CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2007; **19**:921–942
Published online 10 October 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1078

The Open Grid Computing Environments collaboration: portlets and services for science gateways

Jay Alameda[1], Marcus Christie[2], Geoffrey Fox[2,3], Joe Futrelle[1], Dennis Gannon[2], Mihael Hategan[4], Gopi Kandaswamy[2], Gregor von Laszewski[5], Mehmet A. Nacar[3], Marlon Pierce[3,*,†], Eric Roberts[6], Charles Severance[7] and Mary Thomas[8]

[1]*National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, Urbana, IL 61301, U.S.A.*
[2]*Department of Computer Science, Indiana University, Bloomington, IN 47404, U.S.A.*
[3]*Community Grids Laboratory, Indiana University, Bloomington, IN 47404, U.S.A.*
[4]*Computation Institute, University of Chicago, Chicago, IL 60637, U.S.A.*
[5]*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, U.S.A.*
[6]*Texas Advanced Computing Center, University of Texas, Austin, TX 78758-4497, U.S.A.*
[7]*School of Information, University of Michigan, Ann Arbor, MI 48109-1107, U.S.A.*
[8]*Department of Computer Science, San Diego State University, San Diego, CA 92182, U.S.A.*

SUMMARY

We review the efforts of the Open Grid Computing Environments collaboration. By adopting a general three-tiered architecture based on common standards for portlets and Grid Web services, we can deliver numerous capabilities to science gateways from our diverse constituent efforts. In this paper, we discuss our support for standards-based Grid portlets using the Velocity development environment. Our Grid portlets are based on abstraction layers provided by the Java CoG kit, which hide the differences of different Grid toolkits. Sophisticated services are decoupled from the portal container using Web service strategies. We describe advance information, semantic data, collaboration, and science application services developed by our consortium. Copyright © 2006 John Wiley & Sons, Ltd.

Received 19 August 2005; Revised 2 February 2006; Accepted 13 March 2006

KEY WORDS: portlets; collaboration; Grid services

*Correspondence to: Marlon Pierce, Community Grids Laboratory, Indiana University, 501 N. Morton, Suite 224, Bloomington, IN 47404, U.S.A.
†E-mail: mpierce@cs.indiana.edu

Contract/grant sponsor: National Science Foundation's Middleware Initiative (NMI) Program
Contract/grant sponsor: Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy; contract/grant number: W-31-109-Eng-38

1. INTRODUCTION

Browser-based science portals date from the beginning of Grid research [1,2] and have served as a focal point for investigating the problems of integrating Grids with Internet protocols, with heterogeneous security mechanisms, and with Web programming languages (Java, Perl, Python). The state of portal development through 2001 is surveyed in a special issue of *Concurrency and Computation: Practice and Experience* [3]. Fox *et al*. [4] provide an analysis and categorization of these first-generation portal efforts.

The portal research summarized in [3] grew out of the Global Grid Forum's Grid Computing Environments (GCE) Research Group. The GCE generally revealed three things. First, all portal projects worked on the same general problems: file transfer and management, job submission and monitoring, data management, and user management. Second, most portal projects had adopted a variant of the 'three-tiered architecture' model, discussed below. Finally, and unfortunately, an important gap existed between projects: no common portal programming interfaces or generally accepted inter-tier communication protocols were in use. This dramatically limited the amount of code sharing and collaboration.

This situation began to change rapidly in early 2002 with the emergence of two important concepts: reusable portal components (portlets) and Web service architectures. Java portlet components became standardized with the Java Specification Request JSR 168 [5]. Web service architectures are summarized in [6]. Modern portal systems have adopted these two cornerstones and follow a general architecture shown in Figure 1. Standard-based portlets provide reusable functional components that can be shared between different portal installations. Web services decouple the portal functionality from its presentation layer. As shown in Figure 1, general-purpose Web service communications (i.e. SOAP) connect portlet applications with remote services. These remote services provide generic views to specialized backend resources. A common example is the GRAM job manager, which can bridge to PBS and LSF queuing systems as well as to the local operating system. OGSA-DAI, which provides a Web service layer of abstraction over relational and XML databases, provides an analogous example for data management. This diagram follows the general 'three-tiered architecture' model of enterprise portal systems [7]. The architecture diagram is greatly simplified, as both the client abstraction layer and service implementations can be very complicated (e.g. Figure 2). However, the sophistication of these implementations is hidden from the other system components.

The power of the general architecture shown in Figure 1 in that it supports not just distributed computing but also distributed development. Different development groups can provide portlet plug-ins and service components that need only expose programming interfaces. The Open Grid Computing Environment (OGCE) [8] collaboration is built upon this model. The general problems to be solved include:

- supporting container-independent portlet development;
- simplifying the development of Grid clients through abstraction layers;
- developing sophisticated supplemental Grid services that manage science applications, Grid information, and data;
- integrating collaboration tools and services to support user communities.

These efforts are fully described in other references given in the appropriate sections. Our purpose here is to provide a summary and place the efforts within the context of Figure 1.

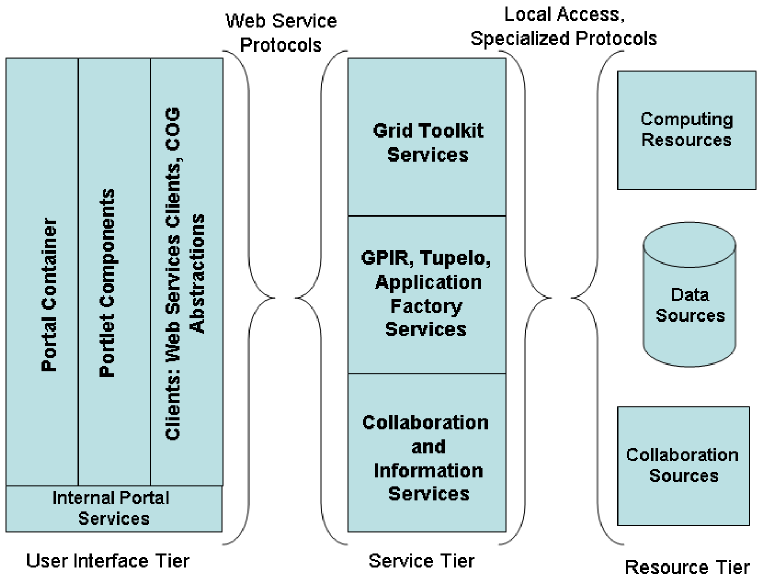

Figure 1. Portlet and services are organized into three logical tiers. The figure is described in more detail in the text.

```
#if ($(proxy))
  Your proxy called $(proxy.getName()) is loaded.
#else
  <form action="$(actionURL)" method="POST">
    <input type="text" name="myproxy_username">
    <input type="text" name="myproxy_password">
    <input type="submit" name="actionMethod_getProxy">
        value="Get Proxy">
  </form>
#end
```

Figure 2. The velocity template snippet illustrates basic Grid portlet development concepts. The template is described in more detail in the text.

This paper is organized as follows. We first review our efforts in Figure 1's User Interface tier. We discuss OGCE's support for Grid portlet development through Velocity. Grid programming interfaces for portals and services using the Java CoG kit are discussed in the following section. Packaging portals is described in the section on GridPort. We then turn our focus to the middle-tier Grid portal services. These include Grid Portal Information Repository services, Sakai collaboration services, Tupelo data management services, and science application service support.

The use of available standards, such as portlets and Web services, enables a great deal of interoperability and provides us with a foundation. Implementing sophisticated portlet clients and services (such as GPIR and Tupelo portlets and services and Sakai portlets) is one of our major activities. However, we must go beyond this foundation to provide a common infrastructure. We highlight here two of these efforts. First, the OGCE, through Java CoG Kit development, has developed a common abstraction layer for developing Grid portlets. This is described in more detail in Section 3, but these developments permeate several projects, including Velocity portlet support (Section 2), various OGCE compatible portlets (Section 4), and various services (such as Tupelo, Section 7). Second, the JSR 168 specification does not support inter-portlet communication and data sharing. This is crucial for Grid portlets, as Grid credentials (typically obtained from a MyProxy server) must be shared with other portlets (GridFTP and Job Submission portlets, for example). The OGCE has developed inter-portlet data sharing services and a simple API used when developing portlets. This is based on a shared singleton design pattern that uses the common CLASSPATH space (/shared/lib in Apache Tomcat) of the Web server and is used to support inter-portlet sharing of credentials. However, the method is general and can be used to share any Java object.

We would finally like to emphasize the packaging and distribution of portlets. JSR 168 provides interoperability of portlets in multiple containers, but the OGCE has emphasized this. The core portlet downloads are built and white-box unit tested (using HttpUnit) with an Apache Maven-based build process. This process supports both uPortal and GridSphere deployments, and it may be extended to support other JSR 168 compatible containers.

2. BUILDING GRID PORTLETS WITH VELOCITY

The portlet API [5] is a very important standard for portal component reusability, but it leaves much to be desired as a development environment: it provides a very basic 'model-view-controller' programming interface for building portlet applications. This superficially seems to compete with more mature and powerful Web application development tools such as Velocity, Struts, and Java Server Faces. Furthermore, one may be concerned that previous development work using these more powerful tools is incompatible with the portlet standard. Fortunately, this is not the case, and one may build bridges between these different tools. As described in this section, the OGCE has developed support for portlet standard-compliant Velocity.

Velocity portlets use the Apache Velocity [9] template engine to render returned markup fragments. Velocity template files contain HTML markup interspersed with expressions that allow access to Java objects and methods. The Velocity portlet code creates a Velocity 'context', a simple mapping of strings to Java objects, and passes this context, along with the template to be rendered, to the Velocity template engine. If one of the especially escaped strings in the template (e.g. '${gridJob}') matches one of the strings in the context mapping (e.g. 'gridJob'), then that template variable is associated with the mapped Java object.

There are several reasons behind OGCE's interest in Velocity. The Velocity template language has the virtue of being powerful, simple and unobtrusive; the Grid portal developer has powerful programming constructs to create a dynamic portlet view, but is not encumbered by complicated, special markup. Another benefit is that Velocity enforces a clean separation of code from presentation, and thus facilitates Model-View-Controller style development. It does this by allowing only the template writer access to Java 'Context' objects and their methods; the template writer may use simple programming language constructions (such as control loops and conditionals) in the template but does not need to know any Java programming. This has the added benefit of removing the temptation to put too much Java code in the HTML presentation layer. In addition to these benefits, interest in Velocity portlet development also has historical roots. OGCE is based on earlier work in portal frameworks such as Jetspeed and CHEF, and the *de facto* portlet development model in those legacy frameworks included using Velocity templates. Hence, several OGCE portlets and portlets developed by users of OGCE were and are developed using Velocity templates. It is thus advantageous to be able to port Velocity portlets from the legacy frameworks to the new standards-based frameworks.

For these reasons, OGCE has developed a JSR-168 compliant generic Velocity Portlet utility from which:

(1) Velocity portlets created in legacy portal frameworks could be ported to new, standard compliant containers;
(2) new, standards-based Velocity portlets could be developed by users of OGCE.

The OGCE generic Velocity Portlet takes care of the task of loading and rendering Velocity templates, so that portlet developers simply need call a setTemplate() method with the name of the template to use. The generic Velocity Portlet also provides a facility by which links and buttons in a portlet's Velocity template can invoke action methods in the portlet code. This is accomplished by having a request parameter with a special prefix and the action method name as the suffix; the Velocity Portlet code looks for a request parameter with the prefix and then uses Java reflection to invoke the specified action method. Finally, this utility provides the template writer with several convenience context variables. Some examples are references to the portlet request, response and preferences objects, and special URLs that the portal understands as referencing the portlet from which they come and that can be used to invoke one of the portlet's action methods or switch to the Help or Edit views.

A snippet of a Velocity template is provided in Figure 2 for illustration. In line 1, `${proxy}` is a context variable reference to some grid proxy credential object, and in lines 1, 3, and 10, there is a familiar if/else/end construction. Here the test is on whether the grid proxy credential object is available. Line 2 demonstrates how to invoke an object's method within Velocity. In line 4, the `${actionURL}` is one of the convenience context variables provided by the generic Velocity Portlet class; using this URL this form will be submitted to the portlet from which it came. In line 7, 'actionMethod_' is the special action method prefix used here in the `actionMethod_getProxy` submit button, signifying that 'getProxy' is the method that should be invoked to handle this form submission. OGCE's Velocity support is documented at the OGCE website and within the OGCE release.

OGCE Velocity portlet support can be used to build general purpose Velocity portlet applications. Our Grid portlet development integrates Velocity action methods with the Java CoG kit, described in the next section. As part of our core Grid portlet release, we provide Velocity-based, JSR 168 compliant portlets for user Grid credential management, job submission and monitoring, GridFTP-based remote file management, and system information view (the Grid Portal Information Repository, described below). We verify portals for deployment in uPortal [10] and GridSphere [11] portal containers. Our portlets depend on no container-specific services and can work with multiple grid installation versions simultaneously, as described below. Inter-portlet data sharing services, based on the singleton design pattern, can be used to share Grid credentials between portlet applications.

3. GRID ABSTRACTION LAYER: THE JAVA COG KIT

Grid and Web service architectures provide a common framework for building science gateways, but we must often go beyond these for several reasons: deployed Grid service middleware includes pre- and non-Web service legacy systems, Web service implementations from different toolkit versions, and services with overlapping functionality but incompatible service definitions. Thus, there is a need for an abstraction layer that can hide these differences.

The Java CoG Kit [12,13] has for years provided support for the development of Grid portals based on Java technology based on user requirements [14]. Much of the recent Java CoG effort with the OGCE has been devoted to providing a higher level abstraction for Grid toolkits: basic capabilities use object-oriented design principles to 'abstract' the Grid middleware, allowing the integration of the significantly different Grid middleware toolkits available today. Such an abstraction protects the investment of the portal and application developers, who may build upon basic tasks such as single sign on, credential management, file transfer [15], file access, and job execution [13]. In addition, the Java CoG Kit also includes a sophisticated but easy-to-use workflow management framework [16]. The Java CoG Kit provides such an abstraction as part of a Java API and an XML-based workflow framework.

The Java CoG Kit has evolved from a project that exposes much of the Globus Toolkit functionality through Java to a framework that contains a significant feature enhancement to the Globus Toolkit architecture. Based on its features, it is used by, and distributed with the Globus Toolkit versions 3 and 4, and OGCE versions 1 and 2. To these technologies belong, for example, the GSI-based Java security libraries, the GridFTP client libraries, most of the client libraries to the classic (pre-Web service) Globus Toolkit, and the myProxy credential client libraries. The Java CoG integrates a variety of commodity tools, protocols, approaches, and methodologies, while simplifying the access to Grid middleware.

3.1. Advanced features

In order to support our vision of simplifying Grid portal development, we have identified a number of higher level abstractions, including Grid tasks, transfers, jobs, queues, hierarchical graphs, schedulers, workflows, and control flows. However, in contrast to other Grid efforts, we have provided a mechanism in our framework that allows the integration of a variety of Grid and commodity middleware in an easy-to-comprehend framework based on the concepts of protocol independent abstractions, providers, and bindings, which are discussed below.

- *Providers.* We have introduced the concept of Grid providers to allow the *binding* of different Grid middleware services at runtime. The programmer thus does not need to worry about the particularities of the Grid middleware and version differences, and so can instead focus on the actual functionality. Through dynamic class loading, we have the ability to carry out late binding against an existing production Grid. This allows, for example, a single portlet client to work with multiple Grid versions, chosen (directly or indirectly) by users through browser requests.

- *Abstractions*. We have identified a number of useful abstractions that help in the development of elementary Grid applications. These abstractions include job executions, file transfers, workflow abstractions, and job queues. These can be used by higher level abstractions for rapid prototyping. As the Java CoG Kit is extensible, users can include their own abstractions and enhance the functionality of the Java CoG Kit.
- *Bindings*. Through these concepts, the Java CoG Kit protects development investments by protecting them from changes to the Grid middleware.

Based on these elementary concepts, we designed a layered architecture that allows the gradual enhancement of capabilities within a portal design need.

3.2. Advanced adaptability

The architecture of the Java CoG kit is discussed in more detail in [13] and [16]. The layered approach of the Java CoG kit provides adaptability towards different Grid middleware versions, as depicted in Figure 3.

Many of the features of the Java CoG Kit are now integrated as part of OGCE as JSR 168 portlets. They are also available as part of a prototype science desktop that can be accessed through Java Web Start. In the future, we will expose our workflow services in a more integrated fashion and enable users to manage their job queues and workflows.

4. PACKAGING PORTALS: THE GRIDPORT TOOLKIT

The latest GridPort release reflects significant changes in the design philosophy typically adopted by Grid portal toolkit developers [17]. Typically, a comprehensive and encompassing API is required to interface to a growing number of Grid services. However, these operate only within the portal framework (e.g. GPDK, GridPort versions 2 and 3 [18], GridSphere, CHEF). As the Grid becomes more commonly used and adopted within the commercial arena, this approach is not sustainable. There is a need to move away from programming to an inflexible API towards interoperable but independent, pluggable component portlets and services. The interoperability must occur at the service level (which is what services oriented architectures (SOA) delivers). GridPort 4.0 reflects two primary approaches: at the architectural level, we see that in order for portal development to scale with current technology trends, the use of SOA, Web services, and a portal component approach (WSRP, JSR-168) are required [19]. Additionally, there is a need to provide simple, robust, and easily installed demonstration portals for novice users.

With these goals in mind, the portlet/user interface tier has been separated from external services, including local portal services (the GP4 architecture is shown in Figure 4). This diagram reflects the layered approach of OGCE, as shown in Figure 1. Where possible, the portlet implementation uses existing APIs (e.g. Java CoG, the Storage Resource Broker [20], etc.). All portlets depend on the

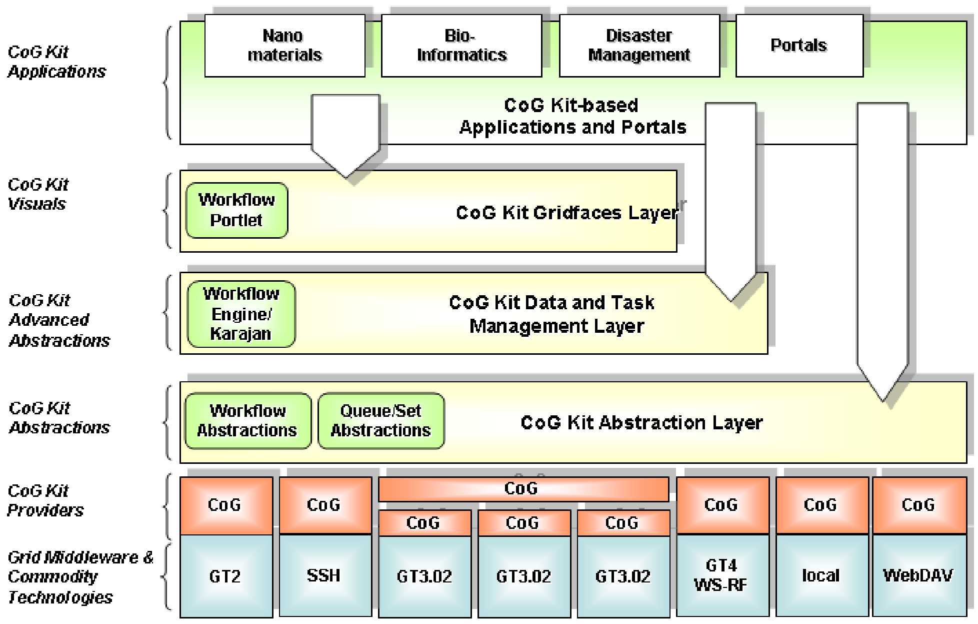

Figure 3. The layered approach of the Java CoG Kit provides mechanisms for abstracting Grid middleware, providing a common programming interface for building portlets to interact with Grid job submission and remote file management services, and it may be extended to support additional services, such as information management. The implementation of the abstraction layer uses dynamic class loading that allows OGCE portlets to interact with different Grid toolkits: the toolkit version of the underlying Grid service is a parameter set by the portlet at runtime, allowing the same running portlet to interact with (for instance) both pre-Web service and Web service versions of the Globus Toolkit.

OGCE portlet credential exchange mechanism. In addition, GridPort contains a Certificate Repository Service for locally storing proxies and there are plans to develop other services for data caching, resource description, and tracking grid usage of individual users. GridPort installs easily via automated installation with Maven (no need to download related libraries, this is done automatically). It also contains a new set of demonstration portlets as well as standalone versions of the GridPort 3.x Web services. The result is an easily installed basic portal that can have users up and running with minimal supporting software installation and Grid configuration procedures. Additionally, the portlets and services can be deployed independently and used by other portals or Grid systems.

The latest release of the GridPort toolkit is GP4.0 [19]. The portal features OGCE portlets, as well as several container services that are being integrated into the OGCE core. The demo portal is composed of portlet components with the same (or advanced) capabilities of previous GridPort portals providing basic Grid access and capabilities including the GPIR Browser for Grid information, GRAM Job Submission, Condor Job Submission, GridFTP file management, and proxy manager portlets as well as MyProxy and GridPort Repository single sign-on Grid authentication modules. The GridPort

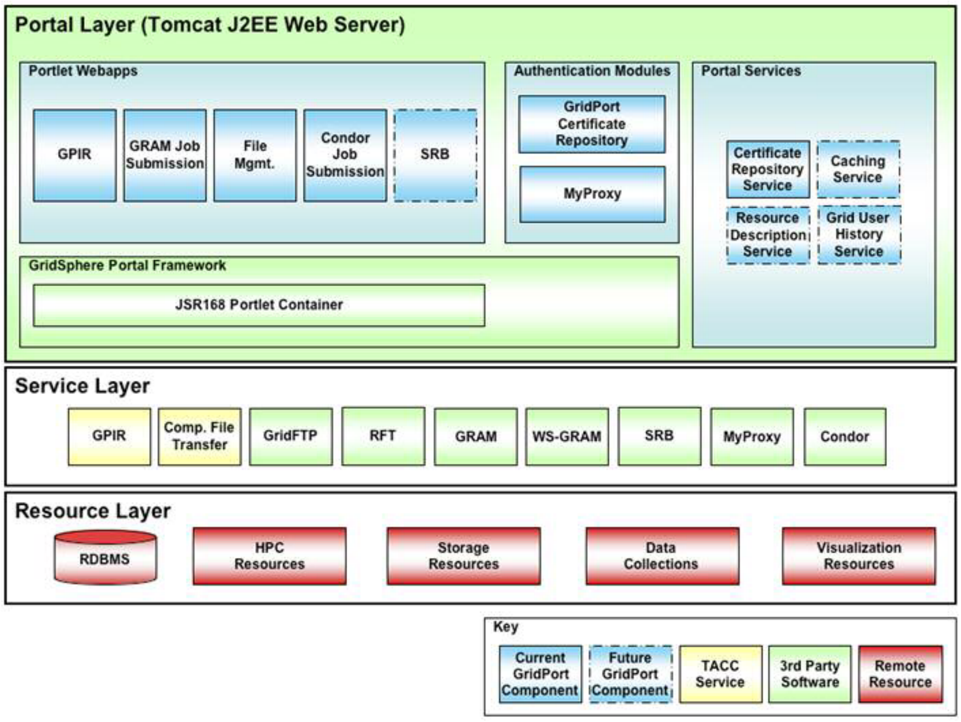

Figure 4. The GridPort architecture follows the general architecture of Figure 1. Grid Portlets using the Java COG Kit interact with remote Grid services. Additional OGCE-compatible portlets interact with supplemental Grid services such as GPIR (described in Section 5) and Comprehensive File Transfer.

demo portal installation takes care of installing the GridSphere portal framework, the GridPort portlets, Grid authentication modules and a custom look and feel into a Tomcat Web server.

GridPort 3.0 included Web services that, upon installation, were installed into a single JBoss container which made it difficult for users to install only the services they wanted. In GridPort 4, the GPIR and Comprehensive File Transfer Web services have been extracted from GridPort 3 and can now be installed separately. JBoss dependencies have also been removed so that all service components install easily into Tomcat. These services now each include a ready-to-deploy lightweight Hypersonic SQL database that makes installing and using them easier than ever before.

5. INFORMATION SERVICES: GRID PORTAL INFORMATION REPOSITORY

While there are numerous sources of information in a Grid environment, we have found that it is necessary to provide a convenient location to store portal related data. The aim is not to replace other

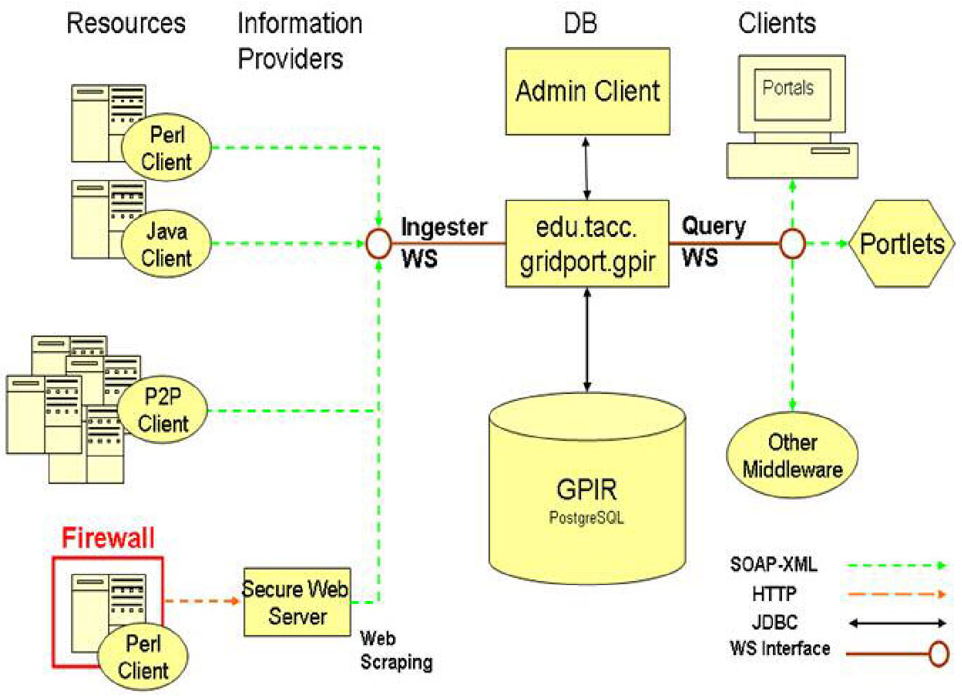

Figure 5. GPIR Service architecture provides Web services to ingest (insert into databases) and query information resources. Web service clients implemented in portlets provide multiple user views to ingested data.

information providers such as Globus MDS2 [21] but, rather, to aggregate and cache Grid and portal-related data in support of rapid and easy portal development. Specifically, the Grid Portal Information Repository (GPIR) provides GridPort with its data persistence needs. The architecture of GPIR is depicted in Figure 5. Perhaps most fundamentally, GPIR provides a place to store data about your Grid that is readily accessible to a portal application. This includes both dynamic data and 'human-centric' data (such as where a resource is located or whom to call for support): (a) 'dynamic' machine-oriented data is updated via a Web service called the 'GPIRIngester' using information providers and data reads are performed via the 'GPIRQuery' Web service; and (b) 'human-centric' data is managed through the GPIR Administration client which is a standard Web application accessed via a browser.

Data sources can be divided into four types: machine gathered or automatic, which can further be divided into standards-based and custom providers; and human entered information, which can be split into 'facts' or observations on the one hand and 'decisions' on the other. Standards-based data sources would include things such as the Globus MDS or Ganglia [22]. By contrast, custom providers might be written to query the resource manager on each resource and report back to GPIR. Factual data might include aspects of the system that exist in the social realm such as the Virtual Organization membership of a system, a textual description of the configuration and general purpose of a system, or

```
- <xs:schematargetNamespace="http://grids,tacc,utexas.edu/schems/infoservices/jobs" elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  - <xs:element name= "ComputeResourceJobs">
     - <xs:complexType>
       - <xs:sequence>
          - <xs:element name="Jobs" maxOccurs="unbounded">
             - <xs:annotation>
                 <xs:documentation>attributes: timestamp and hostname</xs:documentation>
               </xs:annotation>
             - <xs:complexType>
                - <xs:sequence>
                   - <xs:element name="Job" minOccurs="0" maxOccurs="unbounded">
                     - <xs:annotation>
                         <xs:documentation>includes id attribute</xs:documentation>
                       </xs:annotation>
                     - <xs:complexType>
                       - <xs:sequence>
                           <xs:element name="Status" type="xs:string"/>
                           <xs:element name="Owner" type="xs:string" minOccurs="0"/>
                           <xs:element name="Queue" type="xs:string" minOccurs="0"/>
                           <xs:element name="SubmissionTime" type="xs:dateTime" minOccurs="0">
                           <xs:element name="StartTime" type="xs:dateTime" minOccurs="0"/>
                           <xs:element name="Name" type="xs:string" minOccurs="0"/>
                           <xs:element name="Project" type="xs:string" minOccurs="0"/>
                           <xs:element name="Executable" type="xs:string" minOccurs="0"/>
                           <xs:element name="Priority" type="xs:string" minOccurs="0"/>
                           <xs:element name="RunningOn" type="xs:string" minOccurs="0"/>
                           <xs:element name="CPUTime" minOccurs="0">
```

Figure 6. GPIR Job schema can accept many different parameters to describe a job running on a resource.

the departmental and institutional affiliation of the resource, while 'decisions' are aspects of a system that are dependent upon human intervention or decisions such as a system's next scheduled downtime.

When GPIR was first developed there were no standard schemas that could hold all of the data that we required; so we developed some custom schema that would meet our needs. Available schemas include static machine data for information that cannot be provided by information providers, load, status, downtime, jobs, message of the day, nodes, and one for reporting Network Weather Service [23] data. Full schemas are available at [17]. Figure 6 shows part of the job schema, which allows specification of job properties such as status, owner, queue, submission time, and executable among others.

We must provide also a rich user interface to make use of GPIR services. GPIR portlets are a core part of the OGCE release and are developed using Velocity tools described in Section 2. The out-of-the-box OGCE portlets include GPIR portlets that point to default services but which can be easily configured to point to local GPIR installations.

6. COLLABORATION SERVICES USING SAKAI

Much of e-Science activities consist of some combination of collaboration between people, access to common resources, and dealing with scientific data across the scientific endeavor. The Sakai project provides an 'out-of-the-box' portal download, but more recent releases have also featured two important developments in accordance with the principals of Figure 1: Sakai collaboration tools are available as Web services and the OGCE has developed JSR 168-compatible portlet interfaces to these services. Sakai tools have a much richer concept of authorization than is supported in JSR 168-compatible containers, so developing a full integration is an interesting, on-going effort.

Sakai's organizing principle is a 'Site'—users can be members of many sites. Each site has its own set of tools and resources. Sites can be created for many purposes: an 'All Hands' site for all members of the collaboration, a site for a particular committee, a site for a particular experiment. Each site has its own 'contextualized' set of tools and resources that can be used by the members of the site. Tools that can be included in a site include: threaded discussion, e-mail archive, schedule, announcements, chat and others. In addition to tools, each site has its own content repository capable of storing files and folders accessible to the members of the site. Each site owner(s) maintain the access control and roles for the site. A unique aspect of Sakai is the ability quickly to create a new site and let users join the site, enabling a very organic approach to collaboration.

While there are many ways to support chat, threaded discussions, or mailing lists, Sakai is unique in the notion of contextualizing the activity around a site. This context can be used to organize data outside of Sakai as well; but using a consistent set of contextualized tools for collaboration, it is possible to capture virtually all of the collaborative activity as data. By closely associating the data from the collaborative activity with the corresponding data from science activity, it is possible to enhance significantly the value of scientific data.

In OGCE's Release 1, we provided a single solution that attempted to solve all of the problems of e-Science infrastructure with a single software product. While many adopters found OGCE 1 to be a rich environment 'out-of-the-box' because it was a self-contained solution, it was difficult to expand OGCE 1. In OGCE 2 we are developing a more decoupled, component-based approach providing each of the major components of e-Science (collaboration, portal, and data) using separate technologies that are highly capable and extendible and focusing on the integration of those technologies. Much of the effort in OGCE with respect to Sakai is in the area of smooth integration with the other elements of the OGCE product. This effort falls into two principal categories: adding Web services to Sakai, and building a Sakai JSR-168 Portlet

The 2.1 release of Sakai includes full integration of Apache Axis 1.2. This potentially allows any of the Sakai APIs to be accessed remotely over Web services. Since the Sakai APIs provide access to all of the functionality of Sakai, it is possible to do any operation in the Sakai GUI over Web services. The currently supported services include synchronization of account and group information, automatic creation and maintenance of Sakai sites, and building of different presentation mechanisms (Visual Basic, PHP, and JSR-168) for Sakai sites and materials. When combined with JSR 168 clients for managing the Web service clients, we can integrate Sakai tools into standard-compliant portals, such as shown in the OGCE-based LEAD portal screen shot, Figure 6. Current community work is underway to build Web service interfaces to all Sakai services, which will allow us to build non-user interface clients to Sakai services. We thus see a very promising integration of science application and collaboration tools in the near future. For example, one may potentially integrate science application management tools with calendar tools to track and manage work.

Since the OGCE project is focused on portals and the JSR-168 standard, it was very important to develop a JSR-168 portlet that allowed Sakai elements to be placed in the JSR-168 portals supported by the OGCE project (uPortal and GridSphere).

The recently developed Sakai Web service Portlet (shown in Figure 6) uses Sakai's Web services to log automatically the portal user into Sakai and create and provision their Sakai account as necessary. Once the user is logged in, the list of Sakai sites which the user can access is retrieved using standard

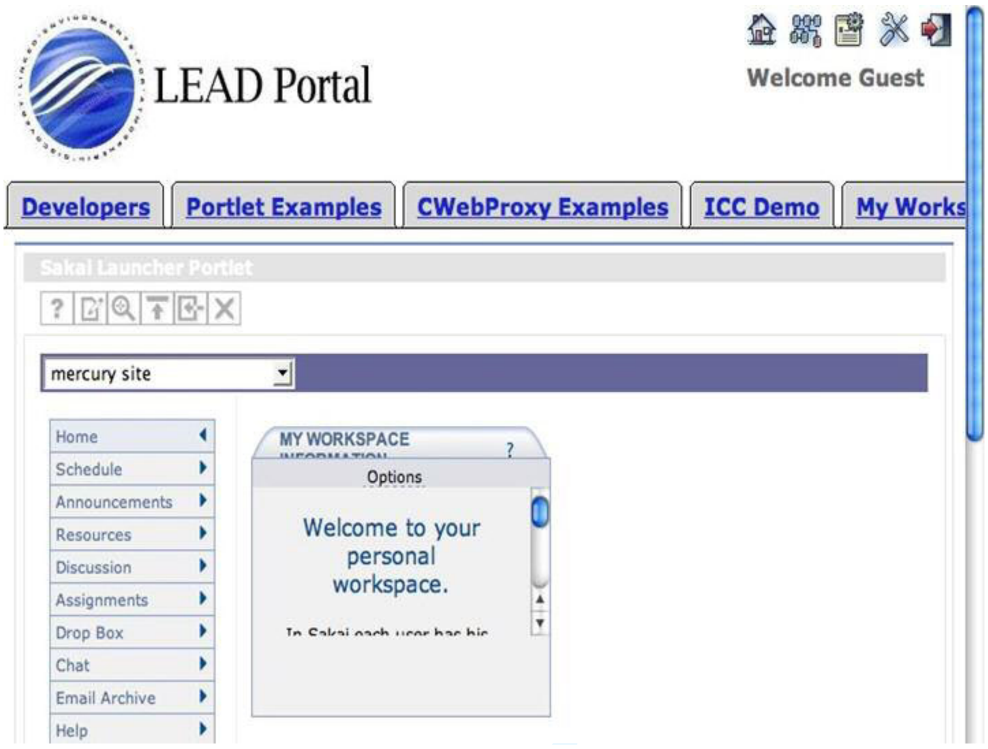

Figure 7. Sakai tools can be made available to portals using JSR 168 portlets and Web services. Illustrated is the Sakai Launcher Portlet (and the associated Sakai Web content) in the OGCE-compatible LEAD project's portal. This version uses a JSR 168-compatible uPortal container.

Web service request/responses. These sites are then presented to the user so that the user can choose between their Sakai sites. Upcoming versions of this portlet will allow the placement of a single site or a single tool within the JSR-168 portal. The goal is to make the Sakai capabilities as flexible as any other portlet within the portal. Sakai effectively appears as a set of virtual JSR-168 portlets in the portal. An example of the Sakai portlet is shown in Figure 7.

The long-term plan for Sakai is to fully integrate the tools into the portal as portlets. However, with the current capabilities of JSR-168, the full integration of Sakai into the portal cannot be done without portal-specific modifications. The Sakai JSR-168 portlet allows a level of integration between Sakai and JSR-168 portals without compromising portability between JSR-168 compliant portals.

7. CONTENT MANAGEMENT: THE TUPELO SEMANTIC CONTENT REPOSITORY

Tupelo [24] is a Grid-based semantic content repository designed to support collaborative data sharing within and between geographically distributed communities. Tupelo supports content versioning, transactional atomicity, and change tracking for all content. In addition, it provides standards-based means of defining arbitrary, user-specified content and metadata schemas. Finally, its API is accessible using Grid services, providing public-key authentication and fine-grained access control. Tupelo represents a working example of the kind of repository being envisioned as part of a next-generation semantic Grid [25]. Tupelo is an outgrowth of the NEESgrid [26] data repository and so has proven applicability in scientific data management. Examples of its use in NEESgrid can be found at [27]. Tupelo's architecture is general and may be applied to general content and data management problems. A complete description of Tupelo is beyond the scope of this paper; see [24] for details.

7.1. Tupelo design principles

Like content management systems (CMSs) and CMS APIs such as Plone [28] and JSR-170 [29], Tupelo provides applications with a storage-neutral means of managing content and associated metadata. Unlike most CMS APIs, Tupelo also allows applications to create and manage metadata schemas, using the Web Ontology Language (OWL) [30]. Tupelo metadata is 'self describing' in the sense that content items are explicitly linked to classes describing them, and can be validated against those classes using Tupelo and OWL APIs. Strong typing of attributes and complex constraints such as controlled vocabularies, class membership, and numerical ranges can be represented, imported, exported, and enforced using generic mechanisms without having separately to maintain application-specific code for carrying out these general tasks.

Tupelo was designed for archiving evolving community data resources, and can accommodate multiple, dynamic, potentially overlapping schemas using namespace-scoped identifiers and schema evolution via version control and change tracking. In addition, its OWL support means that independently developed schemas can be merged and checked for logical consistency using standard OWL reasoning APIs.

Tupelo was designed to support distributed project groups and supports fine-grained access control on all content objects. The Tupelo API is exposed as a Web service using the Open Grid Services Infrastructure (OGSI), and links Grid Security Infrastructure (GSI) authentication to Tupelo's access control system. We plan to adopt the Web services Resource Framework (WSRF) for future releases of Tupelo. The Web service API allows lightweight clients to securely perform meaningful operations on large quantities of metadata without residing in the same runtime environment. This is a deliberate analogy to Grid computing, in which access to computational resources is exposed as a service [2].

7.2. Implementation

Because Tupelo is an archiving system, it is designed to scale to large numbers of objects and files. Although OWL was chosen as the standard means of importing and exporting schema and instance information, most OWL implementations are designed for declarative, sequential-access, batch processing modes. In Tupelo, classes and objects are compiled to database records to take advantage of database indexing. No sequential-access formats are used in Tupelo's storage implementation, but metadata can be imported/exported in OWL XML format for interoperability with existing OWL tools.

Tupelo's access control implementation is based on global, rather than local identities, which allows it to interoperate with certificate-based authentication systems without requiring difficult-to-maintain identity mappings or access to directory services. For instance, GSI-authenticated users can

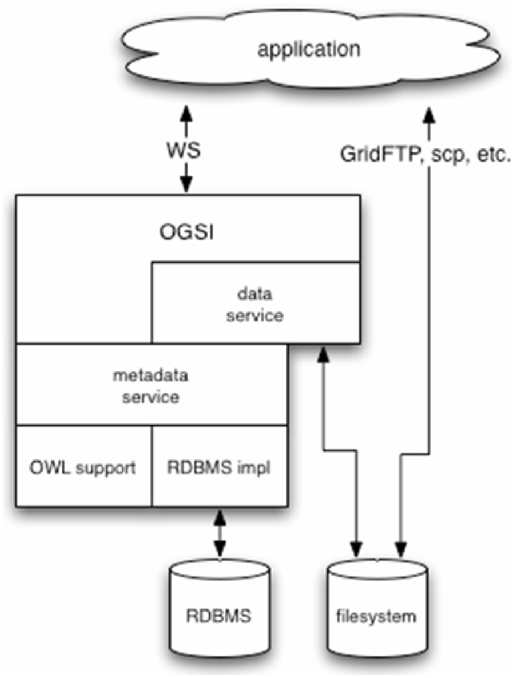

Figure 8. Tupelo 1.1 architecture. Tupelo's metadata service uses an RDBMS (Oracle, MySQL) to store metadata objects, OWL schemas, version control information, and access controls. The data service manages file version data in a filesystem and file metadata using the metadata service. Matadata service calls and data service control calls are made via the OGSI-based Web services interface. Data can be transferred to and from the file service using any protocol; the data service manages staging and access control. The COG4 API is used both internally and on the client side to provide the transfer abstraction.

be organized into access control groups and granted permissions based on their X509 Distinguished Name. Authorization decisions are implemented as SQL joins to achieve scalable performance over large numbers of objects, allowing for fine-grained access control. The Tupelo architecture is shown in Figure 8.

Tupelo's APIs represent a rough superset of the capabilities of a variety of related technologies, including not just CMS APIs but protocols such as WebDAV. A number of these APIs and protocols are candidates for Tupelo to support.

A full technical description of Tupelo is beyond the scope of this paper. Interested readers may learn more about the architecture and portlet programming interface from our GGF 15 Community Workshop [31]. To give a general view of how clients are developed, we provide below a sample fragment of portlet client code for uploading a file and associating it with metadata

```
Context sessionContext = ...
DataServiceFacade dsf = new DataServiceFacade.(sessionContext);
MetadataServiceFacade msf = new MetadataServiceFacade(sessionContext);
```

```
LogicalName ln = new LogicalName(new Identifier("{http://ns}results"));
dsf.putNewVersion(ln, new File("/tmp/results.dat"));
MetadataObject obj = msf.retrieve(ln);
obj.setString(new Identifier("{http://purl.org/dc/elements/1.1/}creator"),
              "Joe Futrelle");
msf.update(obj);
```

In this example, the data service is used to upload local data to a new logical file. The logical name of the file is then used to retrieve a metadata object, which is modified by adding an attribute/value pair.

7.3. Tupelo and portals

The utility of Tupelo in a portal environment is that it can insulate independently developed portal components from needing *a priori* information about content items in order to perform meaningful operations on them. For instance, a data processing application can use Tupelo to validate objects generated by a data producing application against the objects' declared classes, eliminating the need to duplicate validation semantics in every consuming application. Tupelo can also be used efficiently to implement any of a number of strategies for grouping content items, rather than the strictly hierarchical grouping enforced by many CMS APIs, in which each relationship between levels of hierarchy and meaning is unspecified. Tupelo can also enable relatively generic portal applications to be built that can consume content schemas from Tupelo and configure themselves dynamically to provide users with the ability to browse, edit, manage, and query arbitrary types of content. Tupelo's ability to accommodate multiple, evolving schemas provides a data integration framework for developing portal components that manage complex workflows, distributed collaboration, and organizational memory [32].

Tupelo can also serve as a data management provider for workflow systems. It is suitable for capturing metadata and intermediate results as a workflow executes, including linking derived data to source data and processing attributes. We plan to integrate Tupelo with COG4's workflow engine, both as a file transfer provider and as a means of producing and extracting metadata to drive dependencies and conditions in the workflow graph. This will enable Tupelo to interoperate with a variety of other providers in the COG4 framework without requiring workflow authors to use Tupelo-specific APIs.

At the time of writing, Tupelo 2 is under preliminary development. Tupelo 2 is a redesign of Tupelo to integrate better with standard semantic Web and content management technologies, including replacing the RDBMS layer with Kowari [33] and the OGSI layer with the Java Content Repository API [34]. Tupelo 2 is being designed to support cyberinfrastructure portal efforts at NCSA, including the MAEviz [35] and ECID [36] projects, where it will serve to manage provenance information generated by complex workflows.

8. SCIENTIFIC APPLICATIONS AS PORTAL SERVICES

In the previous section we reviewed the use of Tupelo services for data management. This has been applied to problems in scientific data and metadata management, although the approach to content management is general. In this section we review scientific applications that produce and consume this data.

Most portals devoted to a scientific application domain provide a way for the portal user to supply parameters to preconfigured applications and then execute them. For example, a life-science portal would provide a way for a user to supply input data to an analysis program such as BLAST. The portal would take this data and execute BLAST on a Grid computing resource and return the result back to the user. There are several ways to do this. The obvious first idea is to create a portlet specifically for the application. It can be customized to present the user with clear instructions on what data is required as input to the application together with the forms for uploading the data or supplying needed parameters and a button 'execute' which, when pressed, will cause the portal to submit the job to some remote system and wait for the result to return. This approach will work but it has several limitations.

- It is labor intensive—a programmer must build a portlet for each application and most scientists who have applications they want integrated into the portal are not portlet programmers.
- For many applications, it is not practical to make a user wait for a portlet response to finish the job execution. The application execution may take several hours. Consequently, some sort of session state must be maintained in the portal to capture the results of an execution, even when the user has logged out.
- In many cases the user may want to incorporate the application into a larger workflow. In this case it is more useful to have the application available as a service that can be invoked from a tool such as Taverna [37], Kepler [38], the Java CoG (described above), or BPEL based tools [39].

A better approach is to separate the application execution management into a Web service that may be invoked from the portal or from a workflow. This idea of 'wrapping' an application as a Web service is not new [40]. More recently, Soaplab [41] has been developed by the European Bioinformatics Institute to accomplish similar tasks. Our most recent version of the Web service application wrapper, SecureGFac [42] is now being used in the LEAD gateway project, and both portlets and services are available through the OGCE.

SecureGFac is a toolkit that allows application providers to 'wrap' their applications as secure Web services. We will call such Web services application services. SecureGFac has two components: a 'Generic Service Factory' that runs in the portal and is accessible through the Generic Service Factory portlet in the portal. To create an application service, the scientist must follow these steps. First, deploy the application on a Grid host. If an automatic deployment service is available, this can be used instead, but this not in SecureGFac's scope. Second, describe the application. This process involves filling in a 'service map document' (implemented as a portlet) that requests information including: the input and output parameters of the application, a list of environment variables needed to run the application on the specified host and a list of users or groups that are allowed to invoke the service to run the application. Third and last, upload the service map document through the Generic Service Factory Portlet in the Portal.

A sample ServiceMap document is shown below. It contains information about the service port-types, including their operations and the input and output parameters. A ServiceMap document has two main elements: *service* and *portType*. The *service* element describes the service. It has a fully qualified name of the service and a short description of the service. It can also contain metadata about the service. In our simple example of a 'MyApplication' service, the document has the top level structure shown below.

```
<ServiceMap xmlns="http://www.extreme.indiana.edu/namespaces/2004/01/gFac"
            xmlns:lead="http://www.extreme.indiana.edu/lead">
```

```
<service>
  <serviceName targetNamespace="http://www.extreme.indiana.edu/lead">
    MyApplicationService
  </serviceName>

  <serviceDescription>
    This is a web service "wrapper" for 'MyApplication'
  </serviceDescription>
</service>

<portType>
  <method>
    <methodName>run</methodName>
    <methodDescription>
      When invoked, this method will run 'MyApplication'
    </methodDescription>

    <application>
      <applicationName
          targetNamespace="http://www.extreme.indiana.edu/lead">
        MyApplication
      </applicationName>
      <description>
        This is my application. It takes an integer as a command
        line argument and returns the name of the output file that
        it produces
      </description>
    </application>

    <inputParameter>
      <parameterName>inputFile</parameterName>
      <parameterType>String</parameterType>
      <parameterDescription>
        The path to the input file
      </parameterDescription>
    </inputParameter>

    <inputParameter>
      <parameterName>outputDirectory</parameterName>
      <parameterType>String</parameterType>
      <parameterDescription>
        The directory where the output file should be stored
      </parameterDescription>
      <metadata>
        <name>AccessServiceType</name>
        <value>GridFTP</value>
      </metadata>
    </inputParameter>

    <outputParameter>
      <parameterName>outputFileURL</parameterName>
      <parameterType>String</parameterType>
      <parameterDescription>
        The URL of the output file that was created in the output directory
        </parameterDescription>
      </outputParameter>

      <securityPolicy>
        <groupName>MyFriends</groupName>
        <lifetime>24:00:00</lifetime>
      </securityPolicy>

    </method>
  </portType>
</ServiceMap>
```

The *portType* element contains a list of operations (also known as methods). Each operation has a name, a security policy, a description, metadata and a list of default values if any. The main method in our example is 'run'. It takes two input parameters; a file name that we will call 'inputFile' and the output directory that we will call 'outputDirectory'. The complete specification of this method is shown below. There are several things to notice here. First, the run method has an associated 'application' element that uniquely identifies an application. When the 'run' method is invoked by a client, the service will execute the application associated with the run method i.e. MyApplication. Details about the host on which the application is deployed, the path to the application on that host, environment variables that are needed to execute the application and other application deployment details are specified in another document that we call the ApplicationDeployment document. Each parameter can have associated metadata, which can be used by the portal to help guide the user in the selection of a valid value for that parameter. In this case the metadata for the output parameter 'outputDirectory' specifies a requirement that the directory must be accessible by the GridFTP protocol. The output parameter called 'outputFileURL' specifies the URL of the output file generated by the application. This is the response sent to the client when it invokes the 'run' method. Finally, the *securityPolicy* element specifies that all users in the 'MyFriends' group are allowed to invoke the 'run' method (where 'group' is a list of users as recognized by the capability manager service). It also specifies a lifetime for the policy.

At this point the Generic Service Factory takes over. It launches a generic Web service on the remote host where the application is installed and passes the service map document to it. The generic Web service reads the service map document and dynamically configures itself into an application service that is ready to run the application. After the dynamic configuration is done, it generates the Web Service Description Language (WSDL, see [6]) that describes this specific application service. The service registers the WSDL with the registry service so that it may be discovered by clients. The service also notifies a 'capability manager' service with the list of authorized users and groups who can access the various operations in the service.

The application may now be invoked as part of a workflow or it may be directly invoked by a user from the portal. In the latter case, when a user logs into the portal, a list of applications can be searched to find the right one. Selecting the application causes two events to happen. First, the user's identity is used to check the capability manager for a capability token for that service. Next, if the user or a group to which the user belongs has been authorized to use the service, the portal loads the graphical user interface into the portal. The graphical user interface is generated by the application service. It describes all the operations that the user can invoke and allows the user to choose an operation, specify its input parameters and invoke the operation on the application service. The portlet that displays this user interface takes the user's response and translates this into a secure Web service request to the application service.

When the application service receives the request, it starts a thread that runs the client application. In most cases, the application request involves moving data to the host where the application runs. As the data movement and application are running, the application service thread sends event notifications about the application progress to an external event listener to form a log of progress. The portal or other desktop application can listen to this event stream so that the end user has some knowledge about the application's progress. This service factory tool has been used extensively in the LEAD [43] gateway portal to wrap data decoders, data mining tools, weather simulations, and graphical rendering engines. It has also been used in life science applications. A release of this software is available from [44] and [8]. The portlets associated with SecureGFac (the portlet for creating the application service map document and the portlet for application execution) are implemented in using Velocity, as described in Section 2. They use OGCE-compatible inter-portlet data sharing to share Grid credentials.

9. SUMMARY AND FUTURE WORK

We have surveyed the work and the efforts of the OGCE collaboration. The OGCE spans diverse research and development efforts: portlet development environments; Grid computing abstraction layers; advanced Grid services for information, science application, and data management; and services for group collaboration. By using accepted standards such as JSR 168-compatible portlets and Web services in a common portal architecture (Figure 1), we are able to integrate these diverse elements. OGCE tools are used in a diverse set of applications, including support plasma fusion simulations, weather modeling, biophysical simulations, and computational chemistry.

It is doubtful that any one portal will want or need all of the OGCE tool, so we attempt to provide a flexible set of modules that allows portal developers to download as much or as little as desired. Our current 'out of the box' delivery consists of core Grid portlets that may be built into the desired portlet container to provide the developer with a quick demonstration portal. Additional portlets and services may be downloaded from the OGCE Web site. However, as various OGCE services mature, we see the need to provide a single download process that more easily supports distributed development. We are working with several new tools (particularly new versions of the Apache Maven project) that can be adapted to provide a single but flexible download system that can support both minimal downloads (i.e. just providing Grid portlets) as well more comprehensive build processes that can set up entire science gateways at once.

Several challenges still face science portals and other science gateways, including fine-grained authorization and federated identity. These are addressed in related publications in this special issue. One particular challenge of general interest is the simplification of science portlet development. Portlets provide a coarse-grained level of integration and re-use. Environments such as Velocity simplify the portlet development process and application factories can be used to generate simple user interfaces, but we see the need for finer-grained components that can simplify the development of more complicated portlets, allowing them to be rapidly composed from re-usable Grid widgets rather than programmed.

ACKNOWLEDGEMENTS

The OGCE is funded by the National Science Foundation's Middleware Initiative (NMI) Program. The Java CoG Kit is also supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38.

REFERENCES

1. Foster I, Kesselman C (eds.). *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann: San Francisco, CA, 2004.
2. Berman F, Fox G, Hey T (eds.). *Grid Computing: Making the Global Infrastructure a Reality*. Wiley: New York, 2003. Available at: http://www.grid2002.org.
3. Hey A, Fox G (eds.). *Concurrency and Computation: Practice and Experience (Special Issue on Grid Computing Environments)* 2002; **14**(13–15).
4. Fox G, Gannon D, Thomas M. Overview of Grid computing environments. *Grid Computing: Making the Global Infrastructure a Reality*, ch. 20, Berman F, Fox G, Hey T (eds.). Wiley: New York, 2003.
5. Abdelnur A, Chien E, Hepper S (eds.). Portlet Specification 1.0, 2003. http://www.jcp.org/en/jsr/detail?id=168 [11 July 2006].
6. Booth D, Haas H, McCabe F, Newcomer E, Champion M, Ferris C, Orchard D. Web service architecture. *W3C Working Group Note*, 11 February 2004. http://www.w3c.org/TR/ws-arch [11 July 2006].
7. Singh I *et al. Designing Enterprise Applications with the J2EE^{TM} Platform* (2nd edn). Addison-Wesley: Reading, MA, 2002. Available at: http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/ [11 July 2006].
8. The Open Grid Computing Environments Web site. http://www.collab-ogce.org [11 July 2006].
9. The Apache Jakarta Velocity Project Web site. http://jakarta.apache.org/velocity/ [11 July 2006].
10. The uPortal Web site. http://www.uportal.org [11 July 2006].
11. The GridSphere Portal Web site. http://www.gridsphere.org/gridsphere/gridsphere [11 July 2006].
12. von Laszewski G, Foster I, Gawor J, Lane P. A Java commodity Grid kit. *Concurrency and Computation: Practice and Experience* 2001; **13**(8–9):643–662. Available at: http://www.mcs.anl.gov/∼gregor/papers/vonLaszewski–cog-cpe-final.pdf [11 July 2006].
13. von Laszewski G, Gawor J, Krishnan S, Jackson K. Grid computing: Making the global infrastructure a reality, Commodity Grid Kits—Middleware for building Grid computing environments. *Communications Networking and Distributed Systems*. Wiley: New York, 2003; 639–656. Available at: http://www.mcs.anl.gov/∼gregor/papers/vonLaszewski–grid2002book.pdf.
14. von Laszewski G, Foster I, Gawor J, Lane P, Rehn N, Russell M. Designing Grid-based problem solving environments and portals. *Proceedings of the 34th Hawaiian International Conference on System Science*, Maui, HI, 3–6 January 2001. Available at: http://www.mcs.anl.gov/∼gregor/papers/vonLaszewski–cog-pse-final.pdf.
15. von Laszewski G, Gawor J, Plaszczak P, Hategan M, Amin K, Madduri R, Gose S. An overview of Grid file transfer patterns and their implementation in the Java CoG kit. *Journal of Neural Parallel and Scientific Computing (Special Issue on Grid Computing)* 2004; **12**(3):329–352. Available at: http://www.mcs.anl.gov/∼gregor/papers/vonLaszewski-overview-gridftp.pdf.
16. von Laszewski G, Hategan M. Grid workflow—an integrated approach. Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60440, 2005. Available at: http://www.mcs.anl.gov/∼gregor/papers/vonLaszewski-workflow-draft.pdf.
17. The GridPort Project Web site. http://www.gridport.net [11 July 2006].
18. Thomas M, Boisseau J. Building Grid computing portals: The NPACI Grid portal toolkit. *Grid Computing: Making the Global Infrastructure a Reality*, Berman F, Fox G, Hey T (eds.). Wiley: New York, 2003.
19. Thomas M, Barker CJ, Boisseau J, Dahan M, Regno R, Roberts E, Seth A, Urban T, Walling D. Experiences on building a component-based Grid portal toolkit. *Concurrency and Computation: Practice and Experience* 2005 (accepted for publication).
20. Rajasekar A, Wan M, Moore R. MySRB & SRB—components of a data Grid, *Proceedings of the 11th International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, 24–26 July 2002. See also the Storage Resource Broker Web site: http://www.sdsc.edu/srb/.
21. Zhang X, Schopf J, Performance analysis of the globus toolkit monitoring and discovery service, MDS2. *Proceedings of the International Workshop on Middleware Performance (MP 2004), part of The 23rd International Performance Computing and Communications Workshop (IPCCC)*, April 2004.
22. Massie ML, Chun BN, Culler DE. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing* 2004; **30**(5–6):817–840.
23. The Network Weather Service Site. http://nws.cs.ucsb.edu/ [11 July 2006].
24. Futrelle J, Gaynor J, Plutchak J. Tupelo Wiki. http://tupeloproject.org/ [11 July 2006].
25. De Roure D, Jennings NR. Shadbolt NR. The semantic Grid: Past, present, and future. *Proceedings of the IEEE* 2005; **93**(3):669–681.
26. The NEESit Web site. http://it.nees.org [11 July 2006].
27. The NEESgrid Web site. http://www.neesgrid.org/ [11 July 2006].
28. Plone Foundation. Plone: A user-friendly and powerful open source content management system. http://plone.org/ [11 July 2006].

29. Nuescheler D. (ed.). JSR 170: content repository for Java technology API. *Java Specification Request*, 17 June 2005. http://jcp.org/aboutJava/communityprocess/final/jsr170/index.html [11 July 2006].
30. McGuiness D, Harmelen F. OWL Web ontology language overview. *W3C Recommendation*, 10 February 2004. http://www.w3.org/TR/owl-features/ [11 July 2006].
31. Pierce M *et al.* New technologies for science portals. *Global Grid Forum 15 Community Workshop*. http://www.collab-ogce.org/GGF15Workshop/index.html [11 July 2006].
32. Myers J, Chappell A, Elder M, Geist A, Schwidder J. Reintegrating the research record. *Computing in Science and Engineering* May/June 2003.
33. Kowari Metastore. http://www.kowari.org/ [11 July 2006].
34. Content Repository for Java technology API. http://www.jcp.org/en/jsr/detail?id=170 [11 July 2006].
35. MAEviz overview. http://alg.ncsa.uiuc.edu/do/tools/maeviz [11 July 2006].
36. Bajcsy P, Kooper R, Marini L, Minsker B, Myers J. A meta-workflow cyber-infrastructure system designed for environmental observatories. *Technical Report ISDAO1-2005*, NCSA Cyber-environments Division, 30 December 2005. Available at: http://isda.ncsa.uiuc.edu/peter/publications/techreports/2005/meta-workflow-approaches.pdf.
37. Oinn T *et al*. Taverna: Lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience* 2006; **18**(10):1067–1100. DOI: 10.1002/cpe.993.
38. Ludäscher B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones M, Lee EA, Tao J, Zhao Y. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience* 2006; **18**(10):1039–1065. DOI: 10.1002/cpe.994.
39. Slominski A. On using BPEL extensibility to implement OGSI and WSRF Grid workflows. *Concurrency and Computation: Practice and Experience* 2006; **18**(10):1229–1241. DOI: 10.1002/cpe.1004.
40. Gannon D *et al.* Programming the Grid: Distributed software components, P2P and Grid Web services for scientific applications. *Journal of Cluster Computing* 2002; **5**(3):325–336.
41. Senger M, Rice P, Oinn T. Soaplab—a unified Sesame door to analysis tools. *Proceedings of the U.K. e-Science All Hands Meeting*, 2–4 September 2003. See also http://www.ebi.ac.uk/soaplab/.
42. Kandaswamy G, Fang L, Huang Y, Shirasuna S, Marru S, Gannon D. Building Web services for scientific Grid applications. *IBM Journal of Research and Development* 2006; **50**(2–3):249–260.
43. Plale B, Gannon D, Reed DA, Graves SJ, Droegemeier K, Wilhelmson B, Ramamurthy M. Towards dynamically adaptive weather analysis and forecasting in LEAD. *Proceedings of the International Conference on Computational Science*, vol. 2, 2005; 624–631.
44. The Extreme! Lab portal software Web site. http://www.extreme.indiana.edu/portals [11 July 2006].