Methodology for ACGT service integration in the ACGT Portal on the Business Process Layer

Project Number: FP6-2005-IST-026996

Deliverable id: D 14.5bis

Deliverable name: Service Integration into the ACGT portal

Date: 30 November, 2008

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	ACGT
Project Full Name:	Advancing Clinico-Genomic Clinical Trials on Cancer: Open Grid Services for improving Medical Knowledge Discovery
Document id:	D 14.5bis
Document name:	Methodology for ACGT service integration in the ACGT Portal on the Business Process Layer
Document type (PU, INT, RE)	RE
Version:	1.0
Date:	30.11.2008
Editor: Organisation: Address:	Sorin Portase SIVECO 8-10 Bd. Maresal Averescu, Bucharest, Romania

Document type PU = public, INT = internal, RE = restricted

ABSTRACT:

The present deliverable presents and analyses the current status of service integration into the ACGT Portal. It also tries to establish some best practices for future service integration.

Part 1 of this document presents the types of integration methods already available in the ACGT portal offering a thorough analysis of the advantages and disadvantages each type of integration has.

Part 2 contains a set of best practices meant to ease the integration of future services inside the ACGT Portal.

KEYWORD LIST: Java Portal, Services, Architecture, JSR168

MODIFICATION CONTROL			
Version	Date	Status	Author
0.1	20.07.08	Draft	Sorin Portase
0.9	21.11.08	Draft	Radu Gramatovici
0.91	28.11.08	Draft	Radu Gramatovici
1.0	30.11.08	Release	Radu Gramatovici

List of Contributors

- Sorin Portase, SIVECO Romania
- Otto Zelch, SIVECO Romania
- Radu Gramatovici, SIVECO Romania
- Andreas Persidis, Biovista

Contents

Table of Figures

Executive Summary

ACGT is an Integrated Project (IP) funded in the 6th Framework Program of the European Commission under the Action Line "*Integrated biomedical information for better health*". The high level objective of the Action Line is the development of methods and systems for improved medical knowledge discovery and understanding through integration of biomedical information (e.g. using modelling, visualization, data mining and grid technologies). Biomedical data and information to be considered include not only clinical information relating to tissues, organs or personal health-related information but also information at the level of molecules and cells, such as that acquired from genomics and proteomics research.

The ultimate objective of the ACGT project is the development of European Knowledge Grid infrastructure offering high-level tools and techniques for the distributed mining and extraction of knowledge from data repositories available on the Grid, leveraging semantic descriptions of components and data and offering knowledge discovery services in the domain of Cancer research.

In the architecture of the ACGT Grid infrastructure, the portal is the gate to the services. Structured on several layers of access rights, the ACGT Portal is offering information and access to all the actors playing in the ACGT project and more general in the domain of Cancer research.

This document addresses the issue of integration of different ACGT services into the portal. ACGT services are software application developed by ACGT members or by third parties that have to be integrated and ran through the portal. The document explains how to develop the interface of these services for a smooth integration in the portal, but also what are the alternatives if a service is already developed and has to be integrated in the portal.

The document is structured in two parts.

Part 1 of this document presents the types of integration already available in the ACGT portal offering a thorough analysis of the advantages and disadvantages each type of integration has.

Several methods, with different degrees of integration have been used so far in the development of the ACGT Portal:

- Installation of an existing portlet
- Wrapping of an existing applet inside a portlet
- Development of a custom portlet from existing code
- Using iFrames
- Development of a custom portlet from scratch.

Part 2 contains a set of best practices meant to ease the integration of future services inside the ACGT Portal.

These recommendations include:

- The use of the Java Portlet Specification JSR 168 for the development of portal interfaces.
- The use of multi-tier architectures for the development of services in order to allow an easy replacement of the original interface of the service with a (set of) portlet(s).
- The use of Apache Portal Bridges in order to overcome the limitations of JSR 168 API and thus the easier and enhanced development of portal interfaces.

PART 1

Types of integration methods available in the ACGT Portal

1 ACGT Portlets

Several methods, with different degrees of integration have been used so far in the development of the ACGT Portal.

Criteria for choosing one or another method included the specificity of the service, the usage needs of the service in the portal and the availability of resources for integrating the service in the portal.

Five methods for the integration of services in the ACGT Portal have been used so far:

- Installation of an existing portlet
- Wrapping of an existing applet inside a portlet
- Development of a custom portlet from existing code
- Using iFrames
- Development of a custom portlet from scratch.

These methods are presented below, together with examples of services that have been integrated accordingly in the portal.

1.1 Installation of an existing portlet

When the service that needs to be integrated has an already developed JSR 168 compliant portlet, that portlet can be deployed in the ACGT Portal unchanged.

Currently, there are two portlets taken from the Gridge Toolkit and used in the ACGT Portal.

1.1.1 Gridge Authorization Service

Gridge Authorization Service (GAS) is an authorization system, which can be the standard decision point for all components of a system. Security policies for all system components are stored in GAS. Using this policies GAS can return an authorization decision upon the client request. GAS has been designed such a way that it is easy to perform integration with external components and it is easy to manage security policies for complex systems. Possibility to integrate with many Globus Toolkit and operating system components makes GAS an attractive solution for grid applications.

1.1.1.1 GAS Features

The main goal of GAS is to provide functionality that would be able to fulfil most authorization requirements of grid computing environments. GAS is designed as a trusted single logical point for defining security policy for complex grid infrastructures. As the flexibility is a key requirement, it is to be able to implement various security scenarios, based on push or pull models, simultaneously.

Secondly, GAS is considered as independent of specific technologies used at lower layers, and it should be fully useable in environments based on grid toolkits as well as other toolkits. The high level of flexibility is achieved mainly through the modular design of GAS and usage of the complex data structure, which can model many scenarios and objects from the real world. It means that GAS can use many different ways for communication with external components and systems, can use many security data models and can hold security policy on different types of storage systems. These features make GAS attractive for many applications and solutions (not only for those related with grids). GAS has to be the trusted component of each system where is used and it brings about that implementation of GAS was written in ANSI C. This choice makes GAS very fast and stable component, which uses not much CPU power and little amount of memory.

1.1.1.2 GAS Portlet

PSNC developed an administration portlet application for GAS, which was deployed into the ACGT Portal.

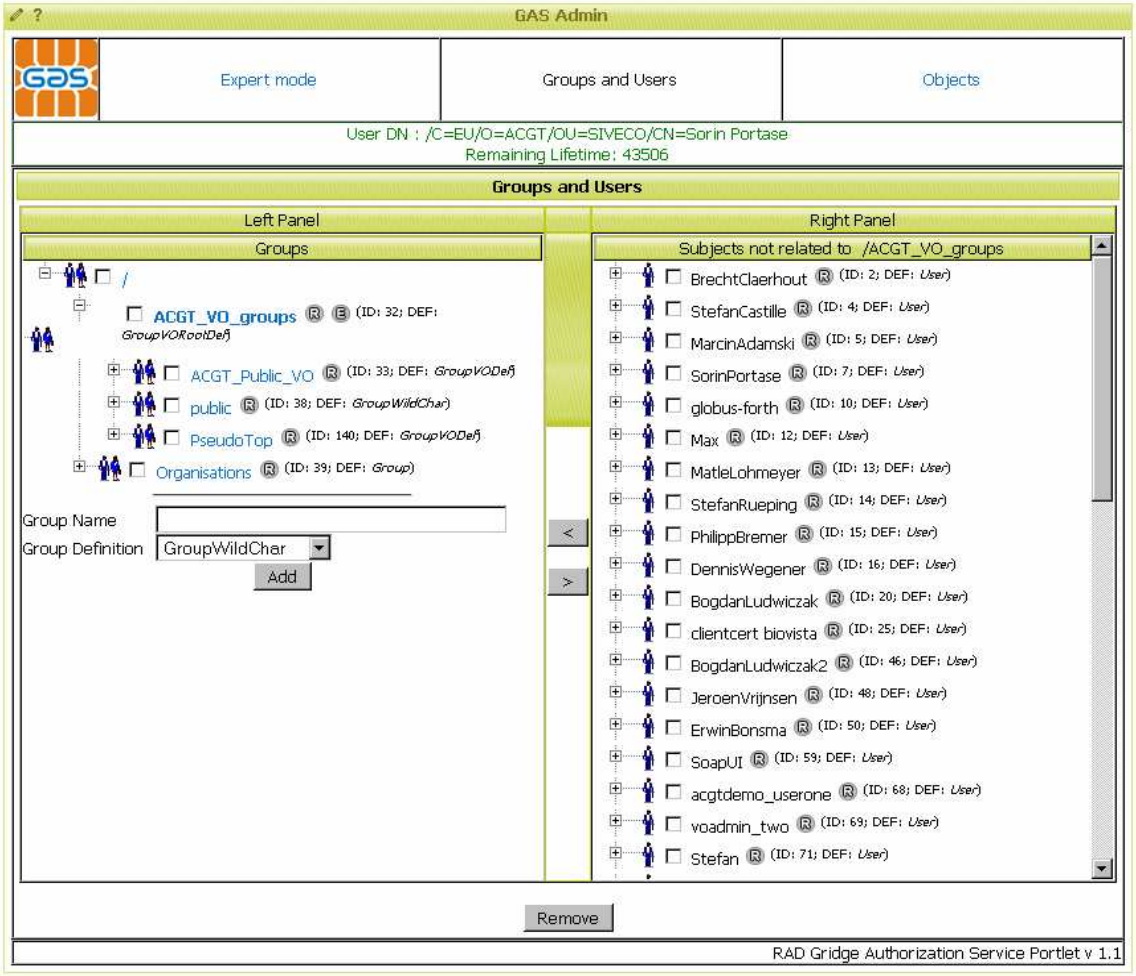

Figure 1: GAS Admin Portlet into ACGT Portal

1.1.2 Gridge Resource Management System

The Gridge Resource Management System (GRMS) is an open source meta-scheduling system, which allows developers to build and deploy resource management systems for large scale distributed computing infrastructures. The GRMS, based on dynamic resource selection, mapping and advanced scheduling methodology, combined with feedback control architecture, deals with dynamic Grid environment and resource management challenges, e.g. load-balancing among clusters, remote job control or file staging support. Therefore, the main goal of the GRMS is to manage the whole process of remote job submission to various batch queuing systems, clusters or resources. It has been designed as an independent core component for resource management processes, which can take advantage of various low-level Core Services and existing technologies. Finally, the GRMS can be considered as a robust system, which provides abstraction of the complex grid infrastructure as well as a toolbox, which helps to form and adapts to distributing computing environments.

1.1.2.1 GRMS Features

GRMS comes in a modular, pluggable and portable architecture, with advanced scheduling algorithms and plugins available for: multicriteria optimization of resource allocations, job re-scheduling (using user-level checkpointing and job migration), time-constraint scheduling, job dependency and workflow management.

Special features of GRMS include:

- Dynamic resource discovery using different information sources to get static and dynamic parameters of machines, queues or jobs in Grid environment.
- Powerfull job description providing wide variety of means to express application and user requirements.
- Framework approach to support different Data Management Systems.
- Complex but flexible workflow job support – precedense constraints based on job status changes.

1.1.2.2 GRMS Portlet

The GRMS Portlet is a graphical user interface client to the GRMS System. The GRMS Portlet was developed by PSNC and deployed into the ACGT Portal.

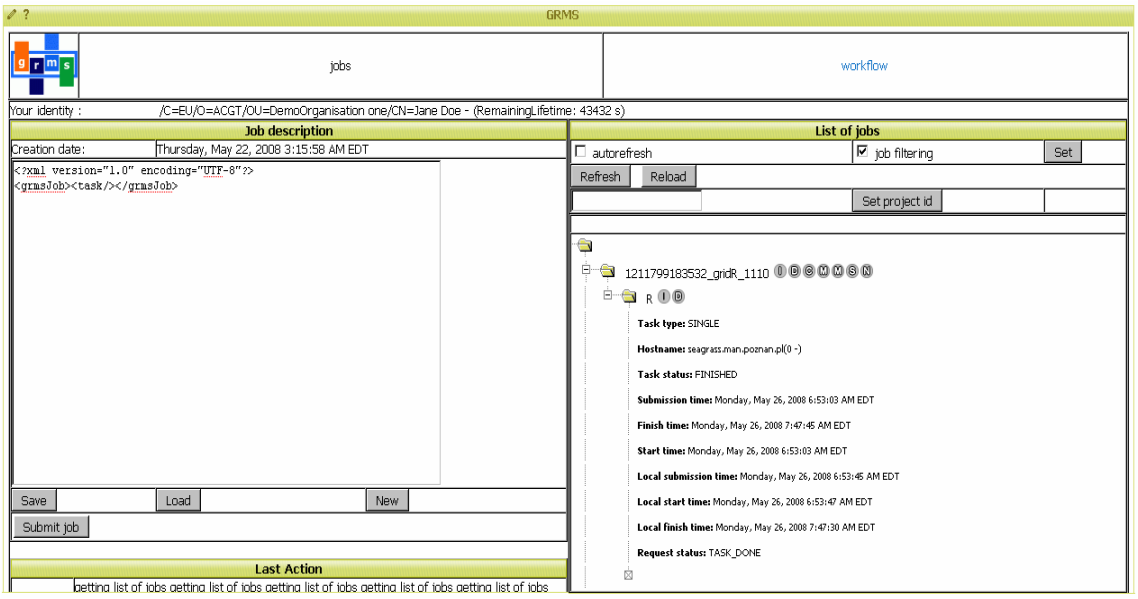

Figure 2: GRMS Portlet into ACGT Portal

1.1.3 Advantages

The reutilization of existent portlets is the ideal solution because time is only spent on configuration. This is made possible by the JSR-168 standard, which provides the interoperability issues between the portlet containers and portlet applications.

1.1.4 Disadvantages

It is very hard to find such portlets that suits your needs perfectly. So most of the time some customization will have de be done. This means that development time will still be needed.

In this respect, it is worth to mention that a VO Management Portlet is being under development by PSNC, with Custodix assistance. This VO Management Portlet will be a variant of the GAS Portlet, simplified and adapted to the specific of the resource management in ACGT.

1.2 Wrapping an existing applet inside a portlet

1.2.1 MyProxy tool

MyProxy is open source software for managing X.509 Public Key Infrastructure (PKI) security credentials (certificates and private keys). MyProxy combines an online credential repository with an online certificate authority to allow users to securely obtain credentials when and where needed. Users run myproxy-logon to authenticate and obtain credentials, including trusted CA certificates and Certificate Revocation Lists (CRLs).

Storing credentials in a MyProxy repository allows users to easily obtain RFC 3820 proxy credentials, without worrying about managing private key and certificate files. They can use

MyProxy to delegate credentials to services acting on their behalf (like a grid portal) by storing credentials in the MyProxy repository and sending the MyProxy passphrase to the service. They can also use MyProxy to renew their credentials, so, for example, long-running jobs don't fail because of expired credentials. A professionally managed MyProxy server can provide a more secure storage location for private keys than typical end-user systems. MyProxy can be configured to encrypt all private keys in the repository with user-chosen passphrases, with server-enforced policies for passphrase quality. By using a proxy credential delegation protocol, MyProxy allows users to obtain proxy credentials when needed without ever transferring private keys over the network.

For users that don't already have PKI credentials, the MyProxy Certificate Authority (CA) provides a convenient method for obtaining them. The MyProxy CA issues short-lived session credentials to authenticated users. The repository and CA functionality can be combined in one service or can be used separately.

MyProxy provides a set of flexible authentication and authorization mechanisms for controlling access to credentials. Server-wide policies allow the MyProxy administrator to control how credentials may be used. Per-credential policies provide additional controls for credential owners. MyProxy supports multiple authentication mechanisms, including passphrase, certificate, Kerberos, Pubcookie, VOMS, PAM, LDAP, SASL and One Time Passwords (OTP).

1.2.1.1 MyProxy Tool Portlet

The MyProxy Tool is an applet developed by Custodix designed for storing user's grid credentials into MyProxy. The applet was wrapped inside a portlet and made available in the public area of the portal.

Figure 3: MyProxy Tool Portlet into ACGT Portal

1.2.2 GridR Session Portlet

1.2.2.1 Use case

The given scenario can be described from the technical point of view as follows: A researcher wants to perform interactive, grid-enabled data mining in the R environment. On

his local machine (client) he develops algorithms using R as user interface. In clinical context, the data to be analyzed are usually so large (~800MB in the present, limited, scenario) that a transfer to the client might be ineffective or not possible. Besides, execution machines in the grid environment might have bigger computational power than client machines. It would be more efficient to just ship the algorithm to the execution machine (the best would be if the execution machine is the machine where the data is located in order to minimize transfer time), execute it remotely on this machine, and transfer the results back to the client.

The R environment is used both as a user interface (client) on the client side and as tool in the grid environment. GridR provides the execution of a single R function or a whole R script (wrapped as function) in the grid. The following subsections give a detailed description on the current implementation of GridR.

1.2.2.2 R as a tool

In ACGT the R environment, namely GridR, is used as a tool for the remote execution of R code in the grid. More specifically, the task of the execution of the R code is submitted as a grid job to a remote grid machine. The current implementation of the server side GridR components that are related to the grid environment is based on several external software components, namely the GT4 grid middleware, an installation of the R environment on the grid machines which will execute the functions remotely and a GRMS-Server installation from the Gridge toolkit on a central machine in the grid environment that is responsible, for instance, for resource management.

On the client side, GridR consists of a set of R functions and involves the Cogkit[11], which is responsible for proxy generation and data transfer, and a GRMS-Client [14].

The client side part is structured around the components “RemoteExecution” (JobSubmission and JobDescription Generator) and “Locking”. The RemoteExecution component is responsible for the execution of R code as a job in the grid environment by transforming the R code to execute into a set of files, creating a job description file in the respective job description language, and submitting the job to the resource management system by the GRMS-client. During this process, the locking component takes care of the consistency of files/variables.

1.2.2.3 R as a client

An R programming language interface that supports the access to the ACGT services is provided in the ACGT environment. This means that R users and developers will have access to distributed resources in a transparent fashion, as if those resources were local. The complexity of the grid is thus hidden from the user.

Again, accessing the ACGT grid environment requires no changes in the core R implementation. In practice grid access is performed through the call of predefined R functions loaded from a package. R users can make use of the grid technology in a transparent way by passing the functions to be executed in the grid as input to one of those predefined functions (grid.apply) in their local code.

1.2.2.4 GridR Session Portlet

TheGridR Session is an applet developed by FhG as a GridR web-based client. The applet was wrapped inside a portlet and made available in the private area of the portal.

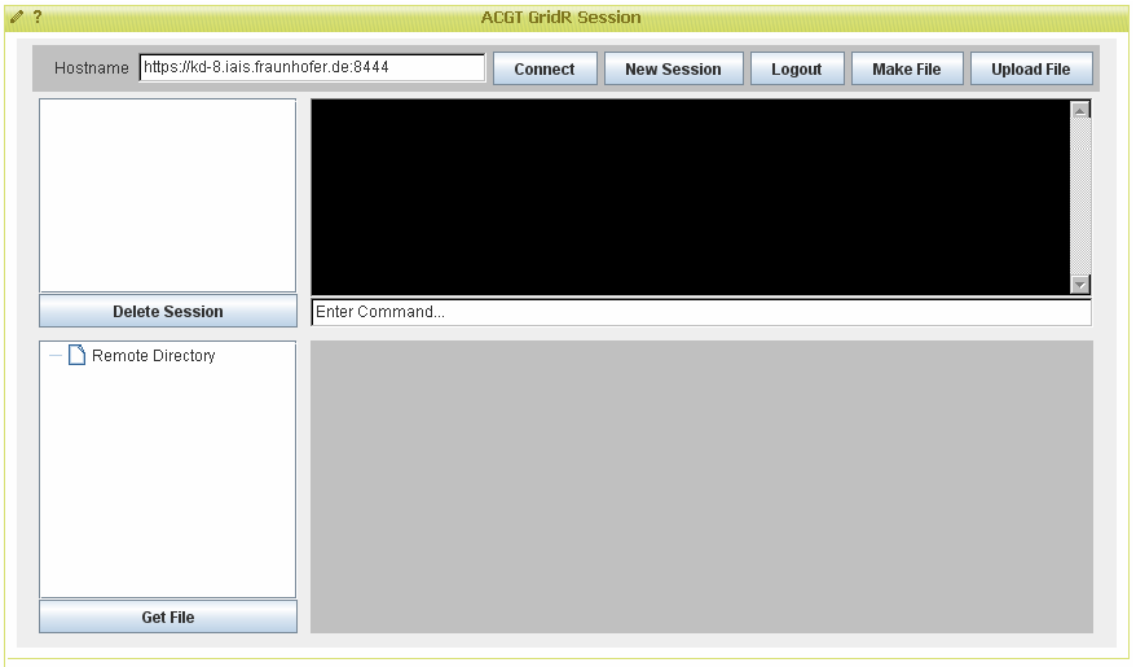

Figure 4: GridR Session Portlet into ACGT Portal

1.2.3 Advantages

The applets allow a richer and a more responsive user interface. Complex applications that otherwise would have to overcome the limitations of the HTTP protocol could greatly benefit from the use of applets.

The technologies involved in applet development are mature and are supported by a large community. Also there are many powerful development tools which make applets easier to develop than web applications.

Another advantage comes from the fact that little or no knowledge of Portlet API is required.

1.2.4 Disadvantages

Applets run on client machine and need additional plug-ins to be installed on client's browser. They also need potential sensitive information to be sent from the portal which can pose problems of security. A workaround is to use a secured transmission channel (HTTPS) between portal and the applet or to try making the transmitted data less sensitive. In our case the credentials sent by the ACGT Portal to the GridR applet have short life span.

Because applets are considered to be potential security sensitive components there are policies that limit their effectiveness. In order to benefit from their full power the code that applet runs must be signed by a certification authority trusted by the user.

The visual integration can be a problem because applets are not aware of portal themes and styles.

1.3 Development of a custom portlet from existing code

Usually every service has some kind of user interface that can become very useful when developing a portlet for that service. This was the case with the DMS Portlet.

1.3.1 Data Management System

Gridge Data Management System is one of the main components of Gridge Data Management Suite (GDMSuite) - a middleware platform providing a uniform interface for connecting heterogeneous data sources over a network. GDMSuite stands for the backbone of the Gridge environment, on which computational services would perform its operations. Gridge Data Management Suite constitutes a bundle of packages, designed for the creation of a complete and robust data management environment. It is intended to fulfill even the enterprise requirements of grid environments in terms of reliability, security and performance.

Data Management System (DMS) is a middleware application, based on SOA model that determines a loose coupling between reusable components, which are platform-, language-, and operating system-independent. Similar to computing and network resources, DMS provides services to manage and retrieve data files in order to support grid jobs. The computational resources managed by DMS can be described by metadata scheme, which allows create an abstract, semantic and explorable layer of resources.

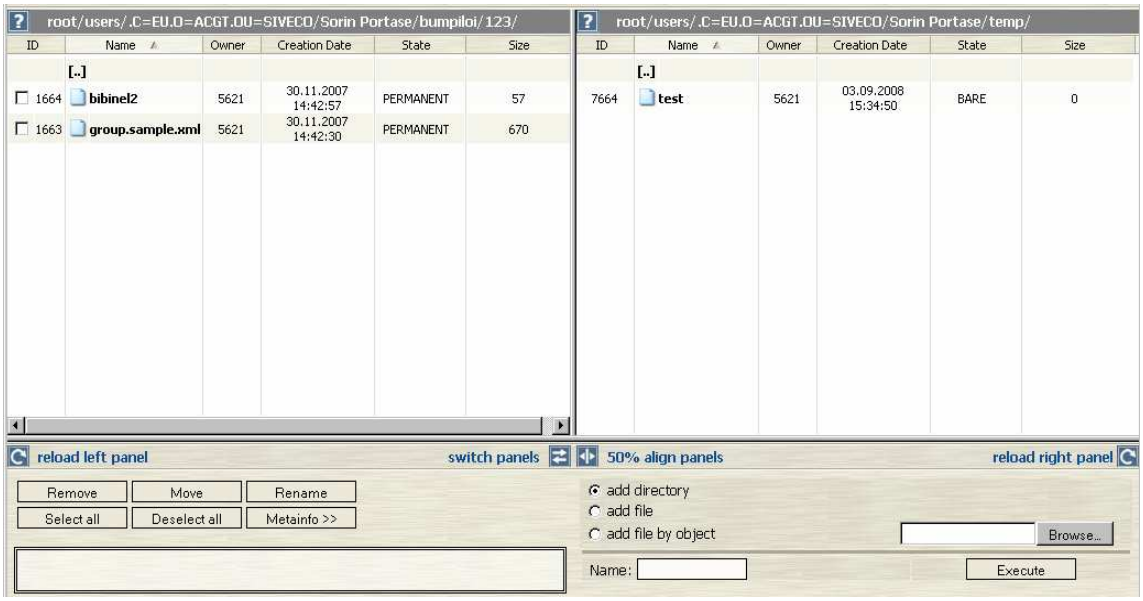

Figure 5: DMS Portal

1.3.1.1 DMS Portlet

The DMS Portlet was developed by SIVECO by emulating all the features available in the DMS Portal. Also the web service layer used by the DMS Portal to access the backend applications was ported into the DMS Portlet.

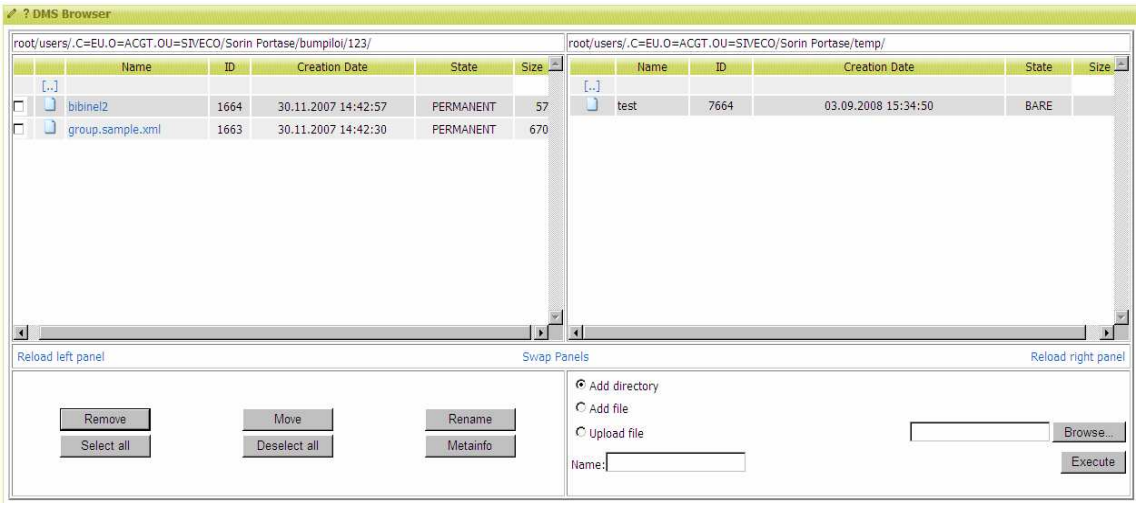

Figure 6: DMS Portlet into ACGT Portal

1.3.2 Advantages

Having an example is always helpful. A lot of time that is to be spent on portlet design and functional analysis can be saved.

1.3.3 Disadvantages

There is still a user interface to develop and usually the development of user interface is the most time consuming activity

1.4 Using IFrames

Sometimes when a service is accessible via a standalone application whose user interface is rather hard to duplicate inside a JSR 168 portlet or when the integration time is short. The quick solution is to wrap that application inside a portlet using an IFrame. It is the case for the ACGT Workflow Editor.

1.4.1 The Workflow Editor

The ACGT Workflow Editor is a graphical tool that allows a user to combine different ACGT services into complex workflows. This tool has a web based graphical user interface developed by FORTH. It supports the searching and the browsing of the available services and data sources and their composition through some intuitive and user-friendly interface. The workflows created can be stored in a user's specific area and later retrieved and edited

so new versions of them can be produced. The publication and sharing of the workflows are also supported so that the user community can exchange information and users benefit from each other's research.

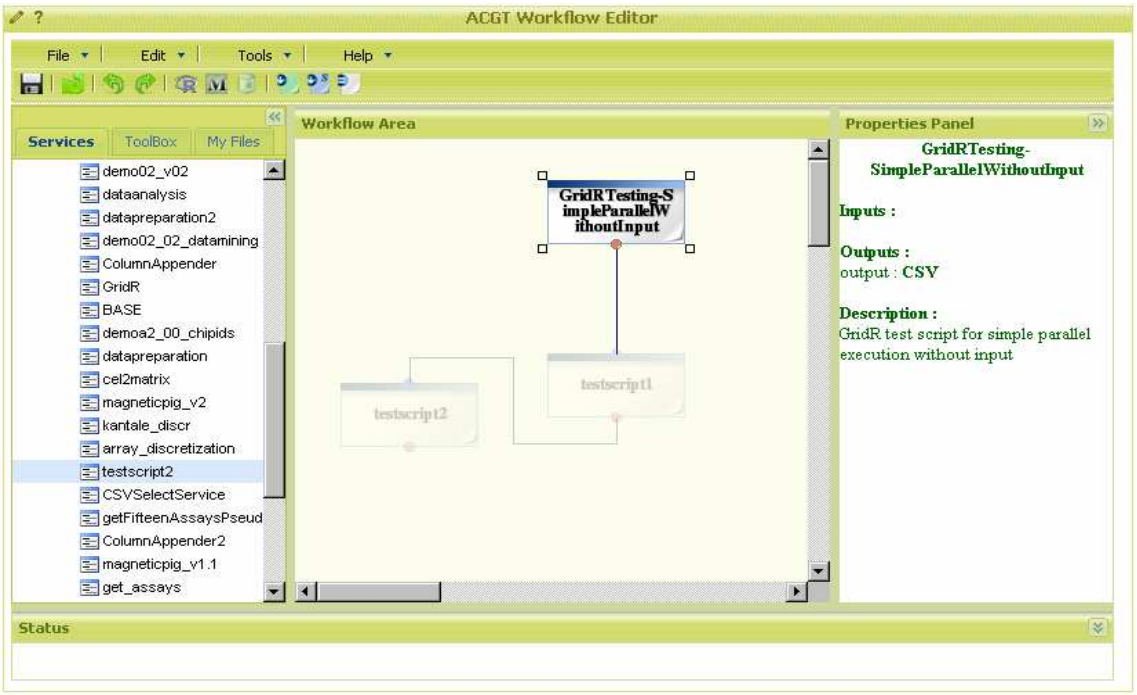

Figure 7: Workflow Editor Portlet into ACGT Portal

1.4.2 Advantages

IFrames are an easy way to include external content. It is very quick and allows legacy applications to be integrated inside the portal

1.4.3 Disadvantages

IFrames undermine the whole portlet principle because the portlet API is just tunneled or side-stepped. Therefore, IFrames should only be used for very special cases, such as surfacing legacy applications. Other potential issues to consider when using an `IFrame` are:

- The IFrame fills its content based on a URL. The URL must be addressable by the browser, therefore the server that is the target of the URL must be accessible by the browser.
- Not all browser levels support `IFrames`.
- If the content is larger than the `IFrame` region, then enable horizontal and vertical scrolling to let the user scroll through the embedded content. Content which contains scrolling regions itself can make it difficult for the end user to manipulate all scrolling regions to view all embedded content, causing usability problems.

- The visual integration and synchronization of the application inside the IFrame with the wrapping portlet and the portal is very hard to accomplish. The portal cannot send theme and style information to the wrapped application which is a serious issue when trying to obtain a uniform look and feel.

1.5 Development of a custom portlet from scratch

When one cannot make use of any situation described above, the only solution is to develop your own portlet solution. The ACGT portal contains two portlet applications that had to be developed from scratch.

1.5.1 Metadata Registration portlets

The user interface to the Metadata Repository are a JSR 168 compliant portlet application that allows a user to browse and edit the Services and Data types stored into the Metadata Repository. It also allows the user to start a service.

The portlets were developed using the available Gridsphere tools and relay on plain JSP and minimal JavaScript for rendering.

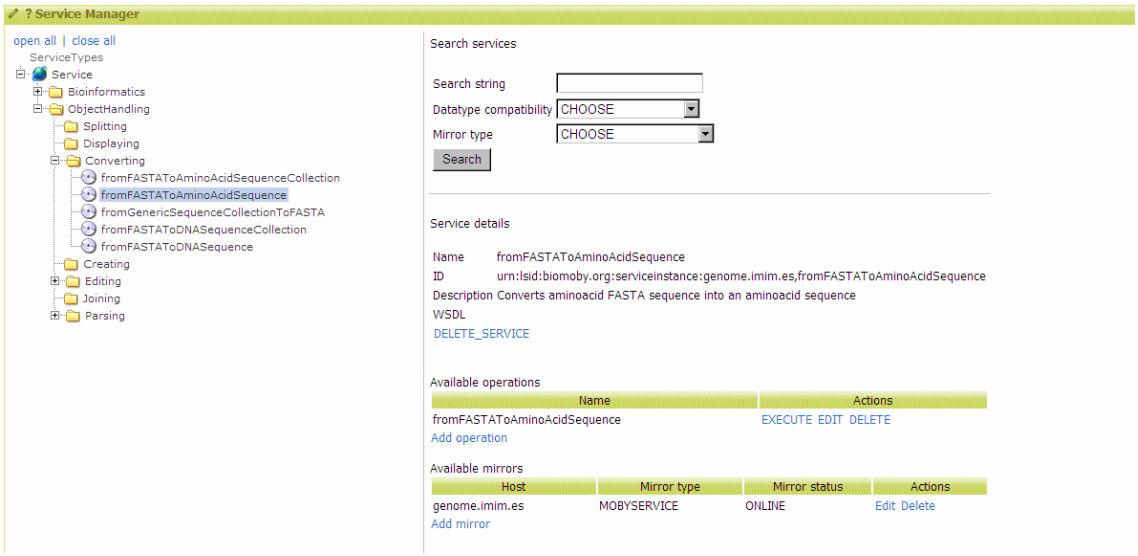

Figure 8: Metadata Registration Portlets into ACGT Portal

1.5.2 Service monitoring portlet

This portlet was designed to offer monitoring capabilities for services started by users using the Service Manager portlet. Its simple design also relies on plain JSP for rendering which gives it robustness. On the other hand the simple approach limits the solutions available for making the user interface dynamic and appealing. This is a common trade-off that has to be addressed when implementing the JSR 168 specifications.

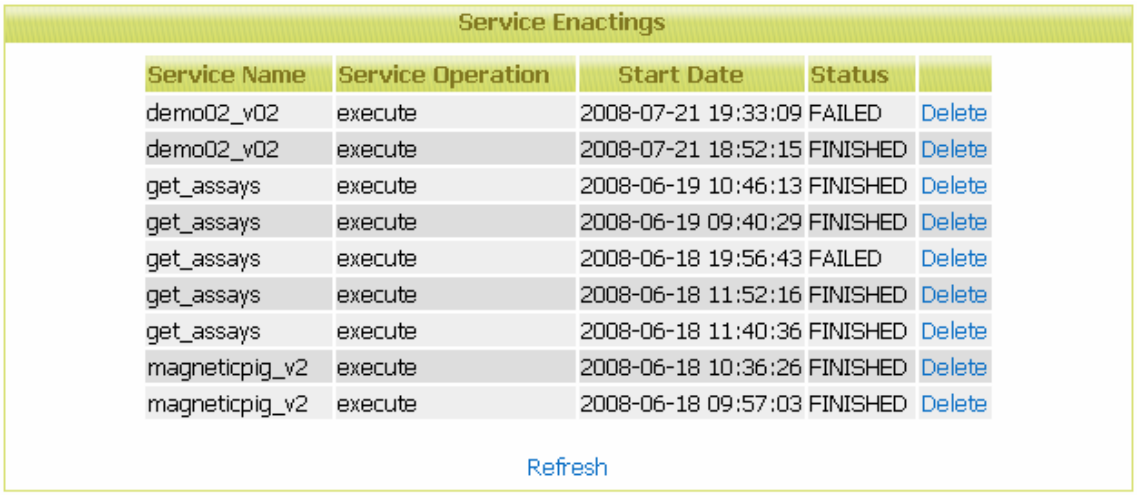

Figure 9: Service Monitoring Portlet into ACGT Portal

1.5.3 Advantages

The main advantage of such a solution is an interface tailored to the exact needs of the portal.

1.5.4 Disadvantages

This solution takes a lot of design and development time. The development time can be improved by using the right technologies but there are still issues concerning the limitations of the current version of the JSR-168 standard. More details about how to speed up development and what tools are available will be given in the second part of the paper.

Another problem is that knowledge of the Portlet API is usually required. As will be seen in the second part of this document, this issue can also be addressed.

1.6 References

[GridR] Dennis Wegener, Thierry Sengstag, Stelios Sfakianakis, Stefan Ruping, Anthony Assi, GridR: An R-Based Grid-Enabled Tool for Data Analysis in ACGT Clinico-Genomics Trials, Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, 2007, 228-235

[WEB_GAS] Gridge Authorization Service Documentation
http://www.gridge.org/content/view/16/97/1/3/

[WEB_GRMS Gridge Resource Management System Documentation
http://www.gridge.org/content/view/18/99/1/3/

[WEB_DMS] Data Management System Documentation
http://www.gridge.org/content/view/19/100/1/4/

[WEB_MyProxy] MyProxy Credential Management Service http://grid.ncsa.uiuc.edu/myproxy/

PART 2

Developing the user interface for a service

2 The Java Portlet Specification

The portlet architecture is an extension to the Java Servlet architecture. Therefore, many aspects of portlet development are common to typical Web application development.

However, the unique aspects of a portal add complexity to the application model, such as:

- Multiple portlets per Web page
- Portlet URL addressing
- Portlet page flow control
- User interface rendering restrictions
- Inter-portlet communication.

You need to carefully consider the design of a portlet to take advantage of portal features instead of falling prey to its complexity. By following the best practices outlined in this document your portlets will be portable, maintainable, and will perform optimally.

2.1 Introduction to JSR 168

The Java™ Specification Request 168 Portlet Specification (JSR 168) standardizes how components for portal servers are to be developed. This standard has industry backing from major portal server vendors.

2.1.1 Advantages of The Portlet Specification

The specification defines a common Portlet API and infrastructure that provides facilities for personalization, presentation, and security. Portlets using this API and adhering to the specification will be product agnostic, and may be deployed to any portal product that conforms to the specification. IT Managers benefit from the ability to support multiple portal products, thus accommodating the unique business needs of various departments and audiences. The compliant portlets can be deployed to all compliant portal frameworks without extensive engineering changes.

For developers, the specification offers code reusability. Developers who want to portal enable their applications can create and maintain one set of JSR 168 compliant portlets. These portlets can be run on any JSR 168 Portlet Specification compliant portal server with few, if any, modifications.

Because this API is specifically designed for portlet creation, developers will benefit from extra functionality beyond the standard servlet functionality. The next section, “What is the Portlet Specification,” discusses these benefits in detail.

This standard also creates a market opportunity for portal tools: IDE’s, test tools, performance measurement tools and reporting interfaces can be marketed to a wider

audience. Portal tool companies can support all compliant portal servers without additional engineering and support costs.

2.1.2 What is The Portlet Specification?

The Portlet Specification addresses the following topics, all covered in this section:

- The portlet container contract and portlet life cycle management
- The definition of window states and portlet modes
- Portlet preferences management
- User information
- Packaging and deployment
- Security
- JSP tags to aid portlet development

2.1.2.1 Container Contract

Much like Java Servlet extensions run inside a servlet container, The Portlet Specification defines a portlet container that manages portlets. A contract is defined for the container to call methods during a portlet's life cycle. The portlet developer can implement these methods to provide the desired functionality.

The life cycle methods called directly by the container are:

- init() Called when the portlet is instantiated by the container. Intended to contain logic that prepares the portlet to serve requests.
- destroy() Called when the container destroys the portlet. Intended to contain logic that cleans up when portlet is no longer needed or the server shuts down.
- processAction() Called after the user submits changes to a portlet. Intended to process input from a user action.
- render() Called whenever the portlet is redrawn by the desktop.

In addition to the methods above, which are called directly by the container, a GenericPortlet class is provided that implements the render() method and delegates the call to more specific methods to display the portlet based on its mode. Developers can extend GenericPortlet and implement as many of these specialized render methods as are necessary for their portlet. These methods are:

- doView() Called by render() when the portlet is in View mode. Intended to contain logic that displays the View page for the portlet.
- doEdit() Called by render() when the portlet is in Edit mode. Intended to contain logic that displays the edit page for the portlet.

- doHelp() Called by render() when the portlet is in Help mode. Intended to contain logic that displays the help page for the portlet.

The processAction(), render(), and specialized methods called by render() accept portlet requests and response objects similar to those passed to the Servlet service() method. Using these objects, the portlet classcan do the following:

- Maintain state and communicate with other portlets, servlets, and JSP's using the portlet session facilities.
- Receive user input from forms displayed in the portlet.
- Create the content for display by the Portal desktop that is sent back to the client via the response object.
- Query portal and portlet state information.

Gridspehre enables sharing of objects between JSP and portlet classes through ActionPortlet model.

```
public class SimpleActionPortlet extends ActionPortlet {

public void init(PortletConfig config) throws UnavailableException
{super.init(config);

          DEFAULT_VIEW_PAGE = "simple/view.jsp";

          DEFAULT_CONFIGURE_PAGE = "simple/config.jsp";

          DEFAULT_EDIT_PAGE = “simple/edit.jsp”;

          DEFAULT_HELP_PAGE = “simple/help.jsp”;

          }

public void doSubmit(FormEvent event) {

          log.debug("in doSubmit");

          TextBean name = event.getTextBean(“name”);

          // output name to Description field of JSP page:

          TextBean desc = event.getTextAreaBean(“desc”);

          desc.setValue(“got a name: “ + name.getValue());

          setNextState(event.getPortletRequest(),
DEFAULT_VIEW_PAGE);

          }

}
```

2.1.2.2 Portlet Mode and Window State

There are two main state items managed by the container for each portlet: portlet mode and window state. This is one of the differences between portlets and servlets.

Each portlet has a current mode, which indicates the function the portlet is performing. As implied by the methods mentioned above, the modes defined in *The Portlet Specification* are *View*, *Edit*, and *Help*. These modes are used by the default render method to decide which lower level display method to call. The mode can also be queried anywhere in the portlet code using the Portlet API to conditionally behave in a manner that is dependent on the current mode.

Window state indicates the amount of portal page space that will be assigned to a portlet. The portlet can use this information to decide how much information to render. The three window states for each portlet are *minimized*, *maximized*, or *normal*.

2.1.2.3 Portlet Preferences

Portlets are often configured to provide a custom view or behavior for different users. These configurations are represented as a persistent set of name-value pairs and are referred to as portlet preferences. In *The Portlet Specification*, the portlet container is responsible for retrieving and storing these preferences through the PortletPreferences interface via the getValues() and setValues() methods respectively.

Portlets have access to the PortletPreferences object when processing requests, but may only modify preference attributes during a processAction invocation. Prior to the end of the processAction method, the store() method must be invoked for changes to be permanent.

To validate preference values, implement the PreferencesValidator class. During runtime, the store() method invokes the validate method of the validator before writing changes to the data store.

2.1.2.4 User Information

The Portlet Specification provides a mechanism for portlets to access user information—such as name, email, phone, and address—in the portlet application's deployment descriptor, and then access them through an modifiable Map object. The Map object can be retrieved through the USER_INFO constant defined in the Request interface. When the Java standard for user information is defined, this mechanism will be discontinued in favor of it.

2.1.2.5 Packaging and Deployment

The Portlet Specification specifies the packaging and deployment of portlets as part of standard Web Application

Archive (WAR) files that may contain other web components, such as JSPs and servlets.

In addition to the web.xml deployment descriptor now found in WAR files, there is an additional portlet.xml descriptor that defines all portlets and portlet-related configurations. An XML schema included in *The Portlet Specification* defines the standard elements for portlet configuration stored in the portlet.xml file.

2.1.2.6 Command line

It is up to each portal server vendor to provide a tool that parses the portlet descriptor file and deploys the portlet. It is likely that these tools will exist in GUI form as well as in command line form. Developers will be able to integrate the command line deployment tools with the Apache Ant tool. This opens up additional development environment possibilities as most development tools provide an interface for Apache Ant.

2.1.2.7 Security

The Portlet Specification includes several features to help developers create secure portlets.

In the portlet deployment descriptor, a flag can be set to restrict the portlet to running only over HTTPS. This is appropriate for portlets that contain confidential information that should be encrypted when sent over the network. In addition, the Portlet API includes authentication facilities for querying user and role information. This allows developers to programmatically control security by introducing business logic based on a user's information and role.

Similar to other Java 2 Platform, Enterprise Edition (J2EE) technologies, *The Portlet Specification* does not address how users and roles are implemented, leaving it to each vendor to apply its own solution. It is likely that the implementations will range from simple file-based user configuration to robust solutions leveraging enterprise directory servers.

The security features addressed by the first version of *The Portlet Specification* are adequate to ensure that portlets can be used for any purpose, regardless of data sensitivity. Nevertheless, future versions will likely have expanded security and authentication features.

2.1.2.8 JSP Tag Library

A JSP Tag library is included to help display portlet pages with JSP technology. For example, a custom JSP tag automatically declares the portlet request and response objects so they can be used within the JSP. Another JSP tag helps construct URL's that refer back to the portlet.

2.1.3 What The Portlet Specification Does Not Address

The Portlet Specification is rather complete in addressing needs for developing individual portlets. However, it is important to remember that portal servers usually offer additional functionality that is not addressed by The Portlet Specification. It is also important for enterprises and service providers to consider such add-ons when deciding on a specific portal server product.

2.1.3.1 Portlet Aggregators

Many developers want to format their portlets on the desktop in a specific way—as tables or tabs on the desktop.

Each portal server vendor will implement its own aggregation and desktop customization features. The Portlet Specification reference implementation provides a basic aggregation framework, which is simpler than those offered by most commercial portal servers.

2.1.3.2 Pre-built Portlets

Many portal server products come with existing portlets that can be deployed and used without any customization. However, besides the GenericPortlet, which is a convenience class for developers, The Portlet Specification does not define any portlets that should be shipped with portal servers.

2.1.3.3 Administration and Portlet Deployment

Most portal server products provide an administrative interface to configure users, groups, security, and other information. These administration interfaces may include facilities for portlet deployment. If not, a separate deployment tool is required for deploying portlets.

Though the specification does stipulate a standard for packaging portlets in a WAR file, each portal server software vendor must provide a mechanism for deploying the WAR files and administering the Portal. Tool vendors that wish to distribute portlets that work with any compliant product should be aware that they must document any product-specific steps to portlet deployment.

2.1.3.4 Common Portlet Functionality

Portlet developers are accustomed to implementing certain functionality across a wide range of portlets—enabling several portlets to identify the user from one login screen, for example. The Portlet Specification and the Portlet API do not address this, but developers can still implement that functionality, using the portal's user principal object.

2.1.3.5 Miscellaneous

Besides those already listed, portal server software vendors will continually attempt to differentiate themselves with new features and product additions. The Portlet Specification, at least in its first iteration, will not address these specialized features.

2.2 Portlet Standard JSR 168 limitations

2.2.1 Missing JSR168 features

The JSR168 specification 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.

2.2.1.1 Portlet Filters

The Servlet API includes support for Filters, which wrap the processing of an incoming request. A filter can inspect the incoming 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 manage and 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.

Servlet filters deployed in a portlet application are not applied to portlets in that application. Filters are only applied to resources in that webapp which are served by the servlet container to a client; this never happens with portlets. Instead, the client requests pages from the portal webapp, and the portal invisibly delegates to portlet classes when it needs them to render.

JSR168 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 workaround below.

Workaround A generic, JSR168-compliant, Filter Portlet has been implemented by Bridges project (currently part of Apache Jetspeed 2.0), which can provide similar functionality to servlet filters. This is done by registering the Filter portlet in the portlet.xml, and parameterising it with the name of the 'real' portlet class. The Filter portlet then acts as a wrapper around the real portlet.

Another option is to use aspect-oriented programming 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 'filter' replacement should be relatively easy to modify to support real portlet filters when it is time to upgrade to the next standard.

2.2.1.2 Inter-portlet Communication

JSR168 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.

Conversely, many Portals make their own detailed provisions for inter-portlet communication, using a variety of models and configuration methods, and have done so since before JSR168 was published. The use of such communication services saves considerable development time, but the portlets developed are then portal-specific.

Workaround: A library could be implemented for portlet messaging which works within the limitations of JSR168. This enables developers to write communicating portlets easily without

needing to implement the messaging system as well, thus partially levelling the field between JSR168 and portal-specific messaging. Of course, an arbitrary third-party portlet will still not be able to participate in such communications without being rewritten to use this messaging library - but that level of compatibility must wait for an official inter-portlet communication API in the next specification.

2.2.1.3 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 JSR168, 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 JSR168 portlet is unable to do, including:

- modify page layouts, e.g. add a new portlet to a page.
- add or remove pages
- manage users and roles
- set page or portlet access permissions
- find out what other pages there are on the site, and link to them
- find out what other portlets exist on a page, or on the whole site
- get a unique ID for its own "portlet window"

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.

However, the remaining feature - knowing a portlet window's own ID, preferably one which persists across sessions - is a much simpler one which can be critical when developing portlets as communicating components. This is particularly relevant when there are multiple windows showing the same portlet (the concept of portlet windows is discussed later in this chapter). The Portal itself will have assigned an internal ID for each portlet window, to be able to store and retrieve portlet preferences etc., but JSR168 provides no simple way to access this ID from within the portlet. The method `RenderResponse.getNamespace()` may be suitable in some situations, but only provides a namespace for the portlet window which is

unique for the current page (PLT.12.3.4) - it is not guaranteed to be unique across the whole site, or even consistent across different requests in the same session.

Workaround It is possible to generate a 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. 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 might be to make 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, it would be able to retrieve the portal's unique ID for the portlet window as follows:

- 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.
- get a list of the attributes using PortletSession.getAttributeNames(APPLICATION_SCOPE)
- find the full (namespaced) name of the attribute that was just set, by iterating through the list and using PortletSessionUtil to find the one containing the PORTLET_SCOPE name
- break down this full name into parts as described in PLT.15.3 to extract the Window ID: "javax.portlet.p.<ID>?<ATTRIBUTE_NAME>"
- store this portlet window ID for later retrieval (e.g. in a PORTLET_SCOPE session attribute 'windowID')
- clean up: remove the PORTLET_SCOPE attribute that was set earlier

2.2.1.4 Session ID

Sometimes it is useful for portlets to know the ID of the user's whole portal session, not just the IDs of the individual portlet application sessions (which can be retrieved with `PortletSession.getId()`). This is particularly so when portlets in different applications need to communicate. The ideal identifier is simply the Portal application's own session ID - but this session, and its ID, are not directly accessible to portlets. The JSR168 method `PortletRequest.getRequestedSessionId()` can almost provide equivalent functionality, as it returns the session ID included in the client request. However, unlike `HttpSession.getId()` or `PortletSession.getId()`, this approach does not work with newly-created sessions - i.e. the very first page render. This can be inconvenient if as part of session creation the portlet needs to set up external resources reliant on the portal session id (like external message stores).

Workaround Apart from using `PortletRequest.getRequestedSessionId()`(and having to account for its absence on the first page visited), two approaches have been investigated: setting a session ID in a cookie visible to all portlets (which also has the first-page problem, and is additionally quite unreliable), or modifying open source portals to pass on the Portal's session ID in the `PortletRequest`. None of these are ideal, but we have not found any other solutions so far.

2.2.1.5 Cookies

Cookies are (usually) small pieces of information stored on the client by the 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 site, the browser includes any cookies set by that site as a header in the request.

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 cookie Header value 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. JSR168 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.

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 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. 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 header during this phase.

Workaround Without support in JSR168 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 directly by the client, not dispatched to through 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, so the cookie will only be available to portlets on subsequent requests.

2.2.1.6 '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. (Some browsers permit binary objects to be embedded in HTML, but 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).

As an alternative, some Portals have chosen to make available an 'Exclusive' mode in which a portlet generates the entire response, not just a fragment. This approach is portal-specific but quite popular, and may also have relevance when considering the use of AJAX in portlets (discussed below).

2.2.1.7 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 admin or security purposes.

However, 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 to a servlet - JSR168 specifically prohibits the `ServletRequest` methods concerned from working in this context

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, and at least one page render before the portlets would have access to the client IP.

2.2.2 Further Issues

Many portlet programmers have encountered issues which are not missing features, but areas of possible confusion often due to differences between portal implementations.

2.2.2.1 Portlet Windows/Instances

Portlets are defined with entries in the `portlet.xml` deployment descriptor. WBJSR168] 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 service 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 entities to operate independently. For example, a 'Stocks' portlet might be configured to show a report for different stocks depending on the page; a portlet which could be pointed at any WSDL file to generate an interface to a web service might be present on different pages,

providing access to web services appropriate to the page; a content management portlet would be set up to display different content on different pages.

JSR168 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." [WBJSR168] The details of creating and managing portlet windows are left to the Portal and Portlet Container.

This leaves several aspects of the concept of a 'Portlet Window' undefined or ambiguous, including:

- Can there be different Portlet Windows of the same portlet, which are associated with different preferences objects?
  - on the same page?
  - on different pages?
- Do different Portlet Windows also maintain associated, separate PORTLET_SCOPE Sessions and render states (from Action requests) as well as preferences?

The answer to the latter question is usually assumed to be 'yes', but the first is more variable. Different portlet containers and portals are implemented using different assumptions.

The strictest approach 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, multiple definitions in the `portlet.xml` for the 'same' portlet (merely differing in portlet name) must be added if multiple independent portlet windows are required.

The most flexible 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.

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, and the way site designers and users add portlets to pages. For dynamic sites which allow users to create pages, the strict intepretation may not be sufficiently flexible, depending on the expected level of portlet reuse.

2.2.2.2 The `PortletSession` and the `HttpSession`: Cross-context handling

According to JSR168 (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 clearly allows for 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), or details of the logged-in user (which the servlet may use to confirm that the request has been suitably authenticated).

However, there are implementation issues which mean that this behaviour is not found in some servlet container/portal setups. In such cases, a servlet which is accessed directly (e.g. one displayed in an IFRAME in a portlet, or in a new window) will not see the contents of the `PortletSession`, but instead its own separate `HttpSession`. This `HttpSession` is in fact the session associated with the portlet's (and the servlet's) webapp, whereas the `PortletSession` is associated with the **Portal** webapp, and these are kept separate following the Servlet 2.3 specification (SRV.7.3). Then, the only way the portlet can send information to the servlet is by appending URL query parameters, which may not be appropriate or possible for large amounts of data. If on the other hand the servlet is included in a portlet's output through the portal, using (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 JSR168.

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.

2.2.2.3 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 technology uses JavaScript to asynchronously fetch a web page in the background, reading in and using the returned data to modify the page seen by the user. This requires an endpoint on the server - usually a dynamically-generated page which takes query parameters - for the JavaScript to fetch. In the context of Java web servers, this endpoint would be a servlet or a JSP.

Portlets may wish to use AJAX in their interfaces - 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 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.

Another potential problem - assuming that the servlet and portlet do see the same session - is that the servlet will only be able to see and set attributes in the APPLICATION_SCOPE portlet session. Thus AJAX calls to a servlet will not be able to modify session attributes belonging to that portlet window, or indeed access any portlet-specific functionality such as initialisation parameters or preferences.

Thus AJAX calls which access portlet functionality (e.g. PORTLET_SCOPE session, initialisation parameters, preferences) are not currently practical using JSR168. To enable this, the portal would need to provide an endpoint for a portlet window which would act in a similar way to the 'Exclusive' mode described earlier: the portlet, in a special render mode, would be able to output the entire page. Then, an AJAX call would not need to access a servlet, but instead could query the portlet 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 revists the portal page. Instead, this usage pattern might require either the ability to specify a portlet mode change for a single request, or a new portlet request type alongside 'render' and 'action'.

2.2.2.4 Authentication Integration: JAAS

Each J2EE application server has a different way of configuring new 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 J2EE applications.

2.2.2.5 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 and Struts.

Conflicts with commons-logging or log4j 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.

The issue with Apache Struts is more complicated. Struts is commonly used as the framework for servlet applications using the Model-View-Controller (MVC) design paradigm, but is not yet directly compatible with the portlet. The Apache Struts Bridge was designed to address this issue.

2.2.2.6 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)` v.s. `PortletSession.getAttribute(name)`). In addition, the problem may arise when using common libraries which do not (yet) provide a portlet.

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`).

One generic, JSR168-compliant solution is to write a number of simple wrappers or adapters, so that a Portlet object can masquerade as an HTTP Servlet object (or vice versa). 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.

3 The multi-tier architecture

As seen with the DMS portlet in section 1.3, a multi-tier architecture may facilitate the modification of a Java Web Application for the integration in a portal. Basically, this modification consists in the replacement of the original interface of the application with a new interface designed as a (set of) portlet(s).

Multi-tier architecture (often referred to as *n-tier architecture*) is a client-server architecture in which, the presentation, the application processing and the data management are logically separate processes. For example, an application that uses middleware to service data requests between a user and a database employs multi-tier architecture. The most widespread use of "multi-tier architecture" refers to ***three-tier architecture***.

Apart from the usual advantages of modular software with well defined interfaces, the multi-tier architecture is intended to allow any of the tiers to be upgraded or replaced independently as requirements or technology change.

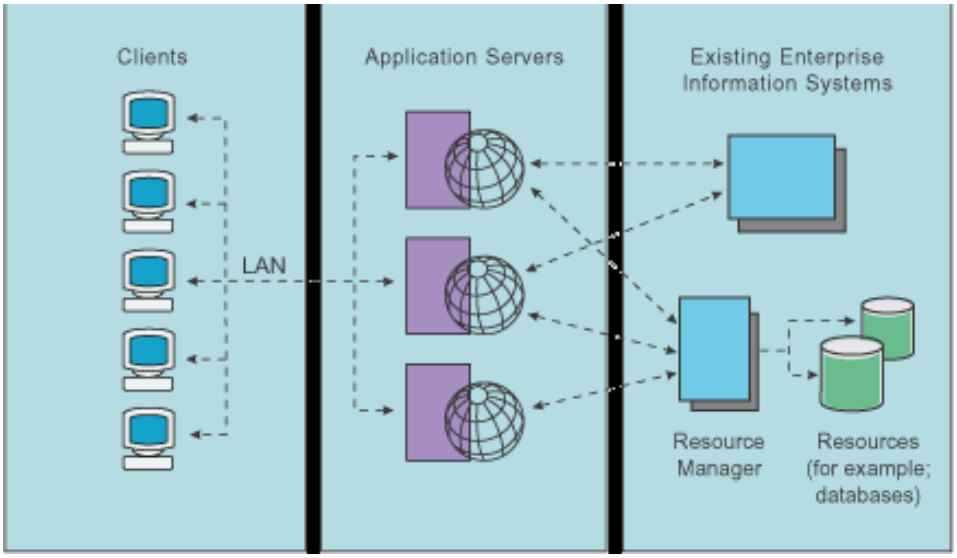

Figure 10: A standard three tier architecture

First tier

Responsibility for presentation and user interaction resides with the first-tier components. These client components enable the user to interact with the second-tier processes in a secure and intuitive manner. Clients do not access the third-tier services directly. For example, a client component provides a form on which a customer orders products. The client component submits this order to the second-tier processes, which check the product databases and perform tasks that are needed for billing and shipping.

Second tier

The second-tier processes are commonly referred to as the application logic layer. These processes manage the business logic of the application, and are permitted access to the

third-tier services. The application logic layer is where most of the processing work occurs. Multiple client components can access the second-tier processes simultaneously, so this application logic layer must manage its own transactions.

If several customers attempt to place an order for the same item, of which only one remains, the application logic layer must determine who has the right to that item, update the database to reflect the purchase, and inform the other customers that the item is no longer available. Without an application logic layer, client components access the product database directly. The database is required to manage its own connections, typically locking out a record that is being accessed. A lock can occur when an item is placed into a shopping cart, preventing other customers from considering it for purchase. Separating the second and third tiers reduces the load on the third-tier services, supports more effective connection management, and can improve overall network performance.

Third tier

The third-tier services are protected from direct access by the client components residing within a secure network. Interaction must occur through the second-tier processes.

Communication among tiers

All three tiers must communicate with each other. Open, standard protocols and exposed APIs simplify this communication. You can write client components in any programming language, such as Java or C++. These clients run on any operating system, by speaking with the application logic layer. Databases in the third tier can be of any design, if the application layer can query and manipulate them. The key to this architecture is the application logic layer.

The current ACGT architecture can be seen as a five-tier architecture, as described by the deliverable D3.1:

- Hardware Layer
- Common Grid Infrastructure Layer
- Advanced Grid Middleware
- ACGT Business Process (Model) Services
- User Access Layer

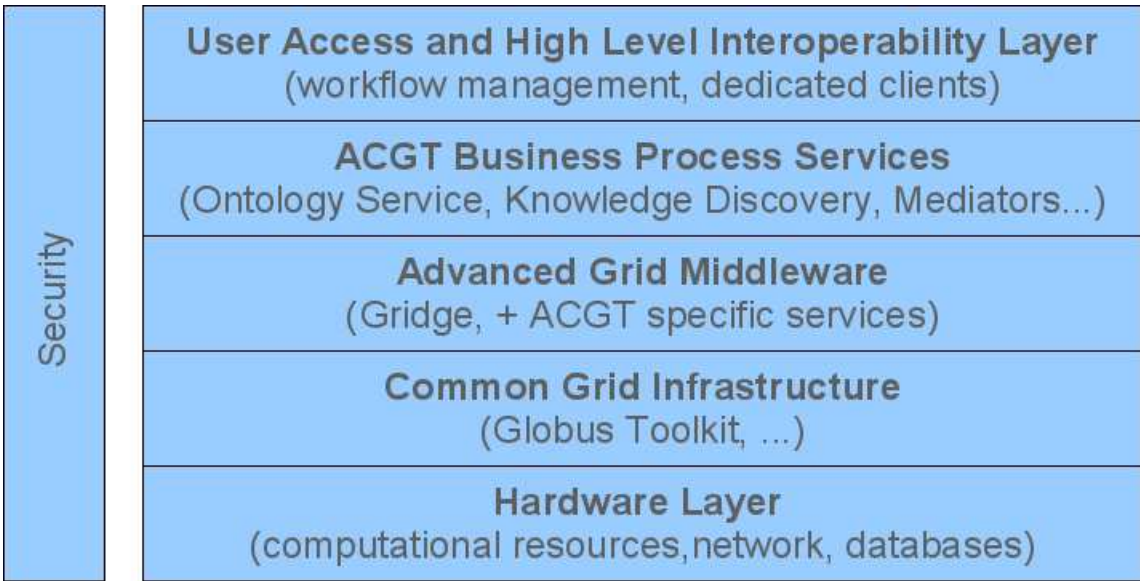

Figure 11: ACGT (initial, simplified) architecture overview

Being complex enough, each tier can be considered multi-tiered itself. This is the case with the User Access Layer, where the ACGT Portal internal architecture can be further split into logically independent layers:

- The application server
- The portlet container
- The deployed portlets

As stated in the JSR 168 specification, the portlets are deployed and packaged just like standard Java Web Applications. Also portlet applications and standard web application are structurally very similar. Hence one can further split a portlet application in multiple layers, which are identical with the standard three-tier approach adopted by many standalone web applications.

4 Apache Portal Bridges

To overcome the limitations of the Portlet API 1.0 in terms of development ease and in order to leverage the existing web technologies the Apache Portal Bridges Project offers a set of components called “Bridges” that aim at offering a transparent way of integration of standalone applications inside a JSR-168 compliant Portlet Container

The portlet bridge acts as the translation engine between the portlet environment and the targeted system. It expresses the portlet request context to the bridged environment, executes the request and transforms the response as appropriate for the portlet environment. It abstracts the details of running in a portlet environment from the bridged environment and more importantly the developers building views within that environment.

In the sequel, we present how this technology may help the development of portal interfaces for web applications.

4.1 JSF Bridge

Java Server Faces (JSF) applications can be run within a Portlet environment with little or no changes using the JSF bridge which provides a JSF Portlet ready to be deployed. Similar functionality is also available from the Apache MyFaces PortletBridge. This bridge is the reference implementation for JSR 301: Portlet Bridge for JavaServer Faces

4.1.1 What is Java Server Faces

The Java Server Faces Specification (v1.2) describes itself as follows in its Overview chapter:

Java Server Faces (JSF) is a user interface (UI) framework for Java web applications. It is designed to significantly ease the burden of writing and maintaining applications that run on a Java application server and render their UIs back to a target client. JSF provides ease-of use in the following ways:

- Makes it easy to construct a UI from a set of reusable UI components
- Simplifies migration of application data to and from the UI
- Helps manage UI state across server requests
- Provides a simple model for wiring client-generated events to server-side application code
- Allows custom UI components to be easily built and re-used

Most importantly, JSF establishes standards which are designed to be leveraged by tools to provide a developer experience which is accessible to a wide variety of developer types, ranging from corporate developers to systems programmers. A “corporate developer” is characterized as an individual who is proficient in writing procedural code and business logic,

but is not necessarily skilled in object-oriented programming. A “systems programmer” understands object-oriented fundamentals, including abstraction and designing for re-use. A corporate developer typically relies on tools for development, while a system programmer may define his or her tool as a text editor for writing code. Therefore, JSF is designed to be tooled, but also exposes the framework and programming model as APIs so that it can be used outside of tools, as is sometimes required by systems programmers.

JSF being a much more popular and mature technology than Java Portlets has a considerably larger and more active support community. Using this bridge, the ACGT Portal can benefit from the tools and support available for JSF web application development. In this way theoretically it is also possible to develop portlet applications with minimal knowledge of JSR 168 API details.

The JSF technology provides a mature and extensible user interface component model. The design of this model makes it easy for application developers to create custom components by extending the standard components included with JSF and to reuse these components across applications. Creating a component shields the page author from the complexities of AJAX by rendering all the HTML and JavaScript code that is required to incorporate AJAX capabilities into an application.

Developers who want to include AJAX support in JSF applications have more than one strategy to choose from. One point to stress is that the programming model differs between component writer and page developer. The strategies can be used to add AJAX support to just a part of your application or to create re-useable JSF components. Though it is a bit of extra work to create an AJAX enabled JSF component, it makes it very easy for page authors to re-use that component.

In the first strategy, using a PhaseListener to fulfil AJAX requests in JSF components, the JSF life cycle and the custom component handles or decorates the AJAX requests. The second strategy, using a Servlet to act as a controller, introduces a Servlet to process AJAX requests to be used with the JSF components. In the second strategy, all asynchronous requests will therefore be handled outside of the JSF life cycle processing. In general, the first strategy is more common and the second strategy is more of a special case.

Both approaches can be used in the following cases:

- Legacy applications that aren't JSF based could use this approach because the data that may be affected by the AJAX call would be managed by a legacy mechanism. Adding AJAX functionality in a tied approach, can limit liability to an existing production application

- When the AJAX call doesn't affect the JSF view state that is associated with the web page.

- When the component developer is not willing to pay the performance hit of saving, transferring and restoring component view state. For example, this could be the case when a polling action is performed through an AJAX call.

- When the call retrieves read only data.

4.1.2 Architectural Overview

The following diagram depicts the overall architecture of Java Portlet Application containing portlets that use the bridge to execute Faces pages to generate its portlet output.

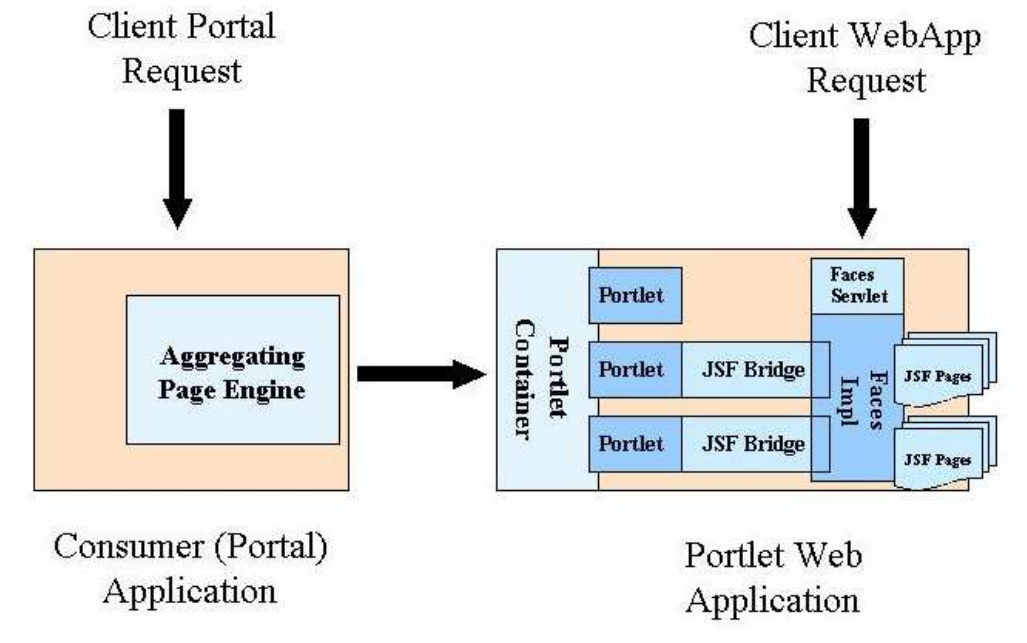

Figure 12: Portlet Web Application Architecture containing JSF Portlets

This diagram shows two entry points to the portlet web application.

The entry point depicted on the right (Client WebApp Request) depicts the usual execution of a Faces application. The client makes a direct (browser/http) request to the web application to execute/present a Faces page. The FacesServlet receives this request and executes the targeted page.

The entry point depicted on the left (Client Portal Request) depicts the corresponding execution of the Faces application when its incorporated into a consumer (portal) application as a portlet. In this case the client sends her request to the consumer (portal) web application, which resolves the request to one of its pages containing in part one or more portlets.

The consumer executes a page engine that composites the client response from its own mark-up and that supplied by contained portlets. The consumer executes and renders the portlet by making calls to the portlet container that hosts the portlet. The portlet container, in turn resolves the consumers request to a targeted portlet and executes that portlet according to the Java Portlet 1.0 APIs/specification. For each portlet whose behaviour is implemented using the Faces technology, the portlet activates and executes its own instance of the bridge by using the APIs defined in this specification. The bridge provides the intermediation that translates between portlet and Faces semantics such that the portlet/portlet container needs to have no knowledge about executing the Faces page and the Faces page needs to have no knowledge its executing in a portlet environment.

For each request the bridge sets up the necessary Faces environment and executes the Faces lifecycle causing the Faces page/view to be processed / rendered. Part of this implementation acts as Faces controller much as the FacesServlet does in the direct client request world. The other part of this implementation is provided by implementating a variety of (standard) Faces extensions.

The end result is a portlet web application that presents one or more Faces page collections (distinct grouping of pages that might otherwise be represented as discrete applications) directly to an end user via the servlet container and indirectly to the user as a portlet contained in another application's page. Most such page collections should run unchanged whether executed in only one or both of these situations. They can even be executed together concurrently.

4.2 Other technologies covered by Apache Bridges

4.2.1 Struts and PHP

Pure servlet based frameworks like Struts or PHP 4 (with the Java binding) are very difficult to use in a Portlet environment because they only support and understand a single request event (JSR-168 describes two portlet phases: action handling and response render).

For a Portlet, the request processing has to be handled in two separated action and render events. Furthermore, the Portlet Specification doesn't allow servlet access during the action request event.

A distinctive feature of the Portals Bridges is that it removes these barriers and thereby allows the mapping of an action request event to the underlying servlet handler implementation (e.g. a StrutsAction).

All that is needed is an implementation of the generic ServletContextProvider interface, provided with The Portals Bridges Common library, for a specific Portal.

The Portals Struts Bridge furthermore provides a truly transparent mapping of the two request events onto the underlying Struts Action mappings and enables an existing Struts Application to be run within a Portlet context with very little changes.

4.2.2 Velocity

Although Java Server Pages (JSP) are very much promoted for Portlet View rendering, other technologies can be used just as easily. The Portals Velocity Bridge provides several ready to use Velocity based Portlets, together with a library of useful scripts.

4.2.3 Groovy

The Groovy Portlet Bridge allows using groovy scripting for rapid portlet development. A groovy script portlet can be implemented like Java-based portlets, and so developers can use any techniques used in Java. For example, a groovy script portlet can extend org.apache.portals.bridges.common.GenericServletPortlet to simplify implementation. Therefore, GroovyPortlet is very simple, but it will be very useful for developers.

4.2.4 Perl

Perl based web applications can be accessed as a Portlet using the Portals Perl Bridge.

4.3 Common Utilities and Interfaces

The Portals Bridges Common library provides, besides the already mentioned ServletContextProvider interface, several additional utilities like a GenericServletPortlet which you can use for simple JSP based Portlets (or any invokable view Render technology). Or use it as base class for your own Bridge.

Other useful utilities included are a PreferenceHelper and a simple PortletMessaging component.

4.4 Portlet Filter

The Portlet Filter provides a FilterPortlet which enables you to add processes to run before and after accessing the target portlet. The feature provided by FilterPortlet is like a Servlet Filter. (Portlet Filter may be included in the future Portlet Spec, but currently it's not included yet.)

4.5 References

[NOV2003]	J. Novotny, M. Russell, O. Wehrens, GridSphere: An Advanced Portal Framework, 2003
[WEBSTR]	Jakarta Struts framework HTTP://JAKARTA.APACHE.ORG/STRUTS/INDEX.HTML
[WEBABR]	APACHE PORTAL BRIDGES HTTP://PORTALS.APACHE.ORG/BRIDGES/
[WEBSPR]	Spring framework HTTP://WWW.SPRINGFRAMEWORK.ORG/
[WEBSER]	Java Servlet 2.3 and Java Server Pages 1.2 Specifications HTTP://JAVA.SUN.COM/PRODUCTS/SERVLETS/
[WEBHEP]	COMPARING THE JSR 168 JAVA PORTLET SPECIFICATION WITH THE IBM PORTLET API
[WEBIBM]	What is a Portlet? HTTP://WWW-3.IBM.COM/SOFTWARE/WEBSERVERS/PORTAL/PORTLET.HTML
[WEBJSR168]	JSR 168: Portlet Specification HTTP://WWW.JCP.ORG/EN/JSR/DETAIL?ID=168
[WEBJSF]	Java Server Faces http://java.sun.com/javaee/javaserverfaces/
[WEBJSR301]	JSF Portlet Bridge Specification http://www.jcp.org/en/jsr/detail?id=301

[WEBAJAX] Using JavaServer Faces Technology with AJAX
https://blueprints.dev.java.net/bpcatalog/ee5/ajax/usingJSFwithAJAX.html