Department of Computing, Imperial College London
University of London

Investigating Portlet Technology for Interactive Analytics

Michelle Anne Osmond

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy of the University of London and the Diploma of Imperial College

August 2007

I hereby declare that this thesis entitled "Investigating Portlet Technology for Interactive Analytics" is entirely my own work, except where specifically acknowledged in the text.

Michelle Osmond August 2007

Abstract

This thesis examines and extends portlet technology as an interactive web interface for performing scientific research with the Discovery Net e-Science infrastructure. This approach allows services from different sources to be integrated as components within a single unified web interface, known as a Portal.

Some of the restrictions of developing portlets become particularly significant in a research context. We discuss a number of the issues encountered while developing the Discovery Net portlets to the portlet standard, JSR-168, and our solutions. The two most significant issues encountered are as follows:

1. A research portal needs to provide customised and continually updated information and tools to its users, but traditional portlets are focused on providing a single type of service. Therefore to add new features to the portal, the administrator must obtain and install new portlets.
2. The lack of inter-portlet communication (IPC) in the current Portlet specification (JSR-168) hinders development of standards-compliant portlets. IPC is critical for analytical 'dashboard'-style sites where each portlet is treated as a component which may interact with others.

We discuss in detail our research to provide solutions for these issues by enhancing and extending the use of portlet technology, in particular concerning:

1. Discovery Net's web deployment system, which offers a flexible solution by providing access to a modifiable store of workflows, with support for customised web interfaces and visualisation of results, all without any intervention by the portal administrator.
2. Our development of a JSR-168-compliant IPC library.

To illustrate and drive this work, several example applications have been examined and supported, including Discovery Net e-Science demonstrations and a production-quality Translational Medicine Portal.

Acknowledgements

Thanks first to my supervisors, Yike Guo and John Hassard, for their continual support and encouragement, and for introducing me to such fascinating subject areas.

This project has been truly multidisciplinary, and I am very grateful to all the people in the collaborating research groups and companies who have offered help, advice and criticism. Particular thanks go to Moustafa Ghanem, Anthony Rowe, Jameel Syed, Vasa Ćurčin, Anton Oleynikov, Patrick Wendel, and Mark Richards.

My gratitude also to ICSF for letting me look after the library, and to my family and friends, especially Michael Wright, for always being there.

Contents

Chapter 1. Introduction 11

1.1 Aims 13

1.2 Main Contributions 14

1.3 Technology 14

1.4 Thesis Structure 14

Chapter 2. Background 17

2.1 Web Services 18

2.2 e-Science and the Grid 19

2.3 Discovery Net 20

2.4 Java Servlets and JSP 22

2.5 Virtual Learning/Research Environments 23

2.6 Portals and Portlets 24

2.7 Conclusion 37

Chapter 3. Analytical Portal Design 38

3.1 General advantages of Web Portals 38

3.2 Requirements of an Analytical Portal 40

3.3 Functional requirements for the Discovery Net Portal 42

3.4 Non-functional requirements for the Discovery Net Portal 43

3.5 Discovery Net portlets 44

3.6 Portlet Message Flow 54

3.7 Software constraints for the Discovery Net portal 56

3.8 Choice of Portal implementation 57

3.9 Business Intelligence demonstration 62

3.10 Conclusion 67

Chapter 4. Web Interfaces to Research Services 68

4.1 Creating web interfaces 69

4.2 Discovery Net Service Deployment 75

4.3 Discovery Net's demonstration scientific applications 93

4.4 Using multiple services 106

4.5 Conclusion 108

Chapter 5. Inter-portlet communication 109
5.1 Portal-specific IPC Implementations 111
5.2 Implementation approaches for JSR-168-compliant IPC 118
5.3 Design of a JSR-168-compliant IPC library 129
5.4 Page Navigation 141
5.5 Limitations of the IPC library 142
5.6 Conclusion 143
Chapter 6. Translational Medicine Portal: A Case Study of a Complex Portal 145
6.1 Aims of the Portal 145
6.2 Design Challenges 146
6.3 AJAX (Asynchronous JavaScript And XML) 150
6.4 Portlets developed 152
6.5 The Translational Medicine Portal 167
6.6 Conclusion 171
Chapter 7. Limitations and Improvements for JSR-168 portlet technology 173
7.1 Context 173
7.2 Limitations in Portal Design 175
7.3 Limitations in Portlet Implementation 176
7.4 Considerations for the next Portlet Specification 196
7.5 Conclusion 197
Chapter 8. Conclusions and Future Work 198
8.1 Future Developments 201
8.2 Future Work 203
8.3 Final Thoughts 204
Glossary 205
Bibliography 207

Appendices

A1. Functions for retrieving Portlet IDs 218
A2. Discovery Net Papers 220

List of Tables

Table 3.1: Feature Comparison of Traditional and Analytical Portals. 42

Table 3.2: Message inputs and outputs of Discovery Net Portlets.................................. 55

Table 3.3: Portal Comparison examining suitability for hosting Discovery Net portlets. 60

Table 5.1: Comparison of Portal-specific implementations of Inter-portlet Communication (2005) 117

Table 5.2: Approaches to storing IPC configuration: GUI presentation......................... 134

Table 5.3: Approaches to storing IPC configuration: storage location. 134

List of Figures

Figure 2.1: Discovery Net software. 21

Figure 2.2: Overview of Discovery Net functionality 21

Figure 2.3: "My Yahoo"... 26

Figure 2.4: Portal architecture overview.. 28

Figure 2.5: Elements of a Portal Page 30

Figure 2.6: Example portal page request sequence 32

Figure 2.7: Portlet applications... 35

Figure 2.8: Example portal directory tree 36

Figure 3.1: Service Portlet 45

Figure 3.2: Discovery Net Portal architecture 47

Figure 3.3: Service Index Portlet 49

Figure 3.4: Tasks Portlet.. 50

Figure 3.5: Userspace Index Portlet 51

Figure 3.6: Userspace Item Viewer Portlet, showing a table item.................................... 52

Figure 3.7: Message flow between Discovery Net Portlets. 55

Figure 3.8: Oracle Business Intelligence portlet and Oracle Spatial 63

Figure 3.9: Oracle Business Intelligence Portlets 64

Figure 3.10: Oracle Business Intelligence portlets and Discovery Net services 65
Figure 3.11: Business Intelligence workflow 65
Figure 3.12: Decision support with Discovery Net services 66
Figure 4.1: European Bioinformatics Institute online services 70
Figure 4.2: Partial WSDL of an EBI web service 72
Figure 4.3: Generic SOAP client 73
Figure 4.4: Comparing approaches to service description by Discovery Net and standards. 74
Figure 4.5: Black-box deployment of a Discovery Net workflow 76
Figure 4.6: Deployment strategies 76
Figure 4.7: Form presentation of basic property types 79
Figure 4.8: JavaScript-enhanced form properties 80
Figure 4.9: Applet form properties 80
Figure 4.10: Rendering and submission of a service form (Discovery Net Portal version 2.0) 82
Figure 4.11: Modified procedure for rendering custom parameters(Discovery Net Portal version 3.0) 83
Figure 4.12: Display of arbitrary file types in the portal.. 85
Figure 4.13: Using browser plugins to display results 86
Figure 4.14: Result display using applets 86
Figure 4.15: HTML-formatted result display 87
Figure 4.16: Display of tabular data 88
Figure 4.17: Custom layout of a service form 89
Figure 4.18: The Deployment Layout Editor... 90
Figure 4.19: Creating a wizard-style service layout 91
Figure 4.20: Wizard-style service layout in the portal 92
Figure 4.21: GUSTO sensor network architecture. 94
Figure 4.22: Sensor clusters shown on a map.. 94
Figure 4.23: GISViz visualisation 95
Figure 4.24: Snapshots of GISViz animation 96
Figure 4.25: A GUSTO service for retrieving pollution data 97

Figure 4.26: A GUSTO service for retrieving scatter plots 98
Figure 4.27: Comparison of protein profiles with a mirror plot 100
Figure 4.28: Visualisations of GM scenario results deployed in the portal 101
Figure 4.29: The GISViz applet deployed in the portal 102
Figure 4.30: Two source images used for earthquake analysis 103
Figure 4.31: The Earthquake application in the portal 105
Figure 4.32: Discovery Net service instances 106
Figure 5.1: Scenario: communication between two portlets 109
Figure 5.2: Message board model for IPC 122
Figure 5.3: Simple IPC using a shared session attribute. 125
Figure 5.4: Cross-context IPC 126
Figure 5.5: IBM WebSphere's portlet "wiring" tool 128
Figure 5.6: Oracle Portal's page editing mode 128
Figure 5.7: A newly-added portlet reading an existing message 131
Figure 5.8: Abstraction of message names 133
Figure 5.9: Dynamically-generated IPC configuration forms 136
Figure 5.10: UML diagram of the IPC Library 137
Figure 5.11: Concrete implementations. 138
Figure 5.12: Code fragment showing the use of the IPC library 139
Figure 5.13: IPC library modules 141
Figure 6.1: Oracle Discoverer 147
Figure 6.2: Oracle Discoverer (2) 148
Figure 6.3: Overview of the Windber Translational Medicine portal 149
Figure 6.4: Interactions on a normal web page (not using AJAX) 150
Figure 6.5: Interactions on a web page using AJAX 151
Figure 6.6: The OLAP browser 153
Figure 6.7: Graphical display of rows in the OLAP browser 154
Figure 6.8: Configuration of rows and columns in the OLAP browser. 155
Figure 6.9: Retrieval of raw data from the OLAP browser 156
Figure 6.10: The OLAP browser with a categorical column 156
Figure 6.11: Defining and displaying a row hierarchy in the OLAP browser 158

Figure 6.12: Multiple levels of row expansions, governed by the row hierarchy......... 159
Figure 6.13: Binning management interface.. 161
Figure 6.14: The wizard for creating or editing binnings... 161
Figure 6.15: Manual customisation of numerical bins.. 162
Figure 6.16: Manual customisation of categorical bins... 162
Figure 6.17: The binning wizard... 163
Figure 6.18: Patient Sets portlet, allowing sets to be saved, edited and deleted........... 165
Figure 6.19: Patient Sets portlet, allowing expansion and selection of sets.................. 165
Figure 6.20: The Patient Viewer portlet 166
Figure 6.21: The OLAP browser page 167
Figure 6.22: The Binning page... 168
Figure 6.23: A Patient Viewer page, allowing users to specify a patient ID to view.... 169
Figure 6.24: A Patient Viewer page 169
Figure 6.25: A workflow page 170

Chapter 1. Introduction

The modern researcher makes extensive use of personal computers to do their work, using both 'office' software and domain-specific programs. A growing number also use remote computational resources to perform specialised tasks - from using a service provided through a form on a website, to submitting jobs through the command line to a Grid cluster.

Remote computational services offer many benefits: the end user does not need to deal with installation, specialised hardware, administration or maintenance, or even understand their details. Indeed, when the remote service being offered is resource intensive, e.g. a large database, it would be quite impractical to install it locally on every client machine. Remote services are thus economical, with a single installation able to serve many users over a wide geographical area, allowing users to roam between locations and computers without losing access to their data or software tools. With broadband internet connections becoming ubiquitous even in the home, latency and network bottlenecks have become less of a concern, except for services with high data volume or extremely high demand.

The increasing use of remote services has highlighted the value of user-friendly interfaces which reduce or eliminate the need for specialised training. One of the most popular methods of accessing such services currently is through a web browser: many users are both familiar and comfortable with existing web paradigms such as online shopping and internet banking, both of which provide user-oriented and robust interfaces to back-end systems. Acceptance of the web browser’s role in accessing services in everyday life is increasing: for example according to one UK survey, 38% of people were planning to do some of their 2005 Christmas shopping online, compared to only 22% in 2001[156].

In contrast, access to services for scientific research is regularly through remote shell input or custom client programs (for example, MDL's CrossFire Commander software[57] for querying the CrossFire Beilstein organic chemistry database[133]). It remains to be seen to what extent these methods of access will be replaced by the web, as the capabilities of a browser-based interface are generally inferior to those that can be offered by a custom client program. Nevertheless, the process has already begun; to continue the previous example, MDL has recently developed a rich web interface, DiscoveryGate[58], as an alternative method of accessing the Beilstein database.

Web portals are sites which gather together many services, often on the same page, for quick and convenient access. A typical portal today, such as *My Yahoo*[23], will offer users a personalisable "home page" which can contain many fragments of content ('portlets') showing for example news, entertainment, weather reports or stock quotes. Portals are also used in business intranets, where they can offer consistent and centralised access to business services, and more recently, business process management[157]. However, the portal concept is also being applied in the scientific research community, as project websites develop which provide access to the collection of research data or services hosted by the project (e.g. the NIST Data Gateway[102], which provides access to many of its scientific and technical databases through both web interfaces and purchasable local software versions).

These research portals may be available to the public or only to authorised users, and may be based on bespoke code or upon an existing 'portal' solution such as GridSphere[78]. They may concentrate on providing access to services owned and hosted by the organisation running the portal, or they may act more as a convenient aggregation point, collecting or integrating interfaces to many remote third party services (e.g. Indiana University's Bioinformatics Portal[90] which provides guidance and services to its researchers, or BioTeam's iNquiry[127], a commercial system combining a cluster and a portal for hosting and accessing in-house life science services). In this thesis, we continue this progression and consider the potential of web portals as the primary method of access to research services.

1.1 Aims

The core of this thesis concerns the use of portlet technology in a web portal for scientific research.

We address three main issues:

1. The requirements of a web portal intended for scientific research.
2. The suitability of the JSR-168 Portlet specification for research portals.
3. Methods for allowing interaction with research services through a web interface.

We review the ways in which a web portal can be used to provide access to computational services, and how this changes the research portal's focus and requirements compared to more traditional web portals. We illustrate these differences with the design of a new portal for the Discovery Net[39,177] project at Imperial College London.

We also assess the use of the JSR-168 Portlet Specification[16] for developing portal components. We discuss the benefits and limitations of the standard, show how these affect the design of portlets for research services, and provide alternative solutions and workarounds where necessary. In particular, we present a portable, standards-compliant library for inter-portlet communication (IPC), a practical solution to one of the more significant (and internally acknowledged) omissions of the JSR-168 specification.

Finally, we introduce the problem of managing and presenting web interfaces to a dynamic pool of research services, and outline Discovery Net's approach to the specification and automatic generation of rich, user-friendly web interfaces. Discovery Net provides tools to bring together the data and analysis components which make up a larger task, describing the whole process as a "workflow". These workflows are the research services, which can then be made accessible via the web.

1.2 Main Contributions

1. Analysis of the requirements for a Research Portal, illustrated by our development of Discovery Net portlets and their use in a number of demo applications, most notably a Translational Medicine Portal.
2. Development of a JSR-168-compliant library for inter-portlet communication.
3. Evaluation of the strengths and weaknesses of JSR-168, resulting in advice and solutions for portlet developers, and suggestions for the next portlet specification, JSR-286.

1.3 Technology

The Discovery Net software is a server-client system for workflow editing and execution (Chapter 2). Both server and client are built using Java J2EE[11], running on JBoss Application Server 3.2.4[50]. The system has been commercialised by InforSense Ltd. as InforSense KDE[97] ("Knowledge Discovery Environment"). The work described in this thesis has been done using KDE versions from 1.9.2 (2003) through to 3.0 (2006). Although this thesis will refer to "Discovery Net" software, this is usually interchangeable with "InforSense KDE".

The new Discovery Net portal discussed in this thesis was developed using existing JSR-168-compliant portal implementations. Early development used Apache Pluto[30], which is part of the Apache Portals project, and is the reference implementation of the Java Portlet Specification (JSR-168). For later development and final integration with Discovery Net, Apache Jetspeed 1.6[28] was used. The selection of Jetspeed 1.6 from a range of candidates is discussed in Chapter 3.

1.4 Thesis Structure

Chapter 1. Introduction: Introduction to the problem and an overview of the thesis.

Chapter 2. Background: Background information on e-Science and the Grid, web services, Discovery Net, portals, portlets and virtual learning environments.

Chapter 3. Analytical Portal Design: We discuss how the aims and thus technical requirements of an Analytical or Research portal differ from those of a more traditional portal. We apply this to the design of the Discovery Net Portal, and outline the process of converting the existing Java servlet-based[13] portal to a Java portlet solution. We compare several available portal implementations to determine which ones best fit our needs, and end with a demonstration of a portal which combines third-party Oracle Business Intelligence portlets with Discovery Net services.

Chapter 4. Web Interfaces to Research Services: We describe methods of dynamically-generating web interfaces to remote services, and explain the benefits and problems with these different approaches. Discovery Net's solution for "web deployment" of workflows is described, including the process of encapsulating workflows, and the necessity of providing feature-rich, customisable service interfaces to satisfy user demands and expectations. We illustrate the possible variety of services using Discovery Net’s demonstration scientific applications.

Chapter 5. Inter-Portlet Communication (IPC): We introduce the topic of IPC and show how it is implemented in different portals. We then describe the design and development of our JSR-168-compliant IPC library: one of the main contributions of this thesis.

Chapter 6. Translational Medicine Portal: A Case Study of a Complex Portal: The work on IPC and Discovery Net portlets is brought together with a real-world example: a portal for translational medicine developed for the Windber Research Institute.

Chapter 7. Limitations and Improvements for JSR-168/WSRP 1.0 Portlet technology: As a result of our experiences, we discuss the limitations of using JSR-168 portlets, and describe some solutions to common problems. Finally, we highlight a list of features for consideration in the next portlet specification.

Chapter 8. Conclusion and Future Work.

Appendices

A1. Retrieving a Portlet Window ID: Example code illustrating some of the methods described in Chapter 7.

A2. Discovery Net Papers: We include for convenience some conference papers which give further details on some of the projects discussed in this thesis.

1. *Sensor Grids for Air Pollution Monitoring:* A paper on Discovery Net's GUSTO scenario, from the All Hands Meeting 2004[152]
2. *Distributed BioSensor systems for GM Crop Monitoring:* A paper on Discovery Net's GM Crop scenario, from the All Hands Meeting 2004[169].
3. *Adopting and Extending Portlet Technologies for e-Science Workflow Deployment*: A paper on the limitations of JSR-168 presented at the All Hands Meeting 2005[155].

Chapter 2. Background

The continuing advances in computer technologies are having a profound impact on the way people live their lives. In particular, the wide adoption of the internet and ever increasing processing power have led to new ways of conducting business and doing scientific research.

However, the demands we put upon computers easily keep pace with or exceed their improving capabilities. The network infrastructure needs to support future data- and processing-intensive experiments, for example the distribution, storage and analysis of the data from the Large Hadron Collider at CERN, due to start up in 2007[79]. As data in all scientific fields continues to accumulate rapidly, the methods for marking it up with semantic metadata and searching through it will also become increasingly important, hence work on the Semantic Web[144,175] and the Semantic Grid[128,130].

The initiative to develop internet technologies to satisfy this demand has been given the name 'e-Science' in the UK[5], and the proposed architecture to support it is called the Grid[141]. This includes everything from the mechanics of data storage and transfer, security, computational cluster management and task submission, to semantic service registration and discovery. "Web Services"[10] are one of the core technologies, providing a standard, platform-neutral communication method between computer systems using SOAP[17]. Our project, "Discovery Net"[39,177], is part of the UK e-Science Programme[5] and has developed an infrastructure for scientific analysis focusing on several research application demonstrations. The Discovery Net software allows scientists to build and execute 'workflows', flexibly combining analysis and data components to perform parameterisable tasks. This thesis concentrates on the ability to 'deploy' these workflows to the web, providing a user-friendly web interface to execute the analysis workflow and inspect its results.

Web interfaces are increasingly commonly used to access computational services. With the Discovery Net system, we need to be able to provide access to a dynamic, ever-growing pool of deployed services. Software for creating *web portals* (e.g. PHP-Nuke[4], Apache Jetspeed[29]) provides a base for managing and presenting website content by bringing multiple component *portlets* together to make up a single page. Portals can allow users (as well as site administrators) to customise page layouts and modify portlet preferences to fit their own interests. We will be considering the use of such web portals as a solution for finding and accessing Discovery Net services, and allowing users to personalise their pages for quick access to the services and data which they use most frequently.

More detailed descriptions follow for each of these base technologies. A review of specific methods for creating web interfaces to services is also given later, at the beginning of Chapter 4.

2.1 Web Services

The term "Web Services" covers a range of open and platform-neutral standards allowing direct communication between programs across a network.

Simple Object Access Protocol (SOAP)[17] is the communication protocol most often used, and WSDL (Web Services Description Language)[6] is used for programmatic description of the service interface. WSDL describes what functions are available to call, and how to call them, but does not include semantic information to aid human comprehension of what the parameters or methods are for. Web services can be indexed and found using registries based on the UDDI (Universal Description, Discovery, and Integration)[9] standard from OASIS (Organization for the Advancement of Structured Information Standards).

Further advanced standards and specifications build upon this base, for example extended descriptions of service interfaces can be written using languages such as OWL-S[19] (previously DAML-S), a web service ontology based upon OWL (Web Ontology Language), and security can be added using the OASIS standard WS-Security[113].

These common, XML-based standards allow flexible integration of web services into existing systems, reducing the technical effort required both in programming access to web services and in supporting heterogeneous platforms (many code libraries are available, e.g. Apache Axis[8] for Java and C++, and SOAP::Lite[18] for Perl, while there is integral support in Microsoft's .NET with ASP.NET web services[34]). Current and legacy systems can also be wrapped, exposing them for easy access through web service interfaces.

Web services have been adopted by many communities in recent years, in particular in business and research, to provide various forms of remote functionality such as database access, submitting data, and computational services.

2.2 e-Science and the Grid

"e-Science is about global collaboration in key areas of science, and the next generation of infrastructure that will enable it." - John Taylor, Director General of the Research Councils, OST (CCLRC e-Science Centre)[36]

The infrastructure for e-Science (the Grid) will allow scientists anywhere in the world to access and store terabytes of data, control remote instruments, perform processing-intensive distributed analysis and data mining, use and combine remote services, and interact with colleagues using shared visualisations. For practical use and widespread adoption, the technology must also provide secure, manageable, auditable and dependable access to resources.

Grid middleware is software which smoothes over the heterogeneity in these large scale systems, e.g. in terms of hardware, network protocols and security systems, providing a standardised interface to the users. The *de facto* standard middleware is the Globus Toolkit[139], developed by the Globus Alliance[110]. Globus provides resource and information management, job execution and monitoring, and security. Another commonly used piece of software is Condor[91], a cycle-stealing distributed execution environment which can be used in combination with Globus to manage jobs. Sun's N1 Grid Engine (formerly Sun Grid Engine)[109] is another cluster/job management engine.

The Globus Toolkit version 3 introduced the Open Grid Services Infrastructure (OGSI)[140], which presents grid resources as services using an adaptation of the Web Services model. Grid services may be persistent or transient (created on demand by service factories), and as each user of the service is likely to be accessing their own personal instance of the service, grid services maintain state information. Globus, IBM and HP then developed the Web Service Resource Framework (WSRF)[145,150], which takes many of the concepts and structures from OGSI and applies them as new layers of the web services infrastructure, effectively providing stateful web services by using 'resource' web services to hold data. These new layers[25] make use of the WS-Addressing[111] specification for addressing messages to web services, and include:

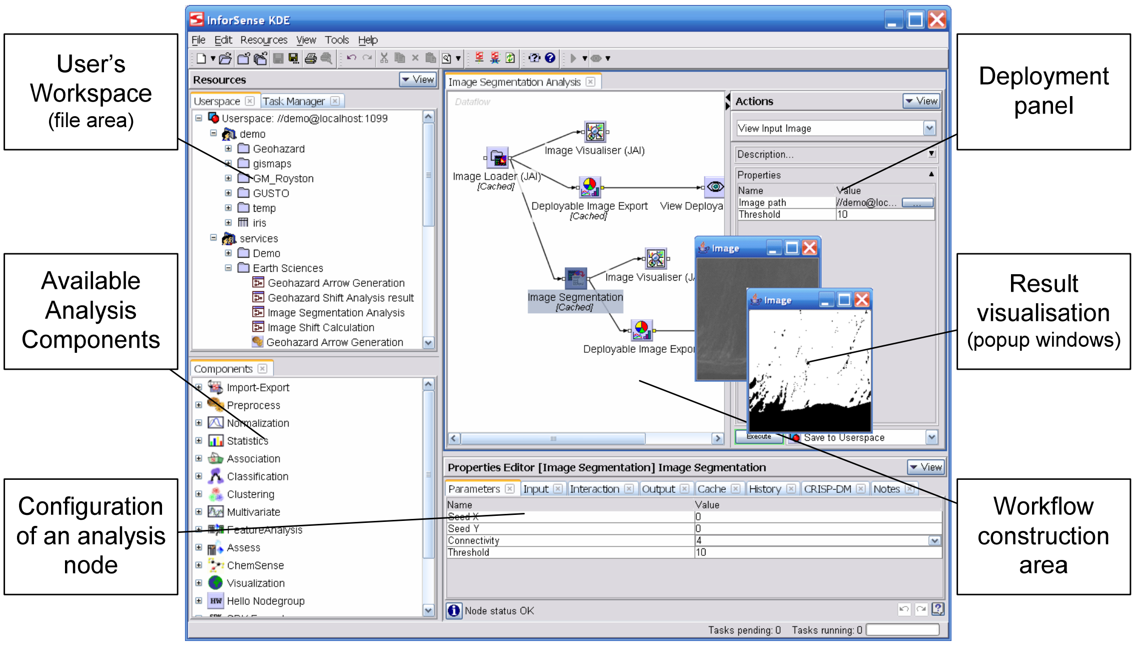
- WS-Notification for publication of messages, subscription, and triggers.
- WS-ResourceProperties for storage, update and retrieval of service data.
- WS-ResourceLifetime for management of resources.

WSRF is used instead of OGSI in version 4 of the Globus Toolkit[139] (April 2005).

2.3 Discovery Net

Discovery Net[39,125,177] is a workflow-based grid computing platform developed at Imperial College London, and commercialised by InforSense Ltd. as InforSense KDE (Knowledge Discovery Environment)[97]. It provides user-friendly access to data pre-processing, analysis and mining components (Figure 2.1), allowing scientists to build and execute reusable workflows for data mining and decision support. It also provides a range of visualisers for inspecting data. New components can be written in Java and plugged in to provide custom domain functionality.

Discovery Net uses a standalone server which manages data storage and workflow execution. The Discovery Net Java Client can be installed on user machines, or launched across the network using Java Web Start[48], and provides a user interface for accessing the userspace (file storage area), and editing and executing workflows (Figure 2.2). Discovery Net workflows may alternatively be accessed as web services, through custom APIs, or through the Discovery Net Web Portal (which is based on Java Servlets, which are described in Section 2.4).


User's Workspace (file area)
Available Analysis Components
Configuration of an analysis node
Deployment panel
Result visualisation (popup windows)
Workflow construction area
InforSense KDE
Resources
Userspace
Task Manager
Components
Image Segmentation Analysis
Image Loader (JAI)
Image Visualiser (JAI)
Deployable Image Export
Image Segmentation
Actions
Execute
Save to Userspace
Properties Editor [Image Segmentation] Image Segmentation
Node status OK

Name	Value
Seed X	0
Seed Y	0
Connectivity	4
Threshold	10

Figure 2.1: Discovery Net software, based upon Inforsense KDE, in use. The environment provides per-user file management (top left), analysis components (bottom left), and workflow composition and execution (right).

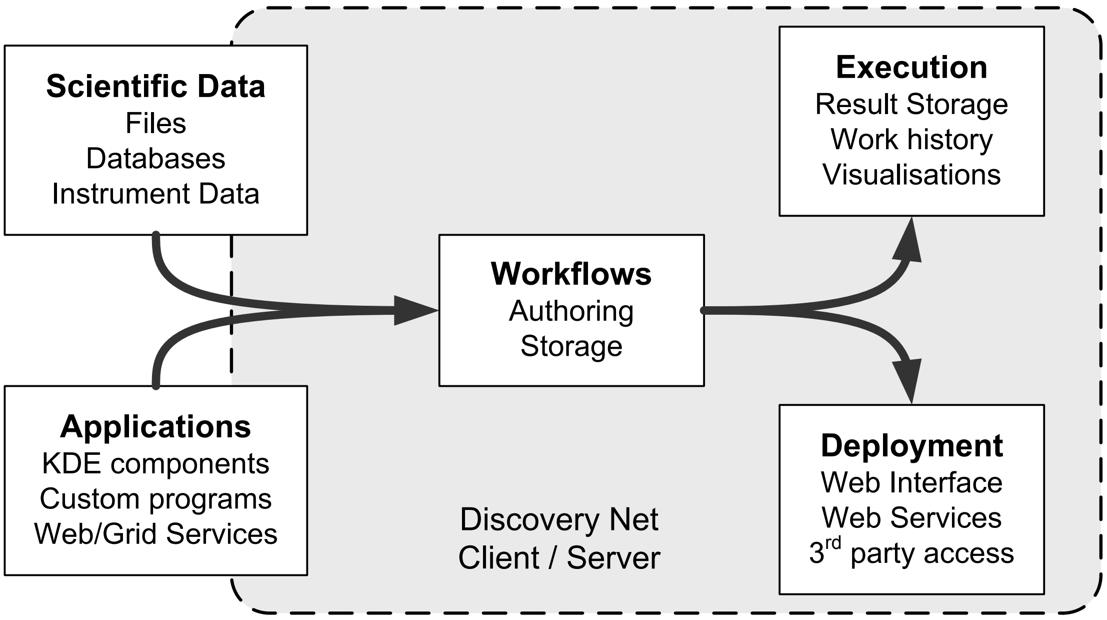


Figure 2.2: Overview of Discovery Net functionality

Discovery Net demonstrates its grid-based data mining platform with respect to a particular set of testbed applications, in collaboration with a number of other research groups in Imperial and companies. These applications are further discussed in Chapter 4.

2.4 Java Servlets and JSP

Java Servlet[13] and JSP (Java Server Pages)[12] technology allows web developers to create 'active' or server-generated web pages in a similar way to that allowed by other web server technologies like Perl, PHP and ASP. The Java base allows for easy integration with existing J2EE components and the use of any Java libraries. Servlets are compiled Java classes which receive requests from a client (usually a web browser) and generate and send a response (the HTML contents of the web page). JSPs allow developers to use a hybrid syntax where HTML may be mixed with Java code or special 'tags', and are compiled on-the-fly to a servlet by the server upon receiving a request. In development, JSPs are closer to scripting languages like PHP, and are well-suited for modification by non-coders (e.g. the style-designers of a page).

Servlets are a well-established technology, and there are many libraries and tools available to make life easier for servlet developers. For example, when developing a servlet page, the servlet code is often split up into multiple parts: first, the request is processed by the servlet, which does any intensive processing (business methods) and data retrieval, and is then passed on to a JSP, which is lightweight and serves mainly to generate the page layout and display the data. This adoption of the Model-View-Controller pattern[3] is commonly seen, and there are a number of toolkits available to allow such structured code design over an entire site, as well as to simplify and standardise universal requirements such as processing input forms and dealing with page flows (e.g. Apache Struts[32], JavaServer Faces[22], Jakarta Tapestry[47]).

A web site will be typically made up of a collection of servlets, JSPs, and static resources such as HTML files, images, and other binary files such as Adobe PDFs. Often it will also access a separately-managed database for dynamic information. The collection of resources - servlets, JSPs and other files, but not the database - will be gathered together in a common location as a named *web application* (often abbreviated to *webapp*) on the J2EE server. A single server may host many web applications, and ensures that each operates without interfering with the others - this allows new webapps to be added in without affecting existing ones, so that one server can host many sites ('applications'), e.g. at a web hosting company. Thus, a webapp is typically self-contained, and interaction or communication between webapps is not usually expected or encouraged. As we will see however, the portlet model challenges this convention.

2.5 Virtual Learning/Research Environments

One of Discovery Net's aims is to provide a common environment for performing and sharing the results of research within a "virtual organisation", with an emphasis on ease of use and collaborative work. This is a growing field which is often referred to as "Virtual Research Environments" (VREs) or "Virtual Learning Environments" (VLEs). End-users typically interact with the VRE through a web portal. This topic has been the subject of several recent research programs and workshops, many of which supply useful summaries and examples of projects in the field (e.g. from JISC[20,126,164,166], UK e-Science[153,165] and GGF[129,135]).

There are a number of projects aiming to provide software solutions - usually implemented as academic web portals, for easy access by students and researchers. While the interfaces and presentation chosen vary, these projects have the common theme of trying to make the use and management of distributed, shared resources as transparent and intuitive to use as possible. They also usually include collaboration tools such as forums and chat systems. There is an overlap in functionality between VREs and VLEs: the latter additionally include modules for course content management and publication, and other course-related features such as online tests, timetables and coursework submission systems. However, the research aspect of providing access to computational services is less well supported by generic tools, typically requiring custom programming for each institution.

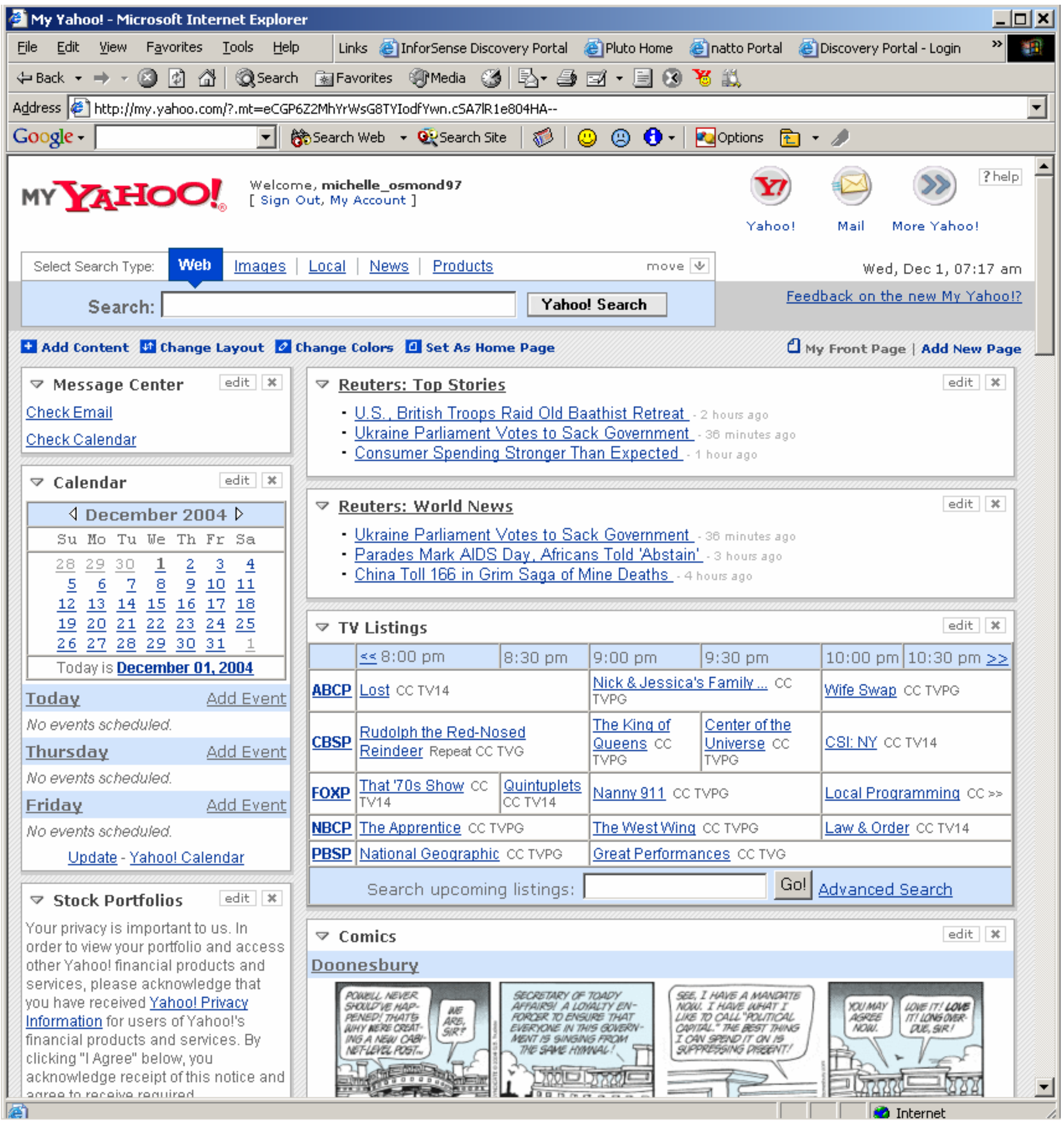
WebCT[85] is an example of a commercial web portal used in educational environments. JA-SIG's uPortal[83] provides a higher-education-focused, JSR-168-compliant portal solution, which has the advantage that third party portlets - for example Grid portlets from the NMI's Open Grid Computing Environments (OGCE) project[63], or indeed Discovery Net portlets - can be plugged into the provided portal with minimal effort. Others, such as Groove Virtual Office[45] (recently acquired by Microsoft) and GRENADE[154] (a grid-enabled Linux KDE desktop), are focusing on tearing down the basic user perception of grid resources as being remote and requiring special client software or web portals to access: instead they provide access to remote data and services by "grid-enabling" the existing desktop environment.

There are many real-world education and research portals in use, both project demonstrators and production portals. The UK's National Grid Service[100] (NGS) allows access to its computational and data services through a web portal running the JSR-168-compliant Stringbeans portal[76]. An alternative portal[104] to the NGS, using the P-GRADE Grid portal[103], provides more advanced tools for job submission and management, again based on JSR-168. The European INFN Production Grid[96], used by several international physics projects, also has a user-friendly portal interface, the GENIUS Grid portal[95,167], which uses a custom architecture with specialised layers to support different projects. On a smaller scale, but perhaps the most similar to Discovery Net's approach as described in this thesis, the myGrid project portal[172] allows users to parameterise and execute workflows and visualise the results, and is built with JSR-168 portlets in GridSphere[78].

2.6 Portals and Portlets

Various forms of 'portal' and website management software emerged early in the development of the Web to meet common requirements of website developers, and the general portal concept is referred to by many names. The use of pre-built software allows developers to get a basic site skeleton up and running quickly. Portals provide a framework allowing modularisation of website content, and the addition of common site functionality such as forums or polls as downloadable components. The structure introduced by the portal makes it easier for multiple developers to work on a large site, and helps in long-term maintenance and updates. The portal software usually also provides site-wide services such as content/page management, user management and security, saving further development time. PHPNuke[4] and PostNuke[70] are examples of commonly-used portals, particularly for small or enthusiast sites. In contrast high-profile sites such as BBC News[88] or Amazon[87] tend to use their own in-house implementations, which still share many features with typical portals, such as the visible modularity in their page designs. News and weblog systems (e.g. Moveable Type[60], Slashcode[73]) are types of cut-down portals which include content management features, specialised for their particular purposes and generally requiring very little configuration to produce a working site. Advanced Content Management Systems such as Typo3[82] can also offer many of the same portal features. In business, "Enterprise Information Portals" - often heavyweight commercial solutions such as Oracle Portal[66] or IBM WebSphere[46] - are used to provide employees with access to information and for direct integration with business process management systems.

In portal terminology, portlets are the code modules which are usually presented to the user as boxes on a web page (Figure 2.3). The content of a portlet usually corresponds to a service or a piece of information which would traditionally have been presented on a single web page, but by 'wrapping' this information or service as a portlet, it can be displayed simultaneously with other portlets on the same page. The page layout may be determined by an administrator or individual user. Thus a single Portal web page may (and probably will) contain many portlets, and/or multiple instances of the same portlet, which may be completely independent or interoperate with each other.



My Yahoo! - Microsoft Internet Explorer

File Edit View Favorites Tools Help | Links InforSense Discovery Portal | Pluto Home | natto Portal | Discovery Portal - Login

Back | Search | Favorites | Media

Address http://my.yahoo.com/?.mt=eCGP6Z2MhYrWsG8TYIodfYwn.cSA7lR1e804HA--

Google | Search Web | Search Site | Options

MY YAHOO!

Welcome, **michelle_osmond97**
[Sign Out, My Account]

Yahoo! | Mail | More Yahoo! | ? help

Select Search Type: **Web** | Images | Local | News | Products | move

Wed, Dec 1, 07:17 am

Search: [ ] Yahoo! Search

Feedback on the new My Yahoo!?

Add Content | **Change Layout** | **Change Colors** | **Set As Home Page** | **My Front Page** | **Add New Page**

Message Center edit ×

Check Email
Check Calendar

Calendar edit ×

December 2004

Su	Mo	Tu	We	Th	Fr	Sa
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1

Today is **December 01, 2004**

Today Add Event
No events scheduled.
Thursday Add Event
No events scheduled.
Friday Add Event
No events scheduled.
Update - Yahoo! Calendar

Stock Portfolios edit ×

Your privacy is important to us. In order to view your portfolio and access other Yahoo! financial products and services, please acknowledge that you have received Yahoo! Privacy Information for users of Yahoo!'s financial products and services. By clicking "I Agree" below, you acknowledge receipt of this notice and agree to receive required

Reuters: Top Stories edit ×

- U.S., British Troops Raid Old Baathist Retreat - 2 hours ago
- Ukraine Parliament Votes to Sack Government - 36 minutes ago
- Consumer Spending Stronger Than Expected - 1 hour ago

Reuters: World News edit ×

- Ukraine Parliament Votes to Sack Government - 36 minutes ago
- Parades Mark AIDS Day, Africans Told 'Abstain' - 3 hours ago
- China Toll 166 in Grim Saga of Mine Deaths - 4 hours ago

TV Listings edit ×

	<< 8:00 pm	8:30 pm	9:00 pm	9:30 pm	10:00 pm	10:30 pm >>
ABCP	Lost CC TV14		Nick & Jessica's Family ... CC TVPG		Wife Swap CC TVPG	
CBSP	Rudolph the Red-Nosed Reindeer Repeat CC TVG		The King of Queens CC TVPG	Center of the Universe CC TVPG	CSI: NY CC TV14	
FOXP	That '70s Show CC TV14	Quintuplets CC TV14	Nanny 911 CC TVPG		Local Programming CC >>	
NBCP	The Apprentice CC TVPG		The West Wing CC TVPG		Law & Order CC TV14	
PBSP	National Geographic CC TVPG		Great Performances CC TVG			

Search upcoming listings: [ ] Go! Advanced Search

Comics edit ×

Doonesbury

POWELL NEVER SHOULD'VE HAPPENED! THAT'S WHY WE'RE CREATING A NEW CABINET-LEVEL POST...
WE ARE, SIR?
SECRETARY OF TOADY AFFAIRS! A LOYALTY ENFORCER TO ENSURE THAT EVERYONE IN THIS GOVERNMENT IS SINGING FROM THE SAME HYMNAL!
SEE, I HAVE A MANDATE NOW. I HAVE WHAT I LIKE TO CALL "POLITICAL CAPITAL." THE BEST THING I CAN SPEND IT ON IS SUPPRESSING DISSENT!
YOU MAY AGREE NOW.
LOVE IT! LOVE IT! LONG OVERDUE, SIR!

Internet

Figure 2.3: "My Yahoo" (2004) is a user-customisable portal with many available portlets including headlines, weather, calendar, movies and TV listings.

2.6.1 Portlet Standards

From 2000 onwards, portals became popular for use in company intranets, acting as a central point for aggregation of information, communication, and integration of business services. Many companies, including IBM, Oracle, Microsoft, Plumtree, Vignette, SAP and BEA offered their own 'Enterprise' portal software, providing similar features but with proprietary APIs for portal and portlet development. Thus, portlets developed for one portal would have to be rewritten if it was later decided to migrate to another system.

The Java portlet standard, JSR-168[16] (released in October 2003), was defined with the aim that developers could write reusable Java portlets that could be used in any standard-compliant Portal server. This would lessen the cost of moving to a different Portal implementation, and also allow generic portlets (e.g. a webmail, chat or forum portlet) to be developed once and then used on any portal. JSR-168 standardises features which were already common in many pre-existing portals, but cannot include every feature that these portals offered. This may have been one reason for the relatively slow release of JSR-168 compliant versions of the more established portals; however by the end of 2005, most current versions of the portals mentioned included JSR-168 support, with the notable exception of Microsoft SharePoint Portal Server[59].

JSR-168 specifies a standard way for programming portlets in Java. The WSRP (Web Services for Remote Portlets) standard[14], approved as an OASIS standard in August 2003, complements JSR-168: it defines web service interfaces so that a portlet may be implemented as a web service which a compliant portal server can access. A WSRP portlet may thus be remote from the portal server using it, and may be written in any programming language. WSRP's goal is to allow interaction with presentation-oriented, interactive web services: i.e. web services which provide their own human-oriented user interface (the visible portlet). WSRP portlets and JSR-168 portlets effectively offer the same features; JSR-168 may be seen as a Java-specific implementation of WSRP features. Some portals, such as Oracle Portal, in fact support only WSRP portlets, but also provide internal tools to wrap and expose JSR-168 portlets as WSRP web services. Figure 2.4 gives an overview of this architecture.

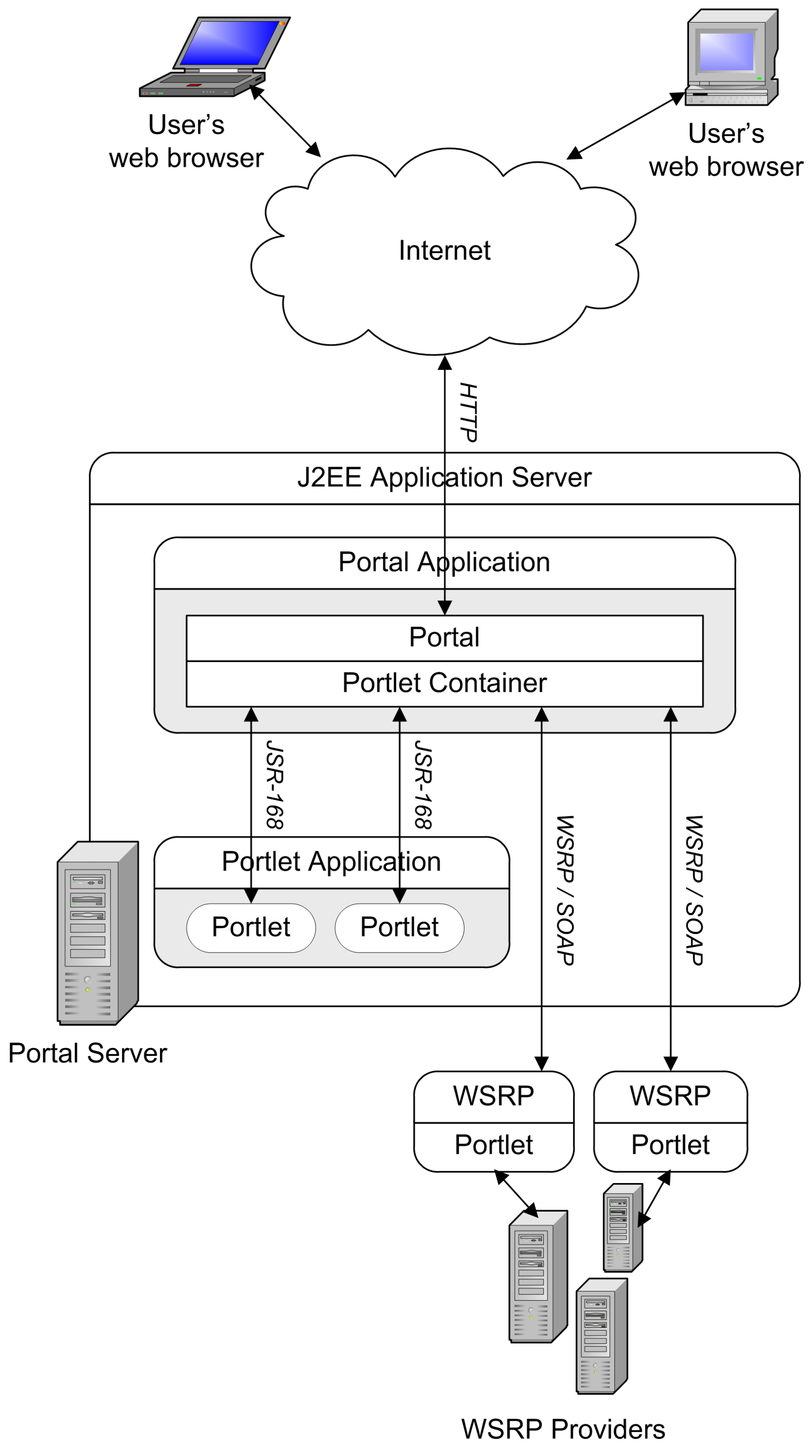


Figure 2.4: Portal architecture overview.

2.6.1.1 Java Portlet Standard (JSR-168)

JSR-168 portals are implemented as Java web applications, hosted on Java application servers (servlet containers) such as Apache Tomcat[33] or JBoss Application Server[50]. JSR-168 portlets are typically placed in their own web applications ("portlet applications"), with a portlet deployment descriptor file `portlet.xml` which provides an index of the available portlets. The portal application thus needs to be able to inspect and access the other web applications on the same server which contain portlets, and this is usually effected as part of an initial portal-specific deployment process for each portlet application (e.g. by inserting a servlet from the portal in each portlet application).

In JSR-168, the functionality of a portal is abstracted into the *portlet container* and the *portal server* (Figure 2.4). The portlet container manages the portlets using the interfaces defined in JSR-168, and makes them available to the portal server. The portal server provides the actual implementation of portal features and services such as page aggregation, user management and preference storage, which is outside the scope of JSR-168 and therefore may be differently implemented in different portals.

Apache Pluto[30] is the JSR-168 reference implementation of a portlet container, and also includes a portal server with minimal features (intended for testing, not production use). Other JSR-168/WSRP portals include Apache Jetspeed 1.6[28], Jetspeed 2[29], Stringbeans[76], eXo[41], Liferay[52], GridSphere[78], JBoss Portal[51], IBM WebSphere[46], and Oracle Portal[66].

Figure 2.5 shows the terminology used in JSR-168 to describe the components of a portal page. A portlet window is a particular instantiation of a portlet defined in the portlet deployment descriptor. Multiple portlet windows may be based upon the same portlet entry in the `portlet.xml`. Usually, each portlet window is treated as an independent module, and has its own private scope in the user session (referred to as the “Portlet scope”) to save data. Each portlet window is responsible for generating its own fragment of the page, and the portal deals with aggregation and layout of the fragments, delivering the final page to the user's browser.

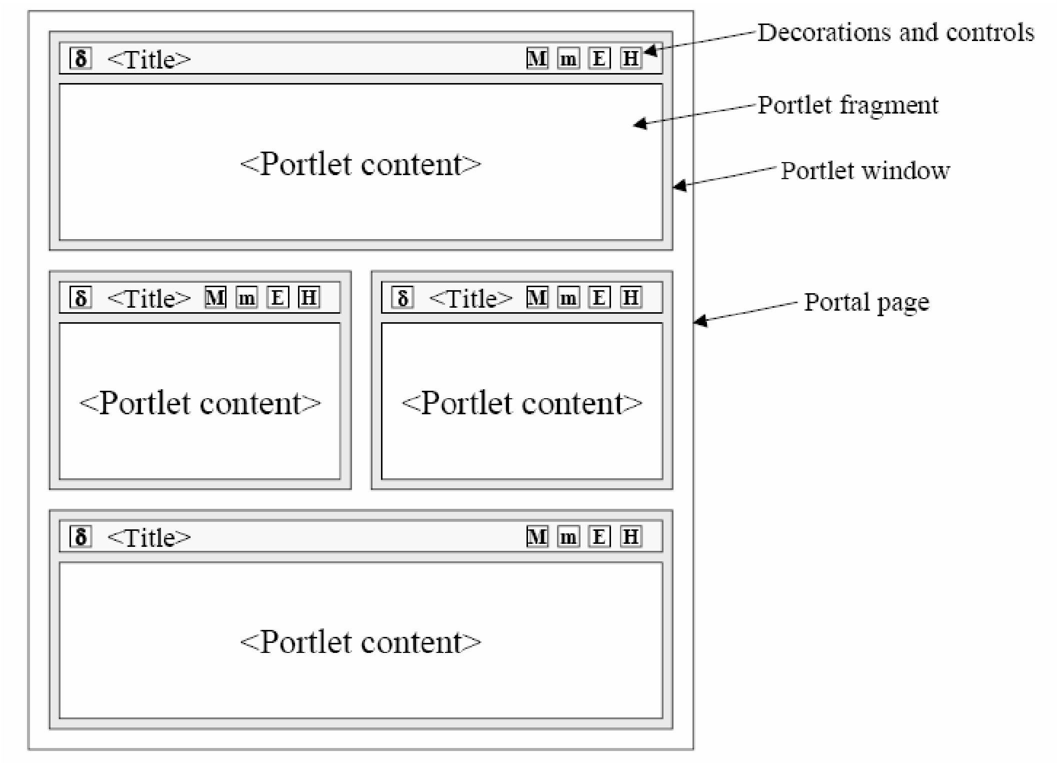


Figure 2.5: Diagram from JSR-168 specification[16] (Figure 4-1 "Elements of a Portal Page")

Portlets within the same portlet application can share data using the "Application scope" of the user session. Portlets in different portlet applications cannot communicate easily, and so it is generally recommended that related portlets are packaged in the same application.

The Model-View-Controller architecture[3] is commonly used in web applications to separate the data model, the active control code, and the page display code. JSR-168 defines the portlet lifecycle in a way to explicitly support and encourage this coding practice. When a user clicks on a link in a portlet, or submits a form, the browser sends a new request to the portal. The portal examines the request, which may be one of two types: a *Render* request or an *Action* request. If it is an Action request, the portal sets off the originating portlet's Action phase, invoking the portlet's `processAction` method, and once that has completed, it sets off (possibly in parallel, using threads) the Render phase of all portlets on the current page, to generate portlet display fragments (invoking their `render` methods). The portal gathers together these fragments to make up the whole page, and sends the response back to the client browser, which displays it to the user (Figure 2.6). Thus, the originating portlet has a chance to perform operations and change portlet state in the Action phase, before the Render phase occurs. On the other hand, if the incoming request is a Render request, the Action phase is bypassed and the portal simply re-renders all the portlets on the page.

There are several further differences in developing portlets when compared with Java servlets, as there are some things that must be delegated to the portal. Stefan Hepper's "Best Practices" document[173] describes these well. For example, on a normal dynamic web site, a user may click on a link or form button which sends *query parameters* to the target page (e.g. in the URL `http://www.google.co.uk/search?q=portlets`, the query parameter 'q' is sent with the value 'portlets'), which will be picked up and acted upon by the page when it re-renders. A portlet uses a similar approach, but must delegate the generation of the target URL to the portal, so that the portal can encode the query parameters alongside other information relating to page state and other portlets. As part of this delegation, the portlet is also able to specify whether the target URL should result in an Action or a Render request. When the portlet window re-renders, the portal will ensure that the portlet window's own query parameters from the target URL are made available to it. The portal also provides access to portlet features and services, such as a method to retrieve the portlet instance's initialisation parameters (which are specified in the `portlet.xml`).

One particularly useful service provided to portlets is *portlet preferences*. These are user-specific settings for a portlet window that are persisted by the portal across browser sessions. By editing their preferences, a user can personalise a portlet on their version of the home page - for example, setting a home town for a Weather portlet. The method of persistence (e.g. file store, database) is dependent on the portal implementation and is often configurable by the portal administrator; JSR-168 provides a simple API allowing portlet code to save and retrieve preferences, which is independent of the actual persistence method used.

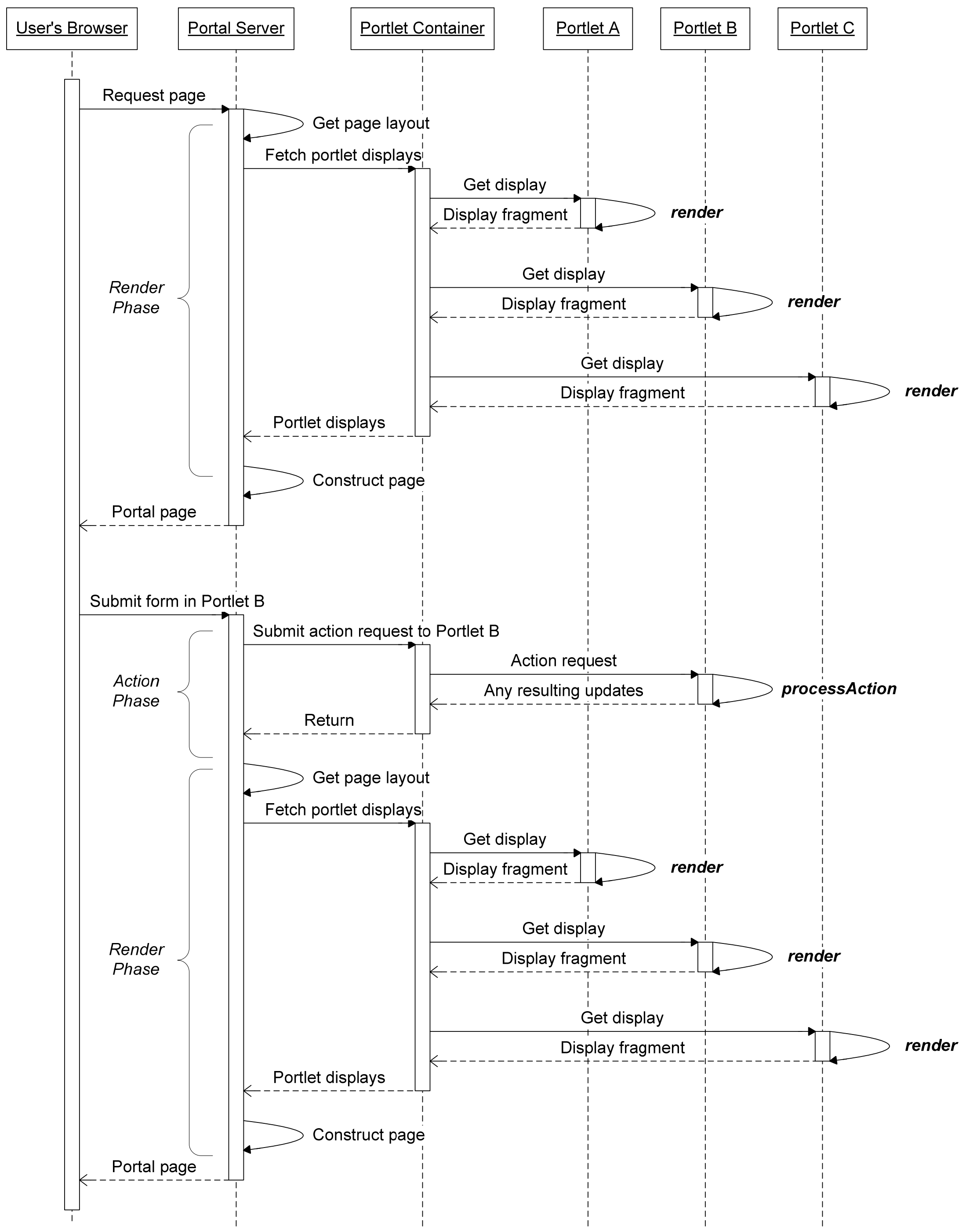


Figure 2.6: Example portal page request sequence. A user first visits a portal page, sending a Render request to the portal. Next, they submit a form from one of the portlets displayed. The portal processes the resulting Action request, first allowing the targeted portlet to process the form submission, then re-rendering the page with all its portlets to return to the user.

JSR-168 does not dictate the method of page management, layout or aggregation - that is left to the implementers of the portal server to decide. Portals always provide some way of creating new pages without code, but the exact method varies from editing XML configuration files to drag-and-drop graphical web interfaces on the running portal itself. Certain aspects of the portlet rendering process are mentioned in JSR-168 but are optional - such as caching, or parallel (multi-threaded) rendering of portlet windows.

Some features such as inter-portlet communication and portlet filters (which would provide similar functionality to servlet filters[13]) were left out of JSR-168, although they are often available in other Portal implementations. The limitations of the JSR-168 specification are discussed in detail in Chapter 7.

2.6.1.2 Web Services for Remote Portlets (WSRP)

As discussed earlier, the OASIS standard Web Services for Remote Portlets (WSRP)[14] is a parallel specification to JSR-168 which defines a way of accessing a portlet through web services. In WSRP terminology, the host of the WSRP web services is the *Provider*, and the direct user of the web services is the *Consumer*. Thus in the context of WSRP, the Consumer is the portal server - not the browser client or the human end user.

A WSRP Provider may host any number of WSRP portlets. A portal which supports WSRP acts as translator, aggregator, and middle-man, accessing the remote WSRP Provider to retrieve GUI fragments for presentation to the user, and then returning the user's response to the WSRP Provider for processing. As web services use language-independent protocols, WSRP portlets can be implemented in programming languages other than Java, as long as they are accessible through a WSRP interface. Some JSR-168 portals additionally provide WSRP export of their hosted JSR-168 portlets, so a Java JSR-168 → WSRP conversion is generally quite easy (this is necessary for portals which have WSRP, but not direct JSR-168 support). Thus, through WSRP, a standard mechanism exists for providing platform-neutral web service (portlet) GUIs to remote Consumers (portals).

2.6.1.3 The relationship between JSR-168 Portlets and Java Servlets

JSR-168 portlets are very similar to servlets, but differ in numerous small and significant ways[173]. A portlet can perhaps be considered as a version of a servlet that has been modified to permit modularisation and aggregation of multiple portlets on a page; to do this, some new features have been added and some existing ones removed. Many of the servlet API interfaces and classes have portlet equivalents: e.g. instead of `doGet` and `doPost`, portlets use `render` and `processAction`; instead of an `HttpSession`, portlets have a `PortletSession`; the `PortletRequest` and `PortletResponse` interfaces closely mimic the `HttpServletRequest` and `HttpServletResponse`. However the Portlet versions of servlet classes do not actually inherit from or implement the Servlet classes or interfaces.

Portlets are deployed to a servlet container or J2EE server in a web application with an additional configuration file: the portlet deployment descriptor `portlet.xml`, which lists and parameterises the available portlets. A single such "portlet application" may contain any number of portlets, and indeed any portlets which are intended to work together should be packaged together in the same portlet app. A portlet application can contain all the types of resources found in a normal web application, and portlets should be able to make use of these resources (files, servlets, JSPs). The specific procedure for installing portlets to a given portal may vary, but in the end - perhaps after the portal has checked and tweaked some configuration files - the portlet application will be deployed as a webapp.

A portlet is frequently viewed as a mini-application in its own right. It therefore requires some degree of insulation from the other portlets in the same application, or on the same page. One way in which the portlet specification supports this is by extending the existing servlet concept of the session, splitting the `PortletSession` into two scopes: the `PORTLET_SCOPE` session, which is only visible to that portlet instance, and the `APPLICATION_SCOPE` session, which is visible to all portlets in the same portlet application (Figure 2.7). The portal also scopes portlet 'render' attributes (values which are to be passed on from the portlet request-processing code to an included JSP for display), so that a render attribute set by one portlet on a page is only seen by its own included JSPs, and not by the JSPs of other portlets on the page.

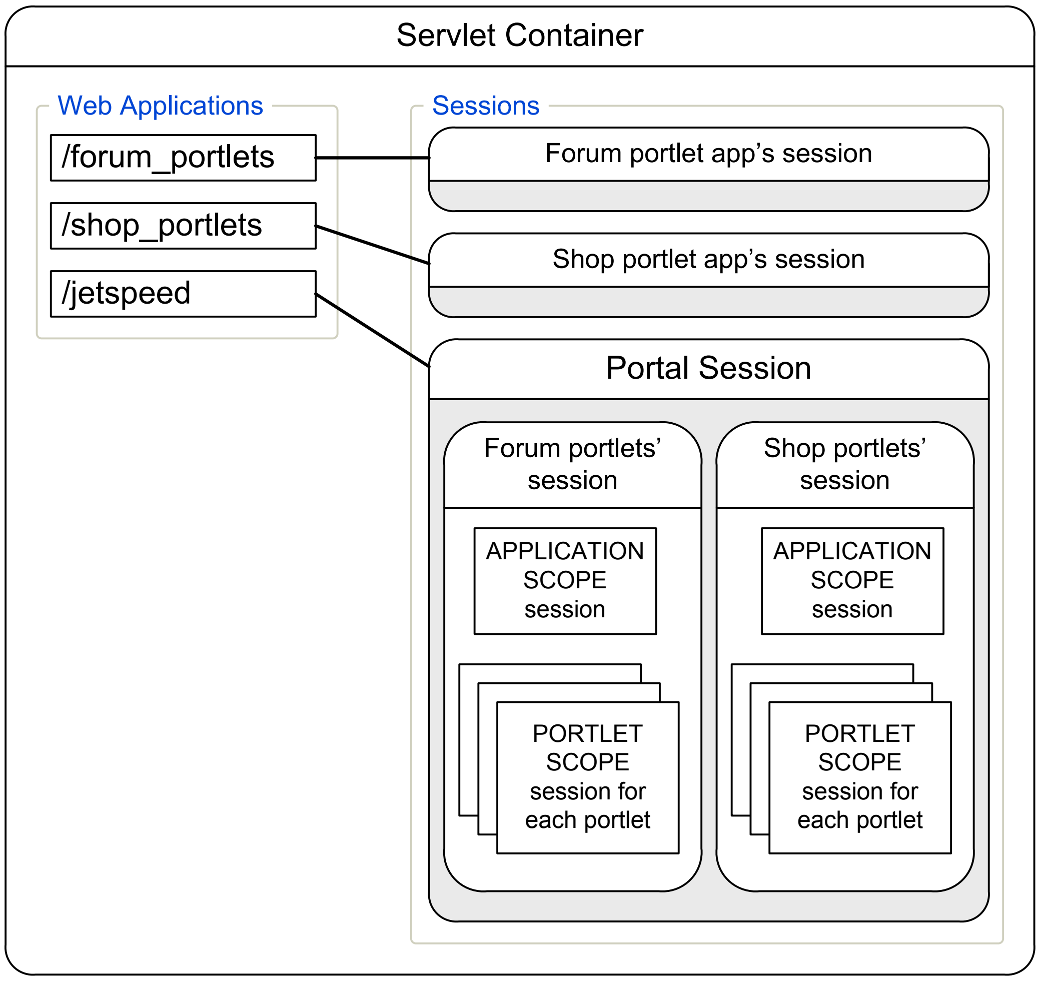

Servlet Container
Web Applications
/forum_portlets
/shop_portlets
/jetspeed
Sessions
Forum portlet app's session
Shop portlet app's session
Portal Session
Forum portlets' session
Shop portlets' session
APPLICATION SCOPE session
APPLICATION SCOPE session
PORTLET SCOPE session for each portlet
PORTLET SCOPE session for each portlet

Figure 2.7: Portlet applications, and the Portal itself (here, Jetspeed Portal), are all deployed as web applications in a servlet container. A portlet application may contain many portlets. Each portlet has its own private session area, but can also access a shared session area visible to all portlets in the same application.

Thus, developing a Portlet is a similar, but somewhat different and parallel, procedure to developing a servlet. A portlet application is an enhanced webapp which may include both portlets and servlets, and portlets can - and often do - make use of servlets, most commonly for generating images or serving other binary files to the user.

The Portal itself is also deployed in its own webapp in the servlet container. As noted before, webapps are kept separate by the servlet container, and this must be worked around by the Portal which needs to be able to access portlets in their own webapps. As a result there is usually some extra configuration involved in registering a new portlet application with a Portal - and often a requirement to put Portal libraries in a common/shared directory on the server.

A full Portal deployment therefore includes a single Portal webapp plus any number of portlet webapps, each of which contain one or more portlets (Figure 2.8).

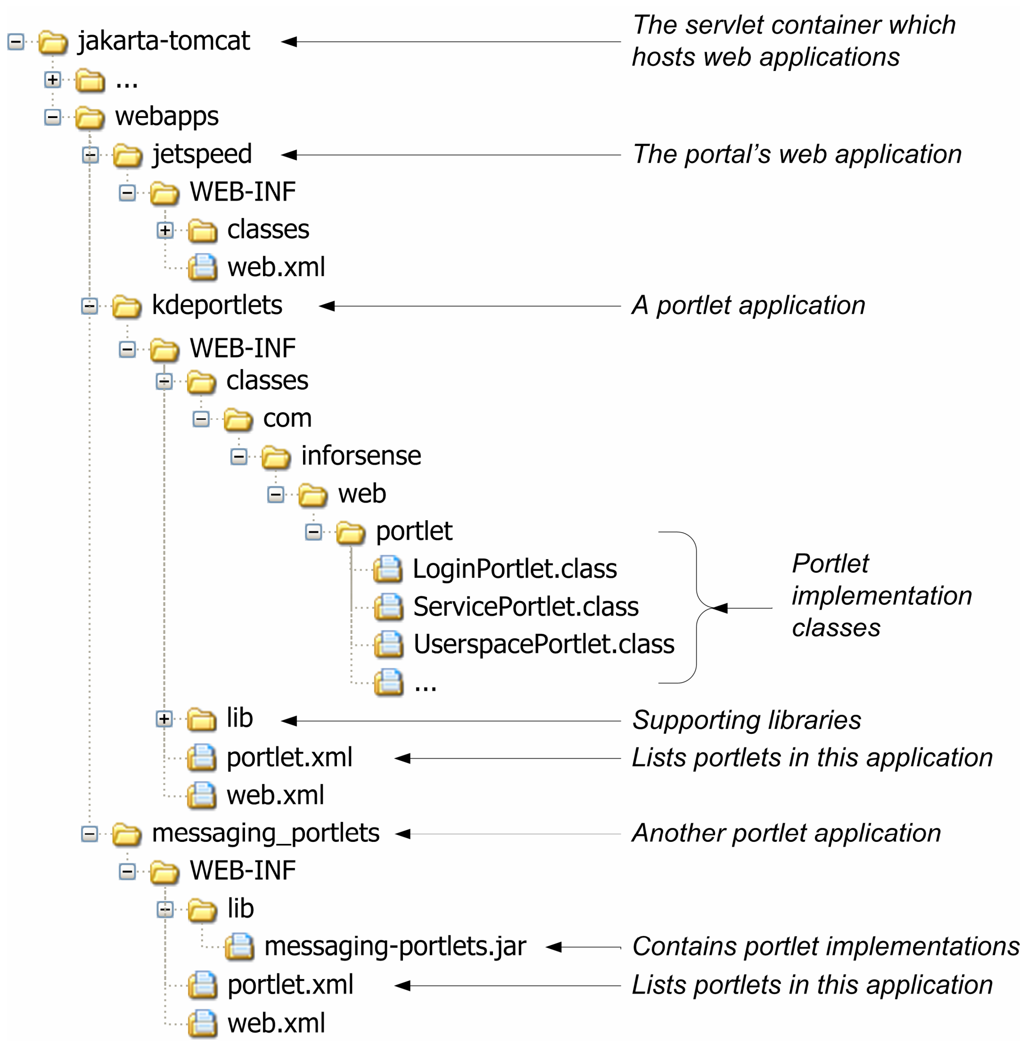


Figure 2.8: Example directory tree showing the location of the portal and portlets.

2.6.1.4 Future Portlet Standards

The next versions of both the Java portlet standard and WSRP are under development.

The Portlet Specification 2.0 is being developed as JSR-286[106]; the expert group was formed in December 2005 and as of July 2007 has reached the Public Review stage.

WSRP 2.0[14] has been in development since late 2004; it aims to coordinate with JSR-286, and will also soon begin its Public Review stage.

2.7 Conclusion

This chapter has introduced the main technologies used for providing computational Web and Grid services and for developing web portals. Following an overview of portal usage, we have described the features of the current Portlet Standard (JSR-168) and the Web Services for Remote Portlets (WSRP) v1.0 specification. This provides a basis for discussing the design and development of analytical portals using JSR-168 portlets in the following chapters.

Chapter 3. Analytical Portal Design

In this chapter we consider the use of a "web portal" as an interface for interactive analytics, allowing users to visualise data and perform analysis using remote services. We discuss how the Discovery Net web portal was developed as a set of interacting portlets, and how our style of implementation and usage scenarios affected our choice of portal implementation. Finally, we present a concrete example illustrating the points made in this chapter: a demonstration of a retail portal built using a combination of Oracle and Discovery Net portlets.

3.1 General advantages of Web Portals

A website typically contains multiple sections, each providing different information or tools to the end user. Modularising the back-end code makes it easier for a developer to concentrate on one aspect of the site, allowing multiple developers to work on different sections simultaneously, and reducing the learning curve for new developers. Modularisation encourages a design where the different modules are, if not completely independent, only loosely coupled, so that changes to one module have minimal impact (requiring updates or fixes) on other parts of the site.

Using *portal software* as the framework for a website aids and encourages such modular design, by providing a structure and interfaces to which modules can be developed, plugged in, and aggregated into web pages. As discussed in the Introduction, there are many portal implementations available, using a variety of approaches and back-end technologies (e.g. Perl[81], PHP[69], ASP[34], Java Servlets[13]), and the Java Portlet specification (JSR-168)[16] is a relative newcomer in the portal crowd. As many of the older Java portals (such as IBM WebSphere[46] and Oracle Portal[66]) offer a more extensive feature set, JSR-168's main attraction is that it is an established Java standard.

This promises that JSR-168 portlets (the web modules) will be portable between different JSR-168 portal implementations, and provides some assurance that support will continue for several years in the future. Also, the fact that the technology used is Java will allow easy integration with existing applications based upon J2EE[11].

The JSR-168 community is an active one[114,161,49]. Since the specification's final release in October 2003, many JSR-168-compliant portals have appeared, varying from barebones open-source (e.g. Apache Pluto[30]) to full-featured, heavyweight commercial offerings (e.g. Oracle Portal). Website developers also have the option of importing existing third-party portlets to add functionality with minimal development effort. A small number of third-party JSR-168 portlets[15,42,99] are available for free or to purchase (e.g. for integrating news, forums or email to a portal) and most portal implementations also bundle their own selection of portlets which may be used or ignored as the site developer chooses.

A Portal typically provides a number of services which can save considerable development effort. Most importantly, the portal will provide a mechanism for constructing web pages by aggregating portlets. This is of course a critical feature of the portal, but the separation of page layout and navigation from the development of portlets also allows further division of responsibilities within a large web team. The method of constructing pages is different in each portal, from manually editing configuration files before the server is started, through to a drag-and-drop GUI on the portal itself at runtime. A strong driver for using portals is personalisation of pages, usually by allowing users to modify which portlets appear on their "home page", and to configure them with custom settings. Further, some sites may allow users to create their own new pages, effectively building their own sub-site using the portlets provided. Portals which allow users to personalise their pages must usually include a user system, which may also include a Roles or Groups model. User management and role-based access to protected pages or portlets may therefore also be counted among the services potentially provided by the Portal.

In summary, the key benefits from using a Portal, and in particular JSR-168, are:

- Modularisation of website content; separation of site structural design from portlet development
- Generic services provided and managed by the Portal, e.g. security, user management, page management
- Ability to host portlets in any JSR-168-supporting Portal
    - Choice of Portal implementation
    - Easily share portlets for hosting in other portals
    - Easily plug-in 3rd-party JSR-168 portlets

However, as will be described in detail in Chapter 7, there are several limitations involved in developing JSR-168 portlets. Many of these issues have workarounds (of varying effectiveness), which are also discussed in that chapter.

The next version of the portlet specification (JSR-286) is currently in progress; this continuing development, building upon JSR-168, should act as a further encouragement to support the JSR-168 standard for developers of both portlets and portals, as later migration of code from JSR-168 to JSR-286 is likely to be easier than migration from a proprietary portlet interface. The new standard will address many of the features lacking from the original specification, and hopefully also some of the additional limitations described later.

3.2 Requirements of an Analytical Portal

Most web portals today focus on presenting information to the end user. While many also provide interactive features such as facilities for users to contribute content (e.g. through polls, comments, reviews, forums), or initiate business processes (most commonly, buying from a shop), the core requirement is still the provision of information. Thus "Content Management Systems" (CMS) have been developed for the storage, maintenance, aggregation and presentation of information. CMSs often include standalone client applications for performing administrative tasks, although some (e.g. Typo3[82]) have implemented these as web-based applications. An alternative approach for managing content is through *wiki* software[112], which allow web-based content editing, potentially by any user of the site. Many Portal solutions provide ways of embedding both existing CMSs and wikis.

In contrast to other portals, the focus of analytical portals is to provide a wide variety of rich web interfaces to an ever-changing pool of available services, making use of at least one robust back-end system for managing service execution (Table 3.1). Data management is also a required capability, but this time from the point of view of using data as input to services and storing/accessing service results, and the portal may delegate this function to the back-end analysis engine. Conventional content management, for managing the content of static pages on the site, is still necessary but can be assumed to be a standard service that will be available in most modern portals.

A common feature in portals is a web-based front end for managing and composing portal pages. This is typically mainly used by the portal administrator for creating the static parts of a website. However, analytical portals can make greater use of this feature by additionally allowing their users to create their own personalised sub-sites (of multiple pages) using the different analysis portlets provided by the portal. There may also be an intermediate level of users or administrators who can create 'group' or 'role'-based pages which are available to a group of users (e.g. all participants in a particular project).

Therefore, we envision an analytical portal which does not need to provide a pre-designed website with custom pages for every possible tool (and associated navigational systems), but instead can allow users to build their own version of the site, focusing on providing a library of functional portlets which each user can combine and personalise in the way most useful to them. The personalised page creation system thus gains greater significance for analytical portals than in other portals, becoming a core part of how the portal is used, rather than a peripheral gimmick.

Traditional Portal	Analytical Portal
Focus on providing information	Focus on providing data and interactive analysis tools
Document management and presentation	Data storage and provision, Application hosting and execution
Static page/site design	User-customisable page contents User-created new pages
Design main features from the beginning, and launch completed site	Hosted applications may be changed and added to at any time
Limited/controlled user-generated content, e.g. forums, comments	High proportion of user generated content, up to and including the analysis applications themselves.

Table 3.1: Feature Comparison of Traditional and Analytical Portals.

3.3 Functional requirements for the Discovery Net Portal

Our concrete example of an analytical portal is the Web Portal for Discovery Net's data mining engine, which executes Discovery Net workflows.

The web portal needs to provide access to a subset of Discovery Net's features: primarily, it must provide a web interface to parameterise workflows and submit them for execution. It does not need to include full workflow editing capabilities, which is already catered for by the Discovery Net Java client.

The features which must be provided by the Discovery Net web portal are:

- Discovery Net Userspace
  - data storage, management and retrieval
- Discovery Net Services (deployed workflows)

    - inspection of service metadata and workflow details
    - parameterisation through a user-friendly web form (including use of applets for inputting special types of parameters, e.g. molecule sketch)
    - ability to save/load 'bookmarks' of frequently used parameter settings
    - execution through the web form
    - task monitoring and management
    - display and storage of service results
    - ability to pass on a service result from one service as an input to another
- Discovery Net Server Administrative functions
- Help pages

In addition, related to the Portal implementation:

- A web-based interface for creating new portal pages is essential so that users can create their own personalised pages.

3.4 Non-functional requirements for the Discovery Net Portal

Usability: Users should be able to discover how to manage data in their userspace, and parameterise and execute deployed services, with no or minimal training required. Some training or documentation may be necessary to enable users to create their own personalised pages.

Error handling: Errors resulting from service execution must be reported in sufficient detail for the user to understand what went wrong (whenever possible).

Security: Users must log in to the portal before accessing any Discovery Net userspace or service functionality. Roles must be used to determine which parts of the userspace and which services are accessible to different users.

User/Role Management: The portal's user system must be integrated with Discovery Net's user/group system, so that the two user registries do not need to be manually synchronised.

Robustness: The portal runs on the same application server as the main Discovery Net engine, and so must be robust enough in operation to survive weeks and months of continual uptime. Resources must be managed and released efficiently.

Internationalisation: It must be possible to fully internationalise the portal pages.

3.5 Discovery Net portlets

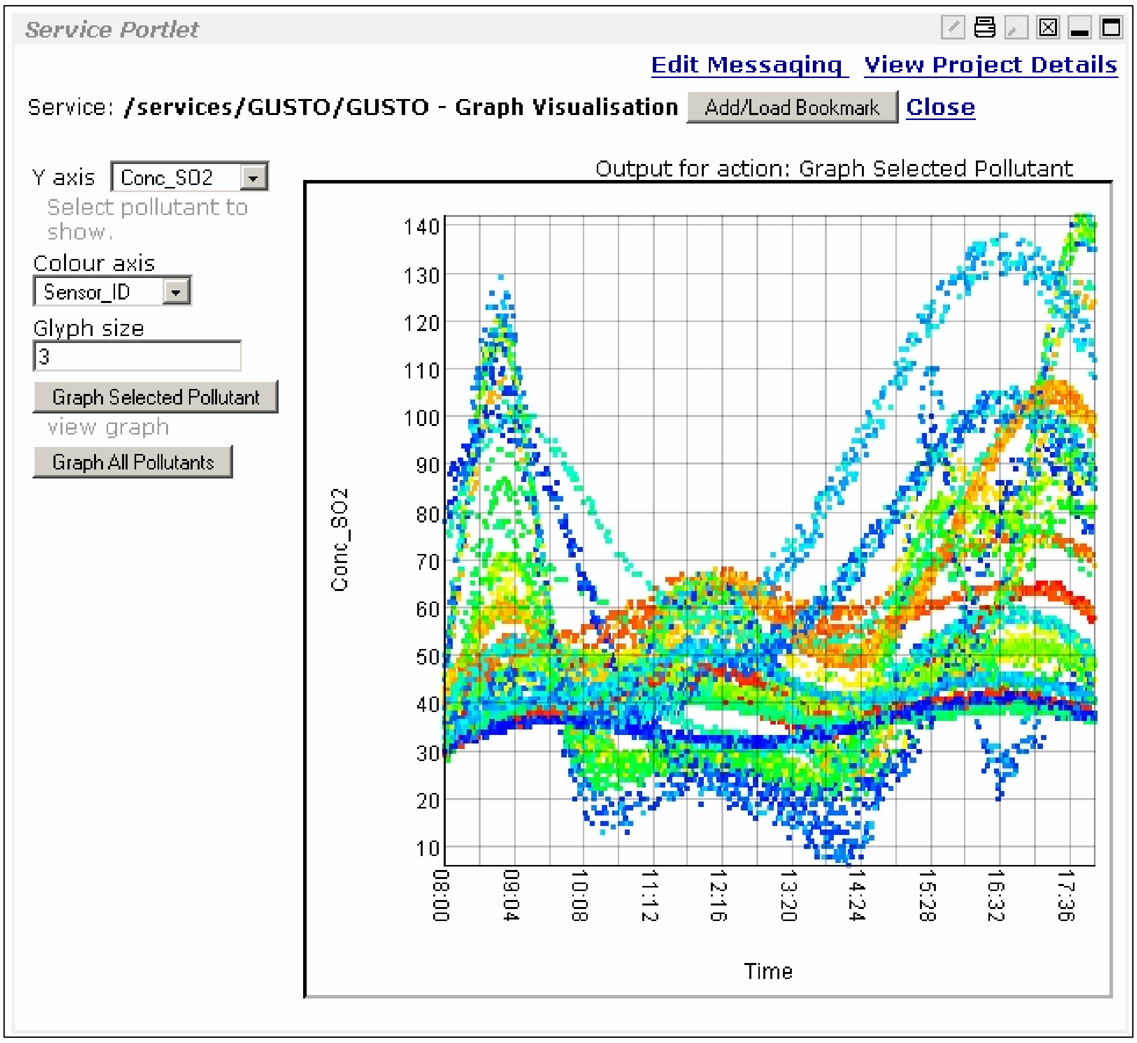
The original Discovery Net web portal already provides access to most of the required features, as a fixed set of web pages using servlets organised with Apache Struts[32]. However, sites based upon this architecture are not well suited to conversion to portlets, except as a monolithic whole; an entire site can be embedded into a page using an IFRAME, or utilities such as the Apache Portals' Struts Bridge[31] can wrap an existing Struts application as a single portlet. These approaches are suitable for quickly integrating an existing application into a portal site - particularly when there are many such applications to convert.

However, we wanted to split up the different types of Discovery Net functionality into separate portlets, taking advantage of their component-like nature to provide users with new ways of accessing Discovery Net services. Of particular interest was the new possibility of having multiple services on the same page, and allowing results to be passed from one service as input to the next.

Therefore the first priority was in identifying suitable portlets: sections of functionality which were loosely coupled and reusable. These broadly correspond to the pages in the original web portal, but some aspects (such as the different userspace functions) have been split up further, for clearer control of dependencies between portlets, and flexibility in arranging page content.

3.5.1 The Service Portlet

This portlet is the key to providing the Analytical Portal. It displays a service form interface for a selected service instance, allowing the user to parameterise it, submit it for execution, and view the results (Figure 3.1). The selected portlet is discovered either from an input message from another portlet (e.g. the Service Index portlet), or from explicit configuration by the user in this portlet's Edit mode. Such direct configuration to select a service is saved in the user's preferences for that portlet window, so the portlet will continue to display the selected service on future visits. Multiple Service portlets can thus be added to pages and separately configured to create personalised 'dashboards'.



Service: **/services/GUSTO/GUSTO - Graph Visualisation**

Y axis: Conc_SO2
Select pollutant to show.

Colour axis: Sensor_ID

Glyph size: 3

Output for action: Graph Selected Pollutant

Figure 3.1: Service Portlet. In the default layout, the parameters are input on the left and the result shown on the right.

Service 'bookmarks' of a service's frequently used parameter settings can be both saved and loaded. Service bookmarks are stored as files in the userspace, so can be shared with other users simply by moving them to a shared userspace directory.

The portlet also allows the user to view details of the underlying workflow. This has the same effect as viewing the service in the Userspace Item Viewer portlet (Section 3.5.2.4).

Finally, the advanced user can configure the portlet to pass on the result(s) of the service to be used as input parameters in another Service portlet. This is implemented using the IPC library and is configurable in the portlet's Edit mode.

The design of this portlet is key to the flexibility of our analysis portal. A single Service portlet implementation acts as the interface to any deployed Discovery Net service (Figure 3.2). This is in contrast to the more common approach of providing a bespoke web interface - either with static web pages or a custom portlet - for each service. This usual approach has the disadvantages of low flexibility and extremely high maintenance on the part of the portlet developer (who has to create a custom interface for each service) and for the portal server administrator (who has to install the new pages/portlets, and keep them accessible and up to date as the pool of available services changes). It does however allow for completely customised web interfaces, as each can be hand-tailored to the service's requirements, both in inputting parameters and the treatment of results.

On the other hand, our approach of using a single Service portlet does not require any ongoing maintenance or manual intervention by either the portlet developer or the portal administrator, as the portlet can show any Service without further custom programming, and is always up-to-date with the list of services available. This is made possible by introducing standard ways of describing web interfaces to Discovery Net services, which the Service portlet can then transform to generate the final HTML code. We discuss this approach to the problem of dynamically generating rich, user-friendly service interfaces in Chapter 4.

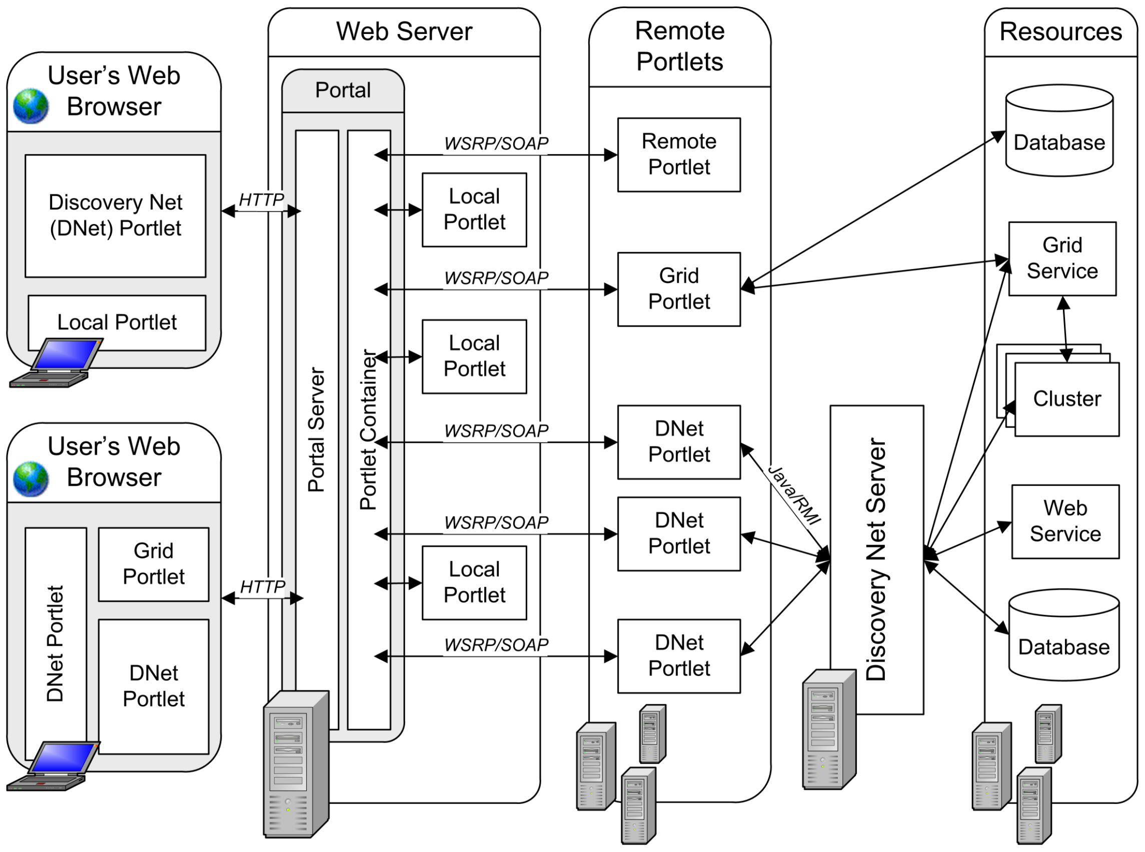


Figure 3.2: Discovery Net Portal architecture. A single Discovery Net Service portlet can be configured to provide an interface for any Service (deployed workflow) on the Discovery Net Server. The Service workflow may access any computational resource, such as Grid Services, clusters or databases.

3.5.2 Supporting Portlets

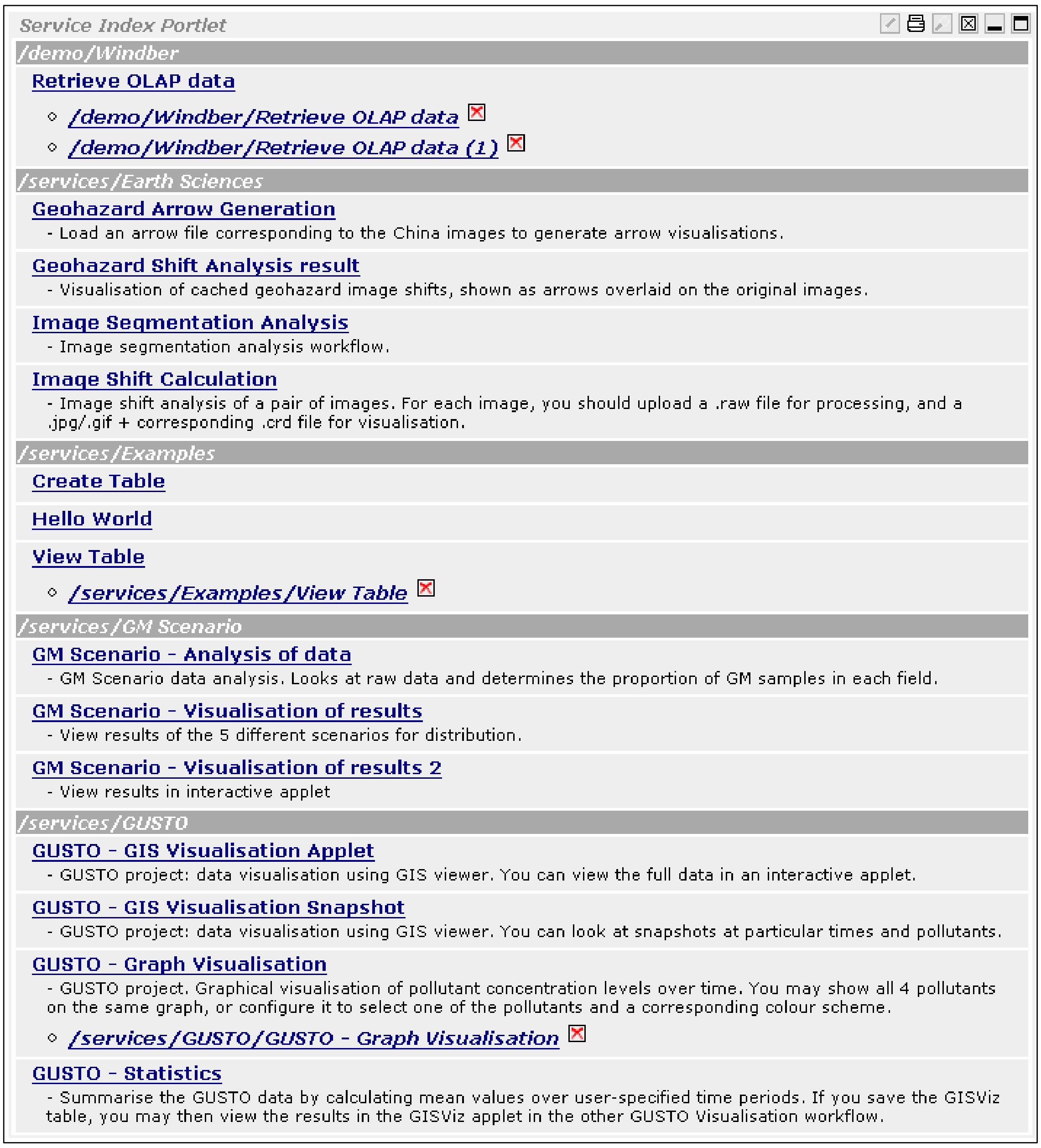
3.5.2.1 Service Index Portlet

This portlet lists the deployed Discovery Net services available to the current user (Figure 3.3), and allows the user to select a service for use in other portlets..

The services are grouped by location in the userspace. In future, they might alternatively be sorted or searched using service metadata, but such metadata is not currently available.

By default, all visible services are shown. The portlet can be configured in edit mode to show only services under a specified directory, which is helpful when building focused portal pages for particular projects.

For a selected service to be parameterised and executed, a temporary (per web session) *service instance* must first be created. This service instance is used to maintain state on the server, including caching of intermediate results. Thus the Service Index portlet creates a new instance of the requested service and then publishes a message for other portlets identifying the selected service instance. Later, whenever the service index is re-rendered, it will also indicate the existence of service instances alongside their parent service - these instances may from then on also be directly selected for use in other portlets.



Service Index Portlet

/demo/Windber

Retrieve OLAP data

- /demo/Windber/Retrieve OLAP data
- /demo/Windber/Retrieve OLAP data (1)

/services/Earth Sciences

Geohazard Arrow Generation
- Load an arrow file corresponding to the China images to generate arrow visualisations.

Geohazard Shift Analysis result
- Visualisation of cached geohazard image shifts, shown as arrows overlaid on the original images.

Image Segmentation Analysis
- Image segmentation analysis workflow.

Image Shift Calculation
- Image shift analysis of a pair of images. For each image, you should upload a .raw file for processing, and a .jpg/.gif + corresponding .crd file for visualisation.

/services/Examples

Create Table

Hello World

View Table

- /services/Examples/View Table

/services/GM Scenario

GM Scenario - Analysis of data
- GM Scenario data analysis. Looks at raw data and determines the proportion of GM samples in each field.

GM Scenario - Visualisation of results
- View results of the 5 different scenarios for distribution.

GM Scenario - Visualisation of results 2
- View results in interactive applet

/services/GUSTO

GUSTO - GIS Visualisation Applet
- GUSTO project: data visualisation using GIS viewer. You can view the full data in an interactive applet.

GUSTO - GIS Visualisation Snapshot
- GUSTO project: data visualisation using GIS viewer. You can look at snapshots at particular times and pollutants.

GUSTO - Graph Visualisation
- GUSTO project. Graphical visualisation of pollutant concentration levels over time. You may show all 4 pollutants on the same graph, or configure it to select one of the pollutants and a corresponding colour scheme.

- /services/GUSTO/GUSTO - Graph Visualisation

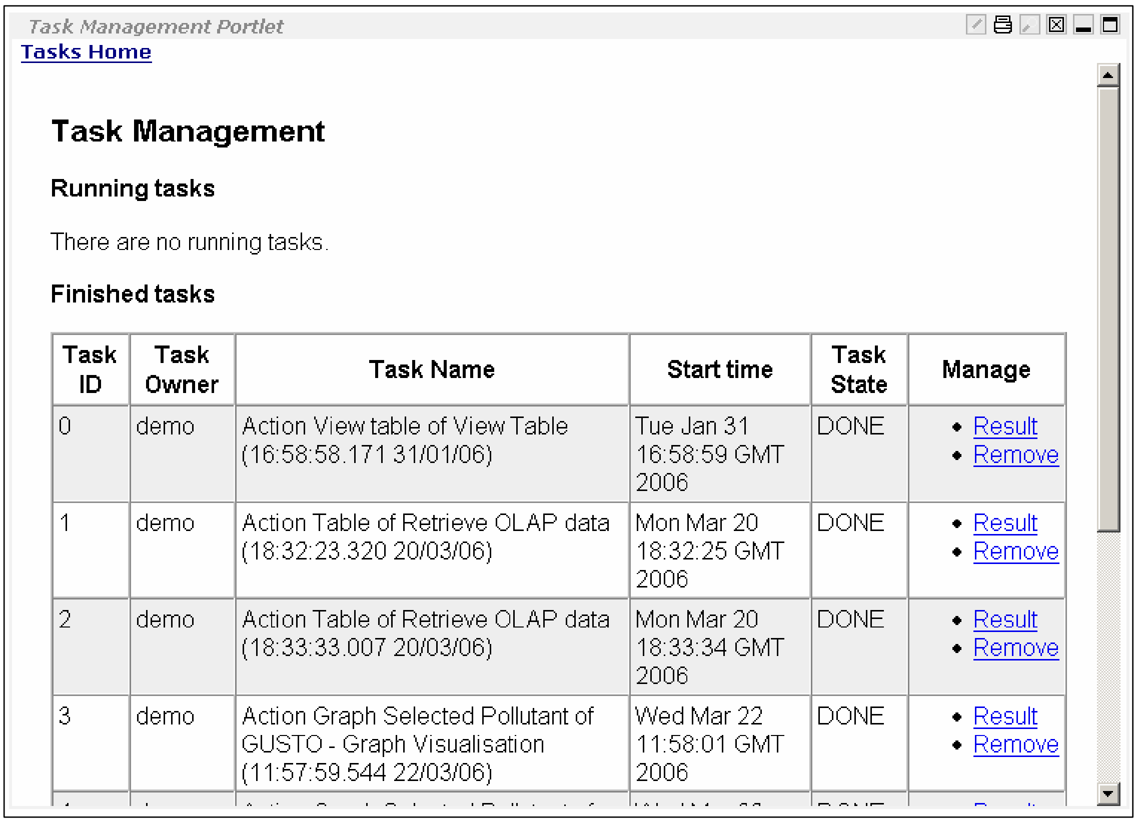
GUSTO - Statistics
- Summarise the GUSTO data by calculating mean values over user-specified time periods. If you save the GISViz table, you may then view the results in the GISViz applet in the other GUSTO Visualisation workflow.

Figure 3.3: Service Index Portlet, showing available services with their corresponding instances in the current user session.

3.5.2.2 Tasks Portlet

This portlet displays lists of running and completed tasks (Figure 3.4). The user may view details of tasks, and the task results. They may also delete old tasks and their results.

This functionality is provided by simply including the original Discovery Net portal's task management servlet within an IFRAME (embedded on the portal page).



Task Management

Running tasks

There are no running tasks.

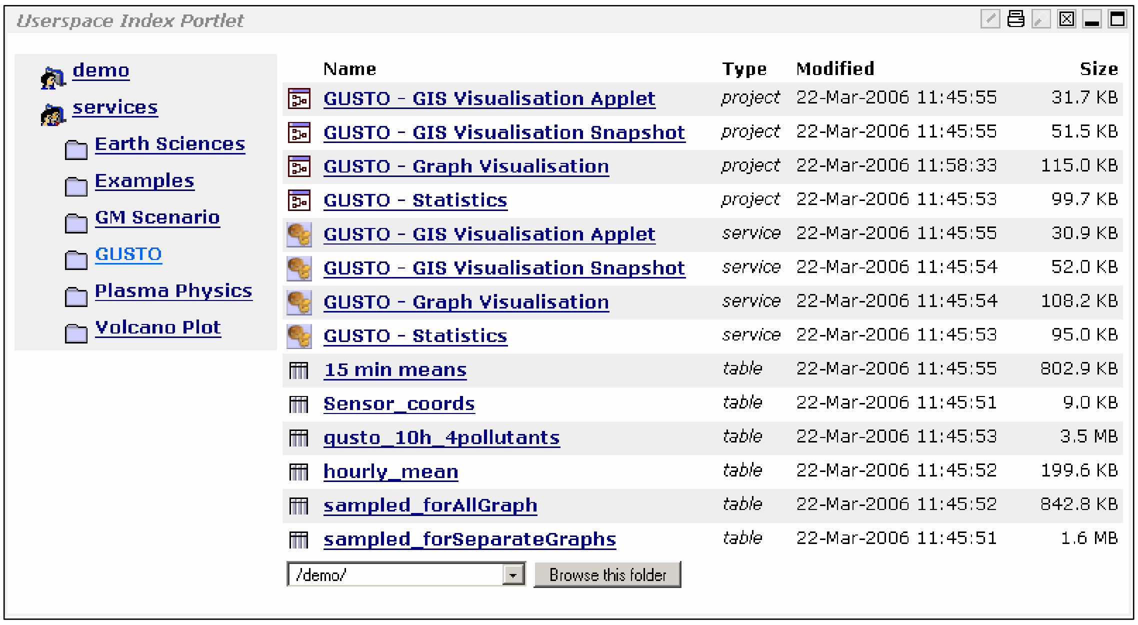
Finished tasks

Task ID	Task Owner	Task Name	Start time	Task State	Manage
0	demo	Action View table of View Table (16:58:58.171 31/01/06)	Tue Jan 31 16:58:59 GMT 2006	DONE	• Result • Remove
1	demo	Action Table of Retrieve OLAP data (18:32:23.320 20/03/06)	Mon Mar 20 18:32:25 GMT 2006	DONE	• Result • Remove
2	demo	Action Table of Retrieve OLAP data (18:33:33.007 20/03/06)	Mon Mar 20 18:33:34 GMT 2006	DONE	• Result • Remove
3	demo	Action Graph Selected Pollutant of GUSTO - Graph Visualisation (11:57:59.544 22/03/06)	Wed Mar 22 11:58:01 GMT 2006	DONE	• Result • Remove

Figure 3.4: Tasks Portlet

3.5.2.3 Userspace Index Portlet

This portlet presents a standard "file-explorer"-style interface for browsing the userspace folders available to the current user (Figure 3.5). The list of items in a folder includes descriptive attributes (type of file, date modified, size).



Name	Type	Modified	Size
GUSTO - GIS Visualisation Applet	project	22-Mar-2006 11:45:55	31.7 KB
GUSTO - GIS Visualisation Snapshot	project	22-Mar-2006 11:45:55	51.5 KB
GUSTO - Graph Visualisation	project	22-Mar-2006 11:58:33	115.0 KB
GUSTO - Statistics	project	22-Mar-2006 11:45:53	99.7 KB
GUSTO - GIS Visualisation Applet	service	22-Mar-2006 11:45:55	30.9 KB
GUSTO - GIS Visualisation Snapshot	service	22-Mar-2006 11:45:54	52.0 KB
GUSTO - Graph Visualisation	service	22-Mar-2006 11:45:54	108.2 KB
GUSTO - Statistics	service	22-Mar-2006 11:45:53	95.0 KB
15 min means	table	22-Mar-2006 11:45:55	802.9 KB
Sensor_coords	table	22-Mar-2006 11:45:51	9.0 KB
gusto_10h_4pollutants	table	22-Mar-2006 11:45:53	3.5 MB
hourly_mean	table	22-Mar-2006 11:45:52	199.6 KB
sampled_forAllGraph	table	22-Mar-2006 11:45:52	842.8 KB
sampled_forSeparateGraphs	table	22-Mar-2006 11:45:51	1.6 MB

Figure 3.5: Userspace Index Portlet

The portlet is intended to be used in combination with other portlets. Other than browsing, its main function is to allow the user to select items for further processing in other portlets. Both files and folders in the userspace may be selected.

The user may specify the default folder path to be displayed on first viewing, in this portlet's Edit mode.

3.5.2.4 Userspace Item Viewer Portlet

This portlet detects the selected userspace item from the Userspace Index portlet and displays its contents in an IFRAME embedded within the portal page (Figure 3.6). Different items are displayed appropriately for their type; some, such as workflows and tables, are pre-processed and formatted for display in a web browser, whereas others (images, custom data files such as molecules, unrecognised file types) are presented to the browser in raw form with an appropriate "Content-type" header so that the browser may use whatever display plugins are installed and appropriate. The HTML table view additionally provides the option to export the table to CSV or Spotfire[75] format.

The user may specify the default userspace item to be displayed, in this portlet's Edit mode.

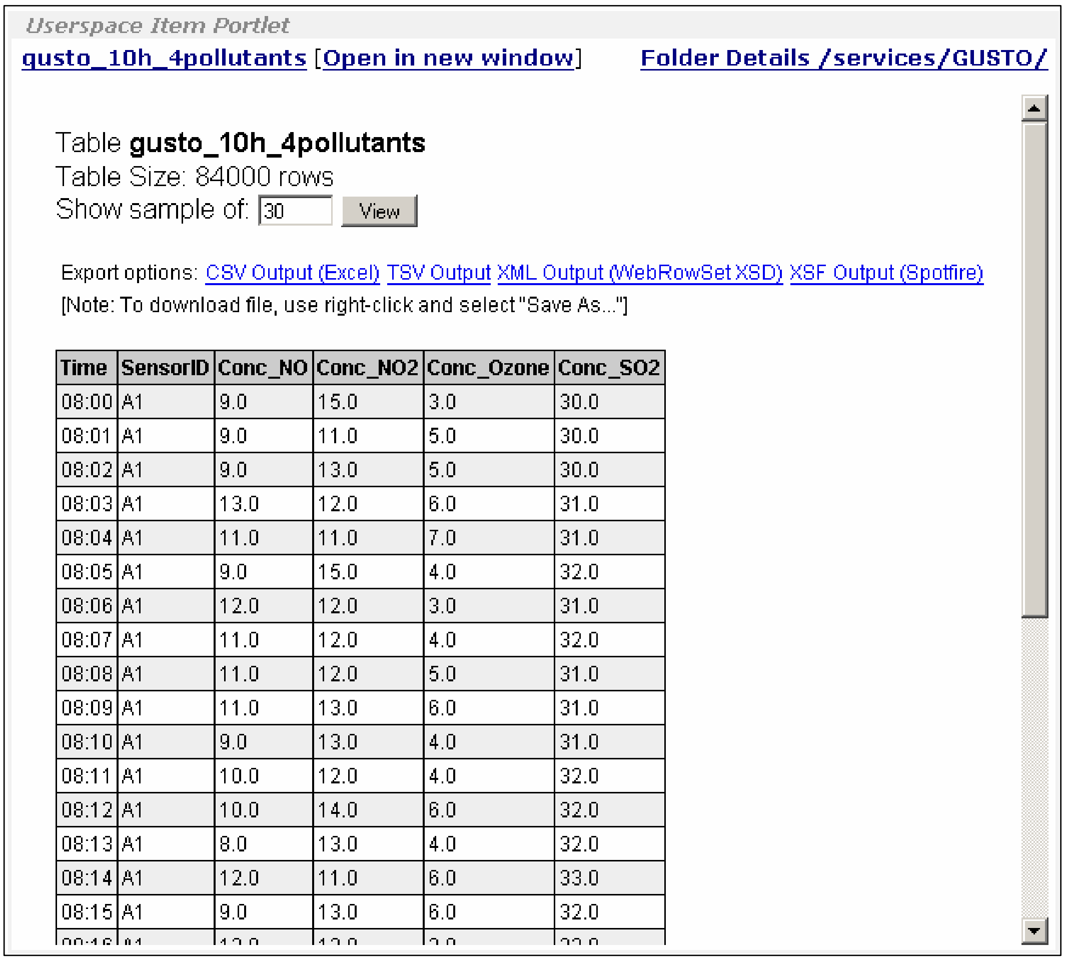

Userspace Item Portlet
gusto_10h_4pollutants [Open in new window]
Folder Details /services/GUSTO/
View

Table **gusto_10h_4pollutants**
Table Size: 84000 rows
Show sample of: 30

Export options: CSV Output (Excel) TSV Output XML Output (WebRowSet XSD) XSF Output (Spotfire)
[Note: To download file, use right-click and select "Save As..."]

Time	SensorID	Conc_NO	Conc_NO2	Conc_Ozone	Conc_SO2
08:00	A1	9.0	15.0	3.0	30.0
08:01	A1	9.0	11.0	5.0	30.0
08:02	A1	9.0	13.0	5.0	30.0
08:03	A1	13.0	12.0	6.0	31.0
08:04	A1	11.0	11.0	7.0	31.0
08:05	A1	9.0	15.0	4.0	32.0
08:06	A1	12.0	12.0	3.0	31.0
08:07	A1	11.0	12.0	4.0	32.0
08:08	A1	11.0	12.0	5.0	31.0
08:09	A1	11.0	13.0	6.0	31.0
08:10	A1	9.0	13.0	4.0	31.0
08:11	A1	10.0	12.0	4.0	32.0
08:12	A1	10.0	14.0	6.0	32.0
08:13	A1	8.0	13.0	4.0	32.0
08:14	A1	12.0	11.0	6.0	33.0
08:15	A1	9.0	13.0	6.0	32.0

Figure 3.6: Userspace Item Viewer Portlet, showing a table item.

An alternate display mode allows a whole userspace folder to be viewed in summary ("detail") mode. This is similar to a normal folder listing, but further details of tables and projects are shown in a compact form.

3.5.2.5 Userspace Item Manager Portlet

This portlet detects the selected userspace item from the Userspace Index portlet, and offers appropriate management options. If the selected item is a file, it allows the user to rename, move, or delete it. If it is a directory, the user may additionally create a new directory within it, or upload a new item.

3.5.2.6 Userspace Upload Portlet

This is a standalone portlet allowing direct upload of tables or files to the userspace. The target directory may be specified in this portlet's Edit mode.

The intention of this portlet is to provide users with a simple component which may be added to a page near to a Service portlet, for convenient upload of resources to be used by the service. However the Service portlet can also allow inline file upload as part of the service parameterisation form, so the use of this portlet is not always necessary.

3.5.2.7 Admin Portlet

This portlet allows a Discovery Net server administrator to perform a number of administrative tasks, such as installing new components or managing users and groups. Like the Tasks portlet, this is implemented by simple inclusion of the original portal's admin servlet within an IFRAME.

The portlet is protected using both the inbuilt authorisation checks in the admin servlet, and role-based portlet permission settings in the portal server, which prevent the portlet from appearing to unauthorised users.

3.5.2.8 Help Portlet

This portlet provides access to the Discovery Net Portal help pages, by inclusion of the original portal's help pages within an IFRAME.

3.6 Portlet Message Flow

Each portlet provides a useful service, but most are not completely independent. For example, the various Index portlets need to communicate with their respective 'detail' portlets. The communication paths between portlets are illustrated in Figure 3.7. As communication between portlets is not provided for in JSR-168, we implemented and used our own portlet messaging (IPC) library, whose design is discussed in detail in Chapter 5.

The most interesting part of this is the ability for messages to flow between Service portlets. This allows the result of one service to be passed on as input to another service (in fact the actual content of this message is the Task ID, as the result itself is stored on the Discovery Net server). This allows multiple related services to be easily used in sequence.

However, the users still need to know which services can be used together. The page customisation features provided by a portal can be used to produce better ways to publish and access a sequence of services than hunting through a simple service index.

A portal user with admin rights - for example the leader of a project group - can create new portal pages and share them with other users. A new page or set of pages can be created to provide access to a specific sequence of services, simply by laying out multiple service portlets on a single page (or in a sequence of tabbed panes) and configuring each service portlet to show the correct service. Then, new users can simply be directed to these custom pages. The page creator can also configure messages between the portlets so that service results are automatically fed into the appropriate input parameters of other Service portlets; this configuration can be saved as the default, so that from then on the users need not even be aware of the messaging configuration and can just use the services as intended.

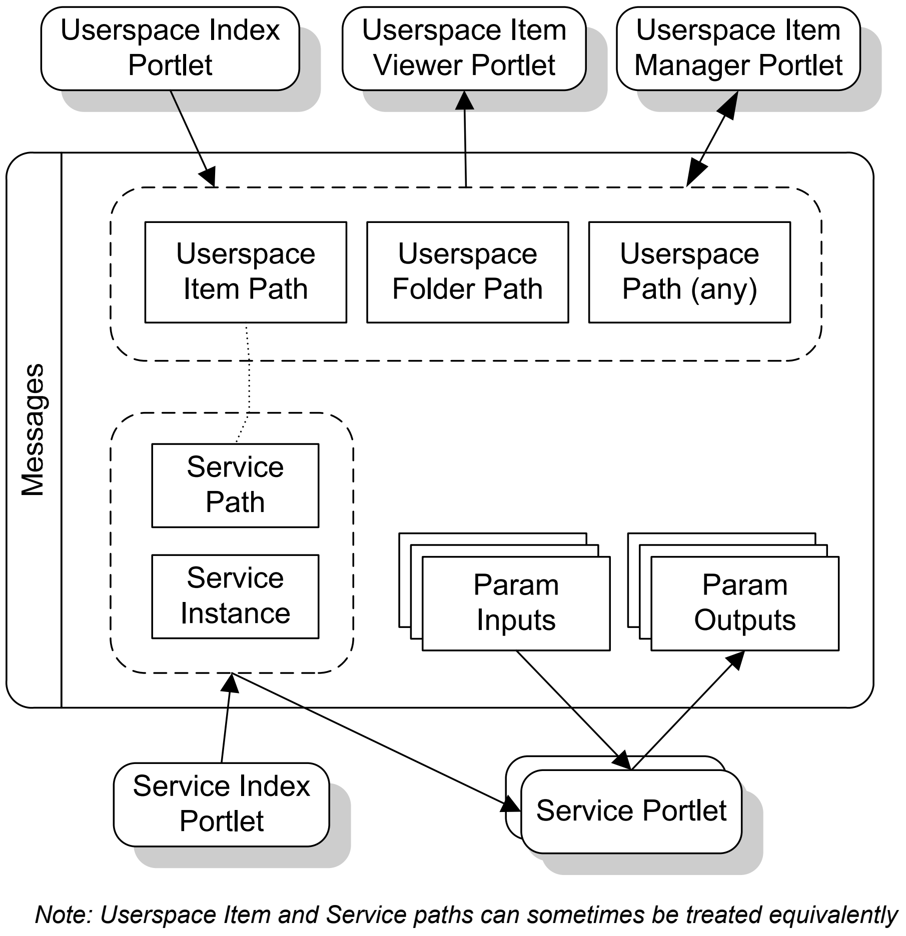


Figure 3.7: Message flow between Discovery Net Portlets.

Input Messages	Portlet	Output Messages
(none)	**Userspace Index**	Userspace Path Userspace Folder Path Userspace Item Path
Userspace Item Path Userspace Folder Path	**Userspace Item Viewer**	*(none)*
Userspace Path	**Userspace Item Manager**	Userspace Path Userspace Folder Path Userspace Item Path
(none)	**Service Index**	Service Path Service Instance Name
Service Path Service Instance Name Dynamic parameter inputs	**Service**	Dynamic action outputs

Table 3.2: Message inputs and outputs of Discovery Net Portlets.

3.7 Software constraints for the Discovery Net portal

Initial Discovery Net portlet prototypes were developed using Apache Pluto[30], the reference JSR-168 portlet container. Once the basic concept had been proven, we assessed the available portal software to choose one for integration and distribution with Discovery Net.

The Discovery Net portal must be embedded in the existing Discovery Net server installation, which is a J2EE application running on JBoss 3.2.7[50]. The original servlet-based portal is still included in the distribution; the JSR-168 portal and portlets simply provide an alternative, and so must be able to co-exist. Thus the technical constraints for the Portal implementation include:

- Java (J2EE)-based
- Runs on JBoss
- Allows integration with existing Discovery Net user system.

We also required the portal to be open source (in case custom modifications for full integration became necessary), and free for use in a commercial application.

In addition, for the Discovery Net portlets to work as intended, there are several requirements relating to some specific Portal design choices (discussed in Chapter 7).

- Support for hosting JSR-168 portlets.
- Independent treatment of multiple portlet windows based upon the same portlet instance (necessary to allow multiple Service portlets to show different Discovery Net services).
- Support for turning off caching of portlet views in the Render phase (necessary for the inter-portlet communication library to work).
- Ability for Struts-based servlet applications to co-exist with portlets in a portlet application (the Discovery Net portlets are heavily dependent on the original Struts-based portal).

Finally, the added value of the JSR-168 portal compared to the original portal is the flexibility of allowing users to arrange services on their own pages. Thus, the portal must provide web interfaces for users to create and customise personal pages. In particular, they must be able to:

- add/remove pages
- add/remove portlets on pages
- arrange portlets on pages

3.8 Choice of Portal implementation

A comparison of available J2EE, open source, free portals was carried out in Spring 2005, to find those which best met our requirements. While this analysis was valid at the time, it is important to note that development of portals proceeds very rapidly, and new versions of many of these portals are now available.

We first investigated the features offered by each portal to confirm that they offered adequate support for page management and integration with Discovery Net. Next, likely portals were tested by deploying, in sequence:

- The testsuite JSR-168 portlets provided by Sun/Pluto (to confirm basic JSR-168 support)
- Simple example messaging portlets working with our IPC library (to confirm IPC library support)
- Discovery Net portlets (to confirm Struts support)

These different portlets put increasing demands on the portal, and so if a portal was found inadequate for one set, later ones were not tested with that portal.

3.8.1 Portals not examined

Several portals were considered but initial investigations indicated they were not suitable.

Jetspeed 2[29] was still under development so was not included in the analysis. Jetspeed 1 was the original Apache Portal project, prior to JSR-168's release. Jetspeed 2 is a rewrite which provides native JSR-168 support. However Jetspeed 2's page management and customisation system was not developed until very late in the development cycle, and at the time of the analysis even its design had not been finalised. As this was an important part of our portal usage scenarios, we could not consider the milestone releases of Jetspeed 2 as an option.

JBoss Portal[51] was just released at the time of the analysis, but requires a higher version of JBoss (4) than that used in Discovery Net. However, when Discovery Net upgrades its version of JBoss, this portal will be seriously considered, as the benefits of using a pre-integrated portal cannot be ignored.

Stringbeans Portal[76], developed by Nabh Information Systems Inc., was in early release and not sufficiently mature in features at the time. Additionally, it is available under the GPL licence, and is not free for commercial use.

Sakai[71], also a comparatively recent portal project, started in January 2004 as an evolution of uPortal[83] and CHEF. It is supported by academic organisations in both the US (University of Michigan, Indiana University, Stanford and MIT) and the UK (the JISC Virtual Research Environments Programme[21]). However, although initial plans intended the use of JSR-168, in practice this has not been a high priority and Sakai does not yet include support for embedding JSR-168 portlets.

eXo Portal[41], although open source, does not allow for free distribution in commercial software.

3.8.2 Portal comparison results

Oracle Portal[66] was previously known to meet all of our portlets' software requirements (as shown in the Business Intelligence demonstration in Section 3.9) – except that it was not open source or free, and could not therefore be packaged as part of the Discovery Net system. It did however act as a useful example for comparison. Its main advantages are that it treats each portlet window as a separate instance, allowing multiple Service portlets to operate independently, and provides a high-quality web GUI for page construction and modification.

Apache Pluto[30] is the reference JSR-168 portlet container implementation, and was used in early development of the Discovery Net portlets. However its portal component is basic and not intended for production use, so again is included for comparison.

As mentioned, the original Jetspeed Portal was one of the early generation of portals and did not support JSR-168. However, the Jetspeed 2 developers back-ported the JSR-168 support to Jetspeed 1, which was released as a hybrid Jetspeed 1.6 (named "Fusion")[28]. This allows us the use of Jetspeed 1's mature user and page management features, combined with modern JSR-168 compatibility.

Liferay[52] is a popular portal, particularly notable for its elegant styles. It is available under the MIT license (a business-friendly open source license) and contains excellent documentation and support for integration with JBoss and other application servers, and with external user authentication systems. Unfortunately when we reached the stage of deploying and testing Discovery Net portlets, we found that it was not compatible. This was because it included its own custom support for Struts in portlets, with a modified Struts library, which conflicted with our use of Struts in normal servlets within our portlet application.

uPortal[83] is a portal targeted mainly at academic institutions, but can be used for any subject matter. However while testing the IPC library portlets we discovered that it automatically cached portlet displays, and used its own methods to determine when the display needed to be re-rendered. Normally this would be an efficient approach, but it effectively disabled the IPC library which relies on message retrieval during the render phase. As we were unable to turn off this caching, we had to abandon uPortal as a possible solution.

GridSphere[78], originally based upon code contributed from the commercial portal IBM WebSphere[46], is closely associated with e-Science projects and includes a set of JSR-168 Grid portlets. Its features (such as page editing interfaces) were more 'barebones' than the other portals examined at this stage in testing, but functional. However, it did not allow for multiple independent portlet windows based upon a single portlet definition.

Table 3.3 shows a summary of the results, and includes the exact versions of portals tested. Several portals failed our tests due to small issues that were nonetheless critical to us.

Portal	Version	Runs on JBoss	Supports JSR-168 portlets	Independent Portlet Windows	Configurable Cache-expiry	Struts can be used in portlet apps without interference	Web interface to Customise Pages	Web interface to Add Pages	Integration with external user/role management	Further Comments
Oracle Portal	Developer WSRP preview	n/a	✓	✓	✓	✓	✓	✓	n/a	Commercial
Apache Pluto	1.0.1 rc2	✓	✓	×	✓	✓	×	×	✓	Basic portal provided, not intended for production use
Jetspeed	1.6	✓	✓	✓	✓	✓	✓	✓	✓	JSR-168 support back-ported from Jetspeed 2
Liferay	Enterprise 3.2	✓	✓	×	✓	×	✓	✓	✓	The custom version of Struts included in Liferay conflicts with that used by our application
uPortal	2.4.2	✓	✓	✓	×	?	✓	✓	?	Uses its own mechanism to determine when portlet views need to be refreshed - bad for IPC library.
GridSphere	2.0.2	✓	✓	×	✓	✓	✓	✓	?	Page customisation is not as flexible as other portals'.

Table 3.3: Portal Comparison (2005) examining suitability for hosting Discovery Net portlets.

3.8.3 Chosen Portal

Jetspeed 1.6 was chosen as the most suitable portal to integrate with the Discovery Net server. Among the portals considered, it was the only one to meet all the technical requirements, and additionally provided excellent page customisation tools.

It is notable - and unexpected - that our requirement for independently-treated portlet windows based upon the 'same' portlet (e.g. multiple Service portlet windows) was not commonly implemented in the portals examined at the time. We were therefore willing to bend this requirement if necessary, as similar (though inferior) functionality could be obtained by adding multiple, numbered instances of the 'same' portlet in the `portlet.xml`. However, we still consider this ability to be very important in fulfilling our usage scenarios, and so in future this may turn out to be a key technical requirement for analytical portals in particular. If the treatment of portlet windows is clarified in the next Portlet Standard, this situation should improve.

On the other hand, page customisation (our other key feature identified for analytical portals) was generally quite good, although some portals allowed more precise or varied layout options than others.

Since the comparison was performed, there has been considerable activity in the Portal market and most of the examined portals have released new versions. In addition, a number of new products are now of sufficient standard/maturity that they should also be included in any future comparisons. Of particular interest to us are Jetspeed 2, as we are already using an early version of its JSR-168 engine embedded in Jetspeed 1.6, and JBoss Portal, which would be most easily integrated with a future version of Discovery Net running on JBoss 4.

However, one year after this comparison, we still consider Jetspeed 1.6 to be a good choice, as it remains superior to Jetspeed 2 in page customisation, and JBoss Portal requires a higher version of JBoss than that currently used by Discovery Net. Jetspeed 1.6’s main disadvantage is that it is a discontinued product, and for this reason migration to Jetspeed 2 or another portal will be necessary at some point in the future.

3.9 Business Intelligence demonstration

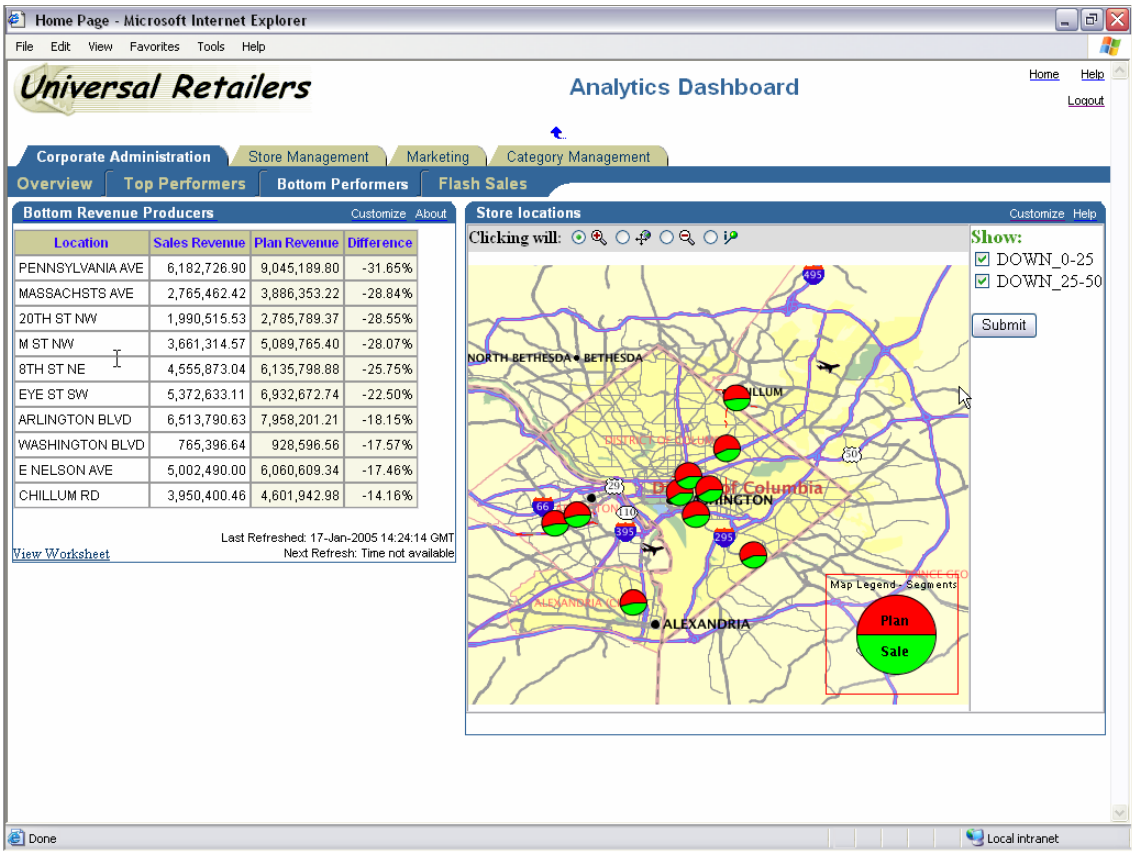
The Oracle Business Intelligence demo application best illustrates the points put forward in this chapter.

This demonstration was one of our first to use portlets, and was based upon Oracle Portal, a component of Oracle Application Server[64]. Oracle Portal is a well-established commercial portal, which supports its own native portlets. It provides a powerful web interface for page creation and management, and treats each portlet window as an independent instance with respect to both sessions and preferences. Oracle also sells many of its tools as portlets (Oracle-native, not JSR-168), including Business Intelligence[65] components such as Discoverer for viewing reports from data warehouses, and Oracle Spatial[67] for visualisation of data over maps.

The demo was performed using a developer pre-release version of Oracle Portal, which supported WSRP portlets and additionally provided a way of hosting and exposing JSR-168 portlets through WSRP. The current version, Oracle Application Server Portal 10g Release 2 (10.1.4), was released in November 2005 and is the first full version to incorporate this functionality.

We show a portal combining both Discovery Net Service portlets and existing Oracle Business Intelligence portlets. By installing our Service portlets, any deployed Discovery Net workflow can easily be accessed through the portal, providing flexible analytical capabilities without significant effort on the part of the server administrator. Discovery Net also includes a plugin allowing development of workflows which integrate closely with Oracle databases and services, which allows for even better integration with the Oracle data warehouse and business intelligence services (Figure 3.11).

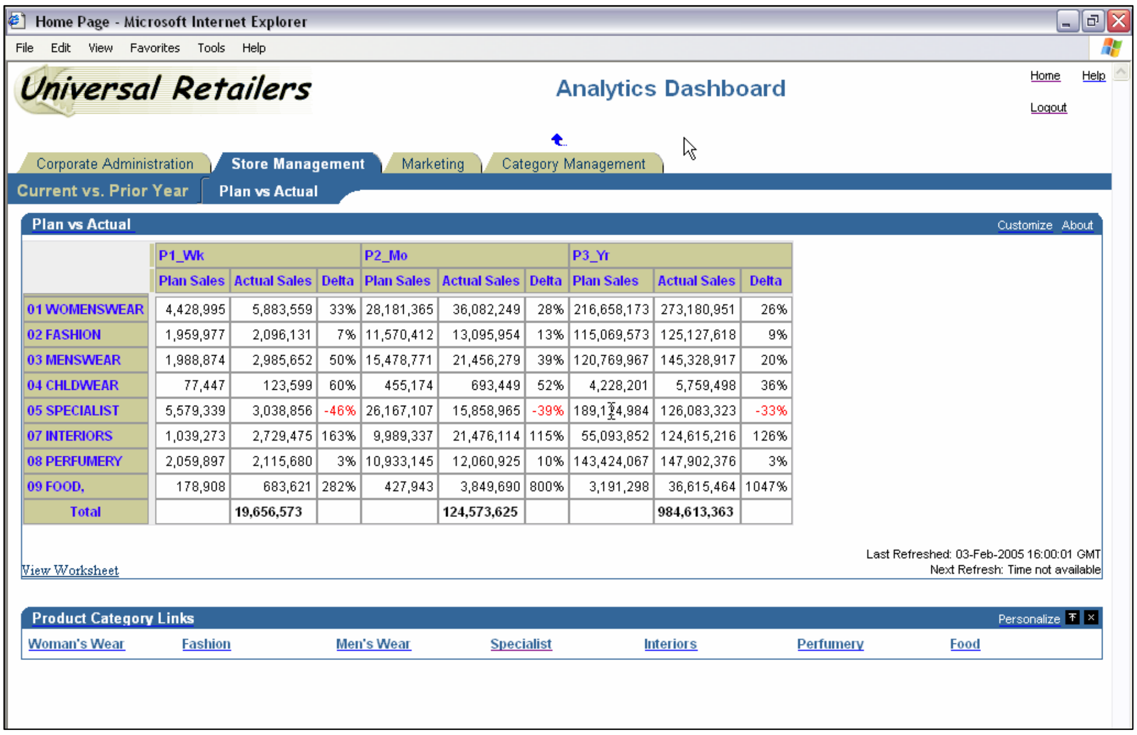
The scenario examines sales data for a large chain of shops. The portal provides information and analysis tools for people in different positions in the organisation. Oracle Business Intelligence portlets present interactive views upon warehoused sales data, allowing general managers to assess all shops' performance, and identify those that are underperforming (Figure 3.8). The store managers can then examine specific sales performance for each product type, drilling down into categories to find out which products are not selling as well as expected (Figure 3.9). Discovery Net services can then be used to analyse the data further, and aid in decision making - for example, association rules can be found which indicate which products are usually bought together, and by what customer demographic (Figure 3.10, Figure 3.12). A marketing manager could then use these correlations to plan marketing campaigns and special offers, to best promote the underperforming products.



Location	Sales Revenue	Plan Revenue	Difference
PENNSYLVANIA AVE	6,182,726.90	9,045,189.80	-31.65%
MASSACHSTS AVE	2,765,462.42	3,886,353.22	-28.84%
20TH ST NW	1,990,515.53	2,785,789.37	-28.55%
M ST NW	3,661,314.57	5,089,765.40	-28.07%
8TH ST NE	4,555,873.04	6,135,798.88	-25.75%
EYE ST SW	5,372,633.11	6,932,672.74	-22.50%
ARLINGTON BLVD	6,513,790.63	7,958,201.21	-18.15%
WASHINGTON BLVD	765,396.64	928,596.56	-17.57%
E NELSON AVE	5,002,490.00	6,060,609.34	-17.46%
CHILLUM RD	3,950,400.46	4,601,942.98	-14.16%

Last Refreshed: 17-Jan-2005 14:24:14 GMT
Next Refresh: Time not available
View Worksheet

Figure 3.8: Managers can identify which shops are underperforming and instruct their store managers to take action. An Oracle Discover portlet is used to display the summary information from a data warehouse, and Oracle Spatial is used to show the same information on a map.



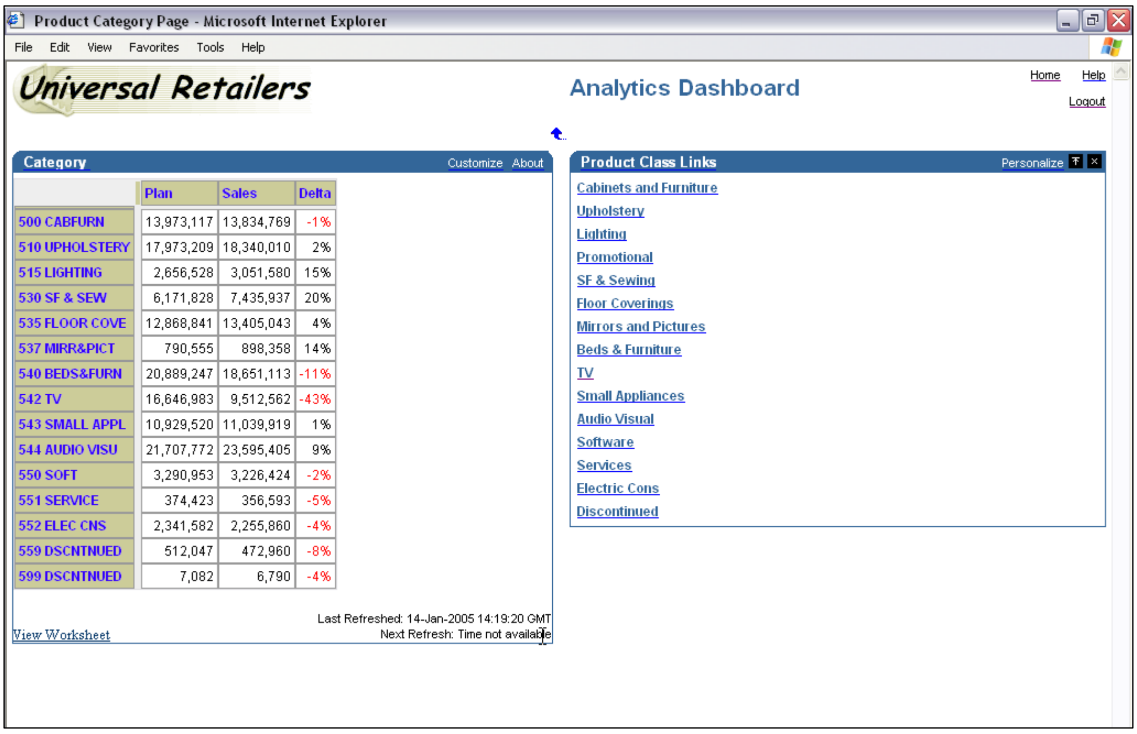
Plan vs Actual

	P1_Wk			P2_Mo			P3_Yr		
	Plan Sales	Actual Sales	Delta	Plan Sales	Actual Sales	Delta	Plan Sales	Actual Sales	Delta
01 WOMENSWEAR	4,428,995	5,883,559	33%	28,181,365	36,082,249	28%	216,658,173	273,180,951	26%
02 FASHION	1,959,977	2,096,131	7%	11,570,412	13,095,954	13%	115,069,573	125,127,618	9%
03 MENSWEAR	1,988,874	2,985,652	50%	15,478,771	21,456,279	39%	120,769,967	145,328,917	20%
04 CHLDWEAR	77,447	123,599	60%	455,174	693,449	52%	4,228,201	5,759,498	36%
05 SPECIALIST	5,579,339	3,038,856	-46%	26,167,107	15,858,965	-39%	189,1[illegible]4,984	126,083,323	-33%
07 INTERIORS	1,039,273	2,729,475	163%	9,989,337	21,476,114	115%	55,093,852	124,615,216	126%
08 PERFUMERY	2,059,897	2,115,680	3%	10,933,145	12,060,925	10%	143,424,067	147,902,376	3%
09 FOOD,	178,908	683,621	282%	427,943	3,849,690	800%	3,191,298	36,615,464	1047%
Total		**19,656,573**			**124,573,625**			**984,613,363**	

Last Refreshed: 03-Feb-2005 16:00:01 GMT
Next Refresh: Time not available

Product Category Links

Woman's Wear   Fashion   Men's Wear   Specialist   Interiors   Perfumery   Food



Category

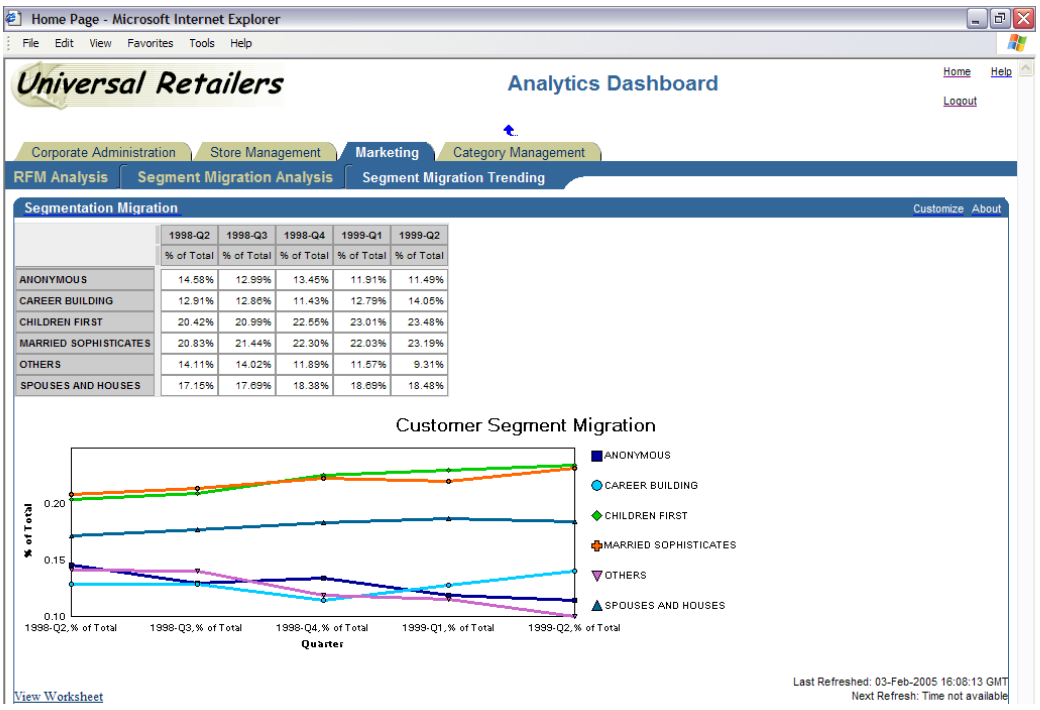
	Plan	Sales	Delta
500 CABFURN	13,973,117	13,834,769	-1%
510 UPHOLSTERY	17,973,209	18,340,010	2%
515 LIGHTING	2,656,528	3,051,580	15%
530 SF & SEW	6,171,828	7,435,937	20%
535 FLOOR COVE	12,868,841	13,405,043	4%
537 MIRR&PICT	790,555	898,358	14%
540 BEDS&FURN	20,889,247	18,651,113	-11%
542 TV	16,646,983	9,512,562	-43%
543 SMALL APPL	10,929,520	11,039,919	1%
544 AUDIO VISU	21,707,772	23,595,405	9%
550 SOFT	3,290,953	3,226,424	-2%
551 SERVICE	374,423	356,593	-5%
552 ELEC CNS	2,341,582	2,255,860	-4%
559 DSCNTNUED	512,047	472,960	-8%
599 DSCNTNUED	7,082	6,790	-4%

Last Refreshed: 14-Jan-2005 14:19:20 GMT
Next Refresh: Time not available

Product Class Links

- Cabinets and Furniture
- Upholstery
- Lighting
- Promotional
- SF & Sewing
- Floor Coverings
- Mirrors and Pictures
- Beds & Furniture
- TV
- Small Appliances
- Audio Visual
- Software
- Services
- Electric Cons
- Discontinued

Figure 3.9: A store manager can use Oracle Discoverer portlets to examine sales performance for different product categories, drilling down in problem areas to identify products that are selling poorly.



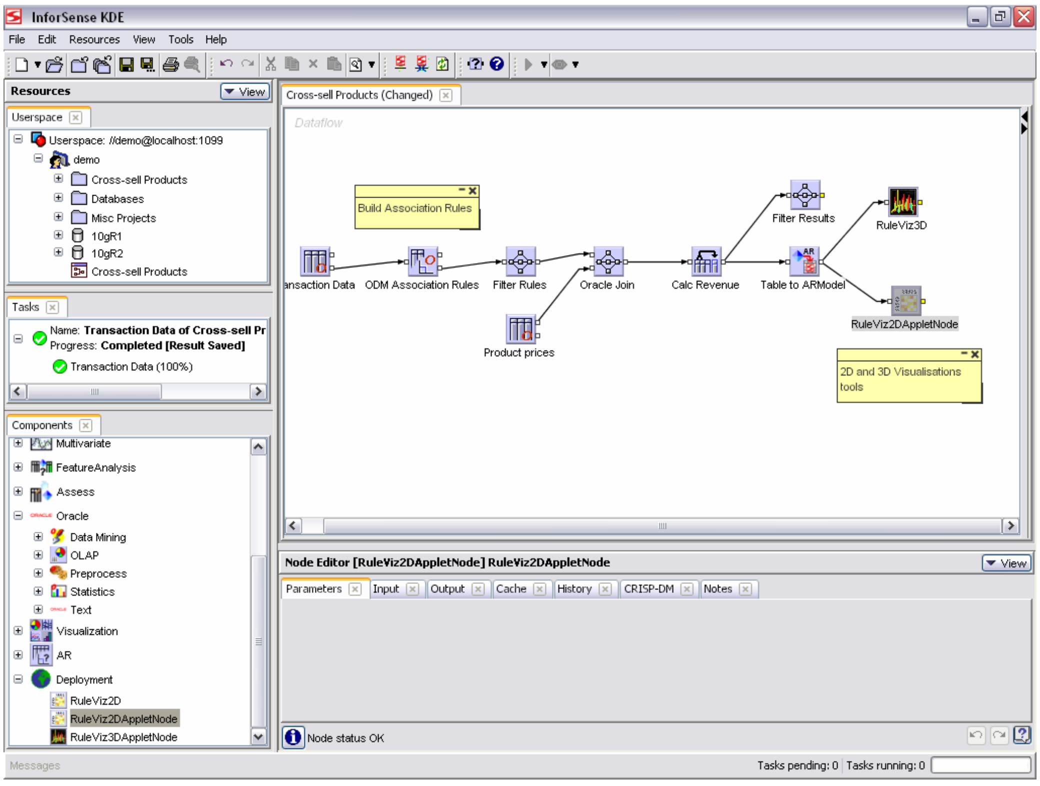
Segmentation Migration

	1998-Q2	1998-Q3	1998-Q4	1999-Q1	1999-Q2
	% of Total	% of Total	% of Total	% of Total	% of Total
ANONYMOUS	14.58%	12.99%	13.45%	11.91%	11.49%
CAREER BUILDING	12.91%	12.86%	11.43%	12.79%	14.05%
CHILDREN FIRST	20.42%	20.99%	22.55%	23.01%	23.48%
MARRIED SOPHISTICATES	20.83%	21.44%	22.30%	22.03%	23.19%
OTHERS	14.11%	14.02%	11.89%	11.57%	9.31%
SPOUSES AND HOUSES	17.15%	17.69%	18.38%	18.69%	18.48%

- ■ ANONYMOUS
- ● CAREER BUILDING
- ◆ CHILDREN FIRST
- ✚ MARRIED SOPHISTICATES
- ▽ OTHERS
- ▲ SPOUSES AND HOUSES

Last Refreshed: 03-Feb-2005 16:08:13 GMT
Next Refresh: Time not available

Figure 3.10: Oracle Discoverer and Discovery Net services can be combined to examine the sales figures broken down by customer demographic.

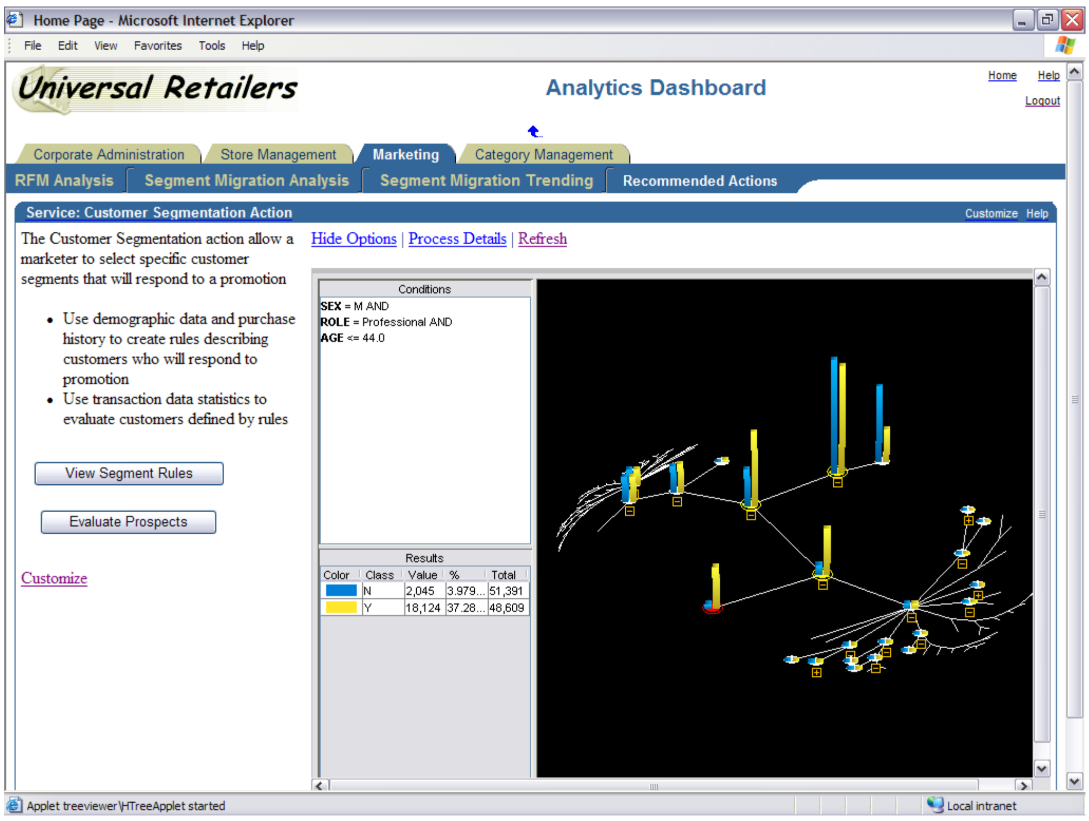


Name: **Transaction Data of Cross-sell Pr**
Progress: **Completed [Result Saved]**
Transaction Data (100%)

Node status OK

Tasks pending: 0 | Tasks running: 0

Figure 3.11: This workflow identifies products which might be best combined in special offers, examining past sales data to generate association rules.


Home Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Universal Retailers
Analytics Dashboard
Home Help
Logout
Corporate Administration
Store Management
Marketing
Category Management
RFM Analysis
Segment Migration Analysis
Segment Migration Trending
Recommended Actions
Service: Customer Segmentation Action
Customize Help
Hide Options | Process Details | Refresh
View Segment Rules
Evaluate Prospects
Customize
Applet treeviewer\HTreeApplet started
Local intranet

The Customer Segmentation action allow a marketer to select specific customer segments that will respond to a promotion

- Use demographic data and purchase history to create rules describing customers who will respond to promotion
- Use transaction data statistics to evaluate customers defined by rules

Conditions

SEX = M AND
ROLE = Professional AND
AGE <= 44.0

Results

Color	Class	Value	%	Total
🟦	N	2,045	3.979...	51,391
🟨	Y	18,124	37.28...	48,609

Figure 3.12: A marketing manager can use Discovery Net services (deployed workflows) to explore the likely effectiveness of different offers to customer segments, aiding in the construction of the marketing campaign.

Many people were involved in the development of this demonstration: my responsibility was for the installation of the Oracle Portal developer preview, the development of portlets for accessing Discovery Net functionality, and their deployment on Oracle Portal.

This demo, the first to use prototype Discovery Net portlets, was very valuable both in showing the benefits to applications of having available easily integratable Discovery Net portlets, and in informing the design and development of the portlet prototypes. An overriding requirement while developing the portlets was that their code must be able to run completely independently of the Discovery Net server environment. This was unlike the original Discovery Net servlet-based portal, which was hosted within the same JBoss application server instance as the Discovery Net server, and had some dependencies on core server code for convenience. As the portlets were based upon and made use of many of the original servlets, these had to be refactored for completely remote server interaction. It was also necessary that all Discovery Net portal code was not reliant on any JBoss-specific settings or features. Having the Discovery Net portlets deployed to an Oracle Application Server on one machine, which then accessed the Discovery Net server on a different machine, demanded resolution of these issues and effectively enforced this code design and the necessary refactoring.

3.10 Conclusion

In this chapter we have discussed the way the aims and thus technical requirements of an analytical portal differ from those of a more traditional portal, particularly with regard to page customisation and the treatment of portlet windows.

We have described the design process of converting the existing Discovery Net portal from servlets to a collection of communicating portlets, and our resulting technical requirements for a portal implementation. We contrast this with the monolithic approach of converting an application to a single portlet.

We then discussed our methods of choosing a portal best suited to these requirements, and present the results of our comparison. Jetspeed Portal 1.6 was selected for integration with Discovery Net, and has shipped with the latest commercial version of Discovery Net, InforSense KDE 3.0.

Finally, we illustrated the value of having portable Discovery Net web components, in a demonstration retail application which combined the established capabilities of Oracle Business Intelligence portlets with the power and flexibility of Discovery Net services.

The Discovery Net portlets developed provide access to the Discovery Net userspace and allow for parameterisation and execution of workflows. Next we will look in more detail at Discovery Net's web ‘deployment’ system for workflows, which enabled the development of our generic Service portlet.

Chapter 4. Web Interfaces to Research Services

The World Wide Web is currently one of the most popular ways of accessing information over the internet. It is easy to learn to use, and accessible on almost any networked computing platform. The web has achieved acceptance and everyday use by many demographics, and has quickly expanded in functionality from the basic provision of information to more interactive services (e.g. shopping, banking and gaming). It is not surprising therefore that it has become one of the most popular channels for researchers to access remote data and tools, whether for pure data retrieval like searching the literature on PubMed[107] and finding information from specialised databases (e.g. RCSB Protein Data Bank[108], NIST Data Gateway[102]), or for active analysis using online services to process data (e.g. NCBI's bioinformatics research tools[101]).

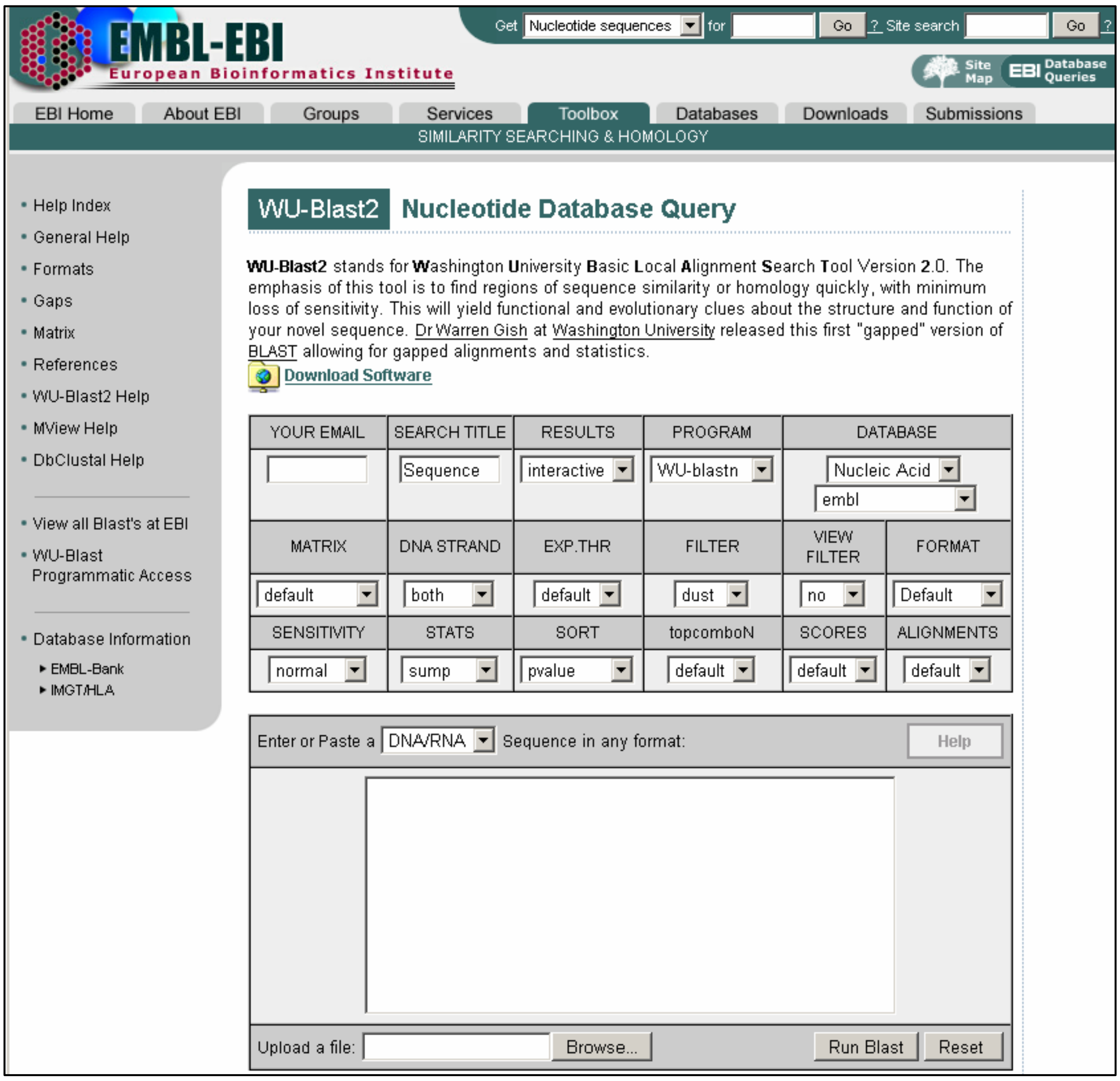
There are of course other methods for accessing remote services. Custom client programs can be distributed and installed, which themselves provide an interface for interaction with the service, and communicate with the remote server(s) using standard or proprietary protocols. This was the approach used by many banks for providing online account services prior to the Web era (e.g. the US bank Wells Fargo first provided a service in 1990 on the Prodigy network[62]), but now, as the web has matured and security features have improved, most provide pure web interfaces (Wells Fargo was the first in 1995, and in the UK, Nationwide Building Society in 1997[62]). Special client programs are also often used to access research databases, typically through a very rich interface tailored to the subject matter. For example, MDL CrossFire[57] and SciFinder Scholar[72] provide access to chemical information and literature, and the Virginia Bioinformatics Institute's pathogen portal project, PathPort[68], provides client software to access their array of web services. Another approach is to not distribute the client software to every users' machine, but instead install a single copy of a client program - perhaps requiring specific hardware - on one machine, and share it throughout an organisation through remote login to a desktop or shell.

Providing a web interface is currently a serious and popular option for new services, as it requires minimal configuration (if any) of the users' machines, and reduces the need to train users, who are already familiar with the conventions of web interfaces. Long term support is simplified, as it is easy to find developers with the web development skills required, and the interface (hosted entirely on the server) may be upgraded for everyone as needed, without any intervention by the users. What it mainly lacks in comparison with custom client programs is the full richness and flexibility of features that only custom programming can provide. However it is still possible to enrich web interfaces through the use of JavaScript[98], Macromedia Flash[54] and Java Applets[1].

Thus web interfaces are now ubiquitous, for performing all sorts of operations from tracking parcel delivery, managing a bank account, or using remote analytical software services. Many networked devices such as routers and scientific instruments often now include an embedded web server, to allow remote configuration through a web interface from any PC on the same local network.

4.1 Creating web interfaces

The majority of web interfaces are - like custom clients - bespoke applications, custom-designed and implemented to access their own particular service (Figure 4.1). Each web interface is typically a wrapper calling a 'real' service, whose behind-the-scenes implementation varies depending on the situation: some services might be executable programs called directly, some might be invoked through web services, others might have their own proprietary communication protocol. In every case, there will have been some non-trivial development work necessary to create, test and debug the web interface in the first place - both the wrapper code, and the design and robustness of the web form page(s).



WU-Blast2 Nucleotide Database Query

WU-Blast2 stands for **W**ashington **U**niversity **B**asic **L**ocal **A**lignment **S**earch **T**ool Version **2**.0. The emphasis of this tool is to find regions of sequence similarity or homology quickly, with minimum loss of sensitivity. This will yield functional and evolutionary clues about the structure and function of your novel sequence. Dr Warren Gish at Washington University released this first "gapped" version of BLAST allowing for gapped alignments and statistics.

Download Software

YOUR EMAIL	SEARCH TITLE	RESULTS	PROGRAM	DATABASE	
	Sequence	interactive	WU-blastn	Nucleic Acid / embl	
MATRIX	DNA STRAND	EXP.THR	FILTER	VIEW FILTER	FORMAT
default	both	default	dust	no	Default
SENSITIVITY	STATS	SORT	topcomboN	SCORES	ALIGNMENTS
normal	sump	pvalue	default	default	default

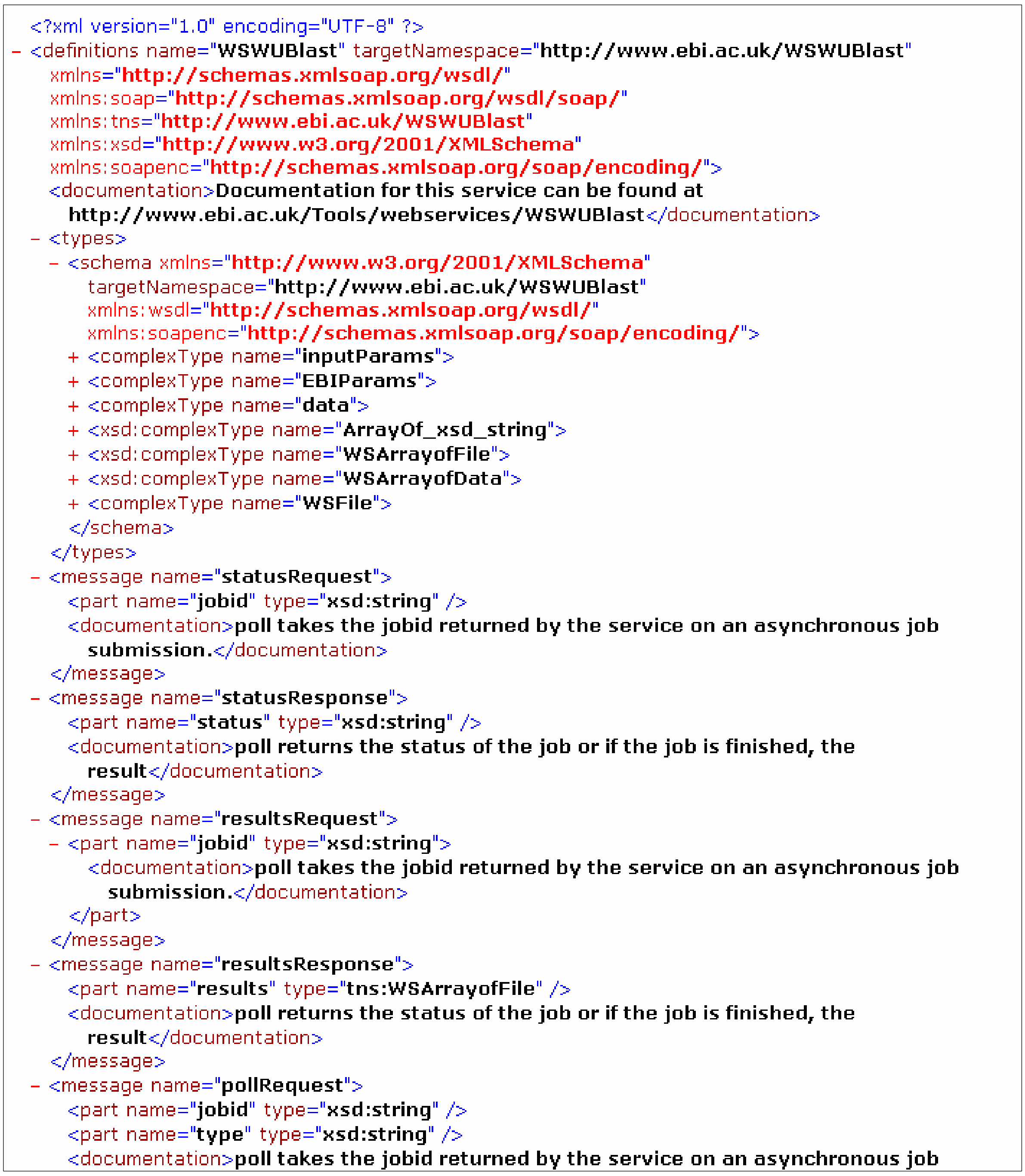
Enter or Paste a DNA/RNA Sequence in any format: Help

Upload a file: Browse... Run Blast Reset

Figure 4.1: The European Bioinformatics Institute (EBI) provides a suite of tools on its website[94], most accessible through web interfaces. It provides a dedicated page or set of pages for each tool, providing users with appropriate documentation and using standard HTML form controls such as combo boxes to improve usability. This service allows users to search databases for matches to a given nucleotide sequence using a BLAST (Basic Local Alignment Search Tools) algorithm[168].

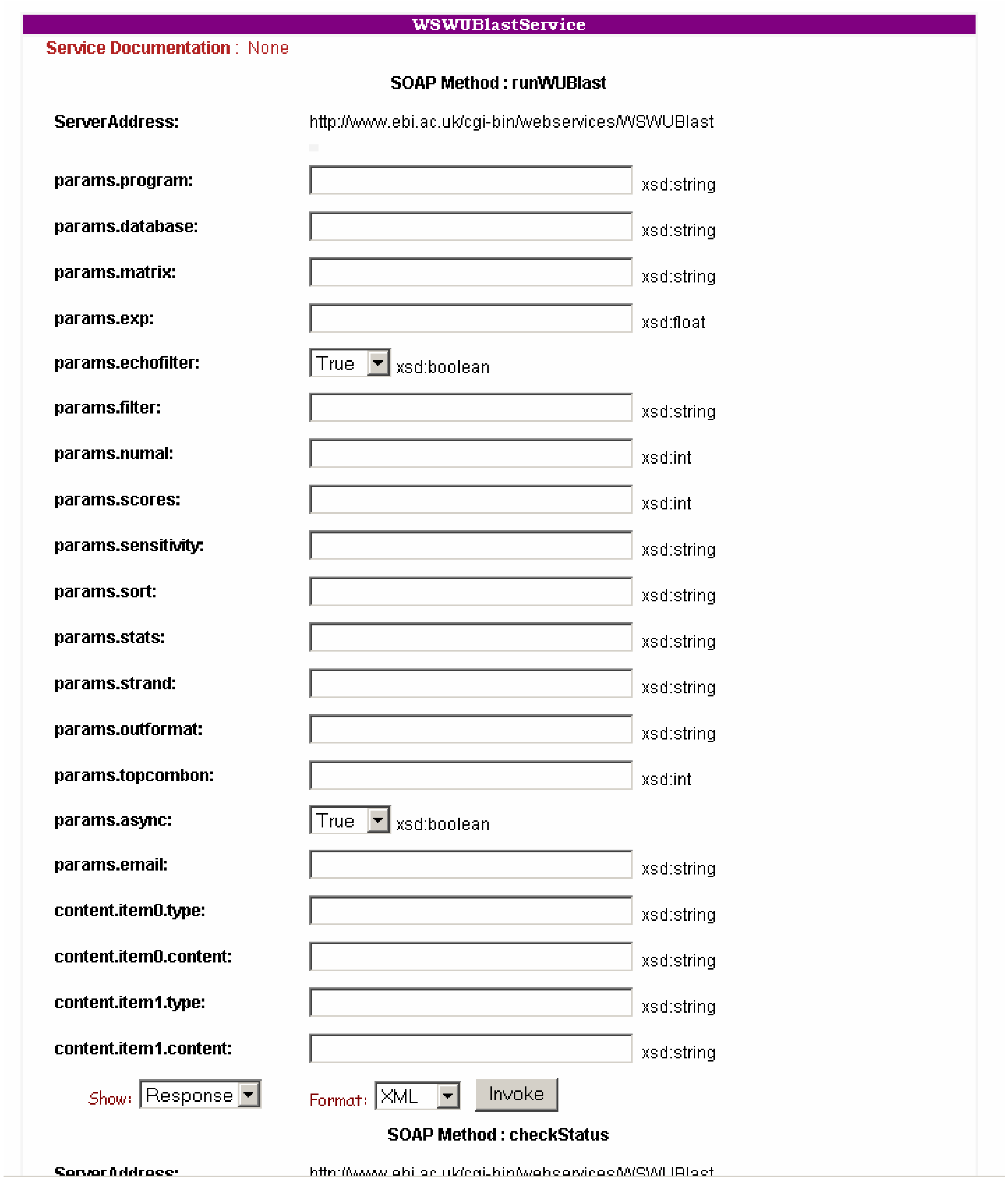
A tempting approach that has recently arisen permits the automatic generation of web interfaces, based upon a well-defined description of how to access a back-end service. Theoretically, if all the services can be accessed using a known standard protocol (web services using SOAP[17]), and the description of each service interface is provided in a standard format (WSDL[6]), a script can be written to dynamically generate both wrapper code and a web form for any service from just the interface description (WSDL document, e.g. Figure 4.2). Thus, to generate a web interface to a web service, it is only necessary to obtain a WSDL description of that service and provide it to the interface-generation script. Such dynamic interfaces have indeed been built (e.g. soapclient.com[43]) and the concept is of great value in eliminating the tedious process of hand-crafting web interfaces, particularly when many services must be exposed.

However, the generated form interfaces, while functional, have generally poor usability: the names of input parameters may not be helpful, usage descriptions are not available, and the web form will usually only provide simple text boxes for entering parameter values (Figure 4.3). In addition, the returned result will generally be in 'raw' form, which may not be suitable for immediate comprehension or reuse. These deficiencies are because WSDL describes the interface syntactically for use by software, not for humans, and so does not have a standard way to include semantic metadata describing the usage or type of the service's functions and parameters (Figure 4.2). As a result, some organisations have come up with extensions to WSDL - some proprietary (e.g. PathPort[68] web services have extra functions such as `aboutService` and `aboutOperations`), others aiming to become standard (WSDL-S[84], DAML-S[174]) - allowing metadata to be included, and so more user-friendly web forms generated.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="WSWUBlast" targetNamespace="http://www.ebi.ac.uk/WSWUBlast"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.ebi.ac.uk/WSWUBlast"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
    <documentation>Documentation for this service can be found at
      http://www.ebi.ac.uk/Tools/webservices/WSWUBlast</documentation>
  - <types>
    - <schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.ebi.ac.uk/WSWUBlast"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
      + <complexType name="inputParams">
      + <complexType name="EBIParams">
      + <complexType name="data">
      + <xsd:complexType name="ArrayOf_xsd_string">
      + <xsd:complexType name="WSArrayofFile">
      + <xsd:complexType name="WSArrayofData">
      + <complexType name="WSFile">
      </schema>
    </types>
  - <message name="statusRequest">
      <part name="jobid" type="xsd:string" />
      <documentation>poll takes the jobid returned by the service on an asynchronous job
        submission.</documentation>
    </message>
  - <message name="statusResponse">
      <part name="status" type="xsd:string" />
      <documentation>poll returns the status of the job or if the job is finished, the
        result</documentation>
    </message>
  - <message name="resultsRequest">
    - <part name="jobid" type="xsd:string">
        <documentation>poll takes the jobid returned by the service on an asynchronous job
          submission.</documentation>
      </part>
    </message>
  - <message name="resultsResponse">
      <part name="results" type="tns:WSArrayofFile" />
      <documentation>poll returns the status of the job or if the job is finished, the
        result</documentation>
    </message>
  - <message name="pollRequest">
      <part name="jobid" type="xsd:string" />
      <part name="type" type="xsd:string" />
      <documentation>poll takes the jobid returned by the service on an asynchronous job
```

Figure 4.2: Partial WSDL of a web service provided by the EBI[94]. This web service provides access to the same BLAST search tool as that shown in Figure 4.1. The WSDL is very verbose, and not intended for direct interpretation by humans.



WSWUBlastService

Service Documentation : None

SOAP Method : runWUBlast

ServerAddress:	http://www.ebi.ac.uk/cgi-bin/webservices/WSWUBlast	
params.program:		xsd:string
params.database:		xsd:string
params.matrix:		xsd:string
params.exp:		xsd:float
params.echofilter:	True	xsd:boolean
params.filter:		xsd:string
params.numal:		xsd:int
params.scores:		xsd:int
params.sensitivity:		xsd:string
params.sort:		xsd:string
params.stats:		xsd:string
params.strand:		xsd:string
params.outformat:		xsd:string
params.topcombon:		xsd:int
params.async:	True	xsd:boolean
params.email:		xsd:string
content.item0.type:		xsd:string
content.item0.content:		xsd:string
content.item1.type:		xsd:string
content.item1.content:		xsd:string

Show: Response Format: XML Invoke

SOAP Method : checkStatus

ServerAddress: http://www.ebi.ac.uk/cgi-bin/webservices/WSWUBlast

Figure 4.3: Generic SOAP clients can read WSDL service descriptions to automatically create interfaces and call the services. However, as there is very little semantic information for the end user in WSDL, these clients cannot make these interfaces user-friendly, and without access to documentation the service may be practically unusable. This figure shows the soapclient.com[43] interface generated from the WSDL in Figure 4.2. Compared to the EBI's custom web interface for the same service in Figure 4.1, this interface is clearly far less usable.

Automatic generation of web interfaces is a concept particularly relevant to Discovery Net: new workflows may be developed at any time, and providing immediate access to these workflows through a web interface is a very valuable feature. It is not practical for the workflow developers (who may not have any programming experience) to create the web interfaces themselves, so an automated approach is the only sensible option. Exposure of workflows through web services is also an important feature, which is compatible with the previous web services discussion.

However, due to the limitations of WSDL regarding metadata, and the lack of a suitable alternative standard, Discovery Net instead uses its own service description language, as an extension of its XML workflow description language, DPML (Discovery Process Markup Language). Rather than using web services as the common wrapper language between the web interface and the services, we use proprietary Discovery Net APIs, but the general approach remains the same (Figure 4.4). The 'deployment' of Discovery Net workflows as services is the main focus of this chapter.

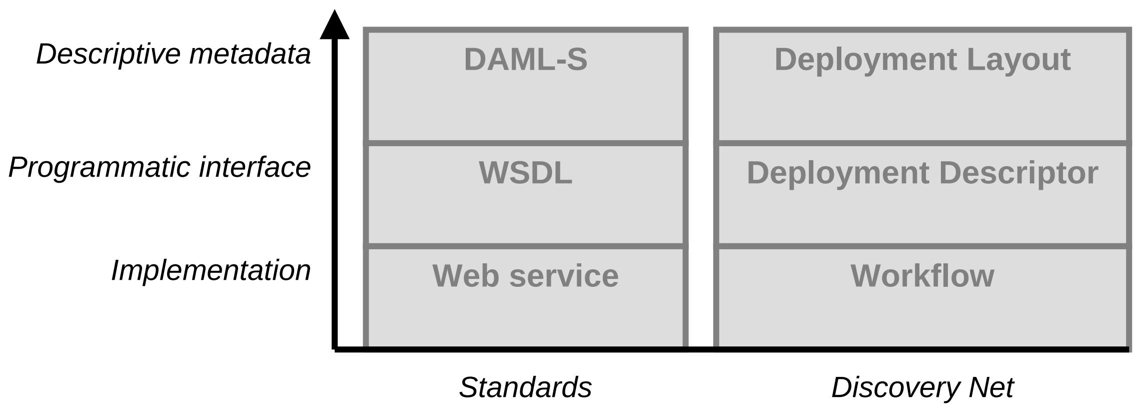


Figure 4.4: Comparing approaches to service description by Discovery Net and standards.

4.2 Discovery Net Service Deployment

The concept of workflow deployment as used in Discovery Net was first described and implemented by Jameel Syed ("Information Structuring for Managing Discovery"[143], Chapter 7). This model behind deployment is still used in Discovery Net, although certain aspects have been re-implemented and extended since.

Generally in this thesis, we refer to a deployed workflow as a *service*. The deployment process is a form of publishing, so that the workflow may be accessed by end users with a web browser through the Discovery Net web portal.

The user of a service's web interface must be able to modify the service's parameters, providing different settings or input data. The simplest approach would be to inspect the workflow and generate a web form which allows the user to modify any of its parameters before execution. However, workflows tend to contain many nodes (usually >5), which themselves contain many parameters (usually between 1-10), and as a whole are therefore quite complex - especially to someone encountering them for the first time. In particular, a new user may not be able to tell which parameters on which nodes can be modified, and which ones have been carefully set by the workflow creator and should not be changed. Even expert users may find it tedious to have to search through all the workflow parameters to find the right ones to adjust.

Thus the core idea, and value, of deployment in Discovery Net is that a workflow developer defines a simplified interface, and the full workflow details are hidden from the end (web) user. The 'deployment' process involves specifying which properties and outputs should be visible and modifiable or executable by the end user (Figure 4.5). This provides a "black-box" view of the workflow, where the user is aware of the inputs and outputs, but does not know or care what is going on inside the box, in the detail of the workflow. This *deployed service* is exposed by the server, and may be accessed by third-party clients either as a web service or through the Discovery Net API (Figure 4.6). Thus, once deployed, a workflow can be used in ways other than the Discovery Net Java client, from command line clients to web interfaces. The Discovery Net web portal uses the Discovery Net API to inspect and execute services.

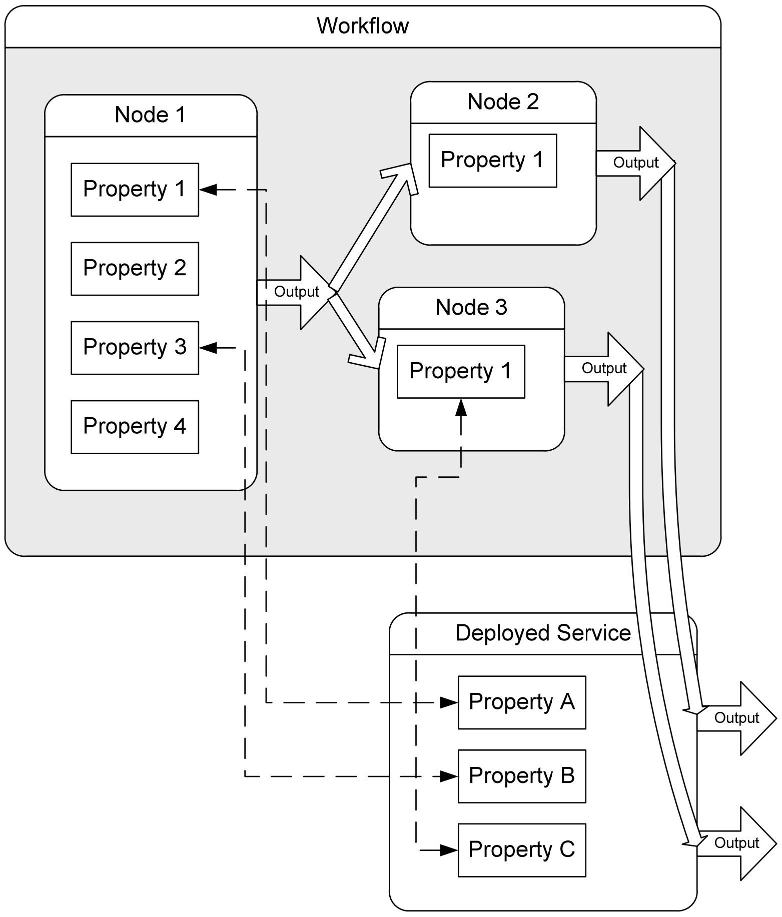


Figure 4.5: Deployment of a Discovery Net workflow is performed by 'promoting' or 'deploying' particular node properties and outputs.

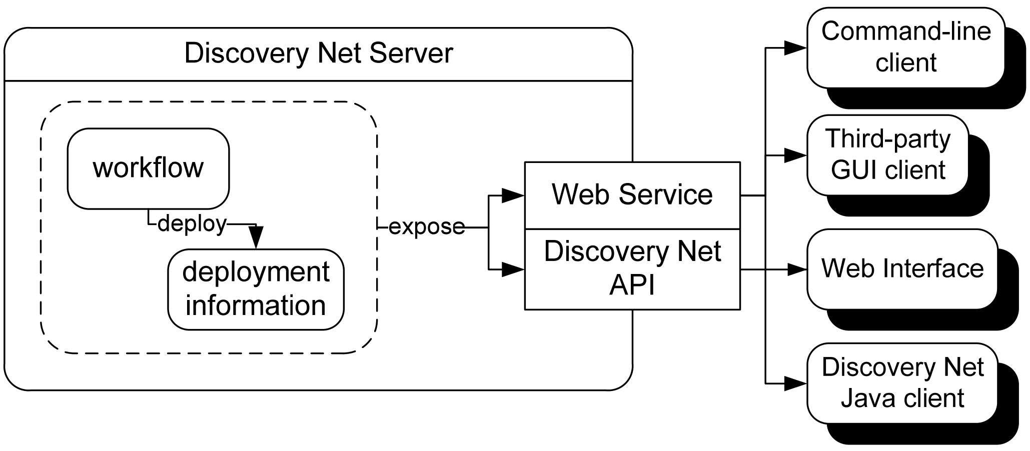


Figure 4.6: Deployment of a Discovery Net workflow effectively publishes a service interface to that workflow, which may be used to execute the workflow either using the Discovery Net API or web services, through a web interface, a Java client, or a command line.

Although I have been significantly involved in the implementation of Discovery Net's deployment system and portal, many of the aspects of deployment described below were designed and developed by other members of the Discovery Net team. The following descriptions are included in this chapter for completeness and due to their core significance to the portal and portlets work described in other chapters. The majority of my work on the core deployment process described here has been in developing support for custom and complex parameterisation web interfaces, and also for providing web visualisations of results.

4.2.1 Storage of deployment information

All deployment information is stored in the Discovery Net workflow description, which itself is stored in the Discovery Net userspace. One workflow may therefore only have one associated set of deployment information; if an alternative deployment is required, the workflow must be copied.

4.2.2 Defining the black box

The core procedure in deploying a workflow as a service is to specify:

- **Properties** : values which may be set by the end user
- **Actions** : output points on the workflow to which the user may make the workflow execute (and receive the result from that output port)

As part of normal workflow editing in the Discovery Net Java Client, the user can select and 'promote' any node property or output to deploy it.

One significant advance in version 3.0 of Discovery Net is the introduction of workflow-level properties, which are visible to all nodes within that workflow and may be used as variables in expressions to set property values. Thus by changing one workflow property, users can automatically configure dependent properties in multiple nodes in the workflow. This feature can also be used as a simple transformation layer, allowing the end user to input simple values into workflow properties which are later converted to more complex node property values. For example, in a node property containing a full SQL query, the end user may only need to change a small part of the whole query - indeed, it may be unwise to allow non-expert users to specify the entire query.

Metadata is included as part of deployment. Each deployed property and action has a name (specified by the deployer) and an optional textual comment, both of which will be displayed on the web form to aid the user. A description may also be added for the whole service.

Type metadata associated with properties is also included. Each property already has basic built-in type information (e.g. String/Float/Date) and constraints, and these have default rendering modes for web deployment (e.g. text box for String, checkbox for Boolean). Some properties may have additional web rendering modes or configuration available and these may also be selected upon deployment (e.g. a multiple selection might use radio buttons, checkboxes or a list box; a molecular formula could be input into a text box, or sketched in a custom applet). The implementation of different property input renderers is discussed further in Section 4.2.3.1.

4.2.3 Creating the web interface

The deployment information, like WSDL in the web services example, is sufficient to allow the Portal code to generate a web interface to the Discovery Net service. It includes both the programmatic interface details (the service execution point, inputs and outputs) and display information (names, comments, layout and rendering instructions).

Programmatic interaction with the deployed service is performed using Discovery Net APIs; although it could alternatively have been done with web services, direct access through the API is more efficient.

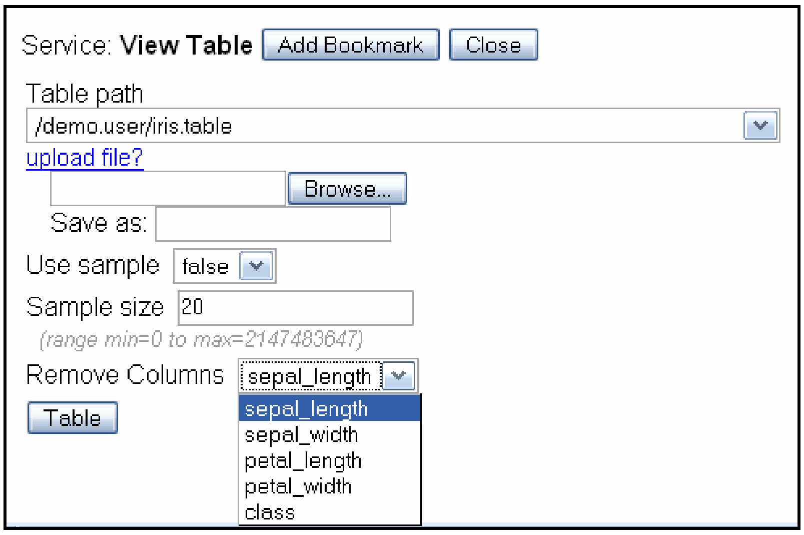
The web display of input properties, results, and page layout has taken the bulk of development time and is the most demanding part of the Portal implementation. It has been through several iterations, aiming to meet user needs in both the deployment process and the Portal. Without the advanced display features offered by this system, the web interface would be far less valuable to the end users, who have high expectations for usability and customisation when creating web interfaces for their services.

4.2.3.1 Input properties

Property values are provided by the user through a HTML form.

For simple property types, this is easy and natural - such as typing in a number, or selecting an option from a combo box. As mentioned, each property already has a default form element appropriate to its basic type (Figure 4.7).

However, other types are more complex, and may be less suited to being input by keyboard into a text box (e.g. String values that require a particular format, such as dates, or SMILES[74] strings for molecular structures). Such property types may require custom input mechanisms: e.g. a combination of JavaScript and custom HTML to add validation or create a more structured form interface (Figure 4.8), or a combination of applets and JavaScript for full programming capabilities (Figure 4.9).



Service: **View Table**

Table path: /demo.user/iris.table

Save as:

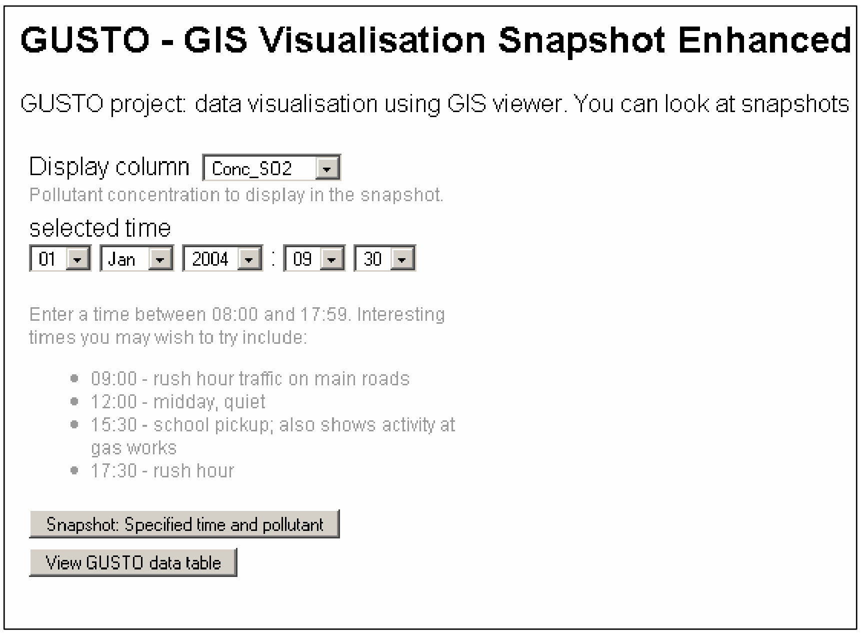
Use sample: false

Sample size: 20

(range min=0 to max=2147483647)

Remove Columns: sepal_length

Figure 4.7: Basic property types displayed using standard HTML form elements. A "userspace path" parameter allows the user to select a file from the userspace or upload one to use as an input parameter



GUSTO - GIS Visualisation Snapshot Enhanced

GUSTO project: data visualisation using GIS viewer. You can look at snapshots

Display column: Conc_SO2

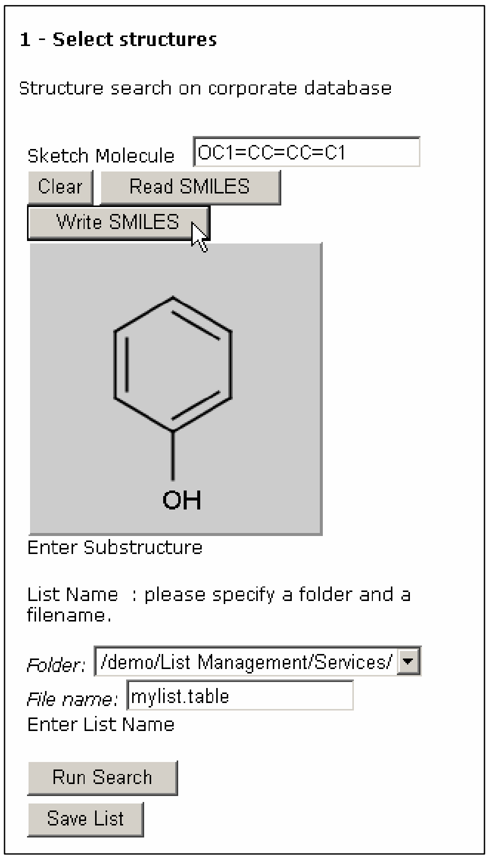
Pollutant concentration to display in the snapshot.

selected time: 01 Jan 2004 : 09 30

Enter a time between 08:00 and 17:59. Interesting times you may wish to try include:

- 09:00 - rush hour traffic on main roads
- 12:00 - midday, quiet
- 15:30 - school pickup; also shows activity at gas works
- 17:30 - rush hour

Figure 4.8: A 'Date' input customised to use multiple combo boxes for more accurate input, instead of a single text box. JavaScript is used to transfer the final String value to a hidden form parameter.



1 - Select structures

Structure search on corporate database

Sketch Molecule: OC1=CC=CC=C1

Enter Substructure

List Name : please specify a folder and a filename.

Folder: /demo/List Management/Services/

File name: mylist.table

Enter List Name

Figure 4.9: Applets can be used to allow users to input more complex types.

Some properties are dynamic: for example, their list of allowed property values may change depending on previous workflow executions and different environments. These properties cannot provide a static list of allowed values upon deployment, but instead must generate and provide them at the time the service form is presented to the user.

There is also a question of where and when validation of property values should happen - it is possible to validate values as the user types them, or just before the form is submitted, using JavaScript on the client; otherwise the values can be checked on the server, and if there is an error the service submission must abort and the page reload, informing the user of the problem. For either approach, validation rules must be defined for every deployed property.

Fulfilling these requirements may be, and indeed has been, achieved in different ways.

In version 2.0 of Discovery Net, we chose to support "custom types" for deployed properties, which were configured in an XML file editable by the workflow developer. Any custom type could be registered as available for any node property, and each custom type had corresponding HTML fragments which would generate the required interface in the web form. When rendering a custom parameter in the service form, the portal would retrieve and insert the fragments of HTML corresponding to that custom type. This general approach - using XML and HTML fragments to define the property input interface - enabled any developer to develop a new input renderer for any property on any node, without needing to modify or recompile code. In addition, as soon as any custom type interface was developed for a property (e.g. a 'date' input), that custom type could easily be registered as available to all appropriate properties on other nodes. However, this approach could not support dynamic properties, whose input interface needed to be generated by active code (e.g. accessing a database for values) at display-time, and as a result these needed special hooks and treatment on the Portal using an entirely separate mechanism.

The stages in the rendering and submission of a service form are shown in Figure 4.10. The rendering of a custom parameter is shown for the version 2.0 mechanism of fetching fragments of HTML to insert into the form.

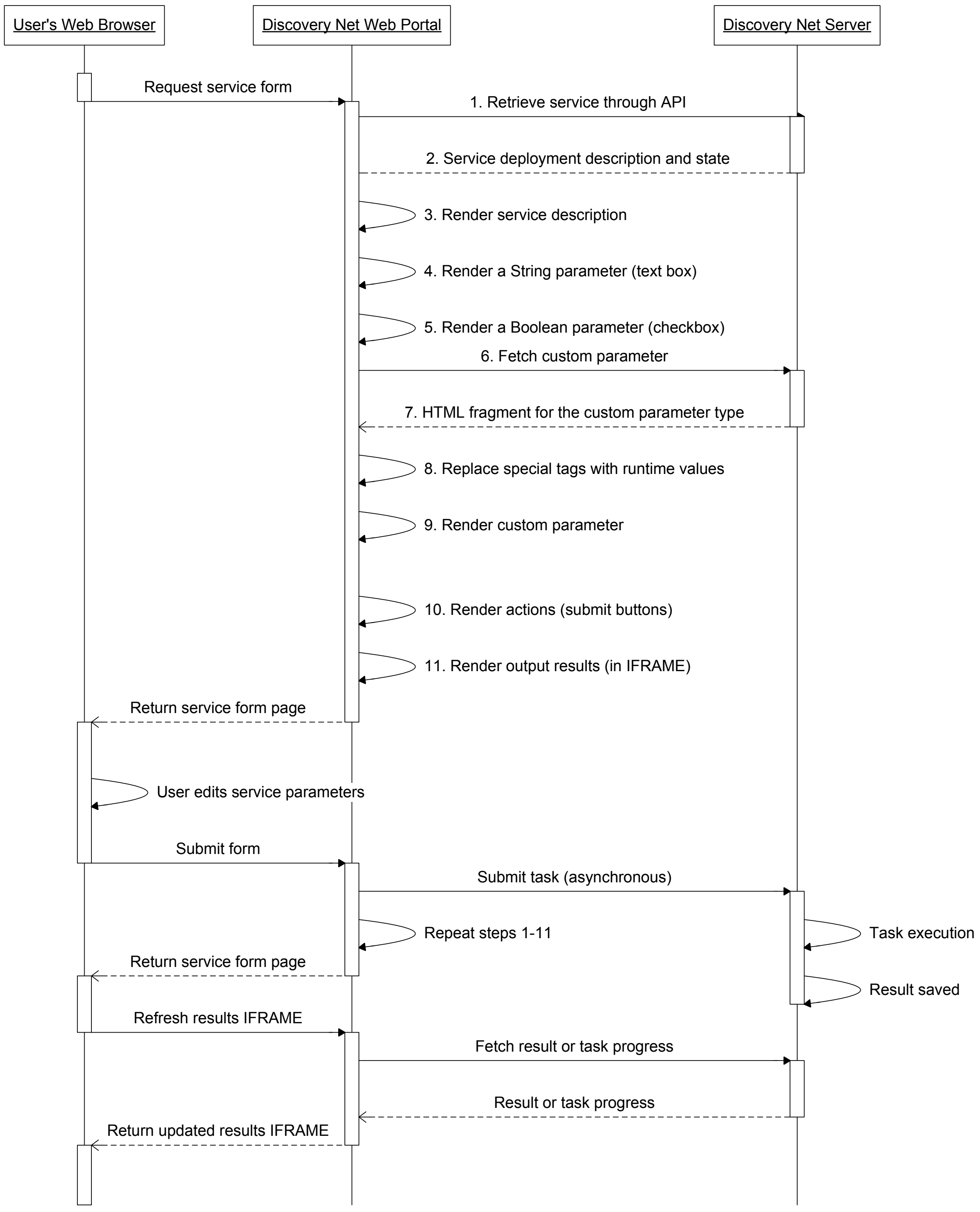

User's Web Browser
Discovery Net Web Portal
Discovery Net Server
Request service form
1. Retrieve service through API
2. Service deployment description and state
3. Render service description
4. Render a String parameter (text box)
5. Render a Boolean parameter (checkbox)
6. Fetch custom parameter
7. HTML fragment for the custom parameter type
8. Replace special tags with runtime values
9. Render custom parameter
10. Render actions (submit buttons)
11. Render output results (in IFRAME)
Return service form page
User edits service parameters
Submit form
Submit task (asynchronous)
Repeat steps 1-11
Task execution
Return service form page
Result saved
Refresh results IFRAME
Fetch result or task progress
Result or task progress
Return updated results IFRAME

Figure 4.10: Stages in rendering and submitting a service form in the Discovery Net Portal version 2.0.

In version 3.0, this approach to custom parameters was replaced by a pure Java implementation, requiring the original developer of a node to explicitly implement or specify any custom web renderers that the node properties would support. This is a less rapidly expandable, but more integrated approach, more suitable for modularity and code maintenance. The portal fetches the custom renderer class from the server and provides this class with access to the web context information. Thus this approach can also support nodes with dynamic requirements: Java code (which is executed on the portal to generate the input interface HTML) is naturally more powerful than simple fragments of JavaScript and HTML, and can (for example) query databases to retrieve an up-to-date list of allowed property values. This model can be represented by a small modification to the previous sequence diagram (Figure 4.11).

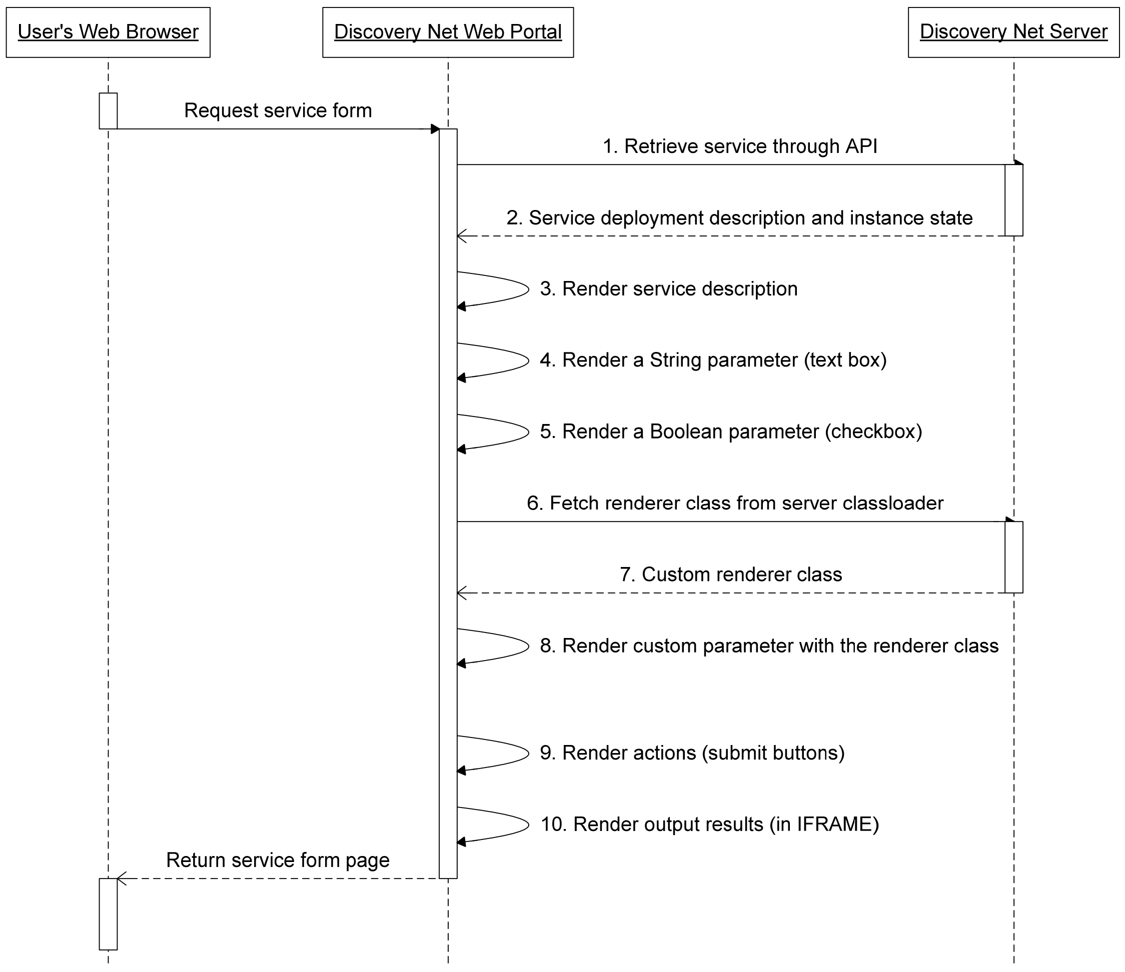

User's Web Browser
Discovery Net Web Portal
Discovery Net Server
Request service form
1. Retrieve service through API
2. Service deployment description and instance state
3. Render service description
4. Render a String parameter (text box)
5. Render a Boolean parameter (checkbox)
6. Fetch renderer class from server classloader
7. Custom renderer class
8. Render custom parameter with the renderer class
9. Render actions (submit buttons)
10. Render output results (in IFRAME)
Return service form page

Figure 4.11: Modified procedure for rendering custom parameters in a service form in the Discovery Net Portal version 3.0.

There is still however inefficiency and duplication of effort for the node programmer, who must define the input property interface for both the Java Client (using Java Swing controls and layout with XUL, the XML User Interface Language[7]), and the web interface. A more efficient approach, not yet explored, might be to describe both input interfaces using the same XUL, which the Portal might then automatically convert to HTML form elements. This would eliminate much of the effort that is currently necessary for parameters using standard form controls, although it might be difficult to implement parameter renderers that need to execute code at runtime to generate their display, and special web-only renderers such as applets would still need custom implementation.

4.2.3.2 Display of stored data and service results

Users need to be able to quickly assimilate, and sometimes interactively explore, the results of service executions. Thus the display of result data is critical: either in a browser through the use of applets or plugins, or with an external program installed on the client - in which case, the data must be easily exportable. The following discussion applies equally to service results and data stored in the userspace - whether these are saved results, or files uploaded directly by the user.

Web browsers are not well suited to visualisation of results, when compared with the flexible interactive visualisers in the Java client. A variety of approaches have been tried, from converting the visualisers to Java applets, to creating cut-down alternative visualisations, to simply ensuring smooth integration from the browser to launch external programs, such as Spotfire[75].

Any node output may be deployed to produce a service result, but not all forms of output data will be suitable for viewing through a web browser. There are two issues at hand: the provision of the raw result data to the client as a result of execution through the web portal, and the visualisation (possibly interactive exploration) of the result data through the web portal.

The first is quite easy to support: tabular and text results are presented using HTML, and if the node needs to return a binary result, MIME-formatted output will be downloaded and treated appropriately by the web browser. Some file types will be automatically displayed by the browser (e.g. images, Figure 4.12), others may be recognised and launched with the appropriate programs (e.g. Microsoft Office documents, PDFs), or simply downloaded.

In addition, browser plugins may provide alternative or interactive visualisations of some file types. For example, the MDL Chime[56] plugin allows browsers to 'recognise' particular chemical file formats, and once installed, the browser will automatically load such files using the Chime plugin rather than simply displaying the raw file contents (Figure 4.13). To ensure that service results (or userspace items) are automatically loaded by the correct plugin, the Discovery Net portal merely needs to ensure that the responses include the correct "Content-Type" header for the file type.

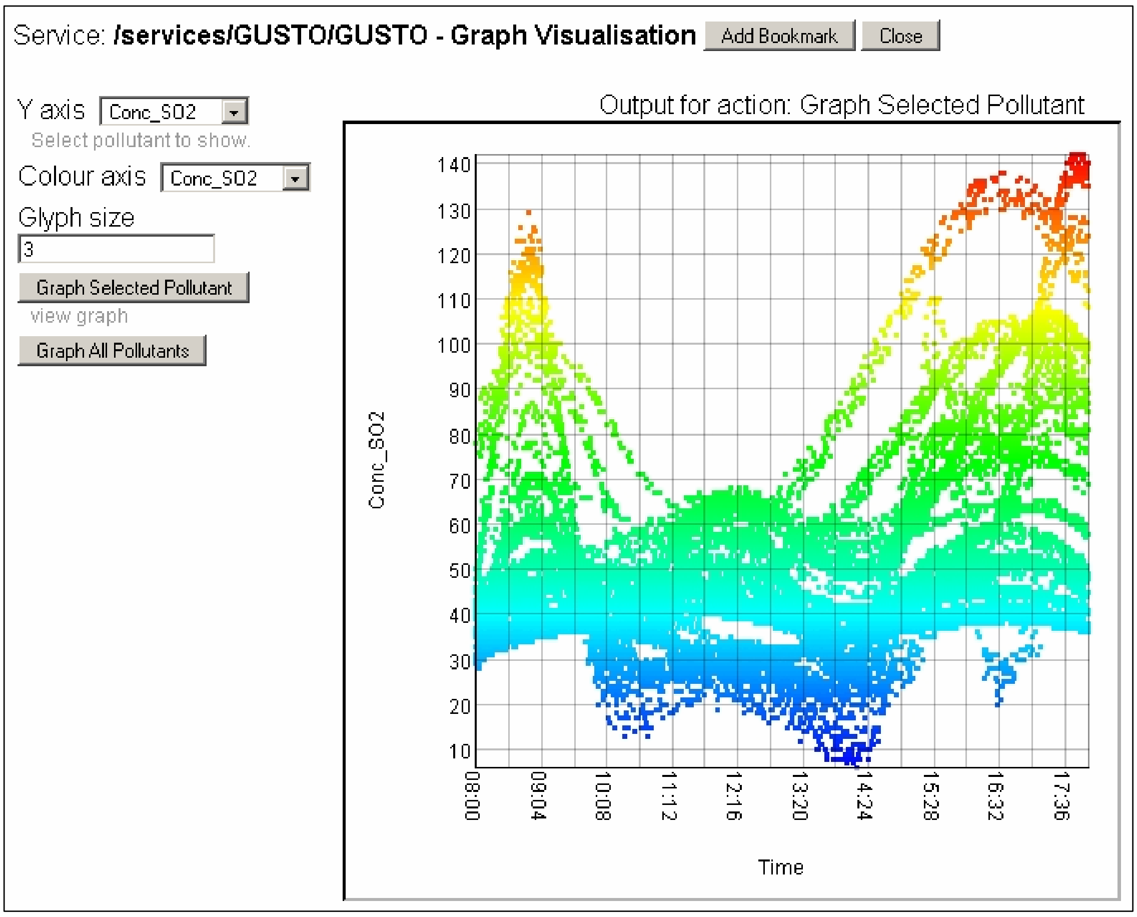
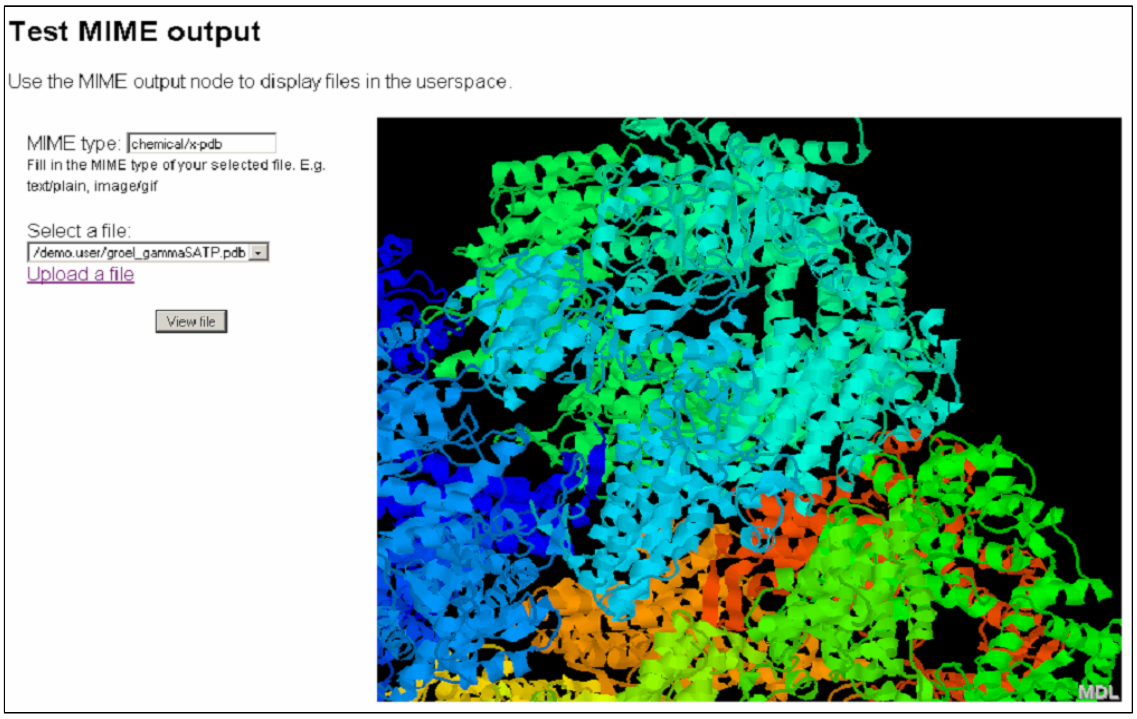


Figure 4.12: Special nodes for the portal generate images (non-interactive) that may be viewed by any browser.


Test MIME output

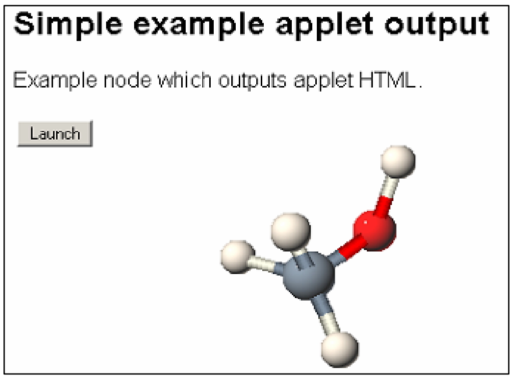
Use the MIME output node to display files in the userspace.

MIME type: chemical/x-pdb

Fill in the MIME type of your selected file. E.g. text/plain, image/gif

Select a file: /demo.user/groel_gammaSATP.pdb

Figure 4.13: Installed plugins on the client browser deal with display of recognised file types. Here the plugin MDL Chime[56] displays a PDB[108] result.


Simple example applet output

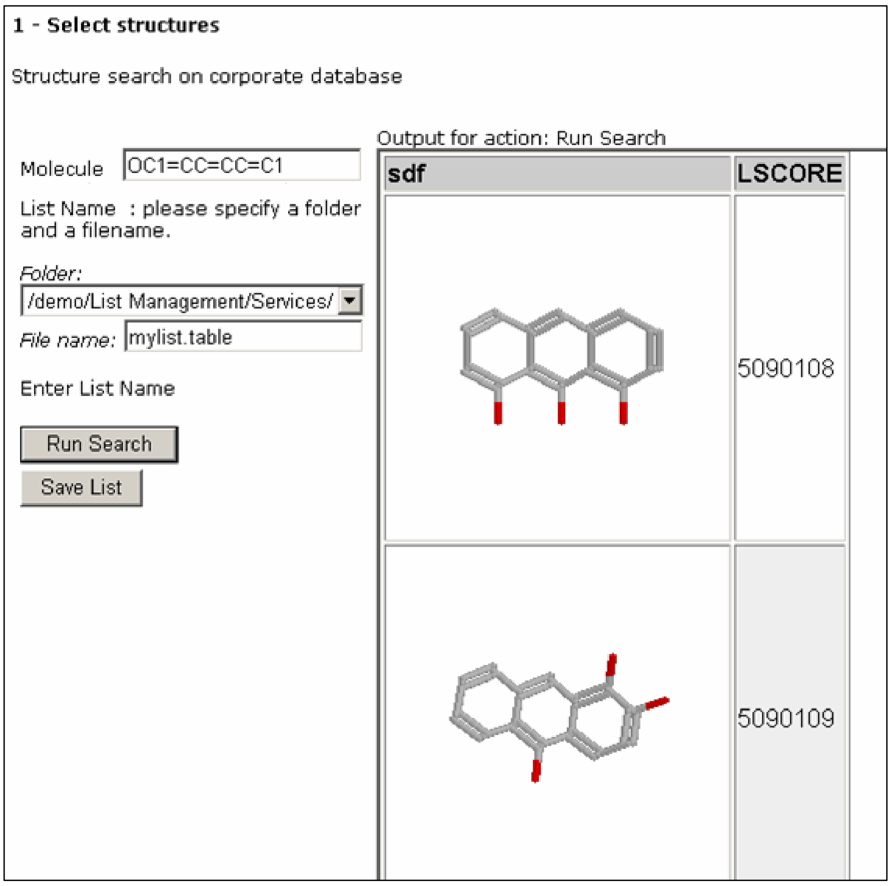
Example node which outputs applet HTML.

Figure 4.14: Output HTML to load an applet showing the result data (the data may be passed as an applet parameter). The applet must be available for download to the client, in a known location, so that the node can correctly generate the HTML.

For interactive exploration of results, it is necessary to use either browser plugins or custom applets (Figure 4.14). There are several complications involved in using applets, as will be discussed later (Section 4.3.1), but this is still a popular option.

Another alternative for richer result presentation is to save an HTML-formatted result with associated resources (e.g. images) to the userspace, and then return its userspace path as the result which the portal will then recognise and display in full to the user (Figure 4.15).

However, despite the appeal of interactive or visual data, the majority of service results will be in tabular or raw form, ready for further processing. Tabular data is displayed in a formatted HTML table (Figure 4.16), but to reduce download time, only the first few rows will initially be shown. Links are also available to export the data to a variety of other formats, including CSV (e.g. for Microsoft Excel) and Spotfire[75].


1 - Select structures

Structure search on corporate database

Molecule: OC1=CC=CC=C1

List Name : please specify a folder and a filename.

Folder: /demo/List Management/Services/

File name: mylist.table

Enter List Name

Output for action: Run Search

sdf	LSCORE
	5090108
	5090109

Figure 4.15: An HTML-formatted result, with associated resources such as images, is saved to the userspace.

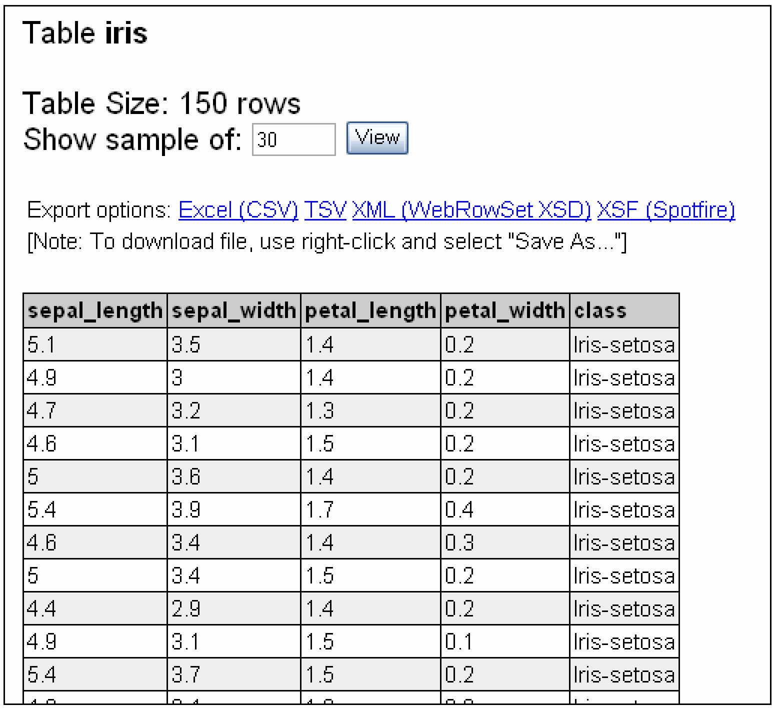


Table iris

Table Size: 150 rows
Show sample of: 30 View

Export options: Excel (CSV) TSV XML (WebRowSet XSD) XSF (Spotfire)
[Note: To download file, use right-click and select "Save As..."]

sepal_length	sepal_width	petal_length	petal_width	class
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa

Figure 4.16: Tabular data is the most common form of result. The portal recognises tabular results and automatically presents them as HTML. Options are provided to save the table to the userspace, or export it to a variety of formats for further processing in other programs.

4.2.3.3 Page layout

The layout of the web interface is the final aspect of customising the display for a particular service.

A few default layouts are provided for all services. These use the simple but functional approach of putting all input properties on the left hand side of the page, followed by buttons for all published actions, and results on the right hand side of the page (Figure 4.12 - Figure 4.15).

The default layouts are often quite acceptable to users. However, some more complex services, particularly those with many properties or outputs, can be better presented with a custom layout (Figure 4.17). The Java Client includes a graphical Layout Editor, which allows the user deploying the service to specify the placement of properties, action buttons, and results on the page grid (Figure 4.18). They can also choose which renderers are used for properties and results, and insert additional elements such as line breaks or explanatory text.

In addition, the Layout Editor allows the different elements of a deployed service to be spread out across multiple pages, in a 'wizard'-like set with "Previous" and "Next" page links (Figure 4.19, Figure 4.20). This is helpful when there are multiple stages within a single service, so that a user may enter properties on one page associated with the first action, then go to the next page for the next action and so on.

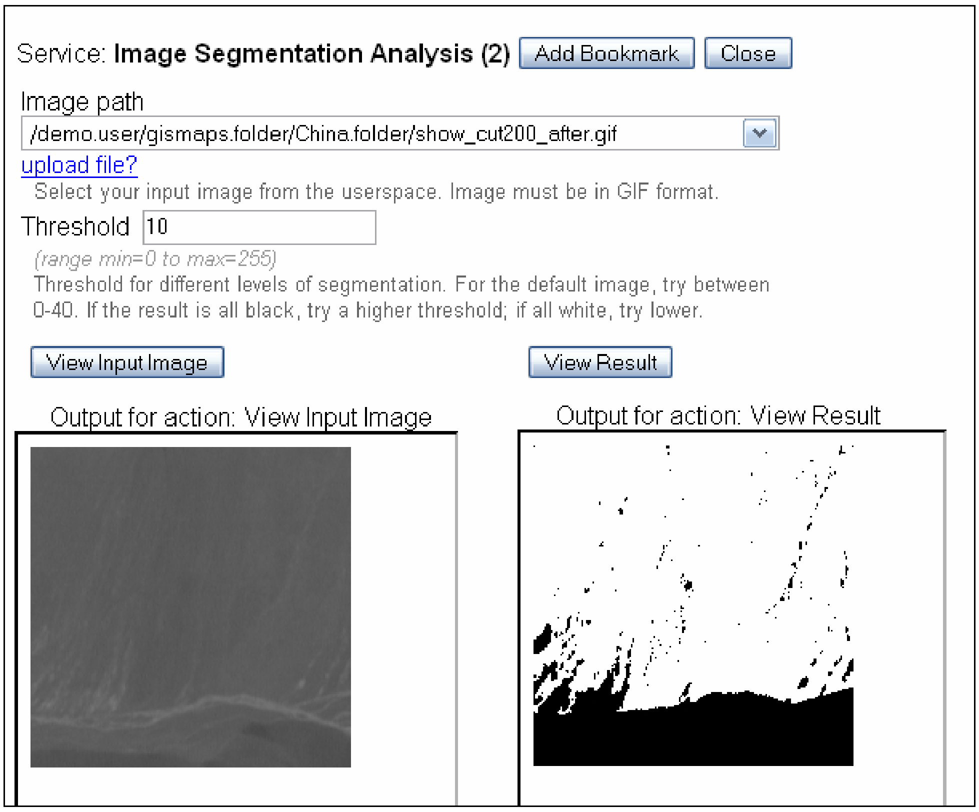

Service: **Image Segmentation Analysis (2)**

Image path

/demo.user/gismaps.folder/China.folder/show_cut200_after.gif

upload file?

Select your input image from the userspace. Image must be in GIF format.

Threshold 10

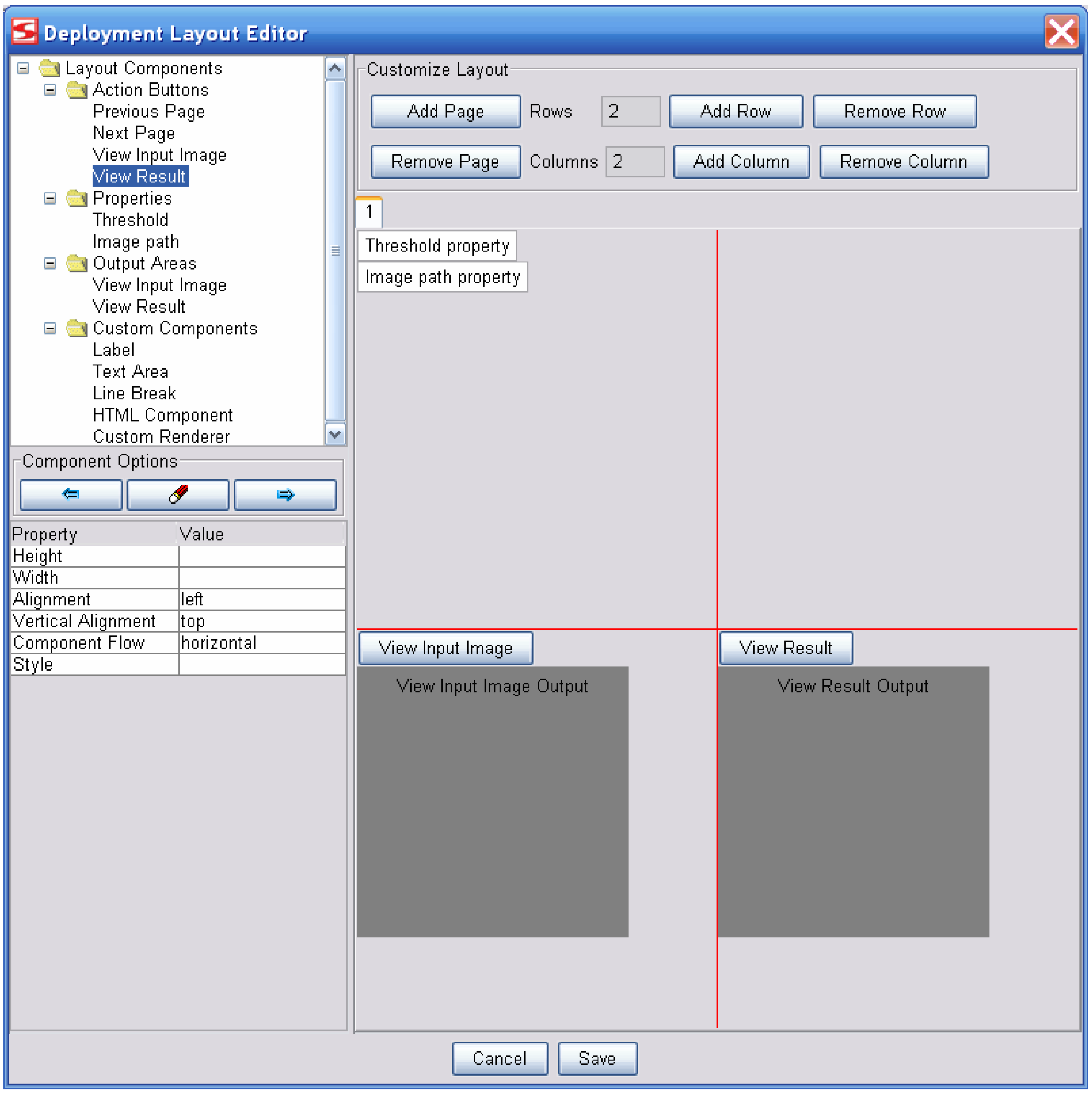
(range min=0 to max=255)

Threshold for different levels of segmentation. For the default image, try between 0-40. If the result is all black, try a higher threshold; if all white, try lower.

Output for action: View Input Image

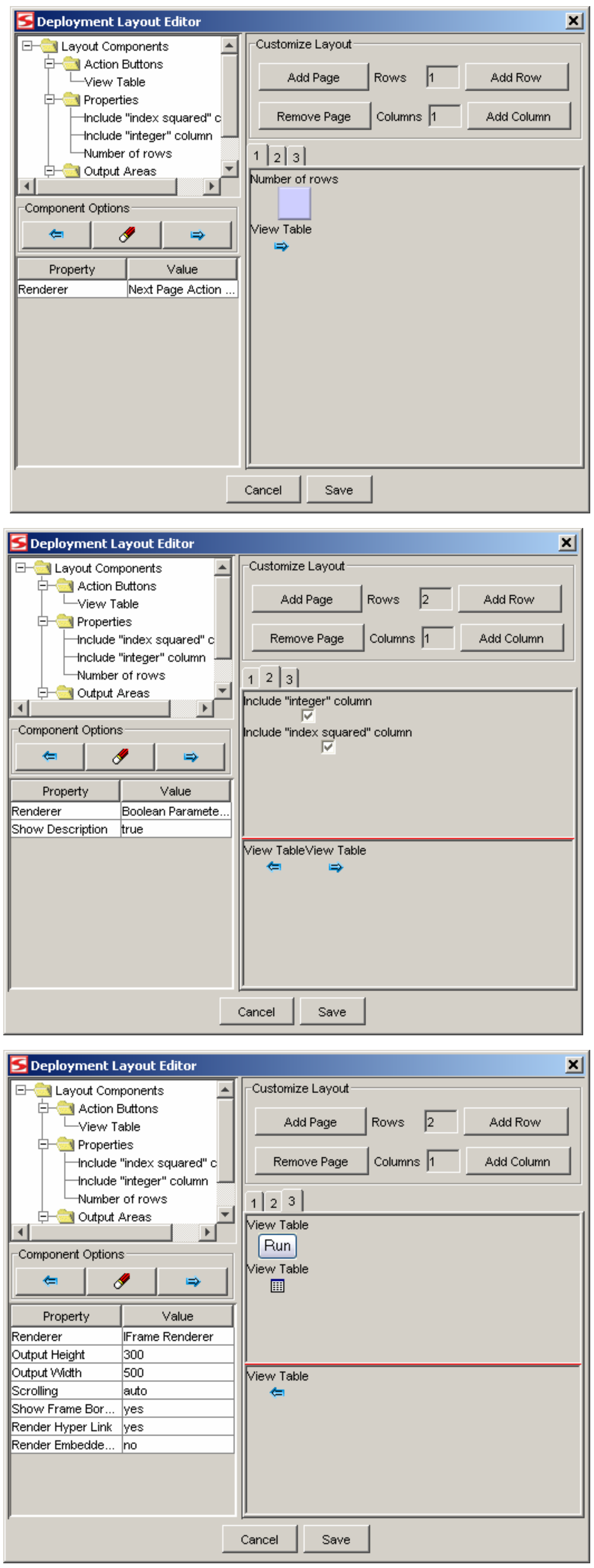
Output for action: View Result

Figure 4.17: A custom layout for an image analysis service, allowing both the input and result images to be shown and compared.



Property	Value
Height	
Width	
Alignment	left
Vertical Alignment	top
Component Flow	horizontal
Style	

Figure 4.18: The Deployment Layout Editor. Users can drag-and-drop every element of the deployed service to their chosen place on the page.

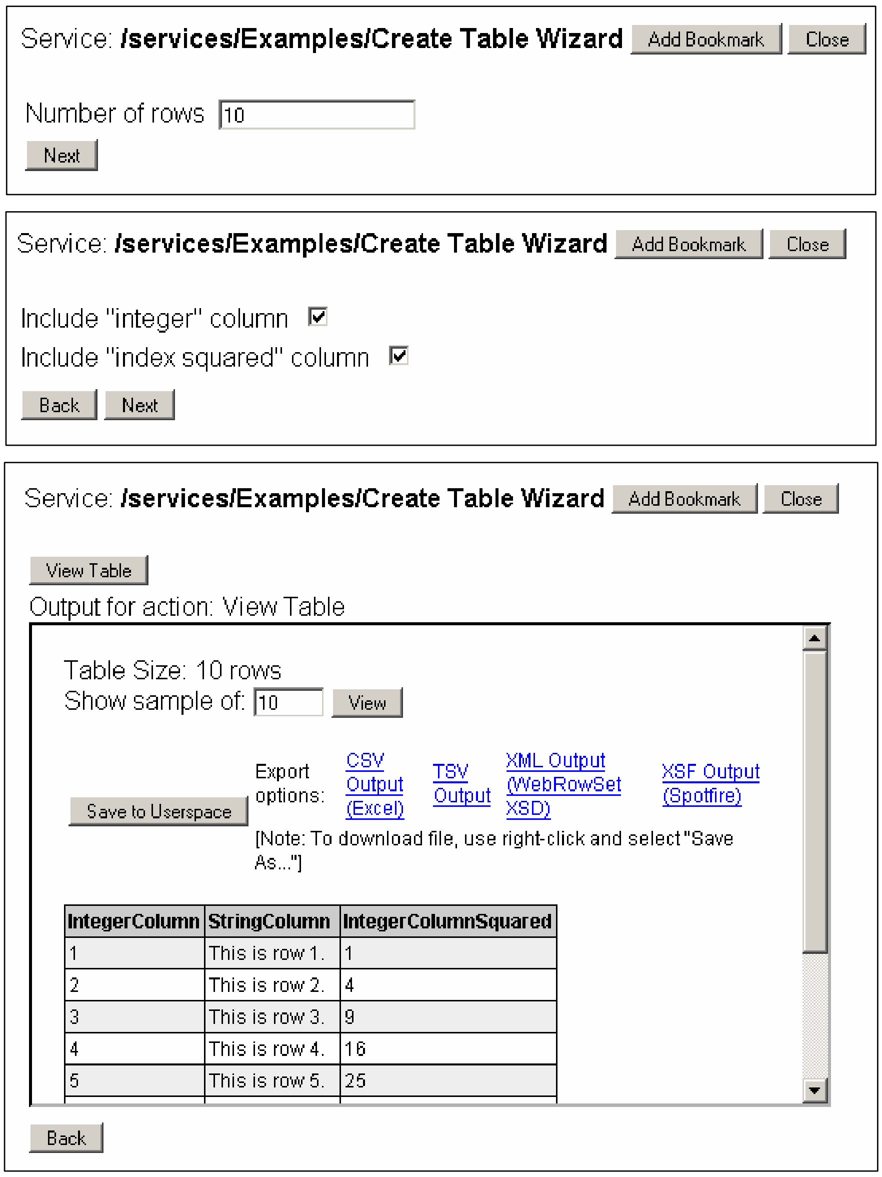


Property	Value
Renderer	Next Page Action ...

Property	Value
Renderer	Boolean Paramete...
Show Description	true

Property	Value
Renderer	IFrame Renderer
Output Height	300
Output Width	500
Scrolling	auto
Show Frame Bor...	yes
Render Hyper Link	yes
Render Embedde...	no

Figure 4.19: Creating a wizard-style service layout, with multiple pages, in the Layout Editor.


Service: **/services/Examples/Create Table Wizard** Add Bookmark Close

Number of rows 10

Next

Service: **/services/Examples/Create Table Wizard** Add Bookmark Close

Include "integer" column ✓

Include "index squared" column ✓

Back Next

Service: **/services/Examples/Create Table Wizard** Add Bookmark Close

View Table

Output for action: View Table

Table Size: 10 rows

Show sample of: 10 View

Save to Userspace

Export options: CSV Output (Excel) TSV Output XML Output (WebRowSet XSD) XSF Output (Spotfire)

[Note: To download file, use right-click and select "Save As..."]

IntegerColumn	StringColumn	IntegerColumnSquared
1	This is row 1.	1
2	This is row 2.	4
3	This is row 3.	9
4	This is row 4.	16
5	This is row 5.	25

Back

Figure 4.20: Wizard-style service layout from Figure 4.19 shown as three pages in the Portal.

One feature that was considered but has not been implemented was the ability to conditionally hide some elements of the service. This would be useful for services with branching possibilities: an initial execution of an early stage of the service might close off or enable other deployed actions and properties. Being able to show properties only as they became relevant would improve usability. Currently, either the unused properties would have to be shown and ignored by the user, or the service would have to be split up into multiple separate deployed services, as discussed in Section 4.4.

4.3 Discovery Net's demonstration scientific applications.

The Discovery Net project included a number of demo applications developed in collaboration with research groups at Imperial College London and DeltaDOT Ltd.[92]. Our development of the deployment features discussed in this chapter was greatly influenced by the requirements of these applications and their supporting workflows.

In this section, we introduce the three applications - GUSTO pollution monitoring, GM Crop monitoring, and Earthquake analysis - and show how their Discovery Net workflows were deployed to the portal.

4.3.1 GUSTO

GUSTO is a Discovery Net project based in the Department of Physics at Imperial College London, developing sensors for environmental monitoring. The sensors determine atmospheric concentrations of particular pollutants (sulphur dioxide, nitrous oxides, ozone, and benzene) to parts-per-billion (ppb) resolution, by measuring UV absorbance by the atmosphere[159]. Multiple distributed sensors would form a monitoring network, producing a constant flow of real-time data; the relatively-cheap GUSTO sensors would be numerous and take frequent readings (every 2 seconds), resulting in a much higher spatial and temporal resolution than is possible in most pollution monitoring networks. Such a network requires data warehousing, visualisation and mining tools to examine both immediate changes and long term trends (Figure 4.21): some of these could be provided by the Discovery Net platform.

This project's aim was to simulate the use of Discovery Net in combination with a GUSTO sensor network to store and process pollution data. Analysis workflows were built to mine the pollution data for trends, and visualisation components were developed to allow the user to explore the data and results interactively. The data and analysis services were then made accessible through the Discovery Net web portal.

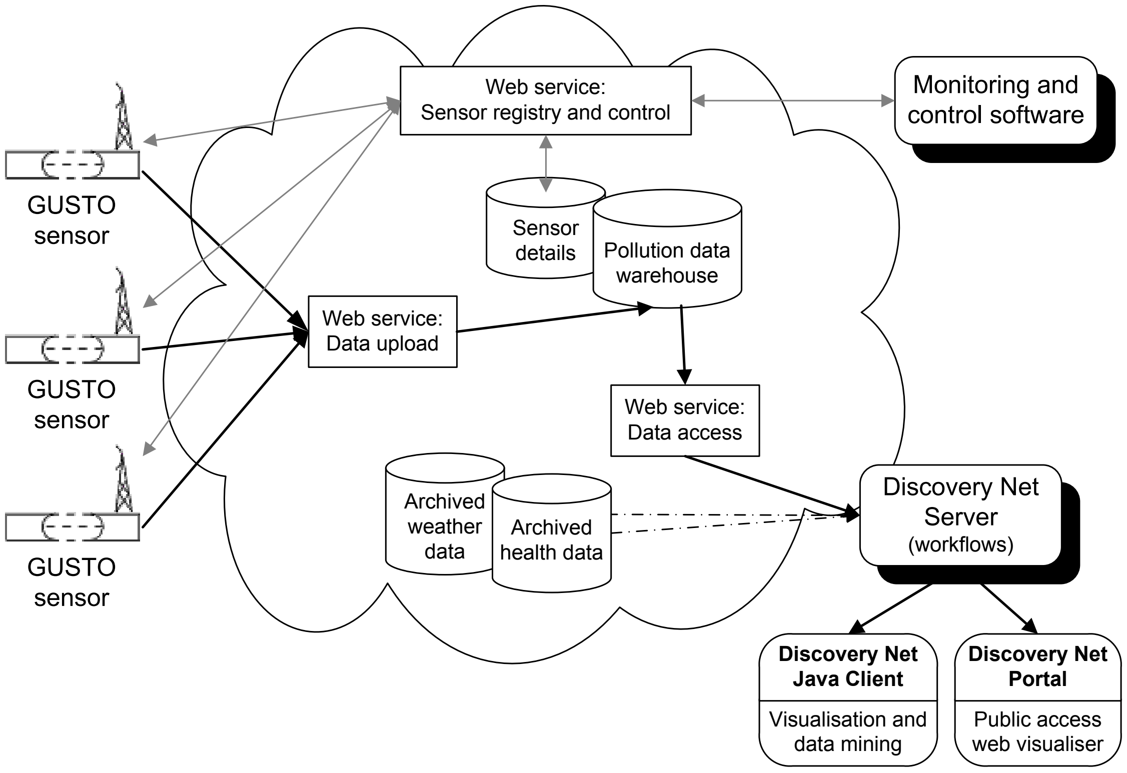


Figure 4.21: GUSTO sensor network architecture.

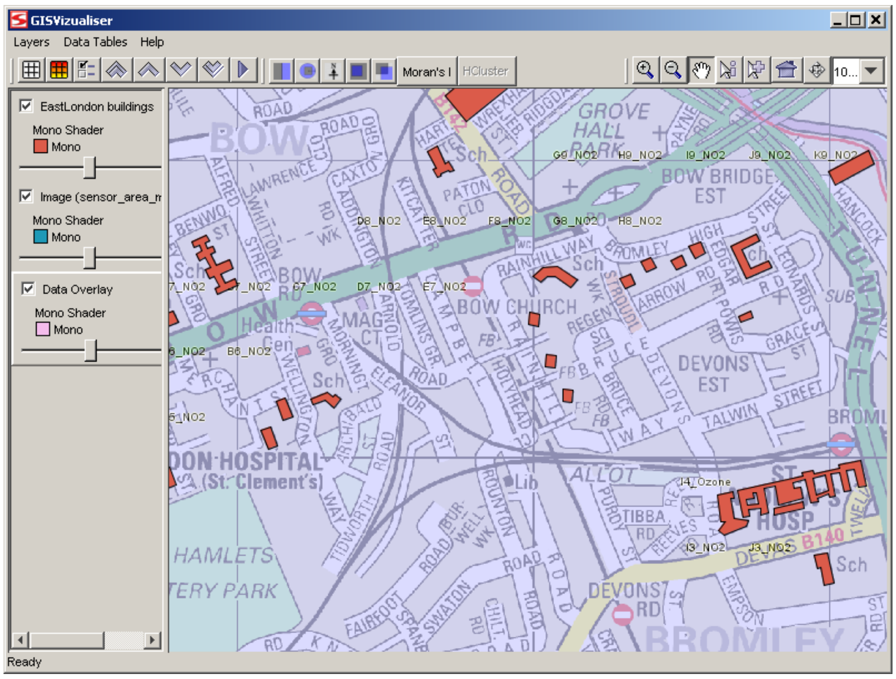


Figure 4.22: One of the identified sensor clusters shown over a map. The Sensor IDs and common pollutant are shown over the sensor positions. This cluster contains sensors along main roads.

A common way of storing and displaying information with a geographical component is by using GIS (Geographical Information Systems)[44] software. This allows data linked to a map to be stored along with the map itself in a vector format. Discovery Net's GIS map viewer (GISViz), originally developed by Salman AlSairafi[170] supports one of the older GIS formats: 'shapefiles', developed by ESRI[40]. The GIS viewer displays both the data stored in the shapefiles and a table of data from Discovery Net, which may be linked to the map vector objects in the shapefile. However this implementation could not initially display data changing over time, as would be required for the GUSTO data, and several new features were requested by the GUSTO team. Some of these new visualisations and the animation feature are shown in Figure 4.22 - Figure 4.24.

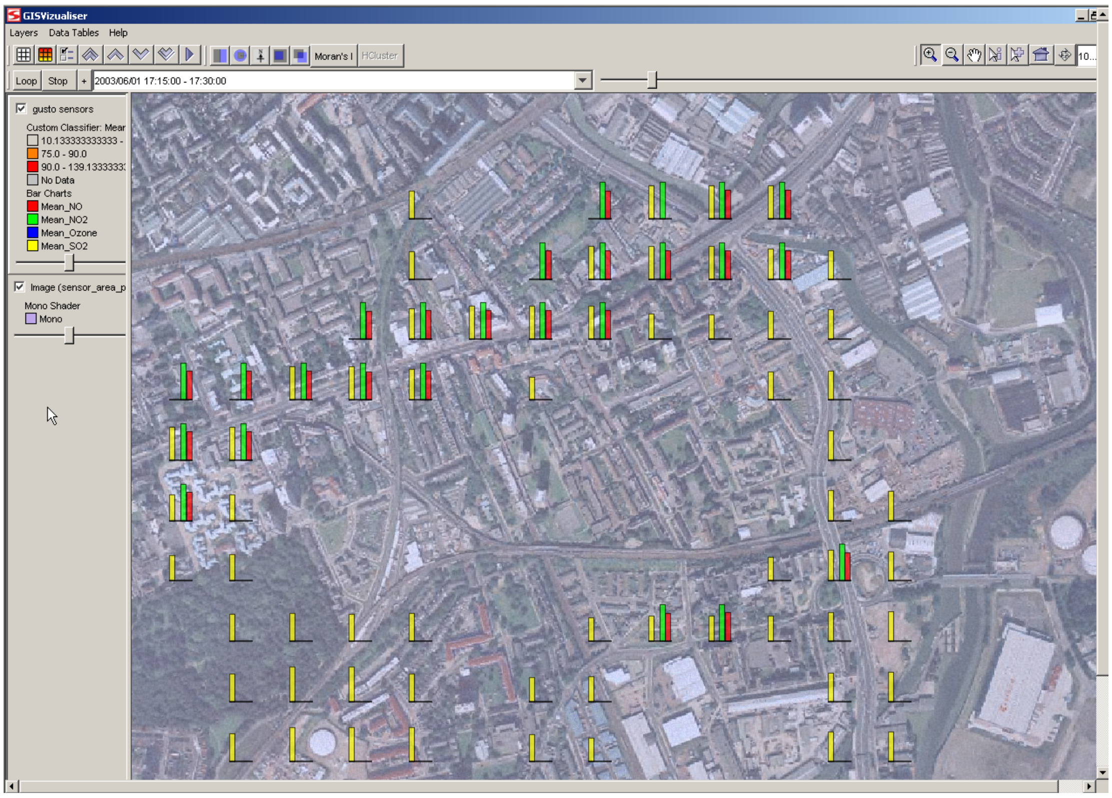


Figure 4.23: GISViz provides many display configuration options. Here, it has been set up for a quick overview, using data that has been averaged over 15 minute intervals. Each sensor location is represented by a bar chart showing the concentration levels for the four pollutants measured. Only readings of dangerously high pollution levels are shown; threshold filtering hides those with lower levels. In the background, an aerial photo has been loaded instead of the vector maps previously used.

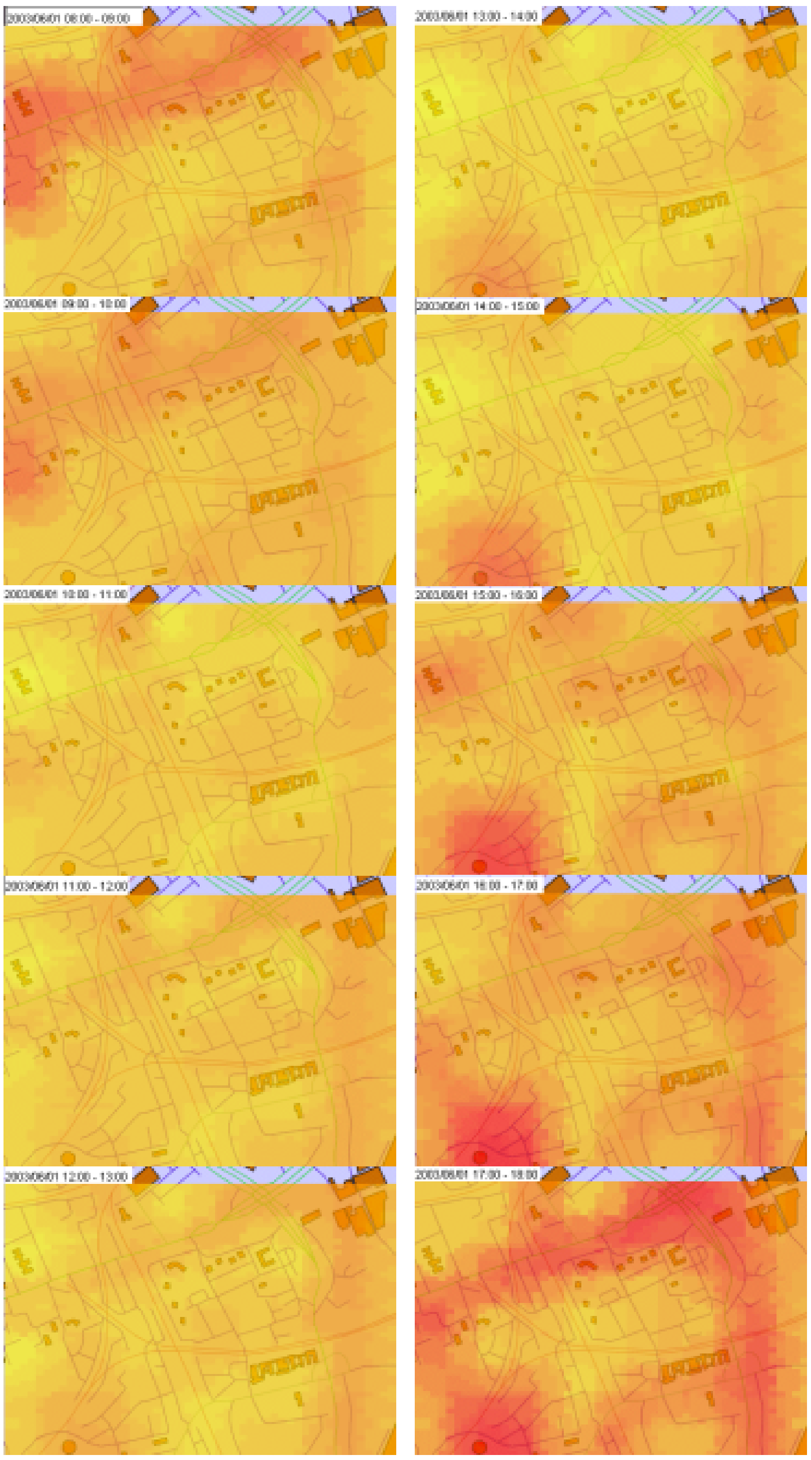
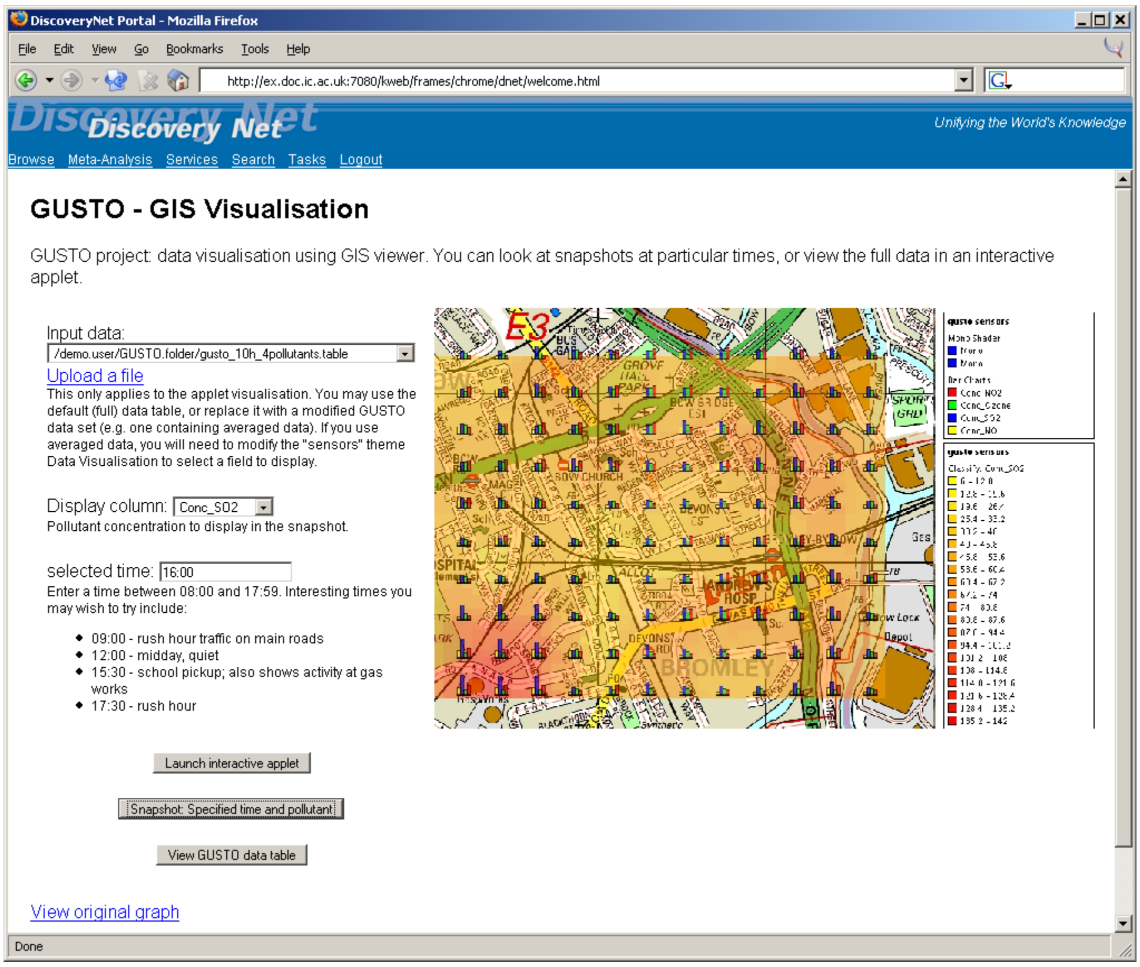


Figure 4.24: Snapshots of GISViz animation, showing how the pollution level changes over time, using the "Surface Layer" visualisation. This visualisation approach is intuitively grasped by users, although it relies on interpolation between sensors and can only show one pollutant at a time.

Deployed versions of the workflows in the web portal are shown in Figure 4.25 and Figure 4.26. The most significant work in this area was in web-enabling the GIS visualiser. The GISViz application embedded in the Discovery Net Java Client was extended and modified to allow its use as an applet in the remote client browser, for interactive visualisation of the service results. In addition, alternate services were produced which generated image snapshots of the GIS map view: these services thus returned simple images as results, which are faster and less demanding on the client machine than the applet approach, though less interactive.


DiscoveryNet Portal - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://ex.doc.ic.ac.uk:7080/kweb/frames/chrome/dnet/welcome.html
Discovery Net
Unifying the World's Knowledge
Browse Meta-Analysis Services Search Tasks Logout

GUSTO - GIS Visualisation

GUSTO project: data visualisation using GIS viewer. You can look at snapshots at particular times, or view the full data in an interactive applet.

Input data: /demo.user/GUSTO.folder/gusto_10h_4pollutants.table

Upload a file

This only applies to the applet visualisation. You may use the default (full) data table, or replace it with a modified GUSTO data set (e.g. one containing averaged data). If you use averaged data, you will need to modify the "sensors" theme Data Visualisation to select a field to display.

Display column: Conc_SO2

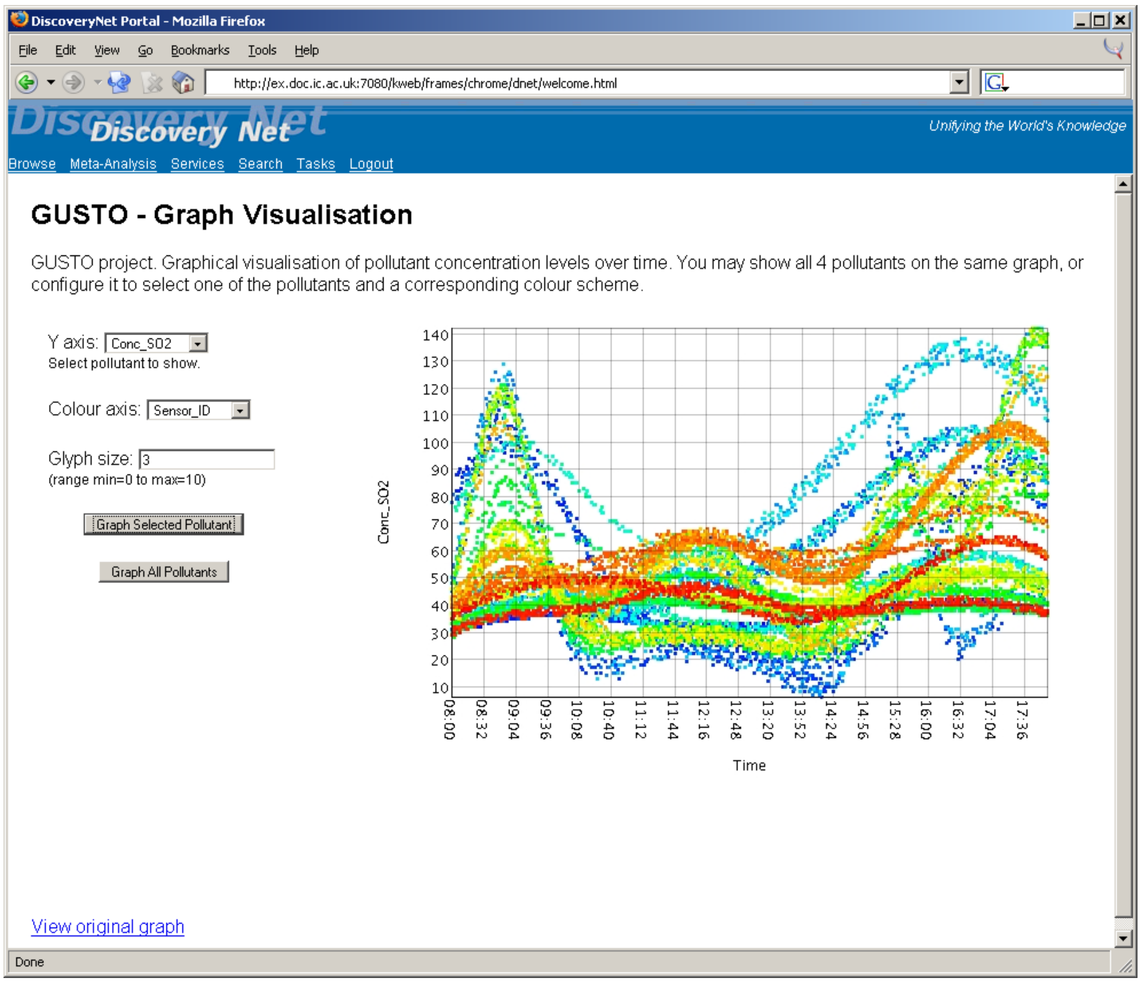
Pollutant concentration to display in the snapshot.

selected time: 16:00

Enter a time between 08:00 and 17:59. Interesting times you may wish to try include:

- 09:00 - rush hour traffic on main roads
- 12:00 - midday, quiet
- 15:30 - school pickup; also shows activity at gas works
- 17:30 - rush hour

Figure 4.25: A GUSTO service allows the user to retrieve the pollution sensor data for a requested time period and pollutant. An image snapshot of a GIS visualisation is presented to the user. The alternative output uses the GISViz applet, allowing the user to adjust the map and data layers and animate the display. The user may also choose to retrieve the underlying data table.


DiscoveryNet Portal - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://ex.doc.ic.ac.uk:7080/kweb/frames/chrome/dnet/welcome.html
Discovery Net
Unifying the World's Knowledge
Browse Meta-Analysis Services Search Tasks Logout
Graph Selected Pollutant
Graph All Pollutants
Conc_SO2
Time
View original graph
Done

GUSTO - Graph Visualisation

GUSTO project. Graphical visualisation of pollutant concentration levels over time. You may show all 4 pollutants on the same graph, or configure it to select one of the pollutants and a corresponding colour scheme.

Y axis: Conc_SO2
Select pollutant to show.

Colour axis: Sensor_ID

Glyph size: 3
(range min=0 to max=10)

Figure 4.26: A GUSTO service which graphically presents a chosen pollutant's concentration over time.

My work in this project focused mainly on the enhancements to the GISViz viewer, and subsequently creating the workflows and deployed services for the web portal. This development process revealed the difficulty of separating existing visualisers from the Java Client in which they were embedded, as all dependencies from the core system must be removed for the visualiser to be packaged as an independent applet. It also highlighted concerns with security and data flow, as the applet needed to be able to download maps and data from the Discovery Net server. Thus full authentication information (username, password and Discovery Net server address) had to be provided to the applet using applet parameters. However, the applet could not be allowed to directly contact the Discovery Net server, as this would not work in the scenario where the browser is accessing a public web portal but the Discovery Net server itself is hidden behind a firewall. Thus the portal had to act as a gateway through which the applet could access userspace files (maps and data). In addition, the portal had to initially acquire the applet JARs from the Discovery Net server and serve them to the client browser. For this, the Discovery Net server had to provide an API for downloading files which were not part of any userspace, but instead were resources associated with particular nodes.

We also investigated the possibilities of using Discovery Net Portal version 2.0's custom parameter mechanism to present a lightweight, user-friendly way of inputting date parameters using JavaScript (Figure 4.8). The method used here is fundamentally quite simple: the real parameter will always be submitted in the service form as a single, String value. However, advanced custom parameter renderers can hide this real parameter's input field, and add new elements to the form for the user to interact with. When the user has set a value using the custom interface, the JavaScript inspects this value and serialises it to a String, setting it as the value of the hidden parameter input. When the form is finally submitted, the portal ignores the extra form elements and looks only at the values of 'real' inputs with names it expects to receive.

This work has been demonstrated at several conferences as part of Discovery Net (All Hands Meeting 2004[152], Fourth International Workshop on Environmental Applications of Machine Learning, EAML 2004[151]), and published in the ECEM/EAML issue of Ecological Modelling[163].

4.3.2 GM Crop Monitoring

DeltaDOT Ltd.[92] is a biotech company spin-off from Imperial College London's High Energy Physics department. Its novel biochip technologies allow fast, high-throughput and label-free mapping of DNA and proteins using biosensors. In collaboration with Discovery Net, it has developed use cases and provided sample data for analysis and mining by the Discovery Net system.

This Discovery Net application concerns monitoring farm fields for contamination by GM crops. The scenario examined an area centred around a known GM crop test field. Samples would (theoretically) be taken by hand from many fields in the surrounding area over a number of days, and each sample would be tested for the presence of GM material.

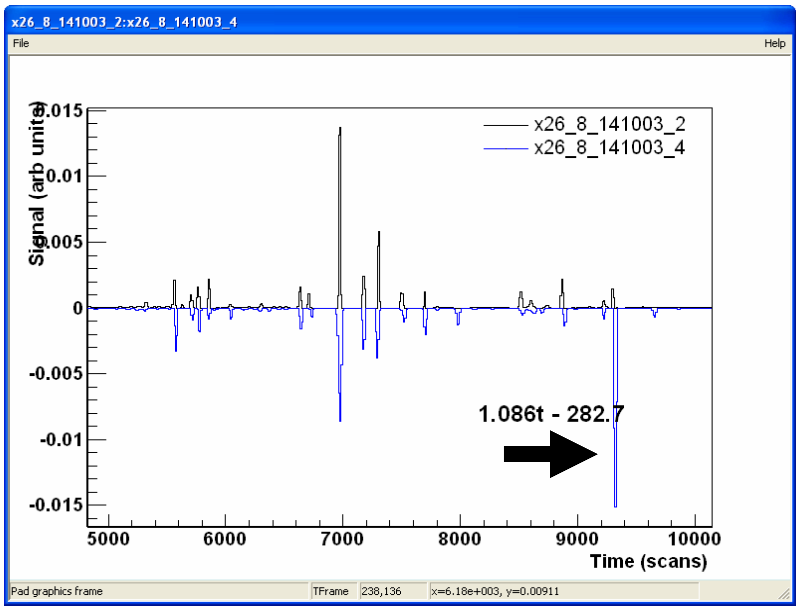


Figure 4.27: A mirror plot for comparing two protein profiles. The additional protein in the GM sample is indicated.

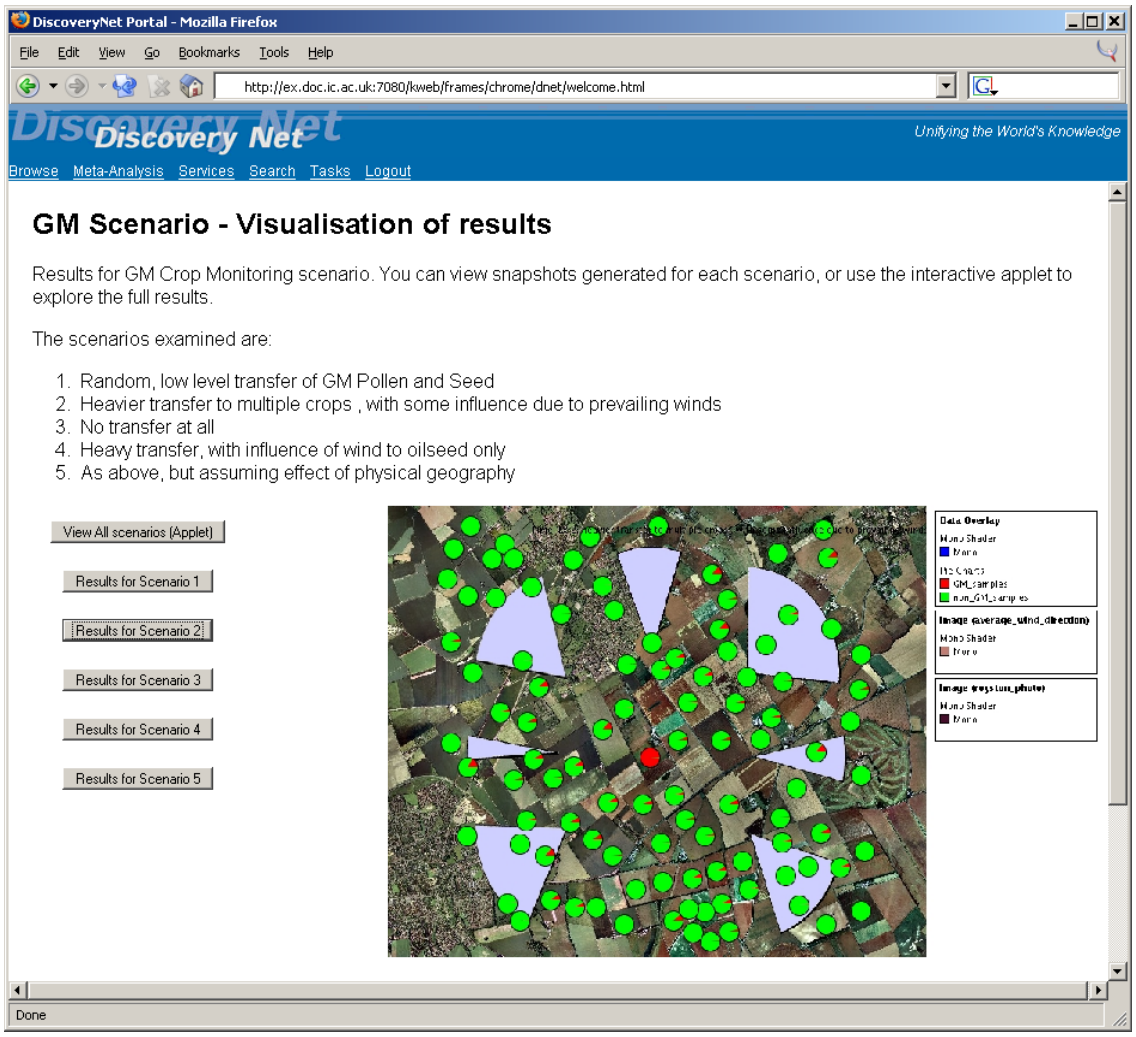
The testing would be performed using DeltaDOT's biosensor technology, which uses capillary gel electrophoresis to identify the components of a cell lysate sample. The result of the analysis is a profile of the proteins contained in each sample, with each protein distinguished by its mass. Figure 4.27 shows the mirror plot used to compare samples; the presence of GM proteins in a sample would suggest contamination by GM seeds or crossbreeding.

Hence, the main analysis in this application focuses on differences between protein profiles, which involves the presence or absence of proteins, or changes in the amount (up/down-regulation) of a particular protein.

Several sets of simulated data were provided by DeltaDOT, each using different initial assumptions, and I developed the necessary workflows and visualisations to process the data. The result of this demonstration was similar to that from GUSTO, with an analysis workflow to analyse and collate the data, visualisations in GISViz, and a focus on the portal for dissemination of data and results (Figure 4.28, Figure 4.29).

The simulated results were visualised using the Discovery Net GIS map visualiser (GISViz) to show the distribution of GM material in different fields for five different scenarios. This required the development of support in GISViz for displaying data associated with arbitrary locations, in contrast to the GUSTO sensors whose fixed positions were displayed using a custom map layer (shapefile).

This project has been demonstrated at conferences as part of Discovery Net, and was published as part of the proceedings of All Hands 2004[169].


DiscoveryNet Portal - Mozilla Firefox
File
Edit
View
Go
Bookmarks
Tools
Help
http://ex.doc.ic.ac.uk:7080/kweb/frames/chrome/dnet/welcome.html
Discovery Net
Unifying the World's Knowledge
Browse
Meta-Analysis
Services
Search
Tasks
Logout

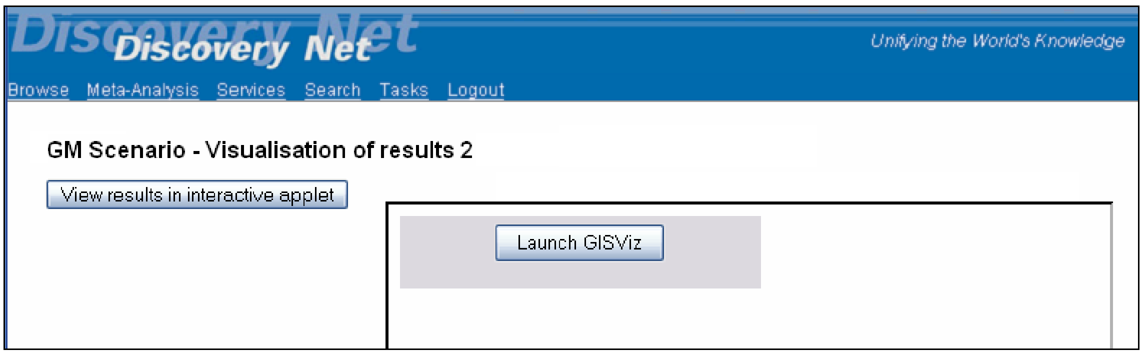
GM Scenario - Visualisation of results

Results for GM Crop Monitoring scenario. You can view snapshots generated for each scenario, or use the interactive applet to explore the full results.

The scenarios examined are:

1. Random, low level transfer of GM Pollen and Seed
2. Heavier transfer to multiple crops , with some influence due to prevailing winds
3. No transfer at all
4. Heavy transfer, with influence of wind to oilseed only
5. As above, but assuming effect of physical geography

Figure 4.28: Visualisations of the scenario results deployed in the portal as GISViz snapshots.


Discovery Net
Unifying the World's Knowledge
Browse
Meta-Analysis
Services
Search
Tasks
Logout
GM Scenario - Visualisation of results 2
View results in interactive applet
Launch GISViz

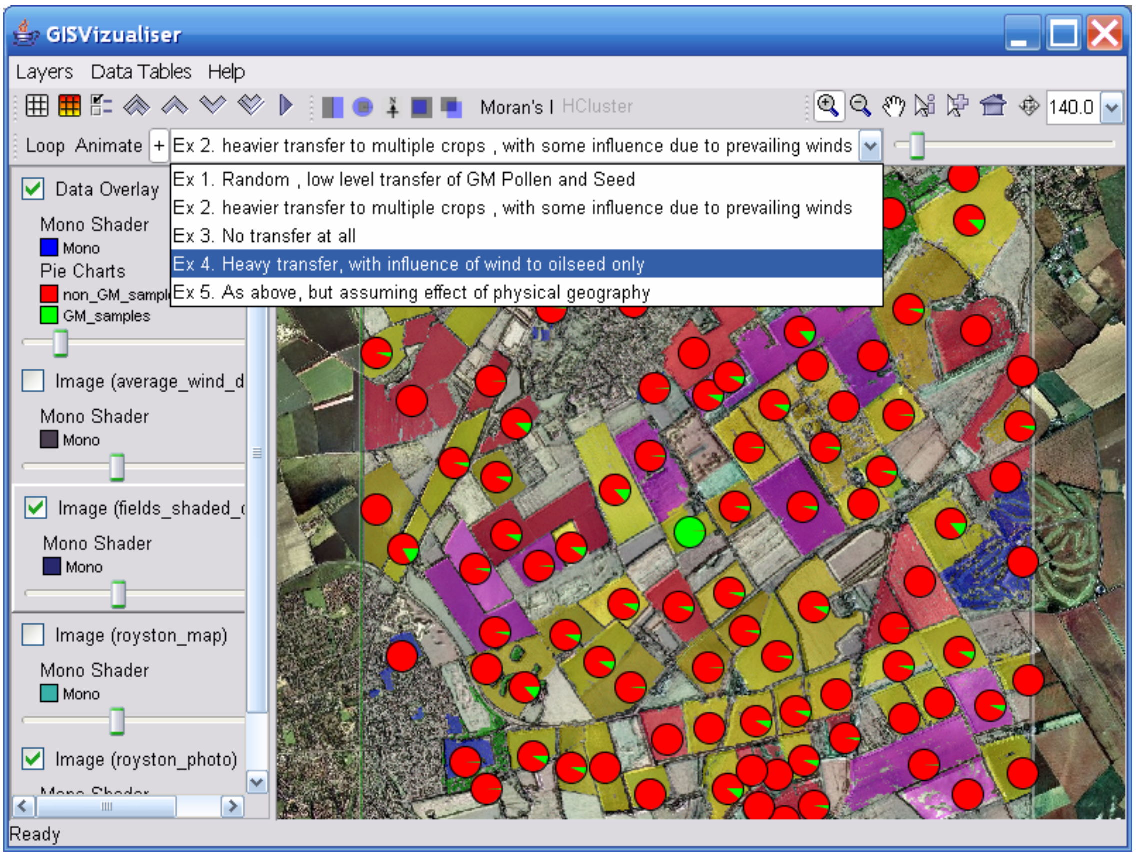

GISVizualiser
Layers
Data Tables
Help
Moran's I
HCluster
140.0
Loop
Animate
Ex 2. heavier transfer to multiple crops , with some influence due to prevailing winds
Ex 1. Random , low level transfer of GM Pollen and Seed
Ex 2. heavier transfer to multiple crops , with some influence due to prevailing winds
Ex 3. No transfer at all
Ex 4. Heavy transfer, with influence of wind to oilseed only
Ex 5. As above, but assuming effect of physical geography
Data Overlay
Mono Shader
Mono
Pie Charts
non_GM_sampl
GM_samples
Image (average_wind_d
Mono Shader
Mono
Image (fields_shaded_
Mono Shader
Mono
Image (royston_map)
Mono Shader
Mono
Image (royston_photo)
Ready

Figure 4.29: The GISViz applet deployed in the portal allows the user to interactively explore the results. They may switch between and compare scenarios with different initial conditions, add/remove backgrounds, and filter results by specified thresholds.

4.3.3 Earthquake Analysis

The Earth Sciences Discovery Net application involves the monitoring of geohazards such as earthquakes using satellite images.

Pairs of images (Figure 4.30) of the same location, taken at different times, are analysed to determine how the landscape has changed. By doing this a picture can be built up of how the surface is moving over a wide area, which is useful for modelling faults and predicting future movements.

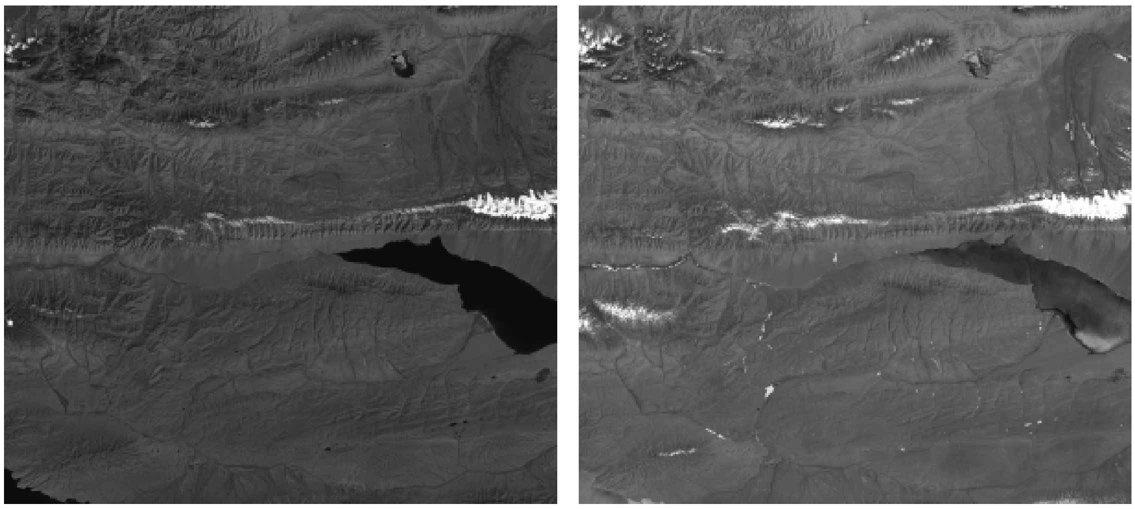


Figure 4.30: The two images of the same area (Three Gorges Dam/Reservoir Region in China) used for comparison. The source images used in the analysis are each 32 MB in size.

The analysis process identifies the movement of each pixel for each image pair, using an image shift algorithm developed by Dr Jinming Ma of the Earth Sciences department at Imperial College London. This analysis of the large source images is intensive and requires cluster processing for reasonably fast results. This was done using 20 computers in the Viking cluster at Imperial (London e-Science Centre[80]). For display, the image is partitioned and the movement averaged to present the result as arrows overlaid on the original image, showing the direction and magnitude of movement. This work has been described in detail in several papers[147,146,142].

I integrated this image shift analysis program into the Discovery Net architecture so that more advanced analysis workflows could potentially be built upon the existing single analysis task. In use, the analysis workflow developed would be executed multiple times with different parameter settings for both the image shift analysis and the result display. I developed and deployed workflows allowing this complete analysis procedure to be conducted through the web portal (Figure 4.31).

Because of the relatively long run time for the image shift algorithm, even with small source images, this application highlighted the requirement for caching when using deployed workflows; at the time of this demo development, services were not stateful and so a workflow consisting of multiple stages (e.g. to inspect a visualisation and then download the raw data, or proceed with further analysis) would need to execute from the beginning every time it was run. This research approach was later supported in Discovery Net version 3.0 by the introduction of stateful services, which cached intermediate results between executions.

a)

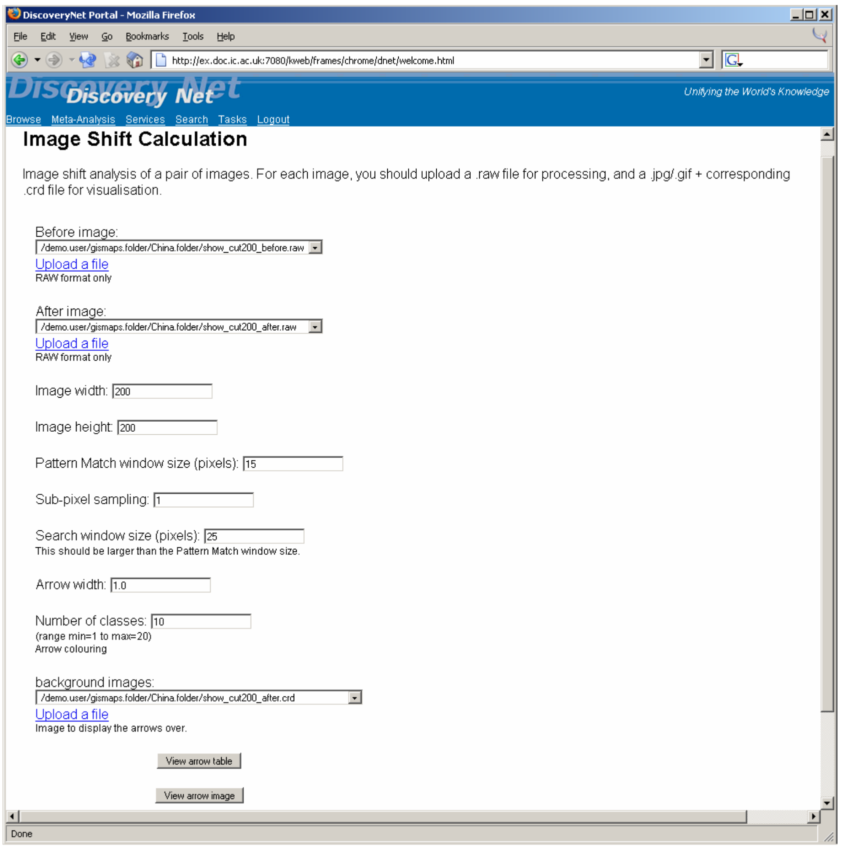

DiscoveryNet Portal - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://ex.doc.ic.ac.uk:7080/kweb/frames/chrome/dnet/welcome.html
Discovery Net
Unifying the World's Knowledge
Browse Meta-Analysis Services Search Tasks Logout

Image Shift Calculation

Image shift analysis of a pair of images. For each image, you should upload a .raw file for processing, and a .jpg/.gif + corresponding .crd file for visualisation.

Before image: /demo.user/gismaps.folder/China.folder/show_cut200_before.raw
Upload a file
RAW format only

After image: /demo.user/gismaps.folder/China.folder/show_cut200_after.raw
Upload a file
RAW format only

Image width: 200

Image height: 200

Pattern Match window size (pixels): 15

Sub-pixel sampling: 1

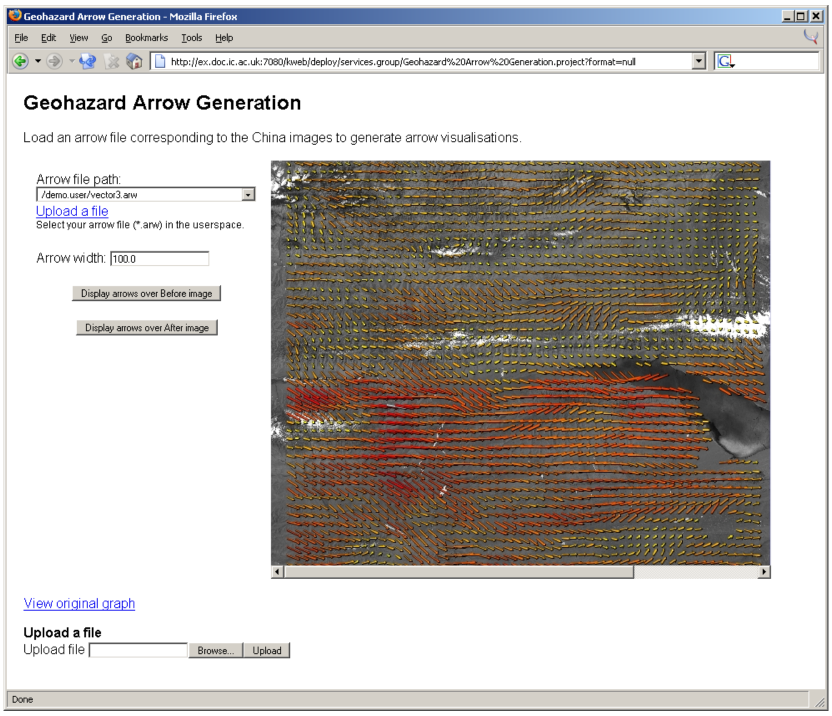
Search window size (pixels): 25
This should be larger than the Pattern Match window size.

Arrow width: 1.0

Number of classes: 10
(range min=1 to max=20)
Arrow colouring

background images: /demo.user/gismaps.folder/China.folder/show_cut200_after.crd
Upload a file
Image to display the arrows over.

b)


Geohazard Arrow Generation - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Browse...
Upload
Display arrows over Before image
Display arrows over After image
Done

http://ex.doc.ic.ac.uk:7080/kweb/deploy/services.group/Geohazard%20Arrow%20Generation.project?format=null

Geohazard Arrow Generation

Load an arrow file corresponding to the China images to generate arrow visualisations.

Arrow file path: /demo.user/vector3.arw

Upload a file

Select your arrow file (*.arw) in the userspace.

Arrow width: 100.0

View original graph

Upload a file

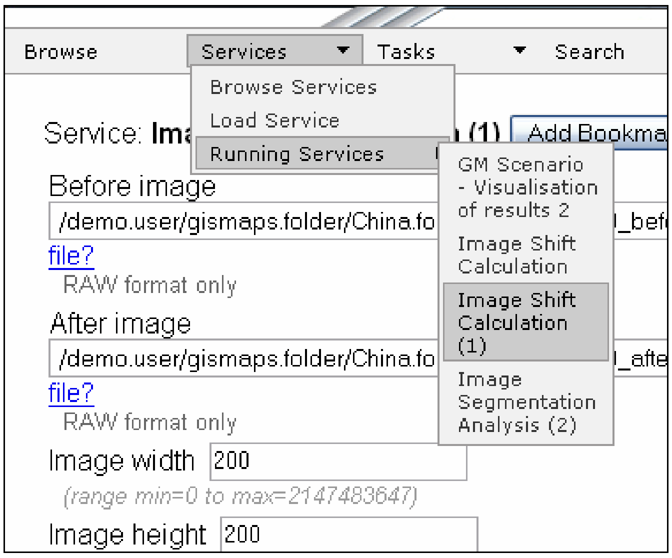
Upload file

Figure 4.31: The Earthquake application allows users to a) first apply the image analysis algorithms to detect movement by comparing two input images b) tweak the result display by modifying arrow rendering. This process could easily be expanded to include further image processing steps.

In this demo, Discovery Net is used mainly to orchestrate the execution of processes on remote resources, and theoretically the demo could be extended to use a number of workflows representing different stages in the analysis process. Each workflow could be deployed as a service, and the services used in sequence, passing data from one to the next, or repeating stages with different parameters if necessary. Thus it clearly showed a need for connecting data flows between multiple services, so that results from one service execution could be passed to another service as input. This informed our later development of communicating Discovery Net Service portlets.

4.4 Using multiple services

Several of the applications studied can use multiple services in sequence to perform a larger task. An alternative approach is to include all analysis steps in a single workflow, and deploy that workflow as a single service with multiple sequential outputs - version 3.0 of Discovery Net stores the state of each deployed service between invocations (Figure 4.32), so such a workflow can be executed in stages without having to re-do all the steps from the beginning every time.



Service: Ima (1)

Before image /demo.user/gismaps.folder/China.fo _bef
RAW format only

After image /demo.user/gismaps.folder/China.fo _afte
RAW format only

Image width 200
(range min=0 to max=2147483647)

Image height 200

Figure 4.32: Discovery Net v.3 supports independent instances of deployed services, each maintaining its own state.

The approach of developing multiple separate services makes sense when the individual steps may be recombined and reused as components of a different analysis procedure, or when the steps themselves can each be used independently. This approach is more flexible, but adds two complications: the user of the web interface will need help navigating between different deployed services in the right order, and results must be passed from one service to the next.

This is a very common situation, which emerges in any use of online analytical services, not just Discovery Net. For example, users of public bioinformatics services may use multiple websites (e.g. NCBI[101], EBI[94]), standalone programs, and/or programs installed on local networks, and usually have to manually pass information between tools by copy-and-pasting results.

Data can also be passed between Discovery Net services by copy-and-pasting. A more efficient method, particularly for large amounts of data, is to have one service save its result to the userspace, and then the user can select that data file as input to later services. Again, however, the user must explicitly control the data flow - first naming the result file, and then selecting it as input. This tedious situation is actually one which the use of Discovery Net was originally intended to avoid, by orchestrating disparate tools in a single workflow system.

The standard Discovery Net Portal does not have any special support for guiding the user between related services, or for passing data between services (other than that outlined above). In this original portal, all services are accessible from one global Service Index page, in which services are ordered by their location in the userspace. This simple approach becomes unwieldy when there are many services to look through, or when the user must repeatedly return to the index page to find the next service in a sequence.

The use of portlets can help in both these areas, as discussed in Chapter 3. However, this is an area which could still use further research to examine orchestration of Discovery Net services at the server tier rather than the portal tier.

4.5 Conclusion

In this chapter, we have discussed the current popularity of web interfaces for accessing remote services, and described how interfaces can be dynamically generated to access well-described services such as Web Services using WSDL. We have detailed the current problems with such approaches and the importance of service metadata in generating user-friendly web interfaces.

We have covered Discovery Net's 'deployment' of services as encapsulated workflows. The necessity of customised service interfaces, rich input methods and result visualisation have been outlined, with our implementation approaches.

We then described in some detail Discovery Net's example scientific applications, which were the drivers and demonstrations for much of this deployment functionality. They primarily focused on deployment of analysis workflows and the details of the web interface, particularly developing web-deployable versions of result visualisers (e.g. the GIS map viewer) using applets. They also revealed the need for stateful services (caching intermediate result data for use in further executions of the same service), which was introduced in version 3.0 of Discovery Net, and for transmission of data between services (a service result being used as input to another service).

We concluded by summarising the issues involved with composing multiple services together in a larger analytical pipeline, which has particular relevance to the use of portlets (Chapters 3 and 5).

Chapter 5. Inter-portlet communication

One of the commonest queries from new portlet developers is how to create a link in one portlet which can trigger a response in a different portlet (typically an Index portlet which triggers the display of an item in a Display portlet, illustrated in Figure 5.1). The problem stems from the way the portal has to keep portlets from interfering with each other: clicking a link in one portlet results in a request which is processed by only that portlet, so all other portlets are completely unaware of the interaction, and cannot see any parameters that might have been passed. Thus, for this scenario, the first portlet needs a way to explicitly send a message to the second, informing it of what action to perform or information to render. This functionality is commonly referred to as inter-portlet communication (IPC).

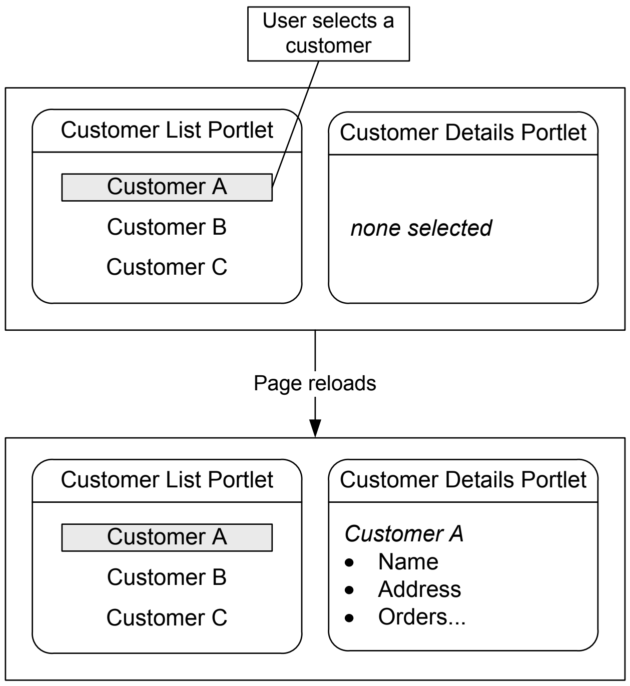


Figure 5.1: Scenario: communication between two portlets

However, the method of inter-portlet communication is not defined in the current portlet standard (JSR-168), although it will be in the next version. Almost every portal has its own custom implementation, with different messaging behaviours and APIs. Non portal-specific (i.e. JSR-168-compliant and portable) solutions are possible, but require the portlet developer to implement their own IPC system.

Another common request is for a hyperlink in one portlet to navigate to a different portal page or portlet. Initially this seems trivial, it being the normal and expected behaviour in non-portlet-based websites: a hyperlink may reload the same page (perhaps with different parameters in the URL), but is more likely to take the user to a different page on the server. However in the context of a portlet, there is a fundamental piece of information which is unknown at portlet development time: the URL of the target portal page. This is because the portlet is likely to be developed well in advance of the portal(s) it may eventually be added to: it is developed as a reusable component with no knowledge of its eventual environment(s). Hence, a single portlet is usually designed to be self-contained; a link within a portlet usually targets itself with new parameters that modify its actions or display, and the generation of this URL is abstracted and delegated to the portal.

When communication between portlets is introduced, the approach of generating self-referencing links in portlets becomes inadequate, as it may make more sense to navigate to a different portal page rather than reloading the same one (for example, if the Display portlet is on a different page to the Index portlet, it is convenient for a selection on the Index portlet to additionally navigate to the Display page). Therefore, navigation between pages is a feature sometimes included and discussed under the topic of IPC as it is often an associated necessity. In this thesis we do not include navigation between portal pages as a critical component of IPC; however, we do discuss how limited page navigation might be supported (Section 5.4).

IPC is not appropriate for all situations: sometimes it may be better to develop one portlet which can perform multiple tasks in different modes, rather than a collection of portlets that communicate amongst themselves. In particular, if the messages to be sent between portlets are large or frequent, and the portlets are tightly coupled and must always be used together in a particular arrangement on the page, this may indicate that the portlets would be more easily merged into one large portlet. However, if the portlets are related but only loosely coupled - each being responsible for well-defined and largely independent tasks - it may be more appropriate to treat them as components, linked using IPC.

In this chapter we will discuss some of the portal-specific IPC solutions available, the possible approaches for IPC implementations within JSR-168, and finally our development of a JSR-168-compliant IPC library.

5.1 Portal-specific IPC Implementations

Portals which implement their own custom inter-portlet communication mechanisms provide a communications API which allows the portlet developer to concentrate simply on sending the messages, without needing to know about the back-end implementation.

Often, the process of linking portlets together with messaging must be performed by a portal developer or admin, using portal-specific configuration tools or APIs. This is acceptable for static site layouts controlled by the portal admin, but does not easily allow changes later. In contrast, the most advanced implementations of IPC can be configured using web-based control panels at runtime, allowing the end user of the portal to set up communication between portlets.

As discussed in Section 7.3.3.1, there is some variation in the treatment of portlet instances, and this affects the way IPC is configured in different portals. Most portals treat each portlet window as an independent source and sink of messages, and this is reflected in their IPC configuration tools, where configuration is done in parallel with creating pages and placing portlet windows. However, some portals are more restrictive and consider the portlet entries in the `portlet.xml` registry as the true portlet instances, so the IPC configuration is tied directly to these and is not associated with the page layout or portlet window IDs.

Some portals have extended their implementation of JSR-168 to support their own IPC system; many simply continue to support their own non-JSR-168 portlets, which often include native IPC support preceding JSR-168 by several years. In either case, use of the portal-specific IPC solutions, even those that extend JSR-168, will result in portlets which can only be deployed on that portal. We outline the features and approaches used by a number of portal-specific IPC implementations below.

5.1.1 BEA WebLogic[35]

BEA WebLogic manages portlet communication using a system similar to Java Listeners[134]. Portlets can be configured to listen to another portlet, identified using the source portlet window's unique ID, which is generated by the Portal. Portlet state is managed with explicit "page flows", which are defined using an editor program. A message is triggered by a user clicking on a link which corresponds to a named action in the page flow. When such a link in a source portlet is clicked, the corresponding function is also called in all listening portlets.

This approach requires the exact communication links between portlet windows to be specified during portlet development.

5.1.2 IBM WebSphere[46]

IBM WebSphere's system for portlet communication is called "message events". Portlets which are in the same portlet application can send and receive messages (which may consist of any Java object), addressed using portlet window IDs. Portlets may implement a `MessageListener` interface, which includes a function that will be called when a message is received.

To process messages, the Portal manages the execution of the following, each time a page loads:

1. all action events (these may send message events)
2. all message events, executing Listener functions on the receiving portlets (these may send more message events, which will be processed until no more are left)
3. all window events (requests to change window state)
4. all renders (any messages sent during this phase will be ignored)

This procedure allows a message to trigger other messages, in a branching message chain which is fully processed before the page renders. Because messages may only be sent in the Action phase, and all message events are dealt with before the render phase begins, the message state is consistent and safe from any further changes when the portlets finally render.

WebSphere's advanced IPC system allows portlets to be developed without specific reference to particular portlet IDs or even agreed message names; the portlets are thus reusable components independent of any particular portal layout. Such cooperative ("Click-to-action") portlets register with a Property Broker, and a "Portlet Wiring Tool" portlet allows the Portal user to map connections between published properties and consumed properties at runtime.

5.1.3 Sybase Enterprise Portal[77]

Sybase Enterprise Portal 6.0's IPC implementation uses the Event-Listener model and is configured at page design-time using custom programs. Published message events are given global names, and are accessible across the whole portal. Limited types of data can be included in messages, as the implementation provides messages to portlets as CGI (Common Gateway Interface) form parameters.

5.1.4 Oracle Portal[66]

Oracle Portal (part of Oracle Application Server 10g) has an advanced IPC system which is configurable at runtime by any page designer. This is a custom implementation which only works with Oracle's own native portlets.

Oracle defines three different kinds of parameters available to its portlets:

- **Private**: Private parameters correspond to JSR-168 portlet parameters, and are only visible to one portlet window.
- **Public**: Public parameters are defined by the portlet programmer, and make up the portlet's published interface: the inputs and outputs of the portlet. At runtime, the page designer can assign values to these parameters (using a constant, a variable, or a page parameter).

- **Page/Page Group**: Page (Group) parameters are created and set at runtime by the page designer, and associated with a particular page or page group. The page designer can then map them to public parameters of portlets. In this way, multiple associated portlets on a page - e.g. each showing different information on a selected item - can be configured by a single change to a page parameter.

Each messaging event, with its associated public parameters, is explicitly defined by the portlet programmer as part of the portlet's published interface. At runtime, the page designer can configure the effect of the event produced by a portlet window: it can set page parameters from event parameters, and/or cause a redirect to a different page.

Like IBM WebSphere's approach, Oracle's IPC implementation allows portlets to be developed and used as components which are independent of any particular portal layout, and configured through a web interface on the portal.

5.1.5 eXo Portal[41]

eXo Portal's IPC implementation uses an extended version of the JSR-168 `PortletContext` to send messages in the Action phase, either directly to a named portlet or broadcast to all portlets in the same portlet application. `MessageListener` classes are registered in the `portlet.xml`, using eXo-specific extensions. The portlet entities referenced by name correspond directly to portlet instances in the `portlet.xml`, not individual portlet windows on pages.

5.1.6 Plumtree Portal[89]

Plumtree is unusual in using AJAX[27] to render and update the display of portlet windows on a page, so that it is not necessary to reload the entire page when a portlet renders. Implicitly this method relies on the portal providing a way to remotely update and retrieve a single portlet window's fragmentary content, outside the context of the aggregated page; this is not a usual feature in portals (see Section 7.3.2.7). This is however a rendering approach which is becoming more seriously considered, as it has advantages in efficiency and speed. It does introduce potential problems with usability and stability, due to the differences in page behaviour (e.g. bookmarks and the 'back' button may not behave as expected), and varying support for JavaScript in different browsers.

The dependence on JavaScript for page rendering allows the messaging logic layer to be moved from the server to the JavaScript on the client. Thus, a user clicking a link in a portlet will trigger a local JavaScript function call instead of a direct page submission to the server. This will in turn trigger JavaScript listeners belonging to other portlets, which can retrieve updated portlet window renders to display to the client, or redirect the client to a different page.

Thus, the standard JSR-168 portlet request handling is largely bypassed for messaging, as all message events are triggered and processed on the client. Messages are not sent within either the action or the render phase, but in JavaScript on the client, and messages can be chained as many times as needed. JavaScript message listeners may invisibly initiate any number of new portlet requests behind the scenes, to update portlet state and retrieve new window renders.

The IPC configuration (e.g. message names, implementation and registration of JavaScript listeners) is hard coded by the portlet programmer, and is not modifiable by portal users at runtime or during page construction.

Since the time of our investigation, Plumtree have been acquired by BEA, which has rebranded the portal as "BEA AquaLogic User Interaction"[89].

5.1.7 Liferay[52]

Like eXo, Liferay adds its IPC support by extending its JSR-168 implementation. It does this by introducing new versions of the JSP tags for generating links in a portlet.

The usual JSR-168 `actionURL` and `renderURL` tags generate URLs which lead to a new Action or Render request targeted to the current portlet, with associated parameters and any other attributes such as changes of window state or portlet mode.

Liferay extends these tags with a new attribute, allowing the portlet developer to specify a portlet name to which the new request will be targeted. This approach allows a portlet to send information to another portlet very easily. However, there are some limitations: messages cannot be broadcast to multiple portlets, and as the originating portlet is no longer targeted, it will have no way of knowing what link has been followed, or even that it has been clicked.

5.1.8 JBoss Portal[51]

JBoss Portal 2.2 provides its own IPC implementation as an extension to JSR-168. Its approach is to allow listeners to intercept events on the server and modify or redirect them.

Using JBoss's IPC, the portlet sending a message does not need to write any messaging-specific code; instead it simply posts the information to itself as a normal Action request. The receiving portlet will have registered a Listener class at deployment time (this is configured in a deployment descriptor file). The Listener can inspect all events, and when it finds an Action event from the named source portlet window, it can intercept and replace it with a similar Action event targeting the listening (also named) portlet window.

Thus in behaviour this would act similarly to Liferay's implementation (although it is more flexible): an Action URL in one portlet can result in that Action being executed on another portlet window.

5.1.9 Summary

	BEA Weblogic	IBM WebSphere	Sybase Enterprise Portal	Oracle Portal	Exo Portal	Plumtree Portal	Liferay	JBoss Portal
Approach								
Event/Listener		✓	✓	✓	✓	✓		
Intercept/Redirect	✓						✓	✓
Configuration								
Hardcoded in portlet source	✓					✓		
Deployment descriptor					✓			✓
Portal page design time	✓	✓	✓	✓				
Run time		✓		✓				
Messaging features								
One to One	✓	✓		✓	✓	✓	✓	✓
One to Many	✓	✓	✓	✓	(✓)	✓		
Message Chain possible		✓	?	✓		✓		
Implementation tier								
Server-side	✓	✓	✓	✓	✓		✓	✓
Client-side (JavaScript)						✓		

Table 5.1: Comparison of Portal-specific implementations of Inter-portlet Communication (2005)

5.2 Implementation approaches for JSR-168-compliant IPC

In JSR-168-compliant implementations of IPC, the portlet developer must completely define and implement the details of the communication system. The developer is necessarily restricted to using only the portlet features and services defined by JSR-168.

Most of the custom approaches to implementing IPC taken by portals are not suitable for use in a pure JSR-168 portlet, as they require portal-level modifications (usually to the portlet request processing procedure) that are beyond the reach of portlet code.

5.2.1 Communication Model

The first consideration is the choice of communication model used.

5.2.1.1 Event/Listener

The Publish-Subscribe channel[137] is a common messaging model where a single message may be sent to any number of registered receivers. The sender may communicate directly and synchronously with the receivers, or it may use a mediating messaging system, which decouples the messaging agents and allows for asynchronous operation (e.g. storing messages if a receiving agent goes temporarily offline).

The Event/Listener model (using the Observer Pattern[134]) is an implementation of this form of messaging which is commonly used in J2EE applications, and is the most popular choice for portlet messaging among the proprietary IPC implementations examined. It is well-defined and reliable, allowing programmers to implement and register Listeners to link portlets with message events. While some portals only permit 1:1 messaging, others allow 1:Many. A proprietary portal has the advantage of controlling the messaging system, and can as a result improve efficiency by re-rendering only those portlets which have received messages, and also may support message chaining.

Unfortunately, this model requires portal-level support to be properly implemented. In particular, we believe this model is unsuitable for a JSR-168 solution for the following reasons:

Firstly, a listener registry and event system must be in place in the portal to call listeners when a messaging event occurs. This registry must be well integrated, so that any portlet windows can receive and act upon messages - even if the user has not yet visited and instantiated the portlet windows in the current session. In the portals examined this has been implemented in a variety of ways, from various modifications to the request processing pipeline (e.g. IBM WebSphere) to a mostly client-side JavaScript/AJAX[27] implementation (Plumtree Portal). The portal must also provide listener configuration tools, which is again reliant on portal-level support to enable configuration of not-yet-visited portlet windows at runtime.

Secondly, configuration of listeners must be complete before the user can be allowed to trigger any message events. Once an event has occurred, it is too late to configure a new portlet to listen to it. This is a design (not implementation) restriction of the model, but it is exacerbated in JSR-168 due to the lack of information about available portlet windows.

Thus, the main problems with the Event/Listener model are due to the way the messaging system must 'push' messages to the receivers. In the context of a JSR-168 portlet, which has limited or no information about other portlets in its environment (and no access to a messaging system with more privileged knowledge), this cannot always be done reliably, and there is also no way of triggering the receiving portlets to act upon the message within the standard portal request sequence.

5.2.1.2 Interceptor

Some of the custom portal implementations examined (e.g. JBoss Portal) use a simpler intercepting filter[131] approach, where portlet actions – normally targeted to the source portlet – can be intercepted and overridden by a custom handler from another portlet.

This filter generally must be configured as part of the portlet application setup, and also requires core support from the portal engine; it is therefore not suited to a JSR-168 compliant implementation.

5.2.1.3 Agent Messaging

The messaging models used by mobile Agents[132] must account for certain limitations of their working environment that have similarities to those encountered by portlets.

In particular, agents may move to different 'places', each of which may contain elements of a messaging system. At any point in time, an agent may wish to send a message to another known agent, without knowing where the target agent is located or whether it is even contactable at that point in time. Alternatively, there may be no known target, and a message may be broadcast for any agent to pick up.

A portlet with no access to a portal-provided messaging system will also be in the situation where it has no knowledge of its audience, and no ability to directly address messages to another portlet. The limited portlet lifecycle (portlet windows are only 'active' when in either their Action or Render phase, which are triggered only at the request of a user visiting the page that contains them) is similar to an agent which is only occasionally connected to a network and able to pick up messages.

Given that we are interested only in agent messaging models where the sending agent does not know the details of the receiving agent, the Blackboard model[132] appears to be the most relevant. This is a simple model where the sending agent deposits a public message at each place it visits, and all receiving agents check for messages at each place. The portal situation corresponds to the simple case where there is only one place where messages can be stored (and agents/portlets can be present, although they could also be 'offline').

This simple model has the advantage over Event/Listener of supporting communication when the receiving agents are not known. It requires a central location, accessible to all agents, where messages may be stored. Its disadvantage is the lack of structure: all messages are effectively broadcast to all agents.

5.2.1.4 Message Board

Each model examined so far has both advantages and disadvantages in the context of portlet messaging. Our final model takes many of these aspects and combines them in a way allowing optimum flexibility while constrained by the limitations of remaining JSR-168 compliant and portal-independent.

Particularly relevant messaging concepts for portlets include:

- Decoupled messaging (Mediator pattern[134]) using a Message Broker[137], so that the sending portlet does not need to know the message's destination.
- Global messaging (broadcast), as the portlets cannot know of all the possible receivers.
- Asynchronous and unreliable messaging, as there is no guarantee that a receiving portlet will ever become active to read its messages.
- Topic-based messaging[137] (e.g. JMS[117]), which is an asynchronous and decoupled approach to the Publish-Subscribe pattern. This adds structure, allowing differentiation between messages but without requiring agents to have knowledge of each other.

We found a combination of the Blackboard model (asynchronous, anonymous, unreliable) and Topics (structured, mediated) to be the most appropriate for the environment of a JSR-168 portlet. In this "message board" model, messages may be published to a public space, and receiving portlets may explicitly read ('pull') any messages they are interested in, in their own time (Figure 5.2). This model does not require listening portlets to be registered before a message is sent; indeed, a new portlet window can easily be added to a page and configured to read a message that was sent previously in the session.

To simplify and abstract the sending of messages by portlets, we introduce a messaging mediator within the portlet application, which manages the distribution of messages to Topics and the delivery of messages when requested by receiving portlets. This will be discussed in more detail later in this chapter.

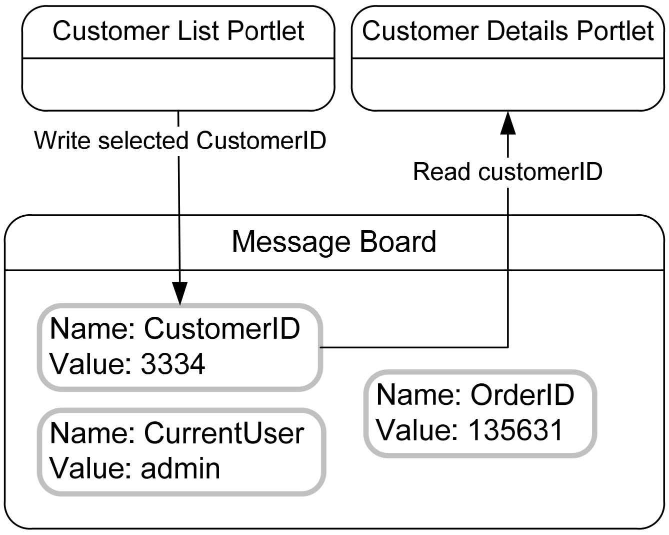


Figure 5.2: Message board model for IPC

5.2.2 Message Timing

Within JSR-168, we are restricted to sending and receiving messages within the standard portlet request handling procedure (Figure 2.6). Thus, messages may only be sent or received in the Action phase (while a request to the portlet is being processed, e.g. from a form submission) or in the Render phase (while the portlet display is being generated).

5.2.2.1 Sending Messages

Those portals whose portlet interfaces are close to JSR-168 typically require that messages are sent in the equivalent of the Action phase. This is because the Action phase is guaranteed to complete before the page starts rendering, so this policy should ensure that all portlets on the page see the same messages while they render. If the portlets were to send messages in the View phase, portlets rendering later on the page might show views inconsistent with other portlets, particularly if the portal uses threading to render portlets in parallel. This approach fortunately also corresponds well with the usual client behaviour that should trigger a message: for example, selecting an item in an Index portlet to trigger a message from that portlet containing the item's ID.

However, sometimes a message may be triggered by a non-client event: namely, in message chaining, where as a result of receiving a message, a portlet may need to send another message. In this case, the portlet will not have the opportunity to go into the Action phase to send the message, and if it is to send at all, must do so in the same Render phase. As this may result in the confusion and inconsistency described previously, this is not recommended - nor will it support multiple stages of message chaining within a single page render. This is one of the main disadvantages of using the Message Board model as opposed to a true Event/Listener system.

5.2.2.2 Receiving Messages

Portlet windows may retrieve messages at any time when they are active - that is, during the Action or Render phases.

Portlets can therefore only retrieve and react to messages when a client views the portal page containing them, and (unless it is acceptable to put 'Update!' Action buttons in every portlet) this must be done in the Render phase. This goes against the recommended practice of making the Render code as lightweight as possible. A slightly more intelligent alternative to the 'Update!' button approach, would be to detect the presence of the new message and simply present a notification to the user that the portlet needs to be manually refreshed - but this would still be a very annoying user interface design decision.

Thus, the messaging is asynchronous and also unreliable - a portlet window which is never viewed will never read and react to a message sent to it. This is efficient and as expected if we consider portlets as simple web pages and tools (interface components) that are only of importance when the user wants to view them. However it does highlight the point that portlets should never be treated as functional components of an independent, back-end engine. Rather, it is wisest to treat a collection of interacting portlets as pure GUI components, front-end widgets, which can each be used to initiate back-end processes but do not in themselves make up an engine.

Thus for messages to be received, the Render phase must take place, which is usually triggered when a portlet window is shown on a page. However, to improve performance, some portals cache portlet render fragments (as permitted by JSR-168

PLT.18) until an explicit action is taken in a portlet, thus preventing the view-phase reaction code from triggering. There is currently no way of informing a portlet window that its content needs to be refreshed as a consequence of a message sent from a different portlet. This is another disadvantage resulting from the lack of portal-level support for IPC. The only workaround (other than the previously mentioned 'Update!' buttons in every portlet) is to disable caching for the receiving portlets by setting their 'expiration-cache' to zero in the portlet deployment descriptor: this may result in significantly decreased performance. Thus, with caching turned off, it is even more important to make Render code lightweight, and if this is not possible the portlet developer may need to implement their own caching mechanism (e.g. storing re-used data in the session).

5.2.3 Message Storage

A messaging system must provide a centralised point of access for messages, visible to all portlets. For this, we must make use of shared message spaces which are provided by or compatible with JSR-168.

5.2.3.1 Communication between portlets in the same portlet application

The best and easiest approach is to store messages in the APPLICATION_SCOPE session (Figure 5.3), which is accessible by all portlets in that application. This is the only easily-accessible shared storage place provided by JSR-168, so it is usually recommended that portlets which need to communicate are packaged in the same portlet application.

However, the issue mentioned in Chapter 2 concerning the sharing of sessions for portlets and servlets may cause problems if servlets are expected to take part in portlet messaging interactions. If the portlets and servlets in the same application do not see the same session, as is the case with certain portal setups, they clearly cannot use it for storing and sharing messages, and alternative external message stores (e.g. databases) must be used.

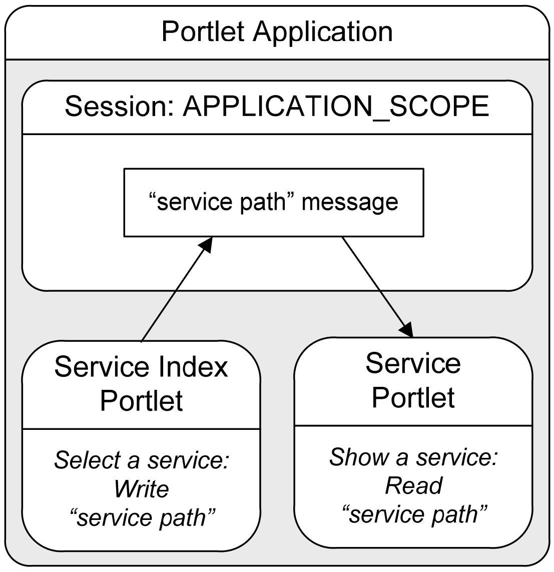

Portlet Application
Session: APPLICATION_SCOPE
"service path" message
Service Index Portlet
Select a service: Write "service path"
Service Portlet
Show a service: Read "service path"

Figure 5.3: Simple inter-portlet communication using a shared session attribute. The Service Index portlet provides a list of services. Selecting a service from this list makes its corresponding interface appear in the Service portlet. This is implemented by storing the service path in a session variable, in the 'Application' scope which is visible to both portlets.

Sometimes the `PortletContext` is mentioned as a possible storage place for shared data. There is one `PortletContext` per portlet application, on which attributes can be set, but - like `ServletContext` attributes - these are globally visible to all concurrent user sessions and are therefore not appropriate for our intended IPC use cases.

5.2.3.2 Cross-context communication, Servlet/Portlet messaging

Web (and thus portlet) applications are kept strictly separate by the Servlet specification, and do not share sessions. Without any JSR-168-provided shared space, it is therefore necessary to use a common external message store that is visible to all portlets, for example an EJB or database (Figure 5.4). This approach is more complex as it requires the additional setup and maintenance of an external message store, so if at all possible it is best avoided, by packaging communicating portlets within the same application.

If communication between portlet applications is required, each portlet must be able to access an identifier representing the user's whole portal session, not just the session ID for that portlet's application (which may differ across applications). A suitable identifier for the portal session may be retrieved through `PortletRequest.getRequestedSessionId()`, but this will only be available once the client browser has agreed to take part in the session, and will therefore not be accessible on the first portal page a client visits. Section 7.3.2.5 mentions some other approaches, but none are completely satisfactory.

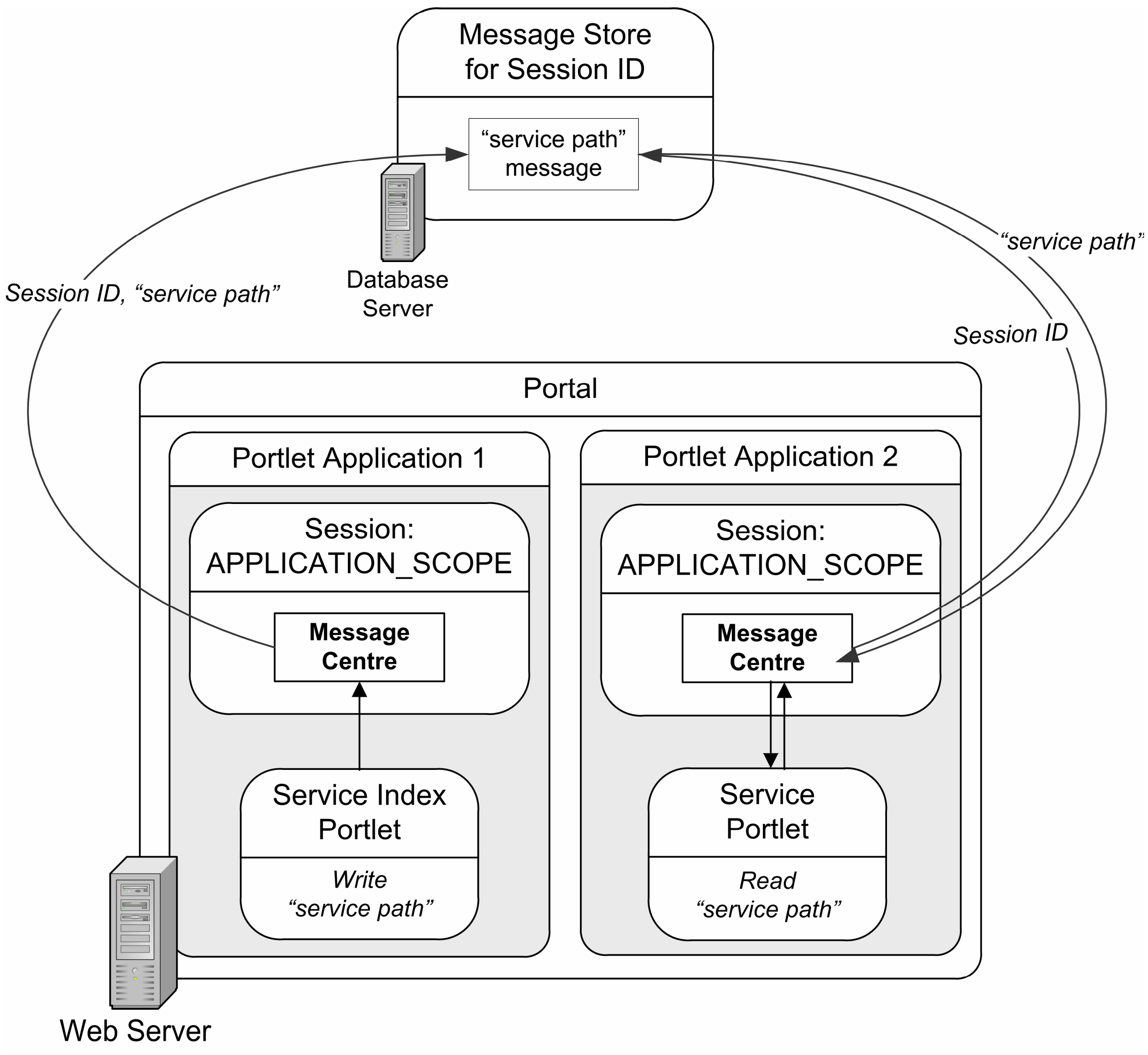


Figure 5.4: Cross-context communication relies on an external message store and a known portal user session ID to connect the different portlet applications.

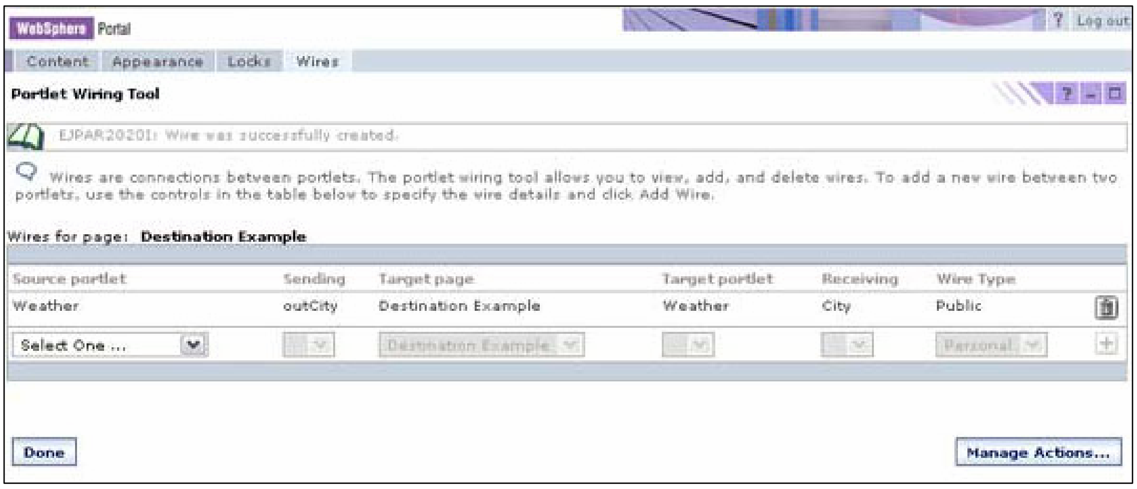
5.2.4 Configuration

When developing a portlet which sends a message, some configuration must be done so that the message can be addressed to its receiving portlet(s). In many of the portal-specific solutions, the ID of the target portlet window is the configuration parameter required, and custom tools are often provided to ease the configuration process. In our "Message Board" model, the public message name is the parameter to be configured, and must match in both sending and receiving portlets.

The simplest level of configuration is to hard-code the message names (or target portlet window ID) in the portlets' code. For message names, this may result in naming clashes, and somewhat restrict the arrangement of portlets in portal pages. For target portlet windows, this will tightly-couple the portlet implementation to the particular portal page layout being used.

A slightly more flexible approach would be to define the message names/portlet windows used in a configuration file (e.g. as initial parameters for the portlets in the portlet deployment descriptor `portlet.xml`), which would allow the deployer of the portlet application (the portal administrator) to modify the message names/portlet windows used before the portal is started.

The greatest flexibility is provided by the ability to change the message names or target portlet windows while the portal server is running, typically through a web interface on the Portal itself. Both IBM WebSphere and Oracle Portal allow such configuration as part of the page layout editing interface (Figure 5.5, Figure 5.6). This allows new portlets to be added and communication channels to be dynamically configured at any time during normal use of the portal.



WebSphere Portal

Content | Appearance | Locks | Wires

Portlet Wiring Tool

EJPAR2020I: Wire was successfully created.

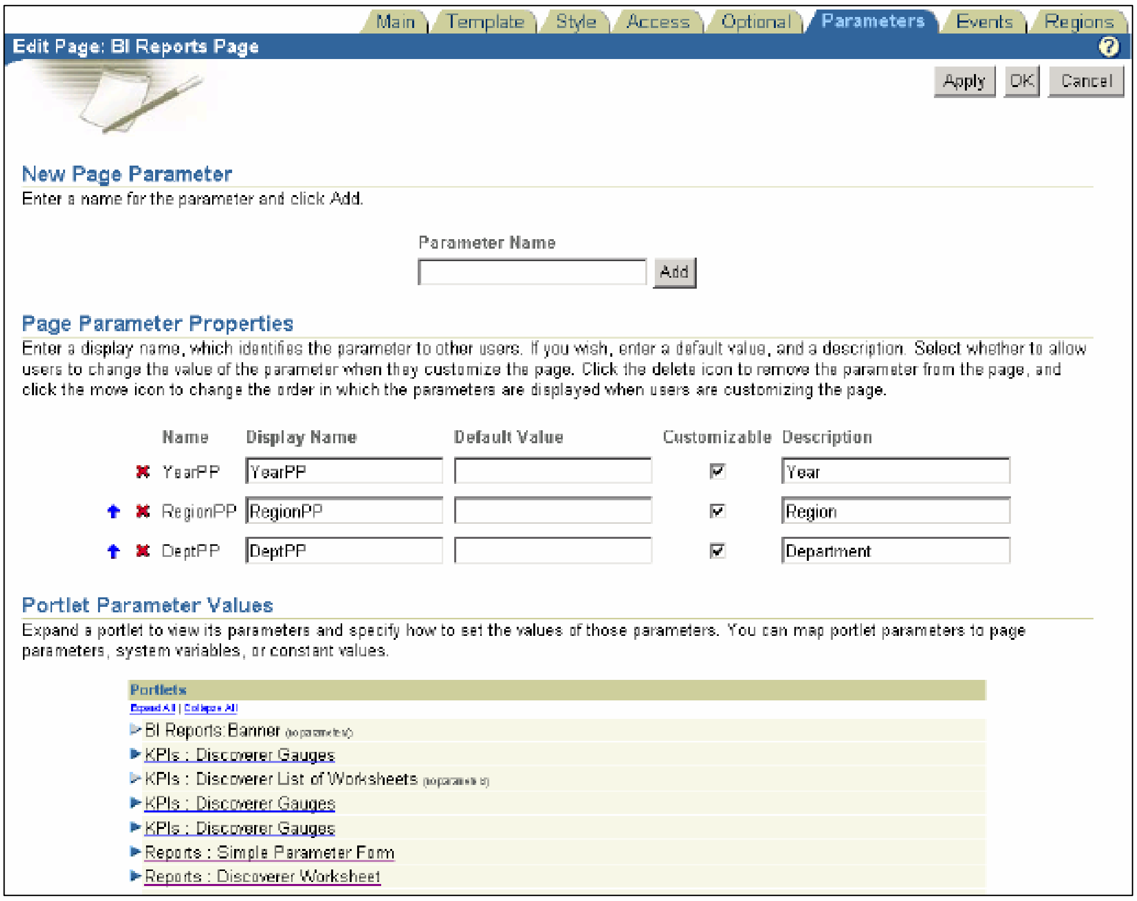
Wires are connections between portlets. The portlet wiring tool allows you to view, add, and delete wires. To add a new wire between two portlets, use the controls in the table below to specify the wire details and click Add Wire.

Wires for page: **Destination Example**

Source portlet	Sending	Target page	Target portlet	Receiving	Wire Type
Weather	outCity	Destination Example	Weather	City	Public
Select One ...		Destination Example			Personal

Done | Manage Actions...

Figure 5.5: IBM WebSphere's portlet "wiring" tool, a custom portlet that allows the user to configure the "wires" between portlets on a page at runtime.



Main | Template | Style | Access | Optional | Parameters | Events | Regions

Edit Page: BI Reports Page

Apply | OK | Cancel

New Page Parameter

Enter a name for the parameter and click Add.

Parameter Name: [ ] Add

Page Parameter Properties

Enter a display name, which identifies the parameter to other users. If you wish, enter a default value, and a description. Select whether to allow users to change the value of the parameter when they customize the page. Click the delete icon to remove the parameter from the page, and click the move icon to change the order in which the parameters are displayed when users are customizing the page.

Name	Display Name	Default Value	Customizable	Description
YearPP	YearPP		✓	Year
RegionPP	RegionPP		✓	Region
DeptPP	DeptPP		✓	Department

Portlet Parameter Values

Expand a portlet to view its parameters and specify how to set the values of those parameters. You can map portlet parameters to page parameters, system variables, or constant values.

Portlets

Expand All | Collapse All

- BI Reports:Banner (no parameters)
- KPIs : Discoverer Gauges
- KPIs : Discoverer List of Worksheets (no parameters)
- KPIs : Discoverer Gauges
- KPIs : Discoverer Gauges
- Reports : Simple Parameter Form
- Reports : Discoverer Worksheet

Figure 5.6: Oracle Portal's page editing mode, allowing message parameter mappings to be configured.

5.2.5 Other features

There are several other notable features displayed by the portal-specific IPC implementations examined:

Oracle Portal includes an intermediate area for messages: page or page-group parameters, which can be used as variables to set input messages on many portlets at once. An alternative implementation with the same effect is to permit a single portlet to broadcast a message to many others, and the equivalent in our "Message Board" model would be to have many portlets reading from a single named message, which would correspond closely to the page parameter.

Some portals include redirection to a different page as part of the reaction to a portlet event. Typically this redirect is to the page on which the receiving portlet lies. We chose not include redirection as a feature in our IPC library implementation, although portlets can still redirect to a different page in their Action phase. This is discussed further in Section 5.4.

The type of data permitted in portlet messages varies from simple Strings to any Java Object.

5.3 Design of a JSR-168-compliant IPC library

5.3.1 Requirements

Our main interest is communication within the same portlet application, as all Discovery Net portlets are packaged together, and thus we will assume in the following discussion that messages are stored in the APPLICATION_SCOPE portlet session. However, in our actual portlet messaging implementation, access to the message store is abstracted out and the implementation can be replaced if cross-context messaging is required.

Hard-coding a pair of communicating portlets to check a known session attribute for a message is the easiest implementation approach (Figure 5.3), and is appropriate when the portlets are being developed for a site whose design is static (although maintenance may become unwieldy if many communicating portlets are required, as the message names must be kept consistent).

However in our portal scenario, we need to develop portlets as reusable components that can be added in any quantity to any page, allowing communication channels between portlet windows to be configured at runtime by the user. For example one page may have a Service Index portlet window whose selections are displayed in a Service portlet window on the same page, and a second page may have two more Service portlet windows which the user has already configured to show two favourite services. Selections on the Service Index should only affect its corresponding Service portlet window, not the two on the other page. In addition, the user may be able to construct their own new pages, and add further portlet windows, which again should not necessarily interfere with existing communications.

Thus, in a changeable environment where many different types of portlets are sending out many different messages, it should be up to the user - not the portlet programmer - which messages the portlet window actually listens to or sends at runtime. Also, ideally, the internal message identifiers used in each portlet's code should not have to match those used in other portlets, nor should the portlet programmer have to worry about naming conflicts with other portlets. For example, a Service Index portlet might publish a selected service as a message named `service_path`, while a Service portlet which works with it would look for the same `service_path` message to see which service interface to display. However, the developer of a Userspace Item Viewer portlet, which was capable of displaying the workflow behind a Service, might look for the message `item_path` as input, to see which userspace item to display. Semantically, the `service_path` and `item_path` are compatible: both are simply paths to items in the userspace. However, as written, it would not be possible for the Userspace Item Viewer to show a service selected from the Service Index, as the hard-coded message names they are using do not match.

As a more advanced scenario, it is possible that a portlet may need to dynamically add or remove message inputs/outputs at runtime, for example a Service portlet which can switch between showing different services which have different numbers of inputs and outputs.

We also need to support the situation where a user adds a new portlet window to a page that uses as input a message which has already been sent (Figure 5.7). As we have discussed, this is not possible with the Event/Listener model, but is supported by the Message Board model.

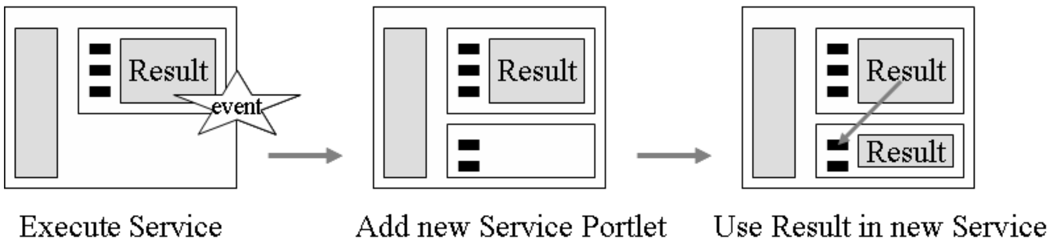


Figure 5.7: Scenario where it is useful for a newly-added portlet to be able to read an existing message. When the first service finishes, the user views the result and decides that it should be processed further. They add a second service portlet to the page, and configure it for the next service. At this stage, they need to take the existing result from the first service and use it as input to the second.

In summary, we require our IPC library to combine the best portal-specific IPC approaches with further features to support our flexible, dynamic portal scenarios:

- Provide an IPC API that is as convenient to use as portal-specific alternatives: it must be easy to use in existing portlet code.
- Allow portlets in both the same and different webapps to communicate (cross-context communication).
- Allow communication between portlet windows on the same or different portal pages.
- Allow different mappings to be maintained for different portlet windows of the same portlet instance (e.g. multiple Service portlet windows).
- Allow a single message to be sent to multiple receivers (e.g. A→B and A→C, even A→A).
- Allow the portal administrator to hardcode mappings (for fixed portal page layouts).
- Allow portal users to modify mappings at runtime.
- Allow portlets to receive as input a message that has already been sent (i.e. not event-based).
- Allow mappings (portlet inputs/outputs) to be added and removed at runtime.

5.3.2 Design

To achieve these requirements we have developed a generic JSR-168-compliant portlet messaging library which provides the necessary features and presents a simple messaging API for portlet developers to use. To remain JSR-168-compliant, no portal-specific code could be used, and some limitations were necessary. Nevertheless this library meets our needs well, allowing our portlets to be written with clean code and to communicate flexibly. It is provided as a standalone library and is freely available from `http://www.doc.ic.ac.uk/~mo197/portlets`.

Throughout this discussion, it should be noted that the overriding requirement of this design is JSR-168-compliance, and as a result it is not as efficient or robust as it might be otherwise. The eventual portlet IPC system defined by the next portlet specification will not have to work within these constraints, and will probably take an entirely different form as a result, perhaps more similar to existing portal-specific implementations.

5.3.2.1 Messaging Model

In our chosen "Message Board" model, portlets may publish named messages which are then available to read at that message box name. Receiving portlets do not by default consume messages but only read them, so that other portlets may also read the same message. The messages persist until they are explicitly overwritten, so that any newly added or loaded portlets can read previously published messages. Thus these messages act as shared state indicators, rather than discrete events. A disadvantage of this approach is that the receiving portlets must themselves maintain a history of messages if they need to detect when a message has changed.

To avoid the problems of hard-coding message names, and to permit users to configure messaging channels between portlet windows at runtime, we have added a translation layer (a Message Broker[137]/ Mediator[134]) between the internal message names used in portlet code and the public names of message boxes visible at runtime to all portlet windows. Thus, in its own code a portlet may refer to a particular output message as "path", but in the portal, the message box to which the message is sent may be configured by the user, e.g. to "Service path for page 1". Similarly, the user can configure the names of message boxes from which the portlet reads each of its input messages. This process of message mapping allows both 1:1, and 1:N messaging (Figure 5.8), as every message channel includes an intermediate message box. It does not however support N:1 mappings, e.g. combining 2 boxes into 1 input, or 2 outputs into 1 box. In practice, this approach to abstraction is similar to Oracle Portal's use of 'page parameters' (5.1.3), although less strictly scoped.

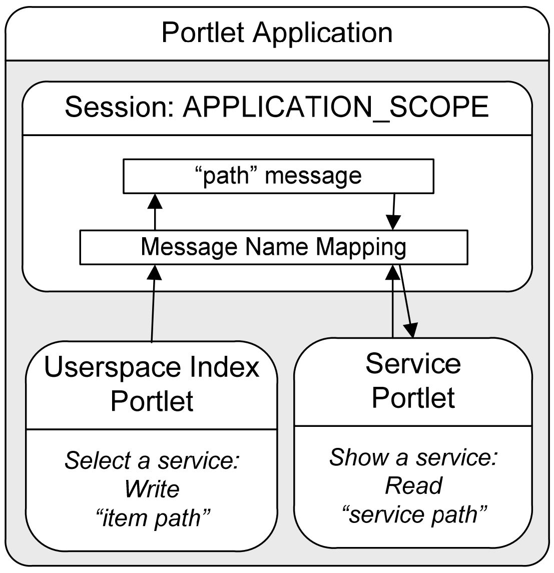

Portlet Application
Session: APPLICATION_SCOPE
"path" message
Message Name Mapping
Userspace Index Portlet
Select a service: Write "item path"
Service Portlet
Show a service: Read "service path"

Figure 5.8: Abstraction of message names. The messaging library mediates access to shared messages, allowing mapping of local names (hard-coded in portlets) to global names (user-configurable at runtime).

5.3.2.2 Configuration

As previously discussed, configuration of IPC may be performed, with increasing levels of flexibility, at:

- Portlet programming time (in code or configuration files)
- Portlet deployment time (in default or initialisation portlet parameters)
- Page development time (with custom tools or through a web interface on the Portal)
- Runtime (through a web interface on the Portal).

The latter, most flexible, option is what we require.

Approach	Advantages	Disadvantages
1. Configure in each portlet	Mappings are kept together with their portlet (intuitive.) Efficient, as mappings are only loaded when a portlet window is first viewed.	Need to edit and save every portlet involved in the communication - potentially cumbersome
2. Configure in a single "wiring tool" portlet	All mappings can be edited and seen at once - fast.	More complicated for user, if there are many mappings visible. All mappings are stored in a single portlet, which would need a way of accessing information about all the portlet windows in the portal.

Table 5.2: Approaches to storing IPC configuration: GUI presentation.

Approach	Advantages	Disadvantages
1. Store in portlet preferences	Easy to implement, JSR-168 mechanism. Flexible: configuration can be set as default preference values by the administrator upon deployment, or by the user at runtime.	Each portlet window must be instantiated/viewed to load its configuration from the preferences, so other portlets will not be aware of its mappings until then.
2. External to portal	All portlets can access the full configuration at any time. Supports cross-context communication.	Extra work and dependencies in setting up and managing an external store.

Table 5.3: Approaches to storing IPC configuration: storage location.

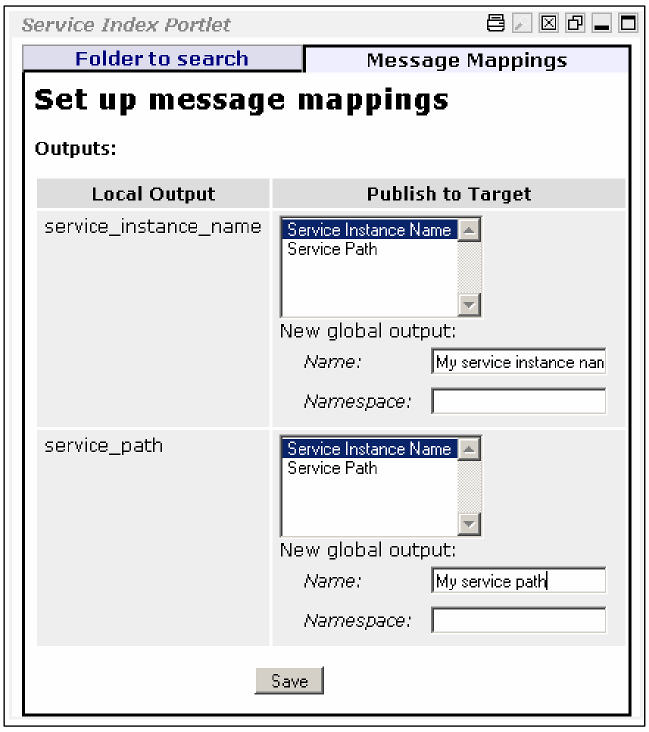
The possible approaches to storage and configuration of the message mappings are summarized in Table 5.2 and Table 5.3. While a single "wiring tool" portlet would be the most elegant approach for configuring message mappings (and is the usual approach taken by custom IPC implementations) it is not possible to implement effectively without portal-level support for retrieving a listing of all portlet windows on the site. Therefore we must spread out the mapping configuration among the portlets to which the mappings belong: allowing configuration of each portlet's input and output mappings in its own Edit mode. For storage of mappings, we decided to use the portlet preferences, as a simple and flexible mechanism which allows values to be set both upon deployment and at runtime by the user. However, we additionally propagate the mappings to a centralised Message Store when the portlet first loads, and in this way can optionally support cross-context communication by providing an externally-persisted Message Store implementation.

To achieve this, the portlets delegate to a shared Message Helper in the application session to publish and read all messages (the Message Helper in turn delegates to a pluggable Message Store). When a portlet first loads, it registers its inputs and outputs through the Message Helper, and from then on the Message Helper refers to and modifies these local-to-global message name mappings as required. The mappings are defined and saved in each portlet's parameters, using the standard parameter persistence mechanism, and may be modified in each portlet's Edit mode. A simple but generic implementation of this edit mode is shown in Figure 5.9: the interface is dynamically generated by inspection of the messages in the Message Store and the current portlet's messaging configuration.

The main disadvantages of this approach are:

1) Both ends of a communication channel must be separately configured: both the sending portlet's output mapping and the receiving portlet's input mapping.
2) Existing mappings (e.g. available output messages) will only be included in the configuration dialog after their parent portlet window has been rendered in the user's current session at least once.

The user interface for IPC configuration could potentially be improved greatly using AJAX[27], but unfortunately this is not possible with JSR-168: AJAX requests, which call servlets, cannot access portlet window services such as portlet preferences, in which the messaging configuration is stored. If however the configuration were to be stored externally (e.g. in a database), an AJAX configuration GUI would be feasible.


Service Index Portlet

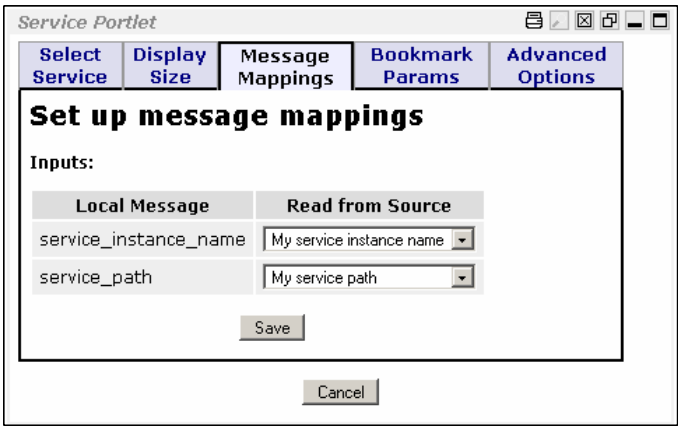
Folder to search | Message Mappings

Set up message mappings

Outputs:

Local Output	Publish to Target
service_instance_name	Service Instance Name Service Path New global output: *Name:* My service instance nan *Namespace:*
service_path	Service Instance Name Service Path New global output: *Name:* My service path *Namespace:*

Save


Service Portlet

Select Service | Display Size | Message Mappings | Bookmark Params | Advanced Options

Set up message mappings

Inputs:

Local Message	Read from Source
service_instance_name	My service instance name
service_path	My service path

Save

Cancel

Figure 5.9: Dynamically-generated mapping configuration forms, accessible in the portlet's Edit mode at runtime. Two portlets are shown, the first configured to publish messages, the second to read them.

5.3.2.3 IPC Library Code

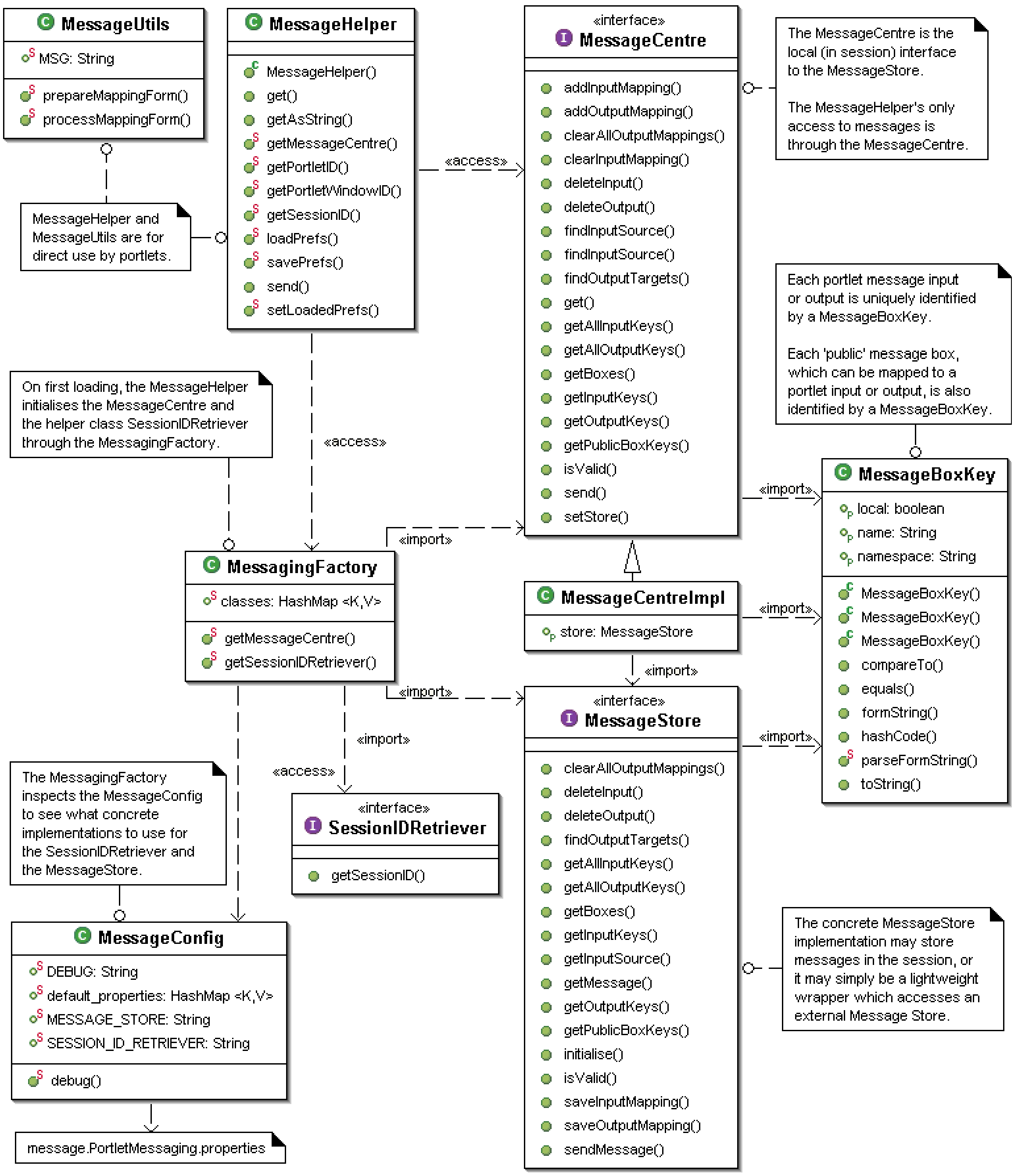


Figure 5.10: UML diagram of the IPC Library

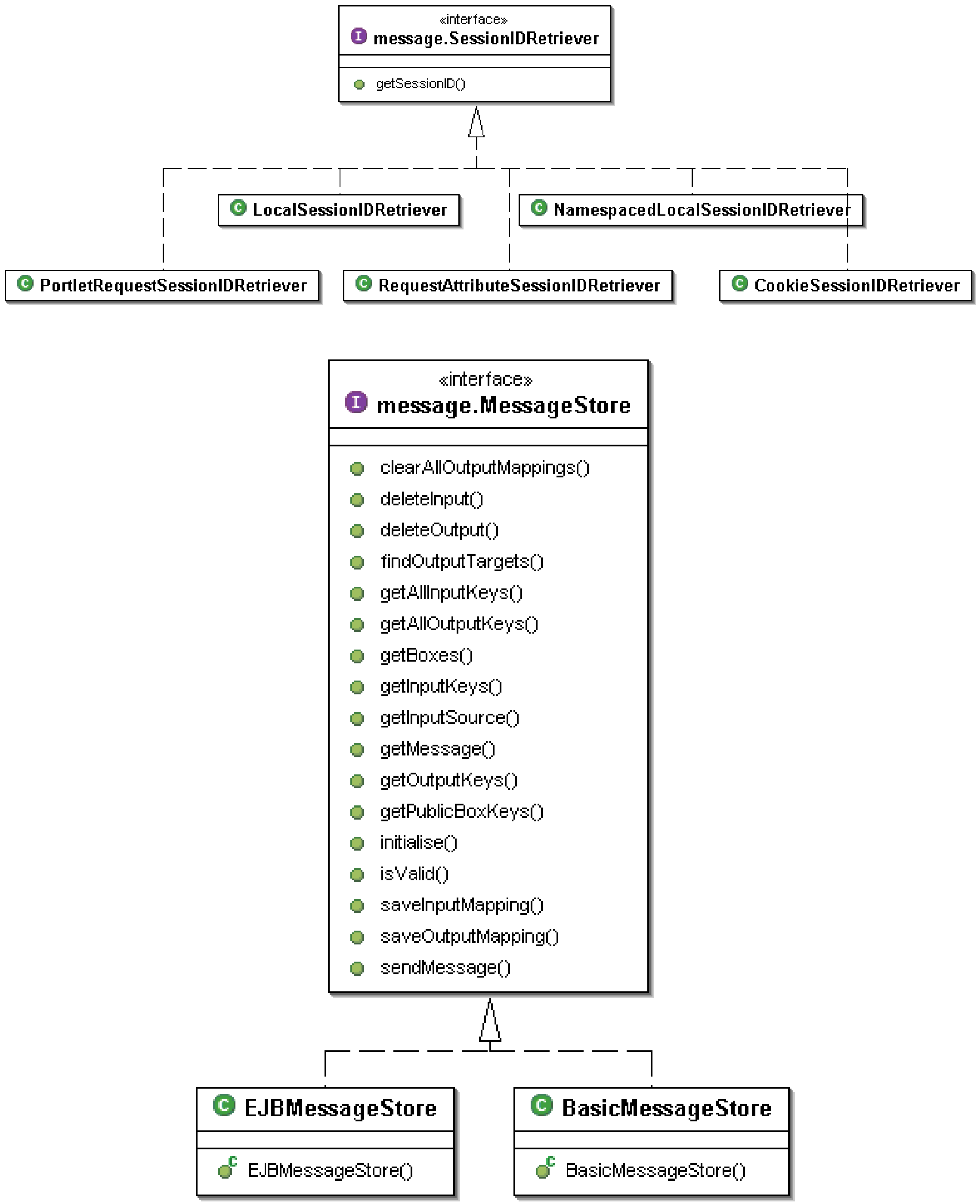


Figure 5.11: Concrete implementations of the MessageStore and SessionIDRetriever interfaces, included in the library.

Figure 5.10 shows the components of the portlet messaging library. The main class intended for direct use by portlets is the `MessageHelper`, which provides a simplified interface appropriate within the context of a portlet window. The `MessageHelper` allows portlets to refer to both incoming and outgoing messages by their own local names, with no explicit reference to the mapping that eventually happens at runtime. The `MessageHelper` must first be initialised with the portlet window's ID, but from then on it will be able to automatically translate each local name to an appropriately namespaced or mapped `MessageBoxKey`, which is the message's global unique name. The `MessageHelper` then passes on (or fetches) the message to the `MessageCentre` (and hence `MessageStore`), which acts as a centralised point of access for both messages and message mappings. Example code that could be used in a portlet's `processAction` and `doView` functions is shown in Figure 5.12.

a)

```
public void processAction (ActionRequest request, ActionResponse actionResponse)
throws PortletException, java.io.IOException  {

        PortletSession portletSession = request.getPortletSession(true);
        String id = getWindowID(request);
        String msg_session_id = MessageHelper.getSessionID(request);
        MessageHelper helper = new MessageHelper(portletSession, id, msg_session_id);

        String myAuthor = request.getParameter("myAuthor");
        helper.send("author", myAuthor);
}
```

b)

```
public void doView(RenderRequest request, RenderResponse response)
throws PortletException, IOException {

        PortletSession portletSession = request.getPortletSession(true);
        String id = getWindowID(request);
        String msg_session_id = MessageHelper.getSessionID(request);

        // load this portlet's inputs and outputs from preferences
        MessageHelper.loadPrefs(request, id, msg_session_id);

        MessageHelper helper = new MessageHelper(portletSession, id, msg_session_id);
        String broadcastAuthor = helper.getAsString("author");

        ...
        writer.write("Selected author: "+broadcastAuthor);
```

Figure 5.12: Code fragment showing the use of the IPC library to a) send a message in one portlet b) receive a message in another portlet.

When implementing portlets that use the IPC library, the portlet developer must include the messaging library JAR in their portlet application, but does not need to make their portlets extend from any special portlet class. Rather, they can treat the messaging library as a separate service to be loaded and called when needed. Of course, as it is not a standard, only portlets which have been designed to use this messaging library will be able to communicate with each other.

When initially loading the `MessageHelper`, it is necessary to read in the message mappings from the portlet preferences. This is best done in `doView`, as this will ensure that the `MessageHelper` is properly initialised as soon as the portlet is first displayed. Once loaded, the mappings are cached in the session, so that subsequent calls to `MessageHelper.loadPrefs` in `doView` do not add more overhead.

As previously mentioned, and shown in Figure 5.12, the portlet developer must provide a portlet ID to uniquely identify the portlet window to the `MessageHelper`. The portal's ID for the portlet window is not directly accessible through JSR-168, but workarounds are available as discussed in Section 7.3.2.4, and the `MessageHelper` provides several functions which can be used to retrieve or generate an appropriate window ID.

The exact implementation of `MessageStore` used is easily configured through a properties file. Hence for example support for cross-context communication with an external `MessageStore` can be configured without recompiling the portlet code. In addition, the implementation of the `SessionIDRetriever` is also configurable, due to the issues mentioned in Section 7.3.2.5, and can be used through the `MessageHelper` as shown in the example code. The configurable parts of the messaging library are illustrated in Figure 5.11 and Figure 5.13.

Thus, the use of the portlet messaging library is split into two parts: first, the code for sending and retrieving messages in the portlet, which uses the local input/output message names, and second, the configuration of message mappings to set up message flows, which can be done either in the default portlet preferences (e.g specified in the portlet deployment descriptor, `portlet.xml`), or at runtime by modifying preferences of individual portlet windows programmatically.

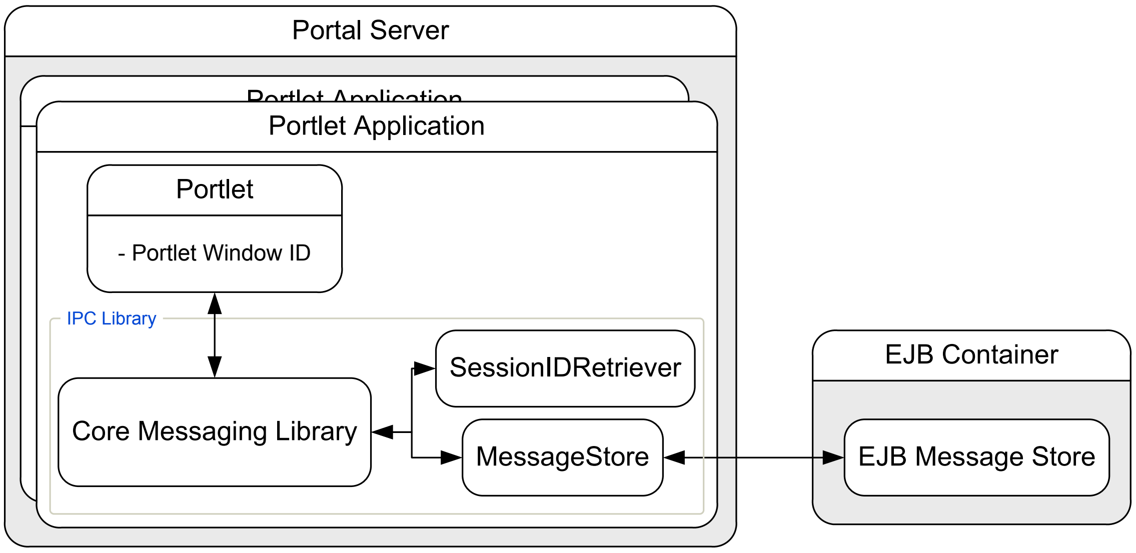


Figure 5.13: Some components of the IPC library can be easily replaced with different versions most appropriate to the situation.

5.4 Page Navigation

As previously mentioned, navigation between pages is often a desired side-effect of sending messages, usually to visit the page which contains the portlet receiving the message. There are a few problems that make the implementation of this feature difficult.

One technical restriction is that a portlet can only change the page (by sending a 'redirect' instruction to the client browser) in the Action phase, as this is the only time that it has access to page headers. Thus, messages sent from a Render phase (as might be done in response to receiving a message) will not be able to trigger page changes. This means that IPC-triggered page changes can only happen with primary messages, sent directly as a result of a user's click - which will probably usually be the desired case anyway.

Secondly, there is the problem of configuring the address of the page to redirect to. This address is entirely situation-dependent, and will be different for each portal implementation, and each website layout. Currently, the user must always look up this address and configure it manually in every instance of the portlet. Even in the case where we know Portlet A always sends the message to Portlet B, and wishes to automatically navigate to the page containing Portlet B, there is no way for the portlet code to find out the target page address using the Portlet API. Ideally, we would want to be able to present (at runtime) a user-friendly configuration interface in the portlet itself, allowing users (or just page designers) to select the target page from a list of all available portal pages. But again, due to the lack of a portal-context-inspection API (as discussed in Chapter 7), this is not possible without using portal-specific code, and instead we are forced to require the user configuring the portlet to find out and type in the full target page URL themselves.

The page-navigation feature is not currently included in our Portlet Messaging Library, due to these technical and design issues. However, they are not critical and may be acceptable if the need is great enough and the limitations are made clear, so this feature may be introduced in the future.

5.5 Limitations of the IPC library

The restrictions of the IPC library are mainly due to the limitations of working within JSR-168, and so cannot be practically resolved at this time. These include:

- Sending messages in the Action phase (not mandated, but highly recommended).
- Receiving messages in the Render phase (thus view caching must be disabled for receiving portlets).
- No explicit support for message chaining.
- Requirement to obtain an ID for each portlet window.

In addition, the current interface for modifying message mappings in the IPC library could be made much simpler to use and understand. A more intuitive presentation would be to allow the user to click & drag to connect outputs to inputs, using JavaScript. Implementing such an interface would have to overcome several concerns: a) an intuitive GUI would need to be developed allowing the user to configure messaging between portlet windows on different pages, b) portlet preferences on more than one portlet window would need to be saved to create a single connection, which would require multiple 'update/save' requests to the portal.

Such a messaging configuration interface would be possible using AJAX, but would first require some additional portal features:

- Centralised support and management of message mappings.
- A method permitting servlet endpoints independent access to portlet windows (e.g. for background-saving of portlet window preferences using AJAX).
- Reliable consolidation of servlet and portlet sessions.
- Methods to retrieve details of other portlet windows present in the portal, including those which have not yet been visited by the user in the current session.

5.6 Conclusion

In this chapter we have introduced the topic of inter-portlet communication and discussed its implementation in different portals.

As one of the main contributions of this thesis, we have described the design and implementation of a JSR-168-compliant IPC library. This library is available for download at `http://www.doc.ic.ac.uk/~mo197/portlets`. The first release was in July 2005, and a version supporting messaging between portlets in different portlet applications was released in September 2005.

Despite the limitations described in Section 5.5, our library still provides a portable, flexible and easy-to-use implementation of IPC for JSR-168 portlets, as a viable alternative to portal-specific solutions. In summary, the IPC Library's features are:

- Simple API
    - Messages are public and have a two-part identifier: namespace + name.
    - Message contents may be any Java object
- Flexible, loosely-coupled communication between portlet windows
    - Configuration in `portlet.xml` default preferences, or through editing portlet preferences at runtime
- Messaging within or across portlet applications

- Pluggable Message Store for different concrete implementations

Portal-specific IPC systems are still generally superior for reliability and integrated messaging, however our library has several advantages which not all portal-specific solutions support:

- Portability through JSR-168 compliance
- Choice of static or run-time message configuration
- Development of completely decoupled messaging portlets
- Late binding to message channels (due to the lack of a strict Event/Listener system)

Future developments for the IPC library will probably include support for page navigation alongside sending messages. While this is not a concept that is strictly tied to messaging, it is a commonly-associated and often-requested parallel feature.

In the process of developing this library, we encountered a number of general limitations with the JSR-168 specification which we have detailed in Chapter 7 and provided as input to the working committees for the next versions of the Portlet and WSRP specifications.

While the focus of this work has been on JSR-168, the WSRP specification is closely related, and so the discussion of messaging models is also relevant to the programming language-independent portlet environment described by WSRP.

In the next Chapter we will describe the development of a real-world portal for Translational Medicine, in which both inter-portlet communication and the Discovery Net Service portlets are fundamental components.

Chapter 6. Translational Medicine Portal: A Case Study of a Complex Portal

In this chapter we illustrate the concepts which have been explored in this thesis so far, with the development of a real-world analytical portal composed out of communicating JSR-168 portlets. This puts into practice the work done in developing the Discovery Net portlets (Chapter 3) and the IPC library (Chapter 5).

The Windber Translational Medicine Portal project is the only one of the applications examined so far that has been developed for production use. The project started in September 2005 and is ongoing; its first production release was in February 2006.

6.1 Aims of the Portal

The intention of this project was to create a web portal which provides both researchers and clinical physicians with access to the archive of patient data that has been built up by the Windber Research Institute[86] over many years.

The first aim was to allow users to interactively explore the available data (initially concentrating on data from a breast cancer study) using summary views, drilling-down through clinical records and questionnaire answers, and selecting groups of patients for follow-up analysis or studies (for example: all patients on a particular medication, who have shown particular symptoms).

Secondly, being able to navigate from the broad overview of aggregated data down to view the detailed clinical data for specific patients in the selected categories was to be a critical feature for this portal. Thus after selecting a set of patients, the researchers needed to be able to both retrieve the details of the patients within that set, and use the set as input to analysis workflows for further research.

6.2 Design Challenges

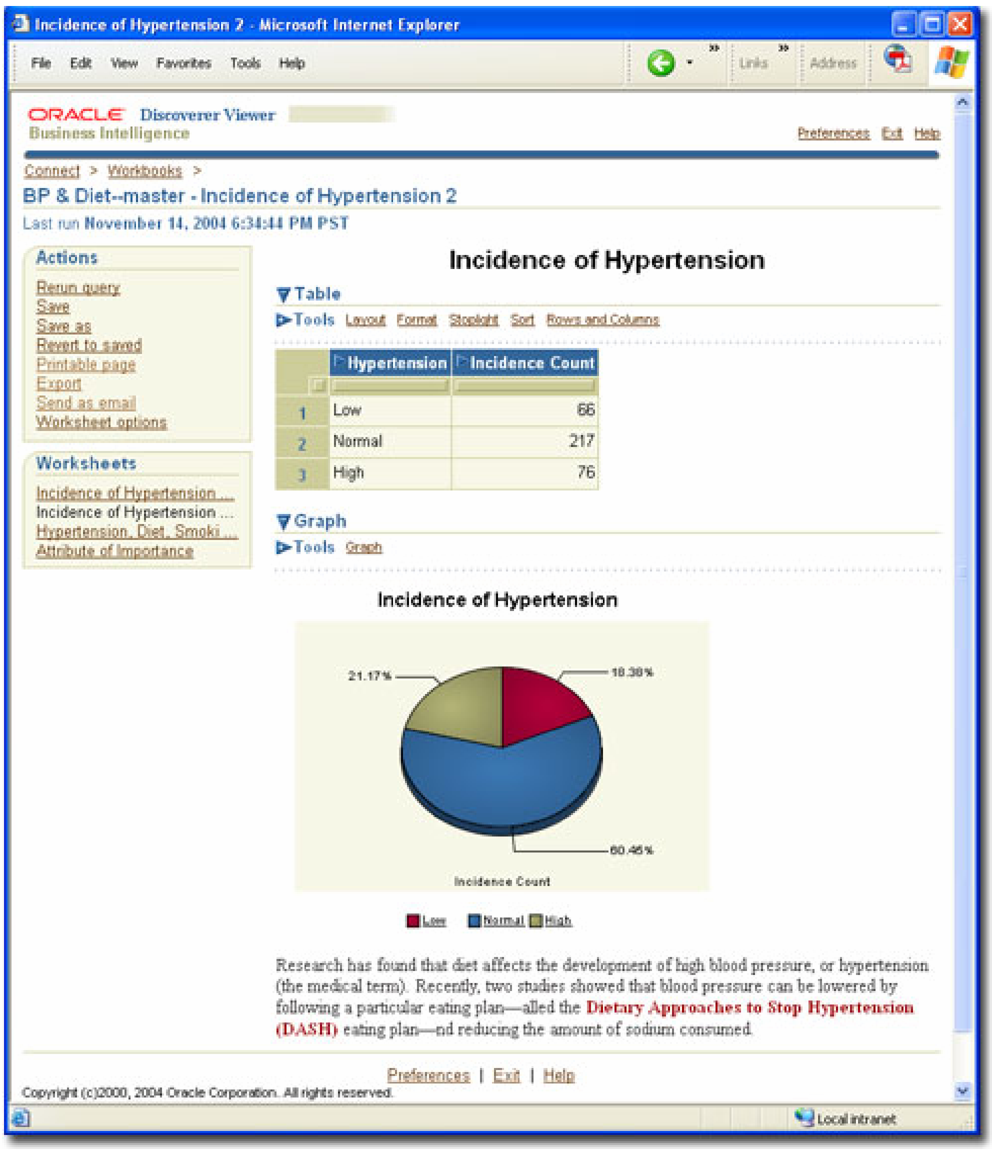
The Discovery Net server and portlets were clearly suitable for the analysis part of the project, allowing researchers to produce analysis workflows and deploy them to the web. However, the ability to browse the patient data was an entirely new application, which we will therefore describe in some detail.

The ability to browse aggregate statistics summarising the contents of a data warehouse is provided by Business Intelligence products using OLAP (On Line Analytical Processing), e.g. Oracle Business Intelligence Discoverer[65]. Typically, such products allow users to view pre-configured or customisable tables of statistics, drill-down to view statistics for subsets of data (e.g. sales figures for a particular month), and generate graphs; Figure 6.1 and Figure 6.2 show this in Oracle Discoverer.

However, most OLAP systems were unable to provide the second requirement: the connection from the aggregate results back to the individual data points which originally made up those results. For efficiency, these systems rely on pre-calculated statistics (stored as 'cubes'), and maintain no connection back to the source data.

We therefore took the approach of developing our own OLAP-inspired system. The patient data under consideration was reasonably small and unlikely to increase rapidly in size, compared to the massive data warehouses which make pre-calculated OLAP cubes necessary. This allowed us to perform all calculations in real time, directly using the source data, which simplified development and ensured that the most up-to-date information would always be used. The resulting system was able to retrieve both aggregated information and the corresponding underlying records.

The database setup, data cleaning and import was an important part of the process, managed by database experts at InforSense and Windber, who also created an interface class (`DatabaseProvider`) for all access to the database. My responsibility in this project was all parts related to the web portal: the binning web interface, the OLAP-like browser, patient set management and patient viewing. Each of these functionalities was implemented as a JSR-168 portlet, for later inclusion alongside the Discovery Net portlets in a portal (we used the Jetspeed 1.6 portal embedded in the Discovery Net server installation). Figure 6.3 shows an overview of the whole project.



BP & Diet--master - Incidence of Hypertension 2

Last run November 14, 2004 6:34:44 PM PST

Actions
Rerun query
Save
Save as
Revert to saved
Printable page
Export
Send as email
Worksheet options
Worksheets
Incidence of Hypertension ...
Incidence of Hypertension ...
Hypertension, Diet, Smoki ...
Attribute of Importance
Table
Tools Layout Format Stoplight Sort Rows and Columns

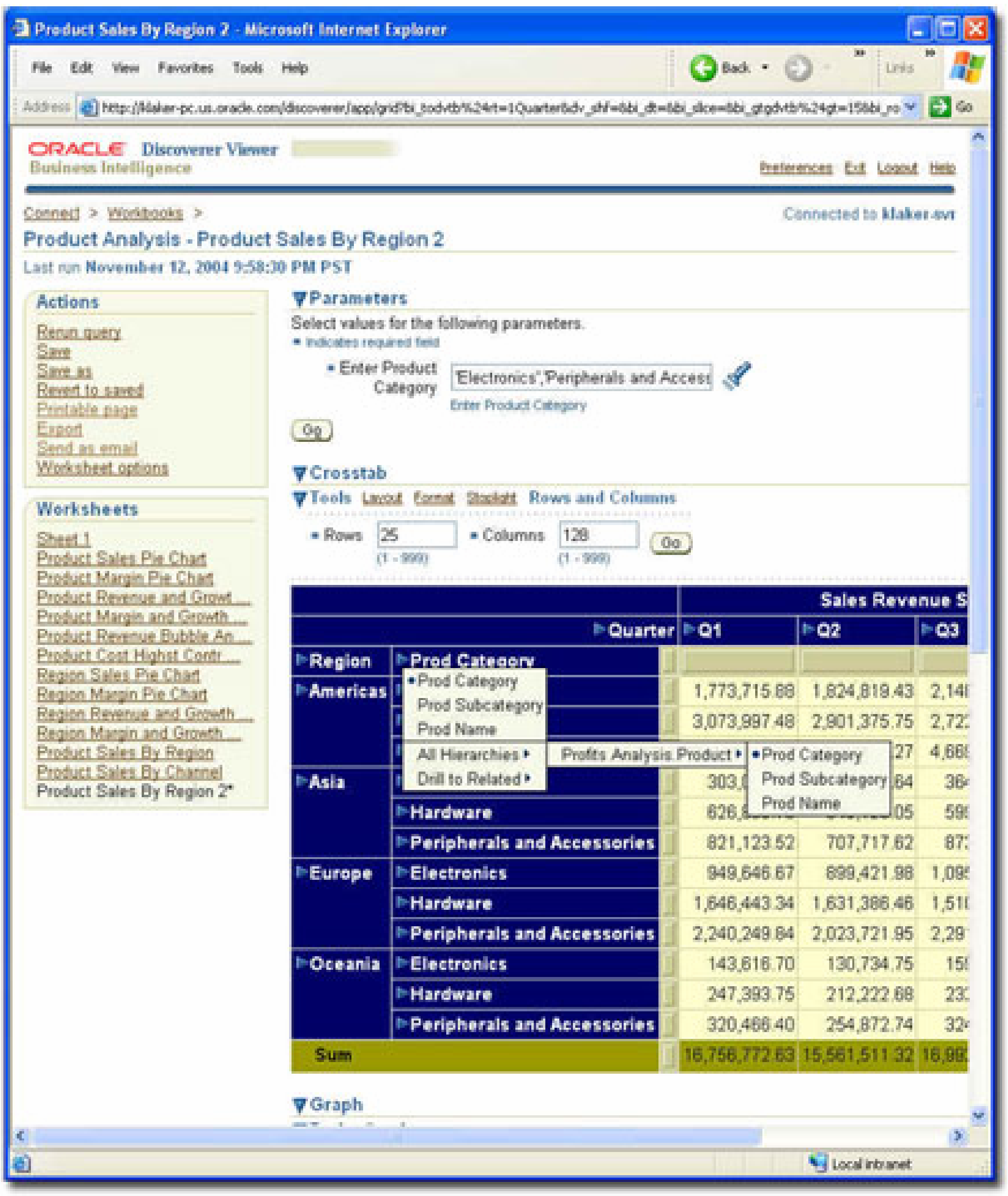
Incidence of Hypertension

	Hypertension	Incidence Count
1	Low	66
2	Normal	217
3	High	76

Research has found that diet affects the development of high blood pressure, or hypertension (the medical term). Recently, two studies showed that blood pressure can be lowered by following a particular eating plan—alled the **Dietary Approaches to Stop Hypertension (DASH)** eating plan—nd reducing the amount of sodium consumed.

Copyright (c)2000, 2004 Oracle Corporation. All rights reserved.

Figure 6.1: Oracle Discoverer allows users to view statistical summaries of large amounts of data, through a web interface. Here, a simple table is shown, with one variable, "Hypertension", which has three possible values, each represented by a row. The "Incidence Count" column shows the result for each row. A pie chart shows the same data. (Image from Oracle documentation[26])



Product Sales By Region 2 - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://klaker-pc.us.oracle.com/discoverer/app/grid?bi_todvtb%24rt=1Quarter&dv_shf=&bi_dt=&bi_dice=&bi_gtgdvtb%24gt=15&bi_ro

ORACLE Discoverer Viewer
Business Intelligence

Preferences Exit Logout Help

Connect > Workbooks >

Connected to klaker-svr

Product Analysis - Product Sales By Region 2

Last run November 12, 2004 9:58:30 PM PST

Actions

- Rerun query
- Save
- Save as
- Revert to saved
- Printable page
- Export
- Send as email
- Worksheet options

Worksheets

- Sheet 1
- Product Sales Pie Chart
- Product Margin Pie Chart
- Product Revenue and Growt...
- Product Margin and Growth...
- Product Revenue Bubble An...
- Product Cost Highst Contr...
- Region Sales Pie Chart
- Region Margin Pie Chart
- Region Revenue and Growth...
- Region Margin and Growth...
- Product Sales By Region
- Product Sales By Channel
- Product Sales By Region 2*

Parameters

Select values for the following parameters.
* Indicates required field

* Enter Product Category: 'Electronics','Peripherals and Access
Enter Product Category

Go

Crosstab

Tools Layout Format Stoplight Rows and Columns

* Rows: 25 (1 - 999) * Columns: 128 (1 - 999) Go

			Sales Revenue S		
	Quarter		Q1	Q2	Q3
Region	Prod Category				
Americas			1,773,715.88	1,824,819.43	2,14
			3,073,997.48	2,901,375.75	2,72
				27	4,66
Asia			303,	64	36
	Hardware		626,	05	59
	Peripherals and Accessories		821,123.52	707,717.62	87
Europe	Electronics		949,646.67	899,421.98	1,09
	Hardware		1,646,443.34	1,631,386.46	1,51
	Peripherals and Accessories		2,240,249.84	2,023,721.95	2,29
Oceania	Electronics		143,616.70	130,734.75	15
	Hardware		247,393.75	212,222.68	23
	Peripherals and Accessories		320,466.40	254,872.74	32
Sum			16,756,772.63	15,561,511.32	16,98

Graph

Local intranet

Figure 6.2: Oracle Discoverer showing a more complex table which contains two categories as rows ("Region" and "Prod Category"), and one category ("Quarter") as four columns (Q1-Q4). The two categories shown as rows are hierarchical; the second category is nested beneath the first. Users can add additional levels of categories to further subdivide each row. (Image from Oracle documentation[26])

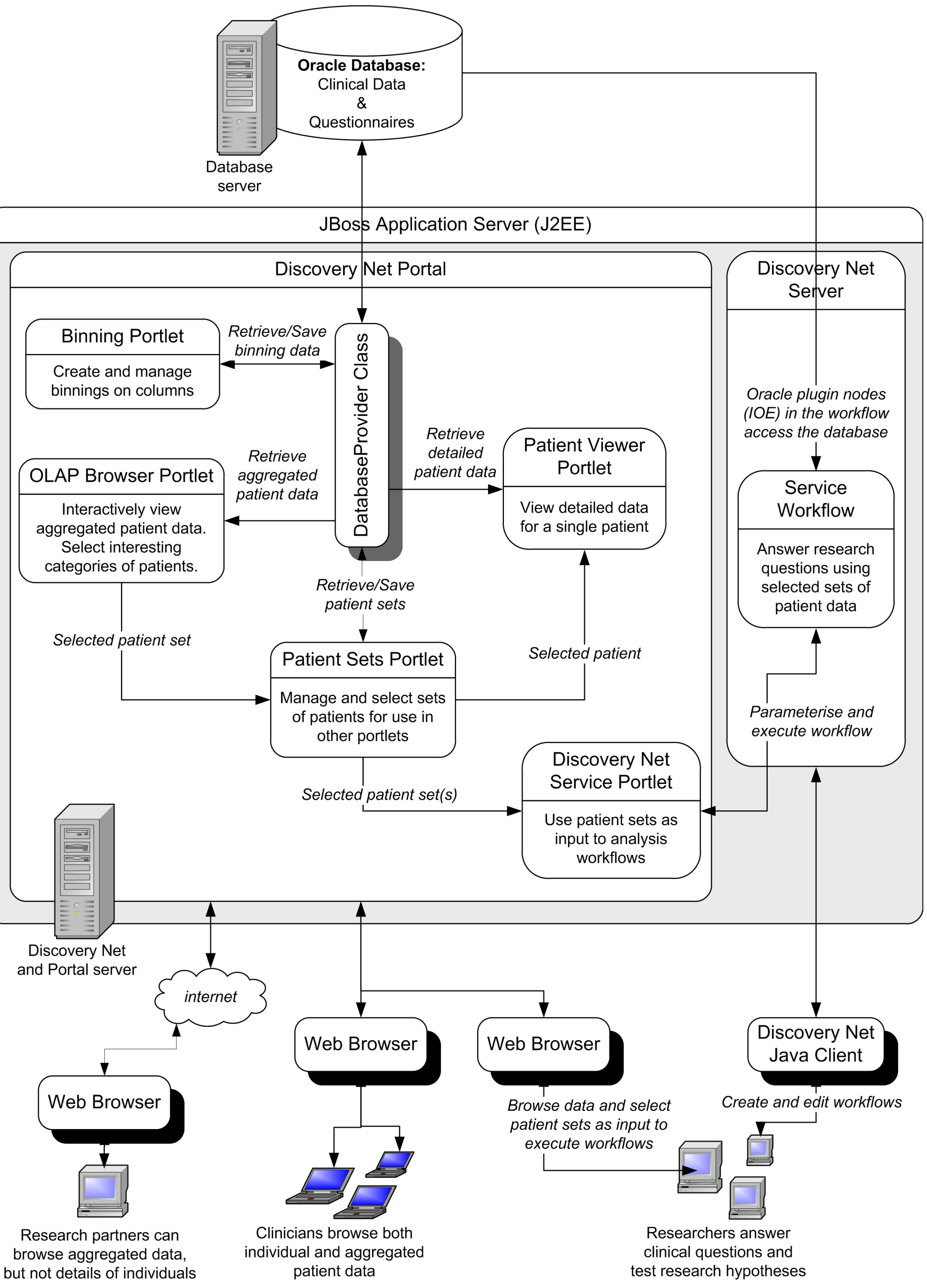

Oracle Database:
Clinical Data
&
Questionnaires
Database server
JBoss Application Server (J2EE)
Discovery Net Portal
Discovery Net Server
Binning Portlet
Create and manage binnings on columns
Retrieve/Save binning data
DatabaseProvider Class
Retrieve detailed patient data
Patient Viewer Portlet
View detailed data for a single patient
Oracle plugin nodes (IOE) in the workflow access the database
OLAP Browser Portlet
Interactively view aggregated patient data. Select interesting categories of patients.
Retrieve aggregated patient data
Retrieve/Save patient sets
Service Workflow
Answer research questions using selected sets of patient data
Selected patient set
Patient Sets Portlet
Manage and select sets of patients for use in other portlets
Selected patient
Parameterise and execute workflow
Discovery Net Service Portlet
Use patient sets as input to analysis workflows
Selected patient set(s)
Discovery Net and Portal server
internet
Web Browser
Web Browser
Web Browser
Discovery Net Java Client
Browse data and select patient sets as input to execute workflows
Create and edit workflows
Research partners can browse aggregated data, but not details of individuals
Clinicians browse both individual and aggregated patient data
Researchers answer clinical questions and test research hypotheses

Figure 6.3: Overview of the Windber Translational Medicine portal.

6.3 AJAX (Asynchronous JavaScript And XML)

The web tools developed for this project make extensive use of the AJAX[27] style of web development, which allows pages to be unusually responsive and interactive. This method uses JavaScript on the client to communicate directly with the server while the user is reading the page, retrieving information in the background in response to user actions. JavaScript is then used to update the displayed page's contents, modifying its Document Object Model (DOM) directly; thus the page does not need to be reloaded in full from the server for every small change, as would be necessary without AJAX.

The AJAX approach requires an endpoint on the server for the JavaScript to fetch (usually a dynamically-generated page which takes query parameters). In the context of Java web servers, this endpoint would typically be a servlet or a JSP.

Figure 6.4 shows the process of requesting and displaying a normal (non-AJAX) web page, which may be compared with the modified sequence using AJAX in Figure 6.5.

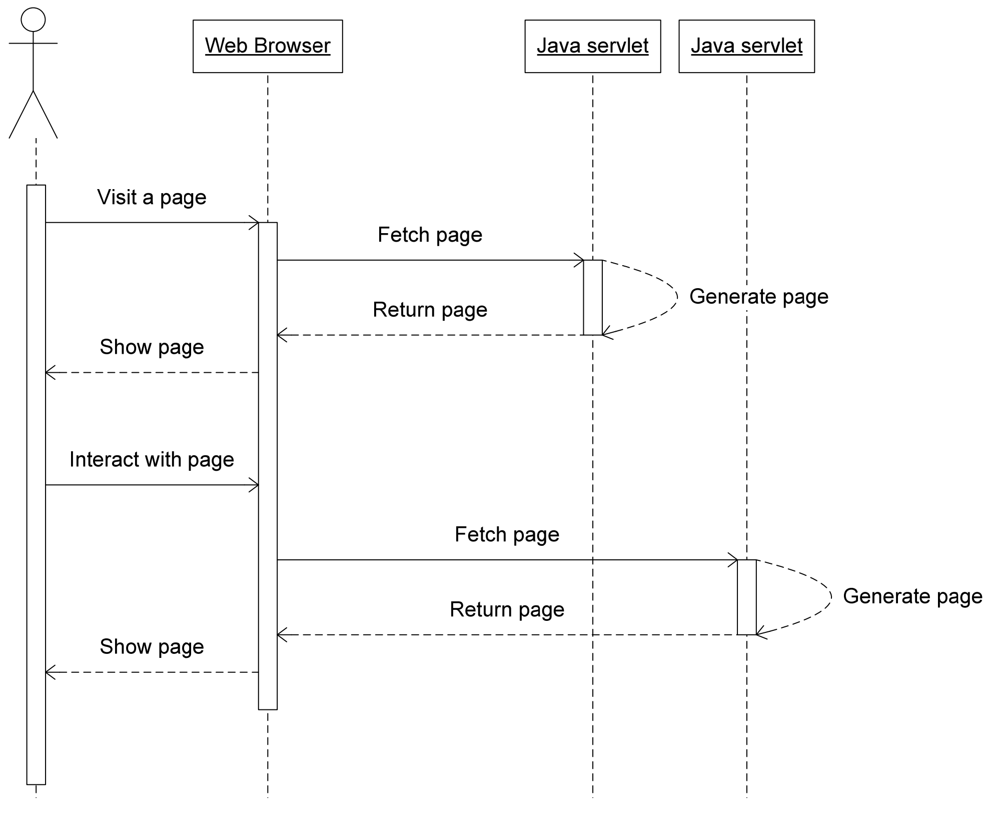


Figure 6.4: Interactions on a normal web page (not using AJAX).

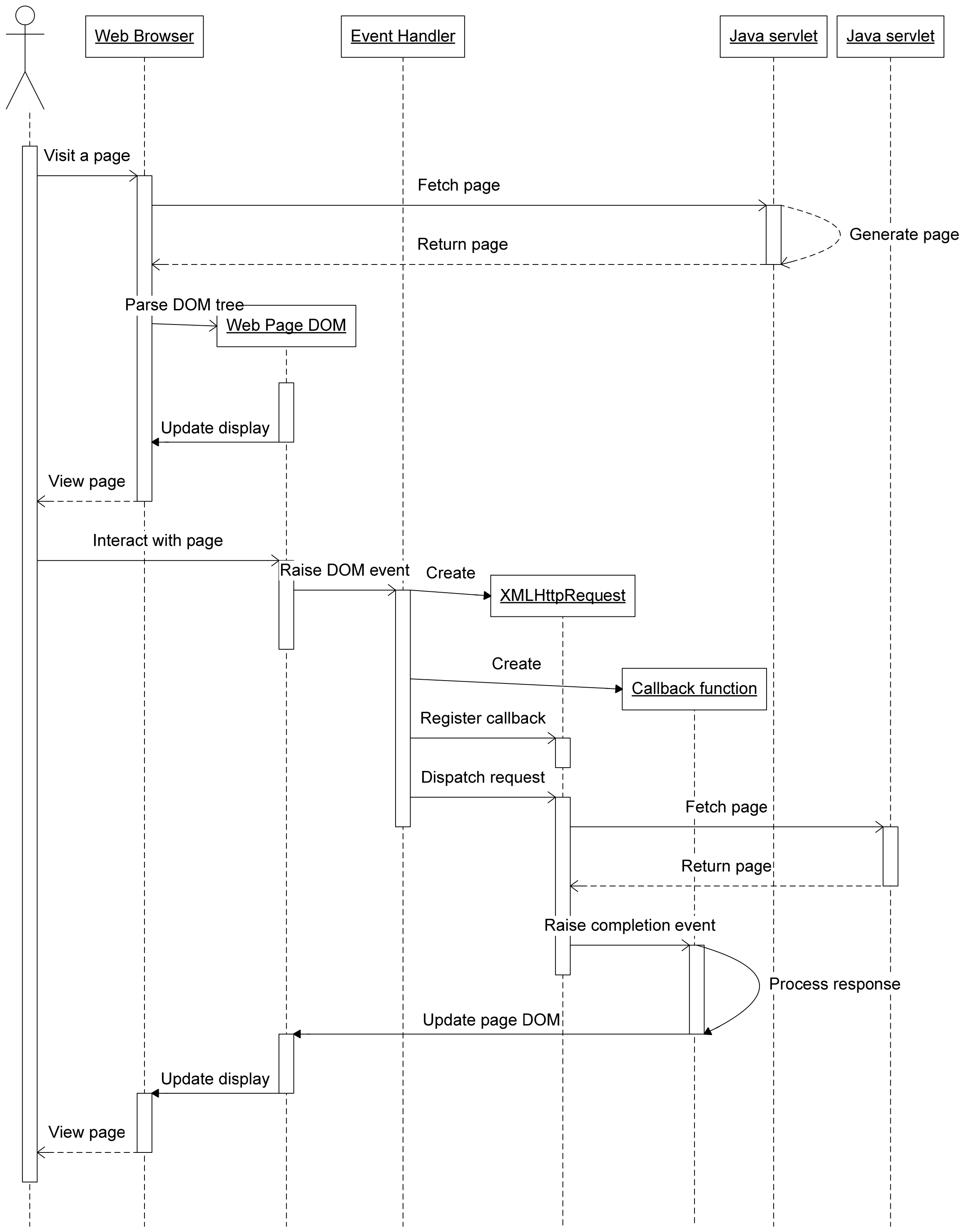


Figure 6.5: Interactions on a web page using AJAX. Based upon a diagram from the IBM developerWorks article "Ajax for Java developers: Build dynamic Java applications"[160].

6.4 Portlets developed

A number of custom portlets were developed to meet Windber's specific requirements, and are discussed below. These JSR-168 portlets use the portlet messaging library (Chapter 5) to communicate with Discovery Net portlets, and so were deployed, for convenience, within the same portlet application as the Discovery Net portlets. They could have been deployed in a separate portlet application, but it would then have been necessary to set up an external message store for cross-context communication, which we considered an additional complexity worth avoiding.

6.4.1 OLAP Browser

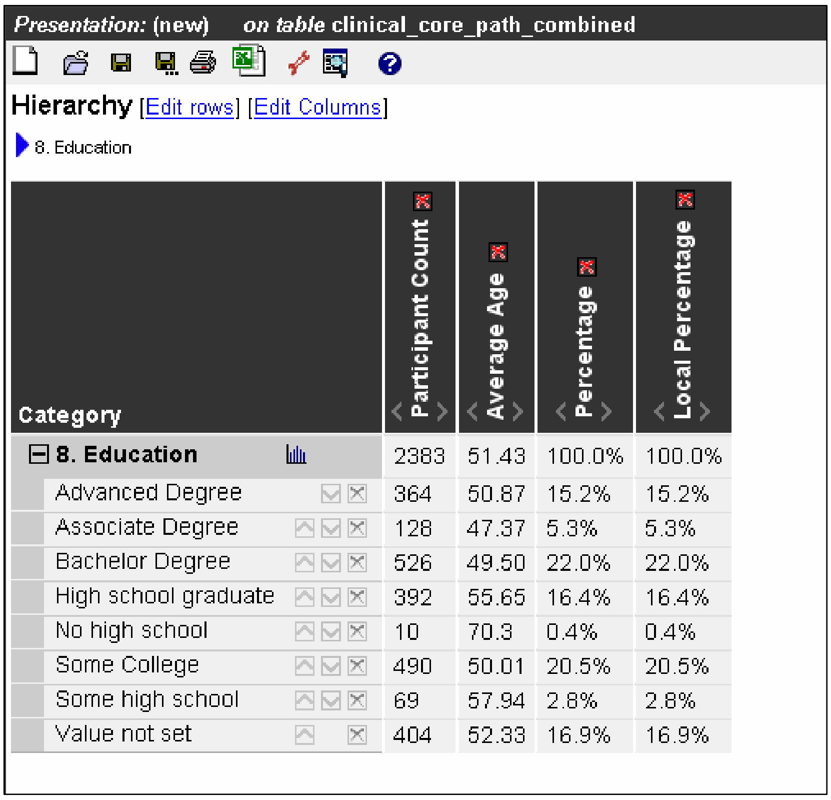
The OLAP browser presents an interactive tabular view of the data, and also allows users to generate bar charts as an alternative visualisation.

Categorical attributes (or binnings), which we will refer to as categories or dimensions, can be added to the table as sets of rows or sets of columns, with an entry for each possible value of the category. If both rows and columns contain categories, the resulting table is a cross-tabulation (crosstab) where each data cell contains the intersection of a pair of category values.

"Measures" are numerical aggregate values, calculated over a particular set of data, and define what information is shown in the table. A measure is defined by an aggregate function applied over a numerical attribute. For example, the record Count (*"Count(CBCPNum)"*) or average Age (*"Avg(Age)"*).

Typically, a user will add a category as a set of rows (one row for each possible category value), and several columns containing numerical measures (Figure 6.8 shows the configuration of rows and columns, and Figure 6.6 shows the resulting table). The data cells in the table will then show the measures corresponding to each row. Each cell can retrieve its data using a unique database query, made up of the measure and the row information which effectively acts as a filter (e.g. "*Avg(Age) WHERE Occupation='Teacher'*"). Users may also add categories as columns, where a single category results in a group of columns being added to the table, one column for each possible category value (Figure 6.10). When adding a categorical column, the user must additionally specify a measure as usual. The resulting crosstab changes the way each data cell constructs its query: the filter for selection now includes parts from both the row and the column.

Users may view the data from any row in the table as a graph (Figure 6.7), or export the underlying raw data (Figure 6.9).


Presentation: (new) on table clinical_core_path_combined
Hierarchy [Edit rows] [Edit Columns]
8. Education

Category	Participant Count	Average Age	Percentage	Local Percentage
8. Education	2383	51.43	100.0%	100.0%
Advanced Degree	364	50.87	15.2%	15.2%
Associate Degree	128	47.37	5.3%	5.3%
Bachelor Degree	526	49.50	22.0%	22.0%
High school graduate	392	55.65	16.4%	16.4%
No high school	10	70.3	0.4%	0.4%
Some College	490	50.01	20.5%	20.5%
Some high school	69	57.94	2.8%	2.8%
Value not set	404	52.33	16.9%	16.9%

Figure 6.6: The OLAP browser showing a single category ('Education') as rows, and several columns with numerical measures. A single row category can also be visualised using a graph, showing the data for one column (Figure 6.7).

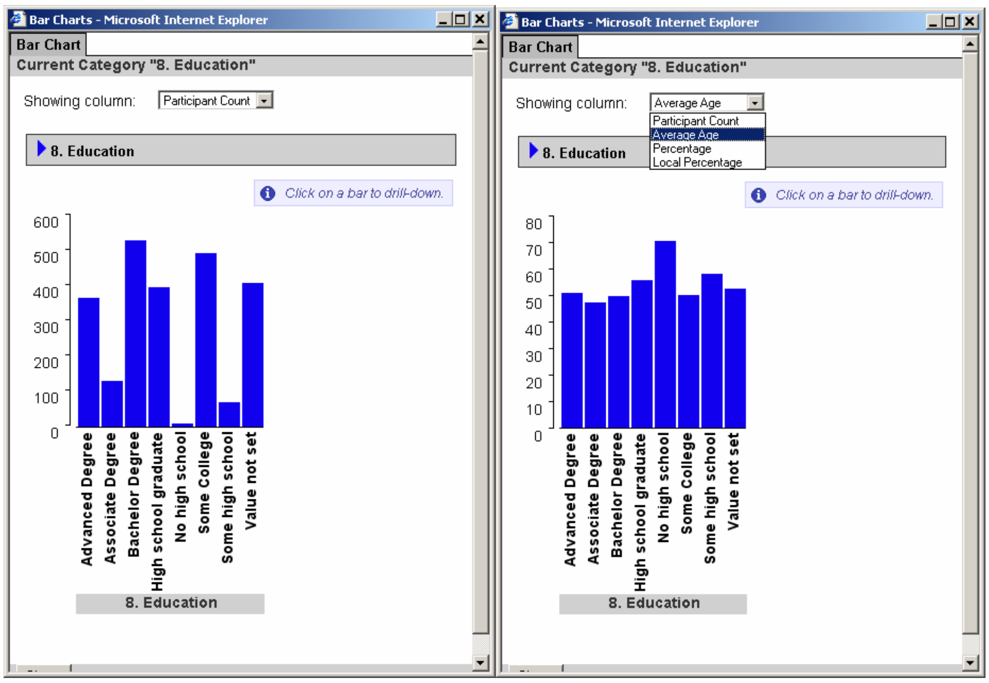
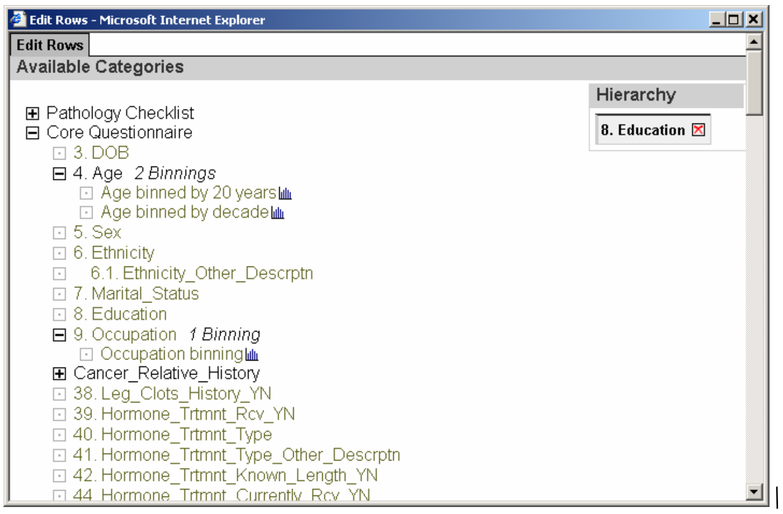


Figure 6.7: The information in the OLAP table can also be viewed graphically.



Edit Rows - Microsoft Internet Explorer

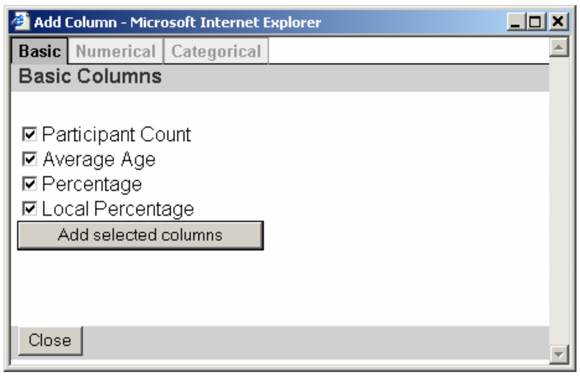
Edit Rows

Available Categories

- Pathology Checklist
- Core Questionnaire
  - 3. DOB
  - 4. Age *2 Binnings*
    - Age binned by 20 years
    - Age binned by decade
  - 5. Sex
  - 6. Ethnicity
    - 6.1. Ethnicity_Other_Descrptn
  - 7. Marital_Status
  - 8. Education
  - 9. Occupation *1 Binning*
    - Occupation binning
  - Cancer_Relative_History
  - 38. Leg_Clots_History_YN
  - 39. Hormone_Trtmnt_Rcv_YN
  - 40. Hormone_Trtmnt_Type
  - 41. Hormone_Trtmnt_Type_Other_Descrptn
  - 42. Hormone_Trtmnt_Known_Length_YN
  - 44 Hormone_Trtmnt_Currently_Rcv_YN

Hierarchy

8. Education



Add Column - Microsoft Internet Explorer

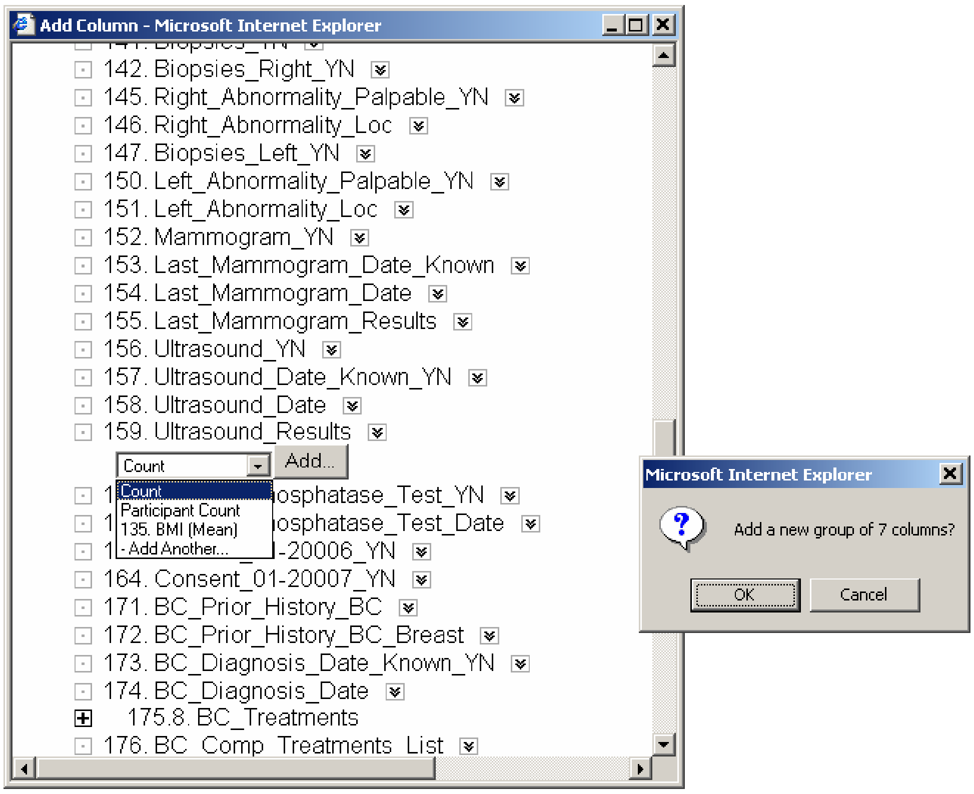
Basic | Numerical | Categorical

Basic Columns

- ☑ Participant Count
- ☑ Average Age
- ☑ Percentage
- ☑ Local Percentage

Add selected columns

Close



Add Column - Microsoft Internet Explorer

142. Biopsies_Right_YN
145. Right_Abnormality_Palpable_YN
146. Right_Abnormality_Loc
147. Biopsies_Left_YN
150. Left_Abnormality_Palpable_YN
151. Left_Abnormality_Loc
152. Mammogram_YN
153. Last_Mammogram_Date_Known
154. Last_Mammogram_Date
155. Last_Mammogram_Results
156. Ultrasound_YN
157. Ultrasound_Date_Known_YN
158. Ultrasound_Date
159. Ultrasound_Results

Count | Add...

Count
Participant Count
135. BMI (Mean)
- Add Another...

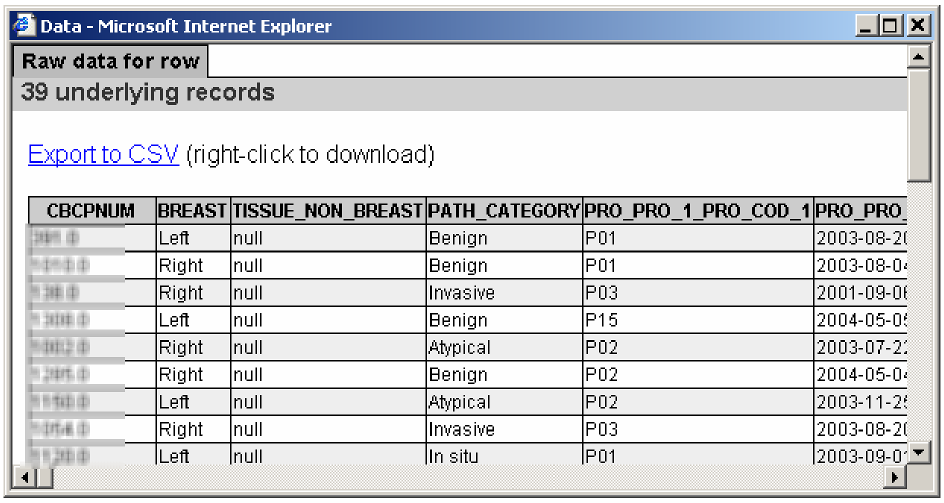
osphatase_Test_YN
osphatase_Test_Date
-20006_YN
164. Consent_01-20007_YN
171. BC_Prior_History_BC
172. BC_Prior_History_BC_Breast
173. BC_Diagnosis_Date_Known_YN
174. BC_Diagnosis_Date
175.8. BC_Treatments
176. BC_Comp_Treatments_List

Microsoft Internet Explorer

Add a new group of 7 columns?

OK | Cancel

Figure 6.8: Configuration of rows and columns in the OLAP browser.



Data - Microsoft Internet Explorer

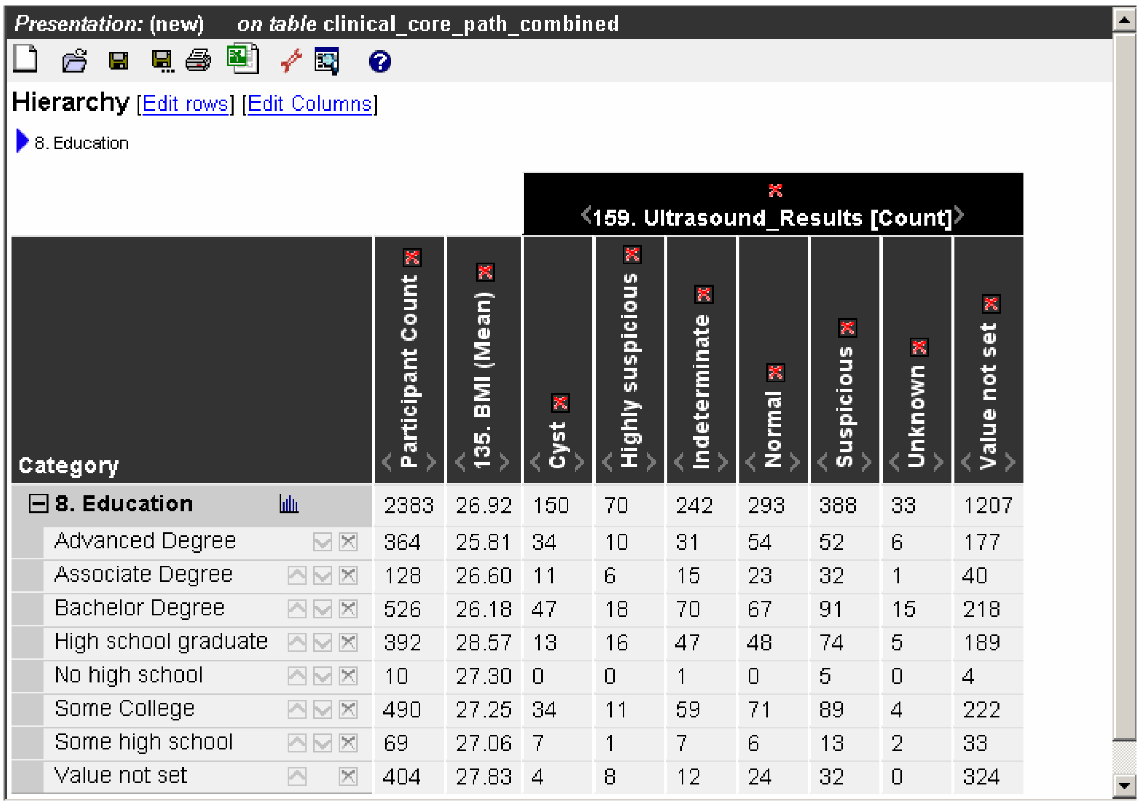
Raw data for row

39 underlying records

Export to CSV (right-click to download)

CBCPNUM	BREAST	TISSUE_NON_BREAST	PATH_CATEGORY	PRO_PRO_1_PRO_COD_1	PRO_PRO_
[redacted]	Left	null	Benign	P01	2003-08-2
[redacted]	Right	null	Benign	P01	2003-08-0
[redacted]	Right	null	Invasive	P03	2001-09-0
[redacted]	Left	null	Benign	P15	2004-05-0
[redacted]	Right	null	Atypical	P02	2003-07-2
[redacted]	Right	null	Benign	P02	2004-05-0
[redacted]	Left	null	Atypical	P02	2003-11-2
[redacted]	Right	null	Invasive	P03	2003-08-2
[redacted]	Left	null	In situ	P01	2003-09-0

Figure 6.9: The raw data records (individual patients) underlying each OLAP browser row.



Presentation: (new) *on table* clinical_core_path_combined

Hierarchy [Edit rows] [Edit Columns]

8. Education

			159. Ultrasound_Results [Count]						
Category	**Participant Count**	**135. BMI (Mean)**	**Cyst**	**Highly suspicious**	**Indeterminate**	**Normal**	**Suspicious**	**Unknown**	**Value not set**
8. Education	2383	26.92	150	70	242	293	388	33	1207
Advanced Degree	364	25.81	34	10	31	54	52	6	177
Associate Degree	128	26.60	11	6	15	23	32	1	40
Bachelor Degree	526	26.18	47	18	70	67	91	15	218
High school graduate	392	28.57	13	16	47	48	74	5	189
No high school	10	27.30	0	0	1	0	5	0	4
Some College	490	27.25	34	11	59	71	89	4	222
Some high school	69	27.06	7	1	7	6	13	2	33
Value not set	404	27.83	4	8	12	24	32	0	324

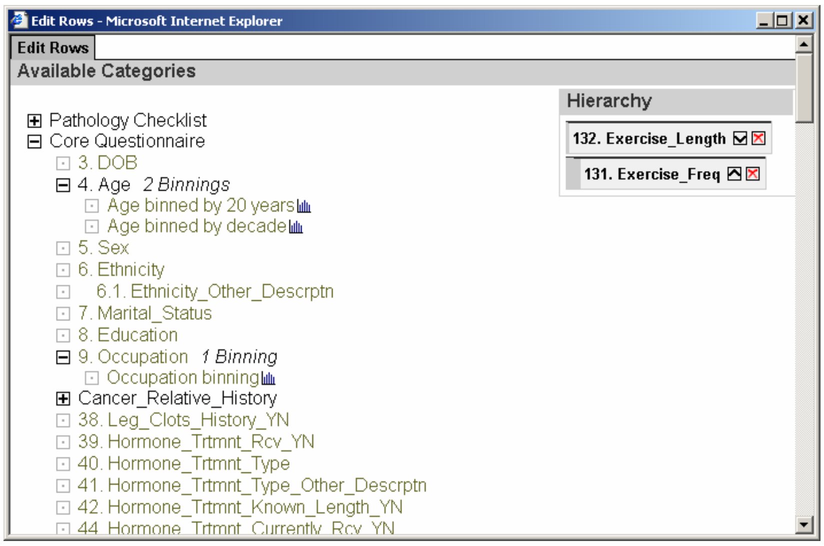
Figure 6.10: The OLAP browser showing a single category as rows, two numerical columns, and a single categorical column (a group of columns), resulting in a crosstab.

This basic layout is enhanced by the ability to "drill-down" into rows: users may expand a particular row of interest and add an additional category beneath it as a new set of rows (Figure 6.11). In early prototypes of this system, users would select a category to expand before expanding any row, and in this way different categories could be shown at the same 'level' of the graph: e.g. with a top level category of '*Occupation*', the "*Occupation=Teacher*" row could be expanded to show '*Exercise*', while an "*Occupation=Housewife*" row might be expanded to show '*Education*'. This allowed complete flexibility in viewing data, but the Windber researchers found it too complex and suggested a simpler approach. Rather than allowing full choice with every expansion, they preferred to define a 'hierarchy' of rows once, which would fix which categories should be expanded at each level. For example, a row hierarchy containing "*Education→ Exercise→ Exercise Frequency*" would ensure that all expansions of '*Education*' rows would result in a nested set of '*Exercise*' rows, and then any expansions of '*Exercise*' rows would expand to show nested "*Exercise Frequency*" rows (Figure 6.11, Figure 6.12).

Thus, users can select attributes and construct measures of interest to create their table, and drill down through the row hierarchy to examine the aggregate information about groups of patients with particular attribute values. They can also customise details of the table display, hiding rows and columns or re-ordering them.

Once the user has identified an interesting group of patients, they can then select the corresponding row, which is defined by a simple "select" query appropriate for accessing the database (e.g. "*Ethnicity='White' AND Age_binning='40-50' AND Exercise_Frequency='Once a week'*"). This process is the only one which results in a full portal page reload: this is so that the select query can be passed along by the portlet code to the Patient Sets portlet, using the portlet messaging library. Patient Sets, each defined by a select query, can then be used on other portal pages – e.g. to view the details of their contained patients, or to use sets of patients as input to a workflow.

a)



Edit Rows - Microsoft Internet Explorer

Edit Rows

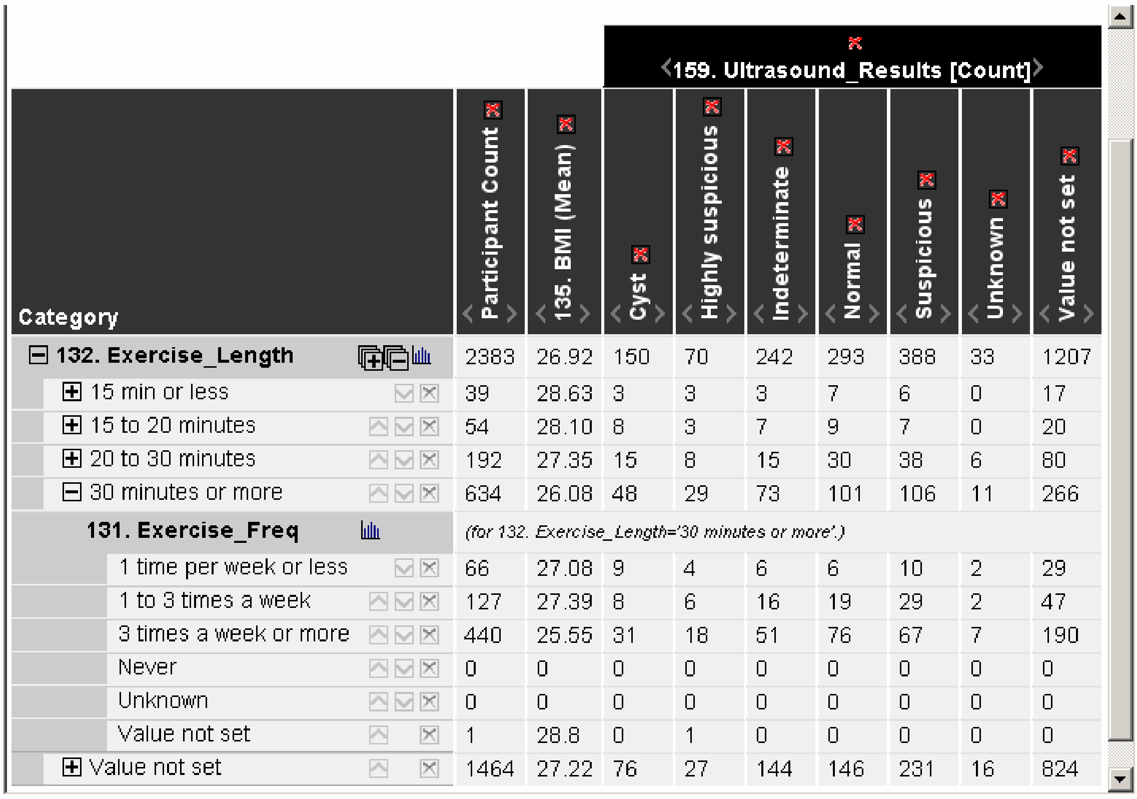
Available Categories

- ⊞ Pathology Checklist
- ⊟ Core Questionnaire
  - 3. DOB
  - ⊟ 4. Age *2 Binnings*
    - Age binned by 20 years
    - Age binned by decade
  - 5. Sex
  - 6. Ethnicity
    - 6.1. Ethnicity_Other_Descrptn
  - 7. Marital_Status
  - 8. Education
  - ⊟ 9. Occupation *1 Binning*
    - Occupation binning
  - ⊞ Cancer_Relative_History
  - 38. Leg_Clots_History_YN
  - 39. Hormone_Trtmnt_Rcv_YN
  - 40. Hormone_Trtmnt_Type
  - 41. Hormone_Trtmnt_Type_Other_Descrptn
  - 42. Hormone_Trtmnt_Known_Length_YN
  - 44. Hormone_Trtmnt_Currently_Rcv_YN

Hierarchy

- 132. Exercise_Length
- 131. Exercise_Freq

b)

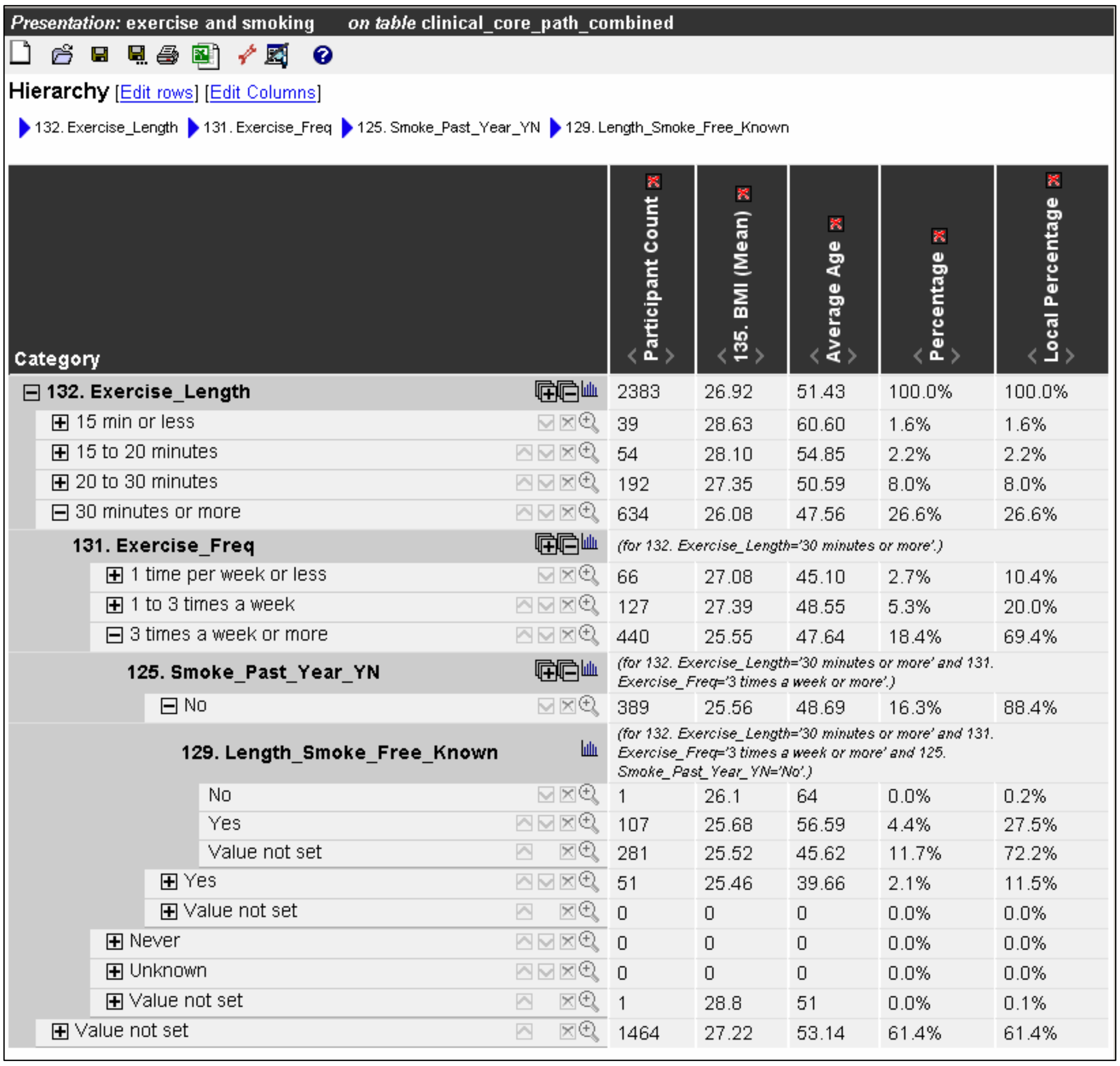


Category	Participant Count	135. BMI (Mean)	159. Ultrasound_Results [Count]: Cyst	Highly suspicious	Indeterminate	Normal	Suspicious	Unknown	Value not set
⊟ 132. Exercise_Length	2383	26.92	150	70	242	293	388	33	1207
⊞ 15 min or less	39	28.63	3	3	3	7	6	0	17
⊞ 15 to 20 minutes	54	28.10	8	3	7	9	7	0	20
⊞ 20 to 30 minutes	192	27.35	15	8	15	30	38	6	80
⊟ 30 minutes or more	634	26.08	48	29	73	101	106	11	266
131. Exercise_Freq	*(for 132. Exercise_Length='30 minutes or more')*								
1 time per week or less	66	27.08	9	4	6	6	10	2	29
1 to 3 times a week	127	27.39	8	6	16	19	29	2	47
3 times a week or more	440	25.55	31	18	51	76	67	7	190
Never	0	0	0	0	0	0	0	0	0
Unknown	0	0	0	0	0	0	0	0	0
Value not set	1	28.8	0	1	0	0	0	0	0
⊞ Value not set	1464	27.22	76	27	144	146	231	16	824

Figure 6.11: Defining and displaying a row hierarchy in the OLAP browser.

a) A row hierarchy configured with two categories.

b) Expanding an "Exercise" row to show information for the second category in the hierarchy "Exercise Frequency".



Presentation: exercise and smoking *on table* clinical_core_path_combined

Hierarchy [Edit rows] [Edit Columns]

▶ 132. Exercise_Length ▶ 131. Exercise_Freq ▶ 125. Smoke_Past_Year_YN ▶ 129. Length_Smoke_Free_Known

Category	Participant Count	135. BMI (Mean)	Average Age	Percentage	Local Percentage
⊟ **132. Exercise_Length**	2383	26.92	51.43	100.0%	100.0%
⊞ 15 min or less	39	28.63	60.60	1.6%	1.6%
⊞ 15 to 20 minutes	54	28.10	54.85	2.2%	2.2%
⊞ 20 to 30 minutes	192	27.35	50.59	8.0%	8.0%
⊟ 30 minutes or more	634	26.08	47.56	26.6%	26.6%
131. Exercise_Freq	*(for 132. Exercise_Length='30 minutes or more'.)*				
⊞ 1 time per week or less	66	27.08	45.10	2.7%	10.4%
⊞ 1 to 3 times a week	127	27.39	48.55	5.3%	20.0%
⊟ 3 times a week or more	440	25.55	47.64	18.4%	69.4%
125. Smoke_Past_Year_YN	*(for 132. Exercise_Length='30 minutes or more' and 131. Exercise_Freq='3 times a week or more'.)*				
⊟ No	389	25.56	48.69	16.3%	88.4%
129. Length_Smoke_Free_Known	*(for 132. Exercise_Length='30 minutes or more' and 131. Exercise_Freq='3 times a week or more' and 125. Smoke_Past_Year_YN='No'.)*				
No	1	26.1	64	0.0%	0.2%
Yes	107	25.68	56.59	4.4%	27.5%
Value not set	281	25.52	45.62	11.7%	72.2%
⊞ Yes	51	25.46	39.66	2.1%	11.5%
⊞ Value not set	0	0	0	0.0%	0.0%
⊞ Never	0	0	0	0.0%	0.0%
⊞ Unknown	0	0	0	0.0%	0.0%
⊞ Value not set	1	28.8	51	0.0%	0.1%
⊞ Value not set	1464	27.22	53.14	61.4%	61.4%

Figure 6.12: Multiple levels of row expansions, governed by the row hierarchy (shown above the table).

The OLAP browser uses AJAX to allow the table presented in the web browser to be very responsive and interactive. JavaScript is used to asynchronously fetch data in the background and update the table on the page, without reloading the page from the server. Thus the methods mentioned above of constructing database queries for each cell are useful; each table cell has its associated query sent to the portal server, and updates immediately when its result is received. Users can view the result in each cell as soon as it arrives, rather than waiting for the entire table of data to be calculated and rendered in a single page request.

The OLAP browser was developed as a set of JSPs, as it was meant to be accessible independently of the Jetspeed portal. A simple wrapper portlet was developed which embedded the interface into the portal page using an IFRAME.

The display state of the OLAP browser (the table layout: its columns, the row hierarchy, its expansion state, and the ordering and visibility of rows and columns) was preserved in the session, so when users navigate to a different page on the portal and later return to the OLAP browser, it shows the table exactly as they last left it. Display persistence was also necessary when generating a CSV export of the table, which needed to be identical to the table layout seen in the portal. Whenever the display state changes as a result of AJAX requests, AJAX was also used to update the state in the session. An alternative persistence approach well-suited to AJAX applications is to store state in a client-side cookie, but in this case the information needed to describe a complex table layout was potentially too large (cookies have a maximum size of 4KB).

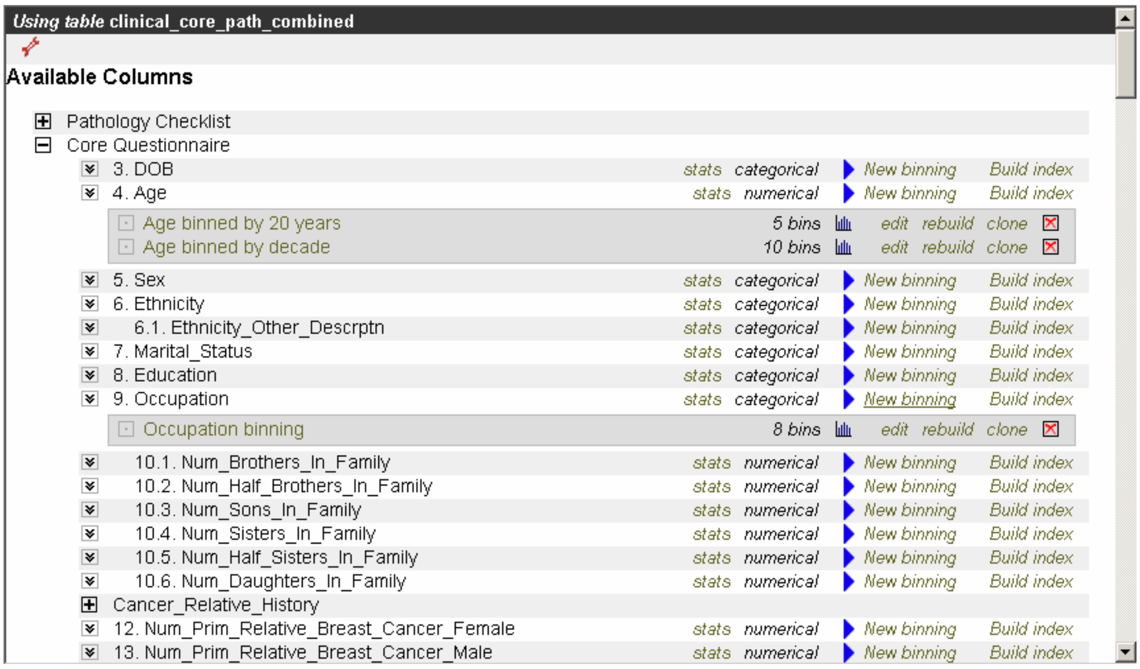
6.4.2 Binning

A binning helper application was developed to allow the user to customise the data for their needs.

The underlying data consists of a number of patient records, each patient identified by a unique ID (‘*CBCPNum*’ in the screenshots). Associated with each patient may be any number of questionnaires and items of clinical data (e.g. tissue sample test results). In this phase of the project, it was considered acceptable to view this as a flattened record: every questionnaire answer or test item can fundamentally be viewed as a simple attribute of the patient.

Attributes (e.g. a questionnaire item) can be either numerical (e.g. ‘*Age*’, with continuous values) or categorical (e.g. ‘*Occupation*’, with a set of possible values). Both numerical and categorical attributes can be binned, and a binned attribute is by its nature categorical (e.g. ‘*Age*’ may be split up into 4 bins, of <30, <50, <70, 70+). It can also sometimes be useful to apply binnings to categorical attributes to simplify them, as these may contain free-form answers with many semantically similar values, e.g. ‘*Teacher*’, "*Primary school teacher*".

A web interface was developed allowing users to create new binnings on columns, and perform a limited number of other administrative functions (e.g. indexing). The binning pages are shown in Figure 6.13 - Figure 6.17.

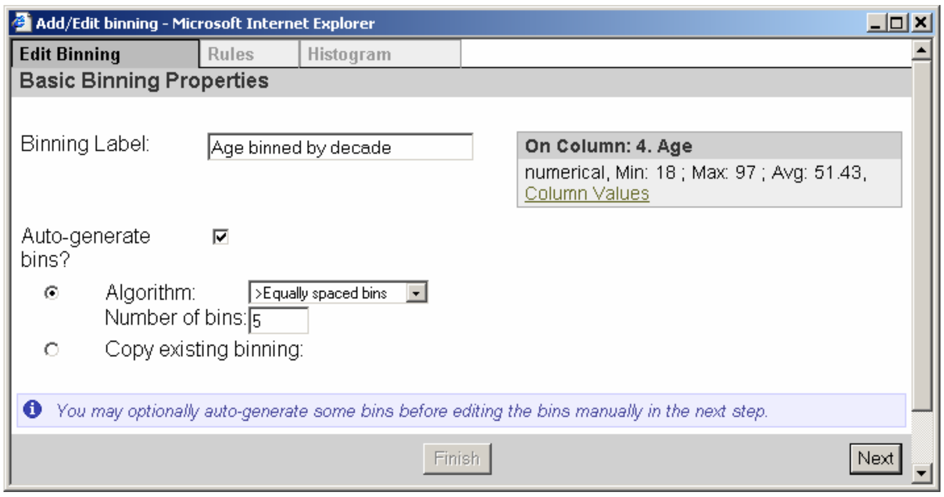


Using table **clinical_core_path_combined**

Available Columns

⊞ Pathology Checklist				
⊟ Core Questionnaire				
3. DOB	stats	categorical	New binning	Build index
4. Age	stats	numerical	New binning	Build index
Age binned by 20 years	5 bins		edit rebuild clone	
Age binned by decade	10 bins		edit rebuild clone	
5. Sex	stats	categorical	New binning	Build index
6. Ethnicity	stats	categorical	New binning	Build index
6.1. Ethnicity_Other_Descrptn	stats	categorical	New binning	Build index
7. Marital_Status	stats	categorical	New binning	Build index
8. Education	stats	categorical	New binning	Build index
9. Occupation	stats	categorical	New binning	Build index
Occupation binning	8 bins		edit rebuild clone	
10.1. Num_Brothers_In_Family	stats	numerical	New binning	Build index
10.2. Num_Half_Brothers_In_Family	stats	numerical	New binning	Build index
10.3. Num_Sons_In_Family	stats	numerical	New binning	Build index
10.4. Num_Sisters_In_Family	stats	numerical	New binning	Build index
10.5. Num_Half_Sisters_In_Family	stats	numerical	New binning	Build index
10.6. Num_Daughters_In_Family	stats	numerical	New binning	Build index
⊞ Cancer_Relative_History				
12. Num_Prim_Relative_Breast_Cancer_Female	stats	numerical	New binning	Build index
13. Num_Prim_Relative_Breast_Cancer_Male	stats	numerical	New binning	Build index

Figure 6.13: The binning interface begins by showing the list of columns in a chosen table. Each column can be expanded to show its corresponding binnings, if any, and the binnings can be created or managed from here.



Add/Edit binning - Microsoft Internet Explorer

Basic Binning Properties

Binning Label: Age binned by decade

On Column: 4. Age
numerical, Min: 18 ; Max: 97 ; Avg: 51.43,
Column Values

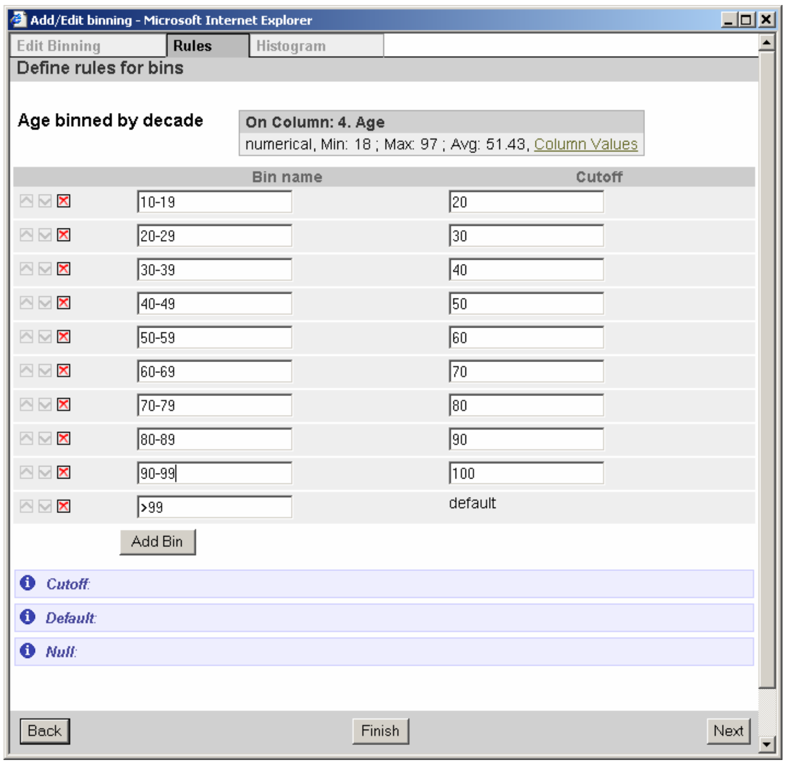
Auto-generate bins? ✓

● Algorithm: >Equally spaced bins
Number of bins: 5

○ Copy existing binning:

You may optionally auto-generate some bins before editing the bins manually in the next step.

Figure 6.14: The wizard for creating or editing binnings. These can be initially created using an algorithm or by copying an existing binning.


Add/Edit binning - Microsoft Internet Explorer

Edit Binning | Rules | Histogram

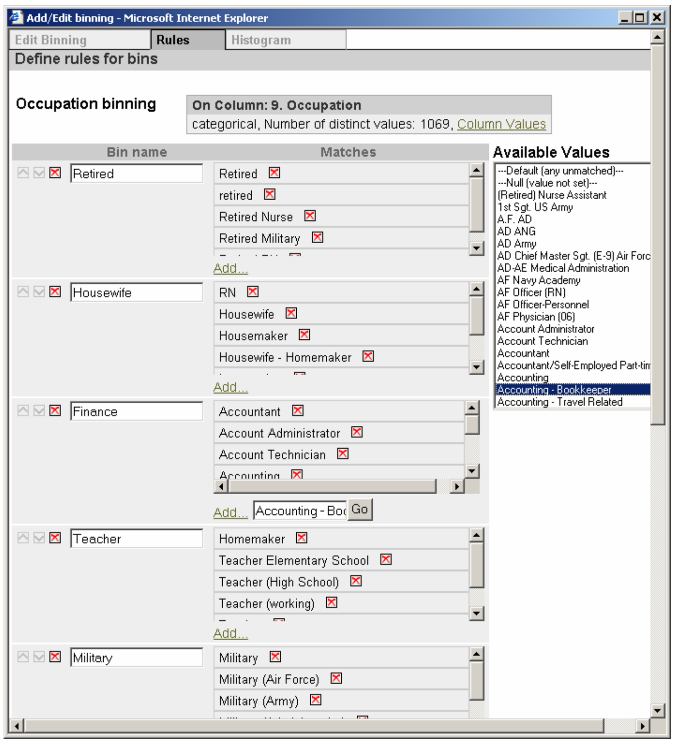
Define rules for bins

Age binned by decade

On Column: 4. Age
numerical, Min: 18 ; Max: 97 ; Avg: 51.43, Column Values

Bin name	Cutoff
10-19	20
20-29	30
30-39	40
40-49	50
50-59	60
60-69	70
70-79	80
80-89	90
90-99	100
>99	default

Figure 6.15: Interface for manual customisation of bins, for a numerical column.


Add/Edit binning - Microsoft Internet Explorer

Edit Binning | Rules | Histogram

Define rules for bins

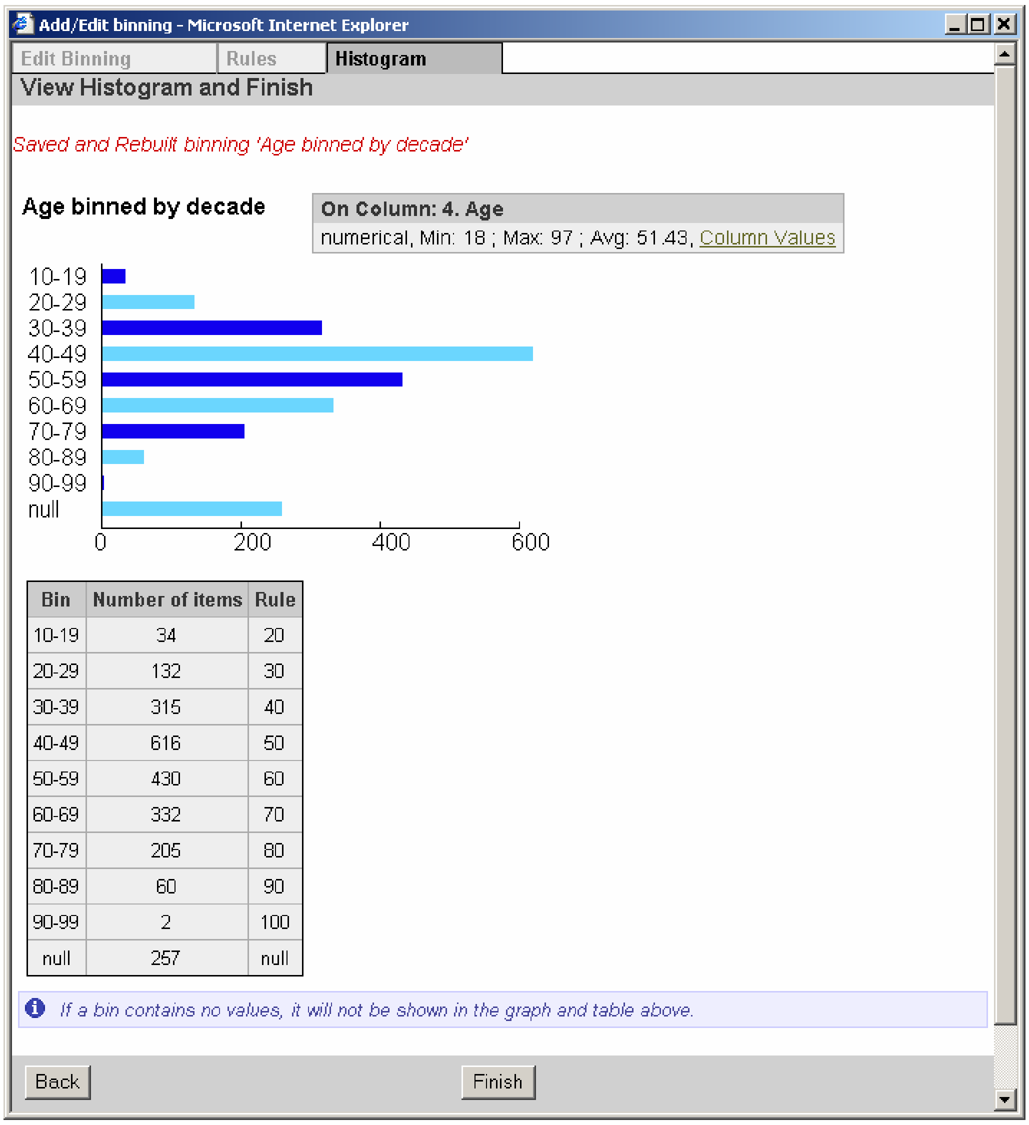
Occupation binning

On Column: 9. Occupation
categorical, Number of distinct values: 1069, Column Values

Bin name	Matches
Retired	Retired, retired, Retired Nurse, Retired Military
Housewife	RN, Housewife, Housemaker, Housewife - Homemaker
Finance	Accountant, Account Administrator, Account Technician, Accounting
Teacher	Homemaker, Teacher Elementary School, Teacher (High School), Teacher (working)
Military	Military, Military (Air Force), Military (Army)

Available Values
---Default (any unmatched)---
---Null (value not set)---
(Retired) Nurse Assistant
1st Sgt. US Army
A.F. AD
AD ANG
AD Army
AD Chief Master Sgt. (E-9) Air Forc
AD-AE Medical Administration
AF Navy Academy
AF Officer (RN)
AF Officer-Personnel
AF Physician (06)
Account Administrator
Account Technician
Accountant
Accountant/Self-Employed Part-tim
Accounting
Accounting - Bookkeeper
Accounting - Travel Related

Figure 6.16: Interface for manual customisation of bins, for a categorical column.



View Histogram and Finish

Saved and Rebuilt binning 'Age binned by decade'

Age binned by decade

On Column: 4. Age
numerical, Min: 18 ; Max: 97 ; Avg: 51.43, Column Values

Bin	Number of items	Rule
10-19	34	20
20-29	132	30
30-39	315	40
40-49	616	50
50-59	430	60
60-69	332	70
70-79	205	80
80-89	60	90
90-99	2	100
null	257	null

If a bin contains no values, it will not be shown in the graph and table above.

Figure 6.17: The binning wizard ends with a visualisation of the bin contents.

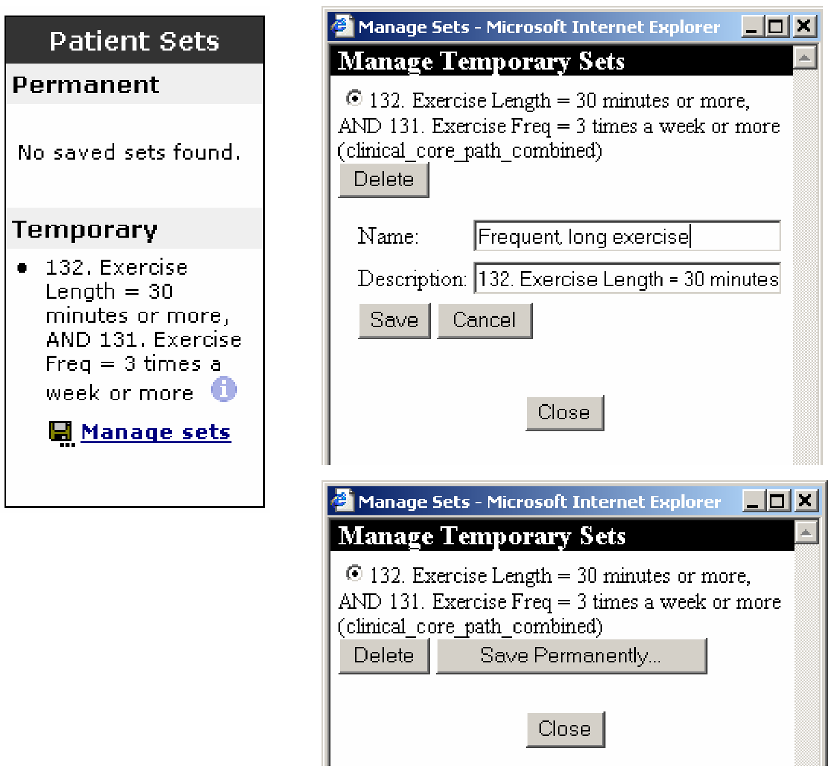
The binning web interface was developed as a set of JSPs, as it was meant to be accessible independently of the Jetspeed portal. These JSPs use AJAX for improved responsiveness and interactivity (e.g. in retrieving the binnings for a column: these binnings are fetched using AJAX and inserted directly into the page, rather than having to reload the entire page and re-render the full column list– or, alternatively, fetching the binnings for all columns when the page loads).

A simple wrapper portlet was developed which embedded the binning interface into the portal page using an IFRAME.

6.4.3 Patient Sets portlet

The Patient Sets portlet is a dedicated JSR-168 portlet (it cannot be accessed outside the context of the portal), which acts like a "shopping basket" of available patient sets (Figure 6.18). Any patient sets selected by the user in the OLAP browser will be added to the list of sets in this portlet. Patient sets can be permanently saved to the database, or simply used in the current portal session. The core of a patient set is the database query which can be used to retrieve the set; each set also has an associated name and description.

This portlet has a number of display modes, appropriate for different portal pages (Figure 6.19). It uses the portlet messaging library to communicate with other portlets: it can send the selected patient ID to the Patient Viewer portlet, and patient set queries as input to Discovery Net Service portlets.

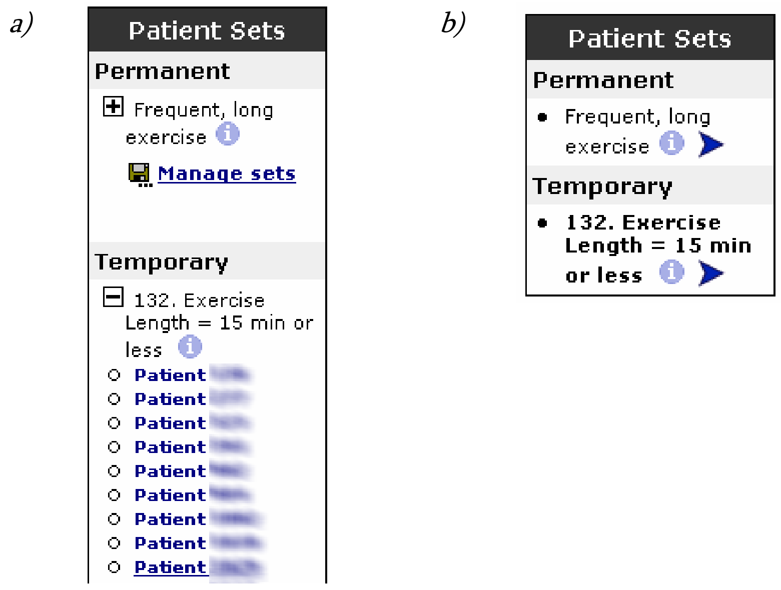

Patient Sets
Permanent
No saved sets found.
Temporary
- 132. Exercise Length = 30 minutes or more, AND 131. Exercise Freq = 3 times a week or more

Manage sets

Manage Sets - Microsoft Internet Explorer
Manage Temporary Sets
(•) 132. Exercise Length = 30 minutes or more, AND 131. Exercise Freq = 3 times a week or more (clinical_core_path_combined)
Delete
Name: Frequent long exercise
Description: 132. Exercise Length = 30 minutes
Save Cancel
Close

Manage Sets - Microsoft Internet Explorer
Manage Temporary Sets
(•) 132. Exercise Length = 30 minutes or more, AND 131. Exercise Freq = 3 times a week or more (clinical_core_path_combined)
Delete Save Permanently...
Close

Figure 6.18: Patient Sets portlet, allowing sets to be saved, edited and deleted.


a)
Patient Sets
Permanent
⊞ Frequent, long exercise
Manage sets
Temporary
⊟ 132. Exercise Length = 15 min or less
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]

b)
Patient Sets
Permanent
- Frequent, long exercise
Temporary
- **132. Exercise Length = 15 min or less**

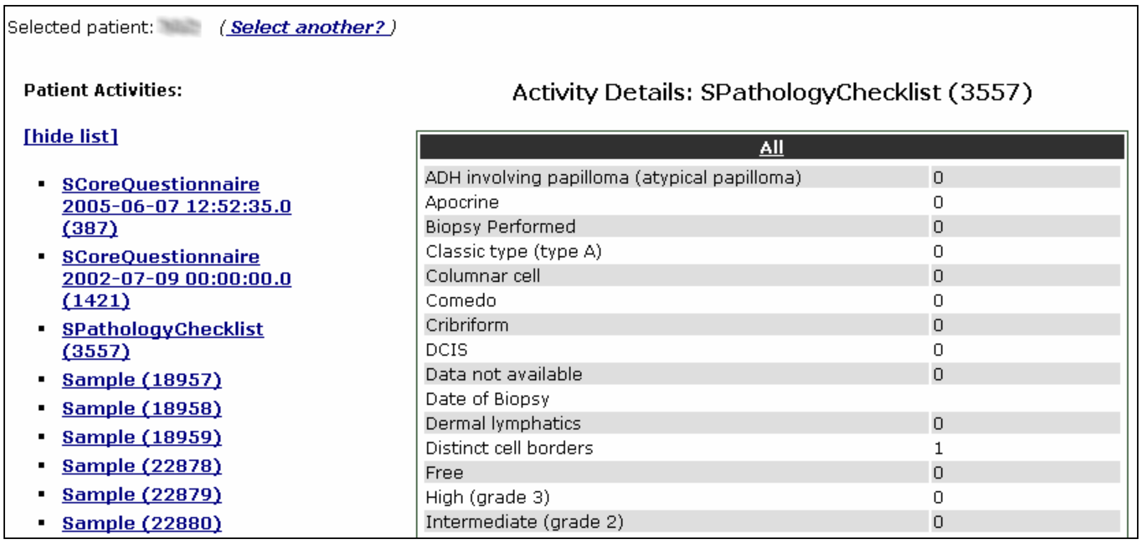
Figure 6.19: Patient Sets portlet.

a) allowing sets to be expanded to show patients within them. An individual patient ID can be selected for use in another portlet.

b) allowing selection of sets for use in other portlets.

6.4.4 Patient Viewer portlet

The Patient Viewer portlet displays fully detailed information for a selected patient. At the moment, this display is quite raw (Figure 6.20), simply displaying the attributes as a table of key-value pairs. Further development work will focus on an interface to explore and present this data.



Selected patient: [redacted] (*Select another?*)

Patient Activities:

[hide list]

- SCoreQuestionnaire 2005-06-07 12:52:35.0 (387)
- SCoreQuestionnaire 2002-07-09 00:00:00.0 (1421)
- SPathologyChecklist (3557)
- Sample (18957)
- Sample (18958)
- Sample (18959)
- Sample (22878)
- Sample (22879)
- Sample (22880)

Activity Details: SPathologyChecklist (3557)

All	
ADH involving papilloma (atypical papilloma)	0
Apocrine	0
Biopsy Performed	0
Classic type (type A)	0
Columnar cell	0
Comedo	0
Cribriform	0
DCIS	0
Data not available	0
Date of Biopsy	
Dermal lymphatics	0
Distinct cell borders	1
Free	0
High (grade 3)	0
Intermediate (grade 2)	0

Figure 6.20: The Patient Viewer portlet displays the clinical data for a specified patient. It may be configured to receive patient ID messages from a Patient Sets Portlet, and also allows users to select patients by specifying the patient's ID directly.

6.4.5 Discovery Net Services

The Discovery Net Service portlet was easily included on portal pages and connected with the Patient Sets portlet using the messaging library (Figure 6.25). This uses the mechanism for reading in messages as service input parameters described in Chapter 3.

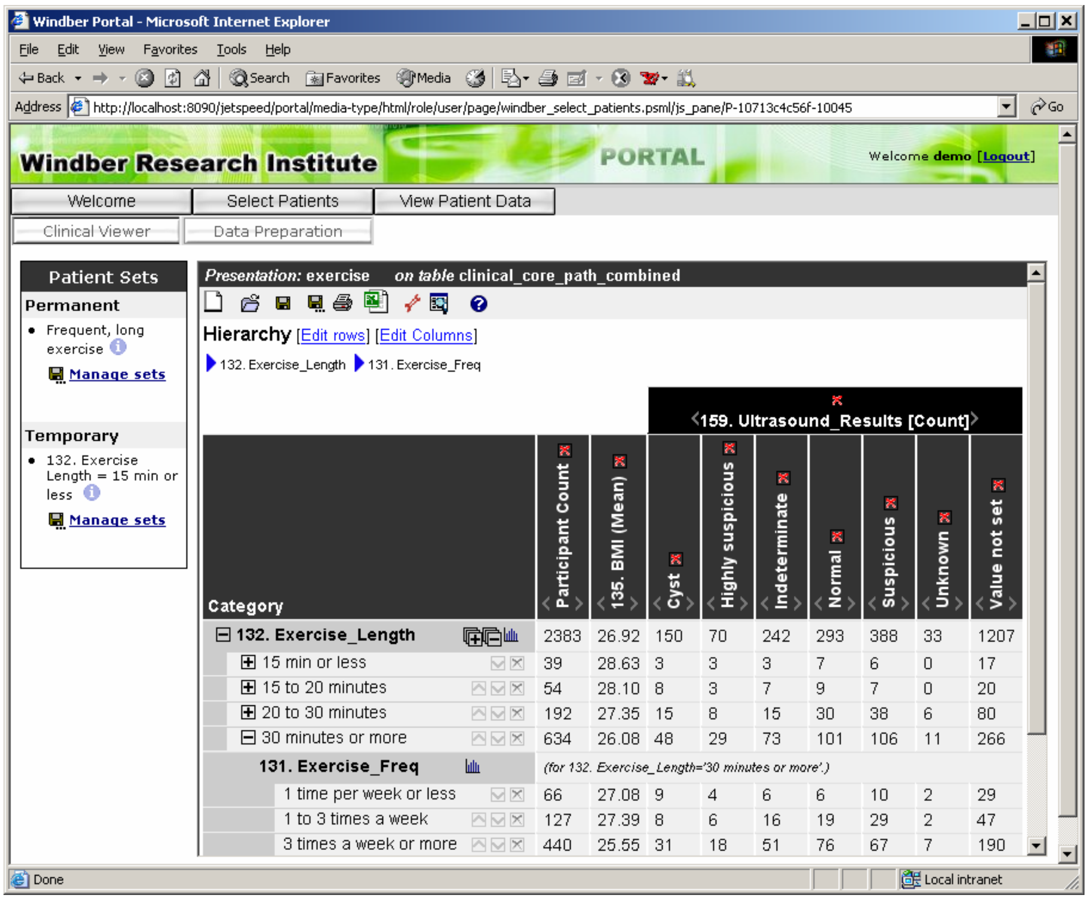
At this time, real analysis workflows are still under development by researchers at Windber, but the concept is proven, and services should be made available in later releases of the portal.

6.5 The Translational Medicine Portal

The final portal contains several pages, shown in this section. The OLAP browser currently provides the bulk of the functionality, allowing users to explore data and construct tables of interesting attributes (Figure 6.21). The data may be exported to Excel for further manipulation. Advanced users can create and modify binnings to customise the data (Figure 6.22). Interesting sets of patients which match the selected attributes can be saved for use elsewhere in the portal.

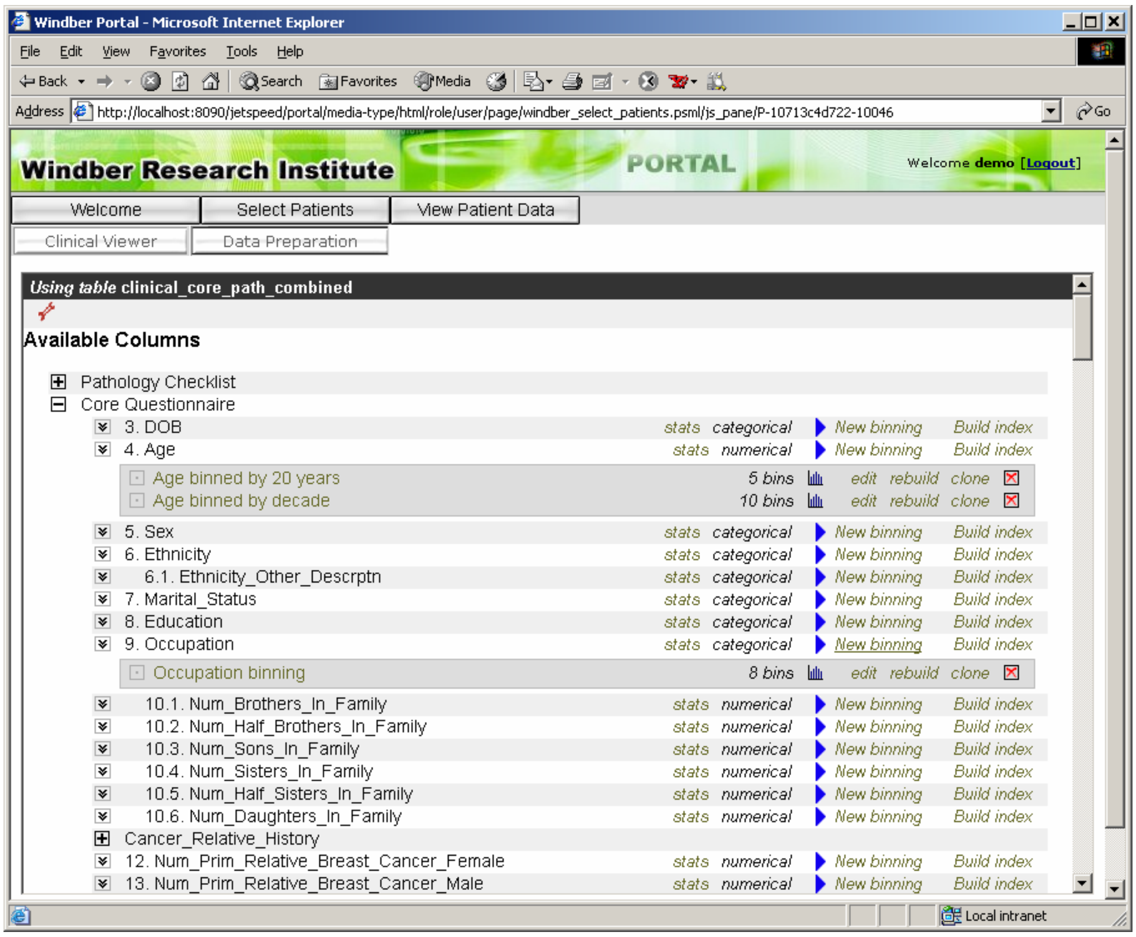
The Patient Viewer page displays detailed information on a selected patient – either chosen from within a patient set, or looked up by patient ID (Figure 6.23, Figure 6.24).

Finally, researchers can also use deployed workflows (Discovery Net services) in the portal, optionally using patient sets as inputs (Figure 6.25).


Windber Portal - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8090/jetspeed/portal/media-type/html/role/user/page/windber_select_patients.psml/js_pane/P-10713c4c56f-10045
Go
Windber Research Institute
PORTAL
Welcome demo [Logout]
Welcome
Select Patients
View Patient Data
Clinical Viewer
Data Preparation
Patient Sets
Permanent
Frequent, long exercise
Manage sets
Temporary
132. Exercise Length = 15 min or less
Manage sets
Presentation: exercise on table clinical_core_path_combined
Hierarchy [Edit rows] [Edit Columns]
132. Exercise_Length
131. Exercise_Freq
Done
Local intranet

Category	Participant Count	135. BMI (Mean)	Cyst	Highly suspicious	Indeterminate	Normal	Suspicious	Unknown	Value not set
			159. Ultrasound_Results [Count]						
132. Exercise_Length	2383	26.92	150	70	242	293	388	33	1207
15 min or less	39	28.63	3	3	3	7	6	0	17
15 to 20 minutes	54	28.10	8	3	7	9	7	0	20
20 to 30 minutes	192	27.35	15	8	15	30	38	6	80
30 minutes or more	634	26.08	48	29	73	101	106	11	266
131. Exercise_Freq	(for 132. Exercise_Length='30 minutes or more'.)								
1 time per week or less	66	27.08	9	4	6	6	10	2	29
1 to 3 times a week	127	27.39	8	6	16	19	29	2	47
3 times a week or more	440	25.55	31	18	51	76	67	7	190

Figure 6.21: The OLAP browser page. Selections from the OLAP browser are stored in the Patient Sets portlet.


Windber Portal - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
Windber Research Institute
PORTAL
Welcome demo [Logout]
Welcome
Select Patients
View Patient Data
Clinical Viewer
Data Preparation
Local intranet

http://localhost:8090/jetspeed/portal/media-type/html/role/user/page/windber_select_patients.psml/js_pane/P-10713c4d722-10046

Using table clinical_core_path_combined

Available Columns

Pathology Checklist				
Core Questionnaire				
3. DOB	stats	categorical	New binning	Build index
4. Age	stats	numerical	New binning	Build index
Age binned by 20 years	5 bins		edit rebuild clone	
Age binned by decade	10 bins		edit rebuild clone	
5. Sex	stats	categorical	New binning	Build index
6. Ethnicity	stats	categorical	New binning	Build index
6.1. Ethnicity_Other_Descrptn	stats	categorical	New binning	Build index
7. Marital_Status	stats	categorical	New binning	Build index
8. Education	stats	categorical	New binning	Build index
9. Occupation	stats	categorical	New binning	Build index
Occupation binning	8 bins		edit rebuild clone	
10.1. Num_Brothers_In_Family	stats	numerical	New binning	Build index
10.2. Num_Half_Brothers_In_Family	stats	numerical	New binning	Build index
10.3. Num_Sons_In_Family	stats	numerical	New binning	Build index
10.4. Num_Sisters_In_Family	stats	numerical	New binning	Build index
10.5. Num_Half_Sisters_In_Family	stats	numerical	New binning	Build index
10.6. Num_Daughters_In_Family	stats	numerical	New binning	Build index
Cancer_Relative_History				
12. Num_Prim_Relative_Breast_Cancer_Female	stats	numerical	New binning	Build index
13. Num_Prim_Relative_Breast_Cancer_Male	stats	numerical	New binning	Build index

Figure 6.22: The Binning page.

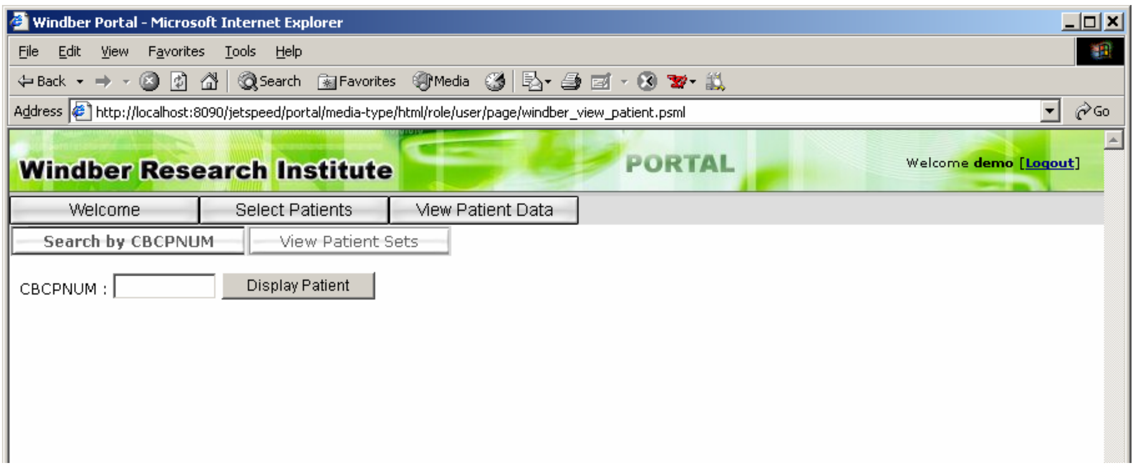
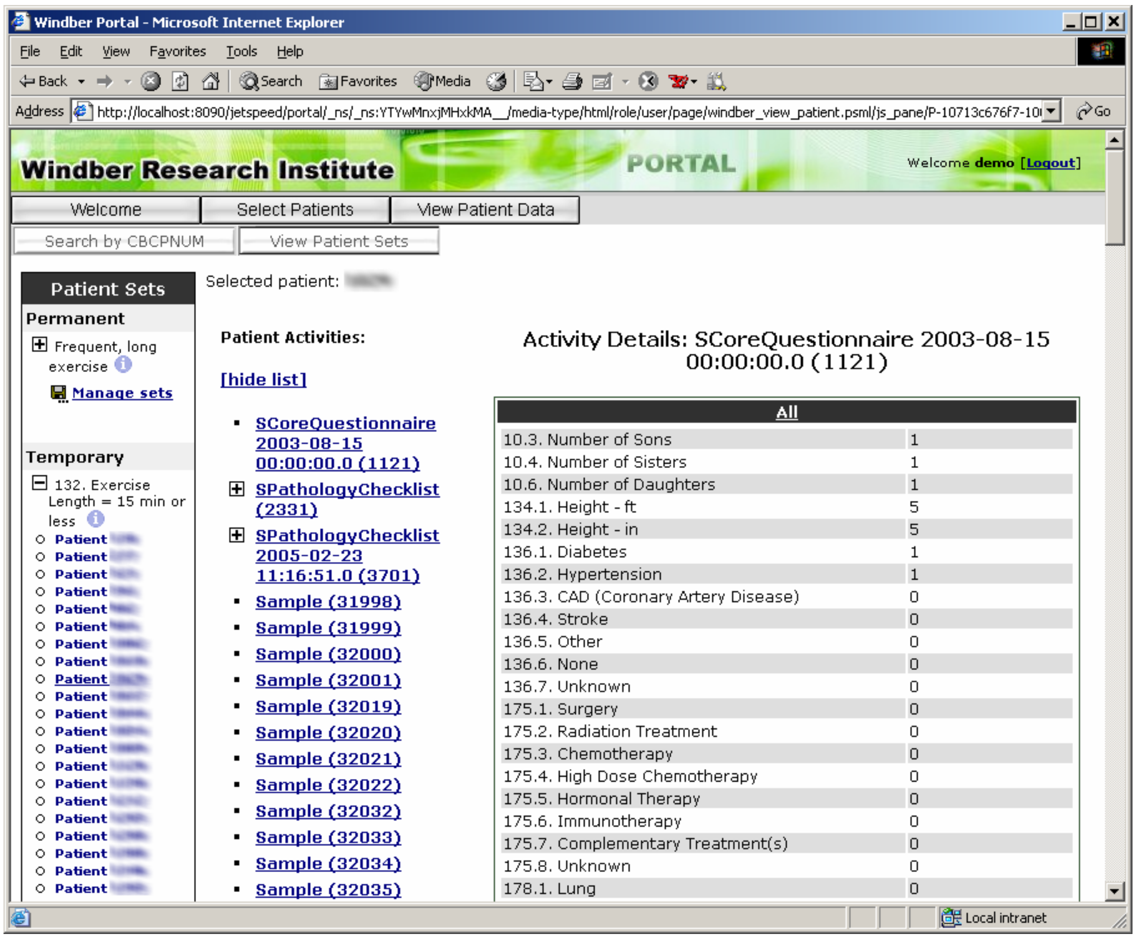

Windber Portal - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8090/jetspeed/portal/media-type/html/role/user/page/windber_view_patient.psml
Go
Windber Research Institute
PORTAL
Welcome demo [Logout]
Welcome
Select Patients
View Patient Data
Search by CBCPNUM
View Patient Sets
CBCPNUM :
Display Patient

Figure 6.23: A Patient Viewer page, allowing users to specify a patient ID to view.


Windber Portal - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8090/jetspeed/portal/_ns/_ns:YTYwMnxjMHxkMA__/media-type/html/role/user/page/windber_view_patient.psml/js_pane/P-10713c676f7-10
Go
Windber Research Institute
PORTAL
Welcome demo [Logout]
Welcome
Select Patients
View Patient Data
Search by CBCPNUM
View Patient Sets
Local intranet

Patient Sets

Permanent

⊞ Frequent, long exercise

Manage sets

Temporary

⊟ 132. Exercise Length = 15 min or less

- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]
- Patient [redacted]

Selected patient: [redacted]

Patient Activities:

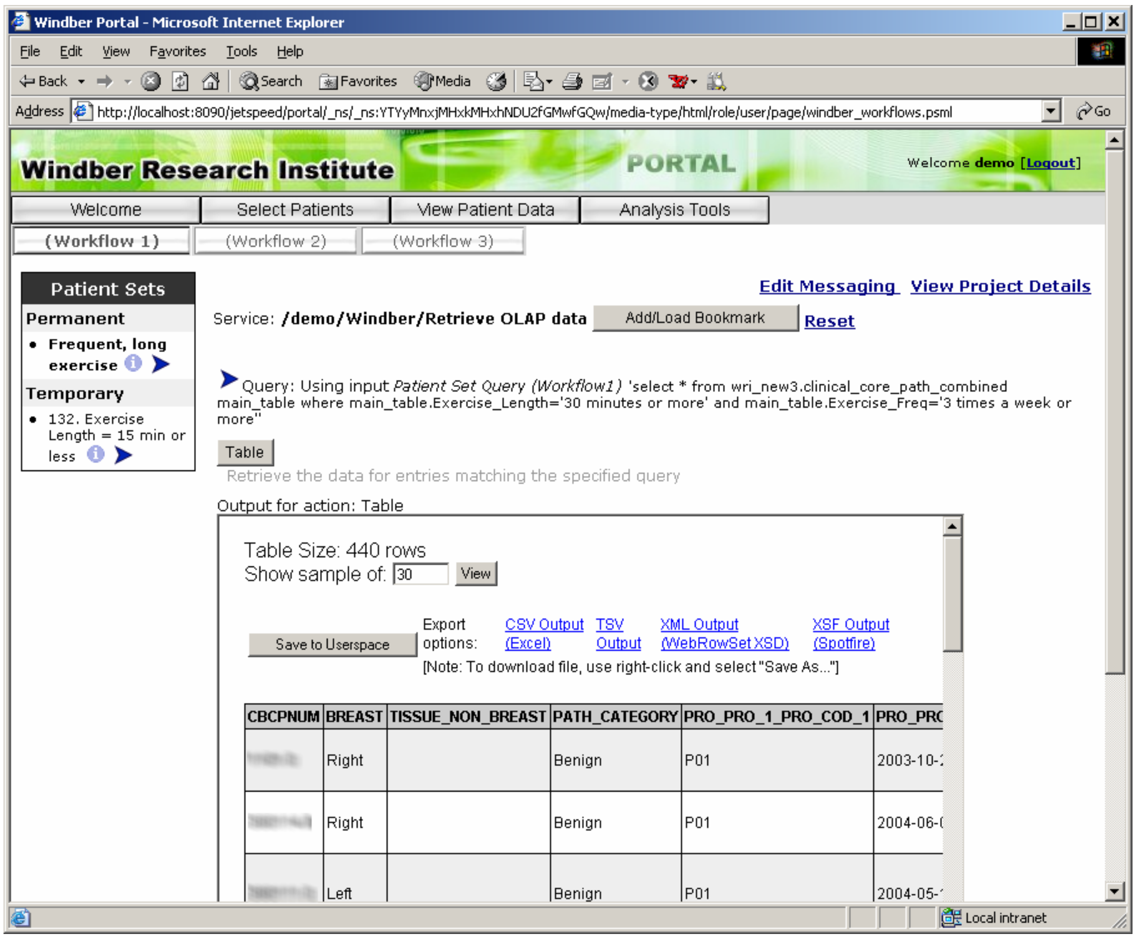
[hide list]

- SCoreQuestionnaire 2003-08-15 00:00:00.0 (1121)
- ⊞ SPathologyChecklist (2331)
- ⊞ SPathologyChecklist 2005-02-23 11:16:51.0 (3701)
- Sample (31998)
- Sample (31999)
- Sample (32000)
- Sample (32001)
- Sample (32019)
- Sample (32020)
- Sample (32021)
- Sample (32022)
- Sample (32032)
- Sample (32033)
- Sample (32034)
- Sample (32035)

Activity Details: SCoreQuestionnaire 2003-08-15 00:00:00.0 (1121)

All	
10.3. Number of Sons	1
10.4. Number of Sisters	1
10.6. Number of Daughters	1
134.1. Height - ft	5
134.2. Height - in	5
136.1. Diabetes	1
136.2. Hypertension	1
136.3. CAD (Coronary Artery Disease)	0
136.4. Stroke	0
136.5. Other	0
136.6. None	0
136.7. Unknown	0
175.1. Surgery	0
175.2. Radiation Treatment	0
175.3. Chemotherapy	0
175.4. High Dose Chemotherapy	0
175.5. Hormonal Therapy	0
175.6. Immunotherapy	0
175.7. Complementary Treatment(s)	0
175.8. Unknown	0
178.1. Lung	0

Figure 6.24: A Patient Viewer page. The Patient Sets portlet allows each patient set to be expanded to show the patients contained within. These patients may then be selected and shown in the Patient Viewer portlet on the same page.


Windber Portal - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8090/jetspeed/portal/_ns/_ns:YTYyMnxjMHxkMHxhNDU2fGMwfGQw/media-type/html/role/user/page/windber_workflows.psml
Go
Windber Research Institute
PORTAL
Welcome demo [Logout]
Welcome
Select Patients
View Patient Data
Analysis Tools
(Workflow 1)
(Workflow 2)
(Workflow 3)
Local intranet

Patient Sets

Permanent

- Frequent, long exercise

Temporary

- 132. Exercise Length = 15 min or less

Edit Messaging View Project Details

Service: **/demo/Windber/Retrieve OLAP data** Add/Load Bookmark Reset

Query: Using input *Patient Set Query (Workflow1)* 'select * from wri_new3.clinical_core_path_combined main_table where main_table.Exercise_Length='30 minutes or more' and main_table.Exercise_Freq='3 times a week or more''

Table

Retrieve the data for entries matching the specified query

Output for action: Table

Table Size: 440 rows

Show sample of: 30 View

Save to Userspace

Export options: CSV Output (Excel) TSV Output XML Output (WebRowSet XSD) XSF Output (Spotfire)

[Note: To download file, use right-click and select "Save As..."]

CBCPNUM	BREAST	TISSUE_NON_BREAST	PATH_CATEGORY	PRO_PRO_1_PRO_COD_1	PRO_PRC
[redacted]	Right		Benign	P01	2003-10-
[redacted]	Right		Benign	P01	2004-06-
[redacted]	Left		Benign	P01	2004-05-

Figure 6.25: A workflow page. The Patient Sets portlet allows each patient set to be selected, for use as input to the workflow.

6.6 Conclusion

The concept of an analytical portal was entirely appropriate for the needs of Windber's researchers, and was so successful that it extended the project well beyond its original conception, which was 'simply' a rebuild of their existing data warehouse.

The use of the Jetspeed portal with both customised Windber and Discovery Net portlets supports multiple stages of the analysis procedure. Researchers can go from initial exploration of warehoused data (Figure 6.21), to selection of interesting subsets of patients, to analysis of those patient sets to perform new research (Figure 6.25) - all within the same Portal environment. The Discovery Net Java Client need only be used for initial creation and editing of the analysis workflows.

The integral portal services for page management and security also proved useful: first in the rapid development of the custom pages making up the Translational Medicine portal, and then in restricting access permissions on certain portlets and pages, so that external research partners could be given access to the OLAP browser, but not to the confidential individual patient information.

This project also demonstrated the usefulness of the IPC library (Chapter 5), in first developing the communicating Windber portlets, and then allowing the new Patient Sets portlet to provide input to Discovery Net Service portlets. The shared message bus provided by the library allowed the portlets to be easily developed as independent but interoperating components.

In summary, the portal/IPC approach allowed us to provide:

1. A user friendly web-based research solution
2. Independent development of portlets customised to the client’s needs
3. Easy integration through IPC with Discovery Net services in the same portal
4. Rapid development of portal pages

However there was one aspect relating to the custom portlet implementation which was less than ideal: the method of integrating a rich AJAX-based web application within a portlet. We approached this by embedding the complex, non-portlet application within an IFRAME – however this has the effect of framing the application within a fixed size box on the page, with its own scrollbars. This was not popular with the users, but the alternative of developing a rich web application completely within a JSR-168 portlet was problematic from a development point of view. The limitations of using AJAX within JSR-168 portlets are discussed in detail in the next chapter.

Work is expected to continue on the Translational Medicine portal for some time, to customise it with interfaces specific for patient data (e.g. using timelines as a way of visualising information over a patient's lifetime), and expand the current tools to allow appropriate treatment of data with a time component. The current generic OLAP browsing technology will also be examined for use by other customers in other application areas.

Chapter 7. Limitations and Improvements for JSR-168 portlet technology

Developing portlets to the JSR-168 portlet standard[16] is a good practice as it widens the choice of portals to which they can be deployed, avoiding vendor lock-in. However there are several limitations involved in developing JSR-168 portlets. Many of these issues have workarounds (of varying effectiveness). We discuss these limitations and their workarounds in detail in this chapter.

The next version of the portlet specification (JSR-286[106]) is currently in progress. The new standard will address many of the features lacking from the original specification, and hopefully also some of the additional limitations described below.

As a result of the research detailed in this chapter, we have summarised the most important issues and submitted a list of topics and potential improvements for consideration to the JSR-286 expert group (Section 7.4). The closely related WSRP 2.0 specification is also in development, and any new features will be coordinated between both specifications, whenever relevant.

In this chapter, references to specific parts of the Portlet Standard JSR-168 and the Servlet Standard[13] will be prefixed by 'PLT.' and 'SRV.' respectively.

7.1 Context

When considering the limitations of JSR-168 portlets, we must first explain what we are comparing it against.

WSRP describes a language-independent method of accessing remote portlets. However in practice, the vast majority of portlet implementations have been in Java, using the JSR-168 specification. Two exceptions are "Go-Geo!"[93] using Perl, and NetUnity[24] using .Net.

In comparison to other, non-JSR-168 portals, the limitations would therefore focus mainly upon those features left out of JSR-168 and the differences between Java and the technology (Perl, PHP etc) used by the other portals. There are numerous different web technologies and frameworks, and a full comparison is well beyond the scope of this report. The final choice of web technology is often made based upon the developers' experience and preference for the underlying framework or language used, or mandated by existing infrastructure, both of which are entirely practical, but not considerations that can be addressed as part of a Java specification.

We will therefore consider JSR-168 primarily in the context of Java web server technologies, the core of which is J2EE[11] and Java Servlets[13] but which also includes non-JSR-168 Java portals. JSR-168 portlets are very similar to servlets, but differ in numerous ways[173] which are described in detail in Section 2.6.1.3. Many of these differences are what lead to the problems and limitations in the way portlets can be used - particularly from the point of view of a servlet developer migrating to use portlets.

7.1.1 Other Java Portals

Non-JSR-168 Java portals based upon J2EE, such as earlier versions of Oracle Portal[66], IBM WebSphere[46], and Apache Jetspeed 1[28], are the precursors to JSR-168 Portals and each provide their own version of 'portlets' and accompanying portal services. These portals are still in use by many sites, and most now support JSR-168 in their latest versions, often in parallel with their own legacy portlet systems.

The portal-specific portlets often boast a wider array of features than those provided in JSR-168 - for example, support for developing MVC[3] portlets (often with Apache Struts), inter-portlet communication, portlet filters (discussed in detail later), and direct use of portal services such as page layout. Use of these features is often very desirable to portlet developers, but such portlets can only be hosted in the chosen portal implementation and are not JSR-168-compliant. Although using portal-specific features directly will always be more efficient and powerful, it is possible to get some of the advantages of these advanced features without sacrificing JSR-168-compliance and being locked down to using a particular portal. Where relevant, these workarounds will be discussed below.

7.2 Limitations in Portal Design

There are some fundamental practical considerations in the design of portal pages and portlet contents that are not specific to JSR-168, but apply to any Portal-like system. Thus at an early stage of development it must be decided whether or not the aims of a site are best met by the portal philosophy.

First is the concept of modularity in a website, and whether this is suited to the content of a particular site. Modularity is clearly beneficial from the point of view of code maintenance; however it may not be optimal for the site designers or end-users, as it discourages direct cross-linking between modules. Such cross-linking between sections can add significantly to the usability of the site, and is a convenience expected by end users. Therefore, despite code modularisation, linking between sections is common and often must be allowed for, usually by avoiding changes to document locations and URL query formats whenever possible. This is manageable in sites where modules are loosely-coupled and do not require extensive cross-linking - such sites will translate well to a portal, as it should be quite easy to split up the functions into mostly-independent portlets. On the other hand, if a site is tightly focused, and most of the pages are closely coupled, it may not be necessary or worth the effort to use a Portal.

The next most significant design consideration in developing a portal site concerns the core feature of allowing multiple portlets to be present on a single page. This means firstly that all these portlets must be able to fit in the limited space available, and secondly, that the page will take longer to generate and display than if it was only showing one portlet at a time. Every time the user interacts with a portlet on the page to submit a form or follow a link, the whole page must be re-rendered - this will probably result in all the render code for all the portlets re-executing, even when the user has only actually interacted with one of them. The Portal may support caching of portlet views to improve performance (a portlet's cache 'expiration-time' may be set in the configuration `portlet.xml`) but this support is not compulsory.

These issues must be taken into account by both the page designer (when choosing which and how many portlets to add to a page), and by the portlet developer. The code which renders the portlet should be idempotent and as lightweight as possible to speed up page loading, and portlet content must be compact if the portlet is to share space with other portlets. In addition, portlet and page designers must be particularly careful to avoid overloading and confusing the user with too much complexity.

7.3 Limitations in Portlet Implementation

Having decided that the generic portal concept fits a website's aims, the choice of portal technology will be dependent on the capabilities of the portal software and the developers' skill set.

Here we discuss in detail the considerations and limitations involved in the development of JSR-168 portlets, many of which will be encountered in the development of any moderately complex portlets, and particularly when considering conversions of existing servlet-based sites. Where relevant, we also include workarounds and our solutions. These may be considered to be in addition to the excellent "Best Practices" document by Stefan Hepper[173] which outlines the basic differences in developing portlets compared to servlets.

7.3.1 Java-only

Although we will not go into a comparison between different languages available for server-side page generation, it should be noted that the language restriction will naturally make migrating an existing non-Java website to use JSR-168 portlets considerably more time-consuming.

Workaround

There are some bridges available, notably Apache Portals Bridges[31], which allow an existing non-Java application to be wrapped as a JSR-168 portlet. This can wrap PHP and Perl applications and other native programs (e.g. MapServer[55]). Such bridges are very valuable for exposing existing and legacy applications, but may not permit full usage of portlet features (depending on the particular implementation).

Alternatively, for a non-JSR-168 but still standards-compliant approach, the portlets could be developed using any language and then exposed as WSRP[14] (Web Services for Remote Portlets) web services. Many portals now support WSRP portlets as well as or instead of JSR-168.

7.3.2 Missing Portal Features

The JSR-168 specification (Section PLT.E) notes some features as missing, and to be considered in a later specification:

- *Portlet filters*
- *Inter-portlet, event style, communication*
- *Allow portlets to produce and influence markup outside of the portlet fragment*

These features and others are discussed in this section.

7.3.2.1 Portlet Filters

The Servlet API[13] includes support for Filters (Section SRV.6), which wrap the processing of an incoming request. A filter can inspect the request and modify the response returned to the client, and can therefore be used for a multitude of tasks from access-checking to transformation of the response content.

Filters are configured in the webapp's `web.xml` configuration file, and can be mapped to individual servlets or particular URL patterns covering many servlets and/or other files. It is then the servlet container's responsibility to apply the filters to any incoming request. Multiple filters can be applied to a single request - for example, a request might first be passed through a filter which replaces special markup in the response with HTML, and then a second filter which compresses the response before transmission.

Counterintuitively, servlet filters registered in a portlet application's `web.xml` (see Figure 2.8 for example directory structure) are not applied to portlets in that application when they are rendered by a portal. Filters are only applied to resources served by the servlet container to the client browser from the same webapp (e.g. `http://myserver.com/myportletapp/page.jsp` would be seen by the servlet container to be in the `myportletapp` webapp); this never happens with portlets.

Instead, the client requests pages from the portal’s webapp, and the portal invisibly delegates to portlet classes when it needs them to render part of a page (e.g. `http://myserver.com/portal/index.jsp?pageid=3` might be the form of the URL visited by the browser, and served by the servlet container from the `portal` webapp). Thus only filters defined in the portal's webapp would be applied to the portlets in a portal page.

JSR-168 mentions portlet filters as a feature which will be defined in the next specification. The functionality provided by filters can currently be duplicated by other methods, such as having portlets inherit from a parent class with the required common features, or the workarounds below.

Workaround

The Apache Portals Bridges project includes a generic, JSR-168-compliant, Filter Portlet, which can provide similar functionality to servlet filters. The Filter portlet acts as a wrapper around the real portlet. This is done by registering the Filter portlet in the `portlet.xml`, and parameterising it with the name of the 'real' portlet class.

Another option is to use aspect-oriented programming[138] to intercept portlet methods and wrap them, which would be very similar to filters but differ in the method of configuration.

Portlet applications using either approach should be relatively easy to modify to support real portlet filters when it is time to upgrade to the next standard, as both approaches keep the filter code completely separate from the portlet code.

7.3.2.2 Inter-portlet Communication (IPC)

JSR-168 leaves the method of communication between portlets to be defined in the next specification, so there is currently no official API through which portlets can exchange messages. Communication is still possible, by using shared session attributes visible to all portlets in the same application, or by using external message stores such as databases. However the detailed implementation of such communication is left up to the portlet developer[158].

Conversely, many Portals make their own detailed provisions for IPC, using a variety of models and configuration methods (see Section 5.1 for examples), and have done so since before JSR-168 was published. The use of such communication services saves considerable development time, but the portlets developed are then portal-specific.

Workaround

As a stopgap measure before the next portlet specification is finalised and implemented by portals, we have implemented a library for portlet messaging which works within the limitations of JSR-168. This enables developers to write communicating portlets easily without needing to implement the messaging system as well, thus partially levelling the field between JSR-168 and portal-specific messaging. Of course, an arbitrary third-party portlet will still not be able to participate in such communication without being rewritten to use this messaging library - but that level of compatibility must wait for an official IPC API in the next specification.

A more detailed description of existing portlet communication systems and the design of the messaging library is given in Chapter 5.

7.3.2.3 Access to Portal Services

Services such as management of pages, portlets, users, roles and access permissions are provided by the Portal hosting the portlets. With JSR-168, there is no way for a portlet to access these services: it cannot inspect its page environment or other portlets, and security is limited to the `PortletRequest` functions `getUserPrincipal/ getRemoteUser` and `isUserInRole`.

As a result there are a number of things a JSR-168 portlet is unable to do, including:

- Modify page layouts, e.g. add a new portlet to a page
- Add or remove portal pages
- Manage users and roles
- Set page or portlet access permissions
- Find out what other pages there are on the portal, and link to them
- Find out what other portlets exist on a page, or on the whole site.

One example where such services would be useful is to allow portlets to generate links to other pages or portlets. If the portlet were able to retrieve a list of available portal pages at runtime, it would be able to present a richer configuration interface to the user. For example, one portlet might redirect to another portal page as part of a longer process involving many portlets, and the administrator or even the users might be allowed to specify which page should be next in sequence.

Portlets might also be able to use information about what other specific portlets were present on portal pages. For example, a portlet might print links to related portlets, if it detected their presence. Or a company might distribute a collection of portlets, but be unable to guarantee which exact portlets were present in any given installation: e.g. a particular forum database might need to be set up, or a particular server-side visualisation or analysis package installed and paid for.

Management and inspection of pages, portlets and users/roles programmatically in portlet code can currently only be done using portal-specific extensions; normally, it is done using the portal's own management tools. While many of these functions would be very useful in some scenarios, a question would remain over how much power a portlet should be given to actually modify its own hosting environment (read-only inspection would be less of a risk, but still perhaps a security concern). Even if access to such functions were to be provided in a later portlet specification, it would be likely that final control would remain with the portal administrator, who would be able to configure the permissions granted to portlets. In addition, different portals currently use different models for managing pages and users; it may not be appropriate or desirable for the portlet specification to specify in detail the approach taken by portals in implementing these services, but this might become necessary if a standardised way for portlets to access the services were provided.

7.3.2.4 Window ID

The portlet window is a particular instance of a portlet on a portal page (Figure 2.5). There is some variance in the exact interpretation (discussed later in this chapter), but generally the portlet window is considered as an independent module that maintains its own state separately from all the other portlet windows in the site.

Knowing the portlet window ID is critical when developing JSR-168 portlets as communicating components. This is particularly relevant when there are multiple portlet windows based upon the same portlet entity (e.g. multiple Weather portlets, each set for a different city). Preferably, this ID would stay the same across browser sessions. The Portal itself will already have assigned an internal ID for each portlet window, so that it can store and retrieve portlet window preferences etc., but JSR-168 provides no simple way to access this ID from within the portlet code. In some situations, the method `RenderResponse.getNamespace()` may be a suitable alternative, but this only provides a namespace for the portlet window which is unique within the current page (PLT.12.3.4); it is not guaranteed to be unique across the whole portal, or even to be consistent across different requests in the same session.

Window IDs can also be used to permit easier use of external data stores - e.g. data that is too large to go in the portlet session/preferences, or too complex to store as a String in a preference. Another example is in a "Message of the Day" or generic "Content" portlet, which ideally could be edited by users with administrative privileges, but whose content would be visible to all users. For example there might be a Content portlet on every page of a fixed portal layout, which only admins could edit. In this particular use case, the content cannot be stored in JSR-168 preferences, as such preferences are user-specific and so only the admin entering the content would see the changes. Thus, to implement this "shared preferences" behaviour, the content must be stored in an external data store. With an arbitrary number of Content portlets (e.g. being added to new portal pages), the Window ID (common for all users) is an ideal key to use to store the content in the external data store. An alternative, not using Window IDs, would be to add separate portlet entities in `portlet.xml` for each Content portlet, including an *initialisation parameter* in each entry which is the key for the content in the remote store. However this is a far more cumbersome approach as it requires changes on the server for every new Content portlet added to a page.

A further use of a Window ID is in namespacing cookies, so that cookies stored (by JavaScript) by one portlet window do not interfere with other windows showing the same portlet entity on other pages. This is particularly important when AJAX[27] is being used to create a user interface whose client-side state must be persisted in a cookie. `RenderResponse.getNamespace()` may be usable here instead, but there is no guarantee that it will provide a namespace unique across the whole portal.

<u>*Workaround*</u>

It is possible to generate an arbitrary unique ID for a portlet window, and store it in its local session when the window is first rendered; this is quite easy to implement (Appendix A1). A per-session ID can be guaranteed to be unique for the portlet application by registering and checking it against a list of assigned IDs in the APPLICATION_SCOPE session. If the ID must persist across multiple sessions, it can instead be generated once and stored in the user's preferences for that portlet window - however, this approach opens up the possibility of ID clashes with portlets which have not yet been initialised in that session, and so an official method of retrieving the ID would be preferable.

Another, more reliable, workaround makes use of the portlet window ID which is used by the portal to namespace PORTLET_SCOPE session attributes, as described in PLT.15.3. Assuming that the portlet is able to inspect the APPLICATION_SCOPE session and access the full name of namespaced session attributes (encoded as "javax.portlet.p.<ID>?<ATTRIBUTE_NAME>"), it would be able to retrieve the portal's unique ID for the portlet window by deconstructing that full attribute name. Example code for doing this is provided in Appendix A1.

7.3.2.5 Session ID

When a user visits a portal with a web browser, it is common for the server to start up a server-side 'session' for that user. With its page response, the server sends back a unique session ID to the browser, which the browser will then include with all subsequent requests. The server can thus use session state to persist information across requests. When the browser is closed, the session is effectively lost as the user will have no way of re-initialising the browser with the same session ID, and eventually the session state on the server will expire and be cleaned up from memory.

Following the servlet specification, each web application has its own separately maintained session. This is done to keep web applications from accessing or overwriting other applications' private session data. As discussed and illustrated in Figure 2.7, the portal and each portlet application are all web applications, and each portlet application has its own session (even if in practice the portlet sessions are 'virtual', within the portal session).

A portlet can retrieve the unique ID of its portlet application session with `PortletSession.getId()`. It is entirely possible that portlets in different portlet applications will see different portlet session IDs. The portal will also have its own session ID. This section is concerned with the retrieval of this ID by portlet code.

There are several use cases where a portlet may need to know the portal's session ID - these are typically when related portlets are deployed in different portlet applications (e.g. due to a modular distribution/licensing scheme).

Firstly, when portlets in different applications need to communicate, they need to find an ID to represent the current user session which they are all part of. The ideal identifier is simply the portal's own session ID. This ID can then be used to store and retrieve messages in an external message store accessible by all portlets.

Secondly, the session ID would also allow back-end resources (used by portlets in different applications) to track proper user sessions, so they could enforce licensing limits, enable clustering, or simply perform logging.

Workaround

The JSR-168 method `PortletRequest.getRequestedSessionId()` can very nearly provide the needed functionality, as it returns the session ID included in the client (browser) request, which corresponds to the Portal session. However, unlike `HttpSession.getId()` or `PortletSession.getId()`, this approach does not work with newly-created sessions - i.e. the very first page rendered by the server, before the client has been informed of the session ID. This can be inconvenient if as part of session creation the portlet needs to set up external resources reliant on the portal session ID (such as external message stores). Thus to avoid this problem, any portlets relying on this method should not be placed on the first portal page.

Apart from using `PortletRequest.getRequestedSessionId()`, two approaches have been investigated: setting an arbitrary session ID in a cookie visible to all portlets (which also has the first-page problem, and is quite unreliable), or modifying open source portals to pass on the Portal's session ID in the PortletRequest. Neither of these are ideal: the cookie approach is more fragile and thus inferior to using `getRequestedSessionId()`, and although the portal-modification gives a better end result, it requires access to portal code and also restricts the portlets to using that specific modified portal. We believe that `PortletRequest.getRequestedSessionId()` is currently the best approach, but the problem of its behaviour with new sessions is important enough to be highlighted.

7.3.2.6 Cookies

Cookies are small pieces of information stored on the client machine by a web browser as name-value pairs, used for client-side tracking of state. They can be set either by a response header from the server, or using JavaScript. When making a request to a web server, the browser includes any cookies set by that site as a header in its request.

Cookies are used for storing state information relevant to the browser session. They are particularly useful if AJAX[27] is used to modify the client-side state of a portlet. AJAX applications work by using JavaScript to send off browser requests invisibly in the background, to submit and/or retrieve data from web pages. Portals provide no AJAX-addressable endpoints to allow direct interaction with portlet windows, and so there is no convenient way for a JavaScript request to update or read the portlet state (e.g. read from or save information to a portlet window's local session). Thus portlets using AJAX may have to store state information in client-side cookies, instead of the server-side portlet session.

Cookies can also be used to communicate information (e.g. login information, IDs of working data sets) between portlets and other, non-portal, web applications on the same server - which themselves might use any technology (J2EE, PHP etc). An external data store, accessible to both applications, could alternatively be used to store this information - however cookies are simpler to implement, for small amounts of data, and do not require maintenance.

Reading Cookies

Servlets can access client cookies using the method `HttpServletRequest.getCookies()`. There is no direct equivalent for this in the `PortletRequest`, however on some Portals the value of the cookie Header can be accessed using `PortletRequest.getProperty("cookie")`. This approach is not guaranteed to work reliably on all portals. According to PLT.11.1.4, this is due to differences in portals and servlet containers which may make the headers unavailable to portlets. JSR-168 does support some headers (such as `Content-Length` and `Content-Type`) with specific access methods in the `PortletRequest` interface, but unfortunately this was not done for cookies.

Hopefully the retrieval of cookies will be addressed in a later specification.

Workaround

Some portals will allow the cookie to be retrieved using `PortletRequest.getProperty("cookie")`, but this cannot be relied upon.

Setting Cookies

Servlets can set cookies by including a special header in the response. This is done on the `HttpServletResponse` using either `addCookie` or `setHeader`. Neither is available to portlets on the `PortletResponse` or its subclasses.

A portlet cannot be expected to be able to set response headers during the render phase, as it can only affect its rendered page fragment (and the beginning of the page response, where the headers are included, may already have been sent to the client by the time the portlet render code executes). However, it should be theoretically possible for a portlet to set a cookie on an `ActionResponse` in the action phase (which occurs before any rendering takes place), as it is already possible to send a 'redirect' response with a `Location` header during this phase, using `ActionResponse.sendRedirect()`.

Hopefully a later specification will permit cookies to be set on an `ActionResponse`.

<u>*Workaround*</u>

Without support in JSR-168 for setting cookies as part of the response, it is necessary to resort to approaches which are more clumsy and fragile. One option is to set a cookie using JavaScript, which can be done by having the portlet generate the JavaScript as part of its rendered display - however this relies on JavaScript being enabled on the client. Another option is to add a servlet to the portlet application which sets a cookie when it is accessed. This servlet must be accessed through a direct URL by the client, not dispatched to or included by the portal, and so it must be loaded through an IFRAME or a popup window generated as part of the portlet display. Both approaches (JavaScript & servlet) have the disadvantage that the cookie is set only as a result of the client viewing an already-rendered portal page. It is also possible that the cookie will never be set, if the user cancels the page or visits another before the relevant portlet finishes displaying (e.g. on a long or slow-loading portal page).

7.3.2.7 'Exclusive' display mode

Portlets generate only a fragment of the eventual response sent to the client. Therefore, if a portlet needs to serve binary content (images, PDFs etc) to a client, it cannot directly include these into the HTML stream. Although some browsers permit binary objects to be embedded in HTML, support for this is not yet widespread.

To provide a file for download, a portlet must cause the client to make a new request - either directly to the file, or to a servlet which retrieves it. This is done by either rendering a link to it, including it in an IFRAME, or opening it in a new window. However, problems related to the treatment of cross-context sessions may be encountered when implementing these approaches on some Portals (discussed below). This means that any portlet state which is relevant to the servlet (e.g. current user, file to show, data to be rendered as a chart) needs to be explicitly passed to it, either through query parameters (in the servlet URL), cookies, or through an external data store. The latter involves extra complexity, and the first two methods send all the data to the client browser, then back again to the servlet. Apart from the inefficiency and extra coding required, if there is a large amount of data to be sent - e.g. to a servlet that generates a chart or other visualisation based on the data - this probably won't even be practical.

As an alternative, some Portals (e.g. uPortal[83]) have chosen to make available an 'Exclusive' mode in which a portlet generates the entire response, not just a fragment. This makes any problems with cross-context sessions irrelevant. This approach is portal-specific but quite popular, and may also have relevance when considering the use of AJAX in portlets (discussed below).

7.3.2.8 Remote (client) IP

Servlets can find out the IP address of the client which sent the request, through `ServletRequest.getRemoteAddr()` or `getRemoteHost()`. This can be useful for limiting access by IP address or logging IPs associated with actions for administrative or security purposes.

However, portlets are currently insulated from finding out information about their actual end user - perhaps to emphasise and simplify the portal's role as intermediary. The `PortletRequest` classes do not provide an equivalent of these methods, and it is therefore not possible for a portlet to directly retrieve the client's IP address from the request as a servlet can. Nor can this be worked around by causing the portlet to dispatch internally to a servlet, using a `PortletRequestDispatcher` - JSR-168 specifically prohibits the `ServletRequest` methods concerned from working in this context (PLT.16.3.3).

Workaround

We have not found a good workaround for this issue. One possibility is to use a combination of servlets and cookies as described earlier, which would require support for reading cookies in portlets, an additional servlet in the portlet application (which retrieves the IP address and saves it in a cookie), and at least one page render before the portlets would have access to the client IP address.

7.3.3 Further Issues

When developing portlets, there are also a number of areas of possible confusion (rather than missing features), often due to differences between portal implementations.

7.3.3.1 Portlet Windows/Instances

Portlets are defined with entries in the `portlet.xml` deployment descriptor. PLT.5.1 specifies that only one actual instance of each portlet so defined will be created by the portlet container (or one per VM in the case of a distributed application). This single instance will be used to process any Action or Render requests targeted to that portlet.

However, the same portlet - with only one definition in the `portlet.xml` - can be placed in more than one position on a site, and often it is necessary for these different portlet windows to operate independently. For example, a 'Stocks' portlet might be configured by the user to show a report for different stocks on different pages; a portlet which could be pointed at any WSDL file to generate an interface to a web service might also be present on different pages, providing access to different web services; a content management portlet would need to display different content on different pages. Conversely, a “shopping basket” portlet which appears on many pages should always act as the same portlet.

JSR-168 defines a *portlet window* as the combination of a portlet and its preferences on a portal page. It also states, "A portal page may contain more than one portlet window that references the same portlet and preferences-object." (PLT.5.2.3). The details of creating and managing portlet windows are left to the portal server and portlet container.

This leaves several aspects of the concept of a 'portlet window' undefined or ambiguous, including:

1) Do different portlet windows also maintain their own, separate, PORTLET_SCOPE sessions and render states (from Action requests) as well as preferences?
2) Can there be different portlet windows of the same portlet which are associated with different preferences and session objects?
    a. on the same page?
    b. on different pages?

The answer to the first question is usually assumed to be 'yes', but the second is more variable. Different portlet containers and portals are implemented using different assumptions.

The strictest approach, taken by Pluto[30] and GridSphere[78], is to assume a simple mapping of portlet definition (in the `portlet.xml`) to portlet window: adding a second portlet window for the same portlet will merely result in a mirroring of content from the first. In this case, if multiple independent portlet windows are required, the portal administrator must add multiple definitions in the `portlet.xml` for the 'same' portlet (merely differing in portlet name).

Perhaps the most intuitive approach is to treat every portlet window on a page as an independent entity, with its own set of preferences, PORTLET_SCOPE session, and render state. This allows for an unlimited number of portlet windows associated with a single portlet definition, and is particularly well-suited to sites which give users the freedom to add portlets to their pages. This is the approach taken by Apache Jetspeed and Oracle Portal.

Intermediate approaches are also possible. For example, a portal might allow portlet windows from the same portlet definition to exist independently on different pages, but not on the same page (we encountered this behaviour in Liferay[52]).

JBoss Portal[51] adds an additional level of abstraction between the portlet definitions and the portlet windows: it allows a "portlet instance" to be dynamically created at runtime, based upon a portlet definition[105]. Each JBoss portlet instance has a modifiable set of associated preferences, and may be used as the base for one or more portlet windows. This approach is the most flexible, allowing both independent and linked portlet windows based on the same portlet definition. To create multiple independent portlet windows, each window must be based on a new portlet instance of a single portlet definition. To create identical/linked portlet windows on different pages - for example, a chat portlet that should retain user settings and state across pages - a single portlet instance can be used as the common base. The disadvantage of this approach is the increased complexity in adding new portlet windows to pages, which must now include steps for portlet instance creation and association.

Differences in portal behaviour on this subject do not significantly affect the process of developing portlets. They do affect the way the portlet deployment descriptor is written (if extra entries are necessary to produce independent portlet windows), and the way site designers and users add portlets to pages. For dynamic sites which allow users to create pages, the strict interpretation may not be sufficiently flexible, depending on the expected level of portlet reuse.

7.3.3.2 The `PortletSession` and the `HttpSession`: Cross-context handling

As discussed in Section 2.6.1.3, a portlet application is a web application which additionally contains portlet classes and a portlet deployment descriptor file listing the portlets. It may also contain anything else that can be in a normal web application, including arbitrary files, JSPs and servlets. These can be accessed and served as normal by the servlet container.

According to JSR-168 (PLT.15.4):

> "The `PortletSession` must store all attributes in the `HttpSession` of the portlet application. A direct consequence of this is that data stored in the `HttpSession` by servlets or JSPs is accessible to portlets through the `PortletSession` in the portlet application scope. Conversely, data stored by portlets in the `PortletSession` in the portlet application scope is accessible to servlets and JSPs through the `HttpSession`."

This behaviour is also described in PLT.3.1: "Bridging from Portlets to Servlets/JSPs".

This is clearly intended to allow portlets and servlets in the same portlet application to easily work together, communicating through the shared session. The session may be used to send data (for example if the portlet wishes to delegate to a servlet to generate a binary file based on that data, such as a graph image or a PDF), or details of the logged-in user (which the servlet may use for authorisation).

However, there are implementation issues which mean that this behaviour is not found in some servlet container/portal setups (notably the Apache Tomcat[33] servlet container). The following description is based upon our experiences and that of many others[136,148,178].

As we have mentioned earlier, the Servlet 2.3 specification requires that the server maintains a separate session for each webapp (SRV.7.3). As the portal and the portlet applications are all web applications, they should each have their own session. This is compatible with the JSR-168 description: the servlets within a portlet application could be reasonably expected to share the same session with portlets in that application.

However, the way that a portal serves pages to the client browser makes the matter more complex. When visiting a portal page, the browser will typically access an URL along the lines of `http://server.com/portal/display?pageid=3`. The resulting page may include portlets from any portlet application on that server, but there is no indication of the source portlet application in the URL. As we discussed earlier (7.3.2.1), to the servlet container, the request is only to the *portal* web application, and the corresponding session is the *portal* application's session. Thus, many portal implementations actually store both PORTLET_SCOPE and APPLICATION_SCOPE portlet sessions within the portal's own session, not in the session of the parent portlet application (illustrated in Figure 2.7).

Consequently, when a servlet or any other portlet application resource is accessed directly by the browser, with the form `http://server.com/myportletapp/getfile.jsp` (e.g. in an IFRAME or popup window), and the servlet container uses the session belonging to the specified portlet application, this session does not contain any of the APPLICATION_SCOPE or PORTLET_SCOPE session for that application's portlets. In this case, the only way the portlet can send information to the servlet is by appending URL query parameters (e.g. `http://server.com/myportletapp/`

`getfile.jsp?name=report.doc`), which due to limited URL length may not be appropriate or possible for large amounts of data.

If on the other hand the servlet is included as part of a portlet's output through the portal, *using JSR-168 mechanisms* (in a portlet render method, `getPortletContext().getRequestDispatcher(jspPath).include(request, response)`, or in a JSP included by the portlet, `<jsp:include>`), the session behaviour is correct as described in JSR-168. However this approach is not suitable for many common use cases for servlets, as it excludes the use of pop-up windows, IFRAMEs, and the provision of binary files to the end user.

From testing with Jetspeed/Pluto on Tomcat 5.0 and the default configuration of 5.5, servlets accessed directly do not share the portlet session. In Tomcat 5.5 configured with `emptySessionPath="true"`, behaviour is correct as described in the portlet specification.

The new `getResource` function provided in the draft WSRP 2.0 specification[14] gives one insight into how this confusion might be resolved in the future. `getResource` allows the portlet consumer (the portal) to retrieve a resource such as a JSP within a portlet application, *while providing the portlet context to that resource*. This would explicitly make the portlet session and other parameter settings available to resources fetched in this way.

7.3.3.3 AJAX in portlets

AJAX has become increasingly popular in developing dynamic web interfaces which can change their display and retrieve information from the server without requiring the user to submit a new request or reload the page. The underlying methods are described in Section 6.3; the key concern here is that AJAX interactions require an addressable endpoint on the server for retrieving or submitting data (e.g. a servlet).

Portlets may wish to use AJAX in their interfaces for faster responses to interactions, particularly considering the additional overhead involved in reloading a portal page, and can deploy the corresponding endpoint servlets (or JSPs) in their portlet application.

One potential problem with this approach is due to the previously-described cross-context sessions issue: if the servlet does not share the same session as seen by the portlet, AJAX calls will be unable to see or update the APPLICATION_SCOPE portlet session state. In this situation, either the necessary state must be included with every AJAX request (as GET parameters, so only a small amount of data can be sent this way), or the servlet would only be able to perform tasks that are completely independent of the portlet state.

Even if the servlet and portlet do see the same session, the servlet will be restricted to seeing and setting attributes in the APPLICATION_SCOPE portlet session. Thus AJAX calls to a servlet will not be able to modify session attributes belonging to a particular portlet window, or indeed access any portlet-specific functionality such as reading or setting portlet initialisation parameters or preferences.

JSR-168 does not support AJAX calls which access any kind of portlet functionality. To allow this, the portal would need to provide a directly-accessible endpoint with a URL for a portlet window, for the JavaScript call to access, but with JSR-168, portlets can only be accessed as part of a whole portal page, not individually. Such an endpoint might act in a similar way to the suggested 'Exclusive' portlet mode described earlier: the portlet, in a special render mode, would be able to output the entire page. Then, provided with an appropriate URL, an AJAX call could access the portlet in this mode directly. However, making the 'Exclusive' render mode simply a new Portlet Mode would not have exactly the required effect: the mode of the portlet firstly can only be changed in an Action phase, and secondly should not have been changed as a result of AJAX requests when the user later revisits the portal page: this usage pattern would only want the ‘Exclusive’ portlet mode change to apply for a single AJAX request. Hence, due to the differences in behaviour, an entirely new kind of portlet request alongside *render* and *action* might be the better approach, rather than adding an ‘Exclusive’ portlet mode.

AJAX support is currently still under discussion in both WSRP 2.0 and JSR-286 expert groups; however regardless of the eventual decision, the WSRP 2.0 function `getResource` (described in the previous section) has potential to improve the functionality of servlets as AJAX endpoints, as it should ensure that these servlets are provided with access to the portlet's context.

7.3.3.4 Authentication Integration: Java Authentication and Authorization Service (JAAS)[2]

Each J2EE application server has a different way of configuring new Java Authentication and Authorization Service (JAAS) login modules, so installation of the portlets which make use of JAAS to access remote resources will additionally require this administrative task.

We found that support for JAAS in different portal/application server combinations was sometimes obscure or unreliable, and workarounds or special configurations sometimes had to be added. In particular we found that several portals (Liferay, Jetspeed 2) deliberately override the JAAS settings of the host J2EE server (e.g. Tomcat). Integration of an existing authentication mechanism with that of a particular Portal may therefore be an initial time-consuming task. This is not an issue with the portlet specification, but more generally with the integration of JAAS and J2EE[162].

7.3.3.5 Library conflicts

In some cases, the variety of libraries available from the servlet container and the portal can result in version conflicts with libraries included by the portlet application. This is particularly noticeable with logging libraries (e.g. commons-logging[38] or Log4j[53]) and Apache Struts[32].

Conflicts with logging libraries are usually resolved by removing copies of the libraries from the portlet application, and simply using the ones present in the server or the portal webapp. Some portlet application deployment processes (e.g. Jetspeed's) will automatically remove such logging libraries.

The issue with Apache Struts is more complicated. Struts is commonly used as the framework for servlet applications using the MVC design paradigm, but is not yet directly compatible with the portlet model - later versions should provide transparent portlets support, but until then, workarounds are necessary. One such workaround is a generic Struts bridge, developed as part of the Jetspeed 2 project, which allows an existing Struts-based application to be used as a portlet, and can also be used with other portals. However some portals instead provide their own built-in methods of using Struts with portlets, e.g. with a modified Struts library, and/or inheritance from a StrutsPortlet class provided by the particular portal. This custom Struts support can interfere with more standard use of Struts for servlets in portlet applications - e.g. in helper servlets that should work independently of the portal. In some cases, when the portlet application is deployed, the portal will modify or replace the Struts configuration or the Struts library used, so that the 'normal' Struts-based servlets may no longer work. Such conflicts may be much harder to resolve.

7.3.3.6 Using common code for portlets and servlets

Portlets and servlets use similar, but different classes for representing HTTP Servlet requests, responses, and sessions. This makes sharing common control code between servlets and portlets difficult: for example, a utility function which would get or set a session variable would not be usable in both portlets and servlets, despite the function calls being very similar in appearance (e.g. `HttpSession.getAttribute(name)` vs. `PortletSession.getAttribute(name)`). In addition, the problem may arise when using common libraries which do not (yet) provide a portlet version (e.g. commons-fileupload[37], which now supports portlets in version 1.1 released Dec 2005).

Workaround

If this issue does not appear in many places, and the code can be modified, it may be acceptable to add a few duplicate functions that simply take different parameter types - however this is bad coding practice. Portal-specific solutions are sometimes suggested to obtain an object of the required class, using special knowledge of how to retrieve a particular object from a request, or by knowing which objects are safe to cast (e.g. Apache Pluto's portlet `RenderRequest` can be safely cast to a `HttpServletRequest`).

Our generic, JSR-168-compliant solution was to write a number of simple wrappers or adapters, so that a Portlet object can masquerade as an HTTP Servlet object (or vice versa). For our requirements - mostly session access - this was sufficient, although this approach will clearly fail if an attempt is made to call a function which is not available on the underlying object. The only significant consideration when developing and using these wrappers was that the different scopes available in a PortletSession must be accounted for: when wrapping Portlet and HTTP sessions, it is necessary to specify whether the underlying session should be treated as a PORTLET_SCOPE or APPLICATION_SCOPE session.

7.4 Considerations for the next Portlet Specification

The Portlet Standard 2.0, JSR-286[106], is currently in development.

Inter-portlet communication and portlet filters are already intended to be part of the next specification. IPC will closely follow the model defined in WSRP 2.0[14] (currently in draft).

In addition, we have suggested the following features to the JSR-286 Expert Group, and offered the use cases outlined in this chapter:

1. Cookies: be able to read them, and set them in the Action phase
2. Clarify support for Portlet Windows (instances of the 'same' portlet on different pages)
3. Retrieve ID of current Portlet Window
4. Retrieve user's Portal Session ID (the same for all portlet applications, and even when session is new)
5. Retrieve the remote (client) IP from the request
6. Some form of 'Exclusive' display mode: this would allow portlets to serve non-HTML content
7. Access to Portal Services
    - inspect/manage page layouts, pages
    - inspect/manage users and roles
8. Allow portlets to be queried by AJAX requests. This would need some form of an 'Exclusive' mode and an actual endpoint on the portal allowing direct requests to a portlet window.

The need for the Session ID and Window ID is much less than in JSR-168, as they were primarily useful for implementing IPC, which will no longer be a concern. Some common use cases remain, but these have alternative approaches.

The 'Exclusive' display mode may become unnecessary if JSR-286 supports WSRP 2.0's new 'Resources' feature. This is a new way of accessing resources within the portlet application such as servlets, that provides the resource with all relevant portlet context, and thus resolves the cross-context session problem. Servlets will thus be able to reliably access the portlet session, as if in a Render phase, although they will be unable to use IPC or other services restricted to the Action phase.

The other potential use of an Exclusive mode is by AJAX applications. Support for AJAX is acknowledged as important and is being seriously considered by both WSRP 2.0 and JSR-286 expert groups. However it may be too complex for inclusion at this stage, and left to version 3.

7.5 Conclusion

In this chapter, we have discussed the results of our experience in using Portal software to manage websites, in particular Portals supporting the JSR-168 Portlet specification. After reviewing the alternative development approaches of using J2EE servlets or other, non-JSR-168 portals, we have assessed the limitations of JSR-168 portlets in both design and implementation, and presented current solutions and workarounds.

As a result of these investigations, we have highlighted a number of features that might be beneficial to add in the next portlet specification, provided relevant use cases, and submitted them to the JSR-286 Expert Group. Any new features in this specification will also be coordinated with the next version of WSRP, also currently under development.

Chapter 8. Conclusions and Future Work

In this thesis we have discussed the needs and the development of analytical web portals, with analysis components provided by portlets accessing Discovery Net services. The most important Discovery Net portlet in this respect is the Service portlet, which provides a web interface for parameterising and executing workflows. In Chapter 4, we described how workflow creators can define a simplified "black-box" model of their workflow, and design its corresponding web interface, all using graphical tools in the Discovery Net Java Client. The level of abstraction introduced by the 'deployment' process enables us to combine the automated (codeless) generation of interfaces with customisation of individual services to create more intuitive and user-friendly web interfaces.

The use of portlets in the Discovery Net Portal has brought several notable improvements in comparison to the servlet-based previous version.

1) The explicit separation of Discovery Net Portal functionality into individual portlets allows page layouts to be easily created and edited (Chapter 3).
2) The most significant advantage is in the ability to treat each Service portlet as an independent component. Thus, multiple Service portlets showing different services can be placed on the same or different pages for easy access. This had previously been a requested feature, but had been completely impractical to implement with the static structure of the servlet-based site.
3) The new (and extremely useful) ability to directly pass results from one Service to another was achieved using our inter-portlet communication (IPC) library.

While developing the new Discovery Net portal and its collection of portlets, we learned much about the specific advantages and limitations of using the JSR-168 portlet standard. As a result of our experiences with JSR-168, we have suggested several new features to the Portlet 2.0 (JSR-286)[106] Expert Group, and supplied corresponding use cases. We have reported on our findings and explained some confusing or problematic situations that may be encountered by portlet developers (Chapter 7). Whenever relevant, we have presented workarounds and solutions to problems, the most significant of which was the development of a JSR-168-compliant library for IPC.

IPC was left out of JSR-168, but is still an essential feature when developing portals. Specific portal implementations often include easy-to-use IPC services for their native (non-JSR-168) portlets. On the other hand, finding a messaging model that fits within the constraints imposed by JSR-168 and developing a corresponding messaging library takes significant time and effort, as well as a good understanding of the limitations of the specification and portlet behaviour. This puts JSR-168 portlet developers (particularly those new to the field) at a severe disadvantage.

We therefore developed a JSR-168-compliant and open-source library for IPC (Chapter 5), which offers a portable alternative to the portal-specific solutions, and has been made freely available for download. This significantly improves the situation for portlet developers, eliminating the hurdle of having to implement the mechanics of a portlet messaging service. We have used this library heavily in our development of interacting Discovery Net portlets, and also in the custom-developed portlets for the Translational Medicine Portal.

The use of portal software such as Jetspeed[29] or Oracle Portal[66] provides valuable, time-saving features including user authentication and management, user preferences, and page creation and customisation. However, some other important features are missing from JSR-168 portals: apart from IPC, support for the recently popular style of web page design centred on 'AJAX'[27] is possibly the most in demand. AJAX is the use of JavaScript for asynchronous fetching of remote resources, which can then be used to update the page display without reloading it. Its use typically produces dynamic and interactive web pages, which behave more like fast-responding client-side applications than static documents. Current JSR-168 and WSRP standards do not provide enough 'hooks' for full exploitation of AJAX methods in portlets, although limited use is possible. It is described in more detail in Section 7.3.3.3 and in Section 8.2 below, and is used with excellent effect in the Translational Medicine Portal's OLAP browser (Chapter 6).

A variety of actual applications have been used to inform and illustrate this work, from theoretical demos to real-world portals. All of the application scenarios have been collaborative efforts, involving other research groups, their students and commercial companies. Different applications have different focuses and requirements: some rely on user-friendly presentation of a few, well-known services; some emphasise flexibility and allow users to create their own custom pages from a large library of services; some use the Discovery Net portlets as just another component in an existing, larger portal.

All, however, take advantage of the Discovery Net Service portlet, which is the core portlet of the system. This can add analysis functionality to an existing JSR-168 portal, allowing users to do more within a familiar Portal environment, and administrators to easily add new functionality to the system. The Service portlet has the great advantage that once installed, it can provide access to any Discovery Net service, whether available at the time of installation or in the future. This significantly reduces the burden on the portal administrator, who might otherwise have to install individual portlets for every service as it became available. The best examples of this kind of integrated portal are the Oracle Business Intelligence demo (Section 3.9) and the Translational Medicine Portal (Chapter 6) developed for the Windber Research Institute. The ability to support these different scenarios is thanks to the wide range of service deployment features providing functionality, and the flexibility of the portlet system in providing final integration and presentation on the web.

In general we have found JSR-168 to be up to the task of building analytical portals, despite its limitations, mainly due to the benefits of being able to deploy our portlets on other portal servers. Experience from developing servlet-based applications is very applicable, as there are close similarities in the portlet model, and portlets can also make use of JSPs and servlet resources. However, it has taken several years since the publication of JSR-168 for portlet support to be added to commercial portal implementations and Java web libraries and frameworks (e.g. Apache Struts[32], JavaServer Faces[22], commons-fileupload[37]). With the ongoing development of the second portlet standard, we expect that standard portlets (JSR-168 and later JSR-286) will continue to become better established and supported with more mature development tools.

Thus in summary, the main advantages that the use of portlets provide to the Discovery Net Web Portal are:

- Time-saving services provided by portal software, particularly the ability for administrators and even users to create and edit portal pages.
- Clear separation of Discovery Net functionality into components, which can be flexibly added to page layouts.
- Communication between Service portlets, allowing results to be automatically passed on from one service as input to another, using our JSR-168-compliant IPC library.
- Integration of Discovery Net portlets with clients' existing JSR-168-compliant portals.
- Third-party portlets can be installed on the Discovery Net Portal to extend its functionality and meet specific user needs.

The main disadvantage of the current implementation is that due to incompatibilities between servlet and portlet development tools and libraries (particularly Apache Struts), it is difficult to adapt and maintain an existing servlet-based site for parallel use as portlets while avoiding significant code duplication. It may in future become necessary to abandon Struts for a portlet-compatible system such as JavaServer Faces.

8.1 Future Developments

We expect there will be interesting developments in the upcoming JSR-286 and WSRP 2.0 specifications, which should be released by the end of 2007. In particular, a standardised method for inter-portlet communication will certainly be defined. We hope that portal implementers will provide user-friendly methods for configuring messaging channels/wires between portlets.

Of more interest is whether explicit support for AJAX-style portlet development will be added - this is currently uncertain, but potentially very useful. AJAX could be used in two contexts: within portlets, as was done in our Windber OLAP Browser portlet, or as a core portal design decision, as a way of updating individual portlets on a page without needing to reload the entire page. The latter has been tried before (e.g. by Plumtree Portal[89]), and has potential for improving the client experience, as large portal pages can be very slow to reload even if only one portlet has changed.

The confusion caused to portlet developers by the separation of sessions seen by servlets and portlets is likely to be resolved by the new concept of 'resources' introduced by WSRP 2.0. This will allow resources such as JSPs to be accessed with the portlet context explicitly provided to them.

Designers of flexible, dashboard-style portals sometimes wish for a way for portlets to modify the current page layout: e.g. to add a new portlet directly on the page as a result of a user interaction with a standard portlet. This is currently not possible, as JSR-168 does not provide portlets with access to the portal page management service. This feature, along with other portal services such as user management, is likely to remain out of reach for JSR-286 portlets, as it is not generally considered to be within the scope of what a portlet should be able to do.

In 2006/7 there has been a lot of activity and interest centred on a related concept: web page 'widgets' (sometimes called 'gadgets' or 'badges') which are usually implemented using Javascript/AJAX or Flash, and which may be easily embedded on any web page while being hosted elsewhere. This is very similar in concept to WSRP's remote portlets, which are hosted on one server and accessed and presented through a different portal. The current situation with web widgets parallels that of the portlet world pre-standardisation, with many different widget providers (e.g. Netvibes[119], Google Gadgets[115], Pageflakes[120], Windows Live Gadgets[122]) offering similar but usually incompatible methods of widget creation and hosting. In contrast with (mainly Java-based) portlets, there is usually more freedom of choice and variety in server-side implementation technology, and the resulting widgets are very simple to embed in web pages, which allows widgets from different sources to be combined easily on the same page. Web widgets have become a quite general and popular consumer phenomenon on the internet, whereas portlets (being mainly associated with J2EE) have gained their main audience in commercial organisations with managed intranets.

A further related concept is that of operating system widgets (e.g. Yahoo Widgets[124], Windows Vista Sidebar Gadgets[123], Mac Dashboard Widgets[118], iPhone Web Apps[116]), whose recent implementations are starting to overlap with those of web widgets as many are now developed using web technologies (usually Javascript). A standardisation effort is beginning (Widgets 1.0[121]), but this is still in its early stages. There is now an opportunity for the widget community to reuse solutions and learn from the experiences of JSR-168 portal and portlet developers.

8.2 Future Work

Serious development on the IPC library has probably reached its natural end, as it will be superseded by the standardised IPC which will be included in JSR-286. However, until support for JSR-286 in portals becomes widespread - which may take several years - the library will remain a useful resource to developers. If appropriate, we may revise the library to use a messaging model more in line with that eventually decided for JSR-286, so that developers using it have an easier upgrade path in future.

Discovery Net services have already undergone many changes over the course of this work. However, further development can be expected in two areas: firstly, semantic metadata for service inputs and outputs, and secondly, a server-side mechanism for explicitly supporting data flow in multiple-service analysis procedures.

Semantic descriptions of service inputs and outputs would add the potential for semi-automated composition of compatible services. This general concept has been explored in semantic extensions to WSDL such as DAML-S[174]. For example, it would be possible to analyse the result output metadata, and provide the user with a list of services which could use that type of result as input, and whose general service description matches the application area of the original service. More advanced would be the ability to offer conversion utilities to intelligently transform a result into a form usable as input to another service. An implementation of these features would probably make use of Semantic Web[175] technologies and concepts, such as RDF[176] for markup and ontologies for semantic descriptions and comparisons[149].

While the Service portlets support the transfer of results from one service to another, this is reliant on the portlet system for setting up the messaging channels. This is only applicable when using the workflows on the portal; the connections are set up on the portal server. Thus the Discovery Net workflow engine is not 'aware' that services are being used together in a sequence, and other workflow clients such as the Discovery Net Java Client or even the traditional servlet-based portal cannot make use of service flows set up on the portlet-based Portal. Future developments are therefore likely to include core server support for multiple-service processes, perhaps using a standard orchestration language such as BPEL (Business Process Execution Language[61]).

8.3 Final Thoughts

Despite complexities resulting from the use of the relatively recent portlet standard, the development of Discovery Net portlets has been well worth the effort. We were delighted when our belief in the potential of a standards-based analytical portal was validated by the reaction of the Windber researchers: what started as a database redesign became a much larger and ongoing long-term project to develop the Translational Medicine portal.

We also hope that the availability of a library for inter-portlet communication will make life easier for JSR-168 portlet developers. The main portlet development websites and communities have linked to the library, and we have received some very useful feedback as a result. Later this year, the much anticipated release of JSR-286 and WSRP 2.0 will eventually provide more flexible and powerful options to portlet developers, hopefully enabling both standardised IPC and AJAX support.

But most of all, we look forward to seeing many exciting and useful innovations in the development of interactive web interfaces over the next few years, as developers learn to exploit the full potential of AJAX-style design, and the typical web portal begins to show more of the characteristics of an Analytical Portal.

Glossary

AJAX	Asynchronous JavaScript and XML, a JavaScript programming language feature enabling the development of very reactive and interactive web interfaces.
API	Application Programming Interface
Applet	A Java application which can be embedded on a web page through a browser plugin.
CERN	European Organization for Nuclear Research
CMS	Content Management System
DOM	Document Object Model, a standardized representation of the elements within an XML or HTML document as a tree.
GIS	Geographic Information System, for programmatic treatment of map-based information.
GM	Genetically Modified
GUI	Graphical User Interface
IFRAME	Inline Frame, an HTML element
IPC	Inter-portlet Communication
J2EE	Java Platform, Enterprise Edition
JAR	Java Archive, a compressed file format containing resources such as compiled Java classes.
JSP	Java Server Pages, a language for dynamic web pages.
JSR-168	Java Specification Request 168, the Java Portlet Standard 1.0 (2003).
JSR-286	Java Specification Request 286, the Java Portlet Standard 2.0 (2007, in progress).
MVC	Model-View-Controller, an architectural pattern for building applications.
OASIS	Organization for the Advancement of Structured Information Standards
OLAP	Online Analytical Processing, for producing rapid results to queries over a large multidimensional data set.

Portal	Typically, any web site which aggregates several different types of functionality and presents them to users. A single page often contains several different services. Users may be allowed to customise the pages to their own requirements. A subset of portals support the portlet standards.
Portlet	A subsection of a web page whose functionality and implementation is completely or mostly self-contained. Portlets may be compatible with one of the portlet related standards, JSR-168 or WSRP.
Portlet standards	JSR-168 is the standard for Java-implemented portlets. WSRP describes a web-service interface for accessing portlets implemented in any programming language.
SOAP	Simple Object Access Protocol, used by Web Services.
Translational Medicine	Methodology enabling closer linking between clinical practice and research labs.
URL	Uniform Resource Locator; a web "link" or "hyperlink".
VLE/VRE	Virtual Learning Environment / Virtual Research Environment, typically a web Portal providing access to educational or research services.
VM	Virtual Machine
Web Application	Either a) an interactive service provided through a web site, or b) a J2EE 'webapp', a server-side implementation approach.
WRI	Windber Research Institute
WSDL	Web Services Description Language
WSRP	Web Services for Remote Portlets, an OASIS standard. Version 1.0 is closely related to JSR-168, and Version 2.0 (in progress) with JSR-286.
XML	Extensible Markup Language
XUL	XML User Interface Language

Bibliography

[1] Java Applets. http://java.sun.com/applets/

[2] Java Authentication and Authorisation Service (JAAS), *Sun Developer Network*. http://java.sun.com/products/jaas/

[3] "Model-View-Controller", *Java BluePrints: J2EE Patterns* 2000. http://java.sun.com/blueprints/patterns/MVC-detailed.html

[4] PHP-Nuke. http://phpnuke.org/

[5] Research Councils UK: e-Science Programme. http://www.rcuk.ac.uk/escience/

[6] Web Services Description Language (WSDL), 2001. http://www.w3.org/TR/wsdl

[7] XML User Interface Language (XUL), *Mozilla* 2001. http://www.mozilla.org/projects/xul/

[8] Apache Axis, *Apache Web Services Project*. http://ws.apache.org/axis/

[9] Universal Description, Discovery and Integration (UDDI) OASIS Standard, 2002. http://www.uddi.org/

[10] W3C Web Services Activity. http://www.w3.org/2002/ws/

[11] Java 2 Platform Enterprise Edition, Version 1.4 (J2EE), 2003. http://java.sun.com/j2ee/docs.html

[12] Java Server Pages, Version 2.0, 2003. http://jcp.org/en/jsr/detail?id=152

[13] Java Servlet 2.4 Specification, 2003. http://jcp.org/en/jsr/detail?id=154

[14] OASIS Web Services for Remote Portlets (WSRP) Technical Committee. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp

[15] Portlet Open Source Trading site (POST). http://portlet-opensrc.sourceforge.net/index.htm

[16] Portlet Specification (JSR 168), 2003. http://www.jcp.org/en/jsr/detail?id=168

[17] Simple Object Access Protocol (SOAP), 2003. http://www.w3.org/TR/soap/

[18] SOAP::Lite for Perl. http://soaplite.com/

[19] An OWL-based Web Service Ontology, OWL-S (formerly DAML-S), 2004. http://www.daml.org/services/owl-s/

[20] Developing Virtual Research Environments, *JISC Circular* 2004. http://www.jisc.ac.uk/index.cfm?name=funding_circular5_04

[21] JISC Virtual Research Environments Programme, 2004. http://www.jisc.ac.uk/index.cfm?name=programme_vre

[22] JSR 127: JavaServer Faces, 2004. http://www.jcp.org/en/jsr/detail?id=127

[23] "My Yahoo", 2004. http://my.yahoo.com/

[24] NetUnity - WSRP Portal and WSRP Portlet Framework for .Net. http://www.netunitysoftware.com/e/

[25] OASIS Web Services Resource Framework (WSRF) Technical Committee, 2004. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[26] "Oracle Business Intelligence Discoverer New Features Overview", 2004. http://www.oracle.com/technology/products/discoverer/htdocs/oraclebi_discoverer_1012_fov.htm

[27] "Ajax: A New Approach to Web Applications", Jesse James Garrett, 2005. http://www.adaptivepath.com/publications/essays/archives/000385.php

[28] Apache Jetspeed 1.6 ("Fusion"), *Apache Portals Project*. http://portals.apache.org/jetspeed-1/

[29] Apache Jetspeed 2, *Apache Portals Project*. http://portals.apache.org/jetspeed-2/

[30] Apache Pluto, *Apache Portals Project*. http://portals.apache.org/pluto/

[31] Apache Portals Bridges, *Apache Portals Project*. http://portals.apache.org/bridges/

[32] Apache Struts. http://struts.apache.org/

[33] Apache Tomcat. http://tomcat.apache.org/

[34] ASP.NET. http://www.asp.net

[35] BEA WebLogic Portal. http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/weblogic/

[36] CCLRC e-Science Centre. http://www.e-science.clrc.ac.uk/web

[37] Commons FileUpload, *Apache Jakarta*. http://jakarta.apache.org/commons/fileupload/

[38] Commons Logging. http://jakarta.apache.org/commons/logging/

[39] Discovery Net Project, 2005. http://www.discovery-on-the.net/

[40] ESRI - GIS and Mapping Software. http://www.esri.com/

[41] eXo Portal. http://www.exoplatform.com

[42] Gems: A collection of small JSR-168 compliant Portlets. https://gems.dev.java.net/

[43] Generic SOAP Client. http://www.soapclient.com/soaptest.html

[44] GIS.com - the Guide to Geographic Information Systems. http://www.gis.com/index.cfm

[45] Groove Virtual Office. http://www.groove.net/home/index.cfm

[46] IBM WebSphere Portal. http://www-306.ibm.com/software/websphere/

[47] Jakarta Tapestry. http://jakarta.apache.org/tapestry/

[48] Java Web Start Technology. http://java.sun.com/products/javawebstart/

[49] Java.net Portlet Community. http://community.java.net/portlet/

[50] JBoss Application Server. http://www.jboss.org/products/jbossas

[51] JBoss Portal. http://labs.jboss.com/portal/jbossportal/index.html

[52] Liferay Portal. http://www.liferay.com/

[53] Log4j, *Apache Logging Services*. http://logging.apache.org/log4j/docs/

[54] Macromedia Flash. http://www.macromedia.com/software/flash/flashpro/

[55] "MapServer Portlet", 2005. http://wiki.apache.org/portals/MapServerPortlet

[56] MDL Chime. http://www.mdlchime.com/

[57] MDL CrossFire Commander. http://www.mdl.com/products/knowledge/crossfire_commander/

[58] MDL DiscoveryGate. http://www.mdl.com/products/knowledge/discoverygate/

[59] Microsoft Office SharePoint Portal Server. http://office.microsoft.com/sharepoint/

[60] Moveable Type, 2005. http://www.sixapart.com/movabletype/

[61] OASIS Web Services Business Process Execution Language (WSBPEL), 2005. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[62] "Online Banking", *Moneyextra Guides* 2005. http://www.moneyworld.co.uk/guides/online-banking-011690.html

[63] Open Grid Computing Environment (OGCE). http://www.collab-ogce.org/nmi/index.jsp

[64] Oracle Application Server. http://www.oracle.com/appserver/index.html

[65] Oracle Business Intelligence. http://www.oracle.com/technology/products/bi/index.html

[66] Oracle Portal. http://www.oracle.com/technology/products/ias/portal/index.html

[67] Oracle Spatial. http://www.oracle.com/technology/products/spatial/index.html

[68] "PathPort: The Pathogen Portal Web Project", *Virginia Bioinformatics Institute*. http://pathport.vbi.vt.edu/main/home.php

[69] PHP. http://www.php.net/

[70] PostNuke. http://www.postnuke.com

[71] Sakai Portal. http://sakaiproject.org/

[72] SciFinder Scholar, *CAS*. http://www.cas.org/SCIFINDER/SCHOLAR/

[73] Slashcode. http://www.slashcode.com/

[74] SMILES, *Daylight Chemical Information Systems, Inc*. http://www.daylight.com/smiles/f_smiles.html

[75] Spotfire - Interactive Visual Analytics. http://www.spotfire.com/

[76] Stringbeans Portal. http://www.nabh.com/projects/sbportal

[77] Sybase Enterprise Portal. http://www.sybase.co.uk/products/developmentintegration/enterpriseportal.html

[78] The GridSphere Portal Framework. http://www.gridsphere.org

[79] The Large Hadron Collider (LHC) at CERN. http://lhc.web.cern.ch/lhc/

[80] The London e-Science Centre. http://www.lesc.ic.ac.uk/

[81] The Perl Directory. http://www.perl.org/

[82] Typo3 Content Management System (CMS). http://www.typo3.com/

[83] uPortal by JA-SIG. http://www.uportal.org/

[84] Web Service Semantics - WSDL-S, 2005. http://www.w3.org/Submission/WSDL-S/

[85] WebCT. http://www.webct.com/

[86] Windber Research Institute. http://www.wriwindber.org/

[87] Amazon. http://www.amazon.com

[88] BBC News. http://news.bbc.co.uk

[89] BEA AquaLogic User Interaction (previously Plumtree Portal). http://www.plumtree.com/products/

[90] CGB Bioinformatics Portal, *Indiana University's Center for Genomics and Bioinformatics*. http://bioportal.cgb.indiana.edu/

[91] Condor Project. http://www.cs.wisc.edu/condor/

[92] deltaDOT Ltd.. http://www.deltadot.com/

[93] "Description of portlet work done for the Go-Geo! project" (EDINA), 2006. http://www.gogeo.ac.uk/geoPortal10/PortletInfo.html

[94] European Bioinformatics Institute (EBI) - Software Tools. http://www.ebi.ac.uk/Tools/

[95] GENIUS Grid Portal. https://genius.ct.infn.it/

[96] INFN Production Grid for Scientific Applications. http://grid-it.cnaf.infn.it

[97] InforSense KDE (Knowledge Discovery Environment). http://www.inforsense.com/kde.html

[98] JavaScript, *Mozilla*. http://www.mozilla.org/js/

[99] JBoss PortletSwap. http://labs.jboss.com/portal/portletswap/index.html

[100] National Grid Service. http://www.ngs.ac.uk/

[101] NCBI Tools for Bioinformatics Research. http://www.ncbi.nlm.nih.gov/Tools/

[102] NIST Data Gateway. http://srdata.nist.gov/gateway/

[103] P-GRADE Grid Portal. http://www.lpds.sztaki.hu/pgportal/

[104] P-GRADE NGS portal. http://www.cpc.wmin.ac.uk/ngsportal/index.php

[105] "Portal Object Management – Dynamicity", JBoss Portal v2.2 User Guide, http://docs.jboss.com/jbportal/v2.2/user-guide/en/html/dynamicity.html

[106] Portlet Specification 2.0 (JSR 286), 2007. http://jcp.org/en/jsr/detail?id=286

[107] PubMed. http://www.pubmed.gov

[108] RCSB Protein Data Bank (PDB). http://www.rcsb.org/pdb/Welcome.do

[109] Sun N1 Grid Engine. http://www.sun.com/software/gridware/

[110] The Globus Alliance. http://www.globus.org/

[111] W3C Web Services Addressing (WS-Addressing) Working Group. http://www.w3.org/2002/ws/addr/

[112] Wiki, *Wikipedia*. http://en.wikipedia.org/wiki/Wiki

[113] WS-Security OASIS standard. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

[114] Yahoo Group: JSR 168 Portlets & Java Portals.
http://groups.yahoo.com/group/portlets/

[115] Google Gadgets. http://www.google.com/ig

[116] iPhone Web Apps. http://developer.apple.com/iphone/

[117] Java Message Service (JMS). http://java.sun.com/products/jms/

[118] Mac Dashboard Widgets. http://www.apple.com/downloads/dashboard/

[119] Netvibes. http://www.netvibes.com/

[120] Pageflakes. http://www.pageflakes.com/

[121] Widgets 1.0, 2007. http://www.w3.org/TR/widgets/

[122] Windows Live Gadgets. http://dev.live.com/gadgets/

[123] Windows Vista Gadgets.
http://vista.gallery.microsoft.com/vista/SideBar.aspx?mkt=en-us

[124] Yahoo Widgets. http://widgets.yahoo.com/

[125] A.Rowe, Y.Guo, D.Kalaitzopoulos, M.Osmond, and M.Ghanem. "The Discovery Net System for High Throughput Bioinformatics". *ISMB,* Vol 19 p.i225-i231, 2003. http://www.iscb.org/ismb2003/paperAbstracts/btg1031.pdf

[126] Andrew Cox. "Building Collaborative eResearch Environments", *JISC Workshop report* 2004. http://www.jisc.ac.uk/index.cfm?name=event_eresearch

[127] BioTeam. "iNquiry", 2006. http://web.bioteam.net/metadot/index.pl?iid=2187

[128] Carole Goble and David De Roure. "Semantic Web and Grid Computing", 2002. http://www.semanticgrid.org/documents/swgc/swgc-final.pdf

[129] D.Gannon, G.Fox, M.Pierce, B.Plale, G.von Laszewski, C.Severance, J.Hardin, J.Alameda, M.Thomas, and J.Boisseau. "Grid Portals: A Scientist's Access Point for Grid Services (Draft 1)", *GGF Community Practice document* 2003. http://www.extreme.indiana.edu/~gannon/ggf-portals-draft.pdf

[130] David De Roure, Nicholas Jennings, and Nigel Shadbolt. "The Semantic Grid: Past, Present, and Future". *Proceedings of the IEEE,* Vol 93 p.669-681, 2005.

[131] Deepak Alur, John Crupi, and Dan Malks. "Core J2EE Patterns", 2001.

[132] Dwight Deugo. "Mobile Agent Messaging Models", Fifth International Symposium on Autonomous Decentralized Systems, 2001.
http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/

&toc=comp/proceedings/isads/2001/1065/00/1065toc.xml&DOI=10.1109/ISADS.2001.917429

[133] Elsevier MDL. "CrossFire Beilstein", 2005. http://www.mdl.com/products/knowledge/crossfire_beilstein/index.jsp

[134] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. "Design Patterns", 1995.

[135] G.Fox, M.Pierce, D.Gannon, and M.Thomas. "Overview of Grid Computing Environments", *Global Grid Forum Memo, GFD-I.9* 2003. http://www.gridforum.org/documents/GFD/GFD-I.9.pdf

[136] Glenn R Golden. "Cross Context Sessions & Pluto", *Jakarta Pluto Discussion mailing list* 2004. http://mail-archives.apache.org/mod_mbox/portals-pluto-user/200401.mbox/%3c83F94AE0-423B-11D8-86CE-000A95A93076@umich.edu%3e

[137] Gregor Hohpe and Bobby Woolf. "Enterprise Integration Patterns", 2003.

[138] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. "Aspect-Oriented Programming", *1241* Proceedings of the European Conference on Object-Oriented Programming,

[139] Ian Foster. "Globus Toolkit Version 4: Software for Service-Oriented Systems". *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779,* p.2-13, 2005. http://www.globus.org/toolkit/

[140] Ian Foster, Carl Kesselman, Jeffrey M.Nick, and Stephen Tuecke. "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", *Global Grid Forum* 2002.

[141] Ian Foster, Carl Kesselman, and Stephen Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organisations". *International J.Supercomputer Applications,* Vol 15 2001.

[142] J G Liu, P J Mason, N Clerici, S Chen, and A Davis. "Landslide Hazard Assessment in the Three Gorges Area of the Yangtze River using ASTER Imagery.", *IEEE IGARSS2003* 21 July 2003.

[143] Jameel Amjad Syed. "Information Structuring for Managing Discovery". Thesis/Dissertation, University of London PhD Computing, 2005.

[144] James Hendler. "Science and the Semantic Web", *Science* Vol 299 p.520-521, 2003.

[145] Jeffrey Frey, Steve Graham, Tom Maguire, David Snelling, and Stephen Tuecke. "WS-Resource Framework and WS-Notification - Technical Overview", 2004. http://www.nesc.ac.uk/talks/385/WS-ResourceFramework_UK_2004-01-28.ppt

[146] Jian Guo Liu and Jinming Ma. "Imageodesy on MPI & GRID for Co-seismic Shift Study Using Satellite Optical Imagery", UK e-Science All Hands Meeting, Nottingham, UK, Sept. 2004.

[147] Jian Guo Liu, Philippa J Mason, and Jinming Ma. "The co-seismic displacement of Ms 8.1 Kunlun earthquake on 14th November 2001 measured from Landsat-7 ETM+ imagery", The 3rd International conference on Continental Earthquakes, Beijing, China, 12 July 2004.

[148] John LaCasse. "How to get Session in portlet", *Jetspeed-User mailing list* 2005. http://www.mail-archive.com/jetspeed-user@jakarta.apache.org/msg15009.html

[149] Jorge Cardoso and Amit Sheth. "Semantic e-Workflow Composition", 2002. http://lsdis.cs.uga.edu/lib/download/TM02-004-Cardoso-Sheth.pdf

[150] Karl Czajkowski, Donald F.Ferguson, Ian Foster, Jeffrey Frey, Steve Graham, Igor Sedukhin, David Snelling, Steve Tuecke, and William Vambenepe. "The WS-Resource Framework (Whitepaper)", 2004. http://www.globus.org/alliance/publications/papers.php#WSRF-Framework

[151] M.Ghanem, Y.Guo, J.Hassard, M.Osmond, and M.Richards. "Grid-based Data Analysis of Air Pollution Data", Fourth International Workshop on Environmental Applications of Machine Learning, 2004.

[152] M.Richards, M.Ghanem, Y.Guo, J.Hassard, and M.Osmond. "Sensor Grids for Air Pollution Monitoring", UK e-Science All Hands Meeting 2004, Nottingham UK, 1 Sept. 2004.

[153] Mark Baker, Hong Ong, Rob Allan, and Xiao Dong Wang. "Virtual Research in the UK: Advanced Portal Services", UK e-Science All Hands Meeting 2004, Nottingham UK, 2 Sept. 2004. http://www.allhands.org.uk/2004/proceedings/papers/223.pdf

[154] Martyn Foster, Daniel Hanlon, Jon MacLaren, James Marsh, Stephen Pettifer, and Stephen Pickles. "Grid-Enabled Desktop Environments: The GRENADE Project", UK e-Science All Hands Meeting 2004, Nottingham UK, 1 Sept. 2004.

[155] Michelle Osmond and Yike Guo. "Adopting and Extending Portlet Technologies for e-Science Workflow Deployment", UK e-Science All Hands Meeting 2005, Nottingham UK, 19 Sept. 2005. http://www.allhands.org.uk/2005/proceedings/papers/446.pdf

[156] Mintel. "Mintel Report: UK Retail Briefing - Electricals Focus - December 2005", 2005. http://reports.mintel.com/sinatra/reports/press_releases/view=press_view&levels=536,1692/press_display/id=197382

[157] Nate L Root. "Say Goodbye to Portal Servers", *Analyst Report: Forrester Trends* 2005. http://www.microsoft.com/office/sharepoint/prodinfo/forrester_mar04.mspx

[158] Navaneeth Krishnan. "Java.net Tip: Inter-portlet communication", https://www.dev.java.net/files/documents/1654/8898/tip1.html

[159] P N Martin. "Measurement of Atmospheric Trace Gases Using Open Path Differential UV Absorption Spectroscopy for Urban Pollution Monitoring". Thesis/Dissertation, University of London PhD, 2002.

[160] Philip McCarthy. "Ajax for Java developers: Build dynamic Java applications", *IBM developerWorks* 2005. http://www-128.ibm.com/developerworks/web/library/j-ajax1/

[161] Punit Pandey. "Blog: JSR 168, WSRP, Portlets & Enterprise Portal", 2006. http://portlets.blogspot.com/

[162] Raymond K Ng and Ganesh Kirti. "JAAS in the Enterprise", *Java Developer's Journal* 2006. http://java.sys-con.com/read/171477.htm

[163] Richards, M., Ghanem, M., Osmond, M., Guo, Y., and Hassard, J. "Grid-based analysis of air pollution data". *Ecological Modelling,* Vol 194 p.274-286, 2006. http://www.sciencedirect.com/science/article/B6VBS-4J6WG0S-1/2/ffb04b0b69410ea465a039655ab01a13

[164] Rob Allan, Alison Allden, David Boyd, Rob Crouchley, Nicole Harris, Liz Lyon, Alan Robiette, David De Roure, and Scott Wilson. "Roadmap for a UK Virtual Research Environment: Report of the JCSR VRE Working Group", 2005.

[165] Rob Allan, Chris Awre, Mark Baker, and Adrian Fish. "Portals and Portlets 2003", *NeSC Workshop Report: Edinburgh July 2003*.

[166] Rob Crouchley, Adrian Fish, Rob Allan, and Dharmesh Chohan. "Sakai Evaluation Exercise", *JISC Report* 2004. http://www.grids.ac.uk/Sakai/sakai_doc.pdf

[167] Roberto Barbera. "The GENIUS Grid Portal", Portals and Portlets 2003, http://www.nesc.ac.uk/action/esi/download.cfm?index=734

[168] S F Altschul, T L Madden, A A Schaffer, J Zhang, Z Zhang, W Miller, and D J Lipman. "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs.". *Nucleic Acids Res.*, 1997.

[169] S.Hassard, M.Osmond, F.Pereira, M.Howard, S.Klier, R.Martin, and J.Hassard. "Distributed BioSensor systems for GM Crop Monitoring", UK e-Science All Hands Meeting 2004, Nottingham UK, 1 Sept. 2004. http://www.allhands.org.uk/2004/proceedings/papers/92.pdf

[170] Salman AlSairafi. "Visualisation and Data Mining of GIS Data". Thesis/Dissertation, University of London MSc Advanced Computing, 2001.

[171] Salman AlSairafi, Filippia-Sofia Emmanouil, Moustafa Ghanem, Nikolaos Giannadakis, Yike Guo, Dimitrios Kalaitzopoulos, Michelle Osmond, Anthony Rowe, Jameel Amjad Syed, and Patrick Wendel. "The Design of Discovery Net: Towards Open Grid Services for Knowledge Discovery". *International Journal of High Performance Computing Applications,* Vol 17 2003.

[172] Stefan Egglestone, M.Nedim Alpdemir, Chris Greenhalgh, Arijit Mukherjee, and Ian Roberts. "A portal interface to myGrid workflow technology", UK e-Science All Hands Meeting 2005, Nottingham UK, http://www.allhands.org.uk/2005/proceedings/papers/404.pdf

[173] Stefan Hepper and Marshall Lamb. "Best practices: Developing portlets using JSR 168 and WebSphere Portal V5.02", 2004. http://www-

106.ibm.com/developerworks/websphere/library/techarticles/0403_hepper/0403_hepper.html

[174] The DAML Services Coalition. "DAML-S: Semantic Markup for Web Services", 2002. http://www.daml.org/services/daml-s/0.7/daml-s.html

[175] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web". *Scientific American,* 2001. http://www.sciam.com/article.cfm?colID=1&articleID=00048144-10D2-1C70-84A9809EC588EF21

[176] Uche Ogbuji. "An introduction to RDF: Exploring the standard for Web-based metadata", *IBM* 2002. http://www-106.ibm.com/developerworks/xml/library/w-rdf/?dwzone=xml&dwzone=xml

[177] V.Curcin, M.Ghanem, Y.Guo, M.Köhler, A.Rowe, J.Syed, and P.Wendel. "Discovery Net: Towards a Grid of Knowledge Discovery", KDD-2002. The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Edmonton, Alberta, Canada, 23 July 2002.

[178] Wayne Holder. "Session, Session, who's got my Session?", *Wayne Holder's Blog* 2005. http://weblogs.java.net/blog/wholder/archive/2005/02/session_session.html

Appendix A1: Functions for retrieving Portlet IDs

The following functions can be used to retrieve or generate a unique ID for a portlet window, and are included as utilities in our IPC Library.

```
public String MSG_PORTLET_ID = "message.portletID";

/** Utility: get (and set, if not yet present) a unique ID for this
*  portlet instance, cached in the session.
*  This uses the portlet context name, so IDs should be unique
*  across portlet applications.
*  If useWindowID is set, it tries to retrieve the actual portlet window ID
*  assigned by the portal. If not set, it generates a random number for the ID.
*/
public String getPortletID(PortletRequest request, boolean useWindowID){
        // see if portlet id has been set in session.
        PortletSession session = request.getPortletSession(true);
        if (session.getAttribute(MSG_PORTLET_ID)==null
              || session.getAttribute(MSG_PORTLET_ID).toString().equals("")){
                // no id set; generate a new one
                String id = "";
                if (useWindowID){
                        id = getPortletWindowID(request);
                }
                if (id==null || id.length()==0){
                        // make a random ID
                        double random = getRandomNumber();
                        id = ""+random;
                }
                String context_name = request.getContextPath();
                session.setAttribute(MSG_PORTLET_ID, context_name+"_"+id);
        }
        return session.getAttribute(MSG_PORTLET_ID).toString();
}


private double getRandomNumber(){
        double n1 = Math.random();
        double n2 = Math.random();
        double random = n1 / n2;
        return random;
}
```

```
/** Utility: Try to find the real portlet window ID, by setting a temporary
*  attribute in the local portlet session.
*  Returns null if unable to do so.
*/
public String getPortletWindowID(PortletRequest request){
        String windowID = null;

        // 1. set a PORTLET_SCOPE attribute with a known name, e.g. "retrieveID".
        // To ensure that there will be no interference from other portlets doing
        // the same, randomly generate part of this name.
        double random = MessageHelper.getRandomNumber();
        String att_name = "retrieveID"+random;

        PortletSession session = request.getPortletSession(true);
        if (session==null){ return null; }
        session.setAttribute(att_name, "test", PortletSession.PORTLET_SCOPE);

        // 2. get a list of the attributes in the session, in the APPLICATION_SCOPE
        Enumeration names_enum = session.getAttributeNames(
                                      PortletSession.APPLICATION_SCOPE);

        // 3. find the full (namespaced) name of the attribute that was just set,
        // using PortletSessionUtil to find the one containing the PORTLET_SCOPE name
        while ( names_enum.hasMoreElements() ) {
                String name = (String) names_enum.nextElement();
                if (PortletSessionUtil.decodeScope(name)==PortletSession.PORTLET_SCOPE){
                        String local_name = PortletSessionUtil.
                                                         decodeAttributeName(name);
                        if (att_name.equals(local_name)){
                                // Found the attribute we set.
                                // break down this full name into parts as described
                                // in PLT.15.3 to extract the Window ID: the name
                                // should be of the form
                                // "javax.portlet.p.<ID>?<ATTRIBUTE_NAME>"
                                String prefix = "javax.portlet.p.";
                                String suffix = "?"+att_name;
                                if (name.startsWith(prefix) && name.endsWith(suffix)){
                                        windowID = name.substring(prefix.length(),
                                                name.length() - suffix.length());
                                }
                        }
                }
        }

        // 4. clean up: remove the PORTLET_SCOPE attribute that was set earlier
        session.removeAttribute(att_name, PortletSession.PORTLET_SCOPE);

        return windowID;
}
```

Appendix A2: Discovery Net Papers

For convenience, we include in this appendix three papers which provide further details on projects discussed in this thesis.

- *Sensor Grids for Air Pollution Monitoring:* A paper on Discovery Net's GUSTO scenario, from the All Hands Meeting 2004[152]. M. Ghanem, Y. Guo, J. Hassard, M. Osmond, and M. Richards.
- *Distributed BioSensor systems for GM Crop Monitoring:* A paper on Discovery Net's GM Crop scenario, from the All Hands Meeting 2004[169]. S. Hassard, M. Osmond, F. Pereira, M. Howard, S. Klier, R. Martin, and J. Hassard.
- *Adopting and Extending Portlet Technologies for e-Science Workflow Deployment*: A paper on the limitations of JSR-168 presented at the All Hands Meeting 2005[155]. M. Osmond and Y. Guo.

Additional Discovery Net papers:

- *The Discovery Net System for High Throughput Bioinformatics*[125]*:* A. Rowe, Y. Guo, D. Kalaitzopoulos, M. Osmond, M. Ghanem. ISMB 2003
- *The Design of Discovery Net: Towards Open Grid Services for Knowledge Discovery*[171]*:* S. AlSairafi, F. Emmanouil, M. Ghanem, N. Giannadakis, Y. Guo, D. Kalaitzopoulos, M. Osmond, A. Rowe, J. Syed, P. Wendel. International Journal of High Performance Computing Applications, Vol 17 Issue 3, 2003.
- *Grid-based Analysis of air pollution data*[163]: M. Richards, M. Ghanem, M. Osmond, Y. Guo, J. Hassard. Ecological Modelling Volume 194, Issues 1-3 , 25 March 2006, Pages 274-286 Special Issue on the Fourth European Conference on Ecological Modelling - Selected Papers from the Fourth European Conference on Ecological Modelling, September 27 - October 1, 2004, Bled, Slovenia.