatcher Object to transfer control*

```
RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher("/response");
    if (dispatcher != null) {
        dispatcher.include(request, response);
}
```

*Filters* are used to examine headers or the content of a request or response. Filters usually do not create responses. 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 the javax.servlet package: Filter, FilterChain, and FilterConfig. The Filter interface can be implemented to define a specific filter. The following methods of the Filter class are recommended for implementation: initialize the filter, provide custom processing, and dispose of the filter [init(), doFilter(), destroy()].

There is variety of scenarios for the usage of filters. Typical ones are implementations of a hit counter or perusal of items in a shopping cart. The Web container uses filter mappings 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 servlet.

*Listeners* are used to monitor events in a servlet's life cycle. The listener receives notification of life cycle events. With this approach, the listener can listen to the Web context, session, or request.

The listeners give the developer (or rather, the application) more global control over the available resources. There are four kinds of event listeners that respond to Web application life-cycle events:

- **Servlet Context Listeners:** These listeners are notified when the servlet context (that is, the Web application) is initialized or destroyed.
- **Servlet Context Attribute Listeners:** These listeners are notified when attributes are added to, removed from, or replaced in the servlet context.
- **Session Listeners:** These listeners are notified when session objects are created, invalidated, or time out.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 60. Usage of HTTPSession object to maintain the session between multiple requests*

```
public class CountryImageServlet extends HttpServlet {
   public void doGet(HttpServletRequest request,
HttpServletResponse response)
       throws ServletException, IOException {
       // Get the user's session and list of images
HttpSession session = request.getSession();
ResourceBundle images = (ResourceBundle)
session.getAttribute("images") ;
ImageList selection = (ImageList)
session.getAttribute("selection");
```

- **Session Attribute Listeners:** These listeners are notified when attributes are added to, removed from, or replaced in any session.

For specific details, refer to Sun's (2004) J2EE 1.4 Tutorial Update 4 or J2EE Developer's Handbook (Perrone, Venkata, & Schwenk, 2003).

**Accessing other Web resources.** Often there is a need to access other Web resources. We have already mentioned the direct method of using a RequestDispatcher object (Figure 59). To include another resource or transfer control to another resource, the programmer can use RequestDispatcher's include() and forward() methods, respectively. An indirect invocation occurs when a Web resource embeds a URL pointing to another Web component in content returned to a client (encodeURL(request.getContextPath() + "/Landscape"). This is also referred to as URL rewriting.

**Session maintenance with servlets.** Note that by default a servlet is multithreaded. This means that only one instance is loaded by the container, and it is initialized once. After that, each request is handled concurrently by threads executing in the service() method. As we mentioned before, the HTTP protocol is stateless, cooperating with the multithreaded implementation of a servlet. Since many applications require processing of several requests to complete one business process, the state of one or more servlets has to be preserved across several requests. State variables cannot be stored in the servlet itself. Therefore, we associate each request with a session. Sessions are represented by HttpSession objects. In this fashion, we can associate object-

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

value attributes with a session. These attributes are accessible by any Web component that belongs to the same Web context (root) and handles the request belonging to the same session (Figure 60).

Our Country application stores images selected by the client as a session attribute. The image selection represents the state of the application for the particular client (names and identifiers of the selected images). The list of images is stored between requests, and the cooperating servlets can access this list to show, change, and delete images from the client's selection.

The handling of sessions has some problems. An HTTP client cannot signal to the server that the session is no longer needed. Therefore, each session is typically associated with a timeout, allowing the resources associated with the session to be reclaimed. The timeout period can be set or accessed using the session's API (get/setMaxInactiveInterval()). If the Sun's Application Server 8.1 is used, the deployment tool also provides the facility to access the timeout period.

## Endnote

[1] We have used Sun Microsystems Application Server 8.1.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter VI

# Java Server Pages (JSP)

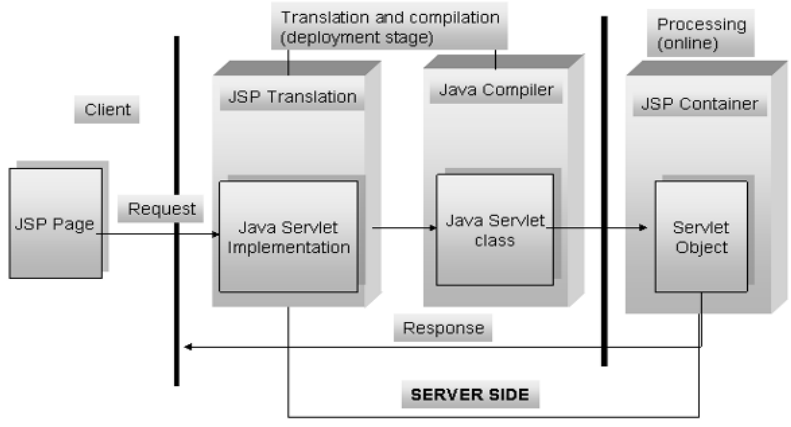
Good Web application design always tries to separate business objects from controls and the presentation. The benefit of using Java Server Pages (JSP) is the ability to separate Web page design from functionality and content dynamics (coding). The main features of the JSP technology are:

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

A JSP page is a text-based document that contains both static data as HTML, WML, SVG or XML, and JSP elements that 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 (examples include BEA WebLogic, IBM WebSphere, Sun Microsystems Application Server 8). A JSP page is a traditional Web page with the HTML mark up 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.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 61. JSP request-response flow*




When a JSP page is developed and deployed (Figure 61), it is translated into a *Java Servlet Implementation* (servlet java code), which is then compiled into the *Java Servlet class*. These two steps are typically completed offline during the deployment stage. However, online translation and compilation are also available. After deployment, the JSP page is called and passed directly to the processing stage where an instance of the servlet class (*Servlet Object*) is created. The *Servlet Object* produces the response, which is then returned to the client's browser via the HTTP connection.

The basic components of the JSP system are:

- **Template Data:** Static elements written in HTML or XML.
- **Directives:** JSP elements that are interpreted at the translation time.
- **Scripting Elements:** JSP elements used to define a variable and/or method declaration, expressions, and blocks of commands (called scriptlets).
- **Action Elements:** Commands that look like tags but are implemented as Java code. Standard actions contain commands like forwarding the request to another resource, creating JavaBean objects, and so forth. In addition, custom actions can be designed via JSP tag extension support.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Details of programming JSP can be found in many sources, For example, J2EE 1.4 Tutorial Update 4 (2004) provides excellent step by step guidelines and explanation and Perrone, Venkata, & Schwenk's (2003) J2EE Developer's Handbook is a very comprehensive programming guide at an intermediate level.

## JSP Development Notes

Since JSP is widely used, we would like to discuss some best practices in developing JSPs. Some guidelines based on practical experience with deployment of Web service endpoints follow.

We recommend that the developers always use the *MVC (Model View Controller) design pattern* (see Figure 72) to separate business logic from control and presentation.

JSP supports the separation of concerns paradigm in terms of the clean *separation of static and dynamic content*. Therefore, inserting too much Java code into JSP documents creates spaghetti code and defeats the purpose of the separation paradigm.

The *include mechanism* is important (include directive). Static data such as headers and footers that do not change too often should be kept in separate files, and by using the include directive, they can be inserted into all pages at the translation phase. However, if the content changes often, then the include page or an appropriate action directive is better.

In the latest versions of JSP it is possible to use *filters*. The filters are useful, for instance, to provide a single sign on mechanism, as authentication can be requested from the JSP page.

JSP documents can be used as *templates*: a typical application requirement is that application *views* (presentation style) have a common layout. A template is a presentation component composed of separate subviews, which together form a page with a specific layout. Each subview (banner, a navigation bar, or body content) is a component. Views sharing a template then have the same layout.

JSP documents are an excellent technology for generating XML with fixed structure. They are useful for generating XML messages in standard formats, where message tags are fixed, and only attribute values or character data change each time the page is served. Furthermore, XML documents can be

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

created from templates, assembling multiple XML documents into a single document.

Our experience shows that developers sometimes do not use standard tag libraries correctly. Standard tag libraries usually provide custom tags that enable looping, perform iterations, evaluate expressions, and make decisions. These tags are called "logic tags." Developers tend to use standard tag libraries to perform many logic functions in JSP pages. Redesigning functionality provided in standard tag libraries and using custom tags for logic provides little benefit and often violates the principle of the separation of logic and presentation. JSP pages are simply procedural programs written in XML syntax. Including custom functionality can mean that they become difficult to maintain, as with any other code.

From the point of view of complexity (when logic handling becomes very complex for standard tag libraries), it is more beneficial to implement logic in a custom tag, servlet, or helper servlet, or to implement the tag handlers using enterprise beans. A thin layer of code (the tag handler class) links the custom tag with the enterprise bean lookup and method invocation. This approach provides the view (JSP page or servlet) with direct access to model data (in the enterprise beans), maintaining separation of presentation from functionality.

*Scriptlets* are pieces of code in a large area of HTML. Changing scriptlet code or other HTML code requires some understanding of what the rest of the code is doing. Scriptlets can easily "redefine" the purpose of a JSP page by expressing program logic within the presentation. Therefore, the scriptlet code should appear in exactly one place: in the JSP page that defines it. If the same processing is required on some other page(s), it must be either included or copied and pasted into the new context. It can be argued that using an include directive may reduce readability. Furthermore, when the scriptlet requires some modification, usually all copies must be updated. Finding all copies of the scriptlet and updating them is a maintenance headache, with the real possibility of introducing unexpected errors.

Scriptlets insert code and logic programming into Web content. This treatment results in the requirement for Web page designers to understand the side effects of the scriptlet programming. Poorly implemented scriptlets can create dependencies on the template data.

Many JSP page compilers show poor error reporting when translating line numbers between the source JSP page and the generated servlet. Unit testing of scriptlet code is very difficult. Scriptlets are embedded in JSP pages;

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

therefore, the only way to test them is to execute the page and test the results. For example, a missed semicolon can cost hours of debugging time, creating nonsense errors.

**Usage of custom tags.** Custom tags centralize code in one place. When a tag handler's class changes, the tag's behavior changes everywhere it is used. Custom tags encapsulate program logic. Therefore, JSP pages can focus on presentation.

Erroneous code in custom tags will not compile. However, the entire context for determining the problem is present in the custom tag code, and the line numbers can be used to find the error.

A custom tag can be unit tested, and errors can be isolated to the tag or the page. Some projects use a specific development role of a Web page developer. Typically, the Web page developers do not know much about programming. Web page developers with limited programming skill can use scriptlets as long as the usage is confined only to presentation logic. Business logic should never be implemented in scriptlets.

## Integrating Servlets and JSP

The guidelines for using J2EE typically recommend implementation of the MVC pattern, and the controllers should strictly be implemented as servlets, because controllers are logical components, not presentation components. Nothing stops the developer to implement the controller as a JSP page that calls RequestDispatcher.forward() directly or to use a custom tag. The code in Figure 62 demonstrates how a controller can be implemented as a JSP page.

The code example shows the scriptlets and tags conditionally forwarding the request to another JSP page based on a request attribute. We have some reservations about this approach since it clearly mixes the processing logic and the presentation view. In this case, the J2EE coding guidelines suggest that the

*Figure 62. JSP as a controller*

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

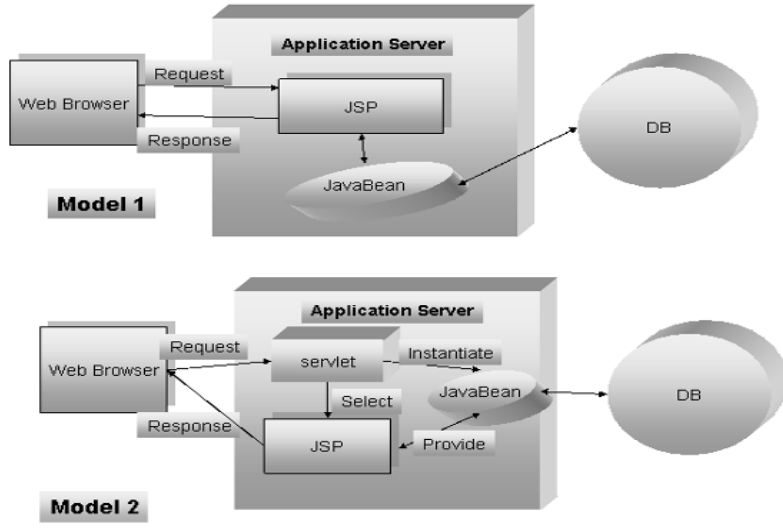
Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

better option is to implement the controller as a servlet to avoid mixing conditional attribute tests with the HTML tags.

We believe that the problem of easily mixing code with HTML tags arises from the JSP specification itself. The JSP specification allows two approaches for building Web applications: JSP model 1 and JSP model 2, which is based on MVC2 (Figure 63).

These two models differ in the location where the processing takes place. In model 1, 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 an implementation could result in breaking all the good design guidelines mentioned previously in this section; the control is bundled in JSP, and portions of the code are embedded in scriptlets or custom tags are interleaved with the presentation. In model 2, the control is performed by the servlet (proper Java code); JavaBean components are used as the model part of MVC, and the presentation is in JSP. The JSP avoids any control scriptlets having to be concerned predominantly with HTML code.

*Figure 63. JSP architectural models (model 1 and model 2)*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Web Module Deployment

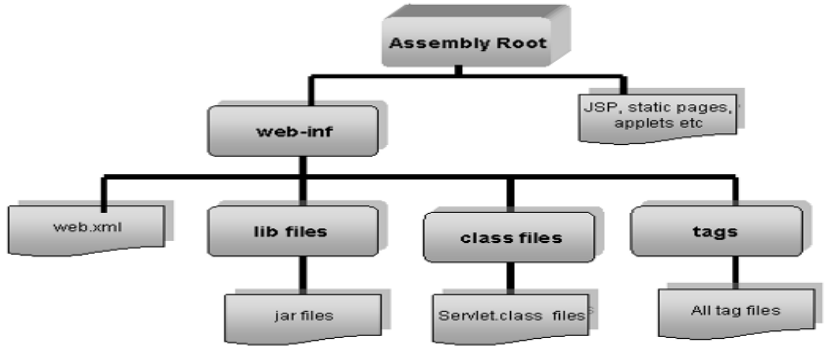
Every Web application (composed of servlets, JSP, beans, and HTML) and its components (Web modules) must be deployed. Deployment ensures that the Web server, application server, and portal server can resolve references to all objects. Application Web modules must be packaged into a war file. There are several methods that can be used to deploy the Web modules:

- Using a deployment tool (for example, the Sun Microsystems Application Server 8 comes with such a 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 (format must be identical to the archive file format [MyApplication.war]) as in Figure 64.
- Using the administrative console to deploy and launch new Web applications already packaged in a war file.

A special directory exists within the application hierarchy named "WEB-INF." This directory contains all things related to the application that are not in the document root of the application. The WEB-INF node is not part of the public document tree of the application. No file contained in the WEB-INF directory may be served directly to a client by the container. However, the contents of the WEB-INF directory are visible to servlet code using the getResource() and getResourceAsStream() method calls on the ServletContext.

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

*Figure 64. Web module archive file layout*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 65. Example of deployment descriptor web.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<Web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/Web-app_2_4.xsd">
  <display-name>Country</display-name>
  <servlet>
    <display-name>LandscapeServlet</display-name>
    <servlet-name>LandscapeServlet</servlet-name>
    <servlet-class>servlets.LandscapeServlet</servlet-class>
  </servlet>
  <servlet>
    <display-name>CountryServlet</display-name>
    <servlet-name>CountryServlet</servlet-name>
    <servlet-class>servlets.CountryServlet</servlet-class>
  </servlet>
<servlet-mapping>
    <servlet-name>LandscapeServlet</servlet-name>
    <url-pattern>/welcome</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>CountryServlet</servlet-name>
    <url-pattern>/response</url-pattern>
  </servlet-mapping>
</web-app>
```

web.xml file controls the application's life. The current complete schema can be found in the Servlet 2.4 specification.

The example in Figure 65 shows two servlets and the mapping into their respective URL addresses.

In general, the servlet is first declared by a servlet tag, display-name tag, servlet-name, and servlet-class. After declaration, the servlet can be referenced by its name and mapped into a URL path. The URL path is declared in the servlet-mapping tag body as servlet-name and respective URL-pattern tag. The Servlet Specification document defines combinations that are available for the URL path.

Similar to applets, servlets occasionally need initial configuration. The configuration information consists of a string or a set of string values that are included in the web.xml file (Figure 66). These values are persistent and are typically read during the initialization phase. This functionality allows the servlet to obtain the specified initialization parameters outside of the compiled code.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 66. Servlet's configuration parameters*

```
<servlet>
  <display-name>LandscapeServlet</display-name>
  <servlet-name>LandscapeServlet</servlet-name>
  <servlet-class>servlets.LandscapeServlet</servlet-class>

 <init-param>
   <param-name>welcome</param-name>
   <param-value>Welcome in Australia</param-value>
 </init-param>
</servlet>

<context-param>
  <param-name>javax.servlet.jsp.jstl.fmt.localizationContext
  </param-name>
   <param-value>images.ListOfImages</param-value>
</context-param>
```

Apart from servlet initialization parameters, there are also the application initialization parameters in the web.xml file as a context-param tag. They can be used to store a collection's name, and the actual list of collection subsets can be listed in the param-value tag, as shown in Figure 67.

If the application is using listeners, they also must be registered. They can be moved to the tag library.

If the URL points to a directory but no filename is given, typically index.jsp or index.html files are used. Since the elements in the web.xml file must be ordered, the welcome-file-list needs to come after servlet tags but before the error-page. An example of a welcome file declaration is in Figure 69. This file defines the default URI returned by the Web container if a partial request (i.e., request that contains the directory path but no specific file name) has been made, provided the file has been properly mapped to a static resource or to a servlet.

*Figure 67. Use of context parameter tag*

```
<context-param>
 <param-name>ImageCollectionName</param-name>
 <param-value>Australia Landscape</param-value>
</context-param>
<context-param>
 <param-name>Northern Territory</param-name>
 <param-value>Victoria</param-value>
</context-param>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 68. Example of listener registration*

```
<!--
 Register the listener that shows specials in the collection. -->
  <listener>
<listener-class>Contry.listeners.CollectionSpecials</listener-class>
</listener>
```

*Figure 69. Welcome file declaration in web.xml file*

```
<welcome-file-list>
   <welcome-file>index.jsp</welcome-file>
   <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

In summary, the deployment of a Web application is based on a deployment descriptor, which is an xml file, and on a number of resources, all of which are packaged in a war file. The class loader resolves the required class name to the mapped resources in the deployment file. For consistency reasons, the J2EE system classes cannot be overridden by any classes of the same name deployed in the deployment file. Resources that are external to the war file (for example enterprise beans) can also be defined in the war file so that they can be found in the Web environment.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter VII

# Web Service Design Issues

Web Services are gaining in popularity because of the benefits they provide. One of the major benefits is their support for interoperability in a heterogeneous environment, which leads to the capability to add systems and solutions that require different platforms. As long as the various systems are enabled for Web Services, the services can be used to facilitate interoperation. Web Services let enterprise application developers reuse and customize existing information assets. Web Services provide developers with standard ways to access middle-tier and back-end services, such as database management systems and transaction monitors, and to integrate them with other applications.

A Web service implementation (service endpoint), a Web service client, and a server's interfaces to the service form a cooperating structure. Because all of the service interfaces have a consistent implementation, developers do not need to acquire new skill or learn new styles to work with various services. It also means that many components can be reused or modified and then reused. Building these components requires consideration of some general rules that both Web service client and service endpoint developers must understand. The following sections discuss design issues that are relevant to building service endpoints, server interfaces, and Web service clients.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Service Endpoint Design

To enable interaction with clients, Web Services must expose their interfaces. Web Services can either be built from scratch or can be added to an existing application, thus making it look like a Web service. There are two separate issues in the design:

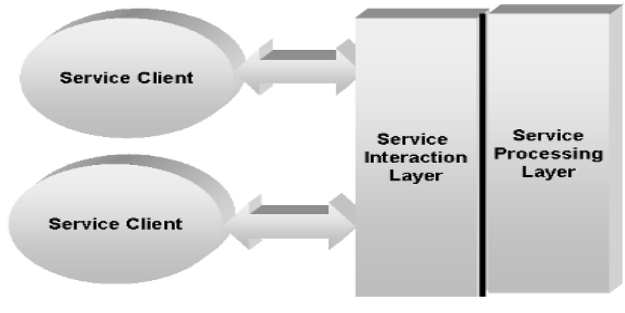
- Design of Web service capabilities, their interoperability, and efficiency of logic flow; and
- Design of business logic to support business processes.

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

Interoperability is the main focus for the design of Web service capabilities. This design should start at a higher level of abstraction (architecture stage), and it should involve assigning the responsibilities to layers. We start with splitting the responsibilities by assuming (similar to XML-based Web service endpoint) that each Web service endpoint has a service interaction layer (service interface for communication with its clients) and a service processing layer, which is the actual implementation of business logic processing (Figure 70).

From this perspective, a typical Web service developer must resolve several design issues:

*Figure 70. High level view of Web service*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- **Service interaction layer responsibilities:** How to design the interface for Web service clients, types of calls clients may make, the parameters involved, and service usage patterns.
- Issues related to the service availability to other parties: for instance, what interfaces should be published and where to publish them; which essential implementation details should the other parties have access to; and what other components should be designed and automatically provided to the other parties.
- **Management of clients' requests:** Issues relevant to receiving and processing clients' requests; which clients' calls should be pre-processed before passing them to the processing layer.
- **Delegation of the requests to appropriate business processes:** There is need to consider the efficiency of passing the request to the business logic, decisions must be made about synchronous or asynchronous processing, and there are issues of potential concurrency conflicts. For example, Entity Beans are by default non-reentrant, and therefore we may end up with multiple copies or concurrency conflicts.
- **Management of responses:** Issues associated with creating and sending the response back to the clients. The designer must decide how components (servlets, JSP, custom tag libraries, etc.) can be used in the particular solution.
- **Collection and handling of problems:** Determine how to collect and report problems and errors, what type of errors should be reported, and how error handlers should be conceptualized.

Considering the design issues outlined, a high level view helps to assign responsibilities to appropriate layers, thus making efficient use of the separation of concerns paradigm. In the following sections, we discuss some of these considerations.

## Service Interaction Layer

The service interaction layer must handle several major responsibilities. The first issue is how to prioritize the approach to the development of the service interface definition:

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- **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 developers can use this approach.

  However, there are some potential problems with this approach. Services represent the implementation of some business processes. The business processes are often subjected to evolutionary changes. It could be difficult to evolve the service interface without making any changes in the corresponding WSDL document. Changing the WSDL may result in rewriting the service's clients. These changes, and the accompanying instability, can affect the interoperability of the service itself.

  The developer must also be aware that different tools may use different interpretations for certain Java types. For example, java.util.Date could be interpreted as java.util.Calendar. Each utility has different representations in the WSDL file. Incorrect interpretation of some Java types may lead to some unexpected problems.
- **Approach 2:** *Develop WSDL description of the service, and then build corresponding Java interfaces*. Similar 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 design and write the application. Another obstacle may be with some programmers who like to use so called method overload[1]. 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.

## Service Endpoint Implementation (Processing Layer)

The processing layer is where the business logic is applied to the Web service request. There are two options for implementing the Web service interface: JAX-RPC service endpoint or an EJB service endpoint. The choice determines whether the endpoint can be embedded in the same tier as the service

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

implementation, resulting in a simpler implementation, or whether a separate EJB tier is necessary. The latter solution typically requires an additional pipeline from the Web tier to the EJBs, hence a more complex solution. The decision whether to use a JAX-RPC endpoint or an EJB service endpoint should be based on the type of business logic processing.

Two options should be considered:

- **The Web service is a new service that does not use existing business logic:** The endpoint type depends on whether the business logic is completely encapsulated within either the Web tier (*JAX-RPC endpoint*), or the EJB tier (*EJB service endpoint*) is required to provide the necessary information (for example, an EJB persistency layer is required to retrieve and store data in the database). The decision regarding where to do the pre-processing of requests is part of the Web tier design.
- **The Web service interface is added to the existing application:** A decision to take this approach should be based on the tier where the existing application preprocesses the requests. If the application needs to pre-process the requests before forwarding them to other parts of the service (in the Web tier), then a JAX-RPC service endpoint is suitable. The EJB service endpoint should be selected when the pre-processing occurs on the EJB tier.

At this point, we also wish to mention some additional constraints relevant to multithreaded access to the Web service based on the components used to implement the service endpoint. Multithreaded access and transaction processing constraints are often important, and yet often the designer does not pay enough attention to these issues. The following constraints should be carefully considered:

- **An EJB service endpoint with a session bean is used to implement business logic:** A typical, and often recommended, implementation is a stateless session bean. With this approach, the EJB container serializes requests to the session bean instances (they are by default not reentrant components), and concurrency constraints are not an issue. If you use a stateful session bean, concurrency must be handled.
- **A JAX-RPC implementation of the service endpoint has to handle concurrent clients within its implementation:** The developer has to

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

include block or method synchronization code to handle multithreaded access. Incorrect decisions could result in performance penalties. For example, large processing sequences in synchronized methods could lead to a non-responsive server.

Processing of transactions differs with different service endpoints. The JAX-RPC service endpoint runs in a Web container, and its transactional context is unspecified. There is also no declarative means to automatically mark the start of the transaction (JTA is one approach to explicitly mark the transaction's start and end). With the EJB service endpoint the transaction context is the responsibility of the EJB container. The developers need to declaratively demarcate transactions and determine the persistent attributes. The business logic then runs under the transactional context defined by the EJB's container-transaction element in the deployment descriptor.

A Web service's methods can be accessed by many different clients. There is often the requirement to enforce different access constraints for each method. The required granularity of access should correspond to the choice of the endpoint as follows:

- If there is a requirement to control service access at the individual method level, then it is more appropriate to use an EJB service endpoint. Enterprise beans permit method-level access permission declaration in the deployment descriptor. The designer can declare various access permissions for different enterprise bean methods, and the container correctly handles access to these methods (this is true for a stateless session bean).
- A JAX-RPC service endpoint does not have a facility for declaring method-level access constraints, so the designer must provide this implementation in the code.

We often need to handle an *HTTP session for multiple request-response pairs*. A JAX-RPC service endpoint runs in the Web container and has complete access to an HttpSession object to implement session handlers. This approach can be used to embed cookies and store client state. However, it is suitable only for short conversations. An EJB service endpoint has no access to Web container state. The EJB Web service endpoint often represents

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

business processes with longer durations, and therefore, it needs additional mechanisms that are discussed later in this section.

*Web Services are not error prone* applications. For simplicity, we distinguish two major classes of errors: application errors and unrecoverable system errors. Application errors may occur within all three tiers of an application (client, Web tier, and EJB tier). We consider application errors to be recoverable errors, providing that appropriate error handling is applied. As in any J2EE application, Web service error conditions may occur while processing the client request. Service-specific exceptions are thrown by the service itself. With a standalone Java application, any exceptions thrown by an error condition can be caught. They then can be propagated up the call stack until the appropriate exception handler is reached. These exceptions in a Web service endpoint belong to the category of checked exceptions, which extend the standard java or javax package. When these exceptions reach the client, the exception type and context can be lost or not clear. These must be well documented, specifically if the client platforms are heterogeneous. We recommend that the exceptions should not be passed to the client in the format as they were generated on the server. Instead, they should be wrapped into a meaningful exception that is specific to Web Services and well documented.

Unrecoverable system errors are more complex and difficult to handle. A Web service may encounter an unrecoverable error such as a lost network connection. On the J2EE platform, it is the container's responsibility to throw a RemoteException, which is automatically mapped to the client side. The client should receive a meaningful message to enable it to understand the situation and adjust behavior accordingly. However, this solution may not be possible on heterogeneous platforms.

## Web Service Client Design

Web applications should be able to support a variety of client types: laptop, desktop, palmtops, mobile phones, and other pervasive devices. Connectivity capabilities span from an Intranet within an enterprise, over the World Wide Web, to wired and wireless networks or heterogeneous networks. Most clients today are thin, browser-based clients. Client design considerations include environmental constraints, user usage patterns, connectivity issues, network configurations suitable for the selected type of clients, security requirements, and performance issues.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Firstly, we want to reiterate the characteristics of Web Services with thin, browser-based clients. The browser clients use HTTP (or HTTPS) as the transport protocol and interact with an application by clicking on hyperlinked text or images or by completing and submitting forms. This class of requests, also called gestures, is translated into HTTP requests, typically mapped to HTTP GET, and delivered to the Web server where they are handled by servlets. The URLs of these requests may include parameters in a query string that determines what data should be retrieved. The other class of user requests, which aims at updating data, is typically mapped to HTTP POST requests. The requests are packaged in a MIME envelope of type application/x-www-form-urlencoded. The envelope also contains the parameters for the update. The body of the request may contain some actions. The issues to be considered are outlined in next paragraphs.

**Network issues and constraints.** The requirement for Web service and portal clients is to connect to the enterprise backbone over a variety of networks. The quality of service (QoS) on these networks can vary from excellent over a company intranet (controlled networks) to poor on a wireless network or Internet. The QoS over the Internet, being an uncontrolled network, cannot be guaranteed at any time. However, user satisfaction with the application depends on the available QoS of the network, its reliability, and application responsiveness. A Web service client is part of a large network configuration, and it cannot be programmed as such because it is part of a distributed application. Three aspects of the QoS that the designer cannot control include:

- **Latency:** May result in unacceptable request-response delay.
- **Bandwidth:** The lack of bandwidth may cause not only delays, but also reductions in the quality of images or video clips.
- **Reliability**

A well-designed enterprise application must address the issues in the previous list on a per application basis. No generic solutions are available, but generic measures are known, such as keeping the size of the Web page to a minimum, transmitting only the necessary information, and creating conversation scenarios where the human client is kept busy while a server transmission is taking place. A good client indicates and handles errors when the server is not reachable.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

**Security issues.** Different applications and networks have different security requirements. These requirements constrain how clients connect to the Web server and the intranet. Internet based clients usually pass their requests and responses through a firewall while connecting to the Web server. The presence of a firewall limits the communication protocol options. Most firewalls are configured for use with HTTP. Therefore, SOAP-based messages pass easily, while CORBA and ORB infrastructure, including Internet Inter-Orb Protocol (IIOP), (Harkey & Orfali, 1998) do not allow simple access to the server. Therefore, Web Services using HTTPS or SSL are particularly attractive compared to RMI or CORBA-based services, which use IIOP.

**Platform constraints.** Client platform capabilities influence application design. Thin, browser-based clients cannot simply draw complex graphics or provide complex processing. It requires a server to render graphics as images and send them with the remaining parts of the response. Thick clients can do the computation and rendering and download only the necessary data from the server. However, their footprints could be unacceptably large, not to mention deployment complications.

Another aspect is the actual **display real estate.** Mobile phones and PDAs provide only small screens as opposed to desktop computers. Applications that have to render data to multiple client platforms pose additional design challenges.

The client has to be able to *validate user inputs* in order to reduce network traffic and meet responsiveness requirements. With a thick client, it is an easy task to embed some validation code in the application. Thin browser clients cannot perform all the validation, but they can apply some heuristics, for example a check that the number of digits in a phone number should not exceed 10 digits. There are also some programmatic options available:

- **JavaScript scripting language.** JavaScript implementations may vary for different browsers, and the application may require testing on multiple browsers. JSP custom tags are useful because they generate simple portable JavaScript.
- **Web form using Java Script.** DOM has hooks to access form elements. When the user submits the form, a JavaScript function is called to validate the form.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Validating user inputs on the client side may positively affect the responsiveness of the interface. Although the validation code seemingly provides good response and errors are detected without consuming communication resources, the client consumes more bandwidth because the validation requires downloading some processing code and an HTML form. To reduce download time, it is preferable to place validation functions used by multiple forms in a separate source file and use the <SCRIPT> tag and <SRC> attribute to access this file. The browser recognizes the <SRC> attribute and in response caches the source file. Client-side validation optimizes response time. It improves the quality of user interface and decreases load, but the designer should not rely on the client to enforce data consistency.

## Maintaining the Conversational State

**Conversational state of clients.** 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 the client and the server. Clients view Web Services as stateless and assume that a service endpoint retains no knowledge of previous client interactions. In some cases, the client makes a sequence of calls to a service to complete the request. In this case, the client application has to maintain the state.

There are several strategies regarding how to *maintain the application state in the Web tier*, as described in the following:

- Use of HttpSession object associated with a servlet as discussed in the Session Maintenance with Servlets section.
- 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. Some firewalls and gateways use filters to prevent cookies from being downloaded to the client. There is also the problem with portability across several types of browsers. This solution, although widely used, has some negative consequences.
- URL rewriting is a strategy that uses encoding the session state within the URL. This approach should be used when we assume that cookies are disabled. When a user makes a request to this URL, the session state is

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

sent to the server. The pages containing rewritten URLs consume some bandwidth. When the server receives the request, it must rewrite every URL in its response. Such treatment increases the size of the response and consumes more bandwidth. More information on the Apache rewrite guide is available at http://httpd.apache.org/docs/misc/rewriteguide.html.

To be more specific, we will now describe the implementation of URL rewriting in the IBM's WebSphere Application Server. The request stores the session identifier in the page returned to the user. The WebSphere Application Server encodes the session identifier as a parameter on any link or form the user submits from a Web page. The servlet code is modified to include either an encodeURL() or encodeRedirectURL() method. When a user clicks on a link that utilizes one of these methods, the session identifier is passed into the request as a parameter, (the session ID is appended to the URL for the Web page).

Another session tracking mechanism is the Secure Socket Layer (SSL) ID that is negotiated between the Web browser and the HTTP server. There are some disadvantages in this approach. One of them is a possible failure of the HTTP server, which would be displayed as an unrecoverable error. The ID is negotiated between the browser and the HTTP server. The failure of the HTTP server results in the loss of this ID. In a clustered environment, a specific IP mechanism directs requests to the HTTP servers. These servers must handle the requests in such a way that the browser requests are returned to the HTTP server that negotiated the ID. The WebSphere uses multiple Application Servers on top of one HTTP server. This means that either cookies or URL rewriting must be enabled for communication between the HTTP server and the application server. Furthermore, the WebSphere Application Server must be specifically configured for this type of session maintenance.

Sometimes, the client needs its *conversational state managed by the service endpoint*. Such a situation, for example, is when a client application has only limited processing capabilities and restricted memory to handle a conversational state. In this case, a service endpoint must maintain the conversational state.

One strategy involves usage of nonreplicable tokens to identify unique communication with a particular client. The service endpoint and the client application pass the token between them with each call during their conversation. The use of tokens also may be used to prevent the client from attempting to invoke multiple sessions for the same application or to eliminate inappropriate invocation of the client through preferences.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Another strategy is to use entity beans to store the state of a conversation; EJB components (mostly entity beans) are often used to implement a service endpoint. The entity beans can serve as a data store during the session (using the persistency capability of the EJB container). The token the application is using can be represented by the primary key of an entity bean. This strategy has two problems, and the actual implementation could be very complex.

- One of the problems is to provide clean up of old session data from the persistent storage (for example a time stamp on the entity bean in combination with a timer for tracking elapsed time).
- The other problem is the tight coupling of the service endpoint and the client by tokens and timeouts. In particular, when a service is published in a UDDI or another public registry and used by many clients, or when changes and maintenance of service are frequent, the client developers whose applications interact with such a service endpoint need to be kept up-to-date about the service interface details (the endpoint's token requirements and implemented timeouts and session tracking). Therefore, developers should use this approach only when the tight coupling is acceptable or required and service endpoint responsibilities are clearly documented.

Sometimes multiple Web service clients are required to *share modifiable state*. This is the situation when several Web clients concurrently update shared data. Data consistency is an issue since one or more clients may be manipulating the same data at the same time.

## Available Web Application Frameworks

The popularity of the Model View Controller (MVC2) architecture sparked development of several Web-tier application frameworks. The main goal of these frameworks is to make the development process as easy as possible and to lower the cost of developing such applications. The following is a brief overview of some available frameworks and associated tools:

- **Java Server Faces (JSF)** is a JSR-127 standard. JSR 127 aims at providing the architecture and a set of APIs for dispatching requests to

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Web-tier model JavaBeans and for supporting maintenance of stateful, server-side representations of reusable HTML GUI components. There is also support for internationalization, validation, multiple client types, and accessibility. The main goals are to allow tool interoperation and the development of portable, reusable Web-tier GUI component libraries.

- **J2EE BluePrints Web Application Framework ("WAF")** is a framework offers a Front Controller servlet and an action class for encapsulating server side actions, template service, generic custom tags, and internationalization support. WAF is suitable for small, non-critical applications, and as a learning framework that will help developers to embrace the principles of Web-tier, efficient application design, and usage.
- **Apache Struts** is an open source server side application framework from the Apache Software Foundation. Struts is configurable; it also has a Front Controller, action classes and mappings, utility classes for XML, automatic population of server-side JavaBeans, validation on Web forms, and some internationalization support. Furthermore, it also includes a set of custom tags for accessing server-side state, creating HTML, performing presentation logic, and templates. However, it is not a standard, and many J2EE vendors are reluctant to invest in creating interoperable and reliable tools with it. The Struts framework has been customized by IBM to support IBM's WebSphere and WebSphere Portal Server-based applications (portlet development).
- **Hibernate** is a powerful, ultra-high performance object/relational persistence and query service for Java. Hibernate lets you develop persistent classes following common Java idioms including association, inheritance, polymorphism, composition, and the Java collections framework. The *Hibernate Query Language* is designed as a "minimal" object-oriented extension to SQL. It provides an elegant bridge between the object-oriented data representation and a relational database. Hibernate also allows you to express queries using native SQL or Java-based Criteria and Example queries. Hibernate claims support for all SQL database management systems. It also claims to have the capability to integrate with all popular J2EE application servers and Web containers.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

# Endnote

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

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

# Section III:

# Putting Portals on the Web

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter VIII

# Portals

In the following sections, the concepts of portals and portlets, but not the entire portlet specification, are explained. First, some fundamental aspects of portlet development need to be presented in order to ensure your understanding. We will start with some definitions pertaining to portal components.

- **Client's point of view:** 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. This definition represents the client's view of a portal application.
- **Developer's point of view:** A portlet is a Java based Web component managed by a portlet container that processes requests and generates dynamic content. A portlet container provides a 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, 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

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

relevant to the user. The purpose of portal solutions is to provide end users 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, and publish and share documents.

A simplified version of the client's perspective defines the necessary elements of portal applications:

- A 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 and can be filtered, personalized, and secured;
- A portal is designed to service thousands of concurrent sessions in an implementation; and
- Portals are enterprise applications that can run on load-balanced Web and application server clusters.

Web developers are facing problems introduced by the 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, and to enable companies to interact with external business partners as well their employees.

Many companies are developing new Internet-based applications focusing on business-to-consumer (B2C), business to employee (B2E), and business-to-business (B2B) solutions. (See also the section on business portal solutions). The core of the challenges currently being faced by Web designers and developers is the integration of disparate user content into seamless Web applications presented through well-designed user interface. It is envisaged that portal technology will provide a framework for building such applications for the Web. Such integration is not easy, and the developers have to take into account many issues such as:

- How to manage the integration of data and applications with legacy and other already established systems;

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- How to integrate with existing Web applications while ensuring correct authorization and authentication for users and satisfactory performance of the system as a whole;
- How to provide consistent access to multiple Web Services provided by other companies; and
- Some existing Web applications are deployed on different software platforms such as .NET or servlet engines[1]. How do we make them work together without investing in additional middleware?

Portlets and portal applications can be bought as off-the-shelf packages and can be customized. The question is whether the company should develop their own portlets and/or even entire portal applications. A similar question refers to reusing and customizing existing portlets. How do we justify the costs? Is this approach practical?

Portlet standards are not well established yet. Our experience to date suggests that most corporate portals are custom developed and often based on various portal implementations. Which portal engine should be used? As you will see later, each implementation of a portal engine provides both positive and negative points that must be considered. Examples of existing products include WebSphere Portal, Apache Jetspeed Portal, Microsoft SharePoint and Vignnette, Plumtree, ATG Dynamo, and Sun Microsystems Sun Java System Portal Server 6.2.

One of the characteristics of portal applications is that of constant change and extensibility. In order to ease and speed up the development process of portal applications, portal vendors provide development toolkits, deployment tools, and frameworks. The purpose of these is to provide an extensible environment and services that allow the developers to deliver new portal applications fast.

## Historical Background

When talking about Web portals, there is no need to go back any further than 1989, when the Web was invented at CERN by Tim Berners-Lee. Then we only had very few Web sites, and it was not until 1992 that the first Web server was set up in the United States.

After that, things went fast, especially with 1993 being the year the Web really started to take off. In that year, where the world in October 1993 had 200

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

known Web servers, Marc Andreesen started working on his Mosaic for X browser, that he later turned into Netscape Navigator. In that same year, The New York Times also mentioned the Web for the first time.

Back in those days, an authoring function was built into most browsers (today you can find this function in the Amaya browser), and so it was, as the Web was supposed to be, really easy to add and edit information on Web pages. Because of this function, many users used a local file as their so-called default homepage.

The year after, in 1994, two PhD candidates in electrical engineering at Stanford University, David Filo and Jerry Yang, started Yahoo! as a way to keep track of their personal interests on the Internet. Yahoo! later went public and is now a major Web portal.

AltaVista, which later grew into the most used search engine on the Internet, was then born in 1995, the year when what was later called the portal race began. Netscape went public and released a new version 2.0 of their Netscape Navigator browser. In this new browser, which became wildly popular and caused Microsoft to take the Web seriously, the default Web site was set to Netscape's homepage. This naturally increased the traffic on Netscape's site.

Except for Netscape, other important so called "portal" sites[2] were also the search engines, including AltaVista. The Web was starting to drown in information overload, and the search engines did their best to try to help desperate users find what they needed.

America Online today has more than 12 million subscribers, who, when they setup their AOL/Microsoft browser by default, start at the AOL main site.

Portals have gone through a long evolutionary process. If we look back to the original personal computing and x286 processor days when each application required the entire screen and used all the computer's resources, the introduction of Graphical User Interface (GUI) and window's based interfaces changed the way we interact with applications. A user can switch between applications and interact with multiple applications simultaneously.

The next revolutionary step was the introduction of Web applications and the Internet. Initially we interacted with the Web application by entering a single URL to access a single Web site. As the Web and browser technology evolved, applets and browser plug-ins for technologies like Java became part of Web applications.

Web applications have changed the delivery of services. Unfortunately, as with many other platforms, these technologies were never properly standardized.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Consequently, it was very difficult to provide cross-browser implementations and to search for services and information. With Web Services, the need to provide dynamic content on the Web pushed the development of Web servers into application servers that could serve dynamic content and technologies such as Java Server Pages (JSPs) and Java Faces[3].

The capabilities of Web Services using application servers included rendering dynamic content and access to applications residing in different locations. However, they could not integrate multiple data sources into a single interface.

Portals have evolved from application servers as the solution for rendering multiple streams of dynamic content. The early portals were proprietary and specific to each implementation. They form an additional layer, and they cooperate with the application server. As these portals were gaining acceptance, there appeared requirements for easy customization and for development and deployment tools. These tools included standards and frameworks to support the building of new portal applications.

Most portals provide personalization and customization. Personalization enables the portal to deliver specific information targeting the individual user, based on unique information about this user. Customization allows the user to organize the look and feel of the portal to suit their individual needs and tastes. Typical portal server deployment tools provide a framework for addressing all these issues along with an open, flexible infrastructure for creating many types of portals, accessible from a wide variety of devices.

To summarize our discussion, we propose that the main purpose of a portal is to provide the following services:

- A server to aggregate content from multiple resources;.
- A scalable infrastructure;
- Frameworks and tools to build and deploy portal components;
- Provision of a unified way of self-customization, and self-administration;
- Single access point to multiple applications and Web Services;
- Standardization and the ability to interchange the portlet applications among portal vendors; and
- Support for the processing of portlet fragments into Web pages.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Generations of Portal Applications

To business users, a portal is all about information access and navigation. The business world also talks about portals as a means to connect multiple Web Services and to personalize the access. Currently, we have seen four generations of portals, and the technology is still rapidly evolving.

The first generation portals were focused on providing static Web content, Web documents, and some live feeds. In a corporate environment, their objective was to provide a single access point to corporate information distributed throughout the enterprise. They typically contained information such as company news, employee contact information, company policy documents, and other key Web links.

Second generation portals added features such as personalization of the view of the content, customization of content, and search capability. They were typically manually deployed and developed within the company. No standards were applied.

Third generation portals focused on specific information and applications. Integration at the data level was also added. Third generation portals incorporated the notion of providing services along with the first generation idea of providing content. Another key feature of third generation portals is support for collaboration. Collaboration portals provide the ability for teams to work in a virtual office. They provide content management services, the mining and organization of related information, along with collaborative services that allow users to chat, e-mail, share calendars, and define user communities. Collaborative portals are typically internal corporate portal installations.

Fourth generation portals are intended to support full-functionality of e-business applications. This involves integration with legacy applications at the component level, as well as with existing Web Services.

Enterprise portals provide employees with a more or less complete set of online tools and services to reduce paperwork and increase their productivity. In addition, fourth generation portals provide services for the entire scale of external e-business applications, including B2B, B2E, and B2C. Some additional employee applications are M2E (Manager to Employee), E2E (Employee to Employee), and X2E (eXternal to Employee). Taken together, all these relationships are considered part of an ERM (Enterprise Resource Management) strategy (Doerzapf & Udolph, 2002). Eckerson (1999) proposed four portal generations (a modified version in Table 3), focusing his

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Table 3. Eckerson's portal generations*

| Generation | Descriptor | Features |
|---|---|---|
| **1** | **Referential** | **Generic focus**<br>**Hierarchical catalogue of pages**<br>**Pull flow**<br>**Decision support** |
| **2** | **Personalized** | **Personalized focus**<br>**Push and pull flow**<br>**Customized distribution** |
| **3** | **Interactive** | **Application focused**<br>**Collaborative flow** |
| **4** | **Specialized** | **Role focused**<br>**Corporate applications**<br>**Integrated work flow** |

analysis on three features: information content, information flow, and technology deployed.

A portal framework also provides access for multiple types of devices, such as PDAs and mobile phones, diverse browsers, version of browsers, and pervasive devices, to address the needs of location aware and small real estate screen communities of users. The important features are facilities for automated personalization based on business rules.

We believe that the evolution of portals does not stop here. Open frameworks and tools to find and utilize common services anywhere and anytime will appear in the next generation of portals. This generation will support the implementation through the Service Oriented Architecture Frameworks. These frameworks may not be implemented on multilayer platforms, but over secure Grids with horizontal separation of concerns.

## Business Portal Solutions

Large organizations have many types of users that rely on their information and services. Customers, suppliers, partners, and employees each have specific and often diverse needs. To address these needs, many different kinds of portals have been implemented.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

These can be categorized as follows:

- **Business-to-Consumer:** This type of extended enterprise portal (extranet) is associated with CRM (Customer Relationship Management) and provides consumers with direct access to a variety of content — for example, product manuals and price lists. Portal customers can also purchase products, check order status, and communicate with customer support. Like any other portal, a B2C portal is usually tailored to match customer needs.
- **Business-to-Business:** Another type of extended enterprise portal, B2B portals participate in SCM (Supply Chain Management) by providing personalized access to business information for suppliers, resellers, and distributors. A typical B2B portal might provide a business partner with access to purchase orders, invoices, statements, and delivery confirmations. Application integration includes many business processes in procurement, billing, manufacturing, and distribution areas.
- **Business-to-Employee:** B2E portals (also known as intranet portals) generally serve as a means to aggregate and disseminate corporate information and services to an organization's employees. There are two basic types of B2E portals:
    - Employee portals provide general access to relevant content such as company news, HR applications, news and document search engines, sources of expertise, reports, and other types of information applicable to all employees or larger groups of employees. Typically, an employee portal also allows for self-service, where an employee can sign up for classes or benefits, and change personal information.
    - Knowledge worker portals (sometimes called professional pages) provide limited access and are aimed at a particular role or set of roles such as sales and sales forecasts or IT personnel. These portals often integrate content in order to support particular business processes.

The portals developed to support information management in Human Resources (HR) require special attention. According to Dessert and Colby

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Table 4. Evolution of portals in General Motors HR*

| Dimensions | 1st Generation | 2nd Generation | 3rd Generation |
|---|---|---|---|
| User<br>Stickiness | Static Web<br>High Usage<br>Search | Dynamic<br>Personalized<br>Robust Search | Anywhere Access<br>Analytics<br>Dashboard |
| Communications<br>& Collaboration | News<br>Chat<br>Jobs | Unified Messaging<br>Targeted Push vs. Pull<br>Role-Based | E-Learning<br>E-Culture<br>Broadcast Media |
| Information<br>Access | Online Publications<br>Links<br>Launching Pad | Dynamic publishing<br>Native Web Apps<br>Content Integration b/w functions | Online Publishing<br>Internet Content |
| Services | Travel Expenses<br>Payroll<br>E-Procurement | Life/Work Events<br>Communities<br>E-Health | Role Based<br>Online Consulting |
| Technology | Web/App Servers<br>Unsecured<br>Basic Login | Content Management<br>LDAP<br>Internet, E-Mail, Chat, IM, | Federated Services<br>Wireless<br>Multimedia<br>Broadband |

(2002), HR portal implementation in General Motors evolved over three stages as depicted in Table 4.

The three stages cover five organizational dimensions. The GM portal streamlines employee communications, indexes relevant content from hundreds of sources, consolidates hundreds of existing internal Web sites into a single point of access, and will bring collaboration and productivity tools to one place on the desktop. It also allows employees to customize the portal for fast and easy access to business applications and personal information. The portal runs on a Sun ONE Portal Server (formerly iPlanet™Portal Server), and it has been designed by Workscape specifically for GM. Workscape's TruSecure certification protects data and network security and projection against physical intrusion. The Sun ONE Portal Server includes secure remote access technology that allows GM employees to securely access their portal with a Web browser from any location; no client software is necessary. The scalability and performance of the Sun ONE Portal Server allows easy integration of additional Sun servers as companies' requirements increase with time.

## What Do Portal Services Deliver?

Although portals are very diverse, they all contain some common elements and provide common services. The diversity in both elements and services is mostly

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

due to vendor implementation. In this section, we discuss typical Portal services and show some specific details of various implementations.

- **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 portals different from Web Services where the end user may have a 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 a high volume of read requests, and they are optimized for fast read access.

The information in the directory is mostly static. As opposed to databases, directories do not support transaction processing. Database access is standardized via SQL, which supports read, write, update, and special functions such as table join, and so forth. A directory contains a collection of objects organized in a tree structure, and the protocol defines the naming model (how entries are identified and organized). Portals use the Lightweight Directory Access Protocol (LDAP) for storing the authentication information. In LDAP, the entries are organized in a tree-like structure called the Directory Information Tree (DIT).

**Security.** The following security standards are playing an important role in the development of portals. The security standards are based on encryption and decryption principles. Encryption is the process of rendering information so it

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

can be read only by intended recipients after it has been decrypted. Such data is mathematically encrypted in a way that makes it unreadable to anyone except those possessing the key, or method, to decrypt it.

*SSL* was created by Netscape, and enhanced in 1996 by IETF (Internet Engineering Task Force) to become the TSL (Transport Layer Security). These standards are similar and used interchangeably. The basis of this public key cryptosystem is ECC (Elliptic Curve Cryptography) Cipher Suites, suitable for mobile/wireless environments. Currently prevalent cryptosystems such as RSA or ECC provide an equivalent level of security.

The XML Encryption Syntax and Processing specification and the Decryption Transform for XML Signature are W3C standards for handling XML documents and XML element-level confidentiality. XML encryption can be used with a key exchange algorithm and public key encryption to encrypt documents. The protocol is a point-to-point type of protocol with a decryption process at each point. The Trust Service Integration Kit (TSIK) from Verisign provides Java APIs that minimize the development complexity for trusted applications. The APIs enable developers to integrate digital trust services and secure XML processing, authentication, real-time certificate validation, and authorization.

The *Security Assertion Markup Language* (SAML) standard defines a framework for exchanging security information between online business partners. It was developed by the OASIS (Organization for the Advancement of Structured Information Standards). To understand the main principles, you need to understand the following key concepts:

*Identity Provider* is the system, or administrative domain, that asserts information about a subject. The *Service Provider* is the system, or administrative domain, that relies on information supplied by the trusted Identity Provider. SAML defines a number of mechanisms that identify a trusted Service Provider. An SAML assertion is digitally signed using an XML signature, and it is therefore self-validating. SOAP is used as the transport media. With SAML, there is a provision for SSO across multiple domains due to its self-validating features[5].

The OASIS work resulted in formulation of the Web Service Security (WSS) specification describing enhancements to SOAP messaging to provide message integrity and confidentiality. The specified mechanisms can be used to accommodate a wide variety of security models and encryption technologies. It relies on standards like SOAP, XML Encryption, and XML Signature. The specification was released in April 2002 by Microsoft, IBM, and Verisign as

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

"WS-Security." It includes the enhancements to SOAP-based Web Services to provide protection through message authentication mechanisms.

IBM in the WebSphere Application Server makes use of the standard Java Security APIs to provide authentication. The WebSphere Portal is configured so that incoming requests pass through an authentication component, such as the WebSphere Application Server, WebSEAL, or other proxy servers. A user's authorization for a particular resource such as a page or a portlet is handled by the portal engine.

**Content management.** Portals often provide services to facilitate connections to content management sources. For example, in the IBM's WebSphere Portal Server, built-in support is provided for several common content types such as Rich Site Summary (RSS), News Markup Language (NewsML), and Open Content Syndication (OCS), along with most XML and Web browser mark-ups.

Specialized content management engines (or tools) may be used to produce content portlets that can publish information content within a particular portlet. This portlet can be placed on a page and can coexist with other content portlets or application portlets. These content management tools often cooperate with the document management tools, or, as a minimum, allow the authors to refer to documents placed inside or outside the content management tool. The result of this process is a portlet that provides:

- Content in the form of free text;
- References to documents;
- Search capabilities;
- Hyperlinks to applications (other portlets that appear on the same or on a different Web page); and
- Hyperlinks to documents or content entries appearing on other portlets and pages.

**Collaboration services.** Not all portal engines provide the framework for portlet-based collaboration. Collaboration services may exist between portlets on a page, between multiple portal pages or between multiple portal applications. A typical requirement is to allow enterprises to integrate corporate portals with groupware applications such as Microsoft Exchange and Lotus

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Domino and expose collaboration portlets as Web Services. Collaboration services provided by WebSphere Portal use a set of predefined portlets. These portlets allow for team cooperation, chat, e-mail, calendaring, and many other collaborative technologies. Later in this text, we will discuss the programmatic and declarative approach to collaboration and the use of the property broker. In WebSphere 5, cooperative services also support page-to-page communication. However, it is difficult to master this technology. Other leaders in portals, such as BEA Logic and Oracle As Portal, support full scale collaboration.

**Search and taxonomy.** The majority of portals offer 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 configured to force it to follow several layers in a site or to extend beyond several links in a site. For example, the WebSphere Portal engine provides a 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.

Specialized *search engines* such as Verity K2 integrate well with portlets. The portlet provides the search query and the search topology, and the Verity engine returns the results of the search. The portlet then displays the results. Another engine, Lucene, is an open-source search engine API now maintained by the Apache Software Foundation's Jakarta project.

**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 a 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 the EDIT or the CONFIG-

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

URE modes of the portlet. In the EDIT mode, the changed values refer to the instance of the portlet that 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.

In both cases (EDIT or CONFIGURE), the changed values are stored in the underlying database and survive a 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 or CONFIGURE modes must be enabled by the portal administration function, and a record of their use 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.

Since these modes are designed to let the non-technical administrators configure a portlet without any technical knowledge, they can be accessed through normal portlet functions.

**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 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. SSO enables the access to multiple Web Services without requesting additional authentication.

Portals are considered as natural entry points for Web Services in the enterprise. They are able to provide a front end for users to Web Services. A Web service containing business information can be accessed through corresponding portlets that make the business information available to Web Services. The means to achieve this integration lies in WSRP (Web Services for Remote Portlets). WSRP introduces the notion of presentation Web Services. These portals can have a user-facing (presentation-based) Web service that

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

could be provided by a third party or in the same company and just plugged into the portal acting similarly to any other local portlet.

## Endnotes

1 A servlet engine is a Web application that runs servlets. Servlets communicate with the Web browsers.

2 The word "portal" in this context refers to the appearance and composition of the screen, rather than the technology used.

3 JavaServer™Faces is Sun technology that simplifies building user interfaces for JavaServer applications. With the well-defined programming model that JavaServer Faces provides, developers of varying skill levels can quickly and easily build Web applications by assembling reusable UI components in a page, connecting these components to an application data source, and wiring client-generated events to server-side event handlers.

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.

5 More information can be found on the OASIS Web page: http://www.oasis-open.org/home/index.php.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter IX

# Portal Development Framework

To understand portal applications and the principles of their development, we need to distinguish between a portal and a portlet. A *portal* is a framework that lets you plug in new features or extensions called portlets. In the same way that a servlet is an application component within a Web server, a portlet is an application component within a portal. A portal is a Web application. Developing portlets is the most important task in making portals function as the user's window to information and tasks. Portlets are an encapsulation of content and functionality. They are reusable components that combine Web-based content, application functionality, and access to resources.

*Portlets* must be considered as complete applications, having multiple states and view modes. They have messaging and event recognition capabilities. Portlets are assembled in portal pages that make up a portal implementation. An important issue is the design of a portal page as the aggregate of several different portlets.

Portlets are similar to Windows applications in that they present their content in a window-like display on a portal page. The portlet window has a title bar that contains basic controls to maximize, minimize, edit, configure, and provides help to the application. From the portal user's perspective, a portlet is a window on a Web-like page that provides access to a specific service or content.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The portal engine (also called portal server) provides the runtime environment - the portlet container - for the portlets that make up the portal implementation. The portlet container concept is fully compliant with J2EE in the sense of container functionality. The portlet container is responsible for providing the environment for the portlet's lifecycle methods such as instantiating, invoking, and destroying portlets. The user interacts with portlets through Action-URIs[1]. By clicking on a link inside of the portlet, the user can perform certain actions, for example changing the display view.

When a user interacts with the portlet, for example by pressing a link, the action results in a new URL (page). However, the user's experience is such that the page did not change; only the portlet content may have changed. If the user presses the back button, the previous URL is displayed, but nothing has changed on the screen. This is because the portlet maintains its state. It is necessary to use so called *Action-URIs* for navigating forward and backward, rather than using the browser's built in navigation (the back button).

The behavior described previously is often a source of frustration for both the user and the designer. It underlines the differences between a Web application using a single window associated with single component (servlet) and a portal application, which is an aggregation of multiple windows.

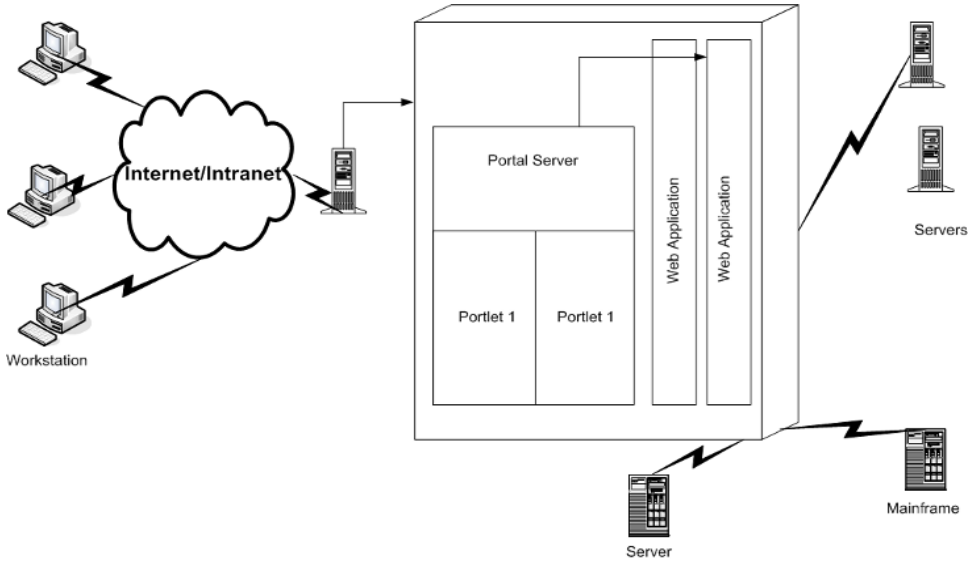
The portal infrastructure supports the core sets of services for the portlets. These services include:

- Access to user profile information,
- A framework for portlets to participate in events,
- A framework to communicate with other portlets,
- Access to remote content,
- Access to user credentials, and
- A framework for storing persistent data.

Part of the conceptual knowledge about portals is the requirement to understand how the portal stack works. In Figure 71, the users connect to the Internet or Intranet using their PCs equipped with a browser. The server has a portal server installed on top of the HTTP server (the HTTP server is designed to handle static Web content) and the Application server (the Application server deals with Web Services and generic Web applications).

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 71. Portlet solution — portlet to Web application*




The Portal server accepts the requests directed to its portal applications and distributes them to appropriate portlets. Configuration files such as web.xml and portlet.xml determine which portlet is invoked to service the request. The request may contain some tasks for a Web application or Web Services. The Web service's responsibility is to retrieve and process data from several components, including Enterprise Java Beans or EIS's mainframes. These components may reside on the same server or on remote servers. Access to remote Web Services is typically via SOAP messages.

## MVC Model and Portlets

We have already briefly mentioned the need to separate presentation from control and business data processing in Web service design. This strategy deals with the separation of concerns at the service level. Because portlets also must be capable of supporting multiple views for multiple devices, the key design pattern is the well-known model-view-controller (MVC) design pattern[2]. It contains three entities:

The *Model* is responsible for encapsulating all the business logic and data required by the system. It should be independent of other components. The

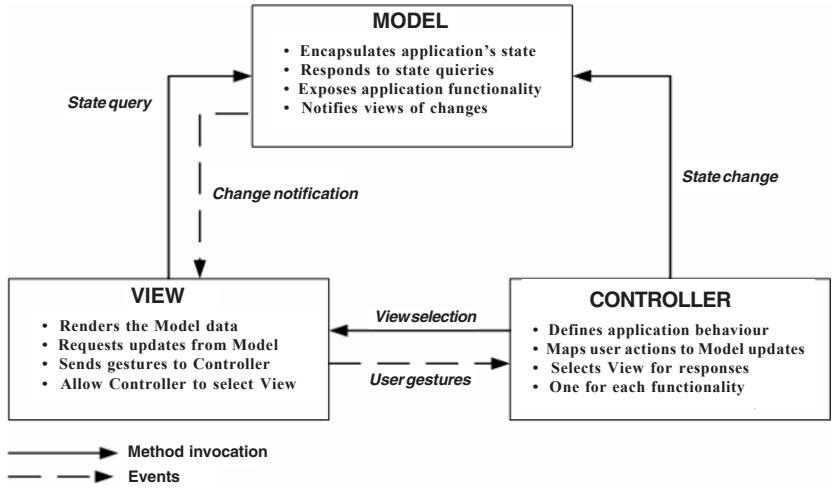
Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Model provides the generic form of business data and applied business rules. In other words, the Model often represents the business process and rules for accessing the data. It also supports generic requests for data and returns generic results so that any View component can use them. In a typical servlet environment, the Model is represented by one or more Java beans, often Entity Java Beans (EJBs), or other Java classes. When a Model's data is required by a portlet, it is typically retrieved from an external data source.

The *View* is the output mechanism used to display the data of the Model component. It is a presentation service. Like the servlet's MVC implementation, the View is traditionally implemented using JSPs or simple HTML. The main difference from the JSP or servlet implementation is that the View returns the aggregated information (because of the HTML markup). The View maintains the consistency of data when the Model changes. There are two variations of the View: in a push Model variation, the View registers itself with the Model for change notifications; in a pull Model, the View calls the Model when it needs to retrieve the most current data. These two variations are represented with connectors to and from the Model in Figure 72. Therefore, the View must not contain page-level tags and must be very aware of the environment in which it is executing. The Portlet APIs provide tag libraries, which help create a dynamic view of resources for the portlet environment.

The *Controller* is the heart of the entire architecture. Each request for data is delivered to the Controller, which is responsible for validating the request,

*Figure 72. The Model-View-Controller design pattern*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

finding which model component provides the required functionality, and selecting the presentation service (View) to display data. The selection of a presentation service is based on the target device or browser. When the presentation service is found, the Controller passes the Model data to the View. The View extracts the specific display data, formats the data for the browser, and renders its output to the browser as part of the portal's aggregation of portlet output.

For portlet development, the MVC pattern has the following characteristics:

- The portlet is only responsible for calling the right Controller, depending on the markup supported by the client.
- Connectors are responsible for physically accessing content sources. Typically, there is one connector per content source type, for example, one connector for POP3 access and one for file-based cache.
- Models represent the content (data) as retrieved through the connector. A model is independent of the presentation.
- Controllers can be used to provide markup-specific content (HTML, cHTML[3], or WML).

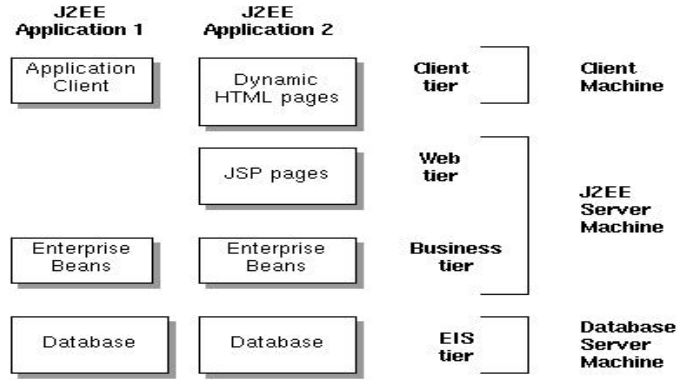
In the MVC structure, there is a distinct separation of data and business logic from presentation service and a controller component for managing the interaction between the data (Model) and the presentation or View. The Controller knows the environment in which the application is invoked, gathers information from the data object to be displayed, and then asks the appropriate View to render the data using the markup language appropriate for the current device.

## Overview of Portal Concepts

A typical J2EE three-tier architecture is in Figure 73. Portal applications follow this architecture. We can view portals as an extension of the Web tier. A portlet container is the extension of a servlet container. A Portal server redirects client requests to appropriate portlets. A Web container manages servlets, JSPs, and filters[4]. A Portlet container provides a runtime environment for a portlet and manages its life cycle. The servlet container is the basis of any portal.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 73. J2EE three tier architecture (http://java.sun.com/developer/technicalArticles/J2EE/Intro)*




Different vendors use different naming conventions for portal components. We used for our examples the IBM WebSphere Portal Engine, and our explanation will adhere to it. The basic portal architecture relies on two components:

- The *Portal server (container)* is a Web-based application server that aggregates, customizes, personalizes, and presents the content of enterprise information systems. A Portal server uses the services of Web application servers to render the collection of portlets (portal page) as one servlet.
- The *Portal engine* enables decoupling of the presentation layer of a Portal server from the portlet implementation details (Figure 74). This allows the portlets to be maintained as discrete components.
- The *Authentication Server* is a third party authentication proxy server situated in front of the Portal engine. Its responsibility is to control the access to portlets by checking access rights during page aggregation or customization. The Struts framework (see sections on Struts Framework) and implementation of the MVC2 Model requires *Java Beans* components to store and retrieve application data.
- The *LDAP directory* is used to query a user's identity and his/her access rights. All deployed portlets must be registered in the Portal registry, and their identity is stored in a database (typically RDBMS). Running portlets

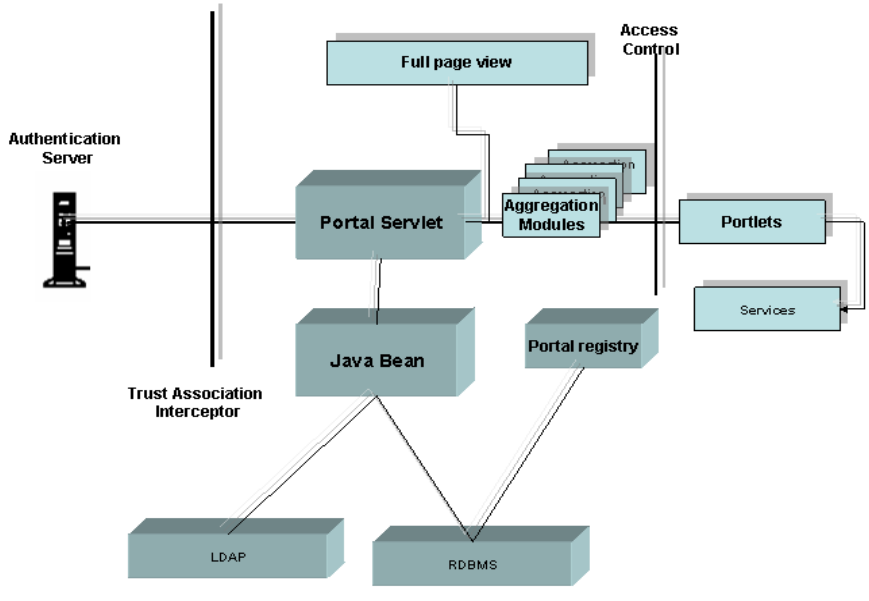
Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

may use all the services provided by the portlet container. In WebSphere Portal, services are components used to extend the functionality such as personalization, search, content management, site analysis, enterprise application integration, collaboration, and access to Web Services.

The *Portal servlet* is the main part of the portal engine that handles the requests made to the portal. The portal requests are handled in three phases:

- The first phase enables the portlets to process requests and send event messages to other portlets or other portal applications, if required.
- The second phase is called the rendering phase. During this phase, the aggregation modules associated with the requesting device (client) render the overall portal page; they collect the fragments of information from all portlets on the page and add standard decorations such as title bars, edit buttons, and so forth.
- In the final stage of rendering, all portlets are transformed into one servlet for response rendering in the client's browser.

*Figure 74. Example of WebSphere portal engine*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Portlet Application and Portal Page

Portlet applications are collections of related portlets and resources that are packaged to form a cohesive information-processing module. All portlets belonging to one application are packaged together, and they share, similar to Web applications, the same context, such as images, property files, and classes. An important feature of portal applications is that the portlets within a portal application can cooperate by exchanging messages[5].

**Concept of a portal page.** A portal page displays content and one or more portlets. The page layout defines the number of content areas within the page and the portlets associated with each content area. Portal pages are often managed by portal administrators. The portal administrator determines the page layout for a specific group of users.

**The portlet window** is the window that surrounds the portlet (includes the title bar and any border images).

**Concept of themes and skins.** Each portal or portal application can have a specific visual theme. Themes represent the overall look and feel of the portal, including colors, images, and fonts. Typically, there are several default themes provided with the standard implementation of Portal. It is possible to have each page in the portal with a different theme associated with it.

The term *skin* refers to the visual appearance of the area surrounding an individual portlet. Each portlet can have its own skin. The skins are available for use with a portlet. The administrator's permission is needed to change the skins.

**Permissions.** Each portal page is subdivided into one or more content areas. Each content area may contain several portlets. The person with the authority to manage the page also controls the authority of others with edit privileges to move or delete the content areas and portlets. Permissions is the term for controlling page settings. Permissions are often granted on the basis of the roles the user holds. For example, the Resource Permissions page in WebSphere defines seven roles:

Portal administrators are allowed to have unrestricted access on all portal resources. Typical administrative responsibilities are:

- Security administrators are allowed to grant access to a resource.
- Delegators are allowed to grant access to other principals.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- Managers are allowed to create, edit, and delete shared resources.
- Editors are allowed to create and edit shared resources.
- Privileged Users are allowed to create private resources.
- Users are allowed to view portal resources.

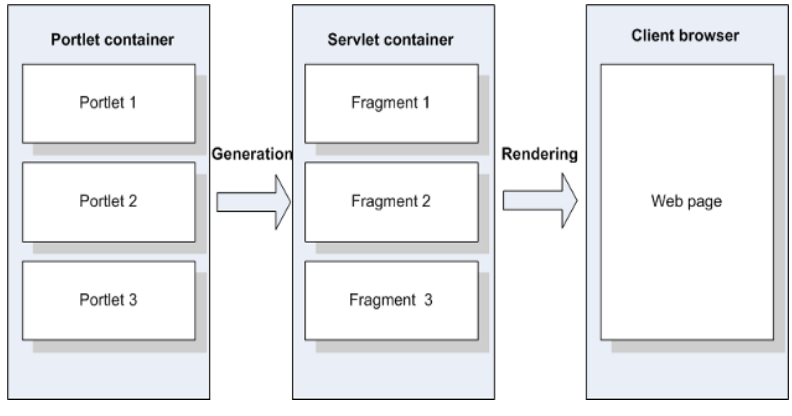
Typical portal transformation is demonstrated in Figure 75. The Portlet container that manages the runtime state of portlets evaluates portlets into fragments in two possible ways: either by making requests from the portlet or by taking a fragment from a cache. Fragments are pieces of HTML markup created by the portlets. Static fragments or frequently used fragments can be cached for faster access. Fragments are then handed to the portal server that aggregates them into portal pages.

Now that you know about portals, portlets, and portal containers at the conceptual level, it is time to have a look at the specifics of building portlets.

## Portal Application Elements

The portal takes advantage of the Web Application Server, making use of its J2EE services. Before we explore details about how the portlets are transformed into servlets and JSPs for rendering, we shall look at typical events that occur inside a portal and how the portal handles a typical client request.

*Figure 75. Transformation of portlets, fragments, and a portal page (gif)*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

In order to understand how portal applications work, we need to introduce some basic elements. Later in this section we will expand the information about each of the following elements:

- Portlet.
- Portlet state determines the state of the Portlet window.
- Portlet mode defines the functional state of a portlet. This is used by the portlet to determine how to manage the rendering request. It means that, depending on the mode, the portlet will render different markup.

## Portlets

A portlet is an application that displays some content in a portlet window. Portlets form the basic building blocks of a portal. A portlet is developed, deployed, managed, and displayed independently of all other portlets. Portlets may have multiple states and view modes. They also can communicate with other portlets by sending messages.

Based on the J2EE container model, portlets run inside the portlet container. Portlet containers are analogous to the way servlets run inside the servlet container of an Application Server. Portal services are components that the Portal uses to extend its functionality. The key functionality includes personalization, search, content management, site analysis, enterprise application integration collaboration, and Web Services. Portlets have access to these services via their container.

Portlets are bound to Java Servlets because they inherit from the HTTPServlet class in the Java class hierarchy, providing the servlet functionality. Servlets and portlets reside in a servlet or portlet container, respectively. However, the portlet container is not a standalone container as is the servlet container. The portlet container is a thin layer implemented on top of the servlet container designed to reuse the functionality provided by the servlet container.

You can think of portlets as being a special subclass of HTTPServlet that includes properties and functionality that allows them to run inside the Portlet container. Though portlets actually run as servlets under the Application Server (Servlet container), they cannot send redirects or errors to the browser directly, forward requests so common with servlets, or write arbitrary markup to the output stream. All communication back to the end user from a portlet is

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

via the aggregation process and property broker framework (this approach will be discussed later).

## Portlet Modes and States

*Portlet modes* are properties of the portal presentation model. Portlet modes allow the portlet to display a different "face" depending on its usage. There are four modes:

- VIEW: This is the initial face of the portlet when created. It generates markup visualizing the portlet state and properties [doViewmethod()].
- HELP: This supports the help mode, and 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: 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.

*Portlet states* determine how the portlet is displayed in the portal. The state of the portlet is stored in the portlet window state object and can be queried for processing that is based on state. The three states of a portlet (to be precise, the portlet window states) 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.
- **Minimized:** Only the portlet title bar is visible inside the portlet page.

The portlet modes and window states are accessible from the portlet window title bar (Figure 76). Clicking on these icons can change the portlet's mode.

The portal administrator grants permission to configure individual pages (of portals or portlets) for one user or for a group of users. He/she can grant the permissions for specified users or user groups to change the page layout in

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 76. Title bar with Edit, Help, Min, Max icons*



order to reflect individual preferences. The meaning of these permissions is that this specific group of users has the authority to change the page layout using the CONFIGURE icon in the task bar. This configuration view allows the authorized users to perform certain actions that change the properties of the portlet and are relevant for all instances of this portlet. This should be compared with the edit view, which allows the authorized user to change certain properties for all instances of the portlet on a given page.

These modes and states must survive longer than one HTTP request-response stateless cycle. Therefore, portlet data is saved, retrieved, or deleted using the PortletData object. The PortletData object is linked to the request object (PortletRequest object), and passed to the portlet during request processing. 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 Lifecycle

The Portlet container is responsible for management of the portlet's lifecycle and for invocation of its life cycle methods. The following list shows details of the portlet's lifecycle and associated calls to handle each request (Figure 77):

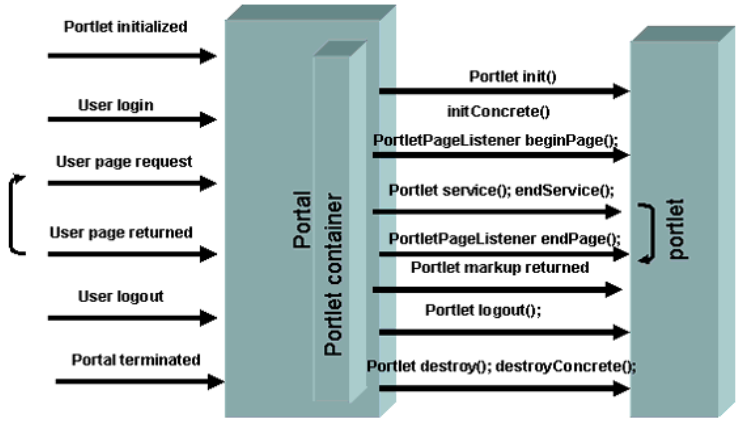
- A Portlet is initialized using its initialization method: Portlet init(). This method belongs to the abstract class portlet[6]. The Portal always instantiates only a single instance of the portlet. This instance is shared among all users the same way that a servlet is shared on an application server.
- After constructing the portlet and before the portlet is accessed for the first time, the concrete portlet is initialized with the PortletSettings object

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

using the initConcrete() method; the PortletSettings object contains persistent data typically set up by the administrator.

- User login: When a user starts the authentication process, the login method is invoked as the response to the user's request [Portlet login();].
- A user's request for a page is processed with the portlet class's service() method. This method is similar to the servlet class' 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, although the order of invocation is not guaranteed. This method can even be called in a different order for each request. This processing differs from servlets where service() is called only once within the request-response cycle — when HTTP GET is processed.
- Portal page return: This is associated with the PortalRequest object. The PortalRequest object is passed to the portlet with login(), beginPage(), endPage(), and service() methods, and it contains the data associated with the user's request.
- User logout: When a user indicates that he/she wants to logout, the Portlet logout() method is called.
- When an administrator takes the concrete portlet out of service on a portal that is running, the destroyConcrete() method is called.
- Portal terminated: This is when a portlet is taken out of service and destroyed. Portlet destroy() is called when a Portal is terminated.

*Figure 77. Portlet APIs in its lifecycle*



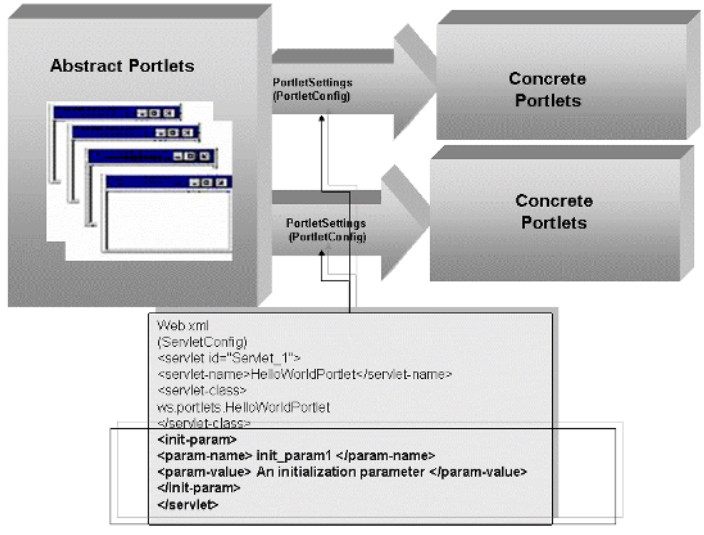

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Portlets are not automatically loaded. If an application needs the portlets to be loaded at startup it must specify this requirement in the portlet configuration file (PortletConfig and Web application deployment file web.xml). When a portlet is called, the Portlet container loads the portlet and invokes the init() method. The init() method assumes responsibility for the initialization of the new instance shared among all users, creating resources and performing the configuration (Figure 78). The settings parameters correspond to servlet initialization parameters in the servlet deployment file — web.xml.

During its lifecycle, the portlet goes through some changes in its settings. The metamorphosis is shown in Figure 79. When the portlet is initialized and an instance of a concrete portlet is created, portlet settings (PortletSettings object contains configuration parameters initially defined in the Web deployment descriptor) are applied to the abstract portlet[7]. This is done by the administrator when he/she deploys a new application (creates the Web archive file .war and associated deployment descriptors) or when a new portlet is loaded. Use of a concrete portlet class or implementation means it is possible to run multiple instances of the same portlet. The same concrete portlet can be shared by multiple users.

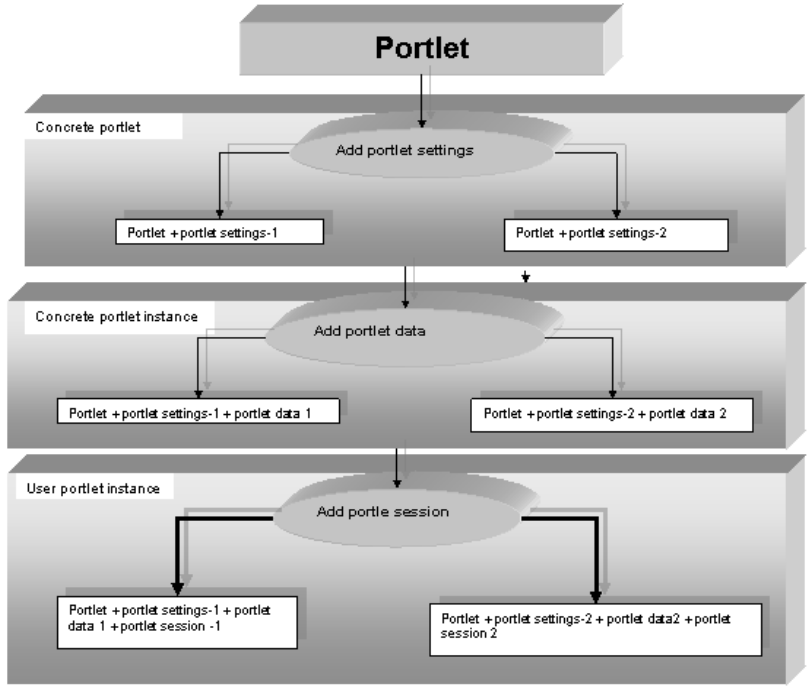
When a concrete portlet is placed on the page, a ConcretePortletInstance is created. The new instance is parameterized by the PortletData object. The

*Figure 78. Concept of the concrete portlet and initial configuration*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 79. Portlet lifecycle and settings*




PortletData object contains configuration data that can be manipulated via the Config mode of the portlet (permissions to edit are required). The PortletData object stores persistent information for a portlet.

When a user accesses the page, a UserPortletInstance is created. A user must login to the portals in order to initiate the creation of a user portlet instance. The portal responds by creating the PortletSession object for each of the user's portlets. The ConcretePortletInstance is now parameterized by a PortletSession object resulting in a new object — UserPortletInstance. There can be many UserPortletInstance objects per one ConcretePortletInstance object. Portlet applications group the related portlets (portlets with the same context) together. By context, we mean all resources, such as java classes, images, skins, and so forth, that belong to the same application

We need to mention the portlet modes again, at this point from the developer's point of view. Portlet modes are defined in portlet deployment descriptor (portlet.xml file) at its initialization phase. However, the actual working configuration of these parameters is maintained by the portlet container in a special object — Portlet.Mode. The current or previous mode for a portlet can be retrieved from the PortletRequest object (PortletRequest.getMode();).

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Core Objects

There are four groups of core objects that 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 the purpose and usage of these core objects.

PortletRequest: this is an object that is passed to the portlet through methods such as login(), beginPage(), endPage(), and service(). It contains requests for specific data. It is similar to HTTPRequest object, as we already discussed in previous sections. This PortletRequest object enables access to important data via requests. 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. The contents of both the *attributes* and the *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 (this is the actual *device* that initiated the request) of 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 Java String type information, which is sent by the user agent, the markup language (for example, "wml"), the MIME type supported by the client (note that retrieving the MIME type does not distinguish between HTML and cHTML markup), and the Capability object with more detailed information about the supported markup language. This mechanism supports the transcoding process for a variety of devices and markup languages.

As mentioned previously, the PortletData object, the PortletSettings object, and the PortletSession object also arrive with the request. There are two other important objects that are used to maintain the portlet's current mode and the portlet window's state. The information about the *current or previous portlet mode* is contained in the 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 the current portlet window* (MINIMIZED, MAXIMIZED, NORMAL). The ModeModifier can be used to change the portlet's mode before it is rendered.

The PortletResponse object encapsulates the information to be returned from the server to the client. This content rendering phase is in principle very similar

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

to servlet response processing. Output is generated, for example, using the Java PrintWriter object.

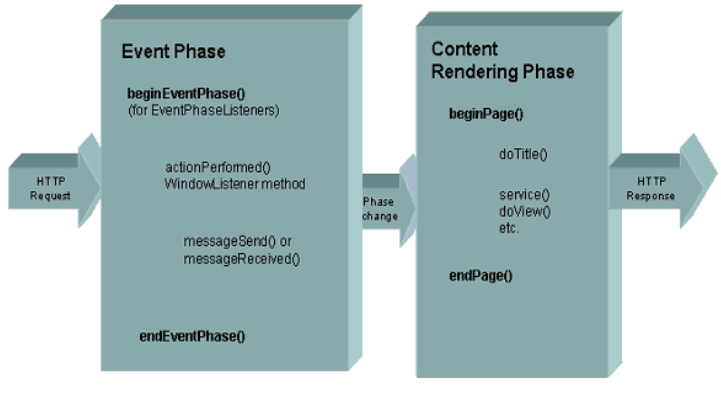
The response object is passed via the beginPage(), endPage(), and service() methods. The response also includes methods for creating the PortletURI object or for qualifying the portlet markup with the portlet's namespace.

The URI points to the portlet instance and can be further extended by adding portlet specific parameters and by attaching actions. For example, let us examine the case where the portlet is in its EDIT mode: the user is entering data and he or she clicks the *Save* button. The portlet must process any posted data before the next markup is generated. This can be arranged by adding the *Save* action to the URI of the *Save* button.

The processing of a request is different from request processing in a servlet. A servlet does all processing in the service() method, which is called only once per HTTP request-response cycle. Furthermore, servlets do not need to exchange messages between themselves. Servlets listen only to their life cycle events. A portlet container supports inter-portlet communication. Portlets have to complete the exchange of messages (communicate) before the rendering phase starts.

In WebSphere Portal, portlet processing is split into two separate phases: event phase processing and content rendering (Figure 80). During event phase processing, the portlet needs to listen to events — life cycle events as well as events occurring at other portlets and components (for example, buttons, menus). The Portlet APIs provide several listeners that extend the listening functionality of ActionListener or WindowListener.

*Figure 80. HTTP request processing in portlet*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

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, an ActionEvent object is created and passed to the ActionListener. This results in an actionPerformed() event that is triggered on the registered portlet followed by doView(), a rendering method. All this activity happens before the rendering phase. The other remaining portlets on the page do not react to this event and will process only the rendering request — the doView() method.

The portal calls the service() method when the portlet is required to render its content. The service() method is typically called many times during a request-response cycle. The order of calls to the service() method is not guaranteed. The service() method is also called either after actionPerformed() is invoked or when page/portlet refreshing is required. Once the content rendering has started, no more messages are delivered to the portlet (they are discarded, and some could be lost).

PortletSession holds user specific data for the concrete 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[8] — with read and write access. The programmer must make sure not to use instance attributes for user-specific data. On the other hand, he has 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[9].

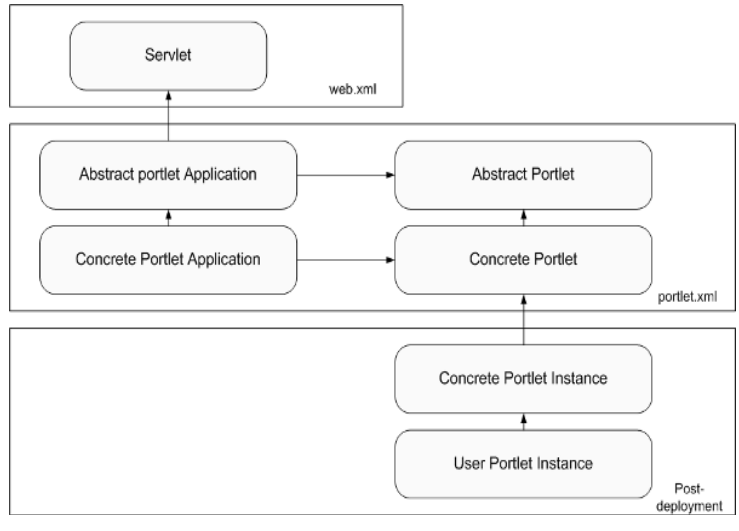
Note that the Servlet parameters are defined in web.xml file, while the abstract portlet definitions are in file portlet.xml. Similar to the servlet's HttpSession object, PortletSession is not available on an anonymous page. However, in cases where the administrator places a portlet on an unauthenticated welcome page, the portlet should provide code to handle it.

In addition to the core objects[10], there are also configuration objects that have been mentioned previously (refer to Figure 81). Due to their unique roles and importance, we briefly reiterate their usage with regard to the effect of their respective descriptors (web.xml, portlet.xml):

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- PortletConfig object provides an initial configuration for the abstract aspects of a portlet. You can find this information in the portlet.xmxl file (config-param tag). 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 object holds data for the concrete portlet instance. The scope of the 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 users.
  - 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.
- PortletApplicationSetting object provides the concrete portlet application with its dynamic data configuration, shared across all concrete portlets in the application.

*Figure 81. Portlet architecture*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 82. Code example of obtaining window mode and configuration settings*

```
boolean maxSuppported = getPortletConfig().supports(PortletWindow.State.MAXIMIZE);
boolean minSupported = getPortletConfig().supports(PortletWindow.State.MINIMIZED);
boolean viewSupported = getPortletConfig().supports(Portlet.Mode.VIEW, request.getClient());
boolean editSupported = getPortletConfig().supports(Portlet.Mode.EDIT, request.getClient());
boolean configureSupported = getPortletConfig().supports(Portlet.Mode.CONFIGURE, request.getClient());
boolean helpSup = getPortletConfig().supports(Portlet.Mode.HELP, request.getClient());
PortletContext context = getPortletConfig().getContext();
```

A PortletConfig object can also be used to determine supported modes and states. Furthermore, this object also provides access to the PortletContext object. The PortletConfig is accessible, for example, via the getPortletConfig() method of the PortletAdapter class. Some methods available in this object are:

- supports(): This method can accept a PortletWindow.State object or Portlet.Mode object and returns a Boolean value indicating whether or not the state or mode is supported by the portlet.
- getContext(): This method will return a PortletContext object and provides the capability to find out which states and modes are supported.

The PortletContext provides a mechanism for the portlet to access the services of the portlet container in which it is running. For example, the PortletContext provides access to the PortletLog, servlet context parameters, as well as any services hosted by the portal, such as the Credentials Vault, persistent connections, and possibly other custom services. The parameters accessed by the PortletContext are the context parameters set in the file web.xml. These parameters are common to all portlets deployed in the same web.xml, regardless of the application to which they belong.

## Portlet Events Phase and Messaging

In the previous section, we mentioned the impact of a portlet listening to an ActionEvent on processing, in particular the rendering phase. A portal server provides the environment for portlets to exchange messages and data. A user can be presented with a portlet on a page that contains a list of choices, for

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

example, a list of subjects offered in a semester at a university. Selecting a subject causes another portlet on the same page to be updated with details of this selection (for example, the subject description retrieved from the online student handbook). This type of processing — propagating requests through multiple portlets — is achieved with portlet messaging and events.

There are three important constraints with regard to portlet messaging[11]:

- Communicating portlets must be on the same page. The portlets that communicate through messaging must reside on the same page. This limitation is introduced for performance reasons only. Portlets are configured on the same page using the XML configuration file and loaded at the installation time. Portlets wishing to receive information from other portlets on the page must listen to incoming events. For example, when a user clicks on a link or a button, the click generates an action event. To receive notification of the event, the portlet must have the appropriate event listener implemented within the portlet class.
- Portlets that implement the PortletMessage interface must belong to the same application (they must be packaged in the same .war file); this limitation can be avoided by using inheritance and extending the DefaultPortletMessage superclass.
- Portlets deployed as Web services cannot send or receive messages.

In a typical scenario, one portlet detects a certain condition and prepares a message that is then sent to another portlet. The receiving portlet obtains the message from the message listener and processes it. Portlets have the capability to both send and receive messages.

Portlets communicate using portlet actions and message listeners. When the link for the portlet is clicked or another event occurs, the action listener is called, which then sends a portlet message to the portlet that is the target of the action.

Portlets in different applications can only communicate through default portlet message objects. Default portlet message objects can only carry strings. For example, the WebSphere Portal 5.01 uses a unique class loader for each portlet application to provide security between applications. The message is typically a custom Java object unique to the application. Since messaging portlets must share this message object, they must share the same class loader, and therefore, they must be part of the same portlet application.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The messages and events come in three different flavors:

- Action events are generated when an 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; and
- 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 portlet container delivers all events to the respective event listeners (and therefore to the portlets) before generating any content that is to be returned to the portal page. Event conflicts, such as when a listener processing the event finds that another event needs to be generated, are handled by the portlet container. A 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. The container only guarantees that the event will be delivered and executed before the content generation phase. The concept of listeners and specifically action listeners is the same as in many aspects of Java technology.

We have already mentioned differences in processing and rendering between portlets and servlets. A servlet does all of its processing in the service() method. A portlet uses two-phase processing - action and service processing. Action processing must complete before a portlet's rendering phase starts.

During the action processing phase, the portlet implements an ActionListener interface and listens to events. When the ActionEvent arrives the actionPerformed() method is invoked with a parameter representing the ActionEvent object. The ActionEvent object is generated, for example, a submit button, and it describes the triggering event. When the actionPerformed() method is invoked, a response object is not available because this is not a rendering phase. All state changes should be handled during action processing, and portlets should use the service phase only to render portlet output. The service() method can be called following the actionPerformed() invocation or when the portal page is refreshed. Consider a page with two portlets, Portlet_1 and Portlet_2. The user clicks on a link in Portlet_1, which results in actionPerformed() and doView() processing on Portlet_1. The page is refreshed with the doView() method called for Portlet_2. The content genera-

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

tion phase has started [using service(), beginPage(), and endPage() methods]. During this phase, all further events are on hold. For example, messages cannot be sent from within the service() method, doView(), or other content generation methods. Attempts to send a message during the content generation phase will result in an exception.

The portal event handling and message processing is executed in four steps in the predefined order:

- **All action events are processed:** The user makes a request to the portal. The portal receives the request and decodes the action's URI that has been sent by the client and propagates an action event to the appropriate portlet. The receiving portlet's action listener is called to process an action event. This is the appropriate time to send messages to other portlets - during the action event processing.
- **All message events are processed:** If a message is sent to a portlet, the portlet's message listener is called to process the message. Portlets can send multiple messages, and so they send messages as responses to received messages. Message processing continues until there are no more message events pending.
- **All window events are processed:** Window events typically deal with sizing operations, such as maximize, minimize, restore, and the portlet's ability to request a specific size. These events are sent to all portlets affected by the resizing. All window events are queued, and their processing continues until there are no more window events pending.
- **The rendering process starts:** When the event processing is completed, the portal aggregator starts calling each container on the page, and the rendering phase is started. When rendering is completed, the page is returned to the user.

## Page Aggregation Process (Rendering Phase)

Portals allow users to select sets of portlets they wish to work with and provide a framework for displaying those portlets in a consistent fashion. A set of applications that should be presented in a common environment is called a page. Page aggregation is the process of collecting information about the portlets (their size, customized features, and content) requested by the user and the device on which the page should be displayed. This information is then used to create a page that is appropriate for the device.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The page layout is divided into rows and columns that are managed by the relevant containers. The row and column containers aggregate the portlets on the page in a matrix, to which the controls are added.

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 following information is required about the user:

- Each user is authenticated during the login process, and this information is kept for the duration of the session. At this point, we are not concerned with the storage location.
- The Client identifies the device through which the user is connected. It is obtained from the request's header and kept for the duration of the session.
- The markup and markup version belong to the device category, and they are obtained from the request object as part of the client information. The supported markups are HTML, cHTML, and WML. Additional markups can be added (WebSphere Portal provides a Markup Portlet Manager for adding new markups). The browser version is also needed for the supported markup. For example, *ie5* for the Internet Explorer browsers and *ns* for the Netscape browsers.
- The language is displayed, based on user preferences or browser set up. As an example, we are familiar with the WebSphere Portal, which determines the language to be displayed via the following algorithm. If the user is logged in, the portal user interface is displayed in the preferred language of the user. If no preferred language is set, the portal UI is displayed in the language set by the client browser, if available. If no browser language is available, the portal UI is displayed in the default language set for the portal. Portlets that do not support any of the previously-mentioned scenarios display their UI in the portlet's default language.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

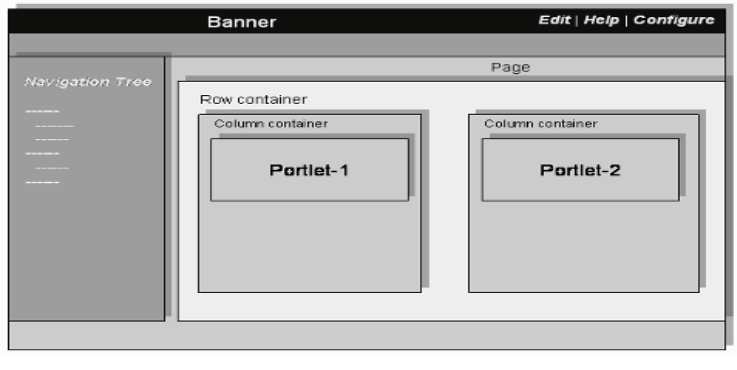
The access control facility determines the level of access to portal pages. Each page is associated with a theme. The name of the active theme is read directly from the currently active page. Depending on the interactions of the user with the portal, different screens may be presented. The screen holds the output of the portlets on a page.

When the portal receives a request, it determines the active page for the current user and renders the page. Once the active page is determined, the layout of this page is used to aggregate the content of the defined applications, arrange the output, and integrate everything into a complete page. A basic portal page layout is shown in Figure 83.

Each page is defined in terms of templates for the portal window, screen, and screen components:

- A theme determines the global look and feel of the pages. Themes affect the navigational structure, banner, colors and fonts, available portlet skins, and other visual elements of a page.
- A skin defines the frame around a portlet. There can be a different skin for each portlet. Each theme must have a default skin associated with it. This skin is used for all pages, unless specified otherwise by the user[12].
- The portal window is composed of the banner and the portal page. The top area of the window is typically taken by the standard information for the enterprise. The banner contains the greeting, a page selection box, tabs to select other pages in the page group, and some additional controls for interacting with the portal, such as a log in button.

*Figure 83. Basic portal page layout*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- The screen holds the output of the portlets on the currently selected page. Another page element called the node represents the hierarchical structure of the presentation and other elements in the portal. Nodes include pages, labels, or URLs, and they are used to navigate the portal structure. The portal has a tree structure that is used to organize the portal into branch nodes, which belong to other nodes that are higher in the tree. The single highest node in the portal is called the content root. Nodes are represented and accessed from the portal navigation menu.
- A page is a type of node that provides portal content, similar to a page on any Web site. Each page displays content that can be customized for the portal user.
- A label is a type of node that has a name and is used to hold other nodes.
- A URL is a type of node that can open locations within the portal or external Web sites.
- A container is an area on a page that contains content. There are three types of containers: row containers, column containers, and cell containers. The content is placed in a container that spans the width of the page (row) or the height of the page (column).

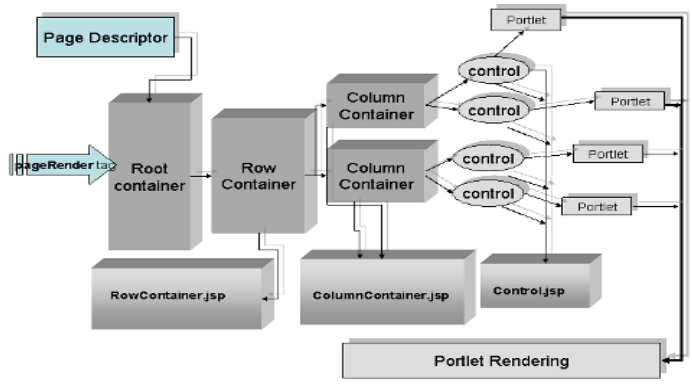
A row container allows portlets to be arranged in a horizontal format. A column container allows portlets to be arranged in a vertical format. Control is maintained by the frame around the portlet. It builds the bar above the portlet output, including buttons to control the state and view of the portlet.

Rendering of page components is done using resources and descriptors, such as JSPs and JSP tags, images, and style sheets. The relationship of the components and their descriptors is in Figure 84. These resources are located in the file structure, which the portal server uses to locate the correct components during the rendering process. The matrix containers RowContainer.jsp and ColumnContainer.jsp render rows and columns, respectively, using their tags. Each column points to one or more rows (children). The Control.jsp descriptor renders the title and control buttons of a portlet and may also render the frame for the portlet.

The root container contains all other containers (row containers, column containers, and control containers) for the given page. The root container is used only as a starting point for the aggregation, and therefore, it does not render itself. The root container triggers the rendering of all the child components (columns and rows) in the page hierarchy. When the pageRender tag is

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 84. Page aggregation mechanism*




found in the relevant jsp file, the root container starts the rendering process of its children. The root container is not visible on the screen. Each child of the root container invokes its template (JSP), which is responsible for rendering all the content of its child. If the child contains further child components, the appropriate tags for children start executing the rendering of all these components, one at a time. Each component invokes its own template, which in turn invokes more components until the leaf of each branch is reached. This results in control moving between component classes and their respective JSPs. Each component writes its content to the servlet output stream.

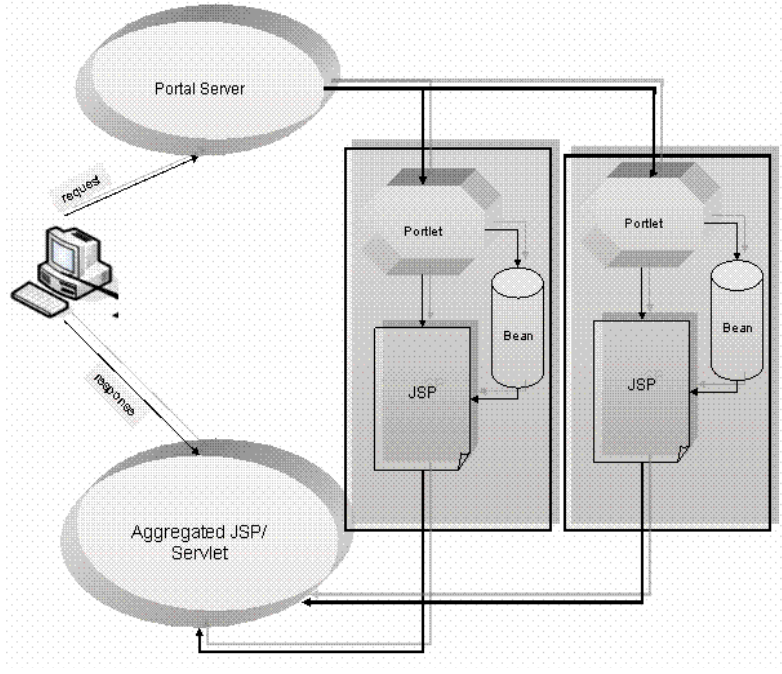
When a Control.jsp is reached, it adds controls such as buttons and a title to the fragment and passes the portlet's output stream to the servlet's output stream. When the entire tree has been processed, the output stream is closed, and the servlet is sent back to the requesting client (the rendering phase is completed).

Each portlet follows the MVC2 pattern. The content of the portlet is updated with regard to the user's request and messages received from other portlets. The *Model* component is typically implemented as a Java Bean(s) for each portlet, and it is responsible for retrieving updated content. The updated content is fed to the output stream associated with the response object from each portlet, and it is further channelled to the corresponding JSP. All JSPs are then aggregated in the servlet. The response is prepared for the rendering (Figure 85).

Portal APIs support many listeners. Some portlets implement listeners such as the PortletSessionListener and the PortletPageListener for recognizing the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 85. Page content aggregation as MVC2 pattern*




life cycle stage of a user's portlet instance during life cycle methods. For example, we can listen to login() and logout() to maintain user session information, or beginPage() and endPage() to react to page events. These listeners are useful because some methods, service() method in particular, are not called by the portal server in a guaranteed order. Listening to a portlet's life cycle makes it possible to insert additional behavior or processing at the beginning or end of the page.

To be more specific, the PortletSessionListener allows a portlet to detect a login() situation, which signals the start of a user instance, initialization of the user's session instance, and storing of the user attributes in the session object. When a user logs out (logout() method), the user's session with the portal is terminated. The portlet should then clean up any resources for the portlet user instance. This could include cleaning up the authentication state of the user or cookies. Similar to Web services or servlets, any state on the backend application should be marked as terminated to avoid any confusion in the Model component.

PortletPageListener allows a portlet to insert markup at the beginning and end of a page. A portlet has no control of the order in which the output from all of

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

the portlets on the page is written. The beginPage() method is called for each portlet residing on the page. This happens at the beginning of each page and before the service() method of any portlet on the page is called. Like the service() method, the beginPage() method for each portlet on the page is not called in a guaranteed order and can even be called in a different order for each request. Prior to the rendering of portlets, the output of beginPage() is inserted into the page aggregation area. This method allows a portlet to output Javascript that is visible to all portlet's service() methods. Note that this method cannot be used to add output to the <HEAD> section or to redirect the portal page. The output is added as the very first entry after the <BODY> tag.

The endPage() method is called for each portlet residing on the page at the end of each page and after all service() methods of all portlets on the page are called. The endPage() method is also not called in a guaranteed order and can even be called in a different order for each request. The knowledge regarding when the method is called is useful if the portlet wants to insert Javascript at the end of the page to be processed, after all other elements of the page have been written.

## Environment For Portals

This section takes you through the fundamentals of creating and deploying portlets. We have included some information about setting up the environment for development and deployment, the structure of deployment directories, and how to create the deployment descriptors. We wish to assert that this is not intended to provide a complete tutorial guide for programmers. The intention is to point to the necessary activities in the deployment process so as to guide managers in the portal development field.

Although it is possible to acquire open source portal engines such as Apache Jakarta Pluto and Jetspeed, we decided to use the IBM WebSphere Portal for Multiplatforms V5.0.2 as an example of a portal engine with commercial strength. WebSphere Portal for Multiplatforms V5.0.2 provides the runtime and development environment for running and developing portlets. It is suitable for running on different hardware platforms. It is composed of several software components that support the entire lifecycle of a portal application. They include:

- Development environment for developers to create their software and test it.
- Test environment to emulate the production environment, thus enabling

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

testing of the suitability and scalability. In this environment we would recommend the use of WebSphere Portal 5.0.2, the database (it can be the default small-footprint database Cloudscape[13] or DB2), and the LDAP.

- The production environment is the runtime environment for daily services to the end user. It uses all components that are necessary, and it should be configured with server clusters.

The basic components of a Portal include:

- **Operating System:** There is a range of supported operating systems, for example, Windows 2000 Server, Windows 2003 Server, Linux Intel Servers based on Linux 2.4 kernel. However, you need to always consult the vendor's recommendation and installation guide. For example, WebSphere Portal Server cannot be installed on some Windows and Linux platforms (for example, WindowsXP, Linux Fedora).
- **Web Server:** Here you have a choice between many servers (for example, Apache server, IBM HttpServer, Microsoft IIS5 server, and Sun ONE Enterprise Edition.
- **Application Server:** IBM WebSphere Application server is recommended for WebSphere Portal. We have also experimented with both the Sun ONE Portal server and Sun Application Server.
- **Portal Server:** We have used IBM's WebSphere Portal for Multiplatforms 5.02 installed on Microsoft Windows Server 2003.
- **Database:** There must be a relational database, for example Cloudscape (this comes with the WebSphere Portal and Application servers), IBM DB2, Informix, Oracle, SQL Server, and so forth.
- **LDAP Directory:** Can be represented by any of the following directories: IBM Directory Server, IBM Lotus Domino Enterprise Server, NDS eDirectory (Novell), Sun ONE Directory Server, and Windows 2000 Active Directory.
- **Portal Content Publishing:** A browser-based system for online authoring, approving, editing, composing, and publishing content material, such as text, graphics, menus, and hyperlinks, in the portal environment.
- **Collaboration Center:** Includes portlets for searching company directories and organizational charts, managing Lotus Team Workplace (for-

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

merly Lotus QuickPlace) workspaces, and interacting with Lotus Web Conferencing (formerly Lotus Sametime).

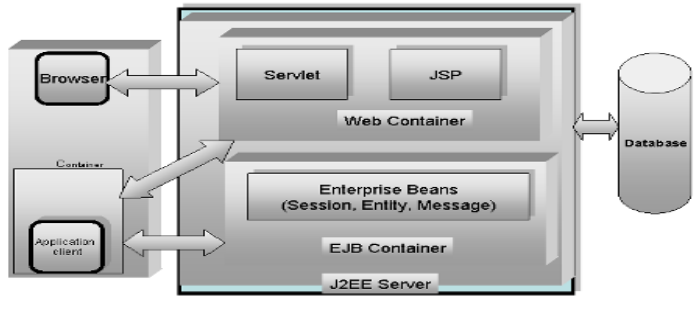
- **Java Virtual Machine (JVM):** It is important to follow the vendor recommendations for JVM runtime, as well as the programming environment.

We need to mention some prerequisites regarding hardware requirements. They are the hard requirements for WebSphere Portal running on WindowsServer 2003. The minimum requirements are processor Pentium® 800 MHz or equivalent, Pentium IV 1.4 GHz or higher, memory 1024 MB or better, 2 GB or more. Furthermore, you need a network adapter to carry IP packets and a static IP address (this means that you require a fully qualified, configured host name because the Portal system must be able to resolve an IP address from its fully qualified host name.)

## Portal Deployment

Because the portal applications are extensions of Web applications, there are some similarities in the deployment process. First, let us look at the standard composition of a deployment descriptor for a Web service and for Web-based applications that are compliant with the J2EE architecture (Figure 86). The client application can be either browser-based or a thick client. Both types of clients access the Web container (Web server) over HTTP protocol to obtain a response to the request in the form of the dynamic content generated either by Web components (servlets and JSP) or by the EJB container. The Web container and the EJB container do not have to reside on the same machine. The

*Figure 86. J2EE Web service compliant configuration*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

persistent storage (database) is accessed via JDBC or using the persistency features of the EJB container and Entity Beans.

Web-based applications have their descriptors stored in a web.xml file. Portlets are assembled into servlets, and the descriptor — web.xml — defining the servlet configuration is similar to an ordinary servlet. The abstract and concrete portlet parameters are defined in the portlet.xml file. Both descriptors have syntax and semantics that are well defined and easy to understand.

The definition of the servlets in web.xml is prescribed in the form of the directory structure shown in Figure 87. The servlet's descriptors must be in the same order as the definition of the portlets in the portlet.xml. All descriptors and resources are stored in the Web archive file — *.war. This archive file is portable and can be moved to any Web container.

As with servlets, portlets are also packaged as *.war or *.ear[14] files with the Web application descriptor web.xml. Each portlet is defined as a servlet within the Web application, including unique identifiers for each portlet, the portlet class, and initialization parameters. The portlet descriptor file (portlet.xml)

*Figure 87. Portal deployment descriptors — setting up file structures*

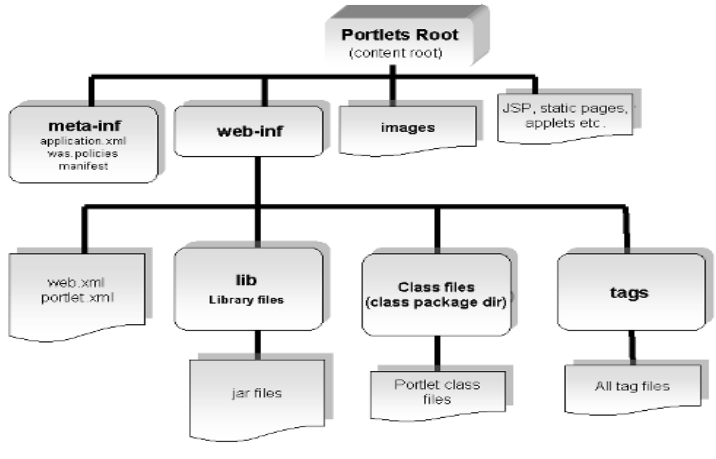



*Figure 88. prtlet.xml opening declarations*

```
<?xml version="1.0"?>
<!DOCTYPE portlet-app-def PUBLIC "-//IBM//DTD Portlet Application 1.1//EN"
"portlet_1.1.dtd">
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

begins with the following XML and !DOCTYPE declarations (Figure 88). It specifies the type of dtd schema used in the descriptor.

Servlets, portlets, and concrete portlets are connected through their respective deployment descriptors. The definition of the servlets in the web.xml file must be in the same order as the definition of portlets in the portlet.xml. The servlet identifier must be referenced by the respective portlet deployment descriptor, using the href attribute of the <portlet> tag. As shown in Figure 89, the href attribute indicates the path of the Web application descriptor in the archive (*.war) file appended by the servlet ID as the anchor. The portlet *Monash* will be rendered in the *servlet_2*. In the file portlet.xml, we need to declare and link the relevant servlet from the web.xml descriptor to this portlet within portlet-name tags.

Similarly, in web.xml file, we need to declare a portlet name within servlet-name tags (Figure 90).

*Figure 89. Portlet and servlet descriptor bindings in portlet.xml*

```
<portlet id="Portlet_2"
href="WEB-INF/web.xml#Servlet_2">
        <portlet-name>Monash</portlet-name>

...
</portlet>


<portlet id="Portlet_1"
href="WEB-INF/web.xml#Servlet_1">
        <portlet-name>HelloWorld</portlet-name>

...
</portlet>
```

*Figure 90. Portlet declaration in web.xml*

```
<servlet id="Servlet_1">
<servlet-name>HelloWorldPortlet</servlet-name>
...
</servlet>

Servlet id="Servlet_2">
<servlet-name>MonashPortlet</servlet-name>
...
</servlet>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 91. Concrete portlet reference href*

```
<portlet id="Portlet_1"
href="WEB-INF/web.xml#Servlet_1">
<portlet-name>HelloWorld</portlet-name>
...
</portlet>
<concrete-portlet href="#Portlet_1">
<portlet-name>My World</portlet-name>
...
</concrete-portlet>
```

Each concrete portlet definition indicates its parent portlet by using href tag of the concrete portlet (Figure 91).

We would like to point to some functional extensions implemented in IBM WebSphere Portal for Multiplatforms 5.02 that we have found useful. Among them, there is functionality to add portlet menus to the navigation tree of the portal and to enable portlets to send message events to other portlets, even if no portlet action event is fired (implementing Event phase listeners). For portlets that implement this interface, the beginEventPhase() method is called before actionPerformed() and prior to messages being sent. The portlet needs to be placed on the currently displayed page to receive a notification, but it receives the notification even when it is not displayed.

## Performance and Implementation

**Use of namespace.** Although a servlet may be a single piece of a much larger Web application, at any given point in time, only a single servlet is fulfilling a user's request. This provides a great deal of predictability in that, as the master controller, it can guarantee what is executed and returned to the client. This is not true of portlets. Each portal page is potentially the aggregation of several portlets. Furthermore, when a servlet executes and returns content to the user, it can be sure that the content it returns will not be affected by any other servlet in the system. This is not true of portlets. A portlet has the ability to write markup to the top of the page even though its normal content is placed inside a cell in a table. This provides a mechanism to include JavaScript functionality that the portlet may need. Therefore, any portlet attempting to access a "forbidden"

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

zone may cause some damage. This is because many portlets can exist on a page, and it is possible that more than one portlet will produce content with like-named elements. There is a risk of namespace collision between elements, causing functional problems within the page. The way to avoid such problems is to use the <portletAPI:encodeNamespace/> tag to encode such resources with the portlet instance name.

**Passing data to view.** Portlets often need to pass some data to the View (typically implemented by a JSP) for rendering. The efficient way of passing data is to use a PortletRequest object and to enclose data in a bean. This way, when the request is completed, the data falls out of scope and is garbage collected. Passing data as a bean allows the JSP to refer to the data as properties on the bean with functions that are intrinsic to the JSP syntax.

**Packaging.** Portlets and servlets packaging follow similar rules. Common functions should be made available to portlets and servlets externally. If the portlet or servlet contains common functions that are replicated across several portlets/servlets, these functions should be isolated and packaged as another archive JAR file; they should also be placed in a location that is in each portlet's or servlet's classpath (typically relative to the assembly root).

**Use of services.** Instead of using additional archive files, it is possible to build a Portlet Service functionality from the common functions that can then be accessed using the common PorletService interface. The PorletService interface enables portlets to use services via dynamic discovery. A PortletService is accessed from the PortletContext.getService() method that looks up the appropriate factory for the service, creates the service, and returns it to the portlet. The PortletService can be invoked only from within a portlet. Often, we need to access various services provided by different vendors, for example, a SearchService, LocationService, or a MailService.

**Portlet configuration data.** Portlets with similar functions can be combined into a single portlet with multiple configuration settings. If during the design phase you find several similar portlets should be coded, it makes sense to actually develop a single portlet that can behave differently based on configuration settings. The common portlet code is specified as part of the portlet application's abstract portlet definition. The application's various concrete portlet definitions have unique configuration parameters (refer to PortletSettings object) that can be used to customize the behavior of the common portlet code.

Servlet configuration is a good place for storing static initialization information for a portlet. For example, the install directory or path information should be

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

stored as part of the servlet initialization. This data is taken from the portlet's (actually servlet) web.xml file, is read only once, and is accessible using the getServletConfig() method on the portlet.

**Session maintenance.** In some cases we need to maintain a session between a portlet and the user. This requires storing the state of the conversation between the user and portlet. In this case, the PortletSession object 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 object. Any persistent data must be stored using PortletData. Portlet user instances differ from each other only by the transient data stored in their PortletSession object.

The PortletSession object is a convenient place to store global data that is user and portlet specific and used during multiple portlet requests. However, there is considerable overhead in managing the session, both in CPU cycles as well as JVM heap consumption. Since sessions may not expire for long periods of time, the data contained in the sessions will remain resident in active memory even after the user is done with the portlet. This problem is similar to using an HTTPSession object with servlets. Therefore, in both cases, the developer should consider carefully what is stored in the PortletSession/HTTPSession objects and how and they should be destroyed or garbage collected.

From experience, we believe the PortletSession object should store only the data that is user specific and cannot be recreated by any other means, such as portlet state information. For example, parsed configuration data (from PortletSettings object) can be recreated from the database at any time, and the PortletSession object should not be used for storage of this type of information. On the other hand, if the portlet runs in a cluster environment where the session is being serialized to a shared database, then every stored data in the session would have to be serializable, and possibly temporal storage provided by PortletSession object is justified.

Like the HttpSession, a PortletSession is not available on an anonymous page. An administrator can place a portlet on an unauthenticated page, such as a Welcome page that is accessible to all users. It is the developer's responsibility to provide a code in the portlet to handle it. The way of handling depends on particular implementation requirements.

The situation is different after a user logs into the portal. Then, a PortletSession object is automatically created for each portlet on a page. To obtain a PortletSession object reference, the getSession() method on the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

PortletRequest object is called. The method returns the current session, or, if there is no current session and the given parameter "create" is true, it creates one and returns it.

**Concurrency handling.** Since portlets are servlets, similar concurrency and performance considerations apply to both. Similar to servlets, a single instance of the portlet's Java class is shared among all requests. Data stored in instance variables will be accessible across requests and across users and can collide with other requests. Data must be passed to internal methods as parameters. Furthermore, information stored in a portlet's instance variables is shared between all *concrete portlet* instances and even between all concrete portlets—with read and write access. The developer must ensure that instance attributes are not used for user-specific data.

Just as with servlet programming, a developer must ensure that portlets are thread safe; this requires limiting the use of expensive string operations, avoiding long running loops, and minimizing the number of objects that are created. If concurrent blocking is necessary (synchronization), developers should block an object rather than method blocking (use of synchronized key word). Another optimization is to strictly use JSP for rendering the portlet's views. In general, views created with JSP are faster than views created with XSL.

**CPU optimization issue.** Every time a new servlet or portlet is created, the new thread is spawn for it. The number of threads processing portlets and servlets is not unlimited. It is important for each portlet to complete processing as quickly as possible so that response time for the whole page is optimized.

**Access to remote locations.** It is typical that many portlets are invoked to process one single request. Each portlet adds its content into the overall page. WebSphere Portal Server for Multiplatform 5.02 allows some portlets to be rendered in parallel. In this fashion the portal server assembles all the markup fragments when all the portlets finish or time out. The parallelism in portlet rendering provides a benefit for applications in which a high proportion of the portlets is accessing remote locations (and not using WSRP) to fetch the content they render.

**Caching.** Portlet output can also be cached. The caching policies are configured in the portlet deployment descriptor, including whether or not the portlet markup can be shared among users or is user specific.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

# Endnotes

1 URI (Uniform Resource Identifier) is a formatted string that serves as an identifier for a resource, typically on the Internet. URIs are used in HTML to identify the anchors of hyperlinks. URIs in common practice include Uniform Resource Locators (URLs) http://www.w3.org/MarkUp/html-spec/html-spec_11.html - BIB2 and Relative URLs. Action tag specifies the URI for the action.

2 A *design pattern* is a common strategy for a problem that occurs repeatedly. It is a generalized solution. MVC design pattern was first published in the classic book *Design Patterns: Elements of Reusable Object-Oriented Software* by Gamma, Helm, Johnson, and Vlissides (1995).

3 Short for ***compact HTML***, a subset of HTML for small information devices, such as smart phones and PDAs, cHTML is essentially a pared down version of regular HTML over the Internet. Because small devices such as cellular phones have hardware restrictions such as small memory, low power CPUs, limited or no storage capabilities, small mono-colour display screens, single-character font, and restricted input methods (the absence of a keyboard or a mouse), there is a need for a simpler form of HTML.

4 Filters are Web service or user-provided interceptors that can be implemented for incoming or outgoing servlets messages. They are input and output filters.

5 This topic will be later discussed in detail.

6 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.

7 These abstract portlets are defined as public abstract class AbstractPortlet extends PortletAdapter.

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

9 Object serialization represents an object as a sequence of bytes that can be stored in a file or transferred to another computer

10 Core objects are PortletRequest, PortletResponse, and PortletSession.

11 Note that this restriction is applicable only to IBM's WebSphere Portal5 and not to Sun Microsystem's Portal Server ONE (6.1).

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

[12] Both themes and skins are implemented as JSP files. Themes can use images and style sheets as well. Skins and themes are deployed independently.

[13] IBM Cloudscape™V10.0 is a pure, open source-based Java relational database management system that can be embedded in Java programs and used for online transaction processing (OLTP).

[14] EAR stands for Enterprise Archive file, which is a package for all necessary J2EE application and deployment files.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter X

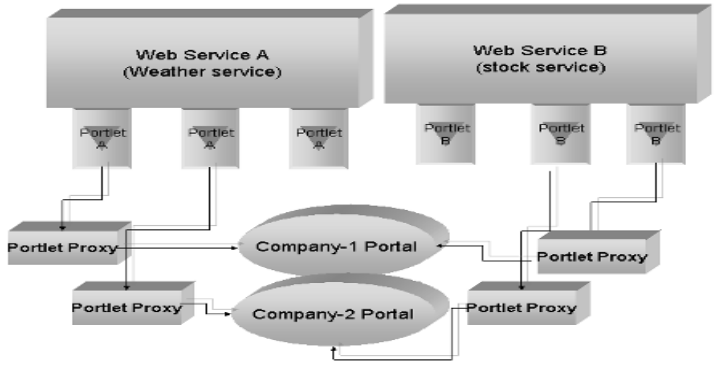
# Accessing Remote Locations

Most of today's portal implementations provide a model that facilitates plugging various components (portlets) into the portal infrastructure. Portlets run locally on the portal server, process input data, and render output. A local portlet and a good caching strategy for the content improves the response times, performance, and scalability of portal systems. However, very often we need to access remote Web services. One solution is to use a local portlet to access a remote Web service via its interface, obtain the required results as a raw data stream, and locally render the results in a fragment. This approach is relevant for data-oriented Web services. An alternative solution is to equip the Web service with an additional interface in the form of a portlet. When the Web service is called, it returns the entire portlet instead of raw data. This approach is suitable for presentation-oriented Web services.

The second solution appears to be more convenient. The remote Web Services appear as remote portlets, including presentation and application logic. Remote portlets produce portlet markup, which is sent to the requesting (local) portal server and is accessed by the enterprise portal through portlet proxies. Instead of just providing raw data or single business functions that still require special rendering on the portal side, remote portlets are visual representations of (remote) Web services. They behave like Web applications that can be invoked through a standard interface using generic portlet proxies on the local portal side (Figure 92).

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 92. Concept of remote portlets and Web service integration*




The vision for the portals of the future is that they must be able to run local portlets, include remote portlets, and share local portlets with other portals by making them available as remote portlets for Web services (the concept is becoming known as Web Services for Remote Portlets, or WSRP).

Jetspeed, WebSphere Portal, and the Sun ONE Portal allow enterprises to plug the portal server into a UDDI registry and make portlets available as presentation oriented Web services. Portals are frequently deployed as a means to make enterprise applications more accessible to end users. Therefore, publishing an application as a Web service further expands the boundaries of application usage.

Portal clients access portals via the HTTP protocol, either directly or through appropriate proxies or gateways like WAP or voice gateways. To accommodate different devices, portals need to support different markup 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** run on the portal server itself. They are deployed by installing *portlet archive* files on portal servers, and they 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 so that they are 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

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 93. Components for access to remote Web Services*

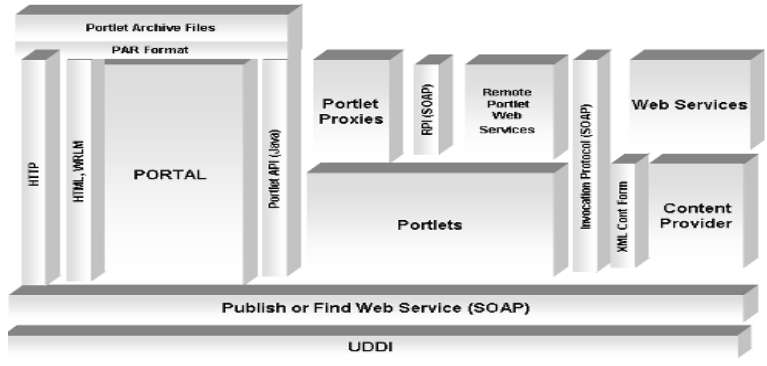



*Figure 94. Obtaining URL as InputStream*

```
protected InputStream getHTTPInputStream(String urlS,
        PortletRequest request, PortletResponse response)
throws MalformedURLException, IOException {….}
```

invoke portlets located on remote servers through a set of Remote Portlet Invocation APIs, where the protocol is based on SOAP. This approach requires employing WSRP. IBM's WebSphere Portal Server provides less complicated methods for accessing remote content—the API ContentAccessService().

**Access to remote systems APIs.** Portlet API ContentAccessService() provides means to access remote system content using (remote) URLs, including URLs located on the other side of a proxy server. This service should be used if you are not certain whether there is a firewall present or if you do not want to use the WSRP framework. The API uses portlet context to obtain the service and locates the URL as a String. This API can also open an SSL connection to remote applications.

In principle, you need to setup an HTTP input stream to a remote system (Figure 94) to read the portlet log file. The method receives the URL to the backend system as a String (urlS). The URL is then checked to determine if the system is local (using isHTTPLocalhostURL()_). If the URL is local

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

(Figure 95), the content access service is not needed. If the URL is remote, the getURL() method (Figure 96) is used to create a java.net.URL object and return an InputStream used to read from the remote connection.

**Using WSRP and UDDI:** Figure 97 shows an example of how a portal finds and integrates a remote portlet published in UDDI. Content or application

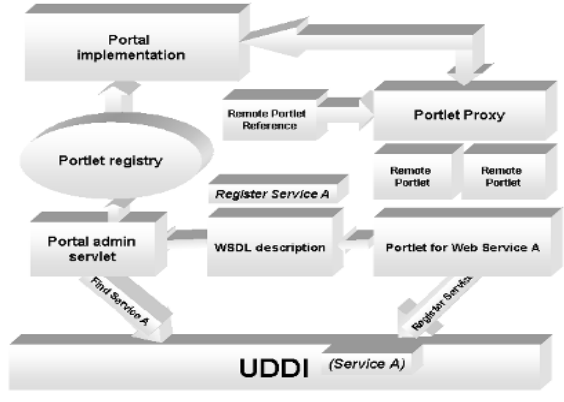
*Figure 95. URL is local URL*

```
if(isHttpLocalhostURL(urlS)) { //local url
if(getPortletLog().isInfoEnabled()) {
getPortletLog().info("getHTTPInputStream: local URL");
}
return(new URL(urlS)).openStream();
}
```

*Figure 96. URL is remote*

```
try {
PortletContext context = getPortletConfig().getContext();
ContentAccessService contentAccessservice =
(ContentAccessService)context.getService(org.apache.jetspeed.portlet.service.ContentAccessService.class);
contentURL = contentAccessservice.getURL( urlS, request, response );
if(getPortletLog().isInfoEnabled()) {
getPortletLog().info("getHTTPInputStream: final URL" + contentURL);
}
return contentURL.openStream();
} catch(PortletServiceException e) {// handle the exception}
}
```

*Figure 97. Locating remote portlets and publishing Web services with UDDI*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

providers implement their service as a WSRP and publish it in a globally accessible directory (such as a UDDI). The directory lets portal providers easily find the desired service. Directory entries, published in WSDL format, briefly describe the WSRP component and offer details about the services. The portal administrator uses the portal's published functions to create remote portlet Web service entries in the portal local registry. Furthermore, the portlet proxy binds to the WSRP component through SOAP handler, and the remote portlet invocation (RPI) protocol ensures the proper interaction between both parties.

## Cooperative Portlets

We often require the portlets on a page to be able to share information and interact with each other by exchanging some information. A portlet can use a server's messaging capability for this purpose provided the portlets reside within the same portlet application. However, this may be too restrictive. The term cooperative portlets is a recent improvement coming from IBM's WebSphere Portal Server 5. It refers to the capability of portlets on a page to share information with each other, without belonging to the same application.

With cooperative portlets, it is possible to coordinate the behavior of multiple portlets. This capability has been introduced for the first time in WebSphere Portal V4 (called Click-to-Action technology) and further extended in WebSphere Portal 5.x. It uses the concept of properties. The data item to be transferred is marked as a property, and its value is then a property value. The property broker mechanism transfers properties between applications. The property broker creates a persistent-like connection between two or more cooperating portlets; this connection is called a wire. Property transfers occur as a side effect of user interactions with a source portlet. This means that users do not have to explicitly select an action on the target from a menu. The user's perception of this behavior is that an action in one portlet "informs" all matching actions in cooperative portlets so that the particular values do not have to be re-entered into target portlets. The transfer is initially triggered from a pop up menu.

The implementation framework for cooperative portlets uses a model called "declare, publish, and share" for information exchange. This model is similar to a callback mechanism. Portlets subscribe to the property broker by publishing properties (typed data items). The same mechanism is used for both a data provider and a data recipient.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The following are the steps needed to implement cooperation:

- One portlet (source portlet) marks something as a *property*, and it publishes it. Published properties are called output properties.
- The receiving portlet is called a target portlet. The target portlet receives the properties as input properties.
- The target portlets can optionally provide actions to process their input properties.

There are several sources of actions. Portlets can receive actions from their own processing; actions can be initiated by the transfer of property values; or a target portlet can receive property changes directly through the PropertyListener interface. Each action is associated with a single input parameter and zero or more output parameters, which provide information to the action about the objects in which the property value should be bound, such as the request or the session. Each parameter is associated with exactly one property.

The mechanism relies on a runtime property broker that matches the data type of output properties of the source portlet with the data type of input properties of one or more target portlets. A successful match allows the portlets to share the property. Several methods are used to initiate the property transfer:

- A user launches a Click-on-Action event from an icon representing a pop-up menu on the source portlet. The pop-up menu contains the list of targets for the action. The user selects a specific target, and the property broker delivers the data to the target. The user can also broadcast the property to all portlets on the page that have declared an action that is associated with a particular, matching input property.
- A wire can be used to invoke persistent selection between two portlets (by holding the Ctrl key while clicking an action, the selection is saved persistently as a connection between two portlets). When the user clicks on the icon again, no selection menu is shown, but the wired action(s) is automatically fired.
- The source portlet can do a programmatic publishing of properties to the broker when it determines that property values have changed. However, the property values are transferred to the target(s) only if wires have been created.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## How the Property Broker Works

The property broker runtime is comprised of two components: a generic *wrapper* portlet used to wrap each portlet enabled for cooperation, and a *broker* component. These two components interact with the WebSphere Portal Server. The wrapper is packaged in the portlet's archive war file. The main purpose of the wrapper portlet is to:

- Intercept calls intended for the application portlet,
- Register actions supported by the portlet with the broker, and
- Initiate data transfer from source to target portlets using the property broker. The broker performs multiple functions:
  - Maintains a repository of actions on each portlet,
  - Maintains the parameter type information,
  - Performs source and target matches at runtime using this information, and
  - Generates additional markup to allow users to trigger data transfers across portlets.

There is a choice of implementation approaches for registering the properties with the broker: *declarative, programmatic*, or a *combination* of these approaches.

**Declarative approach.** Programmers can use a custom JSP tag library to flag sharable data on their output pages. Source portlets can publish their output properties using tags from the JSP tag library. These tags are available in the c2a.tld tag library (Figure 98).

*Figure 98. JSP example using c2a.tld library*

```
<@ taglib uri="/WEB-INF/tld/c2a.tld" prefix="c2a" %>
For other portlets, the portlet tag library and the <portletAPI:init/> tag must also be
included in the head of the JSP. For example:
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init/>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

In order to declare output properties in the source portlet, we need to include the following tags:

- <c2a:encodeProperty/> tag used by the source to insert data and type information in markup that will display the Click-to-Action icon and generate a pop-up menu, and
- <c2a:encodeProperties/> tag, which is used to enclose normal HTML markup and one or more encodeProperty tags with the markup. This tag is provided to support the so-called scatter scenario, where a user can optionally send more than one unit of data to target portlets.

Usually, any property that may be described using an XML type, following the type restrictions, is a candidate for being tagged as a C2A source. If you want to make sure that the Click-on-Action is generated, try to tag as many sources in the JSPs that qualify. Note that the Click-on-Action icon is not generated for sources that do not have any matching actions on the page.

**Programmatic approach.** The property broker provides APIs to give developers control over how portlets handle input and output properties. In some specific situations, the programmatic approach could be preferred over the declarative approach. The following mechanism is used:

- You need to activate/deactivate actions for a session.
- You need to change portlet state, but immediate portlet response is not required.
- You want to publish output properties using the changedProperties() method.
- If the action property is not known at development time (the portlet is generated by a builder application), you can register actions programmatically instead of declaring them in a WSDL file.
- You want to generate markup directly in the portlet rather than using JSPs.

The target portlets can use the setProperties() method to receive change notifications instead of actionsPerformed(), which is typically used with the ActionListener interface.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

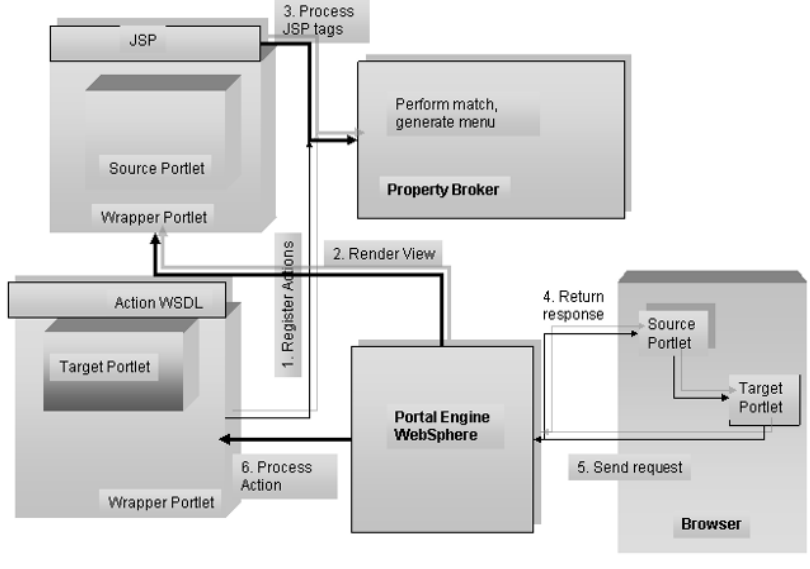
**Property transfer using wires.** The user activates the transfer by clicking on the Click-on-Action icon, and the wired action is automatically fired.

The wired action can be associated with output properties. In this case, the property broker checks for output property values after the action has finished executing. If some properties exist, the property broker examines wires with source ends that correspond to the output properties of the action. If appropriate wires are found, two situations may occur depending on if the target end is an input property or input parameter, as follows:

- The target end is an input parameter: The actions on the target are invoked, and the property value is delivered using the input parameter information.
- The target end is an input property: The property value will be delivered to the portlet's setProperties() method using the PropertyListener interface.

The example in Figure 99 shows some events that may occur in the Click-on-Action scenario. The event flow shown is for the declarative approach. Note that the events are similar for the two other approaches (wired or programmatic approach).

*Figure 99. Request-response flow for cooperative portlets*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- When the target portlet is initialized, the wrapper portlet processes the WSDL file (WSDL action) and registers the actions and properties with the Property Broker.
- During the rendering phase (doView()), JSP Click-on-Action tags in the source portlet tags are processed, and the Property Broker is called to determine matching actions on the page.
- When the rendering phase finishes, the portal assembles the response page and returns it to the client's browser.
- The cooperative action on the source portlet is now ready for invocation.

The property broker is a WebSphere Multiplatform component. It seems to provide an easier and more structured way of implementing inter-portlet and page-to-page conversation.

## Page-to-Page Communication Scenarios

Using the Click-on-Action and the property broker delivery method, users can transfer data with a simple click from a source portlet to one or more target portlets, causing the target to react to the action and display a new view with the results. The property broker in WebSphere Portal 5 also enables passing complex data types between cooperative objects and page-to-page communication. Portlets on a page can cooperate in this way regardless of how they were developed. They do not have to be aware of the existence of the other cooperative portlets.

*Page-to-page communication between portlets* in WebSphere Portal 5 can be implemented using URL parameters, as we have seen previously, or by using cookies. We do not believe that is a good design to use lots of page-to-page communication. However, page-to-page communication is suitable for portlets that are on many pages and do not have much user input data.

The question then is when is it efficient to use this functionality. Often we need to implement coordinated and consistent behavior in situations when one or more portlets on a portal page should automatically react to changes from a source portlet that have been triggered by an action or event in the source portlet. Cooperation and the ability to listen to another portlet's events provide a coordinated and consistent behavior between the portlets on the page.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Typically, there are two scenarios that may require page-to-page communication:

1. **Static information transfer (static link with pre-defined data):** A simple link from a portlet on a source page carries static information to a portlet on a target page, and the portlet on the target page uses the data to update the presentation.
2. **Linking with user input data:** A user enters information on a portlet on a source page and then clicks a link or button. That link carries the user data to a portlet on a target page, which can then use the data, for example, to update some information.

Let us assume we are building a portal for a University Web site, and we wish to include a portlet called ShortCourseList that displays a list of short courses on most of the pages, particularly on the welcome page. The portlet ShortCourseList can call the Web service to obtain current data, or it can use JavaBean and JDBC to retrieve the list of short courses from the database.

Let us discuss the first implementation option — static information transfer. We start with implementing JSP for displaying the list of short courses — ShortCourse.jsp. Furthermore, we need to make some changes to the tag

*Figure 100. Include tag wps to JSP ShortCourseList portlet*

```
<%@ page session="false" contentType="text/html"
import="java.util.*, comunication.portlet.shortcourses.*, shortCourse.util.*"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<%@ taglib uri="/WEB-INF/tld/engine.tld" prefix="wps" %>
<portletAPI:init/>
```

*Figure 101. Pointer to the page name (ShortCourse)*

```
<wps:urlGeneration contentNode="shortcourse.ShortCourse"
portletWindowState="Normal">
<wps:urlParam name="courseId" value="<%= course[i].courseId()%>" />
<a href="<% wpsURL.write(out); %>">
<%= course [i].getName() %>
</a>
</wps:urlGeneration>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

library (in IBM's WebSphere 5, the library is called engine.tld, and it typically holds the skins and themes), and we need to include the tag for processing the ShortCourseList portlet (Figure 100).

We need to define the link to the portlet ShortCourseList on ShortCourse.jsp. This is done by using the wps:urlGeneration tag, which points to the page responsible for displaying the list of short courses.

Finally, we need to enable the receiving portlet ShortCourseList.java to get the courseID and to process the doView method (Figure 102).

We will now look at the second implementation option — linking with user input data. In this scenario, a user wants to search for the available courses on the Monash University site using the search_portlet. The user can enter the text (full text or keywords), and the search_portlet returns the results. The text entered by the user must be passed to the search_portlet. It can be done, for example, using a simple JavaScript and cookie. The main change is done in the definition

*Figure 102. Corresponding processing in the portlet ShortCourseList*

```
public void doView(PortletRequest request, PortletResponse response)
   throws PortletException, IOException
{
// Get the id of the course sent from the ShortCourseList portlet
String courseId = request.getParameter("courseId");
// Make a view mode bean
ShortCoursePortletViewBean viewBean = new ShortCoursePortletViewBean (courseId);
request.setAttribute(VIEW_BEAN, viewBean);
// Invoke the JSP to render
getPortletConfig().getContext().include(VIEW_JSP, request, response);
}
```

*Figure 103. Change in form definition — cookie is used to transfer text*

```
<form name="<%= portletResponse.encodeNamespace("SearchForm") %>">
Enter search text:<br/>
<INPUT class="wpsEditField"
 name="<%= portletResponse.encodeNamespace("searchText") %>"
 type="text"
 onChange="<%=portletResponse.encodeNamespace("Search_Set_Cookie")%>
      ('SearchTextCookie', this.value)"/>
<wps:urlGeneration contentNode=" shortcourse.SearchResult" portletWindowState="Normal">
          <a href="<% wpsURL.write(out); %>">
                  Search
          </a>
</wps:urlGeneration></form>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 104. Cookie processes the request before doView renders the page*

```
public void doView(PortletRequest request, PortletResponse response)
 throws PortletException, IOException
{
     SearchPortletViewBean viewBean = new SearchPortletViewBean();
          Cookie[] cookies = request.getCookies();
          StringBuffer buffer = new StringBuffer();
          if ( cookies != null )
     {
                    for (int i = 0; i < cookies.length;  i++ )
     {
                              Cookie cookie = cookies[i];
                              if ( cookie.getName().startsWith(COOKIE_NAME))
     {
                                        viewBean.setText(cookie.getValue());
                              }
                    }
          }
          request.setAttribute(VIEW_BEAN, viewBean);
     // JSP is invoked to render
          getPortletConfig().getContext().include(VIEW_JSP, request, response);
}
```

of the form. Our form now responds to the text received from the cookie (Figure 103). The SearchPortlet now retrieves the user input text from the cookie and renders the View.

Our experience with page-to-page communication between portlets is limited to WebSphere Portal for Multiplatform 5. It can be implemented using URL parameters (static binding) or cookies (dynamic linking). Common sense dictates that use of URL parameters is suitable for those applications that do not require user input data. The implementation with cookies is recommended if the portlets need to transfer user input data. We believe that page-to-page communication is suitable for portlets that are on many pages and do not have much user input data. It is not a good design to use lots of page-to-page communication in an application.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter XI

# Portlet Authentication and SSO

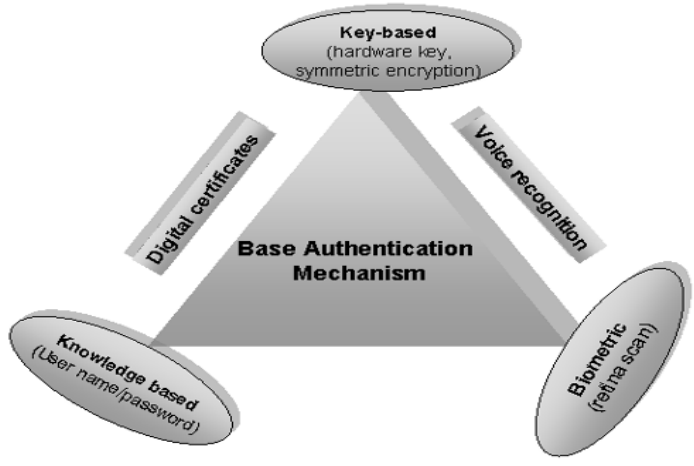
In this section, we will introduce a few important details relevant to the authentication process and Single Sign On (SSO) implementation. The SSO is a popular feature that was established in the portal standard JSP 168 as a mandatory feature. This functionality enables the user to authenticate only once to all Web services and back-end applications to which he or she requires access. The authentication code is propagated to all services for which the user has access permissions. This means that users can log on only once and be known to the various parts of the portal server with the same user credentials.

## Authentication Process

Authentication can be loosely defined as the process of establishing a user's identity and verifying credentials. In a portal environment, authentication may be achieved by providing a unique user name and password pair. The user is assigned a certificate that is validated by using a third-party authentication server, or a smart card or biometric device (Figure 105). The final solution may involve all three methods, and it will require a repository integrated with the system and holding these identities. For example, the WebSphere portal server

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 105. Example of use digital certificates and biometric devices*




uses the authentication services provided by the WebSphere Application Server and CORBA credentials together with an encrypted Lightweight Third Party Authentication (LTPA) cookie to authenticate users. Alternatively, it is possible to authenticate via an LDAP directory, which may hold some application-oriented attributes in addition to authentication information, that is, a hierarchical reporting structure.

The actual situation with portlets is more complicated. The portal provides an integrated view of multiple back-end applications, and, sometimes, access to Web services. Web services typically access additional data resources on behalf of the portal user, which may in turn retrieve further data from other data sources on behalf of this user. In this case, the user's credentials are presented to the Web services by the portlet of the portal page into which the user is logged on.

The area of security is widely covered in books. The Secure Socket Layer (SSL) and Transport Layer Security (TLS) were introduced by Netscape in 1994 as a response to the growing concern over security on the Internet. XML Encryption and XML Signature are W3C standards, which can be used to encrypt elements of an XML document and prove message integrity. Any part of an XML document can be digitally signed, and it becomes self-validating when the document recipient has the signer's public key. An OASIS standard, the Security Assertion Markup Language (SAML) is used to pass authentica-

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 106. Dependencies in security in portal centric design*

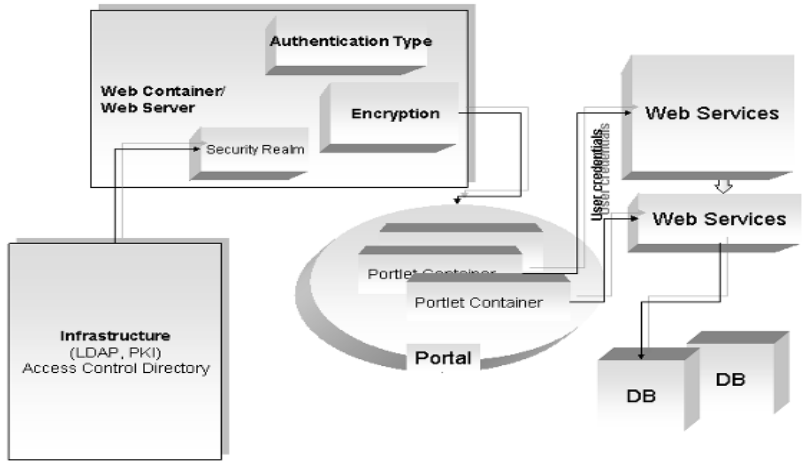
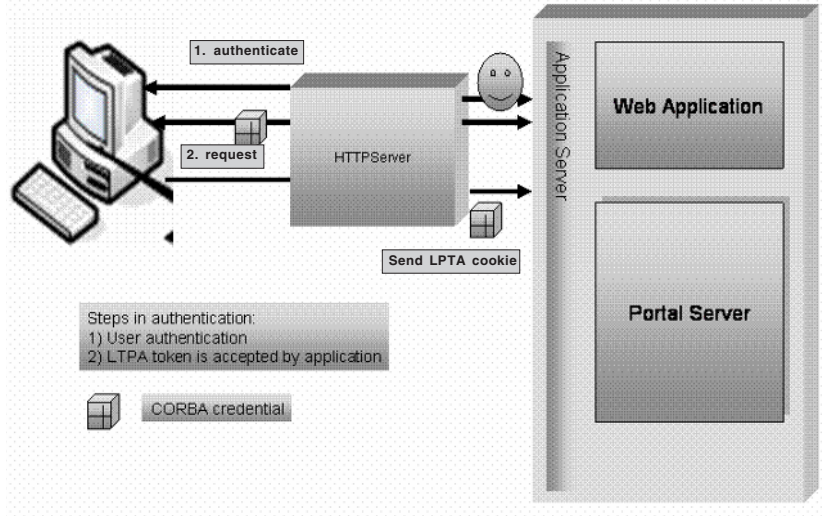



*Figure 107. Example of the authentication process in WebSphere*




tion and authorization information between parties. These technologies are used for SSO in the portal environment.

Web Services Security (WSS) is the standards process managed by the OASIS Web Services Technical Committee. It mostly relies on accepted standards of SOAP, XML Encryption, and XML Signature. The specification

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

— WS-Security — has been released in 2002 by Microsoft, IBM, and Verisign.

We will now look at the security solution for an enterprise system with portals. Security is the heart of this system. The main engine of this system is a Web Server (Web Container). The Apache Tomcat Web Container uses the security of the Servlet 2.4 specification, called a security realm (for additional information, visit http://jakarta.apache.org.tomcat/). Container-managed security is an important concept for portals and should be fully understood by portal developers1. Implementation details are well described in Richardson, Vitale, Len, and Smith (2004).

A portal server typically uses form-based authentication (Figure 107). The user is prompted on an HTML form to enter his/ her identification and password. The portal server requests the application server (HTTP Server) to validate the authentication information against an LDAP registry. The WebSphere Application Server uses LTPA and cookies for the authentication. A CORBA Credential is used to represent authenticated users and their group memberships. Providing the user identification is valid, an appropriate CORBA credential is created, and an LTPA cookie is stored on the user's machine. When the user tries to access a resource or protected information, the application server intercepts the request and redirects the request to the login form. This form posts the user ID and password to the portal, which requests that the application server authenticates the user.

This is not the only option for user authentication. Many third-party authentication servers can be used. These servers must establish a trusted relationship between their proxy and the Application Server. WebSphere uses the Trust Association Interceptor (TAI) module, which converts security information specific to the authentication proxy into a format the application server understands. The scope of the authentication mechanism depends on the capabilities of the third-party product.

Many portlets need to access remote applications or remote Web services. The IBM WebSphere Portal provides a specific service to authenticate remote users — the Credential Vault Portlet Service. The Credential Vault is partitioned into segments. A vault segment contains one or more credential slots. There are two different types of vault segments:

- **Administrator-managed segments:** In this type of vault segment, the creation of new slots is restricted to the portlet administrator.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- **User-managed segments:** In this type of vault segment, portlets can also create new slots on behalf of the user.

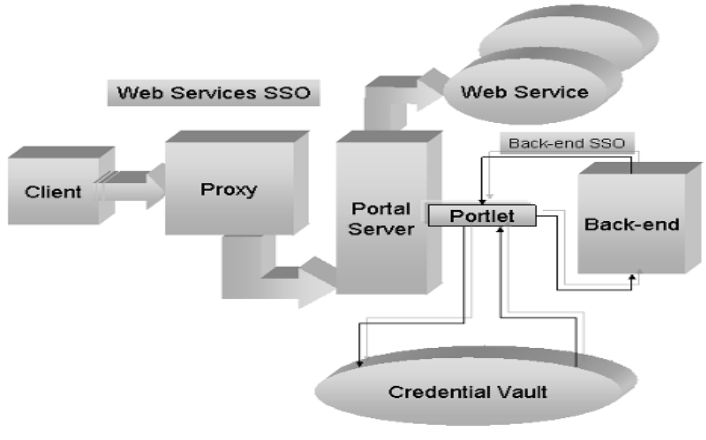
The Credential Vault Portlet Service returns credentials in the form of credential objects — either passive or active. Passive credential objects are containers for the credential's secret. Portlets can use passive credentials to extract the secret and perform the entire authentication on behalf of the user with the back-end applications. Active credential objects do not reveal the secret. On the contrary, they provide business methods (for example, Credential login()) to get connection to other remote servers, and then they authenticate the user there.

Note that setting and retrieving credentials can be performed by portlets for both types of vault segments.

## Single Sign On (SSO)

The majority of portal servers that support single sign on realms are using Web Servers as well as authentication proxies. With the IBM WebSphere Application Server, LTPA tokens provide single sign on (Figure 108). The following mechanism is applied: when a user is authenticated, the portal server creates an LTPA single sign on cookie containing the authenticated user credentials. This encrypted cookie conforms to the format used by WebSphere Application

*Figure 108. Single sign on implementation in WebSphere*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Server and can be decrypted by all application servers in the shared domain, provided they all have the cipher key. This way, all servers in the cluster can access credentials of the user without additional requests for authentication. However, the user's browser must support cookies and have its support for session cookies enabled.

Many portlets require access to remote Web services that could be outside of the portal space. As mentioned earlier in this section, a portal server provides a credential vault service that portlets can use to store a user ID and password (or other credentials) for a user login to an application (Figure 108). The user perceives this mechanism as the ability to sign on to all applications only once, through the portal sign on. The user may not see applications that he or she is not authorized to see. Another effect, of course, is that the change of the password at the portal level will apply to all applications.

## Endnote

1 Each Portal development site should prepare and use a "10 security commandments" document defining governing principles implemented in portlets.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter XII

# Struts Framework

When Java servlets were first invented, many developers quickly realized that they were faster and more powerful than the standard CGI. They were also portable and extensible.

But writing HTML to send to the browser in an endless stream of println() statements was not practical. The next framework — Java Server Pages (JSP) — turned servlet writing inside out. JSP allows developers to mix HTML markup with Java code, and to therefore enjoy all the advantages of servlets along with easy development with HTML.

The next stage was the realization that Java Server Pages and servlets could be used in mutual cooperation in Web applications. The servlets could serve as controllers and drive control flow, and the JSPs could concentrate on producing HTML pages. This actually was the rediscovery of the classic MVC pattern that was first abstracted in the famous Smalltalk MVC framework (Burbeck, 1987).

## Struts in Web Service Endpoint

The Struts architecture can be fully described by the MVC2 paradigm. The Struts project was started in 2000 by Craig R. McClanahan to provide a standard MVC framework to the Java community. Struts provides its own

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

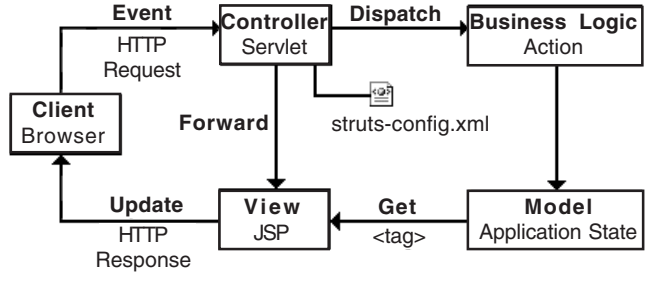
Controller component, which integrates with other technologies to implement the Model, View, Controller pattern. Struts can be seen as a set of cooperating classes, servlets, and JSP tags that make up a reusable MVC2 design. This definition implies that Struts is a framework, rather than a library, but Struts also contains an extensive tag library and utility classes that work independently of the framework. Figure 109 shows an overview of the Struts Framework.

The problem with the classic MVC model and a Web-based client is the stateless connection between the client and server. The stateless behavior makes it difficult for the model to notify the View of changes. The browser has to send an additional query to the server to retrieve modification of the application state. The modified version of MVC (called MVC 2) is in Figure 109.

The Struts framework is a component-based framework that is heavily dependent on a large set of libraries:

- *Client browser* role: An HTTP request from the client browser creates an event. The Web container will respond with an HTTP response.
- *Controller* role: The servlet receives the request from the browser and makes a decision regarding where to send the request. With Struts, the Controller is a command design pattern implemented as a servlet. The struts-config.xml file configures the *Controller* for the application.
- The *business logic* updates the state of the model and helps control the flow of the application. With Struts, this is done with an Action class as a thin wrapper to the actual business logic.
- The *model* provides the actual representation of the information. It also stores the state of the application. The business objects update the

*Figure 109. MVC 2 Model — Struts Framework*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

application state. ActionForm bean within the model represents the application state at a session or request level, not at a persistent level. The JSP file reads information from the ActionForm bean using the JSP tags.

- The *view* is simply a JSP file. There is no flow logic, no business logic, and no model information — just the tags.

The Struts framework includes a set of additional custom tag libraries that allow creation of user interfaces that are fully internationalized and interact with ActionForm beans. ActionForm captures and validates the input for the application. The Action class should control the flow and not the logic of the application. By placing the business logic in a separate package or EJB, we allow flexibility and reuse.

The Controller part receives requests from the client, decides what function is to be performed next, and then delegates the production of the next View to an appropriate View component. The Controller is represented by a servlet derived from the class ActionServlet. This servlet is configured by defining a set of ActionMappings. An ActionMapping defines several attributes (properties), such as a path that is matched against the request URI of the incoming request and usually specifies the fully qualified class name of an Action class subclassed from org.apache.struts.action.Action. Action(s) encapsulate calls to business logic classes (JavaBeans), interpret the outcome, and dispatch control to the appropriate View component that creates the response.

The *Control* layer is composed of several components:

- A controller servlet
- Developer-defined request handlers
- Several supporting objects (Java classes)

The ***View*** layer relies on custom tag libraries. Some of the tags allow the access of control-layer objects; others are generic tags. The components of the ***Model*** are often project-specific.

The Struts framework was designed with the view of easing access to the business end of applications. If you are working on large-scale Web service applications, you would be aware of the difficulties introduced by user change requests and associated development costs. MVC2 decouples the presentation from the business logic and data. Its reliance on programming models used

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

by other products in the development process, such as JDBC, J2EE components, and libraries, makes it an inflexible tool from the perspective of components and environment upgrade.

## Struts in Portal Development

There are two main differences between the portlet and servlet environments affecting the application of Struts in WebSphere Portal — Action Processing and URI construction.

Action processing and rendering in Portlets: As previously discussed, portlet processing is implemented in two phases: an action phase and a rendering phase. Action processing is performed prior to rendering the display view. Only the request object is passed to the portlet during the action phase. The request parameters that are normally extracted from the servlet's request object are no longer available during the rendering phase of the portlet. Since rendering methods, such as doView(), can be called when the portlet page is refreshed without a new event occurring for that portlet, all information required to render the page must be available every time that method is called.

The Struts Portlet framework provides a RequestProcessor that creates an IViewCommand. The IViewCommand encapsulates the information so that the command object can be rendered at a later time, during the rendering method.

**URI construction.** The portlet creates the URI using the PortletResponse.createURI() method. The Struts Portlet framework uses modified Struts tags so that it can create portal links. Struts application paths (for Action) are retrieved or sent using the portlet URI. The portlet URI has a specific format, and a special API is used to generate the portlet URI, including the accompanying information. The Struts link tags have been modified to reflect the requirement. The Struts link tags behave the same as they do in the servlet environment, but the URL is a portlet URL, and the Struts URL is passed as a parameter on the URL. This is complicated, and it means that you cannot view portlet development with the Struts framework as easily.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter XIII

# Writing Portlets:
## A Quick Lesson for IT Managers

This section is intended for IT managers who need to understand the basics of developing portlets. We will take you briefly through the development cycle of a portlet: from writing the code, to compiling, up to deployment. You must start with environment set up and continue with writing, compiling, and testing the portlets.

The code example in Figure 110 shows a simple portlet called HelloMonashWorld class. This class inherits from the class PortletAdapter and processes the following methods:

- init(portletConfig) initializes the portlet and obtains its configuration, which is passed to the method in portletConfig object.
- doView(PortletRequest portletRequest, PortletResponse portletResponse) works in a similar fashion as the servlet's doGet() method; it receives a user request and creates a new object in response.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 110. Simple portlet code example*

```
package jana.wps.samples; //the directory with portlet samples
//import apache library

import org.apache.jetspeed.portlet.*;
//import all io routines
import java.io.*;

public class HelloMonashWorldPortlet extends PortletAdapter {

public void init(PortletConfig portletConfig) throws UnavailableException
  {
    super.init(portletConfig );
  }

  public void doView( PortletRequest portletRequest, PortletResponse portletResponse )
          throws PortletException, IOException
  {
    PrintWriter writer = portletResponse.getWriter();
    writer.println("<p class="wpsPortletText">Hello Portal Monash World!</p>");
  }

}
```

The implementation of the initialization phase uses a portlet configuration object to load the initial configuration parameters (PortletConfig portletConfig).

You can see from the code that the portlet is using similar mechanics for writing output as those found in a servlet; in the doView() helper method, the portlet obtains the PrintWriter object and attaches this object to the response object:

```
PrintWriter writer = portletResponse.getWriter();
```

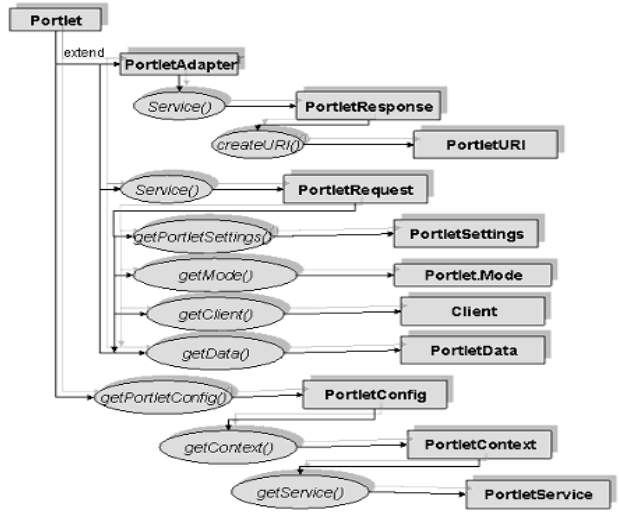
Since we have used the PrintWriter object, we also have thrown the appropriate exception: IOException. When the rendering phase starts (page aggregation), the output stream created earlier is written to the servlet's output.

Figure 111 shows the structure of portlet configuration APIs. The portal calls the service() method when the portlet is required to render its content. To enable portlets to use pluggable services via dynamic discovery, the Portlet API provides the PortletService interface. A PortletService is accessed from the PortletContext.getService() method, which creates the service and returns it to the portlet. A portlet service can be invoked only from within a portlet.

To compile portlet Java source code, the environment variables must be set up (namely CLASSPATH) so the java compiler can find all the *.jar files that are

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 111. Overview of portlet APIs*




required by the application. Since the portal engine also requires support of the Web Services and the Web server, the following jar files also must be in the CLASSPATH variable:

portlet-api.jar
wpsportlets.jar
wps.jar
j2ee.jar
dynacache.jar (IBM WebSphere Portal)

To compile the portlet, a fully qualified path to the application must be provided. The command line to invoke compile looks like this:

```
javac –classpath %CLASSPATH% jana/wps/samples/
HelloMonashWorldPortlet.java
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

All class files must be packaged in the application jar file (Java archive file). Following is the command line that is required to invoke the jar utility and create the application jar file:

```
jar –cf HelloMonashWorld.jar HelloMonashWorldPortlet.class
```

Each application contains two deployment descriptors: the Web application deployment descriptor (web.xml) and the portlet deployment descriptor (portlet.xml). The web.xml file defines the Web application deployment structure. For details on all of the elements of the web.xml deployment descriptor, refer either to http://java.sun.com/j2ee/tutorial or the section on Web Services in this book.

The web.xml structure will look similar to the example in Figure 112.

The portlet.xml elements are defined by the portlet_1.1.dtd. If the installed portal is a WebSphere portal server, this file can be found in the WebSphere\PortalServer\app\wps.ear\wps.war\dtd directory. In the declaration, the following elements must be provided (Figure 113).

The description and display name are meant to ease understanding, while the local name and the class provide the means for loading and referencing the

*Figure 112. web.xml structure*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/Web-app_2_3.dtd">
<web-app id="WebApp">
    <display-name>HelloMonashUser</display-name>
   <servlet id=" jana.wps.samples.HelloMonashWorld-1">
<servlet-name>HelloMonashUser</servlet-name>
        <servlet-class>HelloMonashUser</servlet-class>
   </servlet>
<servlet-mapping id="must-be-unique-ID-recommended-to-include-the–servlet-namespace">
    <servlet-name>HelloMonashUser</servlet-name>
    <url-pattern>/HelloMonashUser/*</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 113. portlet.xml — portlet identification elements*

```
<description>Example of simple portlet</description>
<portlet-name>HelloMonashWorldPortlet portlet</portlet-name>
<display-name>Monash Portlet</display-name>
<portlet-class>jana.wps.samples.helloMonashWorldPortlet</portlet-
class>
```

portlet. Furthermore, we also should mention the welcome-files in the welcome-file-list tag.

The portlet.xml descriptor requires a few more elements (Figure 114). To improve the performance, portlets that are in use are kept in a cache. The expiration-cache tag represents the number of seconds a portlet is kept in a cache. If the value = -1 it means that the portlet will always be kept in that cache. Alternatively, value = 0 means that it would never be cached. The *shared* tag indicates that the content is shared among all users or that the cache should be maintained for each user. The value = *NO* may result in creating a large cache.

The tag <portlet-app> is used to define the abstract portlet application. This abstract application points to the concrete portlet applications. Only one

*Figure 114. portlet.xml additional elements*

```
<cache>
        <expires>0</expires>
    <shared>NO</shared>
</cache>

<!--The next section declares the supported portlet modes and mime types   -->

<allows>
    <maximized/>
    <minimized/>
</allows>
<supports>
<markup name="html">
                <view output="fragment"/>
                <configure/>
                <edit/>
                <help output="document"/>
        </markup>
        <markup-name="wml">
                </view>
        </markup>
</supports>

<portlet-app uid="jana.wps.samples.HelloWorld-1" major-version="1" minor-version="0">
<portlet-app-name>HelloMonashWorld application</portlet-app-name>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 115. portlet.xml file — portlet identity*

```
<portlet id="jana.wps.samples.HelloMonashWorld" href="WEB-
INF/web.xml#jana.wps.samples.HelloMonashWorld-1" major-version="1" minor-version="0">
```

*Figure 116. portlet.xmxl — concrete portlet application*

```
<concrete-portlet-app uid="jana.wps.samples.HelloMonashWorld-1">
        <portlet-app-name>HelloMonashWorld application
    </portlet-app-name>
        ……
</concrete-portlet-app>
```

portlet-app tag is allowed per portlet.xml, and only one portlet.xml is allowed per (war) file. Therefore, each war file may only deploy a single abstract portlet application. The uid uniquely identifies the abstract application in the portal server and must be unique throughout the entire portal environment.

One or more portlet tags are required. These tags define the abstract portlets contained in the abstract portlet application. Each portlet tag maps to a single servlet defined in the web.xml. It is important to understand that there is a one-to-one relationship between the servlets defined in the web.xml and the abstract portlets defined in the portlet.xml. But two abstract portlets can be mapped to the same servlet.

The portlet id must be unique only within the abstract portlet application (Figure 115). It is used by the concrete portlets to create a link to the abstract definition. The tag *href* creates the link between the abstract portlet and the servlet defined in the web.xml.

The tag *concrete-portlet-app* (Figure 116) defines the concrete portlet application deployed in the portal server. This concrete application is instantiated from the abstract portlet application defined earlier in the portlet.xml. A concrete portlet application does not have to contain all portlets defined in the abstract application. However, it must not define more portlets than the abstract application. Each concrete portlet contained in the concrete application maps to one abstract portlet. Mapping of an abstract portlet twice in the same concrete application is not permitted.

The uid must be unique throughout the entire portal environment. The uid is being used by the portal server to update or delete the portlet. If the uid changes

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 117. portlet.xmx — concrete portlet*

```
<concrete-portlet href="#jana.wps.samples.HelloMonashWorld">
  <portlet-name>HelloMonashWorld portlet</portlet-name>
        <default-locale>en</default-locale>
        <language locale="en">
              <title>HelloMonashWorld portlet</title>
              <title-short></title-short>
              <description></description>
              <keywords></keywords>
        </language>
</concrete-portlet>
```

*Figure 118. portlet.xml — application definition*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE portlet-app-def PUBLIC "-//IBM//DTD Portlet Application 1.1//EN" "portlet_1.1.dtd">
<portlet-app-def>
all abstract and concrete necessary definitions as discussed above will be here
</portlet-app-def>
```

from the original, the concrete application will not be updated. Instead, a new concrete application is installed. Thus, change in the uid may result in multiple concrete applications.

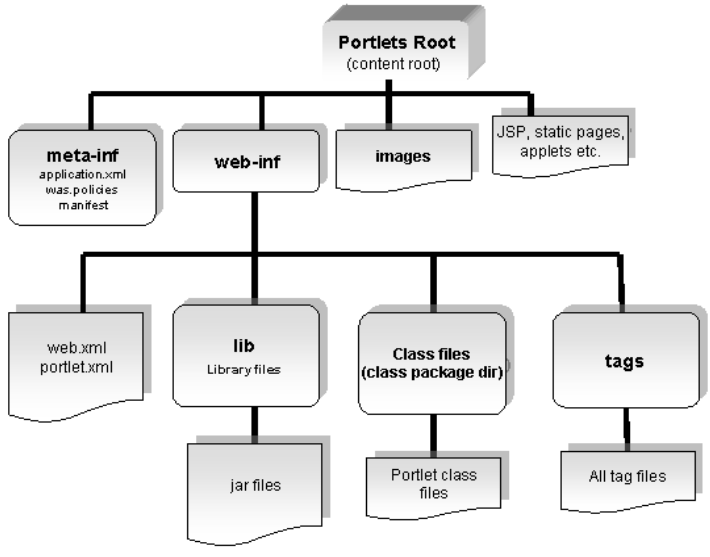
The other important tag is the portlet application's name. Each concrete application is listed in the war file by its name. Although it is permitted to have more than one concrete application of the same name in portlet.xml, deploying more than one concrete portlet application with the same name may result in administrative confusion.

The tag concrete-portlet (Figure 117) encapsulates the definition of the concrete portlet deployed in the concrete application. Any number of concrete portlets may be deployed, but not more than the number of abstract portlets (see portlet tag) defined in the abstract portlet application. The attribute *href* creates a link between the concrete portlet and the abstract portlet.

The portlet.xml definition starts with the DOCTYPE tag (Figure 118), which is the same for each portlet.xml description. This tag defines the dtd that will be used when the xml file is parsed. The tag *portlet-app-def* is the top-level tag, which encapsulates all abstract and concrete portlet application definitions. Only one tag is allowed.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 119. The structure of portlets' deployment war file*




In order to generate the Web application archive file (war file), the files and other resources must be stored in predefined directories starting from the context root. To set up the war file directory, the following structure is recommended (Figure 119).

Several methods exist that pertain to how to generate an archive file (jar, war or ear file). We have already mentioned some in the sections dealing with Web services and servlets (Servlets). Sun Microsystem's J2EE Deployment tool provides an easy way to create archive files. The jar utility is one method used to create a war file, to update or extract portlet descriptor (portlet.xml), and to extract all files[1].

As we mentioned previously, portlet window state can be managed programmatically (window state changes between maximized and minimized). Figure 120 demonstrates how to obtain the window state attribute and how to associate a JSP with each portlet window state that the programmer wants to manipulate.

Generally, with Java J2EE, nearly every resource and object are somehow associated with the notion of context. The PortletContext interface defines a portlet's view of the portlet container within which the portlet is running. PortletContext also allows a portlet to access resources available to it. Using

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 120. Obtaining window state*

```
PortletWindow.State state= portletRequest.getWindow().getWindowState();

//display an image in the max state
if(state == PortletWindow.State.MAXIMIZED){

//include jsp to be displayed in maximized state
getPortletConfig().getContext().include("/WEB-INF/jsp/Maximized.jsp", portletRequest, portletResponse);

   } else{ //otherwise display normal portlet window

    // include JSP to be displayed in normal state of portlet
                getPortletConfig().getContext().include("/WEB-INF/jsp/ViewNormal.jsp", portletRequest,
portletResponse);
```

the context, a portlet can access the portlet log, context parameters common to all portlets within the portlet application. It can also obtain URL references to resources or access Portlet services.

One of the easiest ways to separate markup from main functionality is to use JSP. In contrast to servlets, the concept of portlets is based on HTML *fragments*. JSP should provide only fragments. Since portlets are part of a larger Web page, some tags are not allowed in portlets (For example, <HTML>, <HEAD>, or <BODY> are tags allowed only in the servlet). JSP content example is then restricted to the following example (Figure 121).

A separate JSP could be created to provide the user interface for each portlet mode. As opposed to servlet-based applications, the size of the horizontal and vertical span of JSPs should not squeeze other portlets off the screen. It is advisable to focus on the actual information displayed by the portlet and adjust layout elements accordingly.

*Figure 121. JSP example for a portlet*

```
<%@ page session="false" %>

<!—this directive makes the tag libs available -->
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<center>
    Maximized portlet window is showing now
</center>

<br>
<img src='<%= portletResponse.encodeURL("images/UluruImage.jpg")%>'
width="360" height="283" border="0">
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The look and feel of the portal is determined by portal *themes* and portlet *skins*. Portlets can use styles defined in the portal theme cascading style sheets (css). No decorating elements are required for input elements. A better practice is to use the theme's input class definition:

```
<input class="wpsButtonText" type="submit">
```

JavaScript allows inserting some branching and logic. A better practice is to reduce the dependency on JavaScript. JavaScript implementations and behavior can be different for different browsers. Using a lot of JavaScript may cause the portlets to be dependent on browser type and version.

A well functioning portal keeps track of user activities throughout the pages and portlets. Using a browser's back-button may introduce confusion, and the portal will lose track of the current state. This is because the state of each portlet is maintained, and the back button does not cause the portlet's state to roll back.

## Endnote

1 To create archive files, use the following command: jar cf jar-file. To extract components, use : jar xf jar-file.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter XIV

# Brief Overview Portal Application Platforms

Portal engines come in several implementations. We do not attempt to discuss them all in this book. We provide a high level overview of three architectures: WebSphere Portal for Multiplatforms 5, Apache Open source portal servers (Pluto and Jetspeed), and Sun Microsystems Sun ONE Portal Server.

## WebSphere Portal for Multiplatforms 5

Fourth generation portals such as IBM WebSphere provide a portal framework for enterprise data and Web Services integration. We have used WebSphere Portal for Multiplatforms 5, as discussed already in the previous sections. This portal platform claims to integrate an entire range of enterprise content and applications, and it is mostly based on the J2EE programming model. The architecture supports a scalable framework and allows easy adaptation to the changing requirements of business. This platform also provides several tools for aggregating and personalizing Web-based content. WebSphere Portal for Multiplatform 5 uses the underlying platform provided by the WebSphere Applications Server. Although many libraries provided with a portal server have their historical roots associated with Apache Jetspeed and Pluto projects, WebSphere Portal for Multiplatforms 5 has provided good integration and development tools that make the coding process easier. The development environment provided by IBM Portal Toolkit V5.0 has the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

capabilities to create, test, debug, and deploy individual portlets and Web content. This Portal Toolkit is implemented as a pluggable component to be used with the IBM WebSphere Studio Workbench. The Portal Toolkit provides the following:

- Portlet project development wizards, editing and debugging capabilities.
- Portal projects that allow you to publish your portlet application on your target WebSphere Portal server machine. Your portlet will be displayed on the debug page of your Portal Server. Two test environments are supported: remote and local.

WebSphere Portal 5 is intended to support business processes in B2E, B2C, and B2B collaboration. The core of the WebSphere Portal for Multiplatforms 5 architecture is composed of the presentation services, the portal information access layer, and the portal services layer. The presentation layer includes capabilities for user configuration, Web user interfaces, and transcoding. The portal information access layer provides connectors and an open interface for unique customer connectors. Portal services have been discussed throughout this book many times. They provide user management capability, administration and security, personalization, workflow support, application integration tools, document management tools, Web content management services, collaboration, and search services.

There is also the capability to create multiple portal sites on one instance of a WebSphere Portal — this is called virtual portals. Each site has its own URL, look and feel, pages, users and groups, and search index. All sites share the same WebSphere Application Server instance, installed portlets, and document libraries. Virtual portals enable enterprises to centrally manage many different portals that are targeted to different departments within or outside the organization. Each virtual portal provides assignable administrator roles to allow discrete administration of all virtual portals.

## Apache Portal Tools and Server

At present, there are several open source portal servers. We will discuss two of them: Apache Jakarta Pluto (http://jakarta.apache.org/pluto) and Jetspeed (http://portals.apache.org/jetspeed-1/).

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Apache Jakarta Pluto was developed by IBM as the reference implementation for the Java Portlet API. This is only the reference implementation, and the developers and designers concentrated on portlet container functionality. This portlet container is later "reused" in Jakarta's Jetspeed project.

Jetspeed is an Open Source implementation of an Enterprise Information Portal, using Java and XML. Soon after its conception in 1999, it became apparent that Jetspeed was going to become an engine for Web applications. The initial ideas and implementation were far beyond the original scope of the project. There were many discussions among contributors regarding the future of this project. These discussions led to another project — the Turbine project — based on technology donated by Jon Stevens of Clear Ink. Turbine is now the Web application framework that Jetspeed shares with many other Web applications.

The main purpose of the Jetspeed portal is to make network resources (applications, databases, and so forth) available to end users. The user can access the portal via a Web browser, WAP-phone, pager, or any other device. Jetspeed acts as the central hub where information from multiple sources is made available in an easy to use manner.

The data presented via the Jetspeed portal is independent of content type. This means that content from XML, RSS (a dialect of XML), or SMTP can be integrated with Jetspeed. The actual presentation of the data is delivered to the user, for example, via a combination of Java Server Pages (JSPs) and HTML. Jetspeed provides support for templates and content publication frameworks, such as Cocoon, WebMacro, and Velocity.

## Sun ONE Portal Server

The Sun Microsystems Portal Server ONE (Figure 122) concept is based on the *Desktop* as the primary end-user interface for the Portal Server. Content *providers* and *channels* (separate Web applications) are then used to support aggregation of a variety of content and services.

The *Desktop* provides a mechanism for extending and aggregating content through the Provider Application Programming Interface (PAPI). The PAPI is a Java API that enables you to construct the basic building blocks for creating *channels*. A *channel* is a unit of content with border, header, and icons to customize it. The leaf channels display a specific type of information. Usually,

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

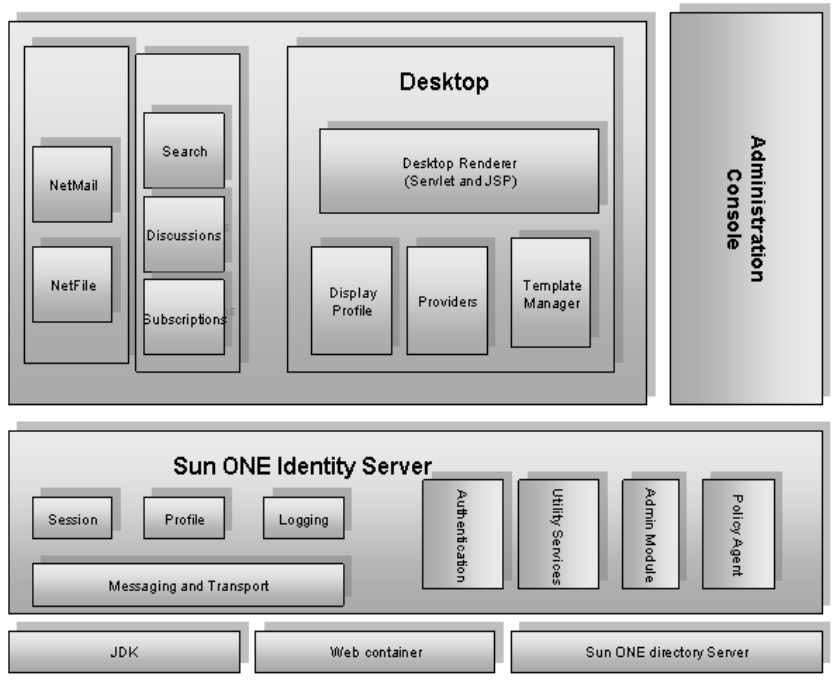
though not always, *channels* contain content. You can also have a *channel* of *channels*. This means that a container *channel* is used to aggregate other channels. A *channel* can also be the entire *Desktop* page. The container *channels* define the layout of the *Desktop*.

The Sun ONE Portal server also provides a search engine with basic and advanced search options for searching Desktop channels. Some services, such as authentication, single-sign-on, logging, profile, and session management are also supported by the Sun ONE Identity Server.

A *provider* is a Java class that is responsible for converting the content of a file, or the output of an application or service, into the proper format for a channel. Other providers offer integration services, such as an XML Provider and a Table Content Provider. The *Provider* interface defines the communication channel between a container and a provider (content provider).

This component-based architecture differentiates between applications such as NetMail or NetFile; services such as search engines; content delivery services; and presentation components (Desktop and desktop rendering engine). The Java runtime environment (JDK) provides support for all Java applications in a Portal server and its underlying components.

*Figure 122. Sun ONE Portal Server Software Architecture*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Several different vendors' Web containers are supported by Sun ONE Web Server software: Sun ONE Web Server, Sun ONE Application Server, BEA WebLogic Server, and/or IBM WebSphere Application Server. The Sun ONE Identity server provides all the necessary core services, such as profile and session management, authentication and logging, single sign-on, various utilities, support for administrative console, and policy management.

The central presentation engine — the *Desktop* — consists of a stack of interfaces. The top layer is the *Desktop Servlet,* which corresponds to the WebSphere portal servlet concept. Lower layers of the stack contain leaf blocks (such as J*SPProvider*, *XMLProvider*, and *URLScaperProvider*), several containers (for example, *JSPSingleContainerProvider*, *JSPTableContainerProvider*, *TabContainer*, and *TabException*), and a set of Provider APIs (called PAPI). The PAPI is divided into four groups: interfaces, base classes, provider context, and exception classes. The developer extends the base classes to create new providers. The Desktop servlet routes client requests for content and processing and passes them on to the specific provider object. The Desktop servlet processes the following actions:

- content: Obtains the named channel's main content.
- edit: Accesses the named channel's edit content.
- process: Processes the named channel's form data.
- logout: Is used to end the user's session.

The Sun ONE Portal Server includes the implementation of the Portlet Specification as defined by JSR 168 Expert Group. In the context of Sun ONE Portal Server, portlets[1] refer to pluggable Web components that process requests and generate content within the context of a portal. The portlets are managed by the Portlet Container. Conceptually, portlets are equivalent to *Providers*. If you prefer using Portlets to Providers and you have already some parts of an application written in a Provider theme, there is a relatively simple way for converting Providers to Portlets.

In Sun's view, the Portal engine can support not only Web applications, but it also can be extended to Web Services. A portal can offer single sign on (SSO) point of access to user-facing Web Services. A portal can also be used to deliver aggregated and personalized Web Services to other Web Services (as opposed to the end user). Furthermore, internal services of a portal, such as location and notification, can be made available as public Web Services to

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

other Web Services and applications. From this point of view, we can look at the Portal as a hub for Web Services.

The Sun ONE Portal Application Server software providers run within the JVM provided by the Web container (Sun ONE Web Server or Sun ONE Application Server). This concept is similar to the WebSphere conceptual architecture, where the Application Server also provides the Web container for the WebSphere Portal.

Currently supported operating systems include: Sun Solaris 9 and 8 Operating Systems (SPARC Platform Edition), Sun Solaris 9 Operating System (x86 Platform Edition), Microsoft Windows 2003, and Red Hat Enterprise Linux AS 2.1.

## Portal Development with Struts

As previously mentioned, the Struts framework has been used to develop Web applications. The Struts framework has now appeared in portal development, and some developers continue using the Struts framework. Struts has some positive, as well as some negative, features. Let us discuss the positive features first.

- **Use of JSP tag mechanism:** The tag feature promotes reusable code and abstracts Java code from the JSP file. This feature allows efficient integration into JSP-based development tools. Many development tools support authoring with tags, and this way we can achieve better efficiency in development and maintenance.
- **Tag library already exists:** The Struts framework encourages new library updates if additional tags are needed. This approach frees the page developers from coding, and to some extent prevents tangled code on JSP pages and extensive use of scriplets.
- **Struts are open source:** Some developers may appreciate the ability to see the source code. However, the programmers in a production environment typically do not have time to study the source code.

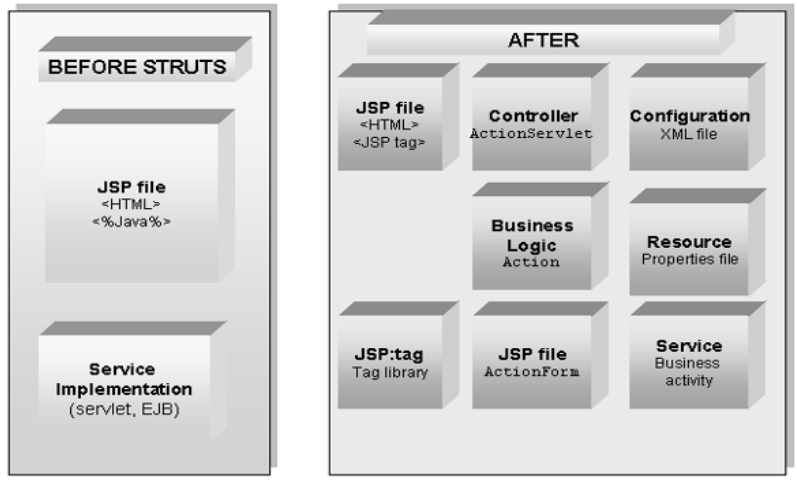
There are also some negative features that should be considered before making this framework the major development environment for portal applications.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The features that should be considered are touched upon briefly:

- *Struts development is still in its early stage*. The framework itself is undergoing changes. The code could contain small or larger flaws.
- *Struts attempted to limit the amount of Java code required in a JSP file via tag libraries*. There is still discussion going on regarding whether Struts provides the correct level of abstraction and if a page designer should get access to Java code. Experience shows that the designers try to add more and more Java code despite the fact they may not have sufficient knowledge of Java. Some frameworks, like Velocity, advocate that no access to Java code should be given to the page designer, and so they provide another language for Web development. Microsoft ASP development certainly supports this approach using ASP scripts.
- Struts is a Web-based solution for MVC2 (Figure 123), which focuses on reducing the development and maintenance effort for applications with HTML, JSP files, servlets, and portlets. Although the current extension for portal development has leveraged its scope, we believe that there is only limited scope for using Struts framework.
- The *separation of concerns* implemented by the Struts framework could introduce additional complexity. This approach certainly extends the learning curve for new developers.

*Figure 123. Struts — Implementation differences*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

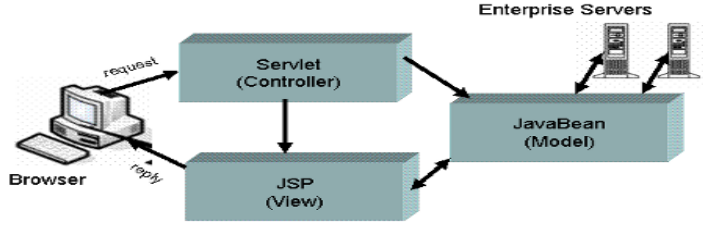
We need to explain the Struts MVC2 implementation (Figure 123). In Struts (AFTER), MVC2 is represented by ActionServlet, Action, and ActionForm. In the original framework (BEFORE STRUTS), the JSP file would typically contain some Java code that implements the evaluation of action logic. With Struts (AFTER), business logic is represented by a single action that is associated with the appropriate JSP file. The configuration file and the Resource are cleanly separated from processing logic, thus preventing entangling of a scriplet's code with HTML tags.

Software tools and frameworks change rapidly. In a few years, development moved from cgi/perl to ASP with VB toward more comprehensive Java and J2EE development environments. Struts contributed to solving some problems with separation of concerns using tags and MVC2. This approach helped in coding reusability and flexibility. By separating the problem into smaller components, we are aiming at true component-based design and supporting reusability of components when changes occur in tools or in applications. Additionally, Struts introduces separation of roles; Page designers and Java developers can focus on their specific tasks. There is trade-off in increased complexity since the Struts framework is more complex than a simple single JSP page. For large applications, Struts may be beneficial.

We would like to look at the implementation of the Model-View-Controller pattern in Struts and compare it with the implementation in the Sun ONE Portal presentation framework. The presentation framework in Sun ONE is based on a modified version MVC, called MVC2, and this corresponds to the Struts MVC2 implementation (Figure 124).

The major modification of this model is in the fact that the lifecycle of the Controller and View is not necessarily the lifecycle of the application. The Model no longer "fires" events to its Views. The Model is more generic, representing the state of the application, and it does not require any logic for registering and unregistering listeners.

*Figure 124. Sun's MVC2 concept*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Table 5. Comparison of MVC2 in Struts to Sun's MVC2*

| MVC2 in Struts | Sun's MVC2 |
|---|---|
| Client browser creates an event (request) | Client browser creates an event (request) |
| Controller (servlet) decides where to send the request | Controller notifies the View of state changes to the Model and manipulates the Model before notifying the View |
| Model represents the state of the application | Model represents the state of the application and persists across multiple servlets |
| Business logic decisions are implemented in ActionServlet and dispatched to the Action class | Model cannot reliably notify View objects about internal state changes |
| View is a JSP file | View is a JSP file |

Table 5 shows differences between Struts' implementation of MVC2 and Sun's view of this programming model. The most important difference is in implementation of the Controller. In Sun's MVC2, the Controller modifies the state of the Model. This enables the Model to maintain the state of the application.

When developing applications with the Sun ONE presentation framework or Struts MVC2, some issues should be considered:

- Code and content coupling should be reduced to a minimum.
- HTTP protocol limitations are associated with a client-initiated request response that prevents the server side code from issuing unsolicited callbacks to the client. Therefore, Swing-like implementation of user interfaces is not recommended.
- Practicality of the MVC2 and component model. Components can be anything from a reusable piece of code to a Swing-like user interface widget. The control flow focuses on the component interfaces. This approach can be seen as improving granularity of applications. The developers must approach design of reusable components carefully to avoid reuse of fine-grained components.

There is also support for building Portlets — Portlet Builder 2.0. This tool is compliant with the JSR168 specification and provides a development environ-

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

ment for writing portlets. Furthermore, a runtime environment is provided for local testing of portlets.

## Endnote

[1] Portlets are compliant with JSR 168 specification.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter XV

# Installation and Maintenance Costs

The decision regarding which Web server, portal engine, or particular technology to use is never an independent, straightforward process. Typically, the manufacturers' recommendations are well founded, and only very experienced implementers, or those who wish to experiment, should mix and match various products. A basic premise can be made (with some caveats) that all major Web/portal products are comparable, although not identical. Current portal implementations do not guarantee compatibility, in spite of the existing standards.

If you had the luxury of building a completely new system, we would recommend to first choose the most volatile (or least compatible) components, such as portal, content and document management, followed by security, availability (more stable and better understood components), ending with choice of operating systems and platforms (the most stable, compatible, and well understood components).

It is very important to remember that the total cost of the installation includes maintenance, ease of deployment, and availability of support services. Of these, the operational maintenance is the most important. Typically, the installation may develop instability and/or performance problems that need to be diagnosed and resolved. It is essential that the installation have access to diagnostic tools that can be used to help assess the following:

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- The performance of various J2EE resources and their contribution to the overall load;
- Capacity planning and the availability of free capacity to add new applications or content;
- Efficiency of load balancing, or the necessity for it; and
- Occurrence of various types of exceptions, failures and their presentation to the end user.

There are three categories of tools that will meet these requirements:

- Operating environment monitoring tools, such as IBM's Tivoli;
- Database monitoring tools, such as Quest Software tools; and
- Code and transaction tracking, such as Borland's Server Tracker.

Finally, the operational costs are dependent on the frequency of the deployment of new portlet applications, or their upgrade. Since the number of components that need to be changed in some way is much higher than in the simple Web deployment, the costs and risks are also higher. Consider the deployment of database changes to several database servers, an LDAP server, and to several application servers, all in a possibly clustered environment.

The cost of changes to application parameters, such as names or URLs, depends on the location of these parameters. In the Unix environment of a few years ago, these parameters would be probably placed in a Configuration.ini file. In a portal environment, there are choices to be made: the parameters can be recorded in JVM parameters; they can be in a Configuration.xml file; they can be placed into edit or configuration mode parameters stored in the portal database; or they can go into an application database. Each option has its advantages and disadvantages. It is necessary to consider whether the parameters need to be changed online or off-line.

It is also necessary to consider the need for multiple environments, namely development, test, QA, and production, especially in high availability applications.

To summarize — the architecture of major components of an installation has a direct cost consequence in terms of operational maintenance, and care should be exercised when making component choices. Combining products of several

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

manufacturers, a practice that has become quite common, is not recommended at the time of the writing of this book.

Using remote portlets, as described in our standards overview, brings yet another level of complexity, as UDDI search and Web Services may be involved. Rendering remote portlets promises huge advantages in the integration effort but is costly from the maintenance point of view, as the complexity is increased considerably.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter XVI

# Standards Overview

In this section, we would like to offer critical and informal views on the standards relevant to building portal solutions, as well as the variations in their implementation by some vendors. There are two standards that are highly relevant to building portal solutions: WSRP (Web Services for Remote Portlets) and Portlet API Specification JSR 168.

- **JSR 168:** Portlet Specification aims at establishing a standard API for creating portlets.
- **WSRP:** Builds on current standard technologies, such as WSDL and **SOAP:** Aims to solve the problem of traditional data-oriented Web Services (these required the applications to be aggregated before any specific presentation logic could be used for presenting the content) by introducing a presentation-oriented Web service that allows interaction with content that has been delivered by other Web Services.

As discussed in the previous section, all portlets on a portal page are aggregated to form one servlet. Further, the page aggregation process is performed by the portal server after all portlets have completed their processing of events and messages. Therefore, it is important to understand the relationship between the JSR 168 (portlet specification) and JSR-000154 (Servlets Specification 2.4, 2004).

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Portlet Specification JSR 168

Java Specification Request ([JSR] 168) (JSR 168 (2003)) deals with definitions for APIs for portlets and addresses interoperability issues for portlets and portals. JSR 168 has been developed using Java Community Process (JCP) services: an open, international, community-based organization that facilitates the development of Java standards, reference implementations, and technology compatibility kits (see http://jcp.org). Sun has co-led this standards effort, creating the specification itself and the TCK (Test Compatibility Kit) (http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/). JSR 168 has had very strong industry support; more than 20 companies participated in the initial definition of the standard. For references, please see http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/pb_whitepaper.pdf and http://jcp.org/aboutJava/communityprocess/final/jsr168/.

The goals of the specification, as discussed in JavaWorld (Hepper & Hesmer, 2003), are:

- Define the runtime environment, or the portlet container, for portlets;
- Define the API between portlet container and portlets;
- Provide mechanisms to store transient and persistent data for portlets;
- Provide a mechanism that allows portlets to include servlets and JSP;
- Define packaging of portlets to allow easy deployment;
- Allow binary portlet portability among JSR 168 compliant portals; and
- Run JSR 168 portlets as remote portlets using the Web Services for Remote Portlets (WSRP) protocol.

The specification defines three entities: portal, portlet, and portlet container. The specification compares portlets to servlets and provides a list of similarities, differences, and unique features. The JSR 168 specification also deals with all aspects of portlet APIs and prescribes in detail the concepts to which any implementation must adhere.

A *portal* is a type of Web-based application that enables personalization, SSO, and content aggregation from different sources. The *portal* also hosts the presentation layer for all aggregated sources. A *portlet container* is an extension to a servlet container. This means that a *portlet container* can be

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

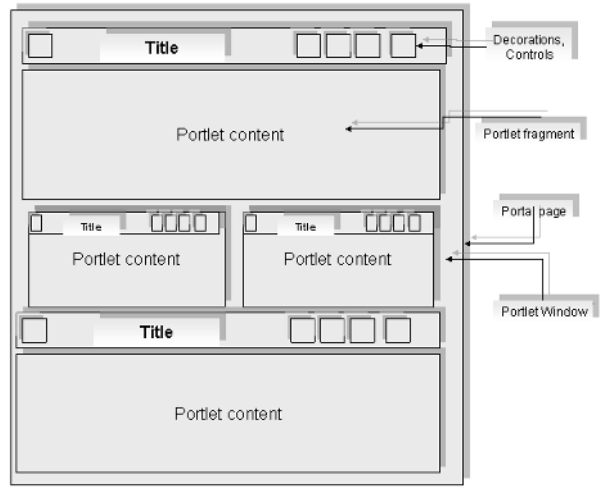
built on top of an existing servlet container, or it may implement all the functionality of a servlet container.

The specification also defines the composition of a *Portal Page* (Figure 125). The *Portal Page* itself represents a complete markup document and aggregates several portlet windows. In addition to the portlets, the page may also consist of navigation areas and banners. A *portlet window* consists of a title bar with the portlet's title, decorations, and the content produced by the portlet. The decorations can include buttons to change the portlet's window state and mode (http://www.javaworld.com/javaworld/jw-08-2003/jw-0801-portlet.html).

A portlet is a Java-based component that processes and generates dynamic content called fragments. The *portlet interface* is defined as the main abstraction of the portlet APIs. All portlets must implement this interface in order to provide default functionality. If the portlet is not hosted in a distributed environment (local portlets), the portlet container must use only one portlet object per portlet definition. Portlets must support "required modes" — Help, Edit, and View, as mentioned in the previous sections — and three life cycle phases with corresponding methods: initialization, request handling, and destruction. There is some space for optional portal vendor specific modes, which portlets may or may not support.

Portlets store preferences in the PortletPreferences object. Preferences can be read and/or written in the action phase, and they can only be read in the rendering phase.

*Figure 125. Elements of a portal page (JSR168)*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The specification section on packaging and the deployment process defines deployment descriptors (portlet.xml) and their relationship to servlets deployment descriptors web.xml. All Web resources except portlets must be specified in the web.xml deployment descriptor. Portlets and portlet related settings are captured in portlet.xml deployment descriptor. The web.xml descriptor also contains information that is relevant to the Web application as a whole: portlet application description, portlet application name, and portlet application security role mapping. The portlet application deployment is a two step process that deploys the Web application into the application server and deploys portlets into the portal server.

Portlet markup fragments are aggregated into a portal page at the rendering phase. The JSR 168 standard specifies rules and limitations on the markup elements. Disallowed tags may impact content generated by other portlets or may break the entire portal page composition. Portlets generating HTML fragments must not use the following tags: base, body, iframe, frame, frameset, head, html, and title. Portlets generating XHTML and XHTML-Basic fragments must not use the following tags: base, body, iframe, head, html, and title.

There are some more restrictions due to the fact that HTML, XHTML and XHTML-Basic specifications disallow the use of certain elements outside of the <head> element in the document. Some browser implementations allow some of these tags (link, meta, and style) in other sections of the document (for example, current versions of Internet Explorer and Netscape Navigator both allow the style tag anywhere within the document). Portlet developers should treat the use of these elements in fragments carefully.

## Portlet Relationship with Servlet Specification

Portals aggregate information from different sources and present it to a client. Traditional data-oriented Web Services require multiple mutually cooperative applications to render specific presentations for each Web application or Web Services. The aggregation happens in a servlet or a JSP that is specifically designed to present data from multiple Web resources. The main problem appears to be in the necessity to create a unique communication interface for each of these services. This type of solution is not well suited for business integration that uses a plug-and-play solution. The solution becomes more complex when data are arriving from remote data-oriented Web Services.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

We start with looking at the relationship of a portlet to a servlet. In previous sections, we have already looked at servlets as part of Web service implementation. The Servlet Specification document JSR-000154 (Servlets Specification 2.4, 2004) documents the requirements for building a servlet container (or servlet engine). We need to understand what differentiates portlets from servlets and also the similarities between them. Following is the summary of similarities and differences between portlets and servlets.

Portlets share some similarities with servlets:

- Portlets are Java-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 dynamic Web content.
- Portlets and servlets interact with Web clients 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.
- Portlets do not have a URL; consequently they cannot be independently displayed in a browser. Servlets are identified by their URL.
- 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 modes and window states indicate the function that the portlet is performing and the size of its real estate on the portal page.
- A portlet can exist several times simultaneously in one or more portal pages.
- Portlets have a more refined request handling in terms of distinguishing between action requests and render requests. Action is originated via the URL by the client or other portlet, as opposed to render, which is called by the portlet container as a response to the portlet's current state.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Portlets have more functionality and more sophisticated behavior than can be implemented with 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 that are independent of the portal's server implementation, for example session information tracking.
- Portlets store objects in two different session scopes: application wide and portlet private.

There are also some features of servlets that portlets lack:

- Portlets cannot change HTTP headers or set response encoding.
- Portlets generate fragments that 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.

Now we go back to our discussions on Web Services to compare the use of Web Services with servlets to the use of Web Services with portlets. The most important factor initially to consider is that, unlike servlets, portlets are only a small piece of a larger presentation. Servlets have the luxury of knowing they will be the only presentation resource returned to the client at any given time. Portlets, on the other hand, must understand that the presentation resource they return will be aggregated into a larger resource that will be returned to the client. As a result, portlets must consider constraints such as screen real estate, portlet interactivity, and events, as well as overall performance.

**Real estate.** Portlets may access a variety of information to assess their current position and situation in the portal. The PortletState, for example, informs the portlet about a user's request to minimize, maximize, or restore the portlet. A portlet should attempt to display the content in accordance with the requested state.

**Content rendering and portlet window state.** If the user has maximized the portlet's window, the content should adequately fill the portal page (or the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

portlet's original size). However, if the user has requested that the portlet be minimized, there is no need to return any content. It is important to remember that, even if it is not returning any content, the portlet is still executed, and any business logic encapsulated in the model will still be performed. It is not possible to dynamically change the state of the portlet except during event handling. It should be noted that the Websphere implementation of Portal recognizes three states — maximize, minimize, and normal, giving a user more choices.

**Portal page aggregation.** A servlet may be a single piece of a large Web application, and at any given point in time, only a single servlet is responding to a user's request. We can therefore predict what is being executed and returned to the client by the servlet. Furthermore, a servlet can be sure that the content it returns will not be affected by any other servlet in the system. Portlets behave differently. Each portal page is the aggregation of several portlets. A portlet has the ability to write markup to the top of the page, even though its normal content is placed inside a cell in a table. This provides a mechanism to include JavaScript functionality for the portlet. The problem is that this capability is available to all portlets in the system. Therefore, the developers must be careful when coding the portlet's elements, such as variable names and functions.

**Inter-portlet communication.** Servlets have the ability to share data, but, since they are executed serially by the client, they cannot interact with each other during a single request. Servlets use their event listeners to be notified about container-managed life cycle events. The listener is the means for executing code that is invoked by an external event. For example, HttpSessionListener interface is used to listen for the session events.

Portlets have the ability to communicate with other portlets and to be affected by other portlets in a single request. They also use listeners to be informed about events that have been broadcasted by other portlets. This inter-portlet communication provides a means to create a dynamic portlet application by utilizing interaction among multiple portlets on the same page (or multiple pages) in the application. For example, one portlet can pass the information that the user requested to other portlets on the same page or other page. This allows the listening portlets to change their presentation or behavior.

**Event handling.** In the servlet architecture, events are represented via HTTP requests and responses. For example, when a user submits a (Web) form, the HTTP method doPost() method is called. Listeners are used only to listen for life cycle events. On the other hand, the portlet event model closely implements the traditional Java event model. Portlets implement appropriate interfaces and

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

register the listeners for them. When events are fired, portlets are notified by the portal server via appropriate listeners. The Portlet API also provides WindowEvents and MessageEvents to support the response to window and message events.

**Security.** Servlets execute in a neutral environment and are responsible for validating the user's authenticity and authority to make a specific request. This is traditionally a function of the *Controller*. Portlets operate only in the context of the portal server and cannot be called directly. The portal server is responsible for authentication and for authorizing all user access. The authentication and authorization is performed prior to the portlet's execution. However, the portlets may perform some authorization in order to associate content with a specific user or role. Therefore, authentication is a daily concern of servlet developers, but it is optional for portlet developers.

## Web Services for Remote Portlets (WSRP)

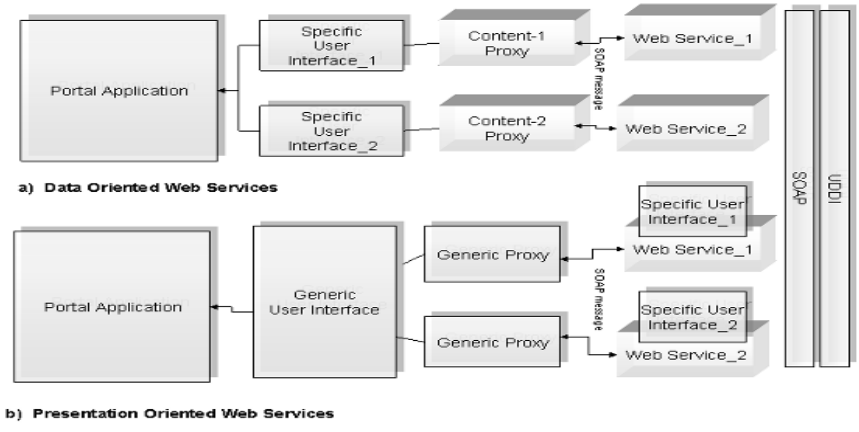
So far, the effort within Sun Microsystems Inc.'s Java Community Process aims at refining a common portlet API for the J2EE server specification, as discussed previously. Portals provide specific presentation logic to aggregate data from multiple sources, which could be legacy systems, EIS, Web Services, or EIS with Web service interfaces. The new specification WSRP (WSRP specification version 1, 2003) is the effort of the working group at OASIS (http://www.oasis-open.org/committees/wsrp), and it aims to provide a set of options for aggregating information from multiple remote Web Services within one portal application.

Traditionally, Web Services are data-oriented (Figure 126). Data-oriented Web Services do not contain any presentation or user interaction functionality. There must be an application that provides the required user interface using Web service data. Therefore, such applications must know the Web service interface (typically WSDL description) to present data in a specific format. Furthermore, a servlet or portlet must be specifically developed to present the information within a Web page or a portal page.

The WSRP specification is intended for presentation-oriented Web Services, user-facing Web Services that can be easily integrated with portals or other applications. They let businesses provide content or applications without requiring any manual content- or any application-specific adaptation by portal

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 126. Presentation and data-oriented Web Services*




presentation logic. Portals can easily aggregate WSRP services without any programming effort. The only effort required is the actual deployment of remote portlets in the local portal server (http://www.javaworld.com/javaworld/jw-08-2003/jw-0801-portlet-p2.html).

WSRP defines producers as presentation-oriented Web services that host portlets and are able to render markup fragments and process user interactions. A consumer is defined as an intermediary system that communicates with producers on behalf of its users. It collects and aggregates the markup delivered by remote portlets and renders it (via aggregated output in a portal servlet) as a portal page. The consumer can aggregate content produced by portlets running on remote machines that use different programming environments, for example J2EE and .Net.

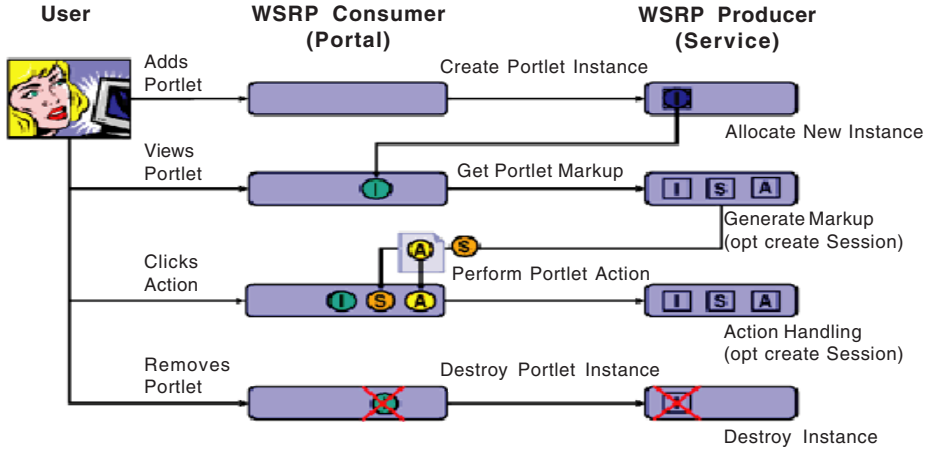
Producers are presentation-oriented Web Services with major responsibilities:

- They are capable of hosting portlets. Portlets generate markup and process interactions with that markup;
- They render markup fragments;
- They process user interaction requests; and
- They provide interfaces for self description, and portlet management.

The interaction between WSRP consumers and producers is shown in Figure 127. A user can dynamically add a portlet to the portal page. In response, the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 127. WSRP enables Web Services to be added and removed dynamically (source: http://www.oasis-open.org)*



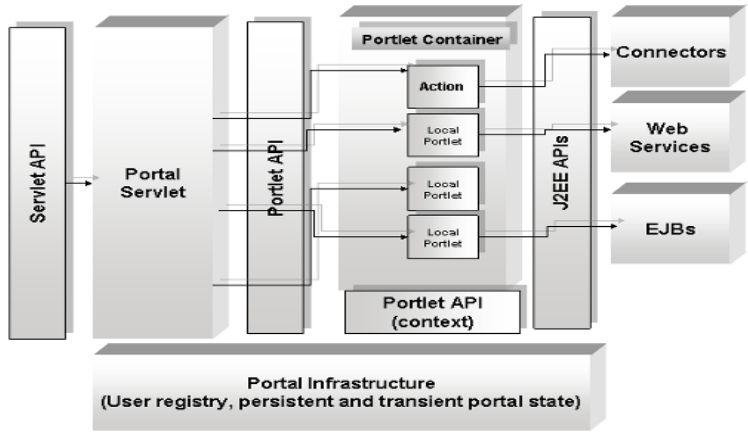

portal invokes the WSRP service. This specifies a new portlet instance, which allocates a corresponding portlet instance on the portal side. When a user wants to view this portlet, the portal obtains the WSRP markup that defines the fragment that is to be displayed. The returned markup contains portlet action links ("A") and/or a portlet session identifier ("S"). When the user clicks on the link (Click-on-Action), a request goes from the browser to the portal. The portal maps the request into the invocation of the WSRP service. The capability to maintain the session identity is provided through the parameters that are passed, such as the session ID; this allows the WSRP service to look up the previous session details. When the user does not want to access the WSRP service any more, the session is closed, the portlet is removed, and its instance is destroyed.

A typical mechanism for portlet invocation is shown in Figure 128. The portal server comprises several entities, components, containers, and corresponding APIs:

- Portal Servlet, which receives the requests, aggregates, and personalizes the content for respective users;
- Portlet container in which portlets run;

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 128. Portlet invocation mechanism and building blocks*




- Components for handling access to back-end services (Web services, Connections, and EJBs); and
- Appropriate APIs (Servlet APIs, Portlet APIs, J2EE APIs).

As component-based services, WSRP services are built on standard technologies such as SOAP and HTTP protocols, UDDI and other compliant registries, and WSDL to describe services. In order to present the content in a portlet, WSRP has to incorporate some essential context elements that are relevant for building the presentation interface (portlet) that has received the SOAP requests. These elements include user profile, client device information, locale, and markup language.

Any standards should be capable of seamless cooperation. The WSRP and JSR 168 are aligned in many aspects *(producer* or *consumer)*. However, there are some differences. Similarities and differences are discussed in the following:

- Portlet modes and window states correspond to the *consumer* side.
- URL encoding and creating URLs pointing to the portlet correspond to both the *consumer* and *producer*.
- Both standards have user-to-portlet interaction, with the action and render phases.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- The state of a portlet fragment is supported in WSRP under the term navigational state and in JSR 168 with the render parameters. The portlet rendering parameters can map to WSRPs navigational state.
- Both standards store transient state across requests with a session[1]. The WSRP sessionID references JSR 168's session when mapping WSRP calls to JSR 168 portlets and vice versa.
- Storing persistent state to personalize a portlet's rendering is realized in WSRP through the properties of arbitrary types, whereas JSR 168 supports only preferences of type string or string array. This means that WSRP *producers* based on JSR 168 use only a subset of the WSRP functionality.
- Information about the portal calling the portlet is called *RegistrationData* in WSRP, and it is the equivalent to a PortalContext object in JSR 168. All data represented through the PortalContext to a JSR 168 portlet is available in WSRP's *RegistrationData*. This equivalence represents the mapping between PortalContext object and *RegistrationData*.

As evident from this list, these concepts provide a reasonable mapping between JSR 168 and WSRP. Thus, we can implement JSR 168 portlet containers that can be accessed via WSRP and expose JSR 168 portlets as WSRP services.

The OASIS standard of WSRP version 1 (www.oasis-open.org/home/index.php) attempts to define the integration of remote content and application logic into a unified user presentation with minimum effort from the application designers. WSRP enables access to remote proxies, so the remote services can request the invocation of J2EE portal servers through their APIs and to dynamically deploy a portlet within another portal server.

The WSRP is intended for use with WSIA (Web Services for Interactive Applications), which is also being developed by the OASIS committee (http://www.oasis-open.org/committees/wsia). WSIA defines well-defined interfaces and contracts on top of the generic one. The motivation for WSRP and WSIA stems out of the fact that the current approach to Web services is not suitable for remote portals. Traditional data-based Web services require the application to provide specific presentation logic for each Web service. When the final Web service is the aggregation of multiple Web services in a specific portal page, a particular interface has to be developed to produce the aggregated view. WSRP defines a standard for interactive, presentation-oriented Web services that can plug and play with portals.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The main points of interaction as defined in WSRP include:

- A WSDL interface description for invocation of WSRP services;
- How to publish, find, and bind WSRP services and metadata;
- Markup fragment rules for markup emitted by WSRP services; and
- Applicable security mechanisms, such as billing information.

The main idea is to enable simple integration of content from internal and external content providers. The portal administrators should be able to pick the required service from the list of published services and to tie new content and applications into the portal. More information is available in an incubated project at Apache (http://ws.apache.org/wsrp4j/) or (http://www.idealliance.org/papers/dx_xml03/papers/04-06-05/04-06-05.pdf.

## WSRP Services as a Solution for Remote Access to Information

The specification of WSRP is an attempt to define a way for Web services to become "compatible" with portals, according to Thomas Schaeck, IBM Corp.'s architect of the WebSphere Portal Server and chairperson of the OASIS technical committee. Some 25 companies are working through OASIS on the effort WSRP and OASIS for Portal Deployment (Rubinstein, 2003). Kinzan Inc., which is on the OASIS committee working on WSRP and WSIA, believes the standards will help define how a portlet installed in one server can be rendered in another without the need to go through the deployment steps.

The driving idea is to create registries of portlets for remote services that allow services to be plugged into portals without any coding and to allow portals to publish portlets for use in other portals. To realize this idea we need more:

- **Aggregation phase alignment:** We have to have established markup fragment rules that portlets adhere to. Even if the View retrieves the markup, there still could be differences in the rules, and the portlet may need to be rewritten. The same comment applies to links (Click-on-Action).
- **Installation and rendering phase:** We need to define how a portlet installed in one server can be rendered on a remote server without having

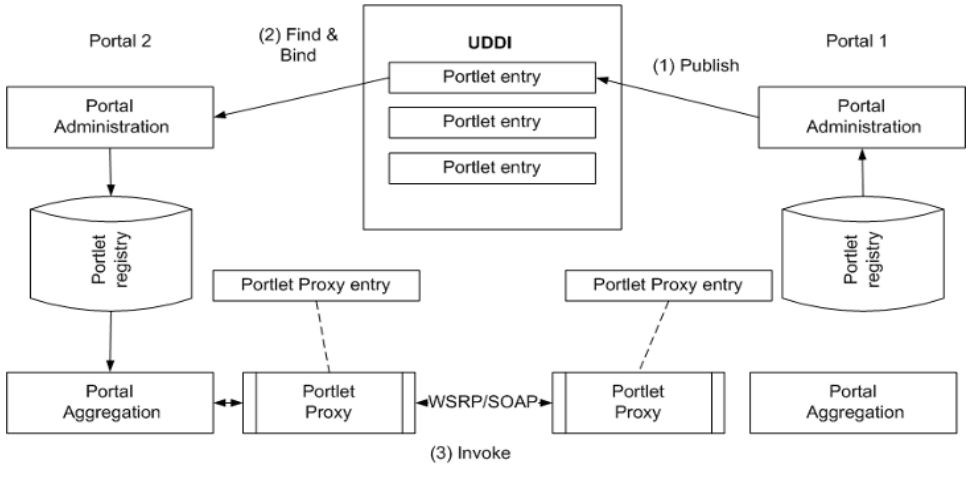
Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

to deploy it as the second instance on this remote server. This issue is also associated with removing and destroying portlets and other garbage collection rules.

- **Publishing portlets:** So far, WSDL allows publishing of Web service interfaces and binding them to ports and hosts. There needs to be also a common approach to publishing a portlet and to binding this portlet to a service using the UDDI directory. Portlets then can be found in UDDI and then bound to a portal.
- **The Portlet API and WSRP should be able to cooperate:** Previously, we have discussed the alignment of both specifications (JSR 168 and WSRP). This alignment is not substantial enough for seamless integration. It excludes those vendor specific portals which do not fully comply with JSR 168.
- **There are two options regarding how to achieve seamless integration of WSRP into portals:**
  - WSRP services may be integrated in portals through portlet proxies written to the Portlet API.
  - On the other hand, portlets may be published as WSRP services.

Figure 129 (Schaeck & Hepper, 2002) illustrates how a portal server publishes a portlet to a UDDI registry as a WSRP service so that other portals can find

*Figure 129. Publishing portlets as WSRP services*



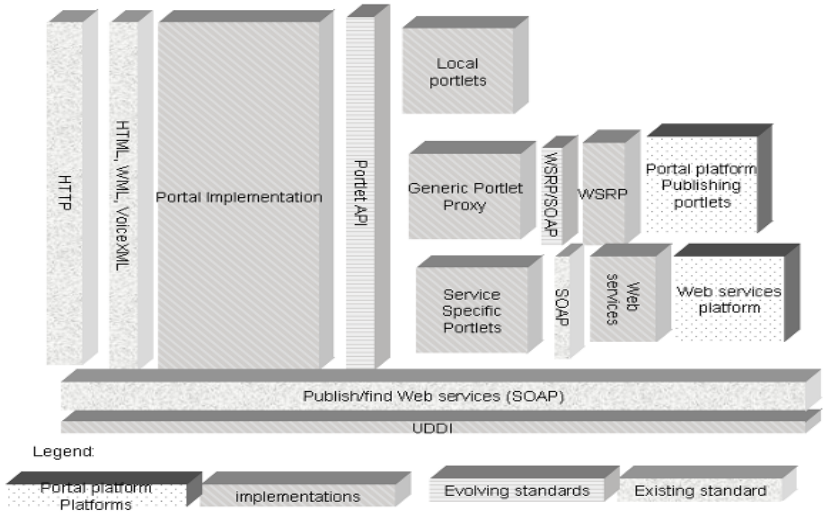

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

and bind to it. Typically, a portal has local portlets listed in a portlet registry. To make a portlet available as a WSRP service, the portal's administration console may allow administrators to select a portlet and publish it in the UDDI directory. An adapter exposes the portlet externally by providing the SOAP operations defined in WSRP.

Once the portlet entry is listed in the UDDI directory, other portals can find it and bind to the referenced WSRP service. To make a WSRP service available as a portlet, we need to create something like a portlet proxy entry in the local portlet registry with the information obtained from UDDI. We also need to wrap the WSRP service in a generic portlet proxy. As you can see, simple functionality that has been meant mostly for forming an aggregated view of multiple data sources has now reached the complexity we often ascribe to giants such as CORBA.

Once the portlet proxy entry is in the local portlet registry, users may for example select it through the customize option and include its instance on their portal pages. When a portlet proxy instance is invoked during page aggregation, the portlet proxy will marshal the method arguments in a SOAP request according to the WSRP standard and send it to the WSRP service. Then it receives the SOAP response from the WSRP service and unmarshalls the result to provide it to the portal.

*Figure 130. Standards relevant to portals*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## WSRP and Portal Interoperability Issues

Standards exist for both the client-to-portal protocols and the different markups. Devices, or gateways, typically access portal servers via standard HTTP and, depending on the device type, different standardized markup types may be used, such as HTML for the typical Web browsers, WML for WAP phones, VoiceXML for access from phones via voice and a voice gateway, and so forth.

Existing, well-established standards (Figure 130) include a Web service description using WSDL, protocols for publishing and finding Web services in directories such as UDDI, and invocation of Web services using SOAP messages. There are still standards such as JSR 168 and WSRP, which are in their evolutionary stages. The internal implementation of portlets can be different in many ways; we feel that there is still a gap between Java Portlet Specification (JSR 168) and some imaginary common portlet standard to which all portal providers would adhere.

Portlets rely on container infrastructure that is accessible via the Portlet API for functions such as access to user profile information for the current user; access to the window object that represents the window in which the portlet is displayed; participation in the portal window and associated action event model; access to Web client information; inter-portlet messaging; and a standard way of storing and retrieving persistent per-user/per-instance data such as PortletSettings object and PortletData objects. Applications that provide portlets typically select a few major portal platforms for deployment. Even in this situation, they still have to implement more than one version of a particular portlet. From a portal server perspective, only a subset of portlets runs on each individual portal server platform. Portal customers may be dissatisfied in situations where their required combination of off-the-shelf and locally developed portlets will not run reliably on their selected portal server.

The lack of standards in the portal space also has led portal server platform vendors to define proprietary APIs for local portal components and for invocation of remote components. The resulting diversity of interfaces creates interoperability problems for portal customers, application vendors, content providers, and portal software vendors for the following reasons:

- Application vendors who want to provide portlets to make their applications accessible through portals need to provide different versions for the different APIs that run on the different portal server platforms. This results in high development costs.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- Customers who develop their own portlets have to invest significant effort into portlet code that will typically run only on the portal server platform on which it was developed. This tends to lock customers into their particular portal server platforms or at least makes it very hard to migrate.
- Content providers have no easy means to publish their content in a form that allows it to be found by and bound to portal servers in a plug and play fashion.
- The smaller portal software vendors have difficulties convincing application providers to provide portlets for their particular platforms.

## Business Integration and WSRP

The diversity in portlet implementation and portal servers seems to be the major obstacle for introducing portlet registries and for using UDDI to register portlet interfaced Web services and WSRP. Imagine the following situation: Company A wants to introduce a B2B solution with Company B. Company A is running WebSphere Portal 5, while Company B is using Microsoft's .NET and SharePoint Portal. Each portal supports slightly different versions of Portlet APIs. Whilst WSRP seemingly provides an elegant solution, even a small amount of incompatibility in portlet APIs could lead to the need to write many new portlets on both sides or to painstaking deployment and customization of several portlets.

The majority of today's commercial portal products support JSR 168, and customers who develop JSR 168 compliant portlet can move them from one vendor's portal to another. WSRP enables development and deployment of portlets that can then be delivered anywhere. This enhances interoperability between portals from different vendors, allows access to presentation-oriented service in other companies, and brings together third party portlets.

WSRP also provides a feasible way to build federated portals. Federated portals consist of a network of interoperating portals, where resources hosted on one portal can be made available to many other portals. The advantage is seen mostly to be in reducing the number of Web assets managed by the company.

Because a portal can access content from a portlet in any location, business units can write and maintain their own set of portlets locally. This can be done for each business unit within its local infrastructure so that all portlets within a single portal do not have to be deployed on a single portal instance. Updates and changes to content, which in the past were difficult due to firewalls and

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

differing deployment schedules, can now be realized on a per business unit basis.

WSRP is seen as a way to reduce the cost of portal maintenance. The argument to support this belief stems from the fact that portlets can be deployed locally on smaller machines and used remotely. This approach results in a reduction in portal infrastructure and total downtime. WSRP services share basic interfaces with WSIA, which is also being developed by the OASIS committee (http://www.oasis-open.org/committees/wsia). As already mentioned, WSRP is intended for use with WSIP. This interdependence could potentially mean some significant limitations on adoption of this technology in industry. The compliance with WSIP recommendation will likely result in additional development costs to the existing applications, thus making the integration more expensive.

So far, portal traffic is managed by a load balancer. Portlets experiencing heavier traffic can be placed on a machine with more computing power, while remote portlets can stay untouched.

## Endnote

[1] WSRP operates over connectionless technologies. Consumer and Producer must handle stateful information (see www.oasis-open.org/home/index.php for more information).

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter XVII

# Managing Portals

## FAQ

Can a development manager with experience in other Web technologies easily become a successful portal development manager? Is there anything specific that a portal development manager must know about the technology? Are the best practices in Web development applicable to portlet development?

Portal development and other kinds of Web development typically include *informational* and *functional* components. The informational components are documents and content (such as hyperlinks, plain textual information, and graphics) that are placed in portlets and that can be equipped with search capabilities. These components may have capabilities for information processing, such as content management, publishing, and document management. The functional components provide access to data processing through functional portlets. Often, the two types of components reside on the same page. A typical but simplistic example would be a portlet that provides access to accounts receivable, with another portlet providing help or training on the subject of receivables. The two portlets are related and reside on the same page.

Software development methodology is a generalized, process-oriented set of assets that strive to be technology independent. However, the methodology will

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

not provide all of the answers for portal development. In particular, a portal development manager must be aware of the technological properties and constraints, because there is a big (and new) range of issues that must be resolved in the development process. Let us list some of them, with suggestions about how to find the answers:

- *Should we publish documents in a portlet or in a Web page?* Depending on the document management system, the number of published documents, and their nature, the answer lies in careful analysis of these properties. Ideally, a document management system should encompass both technologies in such a way that the documents or content are managed by the authors and reviewers and not by the technical staff. Similarly, an information management system should enable publishing both on a Web page and a portlet. The advantage of a portlet is that it may coexist with a different but complementary portlet on the same Web page. So the answer is dependent on the capabilities of the document and information management system and on the business requirements.
- *How do you project taxonomy into portlet navigation?* A site categorization plan is needed before the site is constructed. The navigation should be derived from the taxonomy and not the other way around. It is necessary to stress that the navigation and the taxonomy are not synonymous.
- *Is it necessary or advantageous to implement "portlet-to-portlet" communication?* The nature of the portlets involved will answer this question. In most cases, this will not be needed, but in some cases it will be a distinct requirement. Note that it is possible to have one portlet with several human interfaces (typically implemented as JSPs). The portlet to portlet communication is complex, therefore expensive, and it binds the portlets logically. For example, in a tourist-oriented portlet, the user chooses a geographical area, while another portlet displays information about the weather in the area.
- *How can you ensure that the document management system correctly reflects the portlet content? How can you ensure that the hyperlinks in the portlet link to the version of the document that is the most desirable?* For example, if a document such as a company policy is valid from a certain date, the link should always point to the correct version of the policy. The document management tool should ensure this. The

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

development manager should insist that the publishing mechanism ensures the correct document is available under the correct name in the correct directory. This is a very serious issue in large companies with ever changing policies or bulletins.

- *Should content and function portlets coexist on one Web page?* There is no good reason why not, but one should understand the reasoning. A good example of the coexistence of content and function is the Help portlet that accompanies the functional portlet.
- *Is there a significant cost in deploying content portlets?* The deployment process will become more complex and longer. The deployment is different from functional portlets in that the links between the content management and the portal must be established.
- *Will the single sign-on work in the system?* With legacy systems, this will often be very difficult to achieve without programming changes. Often, as a second best option, the *same sign-on*, rather that the *single sign-on* is implemented. In the same sign-on, the user must sign on several times but with the same credentials (that is, user id and password).
- *What kind of personnel and skills are necessary in the development team?* There is a need for portal and portlet architects, analysts, portal developers, content analysts, content developers, deployment administrators, and testers. It is advantageous to combine these skills in one larger team, as the knowledge and skills need to be interchanged between content and functional developers.
- *Will the style sheet apply to both types of portlets? What about skins and themes?* Although the style sheets may be applied individually to Web pages, it is not a recommended strategy. The style sheets should be lean and mean (for communication efficiency purposes) and shared between Web pages, therefore portlets. Different stakeholders must come to an agreement regarding styles, as well as look and feel, including fonts, styles, colors, various widgets, and ornaments.
- *Where are the risks in development?* The biggest current risk is the lack of standardization in developing portlets. We can assume that this is just a temporary problem that will be overcome. Taking into account that the majority of costs are in maintenance, the development manager will try to minimize the exposure to non-standard technologies. The second large risk is the number of cooperating components in the development process, such as open source tools, frameworks, and programming systems.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Similar to the client-server processing, the number of different skills necessary for the development increases the risk and costs.

- *What kind of support team will be necessary after deployment? How can you formulate the service level agreements?* It is necessary to assume that the composition of the support team will be similar to the composition of the development team; only the size will be different. Therefore, support is quite costly. The reduction of the development and maintenance costs is still not possible to achieve.
- *What is the critical size for a user population that justifies the use of portals?* There is no real reason to dedicate portlets to a large population. It is the "modus operandi," or a person's manner of working, that decides whether the portlet is suitable or not.
- *What "extras" need to be tested in the development phase?* It is necessary to test the whole site, including themes, skins, navigation, all hyperlinks, and, of course, the function, and content.
- *Will the portlet design influence the system performance? If so, how?* One of the factors that influences the response time is the size of themes, skins, and style sheets. These need to be transferred via communication lines, and therefore influence performance.
- *Should the edit mode be used to customize the portlets? How will it change the behavior and navigation in the system?* **Edit** and **configure** modes have different functions, as described in the previous sections. These should be used for customization, as they fit the modern view of the *user administrator* customizing various properties of the deployed portlet, rather than the programmer or a support person. In general, **edit** mode should be used by the users, while **configure** mode will be used by the administrator. The technical personnel should not customize the portlets in production.
- *Should one portlet be shown in more than one page?* Generally, yes, the portlet should be regarded as a building block that can be reused if necessary and convenient.
- *What disaster recovery services should be designed for portals? Are there any differences between portals and Web Services? How can you deal with the situation that some portlets are more critical for business than others?* The general principles of disaster recovery apply to portals as well. However, it is possible to envisage a situation where only one of several portlets on the Web page becomes unavailable. This

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

is specifically relevant in the situation where remote portlets are displayed on a Web page. The interdependency between portlets that exchange messages should be taken into account. In principle, the disaster recovery should apply to the Web server and portal applications, rather than to individual portlets.

- *What is the influence of page size on the performance of the portal?* The themes and skins influence the size of the page; therefore, they influence the perception of the performance of the portal.
- *When using portable devices, will it be difficult to adjust the portal interface to the limited real estate?* Suitability of portable devices to portal interfaces is similar to Web pages, as the portlets exist on a Web page. Portable devices will use a different GUI style than full-scale devices.
- *Since portals integrate many applications, where will the editing and configuration information be stored?* The configuration values are stored in the service database used by the portal. This fact influences the disaster recovery plans, which must take into account the service database and its high availability.
- *What methodology should be used in the development of portlets?* There is no fundamental difference in the development methodology of a portlet and a Web page. However, it is worth mentioning that the portlet generally takes longer to develop than a servlet.
- *Can a portal page URL be bookmarked?* It can, but it should not. Often the portlet creates a session with the server, and it uses various states and/or state information for its functions. If a Web page containing the portlet is evoked though a URL, the state information may be lost. The programmer should prevent duplicate invocation of the same portlet, and he/she should also prevent re-entering the portlet other than through the page navigation. There is a well-known pattern describing the prevention of bookmarked entry, using tokens.
- *How do you provide printing from portlets?* Printing does not come automatically with the portlet. It needs to be programmed. This increases the cost of development.
- *How can you estimate portlet development costs?* It is necessary to take into account the MVC pattern when estimating, as these components will typically be developed using J2EE or pure Java and JSP. A good practice is to develop metrics for each type of component and to categorize the estimates in terms of "complex," "medium," and "simple."

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- *How can you implement the portlet and portal help text?* Both types of Help can be developed using some kind of document management system that can generate portlet fragments. This way, the Help is displayed in a separate portlet and associated with the relevant portlet or portal. The important thing to remember is that traditional (client-server) Help creation systems cannot be employed.
- *How do you charge customers for placing portlets in specific positions on the Web page?* It is possible to create a charging model that utilizes the advantage of placing the portlet in a specific position on the Web page. This is different from the traditional Web page design, as the position of the portlet on the page can be controlled at the deployment/administration level.
- *How do you address the level of sophistication of the portlet GUI? A* GUI is typically created through JSP. The lack of expressiveness of JSP can be supplemented with Javascript. Please be aware that the Javascript can be suppressed by the browser settings, and the portlet may appear to be not working properly.
- *What navigational strategies should be used in portals?* It is recommended to use navigation panes, buttons, and hyperlinks, rather than back buttons and bookmarked URLs. The nature of portlets is such that they maintain the status when the user navigates to another portlet on the same page. This can be confusing to users, as they instinctively expect the portlets to start from their initial state each time the portlets are re-entered.
- *What is the behavior of a portlet when a link within it is hit*? Typically, a link within a portlet takes the user to another Web page. This page will take up the size of the portal page and therefore "replace" the portal page. Alternatively, the page can create a new window and leave the original portal window untouched. However, in the latter case, the designer is left with the problem of how to prevent the user from opening too many new windows unwittingly. Once the new page is displayed, the next problem is one discussed before — how to get back to the original Web page. Various navigational schemes are in use; most involve two-to-three level menu systems implemented as trees or buttons or a combination of buttons and trees. The development manager needs to be aware that the navigation does not come free but needs to be implemented and maintained, especially when the navigation must work in conjunction with the security system.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- *What is the meaning of the minimize and maximize buttons in portals?* The minimize button transforms the portlet window into a very small window, but the window itself does not disappear from the page. The maximize button restores the window to its original size — typically not to the full screen size. The behavior of these two buttons is very different from the equivalent Web buttons.
- *What are consequences of the different behavior of the "back" button?* Since the portlet maintains its state, the use of the browser back button often causes surprises to the user; when pressed, the back button returns the user to the last used state of the portlet; there is no initialization of the portlet. It is recommended to discourage users from using any navigation than the one designed into portal itself. This can be achieved by obscuring the browser navigational buttons and providing the navigation in the functional space of the application or content.
- *How can you manage the uptime of the portal?* It is, of course, most important to keep the portal software running. There are two other situations when the portal or its parts may be down: (1) when installing a new version of the portal infrastructure, and (2) when deploying portlets. The operation and support people must address these situations in the operation and DRP plans. In some situations, the portlets can be deployed while their previous versions are running. The designers of the system must be aware of the service level consequences when deploying mission critical portlets.
- *Caching strategies — what are they and how should they be used?* To improve performance, various caching strategies and products can be used. These strategies must be included in the detailed design of the portlet application.
- *How many portals should there be in an enterprise?* The question of the number of portals should be transformed to the number of types of portal. For example, the portal operating on the Intranet should not be mixed with a different portal operating on the Internet for security reasons. Generally, there is no other good reason for maintaining many different portals except performance optimization.
- *Should there be one code base for the portal applications?* There is no compelling reason to keep the code together. However, for the maintenance and deployment of the code, it may be practical. Remember that the war packaging file contains all classes and resources that are needed for

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

the application. It is possible to package them in such a way that the common or shared resources are accessible from all packages.

- *Is it possible and advisable to implement shortcut keystrokes in a portlet?* It is possible but not advisable. There are two good reasons for it:
  - Imagine that there can be more than one portlet on a portal Web page. The portlets are installed on the page by the portal administrator. Therefore, the decision of which portlets should be on the page is made by the business owner, rather than the designer. Consequently, there could be a clash of keystrokes and confusion to the users.
  - There is a chance that the keystroke is sent to either the wrong portlet or all portlets. It is possible to send the keystroke to the portlet in focus, but that can be risky if the user does not realize it. We recommend that the keystrokes shortcuts not be used, which is also a common practice on the Web pages.

In conclusion, it should be clear that the Web development manager or designer must be equipped with new skills when taking part in portal development, whether the applications are informational or functional. There is a distinct paradigm shift in the portal graphical user interface, the function of portlets in information and data processing, and in the networking capabilities of portlets and Web Services. Despite this paradigm shift, many portal development managers and designers try to mimic the Web paradigm in portal. This raises two questions: why should you use portal in such a case, and why should the new paradigm be used?

Finally, the fundamental change in the portal approach requires a substantial educational and training effort, reminding us of similar efforts that occurred when Graphical User Interfaces first appeared.

## Future Trends

While having a crystal ball is a great privilege that is not given to many people, we can attempt to see the pointers to the future:

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- Convergence of portals — portals will become integrators of applications that are not necessarily homogenous, magnifying the capabilities of the user many times over.
- Standardization of portals — portals, and specifically portlets, will be standardized so that the interchange of packaged products will be possible. Also, the remote portlets, combined with Web Services, will become more common and practical.
- Open source portals and portlets should gain popularity together with the standardization.
- The portal will become the enterprise desktop of the future. The new paradigm will be accepted, together with the single sign-on and security standards.
- Wireless usage of the portal will take over enterprise computing.

If the portal is to become the enterprise desktop of the future, reports Hurwitz Group, the portal market space and offerings must overcome the notion that the portal is nothing more than an overdressed browser interface.

For the market to reach its potential, applications must be delivered through the portal. The portal must make its presence known beyond just the presentation of data through a portal interface. Application delivery and usage through the portal is the validation point. The ability to aggregate, change, and deliver composite applications built from parts of competing vendors' existing applications implies a tremendous amount of programming for the portal software vendors and the portal customers. This programming necessity will slow portal adoption for application use. Over time, as composite applications become the norm, the portal software vendors (and system integrators) will provide tools to reduce the programming needed today.

The changing development practices will see new types of portal development personnel, new tools, and new approaches.

Finally, the opportunity in the wireless portal space will be huge. The form factors available for wireless devices to access and use portals and portal applications will far outnumber enterprise desktop computer portals. Building the composite applications for the enterprise will be the stepping-stone to providing the tools (auto-discovery, auto-aware, auto-connect, and guaranteed delivery) that will drive the wireless portal space.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

# Section IV:

# Grids as Virtual Organizations

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Chapter XVIII

# Grids

## Grid Goals

This section describes some of the more important goals of a Grid system. The Grid carries with it many ideas central to other sharing technologies, such as the network operating system (NOS), and cluster computing. A common aim among these technologies is to maintain a single system image. Such a system would run across multiple machines, allowing its users to reasonably think they were using only one machine, when in fact they are accessing resources on any of the connected machines (Biswas, Lazar, Huard, Lim, Mahjoub, Pau, et al., 1998).

### Transparency

Transparency is a strong characteristic in a NOS and Grid network. (Biswas et al., 1998) provides useful definitions for two levels of transparency: user transparency, which gives users the illusion of one computer; and application transparency, which gives applications the illusion of one computer. User transparency is easier to achieve because it is software that performs the translation somewhere. Application transparency is harder to achieve because it implies an additional software layer between applications and the operating system unless the operating system itself does the translating, in which case it probably counts as a true single system image. Of course, cluster computing

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

(Tennenhouse, Smith, Sincoskie, Wetherall, & Minden, 1997) lists eight network operating system transparencies in that a system might address in order to provide the illusion of a single image.

- **Location transparency:** a user does not need to know where a resource is physically stored in order to use it; no resource would have a file prefix consisting of a machine name (even though the resource might need to be addressed by the operating system using the machine name).
- **Namespace transparency:** the same naming conventions (case sensitivity, directory separators, drive identification) must be used to identify all resources on the network operating system.
- **Logon transparency:** a username and password identifies a user for all resources on the network operating system. This is also referred to as single sign on.
- **Replication transparency:** the network operating system may maintain several copies of important resources on multiple machines in the system, but a user will never see more than one.
- **Local/remote access transparency:** access controls and directory services must be handled invisibly so that remote resources can be used in the same way as local resources, for example, the user should not be able to tell the difference between them anyway.
- **Distributed time transparency:** the internal clocks of all machines in the network operating system must be synchronized. This is crucial to the parallel processing of some tasks.
- **Failure transparency:** the user will never know if a network error occurred because the system should be able to recover and use failsafe techniques to make sure the request succeeds.
- **Administration transparency:** there will be just one administration interface, integrated with all local management services.

While the idea of a network operating system has for the most part declined, the ideals live on in the goals for Grid networking. All of these transparencies suit the idea of a Grid network where a user may never be aware that they are even accessing applications and resources on a Grid network as opposed to their local machine or a single server.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Exploiting Underutilized Resources

Most systems involving multiple computers will have many machines that are not used all the time — or even a majority of the time. A goal of the Grid network is to put this unused CPU time to good use. It could be that a project can only be completed in time if extra computer resources are devoted to it. It could be that a Web-based application needs access to extra processing power on demand to deal with unexpected peaks in usage and still responds reliably to all requests. All of these examples involve making greater use of otherwise idle equipment.

## Parallel CPU Capacity

Many jobs can be decomposed into smaller sub-jobs and run on multiple machines. If these tasks can be completed in parallel, the total time taken to complete the task can be greatly reduced. Perfect scalability occurs when a job run on 10 machines will take half as long as it would if run on five machines. Applications that split jobs into sub jobs must be aware of dependencies between sub-jobs and have ways of managing these dependencies. Bad management of dependencies can lead to deadlock or, more likely, too much time being spent waiting for a sub-job to complete before the next one can be run.

## Grid Applications

The underlying motivation for a Grid network will always rest on what applications will be run on it. Grid applications are scalable. They are able to access new resources as needed in order to cope with increased requirements. They are able to run sub-jobs in parallel to get a job completed faster.

It is important to understand that not all applications can benefit from Grid technology; it does not make sense for many applications run in parallel, many applications do not need to scale. Most office applications would fit into this category. Additionally, there is no easy method for transferring applications to the Grid (Ferreira, 2002). This means that already existing applications must be specifically adapted to a Grid. The modification may involve a complete re-write of the application or the authoring of a wrapper or façade to provide the legacy application with an interface to the Grid.

There are some practical tools that skilled application designers can use to write a parallel Grid application. However, automatic transformation of applications is a science in its infancy. This can be a difficult job and often requires top mathematics and programming talents, if it is even possible in a given situation.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

New computationally intensive applications written today are being designed for parallel execution, and these will be easily Grid-enabled, if they do not already follow emerging Grid protocols and standards.

## Collaboration through Virtual Resources and Organizations

**Virtual resources for collaboration.** Resources accessible through a single computer impose severe limitations on collaborative group projects requiring frequent use of the resource. Grid systems allow computer resources attached to the Grid to be shared by other users of the Grid, including data, equipment, software, services, or software licenses.

Virtual organizations for collaboration: Users of the Grid can form groups with specific privileges assigned to them for access to required resources.

## Access to Additional Resources

**Access to additional resources when needed.** A Grid can provide access to increased quantities of other resources, which a project may need in order to complete in a timely manner.

## Resource Balancing

Some applications that are Grid enabled can split a large task into smaller sub tasks. A Grid can help such applications by moving sub tasks to machines with low utilization. Using this concept of load balancing, Grids can handle peak loads for an organization in two ways: an unexpected peak can be sent to idle machines; or low priority tasks can be rescheduled or canceled to make way for higher priority work.

## Reliability

Grids offer vast potential for fault tolerance and increased reliability with the ability to duplicate tasks to other machines in cases of failure or corruption.

## Management

Grid management can be achieved through the setting of policies and priorities: With the virtualization of heterogeneous resources on the Grid comes the ability to better visualize resource levels and implement controls over them. This in turn

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

implies that each resource must be remotely manageable through a single interface. Policies and priorities can be applied to visualizations, which can be aggregated to provide a single point of management over grouped resources.

This may involve changes to the operating system to support Grid computing, to provide facilities such as resource scheduling by name or parallel processing primitives. This may occur if the techniques utilized in certain Grid networking technologies become popular enough to see inclusion in the OS itself. Until then, it is more likely that such features will remain in APIs and frameworks hosted by the OS.

# Grid Resources

The Grid is made up of many machines connected through a network, each of which shares some or all of its resources. Machines on a Grid can be called nodes, clients, hosts, or just resources. Machines on the Grid may freely share resources with all users of the Grid or may put certain restrictions in place. Note that word "machines" is used instead of "computers." Ultimately, all resources on the Grid are shared through various computer systems, but it is not true that all resources on the Grid are computers as in desktop PCs. For example, machines on the NEESGrid project share resources offered by telepresence and teleobservation equipment (Kesselman, Prudhomme, & Foster, 2004).

In this section, we shall examine what sorts of resources machines on the Grid may offer.

## Computation

As it is most commonly understood, a Grid network supplies spare computational power to meet unexpected spikes in demand for Grid applications. Computational power is provided in the way of CPU time from whichever of the Grid attached machines can spare it. The processors on a Grid do not have to be of the same type or speed. They do not have to support the same OS or be on motherboards that have memory, storage, or network connectivity. As long as they are capable of running the Grid platform, different processors supply similar resources with respect to Grid clients. Generally speaking, they accept a Grid job, process it, and return whatever result is required.

It is, however, possible for a Grid platform to discriminate between processors based on certain QoS requirements. For example, if a job has a high priority,

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

it may be given to a faster machine if the Grid platform is capable of reporting such information about the processors on top of which it is running.

The Grid applications are likely to have Web application front ends, for example accessed through Web pages. Web pages are an ideal medium for many Grid front-ends. The client is a browser enabled with appropriate plugins such as JRE or Macromedia Flash, which implies very little in the way of client installation. Nearly all of the processing in such an application goes on at the "back-end," the Grid network.

There are three common ways for a Grid network to exploit the computational resources of the machines at its disposal. All three have this in common: they all take processing away from the Grid client and put it into the Grid network. The easiest is to run the required application on one of the available Grid machines rather than locally on the Grid client. The second technique is for "embarrassingly parallel" tasks. These are jobs that need to be run many times over, usually with different sets of parameters each time, where there are *no dependencies* between tasks. This means it does not matter which processor runs which job and in what order the jobs are run. The application must know how to send the job repeatedly back into the Grid network for execution and collate results as appropriate for return to the Grid client initiating the request. This is most likely to be some sort of batch processing job.

The third and arguably most complicated technique is to design the Grid application so that it may split up a larger job into smaller sub-jobs that can be executed in parallel on multiple Grid machines. The application must know how to parcel a single job into sub-jobs, send the sub-jobs back into the Grid network for execution, and collate results as appropriate for return to the Grid client initiating the request. If there are any dependencies between sub-jobs, then a scheduler or queuing mechanism may be involved. As with any scheduling problem, complicated dependencies make it harder to calculate a workable schedule to mange those dependencies. Scheduling can quickly become more difficult than splitting a job into sub-jobs.

We can measure how efficiently the Grid network deals with these kinds of tasks with a metric called "scalability." In this context, scalability is a measure of how well a task's completion time can be shortened by adding additional resources. A "perfectly scalable" task is one that if, by doubling the number of processors working on a job, the time to complete the job is halved. This is more likely to occur in the second of the techniques mentioned earlier, and there are many factors limiting how scalable a task can be. For example, if a job can

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

only be split into four sub jobs, putting eight processors to the task in unlikely to halve the completion time.

## Storage

After computational power, the second most common resource shared over Grid networks today is data storage capacity. Grid networks that are for the most part devoted to data storage are often called "data Grids." The type of storage can be primary, such as RAM, or secondary, as in hard disk space. Primary storage is faster, but volatile in that when the power goes off, the data stored on the primary storage is gone. Secondary storage is persistent – data is still there when the computer is turned back on.

It is most important for data storage resources on a Grid network to be accessible through a unified scheme. In particular, the Grid network should implement location transparency, namespace transparency, and replication transparency. A Grid storage user should be able to use a unique name — which may include some form of path — to refer to a file, whether that file is actually copied on five different machines or split up so that parts of it are stored on 10 different machines. A Grid storage user should have access to far more space than a single hard drive can hold, accounting measures notwithstanding. A Grid storage user should expect more reliability from Grid storage than the average hard drive offers. Files can be automatically replicated in the background, cached "locally" for faster access, and always protected from viruses.

Some Grid storage systems offer quite advanced caching, replication, and synchronization mechanisms so that it is nigh impossible to lose a file or even lose access to a file when it is being shared and modified by multiple users.

## Communications

Communication as a resource in the Grid can be qualified as intra Grid communication, that is, bandwidth between nodes in the same Grid or external communication, for example, bandwidth between the Grid and the Internet. In a Grid network where intra Grid and external bandwidth can be accounted for, many applications will find it vital to secure enough bandwidth for particular tasks. Some systems may implement priority scheduling and routing mechanisms to provide QoS guarantees in terms of metrics such as minimum bandwidth, number of lost packets, or latency. Such management systems can better visualize network topology and be able to identify bottlenecks and record performance data.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

### Software and Licenses

When an organization has a limited number of licenses for particular software, the Grid network can assist by providing the software on a single machine accessible to all Grid clients. The software item may only run on one machine, but anyone with access to the Grid network can submit a job to that software from his or her own Grid connected computer. Naturally, this is an enticing proposition for software titles with exorbitant multiuser license fees. Conversely, vendors may be proportionately reticent about supporting an application for the Grid, knowing they will miss out on extra license fees, especially if they have to modify the application to "Grid enable" it.

Knowledge of licenses as another Grid resource can be useful for some advanced license agreement terms, for example, a license that permits a title to be installed everywhere on the Grid but only have a limited number of instances of that title running at the same time. The Grid network needs to be aware of how many instances are running and upon which machines they are running in order to make decisions about where to send a job involving that application.

### Specialized Equipment and Capabilities

Similar to the chapter "Software and Licenses," Grid networks may also be designed to allow access to specialized equipment attached to the Grid. NEESit, discussed later in this chapter, is a very good example of this, where the Grid provides distributed and remote access to equipment for conducting earthquake simulations such as shake tables and centrifuges.

The Grid network may have other specialized resources, apart from hardware. Consider the case of software that can only run on a particular OS or a program that is known to work particularly well on a certain Grid node. If the Grid network is aware of those special capabilities, it can consider that knowledge when deciding where and when to send jobs.

## Grid Architectures: Physical and Logical

There is no single accepted definition of what a Grid network actually is, and there is certainly no agreed upon standard architecture. However, there is a set of principles reflected in the design of most Grid networks that allows the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

topology of the Grid network to scale in response to change, and the influence of these principles shall be discussed in this section.

Recall that the goals behind Grid networks include exploiting under-utilized resources, making use of parallel CPU capacity, providing mechanisms for collaboration, and offering access to additional resources on demand. These goals are signs of a network of resources seeking to expand. Grid networks are designed to be scalable in many different ways. The Grid can grow across organization, geographic, and inter-Grid boundaries.

## What is a Grid Network?

Before we discuss how a Grid topology can scale in response to change, we should attempt to put in place a clear definition of what a Grid network is. We have already mentioned that a goal for Grid networks is to share resources such as CPU time, storage, or specialized hardware. But there are many other types of technology that already do this, such as the Network Operating System (NOS), Cluster computing, or Network File Systems. Do these technologies also count as a form of Grid network? What separates a Grid network from other technologies that are good at sharing resources? Search the Internet, and you will find references to many types of "Grid" technology: Computational Grid, Data Grid, Storage Grid, Cluster Grid, Science Grid, Knowledge Grid, and Sensor Grid are just a few. Do all of these count as Grid networks?

Ultimately, the worth of Grid technology will be derived from the results it delivers, not whether it can legitimately be referred to as a Grid technology. However, we still need a relatively clear definition of the term to prevent different parties from using the same term to mean different things. This book adopts the following definition: a Grid network "*is a system that coordinates resources that are not subject to centralized control using standard, open, general purpose protocols and interfaces to deliver non-trivial qualities of service*" (Foster, 2002). This definition expresses three concepts that the authors of this book find meaningful:

- Coordinates resources that are not subject to centralized control. This differentiates a Grid network from systems that fall under a local system of management, such as a Cluster or NOS. Grid networks provide a way to manage resources that are governed by different sets of policies or

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

owned by different controlling entities, which may even be interested in accounting for access.

- Control using standard, open, general purpose protocols and interfaces. General purpose protocols and interfaces are used to define Grid network primitives with a level of abstraction that allows them to be implemented on heterogeneous systems. Grid network primitives provide mechanisms for authorization, discovery, and invocation. The allowance for heterogeneous systems is vital for a technology that will be distributed across multiple domains of control that are intrinsically unlikely to use homogenous systems. The importance of these open standards differentiates a Grid network from application specific systems.
- To deliver non-trivial qualities of service. It is advantageous to share resources only if by doing so we can improve the way we work. A Grid is something that must coordinate the resources at its disposal so that the work done with the Grid is better than the work done without the Grid. The actual metrics used must depend on the goals of the particular Grid network but may incorporate such measurements as availability, throughput, reliability, security, or latency.

Under this definition, something that can only work on a LAN would not be a Grid, Cluster management systems for example. Technology that does not coordinate resources to enhance usage of those resources would not be a Grid, the Internet for example. All of the examples discussed in the sections on research projects and early adopters count as Grid networks.

## Grid Network Topology and Scalability

The next section broadly discusses how the topology of a Grid network can change in response to growth by the organizations using the Grid. This section demonstrates various ways that Grid technology can meet scalability requirements, allowing global Grid networks that cross organizational boundaries and even co-operate with other Grids.

### *Intra-Organizational to Inter-Organizational*

Consider an intra-organizational Grid network constructed and owned by a single organization. Put in place to share resources between departments or groups within a single organization, the Grid network would most likely be

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

located on a single site and operate within one ore more interconnected LANs. Grid networks that service a single organization or enterprise must still be scalable. The organization may grow and come to take over multiple sites. The need for extending the Grid network will mirror the organization's own growth.

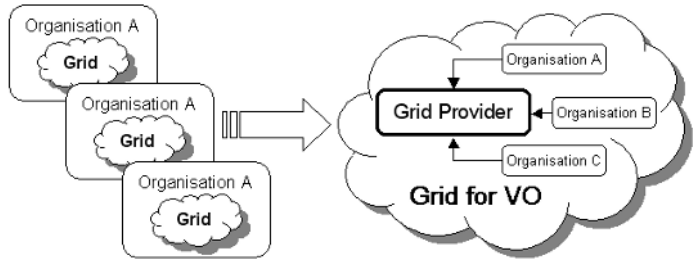
As an organization expands, its need for massive computer-based resources may outstrip its capability to provide for it. The cost to buy and maintain ever-increasing amounts of hardware may become prohibitive. Alternately, the organization might not have an adequate skill base to maintain such a large and ever-growing venture in-house. At this point, the solution could involve joining someone else's Grid network, that is, outsourcing part of their requirements to an external Grid network provider. A single Grid network that consists of participants from different organizations can be called a VO, illustrated in Figure 131. External providers might offer a far more attractive alternative to organizations considering setting up their first Grid network.

This is the kind of economic imperative that will drive Grid networks to grow in scale across organizational boundaries. The Grid will grow from being mostly intra-organizational affairs, shared by departments in an organization, to become inter-organizational business arrangements with Grid network providers supplying resources to members of a VO.

### *Local to Global*

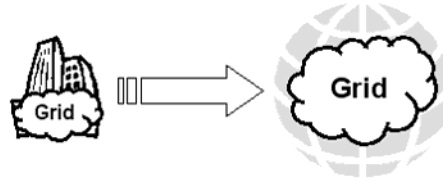
The previous section attempted to show why Grid architectures must encompass the idea of scalability beyond organizational boundaries. As an organization grows, its need for Grid network resources may grow beyond its ability to provide internally, so the organization becomes part of a VO, obtaining certain Grid resources from an external provider. However, some businesses have

*Figure 131. Organizational scalability in the Grid network*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 132. Geographic scalability in the Grid network*




enough financial resources that they can provide for an expanding Grid network to match their own expansion.

When an organization occupies a single building, networks will typically consist of one or more high speed LANs, using Ethernet, or perhaps fibre optic cable. Switches, routers, and hubs will be used to connect the machines together, and one or two machines will form gateways to segregate the organization's network from the Internet. Within the organization's network, it is hoped that Grid access should always be fast and highly reliable.

As an organization grows, it will spread out to take over more sites. Adjoining buildings make it easy for an organization to ensure that shared network resources are just as reliable between sites as they are within sites. However, it is more likely that an organization will spread to a different suburb, state, or even country. As an organization becomes distributed geographically, it is likely to need a Grid network that can still support each site equally. If the organization wishes to expand its own Grid as it expands, a Grid network must be able to scale geographically.

Anyone who has used the Internet knows that even with a high speed connection, accessing some resources can be slow and unreliable. The Internet is an uncontrolled network. If a resource becomes unresponsive or stops working altogether, it is rare indeed that a single phone call to IT support will fix the problem. This does not diminish the need for an organization to scale their Grid network geographically. It only means that the organization must be careful to do all they can to ensure that Grid network coverage in distributed sites is as reliable as possible.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

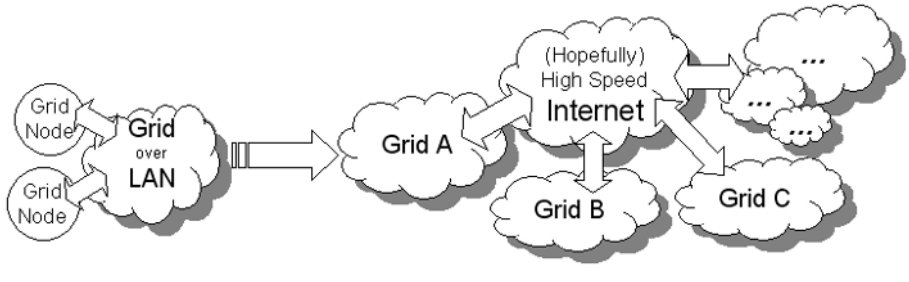
*Intra-Grid to Inter-Grid*

The previous section outlined why Grid architectures must encompass the idea of scalability beyond geographical boundaries. As an organization grows larger, its Grid network can grow with it and still remain the domain of one organization. It was pointed out that unless the new site is literally next door, the Grid would need to scale beyond the LAN, probably to multiple LANs across the Internet. The section on intra-organizational to inter-organizational Grids showed one way to respond to growth by joining a VO that can provide required resources. The section on local to global Grids showed another way of responding by expanding the existing Grid network. This section shows yet another way of responding, this time by having the existing Grid network co-operate with other Grid networks, illustrated in Figure 133.

Many Grid networks specialize in particular areas of technology or service. Grid networks can co-operate even if they belong to different organizations or VOs, as long as appropriate rights are granted. Doubtless, such partnerships will be accounted for in different ways, perhaps through payment or compensatory services. The advantage of this approach is the same as that accrued to any business partnership, which is that each partner relies on the other to provide some capability that could not otherwise be provided.

At first glance, there is not much difference between topology shown in the sections on Intra-Organizational to Inter-Organizational and Intra-Grid to Inter-Grid. The distinction is revealed when you consider the first part of the definition given in the section titled "What is a Grid Network?", that is, a Grid network coordinates resources for itself. Two Grid networks co-operating may be able to submit jobs for each other, but they would not normally make coordination decisions for each other. There is a coherent boundary of control separating Grid networks.

*Figure 133. Inter-Grid scalability in the Grid network*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

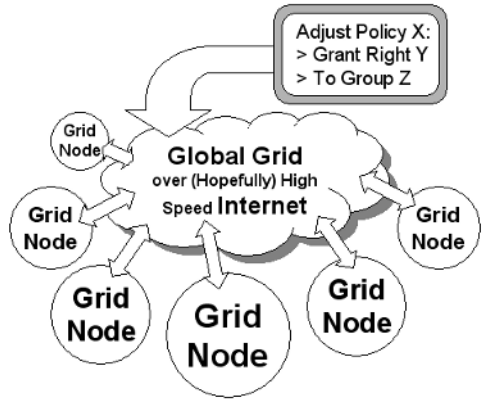
## Visualization

Grid networks should be able to scale without having to change the structures used to manage or visualize resources on the Grid network. Tools to manage Grid resources can be thought of as visualizations into the pools of resources they manage. If such visualizations are aggregated, they can offer control over larger groups of resources. In such a fashion, schemes can be developed to scale the Grid from local to global without changing the structure presented by the visualization. Figure 134 shows conceptually that a visualization aggregates management over the resources in a Grid so that decisions can be made about how those resources are used.

In Figure 134, a management application provides some kind of visualization into resources on the Grid and allows decisions to be made remotely about how to manage resources on the Grid. With an appropriate scheme, visualizations can cover multiple Grid networks, creating a federation of Grid networks, illustrated in Figure 135.

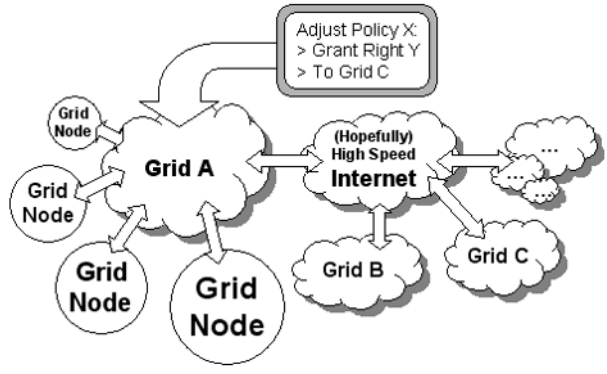
Figure 135 conceptually shows a federation of Grid networks being managed by a visualization tool. The network is split into clusters or sub-networks of machines, where each cluster is a LAN or set of LANs with similar management and administration policies. Grid nodes are computers connected to the network that run whatever Grid client software is required. Nodes within the same cluster will typically communicate through a high-speed network. Nodes in different clusters will communicate through high-speed networks, though this may not always be the case.

*Figure 134. Management of a Grid network through visualization*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 135. Management of a federation of Grid networks through visualization*




Note that "cluster" is not used here to represent a cluster of tightly coupled machines of similar types with dedicated network connections, as in the traditional idea of cluster computing. It is instead meant to represent a large group of computers on a single LAN or collection of LANs that share a common network boundary and are likely to have fast network connections.

The powers of a visualization tool would be curbed by the system of policies and permissions within each Grid network, for example the decisions one Grid can make about another Grid would be a subset of those decisions the Grid could make about itself. However, the visualization should be the same.

# OGSA: Web Services as a Grid Platform

This section shall discuss the approach taken by the defacto standard in Grid computing platforms, Open Grid Services Architecture (OGSA) as implemented by the Globus Toolkit. It shall begin in the section on service-oriented architecture with a discussion of how Service Oriented Architecture is used to create a Grid Web Services framework. The section on OGSA, OGSI and WSRF: from GT3 to GT4 shall examine the current Globus standards, OGSI in GT3 and WSRF in GT4, and the section on the differences between Web

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Services and Grid Web Services will compare them with Web Services in general. Then we shall examine some of the core components that form Globus' Grid Network framework: state management, interface inheritance, lifecycle management, notification, service groups, and addressing. Finally, the sections on anatomy of a WSDL file and anatomy of a GWSDL file shall present an overview of the WSDL and GWSDL formats.

## Service-Oriented Architecture

This section will discuss how general principles behind service-oriented architectures (SOA) are reflected in Web Services in general and OGSA specifically. The approach to Grid networking described here chooses to view a Grid network as a set of services being shared rather than a set of physical resources being shared, for example OGSA is a service-oriented architecture. A service is defined as being a network-enabled entity that provides some set of capabilities, that is, provides a service, through a well-defined interface. The service is invoked through a series of message exchanges between the service and client. This is not the same concept as an object in Object Oriented Programming (OOP) parlance, though it will be seen that there are many similarities.

A virtual organization (VO) is loosely defined as the collection of individuals and organizations that form the community around which a Grid network is centered. Different entities in the VO may perform different functions: resource providers, consumers, brokers, or anything in between. The individuals and organizations that make up a VO will generally be very different from each other. They will have vastly different purposes, structures, cultures, and practices. They are, however, brought together in order to fulfil their roles as part of the Grid network community. Since Grid networks are always formed for a relatively specific purpose, this gives them much in common despite their differences.

It is possible to identify a broad set of technological requirements that all those participating in a VO have in common. These requirements are compelling reasons to become part of a Grid network. The rest of this section will examine the nature of these requirements and the mechanisms available to support them.

### *Interoperability*

A critical requirement for any Grid network is interoperability across the heterogeneous resources offered by the disparate parties within a VO. For the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

SOA described here, interoperability is achieved by creating a layer of common abstraction over all implementations. The abstraction layer consists of two components, listed next.

- Interfaces to model services offered by the Grid network.
- Protocols needed to discover and invoke implementations of the services.

The interface defines the capabilities of a service, but says nothing about what protocols should be used to invoke the service, for example should it? A *binding* links particular implementation of a service with a specific protocol or set of protocols that can be used to invoke the service.

*SOA Requirements*

In choosing an SOA for implementation of a Grid technology, other requirements can be identified. The major ones are discussed next.

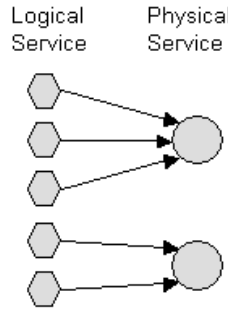
- The interfaces definition framework and protocols chosen will form a set of standards that must be widely accepted for the SOA to be widely accepted. OGSA rests on Web Services. The standards chosen are WSRF, OGSI, and WS.
- The architecture must embed the same sorts of transparencies already discussed in the section on Grid goals. This includes, most importantly, local/remote transparency.
- The SOA must adapt to local OS services, meaning that interfaces must be implemented for specific operating systems. In other words, the Grid framework must be ported to the different OSs on which it will be hosted.
- The SOA must be designed in such a way that there are clearly defined uniform service semantics, that is, the same functionality on different services should fulfill the same contract. This is achieved primarily by making sure the contract of each interface is communicated clearly through documentation.

*Virtualization*

OGSA defines a Grid network of *virtualized* services defined by interfaces. In this definition, virtualization means the encapsulation of a wide variety of implementations behind common service interfaces.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 136. Mapping of logical to physical services*




There are distinct advantages to virtualization. Common semantics through shared interfaces implies consistent access to resources across heterogenous platforms. All services act in the same way, whether they are local or remote. Multiple logical resources or references to resources can be mapped to the same physical resource, shown in Figure 136, meaning that several clients can share a reference to the same remote service.

Composition of services is another major advantage to the virtualization of services. Without having to know how a service is implemented, it can be incorporated into a larger service. Further, management of these compositions should be as easy as management of the individual services, since all services share the same basic semantics, whether it is a composite service or not. The assumption here is that all services will implement and expose common management semantics.

*Service Semantics*

This section briefly discusses the importance of service semantics in an SOA. The semantics of an interface defines the meaning that an interface has within some context; further, the semantics of each function or action or capability within the interface defines the meaning of that action, function, or capability within the context of the interface. For example, the semantics of a CurrencyConverter interface would be that the service knows about different types of currency and how to convert sums between them. The semantics of an action called convert within that interface is that the action would perform currency conversion.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The semantics of an interface should always be given without stating precisely how that effect will be implemented. For example, the CurrencyConverter interface should not indicate whether the service refers to an Access Database, a local file system, or a flat file on a particular remote server.

The choice of binding should not affect service semantics, but this does not prevent a specific binding from being chosen for other reasons. For example, to fulfill a QoS metric, HTTP might be chosen over FTP.

*Stateful Services*

In an SOA, a client communicates with a service through an interface. Interfaces define the capabilities of the service and have no intrinsic notion of state. However, an interface most often implies the existence of some state, which is manipulated by the service each time a message exchange occurs. This section will outline three general relationships that may exist between interface and state: the stateless service, the conversational service, and the service that acts upon a stateful resource (Foster, 2004).

The stateless service does not store data and only uses information contained within input messages. An example of a stateless service would be an image processor that receives an image, manipulates it in some way, and returns the end result without saving it for future reference. The service acts upon data stored in the messages and responds without storing any state data. Problems arise when the amount of state data becomes unwieldy or when the service needs to remember data from the message exchanges.

The conversational service relies on message exchanges occurring in a logical order to determine what processing should occur. The result of an operation depends on a previous operation and can affect subsequent operations. Some Web applications implement conversational state by storing data in HTTP sessions or cookies. The service interprets data in the session or cookies to work out what processing should occur next, as opposed to the client invoking a particular operation. It is argued that conversational services are used by techniques such as WS-Coordination, WS-Context, and WS-Policy, as they are based upon propagation of execution context in message headers (Foster, 2004).

A service that acts upon stateful resources is a service whose implementation manipulates some form of stateful resource in response to the message exchanges between itself and the service invoker. The service is stateful with respect to the client but may be implemented in a stateless manner, as we will see in the next section.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Stateless Implementations, Stateful Interfaces*

The property of statelessness is seen as a good thing because it can increase the scalability and robustness of a system (Foster, 2004). A stateless service can be restarted without having to worry about the history of prior actions. Additionally, it is easy to create and destroy stateless services, perhaps in response to changing loads. Statelessness has advantages for distributed systems that need to offer properties amenable to active load balance algorithms, which is a common requirement for Grid computing platforms. This section demonstrates how an SOA can maintain a stateless interface and a stateful implementation. It also shows how Web Services use explicit state, while Grid Web Services use implicit state.

Some services propagate state explicitly. In a service-oriented architecture, a client typically expects to communicate with a persistent service that exposes a given interface, where the interface has no inbuilt notion of state. Indeed, Web Services are often used to represent persistent utility functions where multiple invocations of the Web service are seen as individual transactions, independent of each other. This does not mean that Web Services are stateless, only that the notion of state is not a feature of the architecture. It has to be implemented by Web service programmers.

Web Services (and many Web applications) propagate state explicitly. The server uses some form of session variable and/or has a backend data store. Requests from the client include an identifying token that the server can use as a reference to look up stored data about the client.

In a Web service, a service requestor usually interacts with the state of their message exchange through indirect reference that will most commonly take the form of a record identifier that is passed to and from service and service requestor with each message. The mechanism for implementing this exchange is built into the application and not the Web Services architecture. The application developer must ensure that each message exchange between service and service requestor propagates sufficient information that the appropriate message exchange state can be looked up in a data store.

In the case study presented later in this chapter, it can be seen from Figure 143 that an order identification must be passed with each message. The service implementation uses that identifier to look up data for the order. In this way, the service is seen by the service requestor to be stateful, but the service is actually stateless since the Web service component itself does not store data. The storage of data is delegated to another component, the data store, which can be remote.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Some services propagate state implicitly. From the perspective of the service requestor, a service is often, for all intents and purposes, an instance of the service with state, that is, an object of some type, perhaps instantiated at the behest of the requestor itself. This pattern is most obvious in distributed object systems such as CORBA, EJB, or RMI. It is also seen in OGSA Grid Web Services under both GT3 and GT4. From the point of view of a service requestor, each instance of the server application holds its own state and typically interacts with just one client or a logical client group of clients.

Central to distributed object patterns are the notions of object state and reference (Vogels, 2003). The reference is the handle by which an object client can refer to the object. It is what a client uses to access the object, invoke operations on it, or examine properties of it. The reference is an item of data that the client can pass around to different processes, ensuring that the object can be uniquely accessed from multiple points within a system. The reference can also be used to determine equality between instances of the same type; the implication of this statement is that a reference is the means by which an object-client can represent the identity of the object to which it is a client.

The notion of state is important to an object, because it is the ability of object instances to hold their own distinct states that separates the notion of having multiple object references — to objects with different states — from having multiple service references, all to a service with a single state. It is generally understood that a service is much like a singleton, a single point of reference to a set of functions and not to be considered an object in its own right.

From the service requestor's points of view, they interact with the state of their message exchange, but state is implied through reference to an instance of the service rather than explicitly propagating state through some identifying token. Further, the implicit recognition of state through reference to a service instance as seen in Grid Web Services is a framework level construct. Identification of state is made at the architectural level and achieved through the URL or GSR under GT3. In GT4, state identification occurs within the messages through WS-Addressing. In both cases, it is handled as part of the Grid Web Services architecture and is not application specific, meaning the application programmer never has to code their own identification mechanisms into the system. Passing an identifier back and forth in Web Services occurs at the application level, implemented by the programmer. The distinct advantage here is that it is a standardized way of handling state references and that an application does not have to implement this feature for itself.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

In the case study presented later in this chapter, it can be seen from Figure 144 that no order id must be passed with each message. In OGSA Grid Web Services, the services are made stateful because OGSA has overlaid portions of a distributed object pattern on top of Web Services. Service requestors access an instance of the service they require, the implication being that each instance is an "object" that provides state and behavior.

In GT3, the service requestor invokes operations directly on the instance of the Web service that it has reference to from the creation operation. Creation occurs when a requestor accesses a Grid service factory and requests an instance of the required service. The reference is a GSH that is discussed in the section on addressing. Structurally, the GT3 Grid Web service is made stateful by the appearance of architecture features such as SDEs.

GT4 Grid Web service interfaces are stateful, but the implementations are stateless. This has come about because GT4 conforms to the WSDL Web service standards, which have no concept of state. GT4 Web service implementations are stateless because the Grid Web service component delegates responsibility for state management to a WS-Resource.

Importantly, client requests never need to contain session or user IDs because the server instance assumes all communications interact directly with its current state.

## OGSA, OGSI, and WSRF: From GT3 to GT4

The Open Grid Services Architecture (OGSA) is the current defacto architecture for most Grid networks. Web Services under WSDL is the OGSA middleware platform, making OGSA a highly interoperable SOA. With the Globus Toolkit 3 (GT3), Globus implemented OGSA with the Open Grid Services Infrastructure (OGSI), which did not converge with WSDL standards as expected. In 2004, the Web Services Resource Framework (WSRF) was announced, which improves upon and will replace OGSI. Additionally, WSRF is fully compliant with the current WSDL standard (1.1), making OGSA Grid networks even more interoperable. The next version of the Globus Toolkit (GT4) will be released in 2005 and will be the first full implementation of WSRF.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Differences between Web Services and Grid Web Services

This section discusses the distinctions between Grid Web Services and Web Services, revealing how object oriented design (OOD) and SOA merge to give Globus' Grid Web Services. Under OGSI and GT3, Grid Web Services are a subclass of Web Services, with the differences made apparent by the use of a different description language: GT3 Grid Web services use Grid Web Services Definition Language (GWSDL) instead of WSDL. Under WSRF and GT4, Grid Web Services may still be thought of as a subclass of Web Services, but they use the same definition language. WSRF has moved the distinguishing features out of the interface definition language.

Grid Services enhances Web Services primarily through the introduction of state as a standard feature of the framework. The OGSI model of Grid services developed by the Global Grid Forum (GGF, http://www.globalgridforum.org) extended WSDL, adding two key requirements (Joseph, 2003):

- **The ability to describe interface inheritance:** this is a most powerful technique for the propagation of base Grid Web Service semantics, since a service can declare itself to inherit from one or more base interfaces.
- **The ability to define service data elements attached to the interface:** this approach involves a mechanism for all Grid Web Services to expose state data to requestors for query, update, and change notification (Foster, 2002).

The technologies that GT3 services are built on (OGSA, OGSI, and GT3) are based on standard Web Services technologies such as SOAP and WSDL. However, the two features identified previously are required by Grid service interfaces and not supported by the current version of the Web service interface language (WSDL 1.1). These features were to be made available in WSDL 1.2/2.0, but at the time of GT3's creation, WSDL 1.2/2.0 was a W3C Working Draft only. It was not a stable standard and was likely to change before it would be accepted. As a temporary measure, GGF created GWSDL as a stand in for WSDL 1.2/2.0 and intended on replacing GWSDL with WSDL 1.2/2.0 when it was accepted (Sotomayor, 2004).

This changed with the announcement of WSRF and GT4, which maintains the two key requirements noted earlier but moves their application into other parts

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

of the Grid Web service design. This removes the need for a specialized definition language, allowing Grid Web Services to be defined by WSDL 1.1.

## State

As already discussed in the section on stateless implementations, stateful interfaces, Grid Web Services ads a measure of OO philosophy to the Web Services paradigm. OGSI and WSRF add these object-oriented principles to the Web Service based SOA through the addition of the service data elements and interface inheritance as mentioned in the section on differences between Web Services and Grid Web Services. This section shall discuss in greater depth how the addition of state is achieved.

In distributed object architecture, a client typically expects to communicate with a single object representing an instance of an interface with attached state. Grid Web Services are stateful by design, meaning a Grid Web service could be expected to remember what operation it just performed and change its action the next time it is invoked, based on previous invocations. Additionally, a Grid Web service can be transient while its state is persistent. This means that a Grid Service might die at a specified time and be re-created as needed — with state intact — as part of a factory/instance pattern (Sotomayor, 2004).

A resource must provide data about itself in order for some controller or user process to make decisions about the resource based on its current state. Such data, used to describe the service, can allow service querying and discovery. State data for a resource may be static or dynamic. Static data never changes; dynamic data may change, frequently or infrequently. This data might allow a management process to decide whether the resource is available based upon current load. It might allow a user process to decide whether the resource is suitable for its purpose based upon some QoS metric (which could also be current load). This section first examines the difference between metadata and state data, and then outlines how state is represented in GT3 and GT4 respectively. Finally, a case study is presented that emphasizes how different styles of state can be implemented in Web Services and Grid Web Services.

The type of data made available depends on what sort of resource it is. A "resource" may be physical or electronic. Physical resources include hardware attached to the computer or even the computer itself (represented by features such as current CPU utilization or storage capacity). Electronic resources are software applications running on the computer. Importantly, physical resources

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

are represented to the Grid through software interfaces. The implication is that all resources on the Grid are software resources, some of which manipulate physical resources. In this context, it makes no difference to a management or user process whether the data is gathered from a physical resource or a purely electronic resource; the relationship is not symmetrical.

*Metadata and State Data*

In general, service data can be metadata or state data.

Metadata is information about the service, as opposed to information directly generated by the service. In a Grid network, metadata would be data that relates to the resource as a Grid service. Metadata is most likely to contain static data, data that never change throughout the service's lifetime. Metadata might include system data, the set of interfaces supported by the service, static cost of using the service, and so forth.

State data is application specific data. It is provided or generated by the service and describes the state of the service. State data relates to the resource's inherent functionality rather than describing the service as metadata does. Application specific data is most likely to contain dynamic data. State changes in a service will most likely occur through changes in application specific data. State data might include runtime data such as the current CPU load or the results of service operations or perhaps dynamic service data, such as the current cost of service, duration, amount of services provided, resulting state, continuation data, data about the actual use of resources, and error conditions.

*State in GT3*

In an OGSI Grid, a structured collection of information associated with a Grid service is called Service Data (Sotomayor, 2004). Service data can be used to classify and index services to allow the querying of service state and discovery of services that match particular service requirements. Service data includes metadata and state data. It can be composed of primitive data types or complex data types. Service data are expressed as individual Service Data Elements (SDE).

Consider the following simple example of a printer service that might require service data (Figure 137). A client should be able to query the available printer services to find out which one is suitable for their needs. We should also be able to use the service data to categorize and index the printer services as Grid services.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The scenario is easy enough: there are various services available with different characteristics that can be represented as service data. The client can query the service data attached to each service and decide which service is most appropriate for the situation. SDEs are linked to a portType (i.e., the Grid service) in a GWSDL file. Each Grid service can have any number of SDEs. Each SDE has its own complex data type defined in XML schema, which means it is defined by the data types of the values it contains and can hold any number of values of different types.

Service data is the architectural mechanism by which a GT3 Grid Web service exposes metadata and state data. Under GT3, a GWSDL can define a wsdl:types element that describes a set of structured data. The Web service architecture allows services to expose operations in their WSDL interface. This extension allows Grid Web Services to expose attributes as well, though the attributes may represent metadata rather than state data.

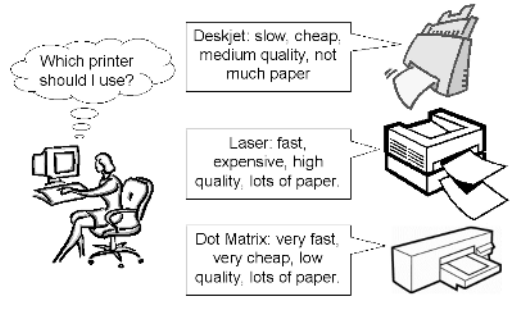
An example wsdl:types element is shown in Figure 138, which might be used for a state data in an image processing service.

*WS-Resource — Service Data in GT4*

In GT4 Grid, Web Services apply a WS-Resource approach (Foster, 2004) to statefulness. The Web service implementation is stateless, but the interface appears stateful to the service requestor. This is achieved architecturally by storing state in a separate entity called a resource, also knows as *stateful resource,* that can be expressed as an XML document and associating it with a Web Service through WSRF mechanisms described in the WS-ResourceProperties specification.

A stateful resource is a structured data entity that stores state data that is associated with and can be manipulated by a Grid Web service. A stateful

*Figure 137. Simple example of service data*




Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 138. Example wsdl:types element*

```
<wsdl:types>
<schema targetNamespace="http://www.example.com/imageProcessor"
  attributeFormDefault="qualified"
  elementFormDefault="qualified"
  xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="ImageDataType">
    <sequence>
      <element name="encoding" type="string"/>
      <element name="maxHeight" type="int"/>
      <element name="maxWidth" type="int"/>
      <element name="maxResolution" type="int"/>
    </sequence>
  </complexType>
</schema>
</wsdl:types>
```

resource must be expressible as an XML document and have a well-defined lifecycle. A WS-Resource is a composition formed by a Web service description being associated with a stateful resource.

WS-Resource's *resource property document* is an XML Global Element Declaration (GED) in a given namespace that defines the type of a stateful resource. The resource is often a compound element, composed of other elements, which are referred to as resource properties. In general, a service requestor does not directly interact with the stateful resource. Instead, a service requestor would begin by accessing the WE-Resource's type by obtaining Web service's WSDL portType. It would then manipulate the Web service through the defined message exchanges, which are in turn expected to read, modify, and query the XML document that represents the stateful resource.

The resource properties document is not usually the actual WS-Resource state in and of itself. It is usually just a representation of the state, also known as a view of or projection on the real state. The document places an XML structure on top of the state that can be used by the service in update and query operations. The service's programmer must make sure that any operation that manipulates the WS-Resource's state through the resource properties document must propagate those state changes to the actual software component that stores the state data.

Stateful resources may be instantiated by a Web service referred to as a stateful resource factory. The same service must assign an identity to the entity, and the identity can have aliases assigned to it by applications using the resource. A stateful resource identity may be passed around to different Web Services.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

When a service requestor needs stateful interaction with a Web service, it must specify the identity of the stateful resource to the Grid Web service. The stateful resource identifier is a reference to a resource and is used by WS-Addressing in GT4. See the section on addressing for more detail.

It is worthwhile noting here that a Grid Web service will have at least three components:

- The interface in WSDL and most probably duplicated in a specific programming language. This is a stateless description of the operations offered by the Grid Web service.
- The resource home, through which resources are discovered and removed.
- The resource itself, which is the stateful data store for a Web service.

The following outlines the steps necessary to define a stateful resource and associate it with a Grid Web service under GT4.

First, create a definition of the Web service resource data type in a wsdl:types element that can be included in the Web Services wsdl definitions element. This can be a compound element, which may even be defined in an external document and imported into the wsdl document. An example wsdl:types element is shown in Figure 139, which might be used for an image processing service.

*Figure 139. wsdl:types element for image processor service*

```
<types>
<xsd:schema targetNamespace="http://www.example.com/imageProcessor"
   xmlns:tns="http://www.example.com/imageProcessor"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="ImageProcessorProperties">
  <xsd:complexType>
    <xsd:sequence>
      <element name="encoding" type="string"/>
      <element name="maxHeight" type="int"/>
      <element name="maxWidth" type="int"/>
      <element name="maxResolution" type="int"/>
    </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
</xsd:schema>
</types>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 140. portType element for image processor service*

```
<portType name="MathPortType"
   wsdlpp:extends="wsrpw:GetResourceProperty wsrlw:ImmediateResourceTermination"
   wsrp:ResourceProperties="tns:ImageProcessorProperties">
  ...
</portType>
```

Next, link it to the wsdl:portType element with a wsrp:ResourceProperties property (Figure 140).

*Case Study in State*

**Service Component in a Book Ordering System**

Consider a service side component within a book ordering system. The component is designed to accept and validate book orders. Other components will handle tasks such as extracting payment, printing an invoice, physically fulfilling the order, and so forth. In a standard MVC pattern, this component is part of the model; data is obtained from a view and given to the component by the controller to validate and act upon appropriately if valid. Implemented as a Web service, this component does not necessarily have any knowledge of the presentation or controller components.

The data accepted by the component can be logically divided into three parts: details of the book or books to be ordered; postal details for delivery of the order; and finally payment details. If the component determines that all details are valid, it should perform some action to place the order, that is, trigger payment routines and a process to ensure the order will be physically shipped. The component could delegate to a controller component, which might persist the data and trigger the next component in the system. Figure 141 shows a possible state diagram for the validation component.

The component's behavior depends on the state of the data given to it: if the data is valid, the book can be ordered, and if the data is not valid, the book cannot be ordered. The range of acceptable values for each data set is decided upon by the business rules of the application and are described in Table 1.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 141. State diagram for component of book ordering system*

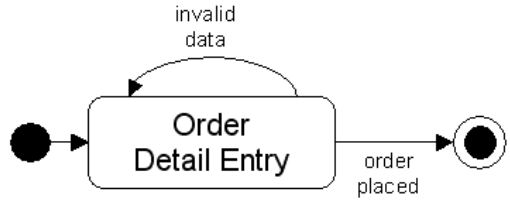



*Table 6. Valid and invalid range of data values for each state of book ordering data entry.*

| | Valid Range | Invalid Range |
|---|---|---|
| Book Detail Entry | Data describes a book in stock. | Data is incomplete or describes a book that is not in stock. |
| Postal Detail Entry | Data describes a person and address that can be shipped to. | Data incomplete or describes a region that cannot be shipped to. |
| Payment Detail Entry | Data describes an existing account. | Data incomplete or describes an account that does not exist. |

## Stateless Web Service

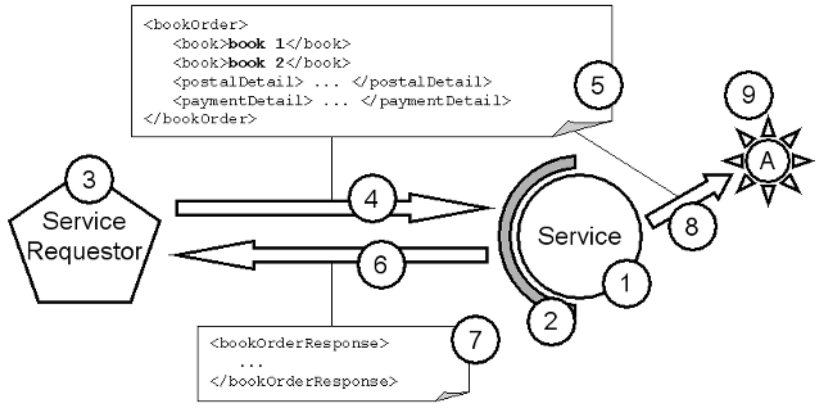
The interaction diagram (Figure 142) shows possible interactions for the first version of the component if it were to be implemented as a stateless Web service. The service requestor in the diagram could be the presentation component or another component capable of talking to the Web service.

A Web service component (labeled 1 in the figure) exposes an interface (2) to service requestors (3). Service requestors invoke an operation on the service (4), with a message that includes the order detail (5). The service determines if the data is valid and replies with a message (6) that indicates whether the order was placed or if not, why not (7). If the data was invalid, the service requestor can take appropriate action and submit the updated data again as needed. If the data is valid, the service can send the valid order detail (8) to a controller component (9), which can continue processing the order.

The service is stateless because it implements message exchanges that do not access or use data not contained in the input message. The component does not need to store any data in this version; the whole order is given to it each time.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 142. Component as a stateless Web service*




## Stateful Web Service and Grid Web Service

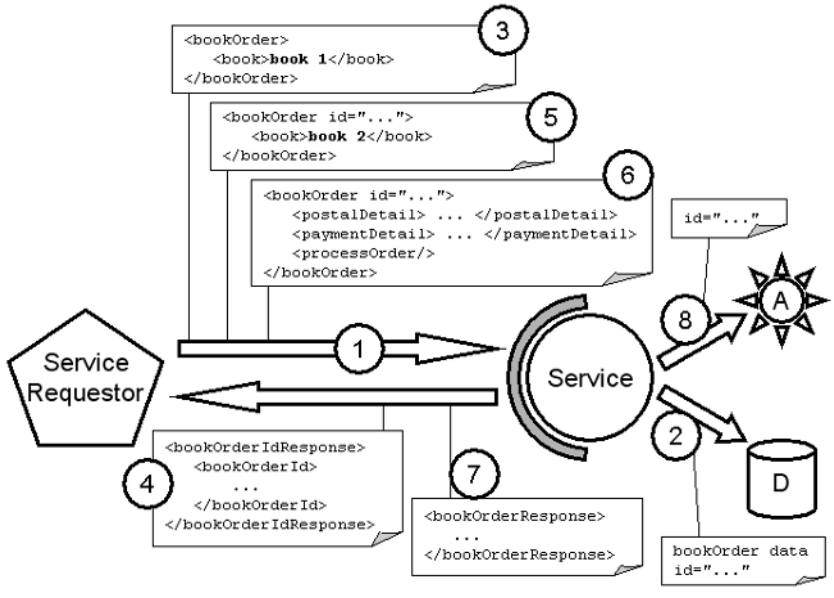
Many applications involve state data that is too large or too complex to be communicated entirely within the XML messages between service requestor and the service. It may even be that the time delay between gaining certain pieces of data is long enough to make it worth storing parts of the data in between requests. In this section, we look at two patterns for implementing stateful Web Services. At least from the point of view of a service requestor, the Web service interface is stateful. The implementations will be stateless, though they will act upon stateful resources (Foster, 2004).

Figure 143 shows an interaction diagram for the second version of the component, now as a stateful Web service. This might be the kind of service that appears as the backend to a book-ordering Web site, for example. Users browse or search a collection of books, data that can be gained from some other component. When the user finds a book he or she wishes to order, the user presses a button to add it to the shopping cart, sending the choice to the Web service to be remembered. When the user has added all the books he or she wants, the user presses a button to proceed to the checkout where he or she can view the order in its entirety, fill out postage and payment details, and finally place the order.

The message schema should define all items of data required for a complete book order. Not all parts of the message are required at once, but all data for the order must be available to the component before the order can be

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 143. Component as a stateful Web service interface with a stateless implementation*




processed. A service requestor can send (1) incomplete messages to the component at different times, and the component will store each item of data against a generated identifier for the order (2). Whenever the component receives a message that does not contain an order identifier (3), the component assumes the message is the first of a sequence for a new order. It can generate an identifier and send a message containing the ID (4) back to the requestor. The service requestor should ensure that it includes that identifier in future messages so that the data is added to the correct order. The user can keep adding (or removing) books (5) to the order and finally goes to the checkout where they fill in their postal and payment details and finalize placement of the order (6). Possibly (6) will be implemented as two or more messages, sending the processOrder element in a separate message.

Although individual messages to the service may not contain details for the entire order, each message must still represent a complete logical communication that can be validated and responded to. The service will wish to check

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

book, postage, and payment details according to the rules presented in Table 6 before adding them to the data store and responding appropriately with a message (7) that indicates whether an individual item of data was valid and if not, why not.

Finally, when the service receives a message (6) that indicates the service requestor wishes the order to be placed, the component need only send the controller an order identifier (8). The controller can use the identifier to look up all details of the order that have been saved to the data store by the component.

Figure 143 shows a Web service that is stateless. The Web service does not store data; it delegates the handling of state to a different component, that is, a data store. Dynamic state needed for the message exchange is provided indirectly by reference within each message in the form of an order identifier. The effect is that even though the service implementation is stateless, the interface is stateful because all requestors use the interface to interact directly with a state belonging to a specific order.

It is important to understand that this is not a standardized approach within the Web services architecture. Mechanisms for identifying state within messages are totally application specific and are not a native aspect of Web Services architecture (Christensen, Curbera, Meredith, & Weerawarana, 2001; Tuecke, Czajkowski, Foster, Frey, Graham, Kesselman, Maquire, Sandholm, Snelling, & Vanderbilt, 2003).

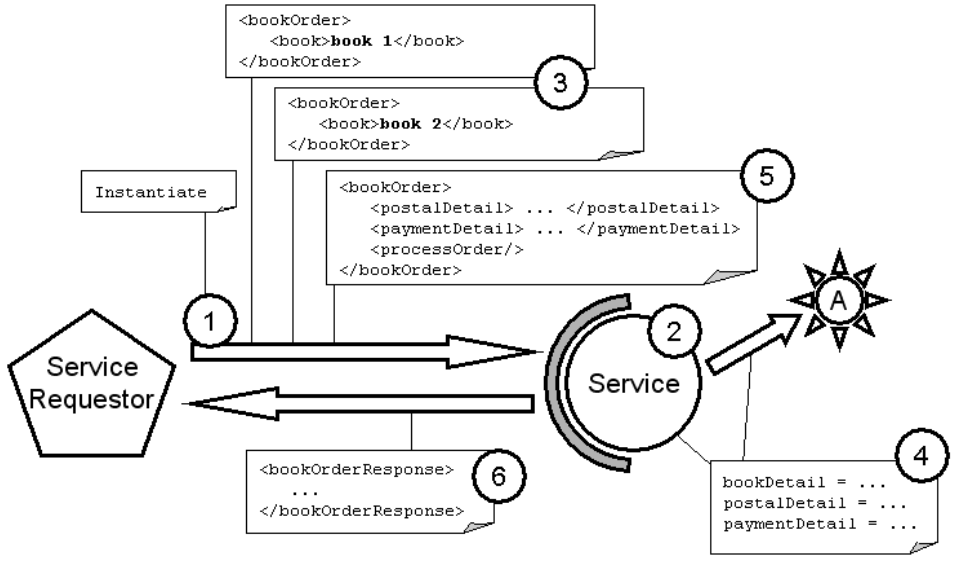
Figure 144 shows possible interactions for the third version of the component, now as a stateful *Grid* Web service. This diagram is general enough that the pattern applies to Grid Web Services under GT3 or GT4. The pattern of user involvement is the same as that shown in Figure 143.

What differentiates the interaction diagram in Figure 144 from the one shown in Figure 143 is that the latter can be characterized by many of the distributed object (Chin & Chanson, 1991) semantics of an OGSA Grid Web service (Foster, 2002) under GT3 or GT4. First, the service is instantiated (1). Under GT3, this involves generating a service instance from a factory (Sotomayor, 2004). Under GT4, this involves creating a stateful resource and linking it to a Web service, an operation which returns a WS-Resource (Foster, 2004). To the service requestor, the component it references is an instance of the Web service state with which it wishes to interact (2).

Again, the message schema should define all items of data required for a complete book order but does not need to have an order identifier. A service

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 144. Component as a stateful Grid Web service interface with a stateless implementation*




requestor sends incomplete messages (3) to the component at different times, and the component will ensure each message is reflected against its state (4). Multiple messages can be accepted for book details (3), adding, or perhaps removing books from the basket, postal, and payment details (5). Possibly (5) will be implemented as two or more messages, sending the processOrder element in a separate message.

Just as before, each message must still represent a complete logical communication that can be validated and responded to. The service will wish to check book, postage, and payment details according to the rules presented in Table 6 before adding them to the data store and responding appropriately with a message (6) that indicates whether an individual item of data was valid and if not, why not.

When the service requestor finally sends a message indicating the user wishes to finalize placement of the order (5), the component sends reference to the state to the controller.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

It should be noted that the Grid Web service is actually propagating an identifier in each message. Under GT3, the identifier is embedded within the URL or GSR. In GT4, the identifier appears within the messages in the format specified by WS-Addressing. However, this is handled as part of the Grid Web Services architecture and is not application specific. The distinct advantage here is that it is a standardized way of handling state references and that an application does not have to implement this feature for itself.

### Final Comments

The scenario is incomplete in two areas of particular note. Firstly, the bookOrder message should have a way of indicating to the component that the service requestor wishes to see pertinent details of the order so far. This could allow the service requestor to show the user all details of the order so far and would certainly be a requirement to allow the user to confirm the order. The bookOrderResponse to such a message may include a complete listing of all books added to the shopping cart so far plus any postal and payment details.

Secondly, it is likely that the system would wish to persist postal details, payment details, and an order history between orders. This way, a user would not have to re-enter postal and payment details after their first order, and they could see which books they had previously ordered. Such details could be made available to the component after a successful log in request. To accommodate this, the bookOrder message schema could include a username/password combination and allow postalDetail and paymentDetail to include identifiers that can be used to look up those details.

It is of course possible to use Grid Web Services in a static sense that is, without ever having to create or manipulate an instance representation of the Grid service with associated state. There are some situations where this is an appropriate design choice. However, many designs find powerful use for object-oriented concepts and should take full advantage of Grid computing patterns accordingly.

## Interface Inheritance

GT3 allows a Grid Web service interface (GWSDL) to specify that it extends one or more interfaces. This is how base Grid Web service semantics are propagated to new Grid Web Services. For example, see Figure 145.

Unfortunately, this is one of the reasons GWSDL had to be created as a specification incompatible with WSDL, which does not provide for interface

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 145. GWSDL interface inheritance*

```
<gwsdl:portType name="ExampleService" extends="ogsi:GridService">
  ...
</gwsdl:portType>
```

*Figure 146. GT4 interface inheritance*

```
<portType name="MathPortType"
   wsdlpp:extends="wsrlw:ImmediateResourceTermination"
</portType>
```

inheritance. Grid Web Services still manifested as Web Services by having a tool that would process GWSDL, remove the pertinent non-WSDL, and copy and paste the inherited interfaces into the new one in order to produce a valid WSDL.

GT4 does much the same thing, but with a notation to the WSDL, so that the interface for a GT4 Grid Web Service is still valid WSDL. For example, see Figure 146.

The wsdlpp namespace refers to the WSDL Preprocessor that in GT4 takes interfaces to be extended and copies and pastes them into the resulting WSDL.

## Lifecycle Management

Grid Web Services have non-trivial lifecycles since they can be long-lived or transient. It is important to have in place mechanisms to manage not just instantiation, monitoring, and the controlled destruction of Grid Web Service instances, but also their persistence and recovery in cases where the service can outlive the client who requested it and even the server that contains it.

In GT3, creation of transient Grid Web Services will usually be at the behest of a client and delegated to a factory. When the service has finished its job, the client either explicitly releases the instance, or the runtime system detects that the instance is no longer being referenced and reclaims the resources through a process of garbage collection.

GT3 provides callback methods in the Grid Web Service interface to allow control of the lifecycle, specifying actions that need to occur pre and post creation, whenever the service is activated or deactivated and just before the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

service is destroyed. Callback methods can be encapsulated in a lifecycle monitor object for re-use between Grid Web Services.

In GT4, the same functionality is provided by the WS-ResourceLifecycle specification.

## Notification

Notification is a popular design pattern also known as the Observer/Observable pattern and used in the Model-View-Controller pattern. Very simply, an *observer* entity asks an *observable* entity to *notify* it when some change occurs that the observer is interested in. GT3 calls the observer entity a notification sink and the observable entity a notification source.

There are two broad styles of notification: pull notification or push notification. Pull notification involves the notification source entity informing the notification sink that a particular type of change has occurred but not including data pertinent to the change. Instead, it is up to the notification source to make a call back to the notification sink and pull that data down. This approach makes sense when notification sinks require different data from the notification source in response to the same type of change. Push notification involves the notification source entity including data pertinent to the change as part of the notification. The notification sink has less control over the data that is pushed to them, but the implementation is simpler than having to make a second call.

Notification in GT3 is strongly related to service data because notification sinks will ask to be informed of changes that occur to specific SDEs on the notification source entity. Through portType extension, a GT3 Grid Web Service inherits from the ogsi:NotificationSource interface, which provides default subscription and notification methods.

GT4 achieves notifications through the WS-Notification specification, which is not actually part of the WSRF but has been adopted for notifications. It provides the ability for a Web service to be configured as a notification producer and certain clients to be notification consumers (or subscribers).

Polling is an even simpler technique for one entity to stay informed about the state of another entity. Polling involves the observer entity making regular requests to the observable entity, asking about its state. There are clear disadvantages to this approach. If the time between polls is very small, network traffic increases, as will the CPU time taken by the observer asking questions and the observable answering them. Multiple observers for the same observable exacerbate this problem further.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

## Service Groups

Aggregation of Grid Web Services is achieved in GT3 through Service Groups. Any Grid Web service can become a service group and implement methods to allow services to be added or removed and existing services to be searched for given appropriate search criteria. Service groups are used as the base for Grid Web service directory services in GT3 — such as the IndexService — because they provide a single point of entry to a collection of services.

In GT4, the same functionality is achieved with the WS-Service Group specification.

## Addressing

Web Services, and thus Grid Web Services, are addressed with URIs, but OGSI under GT3 has added its own addressing scheme. A Grid service URI is called a Grid Service Handle (GSH) and is a globally unique identifier for a Grid Web service instance that will identify the instance into perpetuity. A GSH, however, does not provide sufficient information to allow a client to communicate with the Grid Web service since it does not specify what protocols or bindings should be used. A Mapper Grid Web service maps a GSH to a Grid Service Reference (GSR), which is a WSDL document that does contain binding information for use by the client.

In GT4, the WS-Addressing specification is used to provide an addressing scheme instead of GSH and GSR. Under WS-Addressing, a WS-Addressing Endpoint Reference provides a Web Service and Resource pair, called a WS-Resource. The WS-Addressing specification defines XML messages that identify Web service endpoints with endpoint identification data in the message and allow the Grid Web service to know what stateful Resource to use.

WS-Addressing is not part of WSRF but has been adopted to handle addressing.

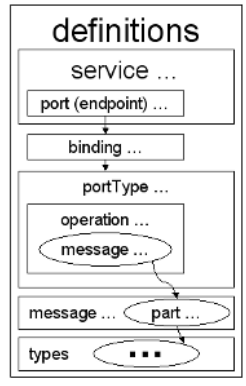
## Anatomy of a WSDL File

WSDL is an XML grammar that describes network services as communication endpoints that exchange messages (Christensen et al., 2001). An overview of the WSDL model can be seen in Figure 147, which will be discussed in greater detail.

Within a WSDL document, *services* contain a set of network endpoints, or *ports* (Christensen et al., 2001). A port contains a network address and is

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 147. WSDL model*




*bound* to a *portType*, where the binding specifies a concrete network protocol and data format. A portType defines a collection of *operations,* which send and receive various *messages*, each of which is defined separately. Messages contain data as logical *parts*, each of which has a *type* defined by the WSDL document. In summary, a port is defined by associating a network address with a reusable binding, and a collection of ports defines a service.

In a WSDL document, portType, message, and operation elements are reusable, abstract definitions in that they are not bound to any particular network protocol or message format. A port is concrete, however, because it is bound to a concrete network protocol and message format; such a port is also known as an endpoint because a caller will have enough information from a port to treat it as an endpoint to network communication. The most common bindings are to SOAP 1.1, HTTP GET/POST, and MIME (Haas, Le Hegaret, Moreau, Orchard, Schlimmer, & Weerawarana, 2004). Bindings are also reusable, since they do not specify an address — the port holds this information.

WSDL is extensible in that any network protocol and message format can be used, as long as the binding is written for it. Additionally, abstract definitions such as binding, portType, message, and types can be re-used by other documents.

Figure 147 shows some of the major elements that appear in any WSDL document that defines a network service:

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- **Types:** a container for data type definitions used to describe the messages that operations will exchange.
- **Messages:** an abstract and typed definition of the data that will be sent and received by operations belonging to the service. A message consists of logical parts, and each of these must be associated with a type.
- **Operation:** an abstract definition of a single unit of functionality offered by the service that includes a description of all input and output messages.
- **Port Type:** a set of abstract operations. This is a reusable element that can be supported by one or more endpoints.
- **Binding:** specification of a concrete network protocol and data format for a particular portType.
- **Service:** a set of related endpoints or ports.

Each of these will be discussed in greater detail in the coming sections.

*WSDL Example: Document Format Service*

Our sample WSDL file will describe a document formatting service that takes a document (as a string), formats it in some undetermined way, and returns the result. The Java interface for such a service might look like Figure 148.

Here is the equivalent WSDL file in Figure 149.

A bit bigger than the Java interface, is it not?

Very broadly, a Web service exposes a port type that is a container for a number of operations that can be invoked. In this case, the service is called DocumentFormatterService. The only operation it exposes is called FormatDocument, which is invoked by a FormatDocumentInput message containing a DocToFormat. The service will act on the parameters received and output a FormattedDoc as part of a FormatDocumentOutput message. Alternatively, if an error is encountered during the processing of the DocToFormat,

*Figure 148. DocumentFormatter Java interface*

```
public interface DocumentFormatter{
  public FormattedDoc format (DocToFormat docToFormat)
      throws FormatException;
} // end DocumentFormatter interface
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 149. DocumentFormatter WSDL file*

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
 name="DocumentFormatter"
 targetNamespace="http://example.com/documentFormatter.wsdl"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://example.com/documentFormatter.wsdl"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsd1="http://example.com/documentFormatter.xsd">
 <wsdl:types>
  <xsd:schema
   targetNamespace="http://example.com/documentFormatter.wsdl"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd1="http://example.com/documentFormatter.xsd1">
   <xsd:complexType name="DocToFormat">
    <xsd:all>
    <xsd:element name="document" type="xsd:string"/>
    </xsd:all>
   </xsd:complexType>
   <xsd:complexType name="FormattedDoc">
    <xsd:all>
    <xsd:element name="document" type="xsd:string"/>
    </xsd:all>
   </xsd:complexType>
  </xsd:schema>
 </wsdl:types>
 <wsdl:message name="FormatDocumentInput">
  <wsdl:part name="docToFormat" type="tns:DocToFormat"/>
 </wsdl:message>
 <wsdl:message name="FormatDocumentOutput">
  <wsdl:part name="formattedDoc" type="tns:FormattedDoc"/>
 </wsdl:message>
 <wsdl:message name="FormatException">
  <wsdl:documentation
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
     In case anything goes wrong!
  </wsdl:documentation>
  <wsdl:part name="formatException" type="xsd:string"/>
 </wsdl:message>
 <wsdl:portType name="DocumentFormatterPortType">
  <wsdl:operation name="FormatDocument">
   <wsdl:input
    message="tns:FormatDocumentInput"
    name="formatDocumentInput"/>
   <wsdl:output
    message="tns:FormatDocumentOutput"
    name="formatDocumentOutput"/>
   <wsdl:fault
    message="tns:FormatException"
    name="formatException"/>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 149. DocumentFormatter WSDL file (cont.)*

```
</wsdl:operation>
 </wsdl:portType>
 <wsdl:binding
  name="FormatDocumentBinding"
  type="tns:DocumentFormatterPortType">
  <soap:binding
   style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="FormatDocument">
   <soap:operation
      soapAction="http://example.com/FormatDocument"/>
   <wsdl:input name="formatDocumentInput">
    <soap:body
     namespace="tns"
     parts="docToFormat"
     use="literal"/>
   </wsdl:input>
   <wsdl:output name="formatDocumentOutput">
    <soap:body
     namespace="tns"
     parts="formattedDoc"
     use="literal"/>
   </wsdl:output>
   <wsdl:fault name="formatException">
    <soap:fault
     name="formatException"
     namespace="tns"
     use="literal"/>
   </wsdl:fault>
  </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="DocumentFormatterService">
  <wsdl:documentation
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
   A service to format documents.
   Todo: implement a way to select a template determining
   how a document will be formatted!
  </wsdl:documentation>
  <wsdl:port
   binding="tns:FormatDocumentBinding"
   name="FormatDocumentPort">
   <soap:address location="http://example.com/formatdocument"/>
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

a FormatException message will be sent, indicating a fault condition and containing a string description of the error.

### *The Definitions Element*

A WSDL file is a description of a Web service given as a set of *definitions*; thus they are contained by a definition's root element (Christensen et al., 2001). Note that definitions and the phrase "the WSDL document" are interchange-

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

able. Within definitions, there are a number of elements and attributes to consider, each defining separate parts of the Web service.

In the following sections, we will see how some elements within definitions are abstract and others are concrete and what difference this makes.

*Target Namespace*

The targetNamespace attribute of definitions assigns a URI reference to the namespace formed by the unique element and attribute names within the definitions element. The Element and attribute names that belong to this definition are said to belong to the target namespace.

Note that we could have left all the "wsdl:" prefixes out of the WSDL shown earlier. This is because of this line:

```
xmlns="http://schemas.xmlsoap.org/wsdl/"
```

It sets the default namespace for the document as the WSDL namespace, so all element names without a prefix are assumed to be in the xmlns namespace.

*Namespace Declarations*

Most of the element and attribute names used in definitions will belong to other schemas. XML namespaces of these other schemas must be declared within definitions in order to bring their element and attribute names within scope of the WSDL document, which can then use them. The definitions element may have any number of XML namespace declarations, each of which appears as an xmlns attribute. Each namespace declaration maps a namespace prefix to a URI reference, including the targetNamespace. Prefixing a name with a namespace prefix indicates the element or attribute's definition lies in the namespace identified by the prefix. For example, consider the following namespace declaration:

```
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

This indicates that any element or attribute name prefixed with "wsdl" (such as "wsdl:message") belongs to the namespace "http://schemas.xmlsoap.org/wsdl/". Note that the "tns" namespace by convention applies to the current document, or "this namespace."

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 150. Example type element*

```
<xsd:complexType name="DocToFormat">
  <xsd:all>
    <xsd:element name="document" type="xsd:string"/>
  </xsd:all>
</xsd:complexType>
```

*Figure 151. Example schema element*

```
<xsd:schema
  targetNamespace="http://example.com/documentFormatter.wsdl"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd1="http://example.com/documentFormatter.xsd1">
...
</xsd:schema>
```

*Types*

Data types are given by a types element. This contains definitions of the data types used to describe messages sent and received by WSDL operations. The types element shown defines our types using XSD type system, but other type systems can be used. If the service uses standard XML data types, then there is no need for a types element since the data types will already be described.

The types element shown also defines two types to be used in this Web service: DocToFormat and FormattedDoc. The DocToFormat element is shown in Figure 150.

The DocToFormat and FormattedDoc elements are complex types that wrap string values (simple types) representing the document before and after processing by the DocumentFormatter Web service. We could have left out the types element and kept these values as plain strings; the reason we did not do this is to demonstrate the use of types and to allow for some extensibility. Perhaps at a later date we may wish to attach other information to a document.

Briefly, we look at the element enclosing the DocToFormat and FormattedDoc complexType elements in Figure 151.

Note that the xsd:schema element includes a targetNamespace attribute whose value is the same as the targetNamespace attribute of the enclosing wsdl:definitions element. This is because the xsd:schema element is a root element in its own right, and we wish to state that it belongs to the same namespace as the wsdl:definitions element. The other namespace declarations

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

bring into scope of the xsd:schema element any namespaces needed — potentially at least — by type definitions within the xsd:schema element.

The interested reader is referred to http://www.w3.org/TR/2001/REC-xmlschema-0-20010502 (Fallside, 2001) for more information about how to write schema elements.

*Messages*

The message element defines service request or response messages . Each message consists of a number of logical parts that communicate some information needed by the receiver of the message. In our previous example, three message elements are defined: FormatDocumentInput, Format DocumentOutput, and FormatException. The DocToFormat element from our example in Figure 152.

The FormatDocumentInput message would be the first message received. It contains one part: a tns:DocToFormat element, a document to be formatted. After the service has processed the document by applying some formatting template, the document should be returned to the caller via another message: the FormatDocumentOutput message. It contains one part: a tns:FormattedDoc element, the newly formatted document.

What happens if something goes wrong during the formatting? Perhaps the document is empty or contains data that the formatting procedure cannot deal with, such as a string of binary data when ASCII characters were expected. In this case, some sort of error message should be returned: the FormatException message. It contains one part: a string that should describe the error that occurred.

It is noted that our WSDL design pays no attention whatsoever to specifying the particular formatting template that should be applied to the DocToFormat. Neither have we allowed callers to tell the service what sort of data to expect within the DocToFormat. Perhaps we just want to test out our FormatException message. By defining DocToFormat and FormattedDoc as complexType

*Figure 152. Example message element*

```
<wsdl:message name="FormatDocumentInput">
  <wsdl:part element="tns:DocToFormat" name="docToFormat"/>
</wsdl:message>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

element types, we allow for the addition of this information at a later iteration of the design process. For now, it is left to the imagination of the reader as to what other parts each message should have.

*Operations*

An operation is an action or single unit of functionality supported by the Web service, which can offer any number of operations. In a WSDL document, an operation is defined in terms of the messages required to support the action. In the operation element, each message is defined as being an input, output, or fault message. This aspect reveals something of the abstract nature of messages. Although message elements may be written specifically for use as input, output, or fault messages, nothing in the message definition explicitly denotes a message as being an input, output, or failure message for some operation. Instead, the operation element defines for itself what messages it uses for input, output, and failure. The name attribute of an operation does not have to be unique, an allowance for operation overloading.

*Figure 153. Example operation element*

```
<wsdl:operation name="FormatDocument">
  <wsdl:input
    message="tns:FormatDocumentInput"
    name="formatDocumentInput"/>
  <wsdl:output
    message="tns:FormatDocumentOutput"
    name="formatDocumentOutput"/>
  <wsdl:fault
    message="tns:FormatException"
    name="formatException"/>
</wsdl:operation>
```

*Figure 154. Example operation element duplicating use of message element*

```
<wsdl:operation name="FormatDocument">
  <wsdl:input
    message="tns:FormatDocumentInput"
    name="formatDocumentInput"/>
  <wsdl:output
    message="tns:FormatDocumentInput"
    name="formatDocumentOutput"/>
  <wsdl:fault
    message="tns:FormatException"
    name="formatException"/>
</wsdl:operation>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

In our example, we defined three messages named FormatDocumentInput, FormatDocumentOutput, and FormatException. Apart from the names, there is nothing in the message definition that explicitly states whether any of these messages should be an input, output, or fault message. The WSDL specification does not require a parser to take the "input," "output," or "fault" suffix of a message name literally — this is purely a naming convention introduced for readability. Consider again the definition of the FormatDocument operation (Figure 153).

Considering that there is no difference in the current definitions of FormatDocumentInput and FormatDocumentOutput, this operation could just as easily have been as seen in Figure 154.

Realistically, we would not want to design the Web service in this way unless we knew for sure that there would never need to be any differences in the type of data communicated as an input parameter to the operation and the returned result of the operation. However, the point has been illustrated: it is the operation definition that decides whether an already defined message will be used as an input, output, or fault message.

Typically, a Web service operation will be invoked by an input message. The input message should contain any parameters required for the operation to proceed, where each parameter is a logical part of the message. Some processing will then occur, and a return message will be sent containing any data that resulted from the operation that needs to be returned to the invoker as logical parts of that message. The return message is modeled as a separate entity: an output message if processing occurred successfully or a fault message if something went wrong. Thus, Web service operations are characterized by a pattern of asynchronous message passing[1].

The input-output pattern of message passing is the most typical model of operation invocation, but it is not the only one. Each pattern is called a *transmission primitive,* and there are four distinct patterns supported by WSDL (Christensen et al., 2001).

- One-way transmission primitives are characterized by a single message received by the endpoint with no expected return message. The message trigger must be defined by the application. No reply from the service is expected.
- Request-response transmission primitives are characterized by the endpoint receiving a message and sending a correlated return message.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- Solicit-response transmission primitives are characterized by the endpoint sending a message and receiving a corresponding message from some client.
- Notification transmission primitives are characterized by a single message being sent by the endpoint. The message trigger must be defined by the application. No reply from the client is expected.

These patterns are called "transmission primitives" because systems are built from the composition of multiple operation invocations.

Operation elements are abstract definitions. Messages are defined for the operation, but no implementation details are specified for how the operation will be called. What network protocol and data format should be used to invoke the operation? These are questions answered in the binding element.

*Port Type*

A portType is a named collection of operations and their messages. It groups together all operation definitions into a single interface for the service. A Web service can offer any number of operations by including them as part of a portType. The name attribute should be unique among all other portType elements within a WSDL document.

The portType in our example is quite a short piece of code (see Figure 155).

Port type elements are reusable, abstract definitions of abstract operations and the abstract messages used by them.

*Figure 155. Example portType element*

The portType in our example is quite a short piece of code:

```
<wsdl:portType name="DocumentFormatterPortType">
  <wsdl:operation name="FormatDocument">
    <wsdl:input
      message="tns:FormatDocumentInput"
      name="formatDocumentInput"/>
    <wsdl:output
      message="tns:FormatDocumentOutput"
      name="formatDocumentOutput"/>
    <wsdl:fault
      message="tns:FormatException"
      name="formatException"/>
  </wsdl:operation>
</wsdl:portType>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 156. Grammar for wsdl:binding element (Christensen et al., 2001)*

```
<wsdl:definitions .... >
   <wsdl:binding name="nmtoken" type="qname"> *
      <-- extensibility element (1) --> *
      <wsdl:operation name="nmtoken"> *
        <-- extensibility element (2) --> *
        <wsdl:input name="nmtoken"? > ?
           <-- extensibility element (3) -->
        </wsdl:input>
        <wsdl:output name="nmtoken"? > ?
           <-- extensibility element (4) --> *
        </wsdl:output>
        <wsdl:fault name="nmtoken"> *
           <-- extensibility element (5) --> *
        </wsdl:fault>
      </wsdl:operation>
   </wsdl:binding>
</wsdl:definitions>
```

*Figure 157. Example binding element*

```
<wsdl:binding
  name="FormatDocumentBinding"
  type="tns:DocumentFormatterPortType">
  <soap:binding
    style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="FormatDocument">
    <soap:operation soapAction="http://example.com/FormatDocument"/>
    <wsdl:input name="formatDocumentInput">
      <soap:body
        namespace="tns"
        parts="docToFormat"
        use="literal"/>
    </wsdl:input>
    <wsdl:output name="formatDocumentOutput">
      <soap:body
        namespace="tns"
        parts="formattedDoc"
        use="literal"/>
    </wsdl:output>
    <wsdl:fault name="formatException">
      <soap:fault
        name="formatException"
        namespace="tns"
        use="literal"/>
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Binding*

The binding element identifies a portType element and specifies a concrete message format and network protocol for the operations and messages that belong to that portType. Any number of bindings can be defined for a given portType. A binding must not specify any address information. As such, bindings can be reused by any number of Web service ports, who *do* specify address information. The grammar for a binding is shown in Figure 156 (Christensen et al., 2001).

The name attribute of the wsdl:binding element must be unique within the WSDL document. Another attribute of the wsdl:binding element references a portType through the type attribute by giving its QName. The grammar indicates that binding information should be given on a per-binding, per-operation, and per-message basis through the use of extensibility elements. The actual extensibility elements used depend entirely on the actual protocol and data format chosen for the binding. A comprehensive discussion of all the available choices is beyond the scope of this book. Instead, we shall briefly discuss our example of binding and refer the reader to further resources for more information. Figure 157 shows the binding element from our example WSDL file.

Our example uses SOAP as the message format and HTTP as the network protocol. Our choice of message format is indicated through our use of elements belonging to the soap namespace within the wsdl:binding element: soap:binding, soap:operation, and so forth. Our choice of network protocol is indicated by the value given to the transport attribute of the soap:binding element. Other URIs may be used to indicate transport mechanisms such as SMTP, FTP, and so forth, but only one protocol can be specified per binding.

Our example provides protocol specific per-binding and per-operation information with soap:binding and soap:operation elements. Protocol specific information about each message is given with the soap:body and soap:fault elements. The reader is referred to Section 3 of Christensen et al. (2001) for more information about each of these SOAP elements.

Note that operation names within WSDL are not required to be unique. If this is the case, identify the operation by providing name attributes for the input, output and fault message elements that match the name attributes of the corresponding message elements.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 158.Eexample port element*

```
<wsdl:port
  binding="tns:FormatDocumentBinding"
  name="FormatDocumentPort">
  <soap:address location="http://example.com/formatdocument"/>
</wsdl:port>
```

*Port*

A port element defines a single endpoint by giving an address to a binding. It is the concrete definition of an abstract portType: a port has an address and references a binding, which references a portType. A port must not give more than one address, and a port must not give any other binding information apart from an address.

The port element from our sample is given in Figure 158.

The name attribute of a port must be unique among all port elements defined within a WSDL document. The binding attribute must have the QName of a binding element, which in our case refers to the binding element defined within our own WSDL file, tns:FormatDocumentBinding. The attribute value could refer to external binding elements as long as their namespaces are brought into the scope of the WSDL file (see the section on namespace declarations).

The actual address is given in an extensibility element, specific to the message format we chose in the section on binding — SOAP. The soap:address element here gives a HTTP address that would have a real implementation of the service, were this a real demonstration!

*Service*

A service is an abstract element that groups a set of related ports. Our service element is shown in Figure 159.

The first thing to notice is that service is the container for port elements such as the one we looked at in the previous section. The name attribute of a service must be unique among all service elements defined within a WSDL document.

A service element groups related ports. The following characteristics should exist between ports within a service.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 159. Example service element*

```
<wsdl:service name="DocumentFormatterService">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    A service to format documents.
    Todo: implement a way to select a template determining
    how a document will be formatted!
  </wsdl:documentation>
  <wsdl:port
    binding="tns:FormatDocumentBinding"
    name="FormatDocumentPort">
    <soap:address location="http://example.com/formatdocument"/>
  </wsdl:port>
</wsdl:service>
```

- Ports are independent: the output of one port should not be required as the input to another. This is not say that a client to a Web service cannot chain together invocations; but no port should rely on such a relationship existing.
- Ports that share a portType but have different bindings or addresses are alternatives that offer semantically equivalent behavior. This allows a client of the Web service to choose from a set of functionally equivalent ports based on criteria such as protocol or message format.
- As mentioned in the first point, ports should be independent of one another. However, this does not prevent applications being built that rely on a particular set of port types being present within the same service. A service's port types can be determined by examining its ports — a port references a binding, which references a portType — and a decision can

*Figure 160. GWSDL root element*

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
  name="DocumentFormatter"
  targetNamespace="http://example.com/documentFormatter.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:ogsi="http://www.Gridforum.org/namespaces/2003/03/OGSI"
  xmlns:gwsdl="http://www.Gridforum.org/namespaces/2003/03/GridWSDLExtensions"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://example.com/documentFormatter.wsdl"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd1="http://example.com/documentFormatter.xsd">
...
</definitions>
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

be made upon whether or not to use the service based upon the presence of a particular set of port types.

## Anatomy of a GWSDL File

Web Services describe themselves with Web Service Description Language (WSDL) files. Grid Web Services represent an extension of standard Web Services and thus describe themselves with Grid Web Service Description

*Figure 161. GWSDL for a document formatting SERVICE*

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
  name="DocumentFormatter"
  targetNamespace="http://example.com/documentFormatter.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:ogsi="http://www.Gridforum.org/namespaces/2003/03/OGSI"

xmlns:gwsdl="http://www.Gridforum.org/namespaces/2003/03/GridWSDLExtensions
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://example.com/documentFormatter.wsdl"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd1="http://example.com/documentFormatter.xsd">
  <import location="ogsi.gwsdl"
    namespace="http://www.Gridforum.org/namespaces/2003/03/OGSI"/>
  <types>
    <xsd:schema
      targetNamespace="http://example.com/documentFormatter.wsdl"
      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsd1="http://example.com/documentFormatter.xsd1">
      <xsd:element name="DocumentFormatterProperties">
      <xsd:complexType name="DocToFormat">
        <xsd:all>
        <xsd:element name="document" type="xsd:string"/>
        </xsd:all>
      </xsd:complexType>
      <xsd:complexType name="FormattedDoc">
        <xsd:all>
        <xsd:element name="document" type="xsd:string"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:schema>
  </types>
  <message name="FormatDocumentInput">
    <part name="docToFormat" type="tns:DocToFormat"/>
  </message>
  <message name="FormatDocumentOutput">
    <part name="formattedDoc" type="tns:FormattedDoc"/>
  </message>
  <message name="FormatException">
    <documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
      In case anything goes wrong!
```

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Figure 161. GWSDL for a document formatting SERVICE (cont.)*

```
</documentation>
   <part name="formatException" type="xsd:string"/>
 </message>
 <gwsdl:portType name="DocumentFormatterPortType"
    wsdlpp:extends="wsrpw:GetResourceProperty"
    wsrp:ResourceProperties="tns:DocumentFormatterProperties">
  <operation name="FormatDocument">
    <input
      message="tns:FormatDocumentInput"
      name="formatDocumentInput"/>
    <output
      message="tns:FormatDocumentOutput"
      name="formatDocumentOutput"/>
    <fault
      message="tns:FormatException"
      name="formatException"/>
  </operation>
 </gwsdl:portType>
</definitions>
```

Language (GWSDL) files. However, each Grid Web service also appears as a Web service and must therefore be described with a corresponding WSDL file. The Globus Toolkit provides tools that generate a WSDL description of a Grid Web service from a GWSDL file. This section examines parts of a sample GWSDL. The root element of a GWSDL file is the definitions element, just as it is in a WSDL file (Figure 160).

The definitions element has a name and targetNamespace attribute. The name attribute simply gives the definitions element a name that does not reflect in any way upon the actual portType itself. The targetNamespace attribute brings with it some important namespaces for OGSI, GWSDL and Service Data elements. There are only two differences between a GWSDL and WSDL file, (illustrated next) that are in accordance with the differences pointed out in the section on differences between Web Services and Grid Web Services. The types element (service data) is linked to the GWSDL portType element through a wsrp:ResourceProperties attribute which names an xsd element. Also, the WSDL is defined to extend a WSRF interface through a wsdlpp:extends attribute of the portType element. Otherwise, this example is much the same format as that shown in the section on WS-Resource — Service Data in GT4.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

# Building Grid Solutions

The following section looks at some examples of Grid networks already implemented or in the process of being implemented by organizations around the world. The research projects reveal the longer term future of Grid networks, implemented on behalf of separate organizations drawn together into VOs to meet different, though compatible, goals. On the other hand, the section on early adopters shows that the current commercial imperative demands that enterprises investing in Grid technology deal with more immediate concerns to do with utilization and providing scalability for existing applications.

## Research Projects

There are a number of well-funded research initiatives with industrial collaboration underway that aim to show how large-scale Grid networks can be built to solve real problems that exist in various commercial, scientific, and governmental areas. These are not purely proof of concept projects: they are strategically sound ideas with tangible applications and in many cases have already begun producing results. Funding bodies for these initiatives include the UK eScience Programme (http://www.rcuk.ac.uk/escience), the European Union's Sixth Framework Programme (http://europa.eu.int/comm/research/fp6/index_en.html), and the National Grid Singapore (http://www.ngp.org.sg). For an in depth view of many of these activities, the reader is directed to (Foster, 2004) as just a few of the projects shall be mentioned here.

*Distributed Aircraft Maintenance Environment (DAME)*

Distributed Aircraft Maintenance Environment (DAME, http://www.cs.york.ac.uk/dame) facilitates improved predictive maintenance and computer-based fault diagnosis and prognostication (DP). In this project, the equipment being monitored are Rolls Royce aircraft engines in flight. The engines have in-built sensors to record data on each flight which is regularly analyzed by a Rolls Royce in-house application and a prognostic application developed by Data Systems and Solutions. Snapshots of engine data are compared with ideal engine models to detect when an engine may need maintenance and prognosticate potential fixes, that is, workflows for maintenance personnel. A summary of the Grid services to be provided is shown next.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- Engine data service to gather data from engine monitoring system into Grid data repositories through ground stations. This will be a transient service, instances of which will be created each time an aircraft touches down, although it is possible to transmit such data while in the air. Such a system would need to integrate the output of multiple complex sensor networks into a coherent data set that can be stored and manipulated by DP systems.
- Data storage and mining service, which store historical data and examine it for patterns at the behest of fault DP algorithms. Based on this historical data, decisions can be made as to whether an engine is in need of maintenance, and if so, what corrective measures are most likely to be needed.
- Engine modeling and reasoning services. These are the fault DP algorithms themselves, responsible for determining if there is a problem and managing workflow for any possible fixes. DP systems need to store and manipulate non-declarative knowledge, such as procedures for diagnosing problems and prognosticating one or more solutions.
- Interface service for the various stakeholders to interact with the system. Includes managing interactions between stakeholders such as maintenance, repair and overhaul staff. It also includes capturing information about the actions they have taken, information which is essential for continual improvement of the system overall.

Since these services are to be used in airlines all over the globe, stakeholders and the systems they use are distributed globally — they cannot be reliably centralized. Service providers will be distributed, using facilities in different countries and controlled by different organizations. Data Grids can be used to provide timely access to the massive amounts of fleet-wide historical data needed by such a system. Integration of separate systems can be modeled as the composition of services: services for sensors, data mining and warehousing, reasoning, and data transport. Grid systems would appear to provide excellent solutions in particular for applications that require manipulation of large amounts of data from various remote sources. This application has another advantage in that fault diagnosis and prognostication solutions can be readily applied to other fields such as medicine, transport, and engineering.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*NEESit*

NEESit is the Network for Earthquake Engineering Simulation IT (for the National Science Foundation) and is built to aid study of the impact of seismic events on modern infrastructure such as buildings, bridges, roads etc. This field incorporates three distinct branches of study and two distinct methods of study. The branches of study are: structural engineers who look at how fabricated physical structures respond to earthquakes; geotechnical engineers who examine how different soils and landforms respond to earthquakes; and other engineers who study tsunamis cased by earthquakes. The methods of study are: experimental, involving small-scale physical simulations with equipment such as shake tables, reaction walls, and centrifuges; and computational simulations built from numeric models.

This work involves Grid data repositories for experimental and computational data. It also requires experimental equipment that can be controlled remotely. This is telepresence equipment to control the experimental equipment and teleobservation equipment to observe experiments and take measurements. Each of these are separate systems that need to interact and will be modeled as services on the Grid.

Stakeholders in this project include research personnel in different facilities that have historically had limited ability to interact and share information in meaningful ways, that is, there is no one single or major center of study or unified process to facilitate data exchange. NEESGrid will advance the study of earthquakes in four ways, discussed next.

- Increase the utility of engineering test facilities by allowing much greater access to equipment such as the shake tables, reaction walls and centrifuges. Distributed centers need to control their own experiments even if the facilities are remote. NEESGrid will incorporate a large-scale distributed telepresence system under which all of the items used for physical experiments will be made network accessible, subject to remote monitoring and control. Thusly, each center will be given the ability to conduct, control and benefit from experiments under NEESGrid.
- Maximize the use of computational methods in earthquake research by granting shared access to simulation data stores and the facilities needed to conduct such experiments and store the results. Centers need long term and real time access to the output of their own simulations and those conducted by other centers.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- Enable much greater collaboration between this diverse community of scientists and engineers through shared access to common resources, allowing researchers in the different sub disciplines to improve understanding of the interaction between structures and soils under earthquake conditions. Shared access to both facilities and experimental data can increase the utility of such facilities and improve the results produced by those users.
- Further development of new research techniques that combine computational and experimental methods. More and more, experimental methods are being replaced by numerical models that can offer much cheaper results and be involved directly in design activities. However, experimental data is needed to validate numerical models. NEESGrid aims to solve this difficulty and allow for the on-demand coupling of computational and experimental facilities.

This project shows that the nature of the Grid reflects the nature of the community that will be using these resources, as a VO is formed by the usually separate research communities involved in earthquake research.

## Early Adopters

The previous section showed a range of mostly research driven projects, there are many examples of corporate "early adopters" who are profitably using Grid technology. This section will briefly examine a few of those.

Hewit Associates pension calculation Grid project ported a mainframe-hosted application that performed complex calculations to an enterprise wide Grid network and reduced transaction costs by 90% (Bort, 2005). This application made good Grid material for two reasons. The first was that the application was very popular but its use was characterized by extreme and unpredictable peaks consisting of a high volume of small but computationally intensive requests. The second reason is that apart from a small database query, each use of the application required little interaction with other systems. This meant the application was relatively isolated and each job had no dependencies on other jobs.

Hewit Associates' second candidate for their enterprise Grid network is a printing composition application that prepares 100,000+ PDF or postscript files. As a single batch job on the mainframe, if the job halted at the end, the whole lot would have redone. When ported to the Grid, it will split the large job

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

into many smaller jobs with no dependencies, and submit them to the Grid to be done in parallel, representing a massive time saving and another CPU intensive task taken away from the mainframe.

Bowne & Co. ported an application to DataSynapse's LiveCluster Grid software that is responsible for helping mutual fund customers meet financial reporting regulations (Dubie, 2005). The application needs to produce hundreds of thousands of reports in response monthly and quarterly spikes in demand, often with only days in which to complete each job. Moving this application to a Grid network has halved production time for the reports.

Wachovia Corporate & Investment Banking in Charlotte, North Carolina, America used DataSynapse's GridServer software to provide on-demand hardware resources to multiple in-house applications, making good use of under-utilized desktops and servers. With this new technology, financial simulations that took hours now take minutes. Wachovia saw an increase in trading volumes of 300%-400% and the number of simulations Wachovia is able to run increased by some 2,500%! The Grid framework has meant that Wachovia can deal with more complex models that previous software was not able to handle (Shread, 2002).

Johnson & Johnson have a Grid network powered by United Devices (Dubie, 2005). Their Grid network implements job scheduling and on-demand computing power for departments of the business that need it. Johnson & Johnson announced recently that they are taking their Grid network global. Some 3,000 nodes are planned, with the global Grid network being put in place to aid applications for their pharmaceutical R&D division.

J.P. Morgan is combining seven separate financial risk management systems on their Grid network (Hamblen, 2004). The applications are designed to assist traders in assessing and managing financial risks for their customers, and the first motivation for J.P. Morgan to pursue Grid technology for these applications was to reduce the cost of managing the applications.

The system will be called Compute BackBone (CBB) and involved the Bank paying an upfront cost of $US4.5 million. The Grid middleware will focus on caching, management, queuing and scheduling. The plan is to have around 2,000 CPUs on the Grid network, running on some 50 Sun Blade servers. This is an investment in new hardware specifically for a Grid network rather than leveraging already existing hardware. J.P. Morgan is obviously willing to put a lot into this project and is expecting to get a lot out of it. CIO Michael J. Ashworth says they are looking for "infinite" capacity and are already seeing

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

considerable savings. CBB saved J.P. Morgan $US1 million in 2003 and an expected $US5 million in their 2004 figures, savings that arise from improved systems management plus lower hardware, development and operational costs. The Grid network allows the bank to recover more easily when a server fails, enables them to deploy new applications much more quickly and to a more scalable infrastructure.

Grid networks implemented by the "early adopters" such as those shown here are single enterprise Grids designed primarily to deal with spikes in demand for applications that are computer intensive and have few dependencies and where dedicated hardware was often bought just for the Grid. This is just the start. The more ambitious forms of Grid networks are those that stretch beyond the single enterprise to form VOs. A provider will make available massive on-demand Grid resources to corporations, which will draw on those resources just as we draw electricity from a power Grid. Such models will come with appropriate accounting, scheduling, and QoS infrastructures.

## Feasibility

The following section examines whether there is a commercial imperative for Grid computing and, if so, in what form will this imperative drive further development.

Does Grid computing have a commercial imperative? In 2002-2003, the Grid computing sector saw approximately $US1 billion in mergers and acquisitions (Boulton, 2003). The reason for this spending, say The 451 Group (*Grids 2004: From rocket science to business service*, 2003), is that major IT vendors wishing to deliver on-demand computing will do so through alliances and acquisitions. Companies involved in Grid spending include Microsoft, Sun, Computer Associates, Intel, Oracle, Platform Computing, Veritas, AMD, Ascential and SAS (Boulton, 2003). Further, a report produced by The Insight Research Corporation (*Grid computing for business markets hits mainstream, says insight research*, 2005) claims that spending on Grid technologies will grow from $714.9 million in 2005 to approximately $19.2 billion in 2010.

What is preventing a widespread adoption of the larger Grid computing strategies by enterprises? Ultimately, still at question is the general ability for Grid technology to produce feasible ROI figures. There are not enough success

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

stories yet. A report from the 451 Group (*Grid computing: Where is the value?*, 2004) posits a reason for why Grid computing has not been broadly adopted yet. It says that most Grid software currently in production are focused towards computationally intensive jobs and are not good at decision making in terms of where and when to run a job in the Grid network. Conversely, current Grid technologies are not good at handling dependencies and offer no advantage when a job involves more communication than computation.

It seems that in the intermediate future, what will most strongly appeal to the enterprise is the ability to derive greater CPU utilization from existing or cheaply purchased hardware. By identifying and making use of idle machines in the enterprise's network, Grid technology can enhance the enterprise's ability to deal with unexpected spikes in demand.

## Beyond the Imperative

The sections on research projects and early adopters show that there is a momentum behind Grid technology sufficient to drive it into the enterprise market. What sorts of scenarios exist to provide the imperative for Grid computing to move out of the scientific and technical realms into commercially oriented business? The scenarios must represent real problems or opportunities faced by commercial organizations. The solutions must require electronic resources on a scale that will be difficult if not impossible for a single organization to provide by itself. This implies sharing of resources between organizations. (Foster, 2001) characterizes this as a need for coordinated resource sharing and problem solving by VOs. A VO is a set of individuals and/or organizations that share resources according to a set of rules that govern transactions between resource providers and consumers, specifying what resources are shared, with whom and under what conditions.

The projects outlined in the section on research projects define middleware infrastructure requirements that are far in advance of what current Grid middleware offers in terms of commonly available software and supporting hardware. These projects will show the way ahead for Grid computing in three very important ways. Firstly, they will show that such ventures are indeed possible and profitable. Secondly, they will establish generic requirements for worldwide Grid middleware to provide shared resources for science and industry both. Thirdly, these projects will demonstrate precisely how VOs will succeed with Grid technology in ways that single enterprises cannot: through co-operation between organizations of disparate natures and far-flung loca-

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

tions seeking a solution for a problem that is too large to be dealt with in a centralized manner.

Entities within a VO will usually have vastly different motivations, sizes, resources, business, and social structures. However, since they are drawn together for a common purpose of sharing electronic resources, certain common technological needs can be identified.

- Sharing models must be flexible, not just client-server or peer to peer, but any mixture in between.
- Security must be a prime concern, covering the three pillars of a secure communication: privacy, integrity, and authentication.
- Resource providers must have flexible, reliable control over how resources are shared. Control must be fine grained, allow for multiple controlling parties, and provide for delegation and the application of differently scoped policies, from local through to global.
- While invariably accessed through computer systems, the nature of the resources to be shared varies greatly. For example, applications, data, CPU cycles, networks and sensors.
- Resource consumers require usage models that can incorporate single or multiple users and allow decisions to be made based on various metrics, including QoS, scheduling, allocation and accounting

These needs provide the imperative for Grid computing: no distributed computing technology currently addresses all of these requirements in a dynamic, cross-organizational manner (Foster, 2001). Further, Grid computing should not necessarily replace existing distributed computing technologies. Instead, it should complement those technologies by bringing them together and opening new realms of scale. Existing technologies can be brought together as compositions of services and new realms of scale come about when existing technologies are allowed to access resources provided across entities within a VO.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

# References

Abbas, A. (2004). *Grid computing: A practical guide to technology and applications (Programming Series)*. Charles River Media.

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

Barry, D. K. (2003). *Web services and service-oriented architectures: The savvy manager's guide*. Morgan Kaufmann.

Beck, K. (1999). *Extreme programming explained: Embrace change.* Addison-Wesley.

Berman, F., Fox, G., & Hey, T. (2003). *Grid computing: Making the global infrastructure a reality.* John Wiley & Sons.

Berners-Lee, T., Fielding, R., Irvine, U. C., & Masinter, L. (1998). *RFC 2396 — Uniform resource edentifiers (URI): Generic syntax*. Network Working Group. Retrieved from http://www.ietf.org/rfc/rfc2396.txt

Berners-Lee, T., & Fischetti, M. (1999). *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor.* San Francisco: Harper.

Berstis, V. (2002). *Fundamentals of grid computing.* IBM RedPaper: 28.

Biron, P. V., & Malhotra, A. (2001). *XML Schema Part 2: Datatypes.* W3C. Retrieved from http://www.w3.org/TR/2001/REC-xmlschema-2-20010502

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Biswas, J., Lazar, A. A., Huard, J.-F., Lim, K., Mahjoub, S., Pau, L.-F. et al. (1998). The IEEE P1520 standards initiative for programmable network interfaces. *IEEE Communications Magazine Volume, 36*(10), 64-70.

Bort, J. (2005). *Grid on the job.* Network World Fusion. Retrieved from http://www.nwfusion.com/supp/2004/ndc6/1025hewitt.html

Boulton, C. (2003). *Storage, grid, intelligence M&As Boom in Q3*. Retrieved from siliconvalley.internet.com. http://siliconvalley.internet.com/news/article.php/3086391

Bray, T., Hollander, D., & Layman, A. (1999). Namespaces in XML. *World Wide Web Consortium*. Retrieved from http://www.w3.org/TR/1999/REC-xml-names-19990114

Burbeck, S. (1987). *Applications programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*. Retrieved from http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

Chin, R. S., & Chanson, S. T. (1991). Distributed object-based programming systems. *ACM Computing Surveys (CSUR), 23*(1), 91-124.

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (n.d.). *JCommerce tutorials*. JCommerce Dev Network for VistaEdge Technologies Inc. Retrieved from http://www.javacommerce.com/tutorials.jsp

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). *Web Services Description Language (WSDL) 1.1.* W3C. Retrieved 2004 from http://www.w3.org/TR/2001/NOTE-wsdl-20010315

Clay, W., Richardson, D. A., Vitale, J., Len, P., & Smith, K.T. (2004). *Professional portal development with open source tools: Java™ Portlet API, Lucene, James, Slide.* Wrox.

Cockburn, A. (2001). *Agile software development*. Addison-Wesley.

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

Credle, R., Atlaf, F., Barnes, J., Galvez, J., Hiranniah, S., Huang, P., & Silveira, F. (2004). IBM WebSphere Portal for Multiplatforms V5 Handbook. International Business Machines Corporation (IBM) International Technical Support Organization. Retrieved from http://www.redbooks.ibm.com/abstracts/sg246098.html?Open

Davidson, M. (n.d.). *The Bean Builder tutorial*. Retrieved from https://beanbuilder.dev.java.net/guide/tutorial.html

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Dessert, M., & Colby, E. (2002) General Motor's employee portal — Liftoff Plus 1 Year: The skys the limit. In *Proceedings of the IHRIM2002 Conference*.

Doerzapf, A., & Udolph, S. (2002) Maximising return on employee relationships. *SAP Insider*, *4*(1), 20-29.

Dubie, D. (2005). *Grid gaining enterprise traction.* Network World Fusion. Retrieved from http://www.nwfusion.com/news/2005/021405grid.html

Eckerson, W. (1999). *Plumtree blossoms: New version fulfils enterprise portal requirements.* Patricia Seybold Report. Boston.

Fallside, D. C. (2001). *XML Schema Part 0: Primer.* W3C. Retrieved from http://www.w3.org/TR/2001/REC-xmlschema-0-20010502

Ferreira, L., Berstis, V., Armstrong, J., Kendzierski, M., Neukoetter, A., Takagi, M., et al. (2003). *Introduction to grid computing with Globus.* International Business Machines Corporation (IBM) International Technical Support Organization. Retrieved from http://www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf

Flynn, P. (2004). *The XML FAQ v4.0.* Retrieved from http://www.ucc.ie/xml/

Foster, I. (2002). *What is the grid? A three point checklist.* Argonne National Laboratory & University of Chicago. Retrieved from http://www.univa.com/grid/pdfs/WhatIsTheGrid_ID.pdf

Foster, I., Kesselman, C., Nick, J. M., & Tuecke, S. (2002). *The physiology of the grid: An open grid services architecture for distributed systems integration.* Open Grid Service Infrastructure WG, Global Grid Forum.

Foster, I., Kesselman, C., & Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications, 15*(3).

Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., et al. (2004). *Modeling stateful resources with Web Services.* Retrieved from http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf

Foster, I., & Kesselman, C. (1998). *The Grid: Blueprint for a new computing infrastructure.* Morgan Kaufmann Publishers.

Foster, I., & Kesselman, C. (2004). *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier.

*Grid computing: Where is the value?* (2004). The 451 Group. Retrieved from http://www.the451group.com/products_and_services/451gars.php

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Grid computing for business markets hits mainstream, says insight research.* (2005). The Insight Research Corporation. Retrieved from http://www.insight-corp.com/pr/2_3_05.asp

*Grid Today* (2002). Wachovia, Morgan Chase bullish on grids. *1*(15).

*Grids 2004: From rocket science to business service.* (2003). The 451 Group. Retrieved from http://www.the451group.com/special_reports/special_report_detail.php?icid=8

Haas, H., Le Hegaret, P., Moreau, J.-J., Orchard, D., Schlimmer, J., & Weerawarana, S. (2004). *Web Services Description Language (WSDL) Version 2.0 Part 3: Bindings.* W3C. Retrieved from http://www.w3.org/TR/2004/WD-wsdl20-bindings-20040803

Hamblen, M. (2004). *J.P. Morgan harnesses power with grid computing system.* Computerworld. Retrieved from http://www.computerworld.com/industrytopics/financial/story/0,10801,90416,00.html

Hamilton, G. (n.d.). *JavaBeans*. Sun Microsystems Inc. http://java.sun.com/products/javabeans/docs/spec.html

Hans Muller, M. D. (2000). *JavaBeans Specification: Getting listeners from JavaBeans.* Sun Microsystems Inc. Retrieved from http://java.sun.com/products/javabeans/docs/getListeners.html

Hanson, J. (2002). Take advantage of the benefits of loosely coupled Web services. *Builder.com.* Retrieved from http://builder.com.com/5100-6386-1050425.html and http://builder.com.com/5100-6386_14-1050425-2.html

Harkey, D., Appajodu, S., & Larkin, M. (2002). *Wireless Java programming for enterprise applications: Mobile devices go corporate.* John Wiley & Sons.

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

Hepper, S. (2003). *Comparing the JSR 168 Java Portlet Specification with the IBM Portlet API.* International Business Machines Corporation (IBM).

Hepper, S., & Hesmer, S. (2003). *Introducing the portlet specification.* JavaWorld. Retrieved 2005 from http://www-106.ibm.com/developerworks/websphere/library/techarticles/0312_hepper/hepper.html

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Huber, J. F. (2002). *UMTS and mobile computing*. Artech House Publishers.

*J2EE 1.4 Tutorial Update 4.* (2004). (For Sun Java System Application Server Platform Edition 8.1 2005Q1 Release Candidate.) Sun Microsystems Inc. Retrieved from http://java.sun.com/j2ee/1.4/download.html#tutorial

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

Joseph, J. (2003). *A developer's overview of OGSI and OGSI-based grid computing*. IBM Developer Works. Retrieved from http://www-106.ibm.com/developerworks/library/gr-ogsi/

Kesselman, C., Prudhomme, T., & Foster, I. (2004). Distributed telepresence: The NEESgrid earthquake engineering collaboratory. In C. Kesselman & I. Foster (Eds.), *The Grid 2: Blueprint for a new computing infrastructure* (pp. 81-93). Elsevier.

Kishore, C., Holley, K., & Tuggle Jr., E. (2004). *IBM Software Group: Migrating to a service-oriented architecture*. Retrieved from http://www-128.ibm.com/developerworks/webservices/library/ws-migratesoa2/

Komatineni, S. (2002). *Understanding UDDI and JAXR.* O'Reilly OnJava.com. Retrieved from http://www.onjava.com/pub/a/onjava/2002/02/27/uddi.html

Mallick, M. (2003). *Mobile and wireless design essentials.* Wiley.

May, P. (2001). *Mobile commerce.* Cambridge University Press.

Nelson, T. H. (1965). (video) Mindful Press 3020 Brudgeway Suite 295, Sausolito CA 94965.

Perrone, P., Venkata, S. R., & Schwenk, T. (2003). *J2EE developer's handbook*. Sams.

Roth, M., & Pelegri-Llopart, E. (2003). *JavaServer Pages™Specification.* Sun Microsystems. Retrieved from http://jcp.org/aboutJava/communityprocess/final/jsr152/

Rotter, M., Kahn, C., & Anderson, P. (2003). Get started with cascading style sheets. *Builder.com.* Retrieved from http://builder.com.com/5100-31-5074849.html?st.bl.html.dvlp

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Rubinstein, D. (2003). *WSRP: An oasis for portal deployment.* Retrieved 2005 from http://sdtimes.com/news/074/story9.htm

Schaeck, T., & Hepper, S. (2002). *Portal standards.* Retrieved 2005 from http://www.theserverside.com/articles/article.tss?l=Portlet_API

Schaeck, T., & Hepper, S. (2005). Portal standards for Web services. *Web Services Journal*, *3*(4). Retrieved from http://www.sys-con.com/webservices/archivesa.cfm?type=reg&volume=03&issue=04

Schmidt, H. (2003). Trustworthy components: Composability and prediction. *Journal of Systems and Software, 65*(2003), 215-225.

Shread, P. (2002). Wachovia gets boost from grid. *Gridcomputingplanet.com.* Retrieved from http://www.gridcomputingplanet.com/features/article.php/1553081

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

Smarr, L. (2004). Grids in context. In I. Foster & C. K. Kesselman (Eds.), *The Grid 2: Blueprint for a new computing infrastructure* (pp. 3-12). Elsevier.

Sotomayor, B. (2004). *The Globus toolkit 3 programmer's tutorial.* Retrieved from http://gdp.globus.org/gt3-tutorial

*Sun ONE portal server developer's guide.* (2003). Sun Microsystems Inc. Retrieved from http://docs-pdf.sun.com/816-6758-10/816-6362-10.pdf

Takagiwa, O., Hulbusch, B., Mueller, M., Pattini, C., & Sanyer, W. (2002). *The XML files: Development of XML/XSL applications using WebSphere Studio Version 5.*

Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J., & Minden, G. J. (1997). A survey of active network research. *IEEE Communications Magazine, 35*(1), 80-86.

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

Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Maquire, T., Sandholm, T., Snelling, D., & Vanderbilt, P. (2003). *Open Grid Services Infrastructure (OGSI) Version 1.0.* GLOBAL GRID

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

FORUM. Retrieved from http://www.ggf.org/ogsi-wg or https://forge.gridforum.org/docman2/ViewCategory.php?group_id=43&category_id=392

Vogels, W. (2003). Web services are not distributed objects. *Internet Computing, 7*(6), 59-66.

Watt, D. (2003). *E-business implementation: A guide to Web services, EAI, BPI, e-commerce, content management, portals, and supporting technologies*. Butterworth-Heinemann.

*Web services activity*. (2004). W3C. Retrieved from http://www.w3.org/2002/ws

*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/

Wladawsky-Berger, I. (2004). The industrial imperative. In Ian Foster & C. Kesselman (Eds.), *The Grid 2: Blueprint for a new computing infrastructure* (pp. 35-34). Elsevier.

WSRP Specification Version 1. (2003). *Web Services for Remote Portlets, OASIS*. Retrieved 2005 from http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf

## Additional Resources

*The Global Grid Forum (GGF)*. (n.d.). Retrieved from http://www.globalgridforum.org

*Jini.org*. (n.d.). Retrieved from http://www.jini.org/

*New to XML*. International Business Machines (IBM) Developer Works. Retrieved from http://www-106.ibm.com/developerworks/xml/newto/index.html

JSR 168 (2003). Portlet Specification. Retrieved from http://www.jcp.org/en/jsr/detail?id=168

UPnP Forum. Retrieved from http://www.upnp.org/

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

*Servlets Specification 2.4* (2004). Retrieved from http://java.sun.com/products/servlet/reference/api/index.html

*Struts*. The Apache Software Foundation. Retrieved from http://struts.apache.org/

*Web Services Description Language (WSDL): An Intuitive View*. developers.sun.com. Retrieved from http://java.sun.com/dev/evangcentral/totallytech/wsdl.html

*Web Services Resource Framework*. The Globus Alliance. Retrieved from http://www.globus.org/wsrf

Webopedia: namespace. Retrieved from http://webopedia.com/TERM/n/namespace.html

Website for NEESit: Network for Earthquake Engineering Simulation IT (National Science Foundation). Retrieved from http://it.nees.org/

XML Information Web sites. http://www.w3.org/xml, http://www.xml.com, http://www.oasis-open.org/cover/xml.html, http://www.ibm.com/developer/xml/

*XML Software*. Retrieved from http://www.xmlsoftware.com/

*XML Tutorial*. JCommerce Dev Network for VistaEdge Technologies Inc. http://www.javacommerce.com/displaypage.jsp?name=intro.sql&id=18238

XML: Extensible Markup Language - Links Page. *Web Developer's Virtual Library*. Retrieved from http://wdvl.com/Authoring/Languages/XML/

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

# Glossary

**Common Gateway Interface:** A standard for interfacing external applications with information servers, such as HTTP or Web servers.

**Complex Type:** A data type that is composed of other types, simple or complex. Also known as composite type.

**Endpoint Reference:** In WS-Addressing terminology, an endpoint reference is the address of a particular WS-Resource.

**Enterprise Archive File:** An Enterprise Archive file represents a J2EE application that can be deployed in a WebSphere application server. EAR files are standard Java archive files and have the file extension .ear. The modules that make up the EAR file are themselves packaged in archive files specific to their types—for example, a Web module contains Web archive files and an EJB module contains Java archive files. EAR files also contain a deployment descriptor (an XML file) that describes the contents of the application and contains instructions for the entire application, such as security settings to be used in the run-time environment.

**Extensibility Element:** XML element used to specify a concrete protocol or message format. Typically used within a WSDL binding to associate technology specific information — also known as binding information — with an

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

abstract definition. Extensibility elements must use an XML namespace different from that of WSDL, implying that such elements 'extend' the WSDL by introducing new namespaces.(Christensen, Curbera, Meredith, & Weerawarana, 2001).

**Global Element Definition (GED):** XML element declarations specify the structure, type, occurrence, and value constraints for an XML element. Global element declarations appear as children of the xsd:schema element. They can be reused by referencing them in other parts of the schema or from other schema documents.

**Global Grid Forum (GGF):** Global Grid Forum (http://www.globalGridforum.org/).

**Grid Web Services Definition Language (GWSDL):** Grid Web Services Definition Language, build upon WSDL.

**IBM Cloudscape:** A pure, open source-based Java relational database management system that can be embedded in Java programs and used for online transaction processing (OLTP).

**Java Server Pages:** JavaServer Pages (JSP) technology enables Web developers and designers to rapidly develop and easily maintain, information-rich, dynamic Web pages that leverage existing business systems. As part of the Java technology family, JSP technology enables rapid development of Web-based applications that are platform independent. JSP technology separates the user interface from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content.

**Jini:** Java middleware based on a service directory providing leased service registration and discovery. Service providers register a service object reference, which may a remote proxy. Service consumers perform lookup operations to find a service that implements the desired interface and may download the reference to any matches they find.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

**Kerberos:** A network authentication protocol. It is designed to provide strong authentication for client/server applications by using secret-key cryptography.

**Lightweight Directory Access Protocol:** An open industry standard that defines a standard method for accessing and updating information in a directory.

**Local Element Definition:** XML element declarations specify the structure, type, occurrence, and value constraints for an XML element. Local element declarations are declarations that appear within complex types or model groups and are not references to global elements.

**Multipurpose Internet Mail Extensions:** An official Internet standard that specifies how messages must be formatted so that they can be exchanged between different e-mail systems. MIME is a very flexible format, permitting one to include virtually any type of file or document in an email message. Specifically, MIME messages can contain text, images, audio, video, or other application-specific data. The fomat has been described in RFC 1521.

**Namespace:** A namespace is a set of names with some form of identification so that a system using the namespace can use names within the space by qualifying each name with the appropriate namespace identifier. XML namespaces are a collection of element and attribute names where the namespace is identified by a URI. (Bray, Hollander, & Layman, 1999). This allows XML documents to use different elements and attributes that have the same name as long as they come from different namespaces.

**Notification Transmission Primitive:** Pattern of message passing characterized by a single message being sent by the endpoint. The message trigger must be defined by the application. No reply from the client is expected. This pattern could be used to implement timed events initiated by the service or a listener pattern whereby a client has asked to be informed by the service when a particular event occurs.

**One-Way Transmission Primitive:** Pattern of message passing characterized by a single message received by the endpoint with no expected return

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

message. This pattern could be used to implement sensor networks, where each client is a sensor that relays current readings to a service responsible for collating the results.

**Port Type:** A service interface — the interface implemented by a Grid service. Usually written as portType.

**Primitive:** In computing terms, a primitive is a low-level operation or object from which higher-level objects or operations can be constructed. A programmer would compose primitives together to create more complex (read: composite) operations or objects.

**QName:** A QName is an XML qualified name (Biron & Malhotra, 2001). An XML qualified name uniquely references an XML element by specifying the namespace name and local part name for the given element. For example, consider the following QName: xsd:complexType. The format of a QName is namespace_prefix:local_part. The namespace_prefix must be associated with a namespace URI reference in a namespace declaration such as what you might see in a definitions element: xmlns:xsd="http://www.w3.org/2001/XMLSchema". The local_part must be the unique name of an element defined in the namespace referred to by the namespace_prefix. For our example, this means there must be an element named "complexType" found at the namespace referenced by the URI "http://www.w3.org/2001/XMLSchema".

**Request-Response Transmission Primitive:** Pattern of message passing characterized by the endpoint receiving a message and sending a correlated return message. This is the standard pattern for most service invocations where the client initiates an operation with a message containing required parameters; the service conducts some form of processing and returns the result in a new message.

**Remote Procedure Call:** RFC 1050 defines client-server protocol for passing parameters and evoking remote procedure using messages.

**Resource Property:** An individual component of a WS-Resource's state, represented by an XML element within a WS-Resource Property Document.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

**RSA Cryptographic Algorithm:** The RSA algorithm invented by Ronald L. Rivest, Adi Shamir, and Leonard Adleman in 1977.

**Service Oriented Architecture (SOA):** Service Oriented Architecture refers to a collection of services that have some means of communicating with each other in a request-response pattern of message passing (Barry). Services within this architecture should be loosely coupled with respect to location, protocol and time (Hanson, 2002).

**Simple Data Type:** A simple type is a data type whose instances are composed of only one item of data. Also known as atomic or primitive types.

**Simple Object Access Protocol:** A communication protocol for communication between applications via Internet. It is platform and language independent, and based on XML. It allows the message to pass through firewalls because it is carried by HTML traffic.

**Solicit-Response Transmission Primitive:** Pattern of message passing characterized by the endpoint sending a message and receiving a correlated message from some client. This pattern could be used to implement leasing where a service maintains a list of clients that have a lease with the service. Periodically the service solicits a response from the client to see if they are active and still wish to maintain their lease.

**Stateful Resource:** A structured data entity that stores state data that is associated with and can be manipulated by a Grid Web service. A stateful resource must be expressible as an XML document and have a well-defined lifecycle. An instance of a stateful resource may be created via a Web service referred to as a stateful resource factory.

**Stateful Resource Factory:** A Web service that can instantiate stateful resources.

**Stateful Resource Identifier:** A reference to a Grid Web service stateful resource, use by WS-Addressing in GT4.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

**Struts:** An open source framework for building Java Web applications.

**Transmission Primitive:** A pattern of message passing. With reference to Web Services this refers to one of the four patterns supported by WSDL: one-way, request-response, solicit-response and notification transmission primitives.

**Uniform Resource Identifiers (URI):** (Berners-Lee, Fielding, Irvine, & Masinter, 1998), the generic term for all types of names and addresses that identify abstract or physical resources. A URL is one kind of URI, but it is a mistake to think that URI's can be looked up on the Web.

**Universal Description, Discovery and Integration:** A Web-based distributed directory that enables businesses to list themselves on the Internet and discover each other, similar to a traditional phone book's yellow and white pages.

**Universal Plug 'N' Play:** Platform and language independent middleware designed for robust connectivity among stand-alone devices and PCs .

**Universal Resource Locator:** Global address system for documents and other resources on the Word Wide Web. General format is protocol:// server:port/path. A unique name of a Web page. URL belongs to a broader group of URI — Universal Resource Identifiers, which identify resources in the Web: documents, images, downloadable files, services, electronic mailboxes, and other.

**Web Archive File:** File format similar to Java Archive File, used for packaging Web applications. The structure of WAR file:

- The static HTML files and JSPs are stored in the top level directory.
- The Servlet and related Java technology class files must be stored in the WEB-INF/classes directory.
- Any auxiliary library JAR files must be stored in the WEB-INF/lib directory.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

- The deployment descriptor is stored as a file named web.xml in the WEB-INF directory

**Web Service Description Language:** An XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

**Web Services:** A set of standards that define programmatic interfaces for application-to-application communication over a network (2004). Standards use XML languages such as UDDI, WSDL, and SOAP.

**Web Services for Remote Portlets:** Portlet provided to the requestor over Web, with interface description that is defined in a public directory service.

**Web Services Resource Framework (WSRF):** http://www.globus.org/wsrf

**World Wide Web (WWW):** A collection of Internet servers that deliver specially formatted documents to clients across the Internet.

**WS-ResourceProperties:** Specification for creation and use of WS-Resource Properties Document.

**WS-Resource Properties Document:** A XML Global Element Declaration (GED) in a given namespace that defines the type of a stateful resource, a.k.a. resource property document.

**WS-Resource:** A composition formed by a Web service description being associated with a stateful resource.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

# Acronyms

**CBB:** Compute BackBone (for J.P. Morgan)
**CGI:** Common Gateway Interface
**CORBA:** Common Object Request Broker Architecture
**CPU:** Central Processing Unit
**DAME:** Distributed Aircraft Maintenance Environment
**DCE:** Distributed Computing Environment
**DOM:** Document Object Model
**DP:** Diagnosis and prognostication
**DRP:** Disaster Recovery Plan
**DTD:** Document Type Definition
**EAR:** Enterprise Archive file
**EIS:** Enterprise Information System
**EJB:** Enterprise Java Bean
**FTP:** File Transfer Protocol
**GAR:** Grid Archive file
**GGF:** Global Grid Forum
**GIOP:** General Inter-ORB Protocol
**GSH:** Grid Service Handle

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

**GSR:** Grid Service Reference

**GTK:** Globus Tool Kit as in GTK3 and GTK4

**GWSDL:** Grid Web Services Definition Language

**HP:** Hewlett Packard

**HTTP:** Hyper Text Transfer Protocol

**HTTPS:** Hyper Text Transfer Protocol Secure

**IBM:** International Business Machines

**IDE:** Intelligent Development Environment

**IIOP:** Internet Inter-ORB Protocol

**ISP:** Internet Service Provider

**J2EE:** Java 2 Enterprise Edition

**JAXM:** Java API for XML Messaging

**JAX-RPC:** Java API for XML based Remote Procedure Call

**JDBC:** Java DabaBase Connectivity APIs

**JMS:** Java Messaging Service

**JSP:** Java Server Pages

**JSTL:** Java Standard Tag Library

**JTA:** Java Transaction API

**JVM:** Java Virtual Machine

**LDAP:** Lightweight Directory Access Protocol

**LPTA:** Lightweight Third Party Authentication

**MIME:** Multipurpose Internet Mail Extensions

**NEESit:** Network for Earthquake Engineering Simulation IT (National Science Foundation)

**OASIS:** Organization for the Advancement of Structured Information Standards

**OGSA:** Open Grid Services Architecture

**OGSI:** Open Grid Service Infrastructure

**OOD:** Object Oriented Design

**ORB:** Object request Broker

**PAPI:** Provider Application Programming Interface

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

**PDF:** Portable Document Format (Adobe Acrobat)
**QoS:** Quality of Service
**R&D:** Research & Development
**REST:** Representational State Transfer
**RFC:** Request For Comments
**RMI:** Remote Method Invocation
**RPC:** Remote Procedure Call
**RPI:** Remote Portlet Invocation
**RSA:** RSA Cryptographic Algorithm
**RSS:** Really Simple Syndication
**SAML:** Security Assertion Markup
**SAX:** Simple API for XML
**SDE:** Service Data Element
**SMTP:** Simple Message Transfer Protocol
**SOA:** Service Oriented Architecture
**SOAP:** Simple Object Access Protocol
**SSAJ:** SOAP with Attachments API for Java
**SSL:** Secure Socket Layer
**SSO:** Single Sign On
**SVG:** Scalable Vector Graphics
**TLS:** Transport Layer Security
**UBR:** UDDI Business Registry
**UDDI:** Universal Description, Discovery and Integration
**UPnP:** Universal Plug 'N Play
**URI:** Uniform Resource Identifiers
**URL:** Uniform Resource Locator
**VO:** Virtual Organization
**W3C:** World Wide Consortium
**WAP:** Wireless Application Protocol
**WAR:** Web Archive file
**WML:** Web site Meta Language

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

**WSDD:** Web Services Deployment Descriptor
**WSDL:** Web Service Description Language
**WSIA:** Web Services for Interactive Applications
**WSIL:** Web Services Inspection Language, a.k.a. WS-Inspection
**WSRF:** Web Services Resource Framework
**WSRP:** Web Services for Remote Portlets
**WSS:** Web Services Security
**XML:** eXtensible Markup Language
**XSD:** XML Schema Definition
**XSLT:** XML Stylesheet Engine

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

# About the Authors

**Jana Polgar** is a lecturer at Monash University in Melbourne, Australia. While working in the Faculty of Information Technology, she teaches subjects focusing on Web services and portal design, and implementation in postgraduate courses. Her research interests include multimedia composition using Web services and portlets. She also has extensive industry experience in various roles ranging from software development to management consulting positions. She holds a master's degree in electrical engineering from VUT Brno (Czech Republic) and a PhD from RMIT Melbourne.

**Robert Mark Bram** is currently a PhD student and works as a portal analyst programmer. His areas of interest include varied forms of networked applications, ranging from grid computing to portals and Web services. Particular areas of interest include techniques for modeling QoS for multimedia systems over a grid network and whether such services might be usefully aggregated in a portal. He spends much time hacking away at code while, at home, two cats, a dog, a daughter and a loving partner all attempt to show him that spare time should not be spent playing computer games.

**Tony Polgar** is currently a portal development manager. He has firsthand experience with the software life cycle, development methodology, and use of various Web/Portal technologies in large retail organisations. Tony has worked

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

for more than 30 years in industry, beginning as a software developer, working in management appointments, and spending 7 years in various IBM development labs in Australia and the U.S. He has managed multiple client server, Web and portal software projects. Tony is a certified project manager with interests in software quality, programming methods, and future technologies. Tony has degrees in mechanical engineering, mathematics, and computing, as well as interests in classical music and photography.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

# Index


## A

action-URIs 135
administration transparency 248
administrator-managed segments 189
adopters 304
America Online 122
Apache Jetspeed 121
Apache Struts 116
application code 60
application server 163
application vendors 234
asynchronous 57
ATG Dynamo 121
authentication 139, 187
authentication server 139

## B

B2B (business-to-business) 120
B2C (business-to-consumer) 120
back-end systems 120
binding 23
bottom-up processing 26
broker 179
business integration 235
business logic 105
business portal solutions 125
business to employee (B2E) 120
business users 34
business-to-business (B2B) 120
business-to-consumer (B2C) 120
business-to-employee 126

## C

caching 170, 243
cascading style sheets (CSS) 205
CBL (common business language) 9
cell containers 159
CGI 82
channel 208
click-to-action 182
code tracking 217
collaboration 32, 124, 130, 143, 163
collaboration center 163
collaboration services 130
column containers 159
commerce XML (CXML) 9
common business language (CBL) 9
common request broker architecture (CORBA) 32, 233
communications 253
complex type 14


Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

concrete-portlet 201
concurrency handling 169
configure 240
container 159
content management 130
content providers 235
controller 116, 137
conversational state 113
cookies 113
cooperative portlets 177
CORBA (common request broker architecture) 32, 233
core XML tags 10
cost 239
CPU 170, 255
cryptosystem 129
CSS (cascading style sheets) 205
custom tags 98
customer 235
CXML (commerce XML) 9

## D

DAME (distributed aircraft maintenance environment) 301
data type definition (DTD) 12
database 163, 217
database monitoring 217
DCOM (distributed component object model) 32
decryption 128
deployment descriptor 25, 87, 109, 147, 199
Desktop 208
DII (dynamic invocation interface) 70
directory services 128
disaster recovery services 240
distributed aircraft maintenance environment (DAME) 301
distributed component object model (DCOM) 32
distributed time transparency 248
document object model 28
DOM 29
domains 9
DTD (data type definition) 12
dynamic 70, 96, 120
dynamic invocation interface (DII) 70
dynamic proxies 70

## E

early adopters 304
ebXML (Electronic Business with XML ) 9
edit 240
effectiveness 4
EJB (Entity Java Beans) 107, 137
EJB service endpoint 107
electronic business with XML (ebXML) 9
encryption 128
enterprise applications 120
enterprise bean 97, 109
Entity Java Beans (EJBs) 107, 137
event handling 225
exception-mapping descriptor 27
extensible markup language (XML) 7, 15

## F

failure transparency 248
Filters 83
filters 96
first generation portals 124
fourth generation portals 124

## G

graphical user interface (GUI) 122
Green Pages 34
grid applications 249
grid architectures 255
grid resources 251
grid service handle (GSH) 291
grid system 247
grid Web services 267
GSH (grid service handle) 291
GUI (graphical user interface) 122

## H

heterogeneous data sources 119
heterogeneous environment 104
hibernate 116

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

hibernate query language 116
horizontal domains 9
HTML code 2
HTTP 85

## I

identity provider 129
IDL (interface definition language) 23
import statements 25
integration 235
inter-portlet communication 225
interface definition language (IDL) 23
internationalization 83
interoperability 55, 104, 234, 262
intra-organizational Grid network 257

## J

Java messaging service (JMS) 22
Java parsers 15
Java server pages (JSP) 94
Java virtual machine (JVM) 164
Java-to-WSDL mapping 26
Java-XML-type-mapping 27
JavaServer faces (JSF) 115
JAX-RPC 63
JAX-RPC invocation models 69
JAXB 59
JAXR 42
JMS (Java messaging service) 22
JSF (JavaServer faces) 115
JSP (Java server pages) 94
JSR 168 219, 220
JVM (Java virtual machine) 164

## K

knowledge worker portals 126

## L

label 159
language 116
LDAP directory 139, 163
eLicenses 254
lightweight third party authentication (LTPA) 187
local portlets 174
local transparency 248
location transparency 248
logic 105
Logon transparency 248
LTPA (lightweight third party authentication) 187

## M

mapping 26
maximize buttons 243
message 23
metadata 271
method-param-parts-mapping 27
methodology 241
Microsoft SharePoint 121
minimize 243
mobile devices 131
model 136
model view controller (MVC) 96, 115, 136
monitoring tools 217
multi-tier component-based architecture 56
multi-tier distributed application model 56
MVC (model view controller) 96, 115, 136
MVC2 139, 160, 213

## N

namespace 11, 39, 167
namespace prefix 25
namespace transparency 248
navigational strategies 242
NEESit 303
Netscape 122
network operating system (NOS) 247
NOS (network operating system) 247
notification 80, 91, 154

## O

OASIS 129
OGSA (open grid services architecture) 261
open grid services architecture (OGSA) 261

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

operating environment monitoring tools 217
operating system 163
operation 23

## P

package-mapping 26
packaging 168
page 159
page aggregation process 156
page-to-page communication 182
parallel CPU capacity 249
parameters 149
performance 240
permissions 141
personalization 131
Plumtree 121
polling 75
port 23
portal 1, 119, 124, 134, 141, 163, 206, 220, 244
portal applications 124
portal content publishing 163
portal deployment 164
portal engine 139, 206
portal framework 125
portal page 221
portal page aggregation 225
portal server 139, 163
portal services 127
portal servlet 140
portal solutions 125
portal technology 82
portal window 158
portlet 134, 145, 168, 174, 220, 238
portlet configuration data 168
portlet container 220
portlet interface 221
portlet lifecycle 145
portlet states 144
portlet window 141, 221
portlet-to-portlet communication 238
portlet.xml 151, 165
PortletAdapter 196
PortletApplicationSetting 152
PortletConfig 152
PortletData 152, 169
PortletPageListener 161
PortletSession 151
PortletSettings 152
presentation services 128
processing layer 107
professional pages 126
property transfer 181
provider 209
pull parser 21

## R

remote access to information 231
remote access transparency 248
remote location 170
remote method invocation (RMI) 32
remote portlet invocation (RPI) 177
remote portlets 174
remote procedure call (RPC) 27, 49, 59
rendering phase 156
replication transparency 248
request-response operation 57
risks 239
RMI (remote method invocation) 32
RPI (remote portlet invocation) 177
row containers 159

## S

SAXParser 17
schema 8
screen 159
scriptlets 97
second generation portals 124
secure socket layer (SSL) 129, 187
security 83, 128, 226
security assertion markup language (SAML) 129
server 139
service 23, 104, 284
service endpoint 104
service groups 284
service interaction layer 106
service semantics 264
service-end-point-interface-mapping descriptor 27

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

service-endpoint-method-mapping 27
service-interface-mapping 27
service-oriented architecture (SOA) 4, 262
servlet 82, 91
servlet context attribute listeners 91
servlet context listeners 91
servlet security 83
session attribute listeners 92
session listeners 91
session maintenance 169
shortcut keystrokes 244
simple object access protocol (SOAP) 32, 48, 177, 219
single sign on (SSO) 128, 186, 190, 239
site analysis 131
skin 158
skins 205
SOA 5
SOAP (simple object access protocol) 32, 48, 177, 219
software 35, 254
software developers 35
specialized equipment 254
SSL (secure socket layer) 129, 187
secure socket layer (SSL) 129, 187
SSO (single sign on) 128, 186, 190, 239
state data 271
stateful services 265
static 69, 96, 183
static information transfer 183
static stubs 69
storage 253
stub 67
stylesheets 19
support 240
synchronous 57

## T

tag handler 97
target mamespace 289
taxonomy 238
technology aware managers 3
template 8, 96, 116, 160
template service 116
theme 141, 158, 205
third generation portals 124
TLS (Transport Layer Security) 129, 187
tModel 38
transaction tracking 217
transparency 247
transport layer security (TLS) 129, 187

## U

UBL (universal business language) 9
UDDI (universal description, discovery and integration) 32
UDDI data structures 37
UDDI operators 36
UDDI replication 36
uniform resource locator (URL) 3, 52, 113, 135
universal business language (UBL) 9
universal description, discovery and integration (UDDI) 32
URL (uniform resource locator) 3, 52, 113, 135
URL rewriting 113
user experience 4
user login 146
user logout 146
user-managed segments 190

## V

variable-mapping descriptor 27
vertical domains 9
view 137
vignnette 121
virtual organization (VO) 262
VO (virtual organization) 262

## W

W3C standards 12
Web container 189
Web module deployment 100
Web page 238
Web server 163
Web service architecture 58

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Web service client design 110
Web service security (WSS) 129, 188
Web services 104
Web services description language (WSDL) 22, 63
Web services for remote portlets (WSRP) 219, 226
Web services integration 132
web.xml 199
WebSphere 121, 206
white pages 33
World Wide Web 110, 122
wrapper portlet 179
WS-security 189
WSDL (Web services description language) 22, 63
WSDL-message-mapping 27
WSDL-return-value-mapping 27
WSDL-to-Java mapping 26
WSRP (Web services for remote portlets) 219, 226
WSS (Web service security) 129, 188

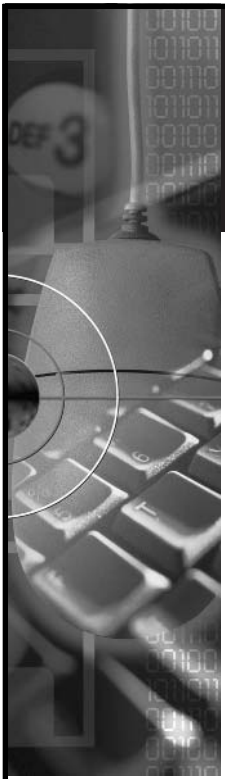
## X

XHTML 14
XLink 12
extensible markup language (XML) 7
XML (extensible markup language) 7
XML authoring 15
XML schema definition language (xsd) 9
XML tags 10
XPointer 14

## Y

Yahoo! 122
yellow pages 34

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.



Experience the latest full-text research in the fields of Information Science, Technology & Management

# InfoSci-Online

**InfoSci-Online** is available to libraries to help keep students, faculty and researchers up-to-date with the latest research in the ever-growing field of information science, technology, and management.

**The InfoSci-Online collection includes:**

- Scholarly and scientific book chapters
- Peer-reviewed journal articles
- Comprehensive teaching cases
- Conference proceeding papers
- All entries have abstracts and citation information
- The full text of every entry is downloadable in .pdf format

**Some topics covered:**

- Business Management
- Computer Science
- Education Technologies
- Electronic Commerce
- Environmental IS
- Healthcare Information Systems
- Information Systems
- Library Science
- Multimedia Information Systems
- Public Information Systems
- Social Science and Technologies

**InfoSci-Online features:**

- Easy-to-use
- 6,000+ full-text entries
- Aggregated
- Multi-user access

**"...The theoretical bent of many of the titles covered, and the ease of adding chapters to reading lists, makes it particularly good for institutions with strong information science curricula."**

**— Issues in Science and Technology Librarianship**



**To receive your free 30-day trial access subscription contact:**

Andrew Bundy

Email: abundy@idea-group.com • Phone: 717/533-8845 x29

Web Address: www.infosci-online.com

InfoSci-Online

Full Text • Cutting Edge • Easy Access

A PRODUCT OF Idea Group Inc.

Publishers of Idea Group Publishing, Information Science Publishing, CyberTech Publishing, and IRM Press

**infosci-online.com**

**Single Journal Articles and Case Studies Are Now Right at Your Fingertips!**

Purchase any single journal article or teaching case for only $18.00!

Idea Group Publishing offers an extensive collection of research articles and teaching cases that are available for electronic purchase by visiting www.idea-group.com/articles. You will find over 980 journal articles and over 275 case studies from over 20 journals available for only $18.00. The website also offers a new capability of searching journal articles and case studies by category. To take advantage of this new feature, please use the link above to search within these available categories:

- Business Process Reengineering
- Distance Learning
- Emerging and Innovative Technologies
- Healthcare
- Information Resource Management
- IS/IT Planning
- IT Management
- Organization Politics and Culture
- Systems Planning
- Telecommunication and Networking
- Client Server Technology
- Data and Database Management
- E-commerce
- End User Computing
- Human Side of IT
- Internet-Based Technologies
- IT Education
- Knowledge Management
- Software Engineering Tools
- Decision Support Systems
- Virtual Offices
- Strategic Information Systems Design, Implementation

You can now view the table of contents for each journal so it is easier to locate and purchase one specific article from the journal of your choice.

Case studies are also available through XanEdu, to start building your perfect coursepack, please visit www.xanedu.com.

For more information, contact cust@idea-group.com or 717-533-8845 ext. 10.

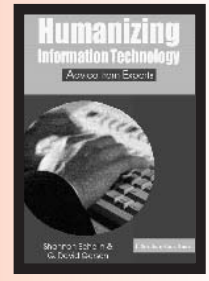
**www.idea-group.com**

IDEA GROUP INC.

# IT Solutions Series

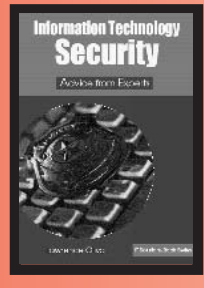
## Humanizing Information Technology: Advice from Experts

Authored by:
Shannon Schelin, PhD, North Carolina State University, USA
G. David Garson, PhD, North Carolina State University, USA




With the alarming rate of information technology changes over the past two decades, it is not unexpected that there is an evolution of the human side of IT that has forced many organizations to rethink their strategies in dealing with the human side of IT. People, just like computers, are main components of any information systems. And just as successful organizations must be willing to upgrade their equipment and facilities, they must also be alert to changing their viewpoints on various aspects of human behavior. New and emerging technologies result in human behavior responses, which must be addressed with a view toward developing better theories about people and IT. This book brings out a variety of views expressed by practitioners from corporate and public settings offer their experiences in dealing with the human byproduct of IT.

*ISBN 1-59140-245-X (s/c) •* US$29.95 *• eISBN 1-59140-246-8 • 186 pages • Copyright©2004*




## Information Technology Security: Advice from Experts

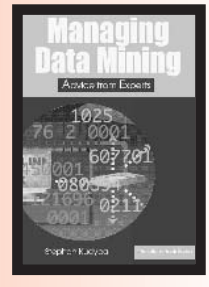
Edited by:
Lawrence Oliva, PhD, Intelligent Decisions LLC, USA

As the value of the information portfolio has increased, IT security has changed from a product focus to a business management process. Today, IT security is not just about controlling internal access to data and systems but managing a portfolio of services including wireless networks, cyberterrorism protection and business continuity planning in case of disaster. With this new perspective, the role of IT executives has changed from protecting against external threats to building trusted security infrastructures linked to business processes driving financial returns. As technology continues to expand in complexity, databases increase in value, and as information privacy liability broadens exponentially, security processes developed during the last century will not work. IT leaders must prepare their organizations for previously unimagined situations. IT security has become both a necessary service and a business revenue opportunity. Balancing both perspectives requires a business portfolio approach to managing investment with income, user access with control, and trust with authentication. This book is a collection of interviews of corporate IT security practitioners offering various viewpoint on successes and failures in managing IT security in organizations.

*ISBN 1-59140-247-6 (s/c) •* US$29.95 *• eISBN 1-59140-248-4 • 182 pages • Copyright©2004*

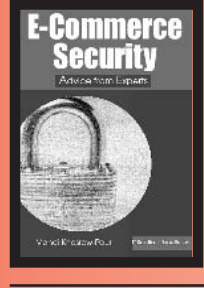
## Managing Data Mining: Advice from Experts

Edited by:
Stephan Kudyba, PhD, New Jersey Institute of Technology, USA
Foreword by Dr. Jim Goodnight, SAS Inc, USA




**Managing Data Mining: Advice from Experts** is a collection of leading business applications in the data mining and multivariate modeling spectrum provided by experts in the field at leading US corporations. Each contributor provides valued insights as to the importance quantitative modeling provides in helping their corresponding organizations manage risk, increase productivity and drive profits in the market in which they operate. Additionally, the expert contributors address other important areas which are involved in the utilization of data mining and multivariate modeling that include various aspects in the data management spectrum (e.g. data collection, cleansing and general organization).

*ISBN 1-59140-243-3 (s/c) •* US$29.95 *• eISBN 1-59140-244-1 • 278 pages • Copyright©2004*




## E-Commerce Security: Advice from Experts

Edited by:
Mehdi Khosrow-Pour, D.B.A.,
Information Resources Management Association, USA

The e-commerce revolution has allowed many organizations around the world to become more effective and efficient in managing their resources. Through the use of e-commerce many businesses can now cut the cost of doing business with their customers in a speed that could only be imagined a decade ago. However, doing business on the Internet has opened up business to additional vulnerabilities and misuse. It has been estimated that the cost of misuse and criminal activities related to e-commerce now exceeds 10 billion dollars per year, and many experts predict that this number will increase in the future. This book provides insight and practical knowledge obtained from industry leaders regarding the overall successful management of e-commerce practices and solutions.

*ISBN 1-59140-241-7 (s/c) •* US$29.95 *• eISBN 1-59140-242-5 • 194 pages • Copyright©2004*

**It's Easy to Order! Order online at www.cybertech-pub.com, www.idea-group.com or call 717/533-8845 x10 — Mon-Fri 8:30 am-5:00 pm (est) or fax 24 hours a day 717/533-8661**

CyberTech Publishing

Hershey • London • Melbourne • Singapore

***Excellent additions to your library!***