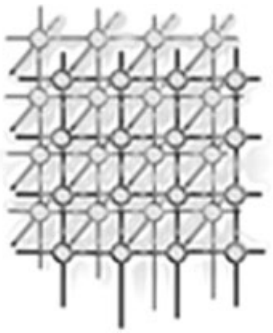
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2007; **19**:1671–1681
Published online 7 June 2007 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1198

Cooperative Grid Vortals



Tomasz Haupt*,† and Anand Kalyanasundaram

Cooperative Computing Group, Center for Advanced Vehicular Systems, Mississippi State University, Box 5405, Mississippi State, MS 39759, U.S.A.

SUMMARY

Vertical Grid portals, or Vortals, proved to be useful by providing access to applications running on high-performance computational platforms and to distributed data enabling data sharing and other forms of collaboration. This paper identifies the commonalities between Vortals in various application domains, i.e. application-independent patterns of orchestrating Grid services employed by a typical Vortal. These patterns can be captured as façades—a software layer between the Vortal business logic and generic Grid services that promotes a horizontal integration. The façades are reusable and considerably simplify the development of new Vortals. Furthermore, the façades foster interactions between different Vortals, and hence making them the cooperative Grid Vortals. Copyright © 2007 John Wiley & Sons, Ltd.

Received 30 December 2005; Revised 17 July 2006; Accepted 13 February 2007

KEY WORDS: Grid portals; Web services; Service Bus

1. INTRODUCTION

Computational power is constantly opening new opportunities for numerical simulations, and in turn opening opportunities for new science. This computational power is expected to be a low-cost alternative for design and validation, boosting efficiency of manufacturing, and be a reliable source of forecasts. Moreover, vast networks of computing resources continue to grow, forming a computational Grid. These computational resources include hardware, software, sensors, instruments, data, information, and knowledge. This enhanced access is empowering scientists and engineers through faster turnaround, improved accuracy and quality, access to real-time data, and improved capability to share results and transfer technology.

However, as computing systems continue to become more powerful they also become more complex, requiring more expertise to use them. Furthermore, the Grid is inherently heterogeneous making it

*Correspondence to: Tomasz Haupt, Cooperative Computing Group, Center for Advanced Vehicular Systems, Mississippi State University, Box 5405, Mississippi State, MS 39759, U.S.A.
†E-mail: haupt@cavs.msstate.edu

even more difficult to use for the domain experts that need it. It is thus imperative to make these computational resources available to scientists and engineers as easily as accessing other utilities, such as power or phone grids. Grid portals, or other Grid Computing Environments, that manage the complex details of the Grid infrastructure have the potential of enabling scientists and engineers to securely access computational resources anywhere and anytime, through customized, intuitive interfaces. The interfaces are critical. They must precisely convey the specifics of the user applications preserving all the functionality the user needs and use outside the Grid environment. Therefore, the concept of a Vortal is as follows: a vertical Grid Portal tailored for a specific application domain.

There is a vast ongoing research and development to create Grid Computing Environments and numerous Vortals have been deployed. Among the most successful are GEON, the Geosciences Network [1], GriPhyN, the Grid Physics Network [2], and NEESgrid [3], the Network for Earthquake Engineering Simulations. Those are just a few examples, and many others will be certainly reported during this workshop. Much ongoing research concentrates on developing tools that substantially simplify deployment of Vortals—often referred to as 'Grid-of-the-box' solutions—such as GridSphere [4] or CHEF/Sakai [5]. While extremely useful, these efforts do not yet realize the true vision of the Grid, i.e. on-demand access to the resources made available by various, independent providers.

One of the most important concepts of the Grid is the virtualization of resources. The resources are exposed to the users as services with all implementation details hidden behind standardized interfaces. For example, a compute server can be 'hidden' behind Globus GRAM interface and consequently the job submission process looks identical regardless of the platform, operating system or scheduler of the target machine. Similarly, OGSA-DAI interface provides a standardized access to databases and file systems. Unfortunately, these interfaces are too low level to capture the specific requirements of a Vortal. Additional software layers are needed to generate RSL strings or SQL queries to satisfy needs of the Vortal-specific applications. These additional software layers make implementations of different Vortals incompatible, and consequently non-interoperable.

This paper discusses interoperable or cooperative Vortals. Based on the experience of developing a number of Vortals, the authors identify the common functionality of Vortals in Section 2 and express it in terms of Vortal façades in Section 3. In Section 4, these Vortal services are decomposed into basic services that are supported by the Grid infrastructure either directly or using custom adapters to accommodate possible difference in the interfaces defined by different service providers. Finally, in Section 5, several Vortals that were developed by employing the approach outlined in this paper are described.

2. FUNCTIONALITY OF A VORTAL

A Grid Portal provides a user interface to a distributed Grid environment. As such, it necessarily generates latencies that discourage the users. Moreover, the application-specific user interfaces offered by the Grid Portals are often too restrictive in that they do not allow the user to do what needs to be done in the way the user wants it to be done. To become relevant for the end user, a good Vortal must offer functionality that is not available without employing Grid infrastructure, while reducing or hiding overheads. The authors' experience developing application-specific Grid Vortals resulted in the classification of Portal features that makes a Vortal an attractive tool.

DOI: 10.1002/cpe

1. The Vortal frees the user from the installation of the application software—this is particularly important if the installation procedure is complex, and requires administrative privileges as well as the experience of a system administrator.
2. The Vortal provides access to high-performance computing facilities, in particular when cycles are explicitly allocated to the Vortal users.
3. The Vortal negotiates the access privileges and other security issues with the resource or service providers on behalf the user given the credentials provided by the user at the beginning of the Vortal session.
4. Most scientific and engineering applications are driven by application-specific scripts or textual input files that specify the actions to be taken. The user must be given the full control of these files, including capabilities for uploading them directly from the user desktop. In addition, the Vortal may provide support for editing and error checking of the scripts.
5. The Vortal should provide a searchable repository of the input files (scripts). It should make the files in the repository shareable between selected groups of users or made them public at the discretion of their owners. Conversely, the Vortal must provide means for the user to upload scripts that they wish to share with the community and generate metadata enabling meaningful queries.
6. The Vortal must provide a means for the discovery, search, and access data files. This includes access to remote file systems, databases, and on-line repositories.
7. The Vortal must provide a workspace where the user may compose and configure a computational task. The front-end tools must provide support for selecting applications, and if applicable, define a workflow. For each application, the user must be able to upload its components (scripts and data) from either their desktop as well as private, group, or public repositories. Many applications support the modularization of the input scripts. This means that a complete application input may consist of many files, possibly organized as a tree of folders and files.
8. The user may want to maintain more than one application at a time, and therefore the workspace should support organizing the applications into a tree of folders and files.
9. A composed and configured application can be submitted for execution on a target system specified by the user, or selected on behalf of the user. The Vortal must hide all details of the submission process, regardless whether it is a single application or a complex workflow.
10. The user must be given a support for monitoring the progress of the workflow and each individual job. The information about each job must be available also after the job completes and its entry is removed from the back-end job scheduler.
11. At any time the Vortal must provide the access to all input files and parameter values used for job submission so that the job provenance is preserved. Any changes in the workspace made by the user after submitting the job must not modify any information that captures the job provenance.
12. The Vortal must provide access to all output files that the job generated during its execution. Access means a capability to preview the files, download for analysis, or real-time streaming.

3. GRID SERVICES ACCESS PATTERNS EMPLOYED BY A VORTAL

The Vortal is an intermediary between the user and the Grid. The management of the users, users' sessions, and users' credentials is typically provided by the portal container, such as GridSphere.

DOI: 10.1002/cpe

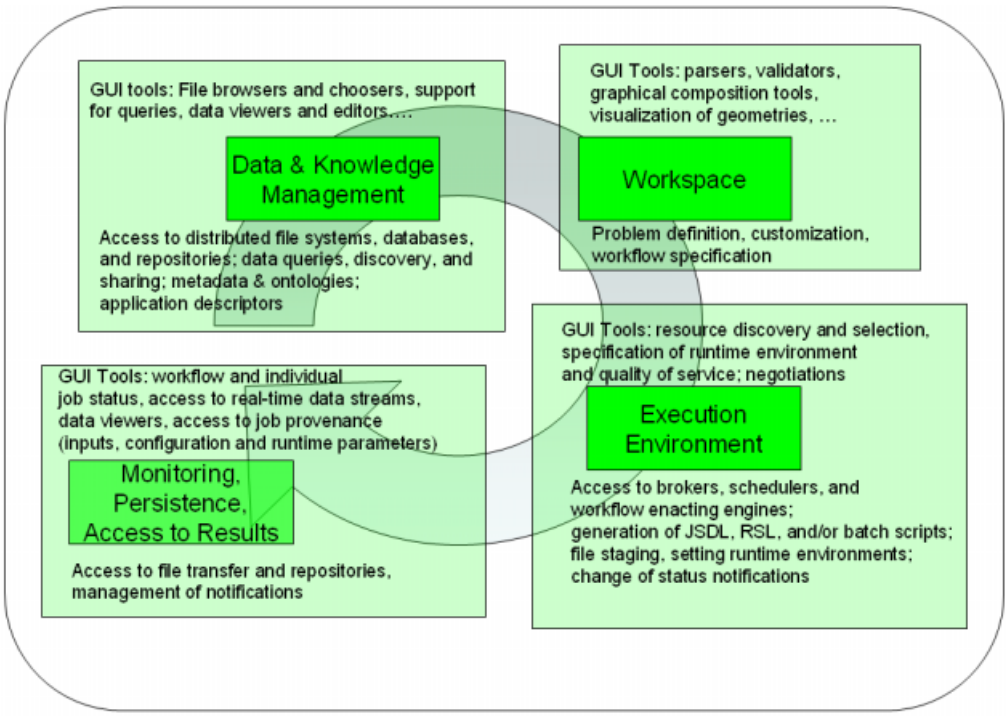


Figure 1. Common features of Vortals.

The business logic of the Vortal—composing, running, and monitoring of computational tasks—is achieved by invoking Grid services to perform actions on the back-end resources. Following the discussion in Section 2, the interaction between the Vortal and Grid services falls into four categories, as depictured in Figure 1. The user needs the access to script and data repositories, to configure the computational task (workspace), to submit the task, and finally to monitor the progress of the task and access the results. These four steps are performed regardless of the application domain and define distinct Grid services access patterns that can be captured and reused between different Vortals. In the remainder of this section we discuss these patterns and introduce façades that implements these patterns.

3.1. Repository façade

Without access to data, computing does not make much sense. Different application domains have different, often very challenging, data access requirements such as processing massive real-time data streams from sensors or performing complex, semantic-driven searches of geographically dispersed databases and repositories containing diverse data in terms of data formats as well as differing

DOI: 10.1002/cpe

conventions, terminologies, and ontological frameworks. Aside providing the access to the 'external data' which is a focus of several projects in progress such as GEON, SCOOP, or GriPhyN, a Vortal must provide a repository for user and community data, including application scripts and job provenance information. Accessing the repository follows a specific pattern of Grid services invocations worth defining as a façade.

The repository is a place where user data are stored. Each data item has associated metadata which provides description of the files and enables searching for repository entries satisfying some selection criteria. The repository provides means for uploading, downloading and manipulating (moving, copying, altering, etc.) data and metadata. The GUI displays the contents of the repository as if it was organized as a tree of directories and files (cf. Figure 2).

Each entry in the repository is labeled by a Uniform Resource Identifier (URI) assigned by the repository. The URI is opaque and it uniquely identifies the metadata record of the entry. The metadata record includes a logical name of the entry—the file name assigned by the user and displayed in the GUI, the entry type (script, input file, output file, folder, etc.), a textual description of the entry provided by the user and additional fields used to query the repository to locate entries of interest. It is the repository implementation responsibility to provide mechanisms for resolving the entry URI into the actual file name and location, i.e. the Uniform Resource Locator (URL). There may be different implementations of this service. The implementation may choose to use the file URL as its URI making the translation of URI to URL trivial, or it may use a Replica Locator service to resolve the URI to the URL of one of many copies of the file, or it may employ other mechanisms.

The repository façade offers a single interface for accessing the repository data separating the front-end developer from the implementation details, such as the interface of the metadata service that may vary depending on what implementation of the service is actually used. For example, given a data item's URI (known to the client by performing a query against the repository or otherwise) the façade, after verifying that the user has sufficient access privileges, streams the contents of the corresponding file back to the client, making all necessary Grid services invocations needed to satisfy the request behind the scene.

3.2. Workspace façade

The central concept of the workspace is an application, which is a container for collecting scripts, data, and other auxiliary files needed to submit a job. The workspace interface provides means for adding and removing files to and from the application and specifying application parameters, for example by editing the scripts (cf. Figure 3). Similarly to the repository, the workspace is organized as a tree with nodes and leaves, each labeled with a unique URI. In contrast to the repository, the leaves of the tree structure are applications rather than individual files. An application may have an internal structure, where the files in the application are organized as a tree of folders and files, mimicking the organization of scripts and data in a file system.

Grouping the files into applications is critically important for a Vortal. It provides an environment where the user composes a simulation, and the Vortal GUI provides means to display the contents of applications, one application at a time. The application defines a 'submission unit', which means that all files comprising the application have to be staged on the target machine, preserving the folder structure, before the application can be started. Finally, the auxiliary files transparently associated with the application 'remember' the run configurations. Therefore, each time when the application is

DOI: 10.1002/cpe

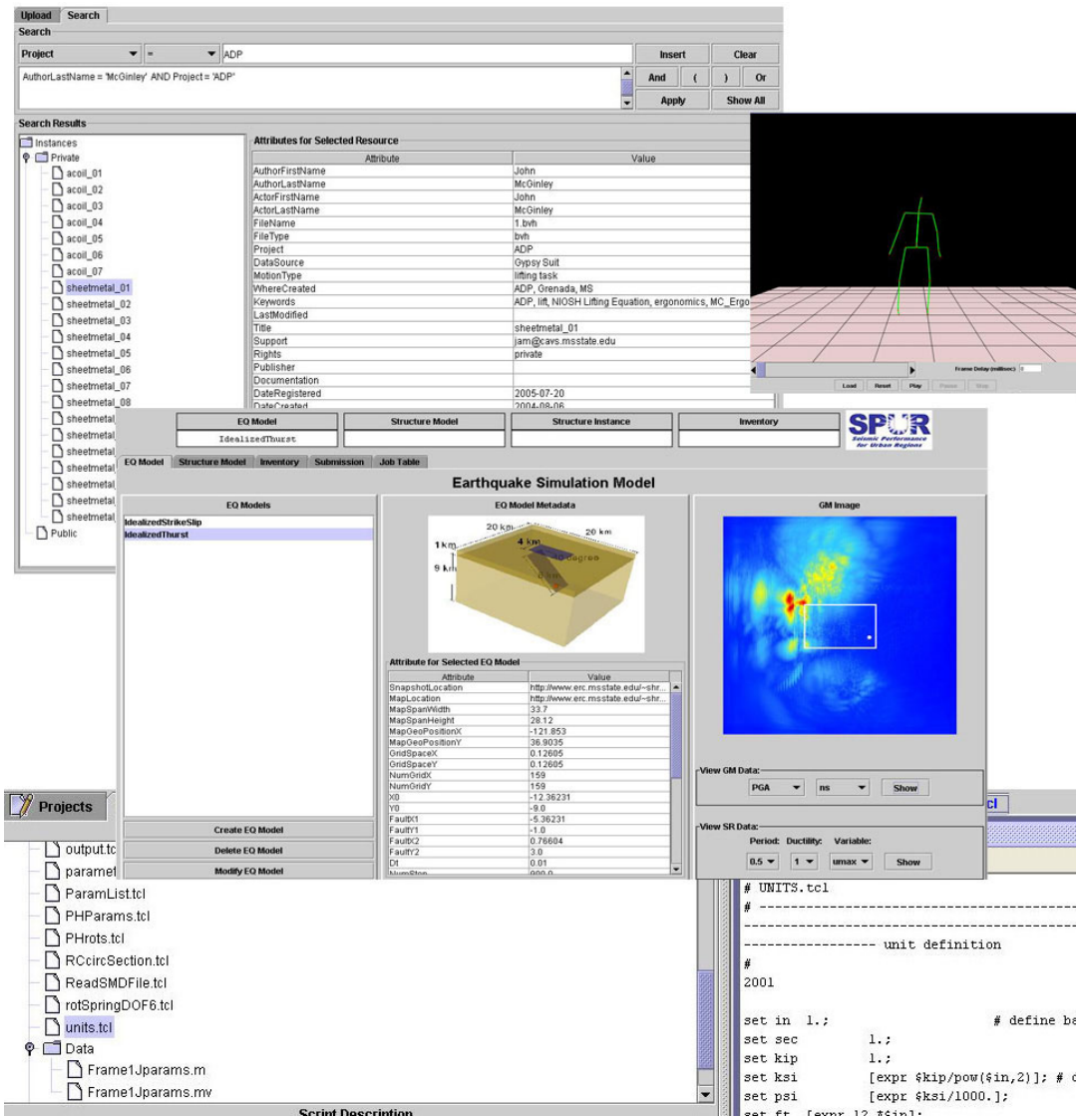


Figure 2. Three examples of the repository façade GUIs: captured motion data repository (top), simulated earthquake ground motion data repository (middle), and repository of OpenSees scripts (bottom). In each case the user selects the data set by examining the corresponding metadata record, by performing keyword-based searches, or by previewing a data set (an animation of the captured motion data, a distribution of a ground motion variable such as Peak Ground Acceleration, or the contents of the script file).

DOI: 10.1002/cpe

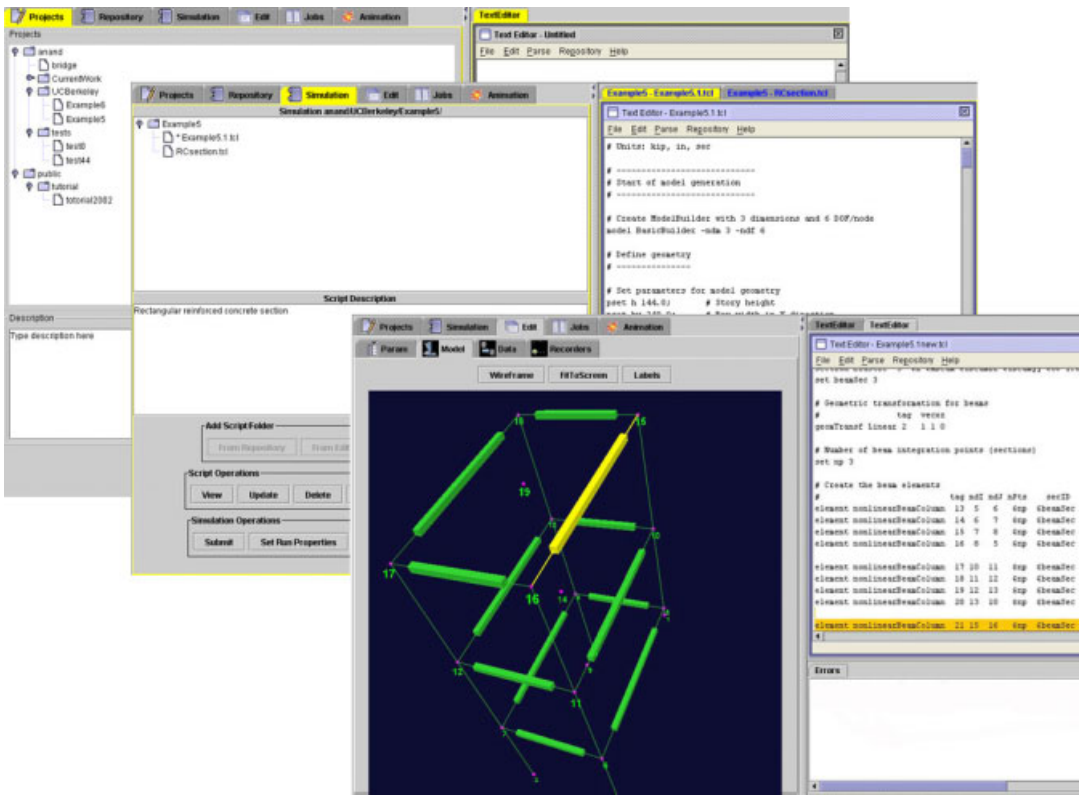


Figure 3. Example of a workspace GUI. The user simulations are organized into a tree (top left). Once a simulation is selected, all scripts comprising the simulations are shown (middle). Each script can be modified either by directly editing the script in the built-in editor or using one of the visual tools provided by the Vortal (bottom right).

to be resubmitted the user needs to introduce only incremental changes instead of going through the configuration process over and over again. Finally, applications prepared for execution in the workspace can be organized into workflows.

The workspace service, similarly to the repository service, aggregates the back-end metadata, data access, replica locator, and authorization services. However, the pattern for how these services are invoked is different and thus justifies a separate façade.

3.3. Job submission façade

The job submission service does whatever it takes to submit a simulation defined in the workspace. Central to the job submission façade is an application descriptor—a metadata record describing the application and providing a complete 'recipe' expressed in the Job Submission Description

DOI: 10.1002/cpe

Language (JSDL) describing what needs to be done in order to submit the application for the execution. A more detailed description of the application descriptor format can be found in [6]. Adding a new application to the Vortal is thus reduced to the generation of a new job descriptor by the application experts. The user may browse existing descriptors to select the right application for the task in hand (through a GUI that displays the application signature and description). For a selected application, the GUI displays its parameters so that the user may adjust their values, and specify the location of input and output files.

During the submission process, the descriptor is parsed to discover the need for the creation of working directories and file transfers (staging the files). Since no scheduler accepts JSDL at this time, the information in the application descriptor is converted to Globus RSL, if Globus GRAM is used, or equivalent. Once the job submission request is created, it is sent to the job submission service, and an entry in the job table of the Job Monitoring Service (described in Section 3.4 below) is created. All notifications from GRAM are forwarded by the façade to the Job Monitoring Service, and, once the job completes, the façade moves the results (output files, standard output, and standard error) to the location specified in the job descriptor. This pattern of invoking Grid services is captured as the Job Submission façade.

3.4. Job monitoring and job table façade

The job table gathers information on all jobs submitted by the user through the Vortal. This includes jobs that never made to the target system (the submission failed) and jobs that completed (successfully or not) and they are no longer accessible through the batch system interface. The user jobs stay in the job table until explicitly deleted by the user. The job table interface provides access to runtime information (date of submission, date of completion, if completed, etc.), job configuration (where it was run, how many processors were used, what working directory was used, etc.), scripts and input data files used for the run (they could have changed in the workspace after the job has been submitted), and to all output files, as shown in Figure 4. Accessing the job status and its results follows yet another specific pattern of accessing metadata, data, and other Grid services that gives the rationale for defining a façade.

4. PROXY SERVICES AND COOPERATIVE VORTALS

Each façade introduced in Section 2 provides a distinct Vortal functionality by custom orchestrating the Grid-level services identified in Section 2 usage patterns. So far, for the Vortals developed by us, five types of Grid services are needed: job submission, file service, replica locator, metadata service, and XML-database service. The standardized interfaces of the façades make the implementation of the Vortal front-end easy. The façades serve as data models, and the front-end implementation just needs to add custom controllers and viewers satisfying the particular application domain needs.

The façades are reusable as long as the interfaces for the Grid-level services are standardized. Even though there is an ongoing GGF effort to achieve standardization, at this time this assumption is not true: there are many alternative implementations of the Grid services. For example, there are many different possible implementations of the metadata service available, from generic DAIS to custom developed services based on J2EE, SRB, or other alternatives. To accommodate the differences in the

DOI: 10.1002/cpe

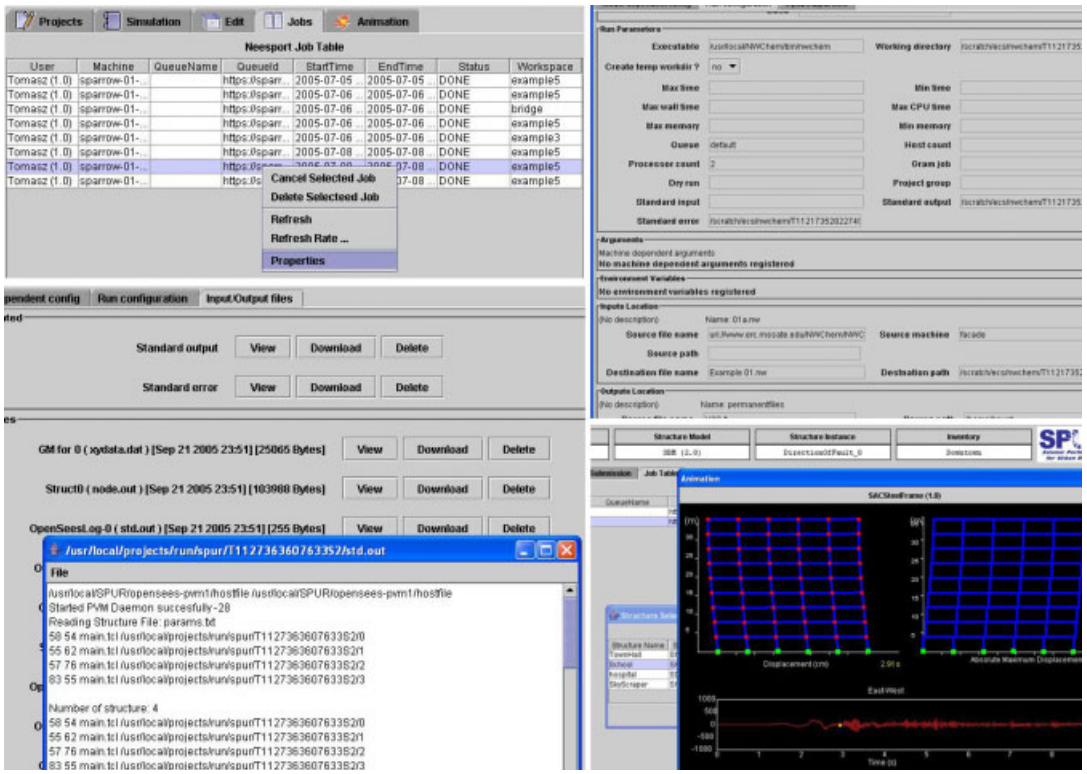


Figure 4. Example of the job table GUI. The GUI shows the status of all user jobs (top left). For each job the user may examine the job properties (top right) and access all its input and output data (bottom left) or preview the results using Vortal tools (bottom right).

interfaces of various implementations of services, the cooperative Vortal infrastructure makes use of proxy services. The proxy services act as clients to the concrete back-end, Grid-level services, and the differences between the interfaces between the proxy and the concrete services are accommodated using adaptors. Each proxy service may have more than one adapter associated with it, and may switch between them in real time depending on the request. For example, the job descriptor passed to the Job Submission service proxy has an attribute describing the access method for the remote system (currently GT2, GT3, GT4, and Kerberos). Depending on the actual value of this attribute, either Kerberos adapter that generates a batch script and submits it for execution using krsh, or Java COG-based adapter that generates the RSL string and invoke GRAM methods is used.

The adapters make it easy to retarget the Vortal to a different middleware or to take the advantage of the new features added to the existing one. New features might include resource brokers, resource schedulers, or other that would increase the quality of services without requiring changes to the proxy services interfaces.

DOI: 10.1002/cpe

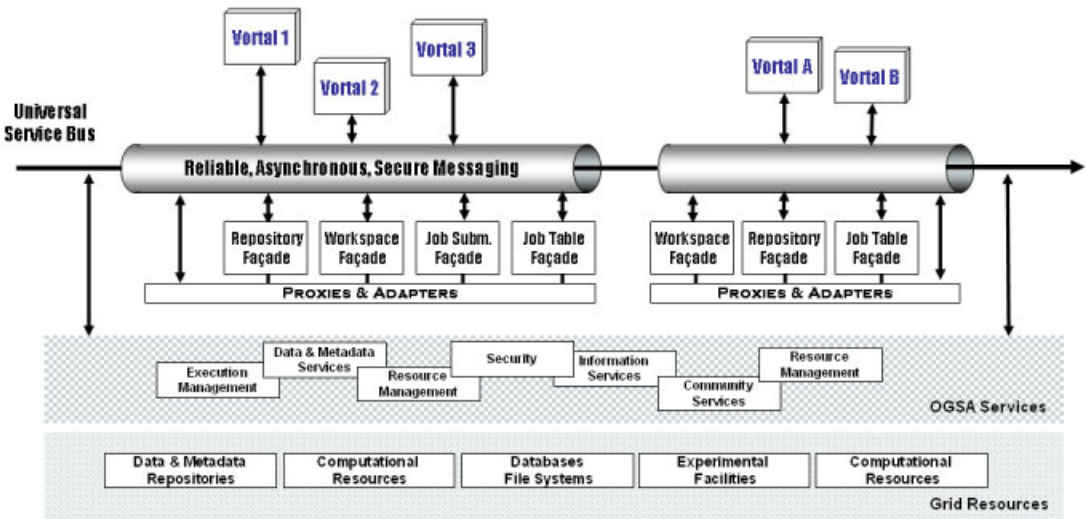


Figure 5. Interoperable Vortals. The Service-Oriented Architecture combined with MOM enables horizontal integration of loosely coupled services, including the Vortal services.

The Vortal services such as façades and proxies are implemented as a loosely coupled Web services and as such they foster cooperation between different Vortals, as shown in Figure 5. The Vortal services interact with the rest of the Grid infrastructure including other local or remote Vortal services through events (i.e. asynchronous messages). The Message-Oriented Middleware (MOM) forms a Service Bus—a term used by the software industry (e.g. Sonic Software) [9]. The Service Bus enables the exchange of messages (either point-to-point or on the subscription basis), and it is responsible for message routing and transformations as well as for providing support for the discovery of services. The Service-Bus-based architecture extends the traditional, centralized 'hub & spoke' architecture which is typical for the 'Grid-out-the-box' solutions, and it promises a more flexible use of the inherently distributed Grid services and resources.

The implementation of the Vortal services as loosely coupled Web services allows advantages to be gained from the use of the Service-Bus-based architecture. In the simplest case, different Vortals may offer different implementations of the façades (invoking proxies with different adaptors), each optimized for a particular type of the resources. A Vortal might thus reuse services offered by other interoperable Vortals on demand.

5. EXAMPLES OF VORTALS

The approach described in this paper has been successfully applied to a number of Vortals developed at Mississippi State University. The list includes NEESport [7], SPURport [8], Captured Motion data repository, and the Vortal for NWChem. NEESport and SPURport are two different portals for

DOI: 10.1002/cpe

the Earthquake Engineering community. NEESport offers the Web interface for remote execution of OpenSees simulations [9]—seismic response of structural and geotechnical systems. Captured Motion data repository is a tool used by ergonomics specialists, and NWChem Vortal provides access to popular computational chemistry simulation software.

Regardless of the application domain, the basic functionality of these Vortals is similar, even though each has a customized front end. Each Vortal uses the repository to share the data and scripts, the workspace to compose the jobs, the job submission service that allows specifying the target system, the batch queue and number of processors to be used. Finally, each Vortal provides support for monitoring the progress of the job and accessing the results. Consequently, each Vortal uses the same middleware. Each front-end invokes methods of the same set of façades that in turn invokes methods of the same proxy Grid services with the potential to invoke the concrete Grid services provided by different third parties. An extreme example is the NEESport. Originally it has been developed for Grid deployed at MSU (Globus Toolkit 2.4 with a dedicated J2EE-based metadata service). To become compatible with the rest of the NEESgrid infrastructure, new adapters were developed to support Globus Toolkit 3.2 and proprietary NEESgrid services, in particular the NEESgrid metadata service (NMDS). With the NEESgrid development transferred for operational use, a new middleware is being developed at SDSC. This change does not affect the architecture of the Vortal, and no modifications in the Vortal front-end are needed.

The Vortal front-end are tailored to match the needs of the end user. The NEESport and NWChem Vortals allow the user to upload arbitrary complex scripts (OpenSees and NWChem, respectively) to the portal and run them using dedicated high-performance computers (at SDSC and MSU, respectively). In addition, the Vortals provide tools that simplify the user task to customize the scripts. SPURport on the other hand provide access to repositories of ground motion data, structures, and inventories of structures allowing the user to investigate the seismic performance of urban regions. Each Vortal provides custom tools to preview the results of the simulations. A practical use of interoperable Vortals, e.g. utilizing the services offered by other Vortals, remains to be demonstrated.

REFERENCES

1. GEON, Geosciences Network. http://portal.geongrid.org [10 April 2007].
2. GriPhyN, Grid Physics Network. http://www.griphyn.org [10 April 2007].
3. Brown GE Jr. NEES, Network for Earthquake Engineering Simulations. http://www.nees.org [10 April 2007].
4. GridSphere. http://www.gridsphere.org [10 April 2007].
5. Sakai Project. http://www.sakaiproject.org [10 April 2007].
6. Haupt T, Pierce M. Distributed object-based Grid computing environments. *Grid Computing: Making the Global Infrastructure a Reality*, Berman F, Fox G, Hey T (eds.). Wiley: New York, 2005.
7. Haupt T, Kalyanasundaram A, Ammari N, Chandra K. NEESport: Grid portal for earthquake engineering community. *Proceedings of the IASTED International Conference on Modeling and Simulation (MS 2005)*, Cancun, Mexico, 18–20 May 2005. ACTA Press: Calgary, AB, 2005.
8. Haupt T, Kalyanasundaram A, Ammari N, Chandra K, Das S, Durvasula S. SPURport: Grid portal for earthquake engineering simulations. *Proceedings of the 2005 International Conference on Computational Science*, Atlanta, GA, 22–25 May 2005. Springer: Berlin, 2005.
9. Chappell D. *Enterprise Service Bus*. O'Reilly Media: Sebastopol, CA, 2004.

DOI: 10.1002/cpe