

CCLRC Portal Infrastructure to support Research Facilities

Asif Akram, Dharmesh Chohan, David Meredith and Rob Allan
e-Science Centre, CCLRC Daresbury Laboratory

1. Introduction

The CCLRC e-Science Centre is working on a number of UK e-Research projects supporting access to large research facilities such as the neutron spallation source (ISIS), the Synchrotron Radiation Source (SRS) and the Central Laser Facility (CLF) and the new synchrotron light source, Diamond in addition to the National Grid Service and the HPCx supercomputing facility. The aim of this work is to create an Integrated e-Science Environment for CCLRC. A clear requirement of facility users is to provide seamless access and integration of these resources, which can be done by developing portal interfaces for each facility or project exposing their services as portlets for re-use between the different portal frameworks and CCLRC projects. This paper outlines the benefit of using portal frameworks and portlets to meet this goal with a Service Oriented Architecture designed to complement desktop tools. An example is given of the portal services being provided for the National Grid Service (NGS) and e-HTPX portals, which include computational and data Grid production resources for research.

2. Portal Frameworks and Portlets

Portal frameworks and Portlets are relatively new technologies with improving specifications and enhanced support from both open source and commercial software companies. A portal is a Web-based application that acts as a gateway between users and a range of different high-level services. It provides personalisation, single sign-on (SSO), aggregation and customisation features. A so-called 2nd generation portal normally consists of different portlets used to process consumer requests to these services and generate dynamic content from the responses. Multiple portlets can be hosted within a single portal as self contained pluggable user interface components (i.e. providing aggregation of different functionality). Portlet interfaces can be developed in different languages but at CCLRC the portal interfaces are mainly based on Java technology and managed by a Java portlet container. Java portlets adhere to the Java Specification Request 168 Portlet Specification (JSR 168), which standardises the interoperability of portlets between different portlet containers. JSR 168 compliant portlets are therefore container and framework independent and can be deployed within any container that adheres to JSR 168 specifications, see [1] and [5]. We emphasise that, because of the use of a standard interface portlets are not confined to one portal framework.

3. Implementation and Standards

The JSR 168 specification [3] is based on the mature servlet standard following a community review in 2003. The behaviour of portlets is similar to that of servlets in many ways, i.e. both portlets and servlets are Java-based Web components, managed by a Web application container, used to generate dynamic content and interact with Web clients via a request/ response paradigm. Unlike servlets, portlets have additional features and also limitations, for example, portlets only generate markup fragments and have pre-defined modes and states, but there are optional extensions allowed. Portlets are compatible with J2EE thus providing additional capabilities to the typical Service Oriented Architecture (SOA). As described above portlets provide persistence to the SOA either through servlet or JSP interacting directly with a database through JDBC, an abstract interface such as Hibernate or using Enterprise Java Beans (EJB). Portlets give new flexibility and extensibility to SOA by enabling the best use of J2EE. JSR 168 allows legacy Web applications to be deployed as portlets in a Portal Framework.

Portlets are not confined to one Portal Framework, based on standard Web Services technology OASIS [4] released the Web Services for Remote Portlets (WSRP) also in 2003 aiming to define a standard for interactive, user-facing Web services to make portlets hosted by different geographically distributed Portal Frameworks accessible in a single portal [1]. Unlike traditional Web services however, WSRP defines a protocol which is focused on transferring the markup fragments generated by portlet producer and consumer. Although WSRP is still at an early stage as far as implementation is concerned, it indicates the future of portlet/ portal development. Ideally, a deployed service with a portlet interface can be published and consumed in many different portals/ Portal Frameworks. This remote sharing of a single portlet will greatly ease the construction of large-scale portal based systems, or Virtual Research Environments, enabling them to be more scaleable, manageable and maintainable by both service providers and portal administrators.

4. Single Sign On

Single Sign-On (SSO) is a key requirement in providing Authentication and Authorisation services to access facilities. We are evaluating technology to determine a security framework which is easily scalable in terms of extending the framework to different user account management systems such as LDAP, Active Directory Service or MyProxy server using X.509 certificates. The aim is to make the user login experience independent of the authentication technology. Single Sign-On is implemented in the NGS Portal using the MyProxy server authentication mechanism to access the portal services, as described in section 4.1 below. NGS Portal Single Sign-On is achieved by extending the Java Authentication and Authorisation Service framework (JAAS) [6]. JAAS implements a Java technology version of the standard Pluggable Authentication Module (PAM) framework, and supports user-based authorisation. Some portal frameworks directly support JAAS, e.g. StringBeans, which is used for the NGS Portal. Authorisation of an NGS user will be accomplished by using JAAS and portal framework role-based access control (RBAC) functionality. JAAS authorisation extends the existing Java security architecture that uses a security policy to specify what access rights are granted to executing code. We are also looking to evaluate another Single Sign-On framework called Java Open Single Sign-On (JOSSO) [7]. Some main features of JOSSO include strong authentication using X.509 client certificates, support for multiple simultaneous authentication systems, integration of Java and non-Java applications and a security model based on JAAS, SOAP Web services, EJB, servlet/ JSP and Struts standards. These technologies will provide links into the UK e-Science Certification Authority and CCLRC's Corporate Data Repository for Grid and facilities users.

4.1 Single Sign-On on NGS Portal

This section explains how the Single Sign-On through MyProxy server is implemented in the NGS Portal release 2.0 using the StringBeans portal framework and JAAS security framework. However in implementing the Single Sign-On on NGS Portal using JAAS, the portal has now become tightly coupled with the StringBeans portal framework and hence JAAS technology does not give flexibility and scalability in extending the another authentication mechanism easily. On NGS every user is provided with an X.509 based certificate which is used by the Grid middleware Globus Toolkit v2.4 [18]. A MyProxy repository has been set up for use by the NGS users to store their time-limited proxies via the MyProxy server, which can be then be easily accessed to execute Grid related applications. The Globus Toolkit uses a PKI-based Grid Security Infrastructure (GSI) for Single Sign-On in a Grid environment. It is therefore necessary to retrieve a Grid proxy credential before accessing any Grid resource. A Web-based portal which supports the full Grid functionalities typically follows two steps to retrieve a proxy credential generated by:

1. A user logs in to the portal using his/ her login details;
2. Once authenticated, the user uses a tool within the portal to retrieve a Grid proxy credential commonly through an external MyProxy server. With this proxy credential, it is then possible to

access remote resources. In this way, as described in Fig. 1a, the user has to remember two separate accounts, i.e., a portal account and a MyProxy account details.

Since it is necessary to retrieve a Grid proxy credential from the MyProxy server for accessing the UK core Grid, a natural solution to simplify the two-step authentication is to authenticate a user through the MyProxy server. As illustrated in Fig. 1b, the one-step login diagram simply authenticates a user using the MyProxy server without a portal authentication check. In the NGS Portal release 2.0, a user account is created dynamically and the user details for the portal are extracted from the proxy Distinguished Name (DN). Once the user account is created, the user can log in to the portal using the existing user profile. The proxy is stored in a session and hence all Grid related portlets requiring a proxy are included in a single Web Archive (WAR) file. The reason for doing this is that the JSR 168 specification does not support sharing of sessions between different portlet applications.

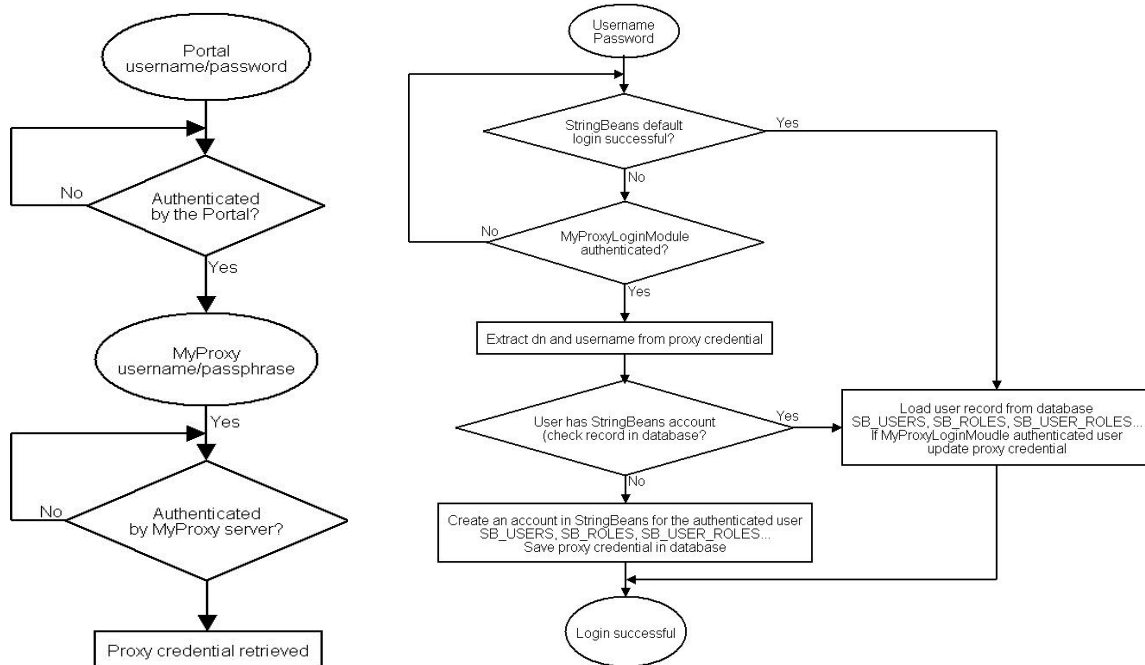


Figure 1a: Two-step portal login

Figure 1b: Authentication flow diagram for NGS Portal

The authentication flow diagram for NGS Portal release 2.0 illustrates the fact that a user is authenticated in two ways. First the portal tries to authenticate the user setup through the portal interface. If authenticated, the user profile will then be loaded and user role(s) will be set. This is the default NGS Portal login process. If the authentication fails logging as a portal user, the user will then be directed to the MyProxy Login Module and will login as a Grid NGS user. Once the user is authenticated, the user will be mapped to a portal local account and the user's Grid proxy credential will be saved in a separate NGS database rather than modifying the portal's own database. The user's role will be set to StringBeans user and NGS user. Different user roles can be set up for authorisation purposes for using various portlets on the portal. For instance users with non-NGS roles will not have any access to Grid related portlets. Once the user has logged in, the whole operation is managed by the StringBeans portal framework. Portlets will be able to retrieve the user's proxy credential automatically from the database.

4. Desktop Clients to Access Grid Resources

The latest trend of application development shows the popularity of Web applications among users and developers. Web applications provide uniform and pervasive interfaces to remote services and resources regardless of user environment. For developers, they are easy to develop and maintain and for users there are no installation and configuration issues. Today Web applications are very mature courtesy of standards like HTML, JavaScript, CSS and XML related technologies but are still lacking in features compared to desktop applications. Desktop and Web applications are complementary and address different sets of requirements, although there is overlap in basic functionalities. For simple distributed resources where user interaction is limited, Web applications are the better choice, but for Grid services where user interaction is more complex, desktop applications may be a better choice. We are aiming to support both desktop and Web interfaces considering the user and project requirements through the use of Web services.

Development is carried out in a modular way to avoid potential failure in large systems, especially where there are complex user requirements. The concept of SOA was developed to avoid the problems of having to re-write large portions of code to accommodate changing requirements. A SOA is essentially a collection of autonomous, self contained, pluggable, loosely coupled services that have well-defined interfaces and functionality with little side effect. A service is thus a function that is self-contained and immune to the context or state of other services. These services can communicate with each other either by explicit messages which are descriptive, rather than instructive or there could be a number of "master" services coordinating or aggregating activities, e.g. in a workflow. These core and atomic services can be integrated into either Desktop or Web applications (scaled down version) for uniform user experience.

The Workflow Optimisation Service for E-Science (WOSE) project started recently with CCLRC, Imperial College and University of Cardiff. The purpose of this project is to investigate optimisation strategies for workflow execution for Web services: (1) using Business Process Execution Language (BPEL) [9]; (2) Simple Conceptual Unified Flow Language (SCUFL) [10]; (3) Business Process Modelling Language (BPML) [11]. The idea is to develop workflows from the point of view of users with limited knowledge of workflow languages but wanting to execute complex scientific procedures using the Grid.

In WOSE, dynamic Web service invocation and interoperation between different workflow languages and workflow engines will be used. We aim to enhance the user experience and provide links for known Web services and for discovery from private or public Universal Description, Discovery and Integration (UDDI) registries. For ease in engineering workflows, we aim to develop a drag-and-drop application. In this case, a robust, dedicated "fat" client is better than a browser-based client. If instead, the client software is used by a large number of people requiring only limited functionality, e.g. swapping one component for another, a Web client is ideal. Browser based clients have an inherent advantage as they do not need to be upgraded on the client side and provide a pervasive access mechanism.

Considering the user requirements we can identify two categories: (1) users executing existing workflow for complex scientific procedures in which they may not be experts; and (2) expert users engineering new or existing processes as workflows. For the first type of users Web Applications are appropriate since executing existing workflows means uploading the workflow and supporting parameters and constraints to the server for use by the workflow engine. The second type of user needs rich client software with advanced functionality. This includes comfort features, such as a polished user interface, drag-and-drop, and a rich set of keyboard shortcuts, and provision to save partial workflows and integrate them as building blocks for complicated and sophisticated workflows.

4.1 Practical Issues

In real world workflow scenarios, workflows are not simple tools that call different Web services in fixed static or dynamic manners. Rather, a workflow engine or workflow developer has to tailor the output from one Web service according to input requirements of the next Web services. This transformation can be simple and automatic or can be complicated involving creation of new data structures (via an XML

Schema) based on the output of previous activities. Things become more complicated when a newly created data structure requires input values from a user during the execution of the workflow. There is no option in current specification to “pause” workflow, notify users to input required parameters and resume the workflow after input. Moreover, Web services orchestrated through workflow are deployed on geographically distributed servers, which may have security and access or service policies, which should be negotiated before execution of the services. Workflow specifications don't have any provision to support these additional policy constraints and there should be a mechanism to plug-in policy parameters in the workflow. Static policy constraints can be kept in the workflow as variables but if they are dynamic and are based on the output of previous activities, user interaction is required. Grid resources are more dynamic in nature and workflows using them are therefore not predictable.

Based on different usage scenarios, we have designed a set of “integration services” to extend existing workflow specifications of BPEL. These services constitute CPU resource overhead and are suitable for Desktop applications rather than Web Applications. Integration services developed or planned for workflow extensions include:

1. Data conversion or Data Structure creation using XSLT for different operation calls of different Web services. Output from any Web service operation should be compatible to the input requirements of next operation; this can be achieved by XSLT, Web Service Invocation Framework. For describing physics and chemistry data we have ongoing work using Semantic Web tools;
2. Messaging services to inform the user of workflow about an unexpected result/ fault during execution (meanwhile pausing the workflow execution for further instructions), these unexpected circumstances can be due to temporary un-availability of a Web service, change in security requirements, or different policy constraints. Workflows are not capable of handling these exceptional scenarios automatically and needs direct or indirect interaction with the user;
3. Maintaining a pool of similar/ compatible Web Services, failure or un-availability of a Grid resource will then not result in premature termination of workflow as dynamically other compatible Grid resources can be tried based on a rating mechanism discussed later. If a system can't find any other compatible service the workflow will have to be paused and resumed later, either automatically based on predefined constraints or with intervention of the user;
4. Integration of local Java classes with the workflow in a pre-defined manner, which may add Soap Headers, Digital Certificates for security, Web Service Policy based constraints/ parameters before calling the Web service. This information is not usually available in WSDL documents and client and service providers should have negotiated and agreed before using/ offering the service. A workflow engine has no way to adjust itself at run time and without these constraints and additional information the workflow can't call the Web service successfully;
5. A rating mechanism to rank similar Web services in a pool based on different policies which are configurable and can be modified according to different Service Level Agreement (SLA) requirements. The rating system will help dynamic selection of the best available service (according to criteria to be defined) from a pool of services;
6. A graphical monitoring tool to monitor the workflow in scientific services when the workflow cycle lasts for days if not weeks. It would be nice to have user interaction with a monitoring tool to stop/ pause/ resume/ cancel/ modify the workflow at run time giving more flexibility to the scientist as the results of their procedure become clear. This is related to steering (computational or experimental);
7. Information Persistence is supposed to store information at different stages (breakpoints) of workflow which includes variables or Java object serializations to re-run the workflow between different breakpoints on demand. Again this could be used in steering where checkpointing and rollback might be required.

Many of these are similar to the integration services provided by a portal on the server side.

5. Portals and Web Service Examples Developed at CCLRC

We give a couple of examples which substantiate our work.

5.1 The UK National Grid Service

The National Grid Service (NGS) [12], funded by JISC, CCLRC and EPSRC, is the core UK Grid, which has been deployed for the production use of computational and data Grid resources in all branches of academic research. Currently the NGS core Grid consists of four clusters - two compute clusters and two data clusters, plus two national supercomputer services. The four clusters are located at the University of Manchester, the CCLRC Rutherford Appleton Laboratory, the Universities of Leeds and Oxford. The two HPC services are the University of Manchester based CSAR Service and the HPCx Service based at CCLRC Daresbury Laboratory. Inately these recourses are geographically distributed and provide an ideal test bed for Grid technology. Three additional nodes from the Universities of Bristol, Cardiff and Lancaster have recently been connected to NGS. Overall the NGS provides access to over 2,000 processors, and has over 36 TB of "data Grid" capacity. In order to provide users with transparent and easy access to these resources, the NGS Portal has been developed and deployed for production usage. Many Web-based portals have been set up for distributed environments to improve the usability of the Grid, for example, the Telescience Portal [13], the HotPage Grid Computing Portal [14] and the Cambridge CFD Grid Portal [15]. Work in the Grid Technology Group at CCLRC Daresbury Laboratory has focussed on integrating Grid services and collaboration tools into a familiar Web portal environment. This is in the context of the Integrated e-Science Environment [16]. Early versions of such portals [17] were developed in conjunction with the Level-2 Grid deployed by the UK Grid Engineering Task Force.

In this paper, we will first give a brief introduction to the prototype portal and feedback from users. Then two releases of the NGS Portal, release 1.0 based on CHEF (a CompreHensive collaboratiVE Framework) and release 2.0 based on StringBeans (a JSR 168 compliant portal framework) are described. Services are exposed in the NGS Portal using portlets which can be customised by the user. Stringbeans is used as the current portal framework. JSR 168 compliant portlets deployed at present in NGS Portal are: MyProxy management, MDS resource discovery, GRAM job submission, GridFTP, Batch job monitor, Storage Resource Broker (SRB), OGSA-DAI and other collaboration tools are being integrated from Sakai.

As an example of this work, we note that OGSA-DAI is a core service of the Globus toolkit developed by the Globus Alliance at University of Edinburgh. It provides an extensible middleware Grid framework for easily integrating data from independent data resources and has the following features: allows data access; allows data discovery; facilitates data integration; and provides a uniform interface to access data. We have developed a portlet interacting with OGSA-DAI Web services using Tomcat and Axis SOAP implementation. OGSA-DAI can be used with or without a security model. The current implementation of OGSA-DAI deployed uses a Message Level Security to access data. Further work will be carried out to provide access to OGSA-DAI services using Authentication and Authorisation mechanisms as described above. This, and other work on OGSA Grid services, was started in the UK OGSA testbed project with CCLRC and the Universities of Portsmouth, Manchester, Westminster and Reading. Some additional tools use Java Webstart or applets to provide necessary client-side functionality, e.g. for uploading a proxy or accessing desktop files.

5.1.1 Cloning the Portal for other e-Science Projects

One benefit of adopting the JSR 168 standard is that it is possible to make the Java code re-usable since JSR 168 defines a set of standard APIs for portlet development. The Grid portlet application we

developed has been successfully deployed in other portals like the Integrative Biology VRE Portal without modification of the source code itself. We are also investigating the use of WSRP, by which a project such as the IBPortal may access portlets from the NGS portal remotely.

Projects with which we are currently working to share code are: Sakai Virtual Research Environment (VRE) Demonstrator, Integrative Biology VRE Demonstrator, e-HTPX e-Science projects, SCARF (a cluster on the CCLRC Facilities Grid) and ReDRESS [19].

5.1.2 Inter-Portlet Communication

An important issue in the current JSR 168 standard is that inter-portlet communication is not supported. As defined in the specification, each portlet application is an independent Web application. It is currently only possible to include all portlets inside one portlet application, as we do in the Grid portlet application, to share session information with each other, for instance. Some portal frameworks like the IBM WebSphere Portal [20] provide an inter-portlet communication feature as part of the portal framework. In fact, there is always a need to communicate at the portlet application level, i.e. to communicate between portlet applications.

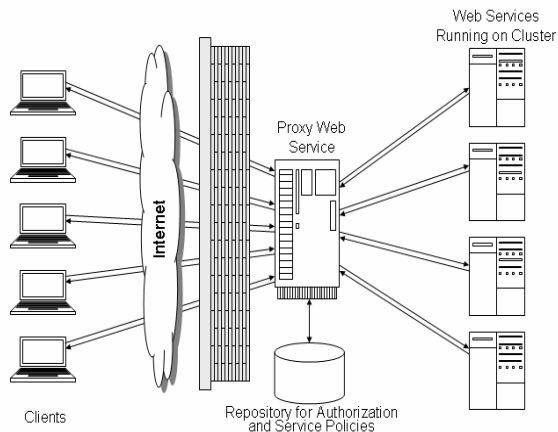
The Java Messaging Service (JMS) [21] has been tested in the BatchJobMonitor portlet. Once a job is finished, a JMS message is sent to a JMS server. This message can then be consumed by any JMS client program. This can be further developed as a notification function within the portlet.

5.2 Web Services and Portals in e-HTPX

e-HTPX is a BBSRC (Biotechnology and Biological Sciences Research Council) funded UK e-Science project for structural biology [3]. The project aims to develop both a communications infrastructure and a suite of related user interfaces designed to allow the planning and remote execution of protein crystallography experiments. The project integrates a number of key services provided by UK e-Science, protein manufacture and synchrotron radiation facilities (Diamond, ESRF). The project implements and tests a number of distributed computing technologies, including those discussed above (especially portal development, communications via Web services and provision of 'gateway' style Web services to provide access to service-endpoints situated inside an institutional firewall).

The e-HTPX portal is a distributable Web application that provides both an interface to input necessary data, and a single point of access to the underlying e-HTPX Web service framework. The portal acts as both a Web service client and a Web service-response hub for collation of responses from different services (example services situated at the Daresbury or Grenoble synchrotron sites include those associated with X-ray data collection and HPC services designed to aid the determination of protein structures). Two portal architectures have been designed in order to suit different research environment needs: a) the service-site portal, which is maintained by each synchrotron, and offers a standard service for the majority of clients; and b) the client-site portal, which can be installed at a client institution such as University of York and allows configuration and close integration of the portal functionality with a client institutional LIMS (Laboratory Information Management System). Despite the differences in portal architecture, both portal implementations act as clients to the services hosted at the synchrotron. When using the portal to submit Web service requests to the synchrotron, authentication is first implemented by the portal application layer (i.e. when logging onto the portal interface). The security credentials are also passed to the synchrotron Web services allowing secondary (site-specific) authentication and authorisation.

Figure 1: Proxy Web Service for Authorisation



With regard to the communication infrastructure at the synchrotron site, it has to be anticipated that firewall policies will limit direct access from external clients to the key HPC and data collection services (i.e. it is most likely that these resources will be located inside internal firewalls, and can only be accessed via known gateway servers). Consequently, an extensible 'gateway' style Web service framework has been designed for accessing these resources. The gateway model means that all client requests are sent to an externally visible 'gateway' Web service before being routed to the ultimate receiver (the

requested service). The client can't submit a request directly to the service and knows nothing of the actual service endpoint (i.e. the endpoint URL). In addition, firewall administrators may implement additional security measures such as IP-recognition between gateway server and service endpoint in the form of Web services handlers. This model represents an end-to-end communication involving three 'actors' or 'roles' (client, intermediary, and ultimate receiver), not a point-to-point communication that implements direct communication between the client and ultimate receiver. The primary role of the intermediary gateway Web service in e-HTPX is to perform authorisation and authentication but by no means is it restricted to any specific task, in fact Web service handlers can be configured in many possible ways to achieve required core or composite functionality. The Gateway Web service determines the requested resource by parsing the XML payload, parses the input parameters for the requested service to validate them, and re-routes the SOAP request (including attachments) to the endpoint service through optional handlers/ providers. In performing these actions, the ultimate service endpoint is ensured to receive properly validated requests only from authorised users.

A Web Services Gateway defines different access points based on type and number of services it is handling, these access points are called channels. Channels are external access points to the gateway Web Service exposed to clients, and for different services there can be different channels. A request to the Web services gateway arrives through a channel and is translated into an internal representation of the service. With the help of Web service handlers, a request can be logged, intercepted, or generally manipulated. After parsing the request, an appropriate provider is used to communicate with the target service. The provider in the gateway acts as a client for the target Web service. The response from the target service flow along the exact same path back to the provider, i.e. each handler used to parse the request will be called again to parse the response. In this sense the layout of the gateway is symmetric. However, one or more response handlers can be deployed which may not parse the request and just pass the request to the next handler in the chain.

The process of deploying a target service with a gateway generates two different *external* WSDL files; an implementation definition and an interface definition. These new WSDL files are used together by the client application to access the actual service(s) and are the externalisation of the service capabilities offered by the *internal* target service. The implementation WSDL definition is used to expose the internal service(s), operations and type of messages involved in operation(s) to simplify the connection process for a client, particularly when dynamic invocation is being used. The implementation WSDL is an abstraction of the service offered to the external world with concrete binding protocol but without physical service endpoint information. Once the client has obtained the WSDL implementation definition, the client can then access the gateway Web service using the WSDL interface definition; referencing the

gateway service endpoint, the WSDL interface definition provides full information about the intermediary gateway Web service.

5.2.1 Two Scenarios for a Gateway Web Service

To protect the e-HTPX Web Service from un-authorized accesses, a Web Service Gateway seems to be a natural and most flexible solution; not restricted only to address security constraint. During the designing stage of the gateway Web service two different approaches were considered for the actual implementation of the solution, we haven't yet evaluated the benefits and limitations of both solutions. The core concept behind the gateway Web service as mentioned in Section 5.2 is to act as an intermediary between the client and ultimate receiver, from the client's point of view this intermediary (gateway Web service) is the ultimate receiver. The ultimate receiver, e-HTPX Web service is deployed on a cluster and that deployment can be achieved in two possible ways which are explained below:

1. **Cloned Web Service:** The complete e-HTPX Web service is cloned and deployed on different nodes of the cluster, thus each node has exactly the same copy of the e-HTPX Web service. This means technically speaking each node is similar with respect to service(s) and feature(s) exposed to the external world. The Gateway Web service can use any monitoring mechanism to monitor each node related to its service load, quality of service and availability of external resources on each node and thus redirect the clients' requests to the most appropriate node (ultimate service endpoint), i.e. node with minimum service load to achieve best SLA. The main advantage of this approach is that; if at any time, any node is unavailable or is overloaded then that node can be easily replaced by any other node for optimization and efficiency at runtime without extra efforts as every node in the cluster is compatible. In this case one "complete" e-HTPX Web service has one service endpoint but there are different equivalent e-HTPX Web service on different nodes with different internal service endpoints.
2. **Disintegrated Web Service:** The other possibility is to deploy different operations of an e-HTPX Web service on different nodes of a cluster or Grid, thus each node is different in regard to the services it is providing. The Gateway Web service now determines the requested resource by parsing the XML payload, parses the input parameters for the requested service, and re-routes the SOAP request (including attachments) to the appropriate node (internal service endpoint). Each node in the cluster has its own unique service endpoint, and each individual node is a physical service binding of the abstract port type in the WSDL file. This approach gives more flexibility at deployment as each operation can be deployed separately and can improve the overall efficiency of the e-HTPX Web service especially when hardware specifications and capabilities of each node are different. The port types (operations) in the WSDL file which are expected to be used more often or need extreme hardware resources can be deployed on the nodes with extra hardware resources and network bandwidth support; whereas port types which will be requested rarely by the client(s) can be deployed on low-performance nodes or even on distant nodes. In this case the complete e-HTPX Web Service doesn't have one single service endpoint, in fact each port type (port type can have one or more operations in WSDL) has a different service endpoint. It is also possible to clone the overloaded operations within the disintegrated Web service and deploy those cloned port types on different nodes, which gives more enhanced performance and monitoring and dynamic replacement possibilities.

5.2.2 Benefits of a Gateway Web Service

Considering the security requirements of our Web service(s) we have recommended a non-functional requirement to implement a gateway Web service. In this scenario, the requirement is for a more secure and flexible way of controlling access to the services consumed and provided across the organisational boundaries. Web services have a lot of benefits for Grid application development, and not only enhance the reliability of Grid applications by utilising the core concept of Service Oriented Architecture but reduce the initial development time and long term maintainability hassle as each service

itself is autonomous and self contained. The greatest advantage of Web services is the flexibility and possibilities for further extension(s); in our e-HTPX Web service example we have extended the core functionality by introducing the gateway Web service. This Gateway Web service provides the access to the distributed components and is coupled with business logic, location of those components and dependency of these components on other external resources, i.e. file system, database, hardware, etc. for added value features like monitoring, load balancing, runtime fault tolerance, reliability. The Gateway Web service is a runtime component that provides configurable mapping between the e-HTPX Web service and requesters. In general gateway Web services can provide the following benefits :

1. Central Access Point for all services: The gateway Web service provides a single, well-known point of access to Web service consumers and users; crossing the organisational boundaries. Grid Applications comprise different geographically dispersed Grid or Web services, and consumers don't need to know and remember their locations; thus gateway Web services hide the inner working details and location of Application components by exposing a well known point of access, which may be the aggregation of different internal services;
2. Decoupling the Deployment: The gateway Web service isolates any changes in the deployment of service(s) from consumers of the service(s). The location of service(s) also becomes transparent to clients of the service which gives more flexibility to the administrator for re-deploying the services for load-balancing, maintenance, upgrades, etc. Over time, the location of the Web service target application and the bindings may change, but these details are handled by the gateway. As a Grid service provider, administrators need the flexibility to change deployment infrastructure without notifying all the service requestors. For example, consider a Grid service deployed on any machine that fails during operation, demands the availability of any process to route the invocations to an alternate service in Grid infrastructure; this routing capability can be part of the gateway or pluggable component;
3. Central Security Control Point: Access control can be applied to Grid services within the domain so only authorised consumers are allowed to access the services. Access to services is controlled by gateway Web service which gives tighter control on security policies and services can benefit from different access policies such as Single-Sign-On (SSO), role based access control (RBAC), content based access control and even location based access control;
4. Protocol Conversion: Access to the Grid Services and Grid Applications are not limited to the HTTP protocol in fact they can use any proprietary protocols for efficiency, security or reliability which must be hidden from the client. The client accesses the gateway Web service using standard HTTP protocol and the gateway Web service converts HTTP protocol requests into proprietary protocol by using Web service handlers before re-directing them to the ultimate service. Similarly Grid Services may produce output in a non-standard protocol and use Web service handlers to transform non-standard protocol into HTTP protocol which is returned by gateway Web service to users;
5. Non-Functional Requirements: A Gateway Web service is best to implement the non-functional requirements of the Grid application, i.e. enhanced user experience, session handling, security, logging etc. Non-functional requirements implemented in gateway Web services provide uniform features, a single place of update, possibility of reuse across different Grid applications, clean business logic implementation and true component based architecture for diverse applications;
6. Process Abstraction: The service invocation approach must be flexible enough to cope with events such as switching frequently between external providers of a similar service without requiring changes to the application and application user interface. Gateway Web services provide this process abstraction naturally and their coupling with Grid applications provide the flexibility to extend, modify and even re-engineer the Grid applications with minimum impact on users. Changes in WSDL implementation definition only requires adjustments in the client's user interface; WSDL implementation definition mirrors the core functionality of the internal service(s);

7. Monitoring: The Gateway Web service is mainly implemented for access control and to enhance security model, but it can be extended in other ways to monitor the services deployed on clusters or remote machines. This monitoring of deployed services helps in achieving the Quality of Service (QoS) according to already negotiated Service Level Agreement (SLA), load balancing, availability, etc.

5.2.3 Gateway Web Service and Dynamic Selection

An internal UDDI registry can be used by the gateway node to obtain the interface description and the implementation description of the Web services for which it is used as proxy. Although it may be easier in the short term to simply put the WSDL files of the Web services on a Web server, or use WSIL, than to implement a solution using a public or a private registry. Use of an internal UDDI registry by the gateway service for dynamic discovery de-couples the gateway node and ultimate target services. Dynamic discovery of services by the gateway node can be based on the criteria mentioned in the client's request, updated load information of different nodes within the cluster. As Web services are becoming more popular, providing internal or external UDDI registries to clients for discovering Web services according to runtime requirements is key for Grid applications. UDDI registries allow a fine degree of classification for Web services that allows consumers to quickly find the Web services to fit their needs and requirements and above all this classification of Web services can be integrated with semantic Web technologies for automatic discovery and consumption of the Web services.

5.3 Final Comment on Portlets and WSRP

Currently there are several open source portal frameworks which support WSRP functionality, e.g. uPortal and eXo Platform, but the WSRP is integrated as an internal mechanism. Other portal frameworks which do not support WSRP therefore can not use the functionality directly. We have also developed a generic WSRP consumer portlet using the WSRP4J API [8]. At a later date the WSRP portlet will also support the WSRP producer functionality. The WSRP portlet will implement a presentation oriented service, interaction and portability. The benefit of using the WSRP consumer portlet would be that it can access multiple remote portlets via the WSRP portlet producers. The advantage is that portlets developed as WSRP producers can be hosted on a single machine but can be consumed from any portal environment and hence deployment and maintenance of the Web application can be much easier. This leverages on the SOA concept. This is ongoing work and is particularly of interest in advanced training for scientists where content can be aggregated and combined with collaboration tools, for instance in the ReDRESS project which is a collaboration of CCLRC and University of Lancaster [2]. WSRP may also be a convenient way for commercial tools to be exposed in portals alongside open source tools. We note that the Sakai project in the USA is currently evaluating such possibilities for Collaborative e-Learning and we are using this in a VRE Sakai demonstrator project, collaboration between CCLRC and the Universities of Lancaster, Oxford and Portsmouth. We plan to have project-specific portals which consume collaboration tools from Sakai and NGS tools from the NGS portal as remote portlets together with bespoke local portlets.

6. Summary

As discussed in this paper, the emergence of portal technology is providing benefits in developing portlet interfaces to applications to meet the current and future requirements of CCLRC facilities support. Portlets can be re-used by different projects, e.g. the high-profile Integrative Biology project (with University of Oxford), and in different JSR 168 compliant portal frameworks. Deployment and maintenance of applications developed as portlets becomes easier and manageable. A community process is already beginning and many portal frameworks come with free-to-use useful portlets. Because rendering is carried out in the framework, applications can be easily accessible and internationalised. Portlets are compatible with J2EE thus providing additional capabilities required in the

SOA architecture. We have also described how Web Service Gateways can be used to provide many of the functionalities encapsulated in a portal server in a way to support Grid applications. Portals used as a rich client can allow users to customise or personalise their user interfaces and even their workflow and application access. CCLRC facilities will be able to leverage the work so far carried out on the NGS and e-HTPX portals which are fully functional and have received detailed user feedback. This demonstrates the usefulness of providing advanced capabilities for e-Research and having the associated business logic in an SOA loosely coupled from the presentation layer for an Integrated e-Science Environment.

Acknowledgements

We thank the UK e-Science Core Programme for funding some of the work on the NGS Portal via the Grid Operations and Support Centre, JISC for funding work on the GROWL and Sakai Demonstrator projects via its VRE Programme, BBSRC for funding some work via a grant for the e-HTPX project and EPSRC for funding some work via grants for Workflow Optimisation Services for e-Science and the UK OGSA Testbed.

References

- [1] R.J. Allan, C. Awre, M. Baker and A. Fish *Portals and Portlets 2003*. Proc. NeSC Workshop 14-17/7/2003 (CCLRC and NeSC, 2004) http://www.nesc.ac.uk/technical_papers/UKeS-2004-06.pdf
- [2] R. Crouchley, A. Fish, R.J. Allan and D. Chohan *Portal Services for Awareness and Training*. Proc. AHM'2004 (Nottingham, September 2004) paper 223
- [3] Rob Allan, Ronan Keegan and D. Meredith *e-HTPX – HPC, Grid and Web-Portal Technologies in High Throughput Protein Crystallography* AHM'2004 (Nottingham, September 2004) paper 101
- [4] *WSRP Specification 1.0 by OASIS* <http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf>
- [5] *Portal specification JSR 168 and API* <http://www.jcp.org/aboutJava/communityprocess/review/jsr168/>
- [6] *JAAS* <http://java.sun.com/products/jaas/>
- [7] *JOSSO* <http://www.josso.org/>
- [8] *WSRP4J* <http://ws.apache.org/wsrp4j/>
- [9] *BPEL, version 1.0*, (31st July 2002) <http://www.ibm.com/developerworks/library/ws-bpel/>
- [10] *SCUFL* <http://taverna.sourceforge.net/docs/beta8workbench/workbench.html>
- [11] *BPML* (March 8th 2001) <http://www.bpml.org/BPML.htm>
- [12] *The UK National Grid Service* <http://www.ngs.ac.uk> and *NGS Portal* <http://portal.ngs.ac.uk>
- [13] S.T. Peltier, A.W. Lin, D. Lee, S. Mock, S. Lamont and M.H. Ellisman, *The Telescience Portal for Advanced Tomography Applications*, Journal of Parallel and Distributed Computing, Vol.63, No.5, (2003), pp. 539-550.
- [14] *HotPage Grid Computing Portal*, <https://gridport.npaci.edu/>.
- [15] X. Yang, M. Hayes, K. Jenkins and S. Cant, *The Cambridge CFD Grid Portal for Large-Scale Distributed CFD Applications*, International Conference on Computational Science 2004 (ICCS2004), eds. M. Bubak *et al.*, (Springer-Verlag, LNCS 3036, 2004), pp. 478-481.
- [16] A.J. Richards, R.J. Allan, G. Drinkwater and K. Kleese, *Building the e-Science Grid in the UK: Advanced Grid Computing*, UK e-Science AHM2003, (Nottingham, UK, 2003)
- [17] R.J. Allan, D. Baker, D. Boyd, D. Chohan, S. Cox, H. Eres, R. Fowler, N. Furmento, J. Giddy, T. Harmer, M. Hayes, N. Hill, J. Hillier, J. Jensen, A. Keane, M. Krznarič, W. Lee, M. McKeown, A. Mills, S. Newhouse, S. Pickles, R. Pinning, A. Richards, A. Saleem and J. Watt, *Building the e-Science Grid in the UK: Middleware, Applications and Tools Deployed at Level 2*, UK e-Science AHM2003, (Nottingham, UK, 2003)
- [18] *Globus Toolkit* www.globus.org

- [19] R. Crouchley, A. Fish, R.J. Allan and D. Chohan, *Virtual Research in the UK: Portal Services for Awareness and Training in e-Science*, UK e-Science AHM2004, (Nottingham, UK, 2004)
- [20] IBM WebSphere, <http://www-306.ibm.com/software/WebSphere/>
- [21] Java Messaging Service (JMS), <http://java.sun.com/products/jms/>