QuakeSim: Web Services, Portals, and Infrastructure for Geophysics

Marlon E. Pierce, Geoffrey C. Fox, Galip Aydin, and Zhigang Qi
Community Grids Laboratory, Indiana University
501 North Morton Street
Bloomington, IN 47404
812-856-1212
mpierce@cs.indiana.edu

Andrea Donnellan, Jay, Parker, Robert Granat
NASA Jet Propulsion Laboratory
M/S 183-335
4800 Oak Grove Drive
Pasadena, CA 90089-8099

Abstract—We discuss significant recent updates and revisions to the QuakeSim portal and Web services, which provide access to geophysical applications, data sets, and real time sensor data. These new developments include a) significant updates to the Web portal, b) a revision of Web Services to better encapsulate applications, c) additional services for generating Keyhole Markup Language markups of maps, and d) support for real-time Global Positioning Data.

TABLE OF CONTENTS

1. INTRODUCTION

The QuakeSim project, led by the NASA Jet Propulsion Laboratory (JPL), researches the interactions of fault systems that produce earthquakes. Closely related to the geophysical research is the information technology and distributed computing development work, which will be the focus of this paper. In addition to JPL, QuakeSim project includes participants from the University of Southern California, University of California (Davis and Irvine), and Indiana University.

QuakeSim's distributed architecture is based upon a multi-tiered Web portal and Web Services model, separating functionality from the presentation layer. This approach is typical of science gateways [1]. Web Services may be generally categorized as providing access to information (databases) and science applications, although from a data flow model, this distinction is one of implementation only. Earlier work on the QuakeSim architecture is described in [2]. The data management and scientific goals of QuakeSim are described in a companion paper, "QuakeSim: Efficient Modeling of Sensor Web Data in a Web Services Environment."

The subject of this paper is the significant revisions to the portal and service infrastructure that were described in [2]. For more information on the QuakeSim project, the geophysical research work, links to the QuakeSim portal, and links to the project's open source code repository and build system, see [3].

2. QUAKESIM WEB PORTAL

QuakeSim was a pioneer in the use of portlet components for assembling portals out of reusable parts. Our earlier work was based on the Apache Jetspeed 1 portlet container, As described in [2], the Jetspeed portlet development model had several undesirable features (particularly, poor support for externally developed Web applications), which led to our development of custom components, particularly the *WebForm Portlet*. All portlets in the initial version of the portal were extension of this component. This component ultimately had limited scalability and performance because of its use of network I/O for communications between the portlet and its Jetspeed container.

The Java Specification Request (JSR) 168 standardized (and simplified) the portlet model used by Jetspeed and other containers, making these older containers and their associated components obsolete. Although the Jetspeed container served QuakeSim adequately, it required updating to a standard-compliant container to enable interoperability with the rest of the science gateway community. In particular, interoperability with the closely related SCIGN Data Portal [4] project's GPS Explorer portal [5] is desirable so that we can exchange user interfaces.

The QuakeSim portal's current version has been developed using the JSR 168-compliant GridSphere container [6]. In

place of the Web Form Portlet, we have adopted the Apache Portlet Bridges project. All portlets are now developed initially as standalone applications using Java Server Faces (JSF). The JSF portlet bridge can then be used to convert these applications into portlets relatively quickly. We find this model to be preferable to working directly with the portlet's programming interface. A sample screen shot of a portlet is show in Figure 1.

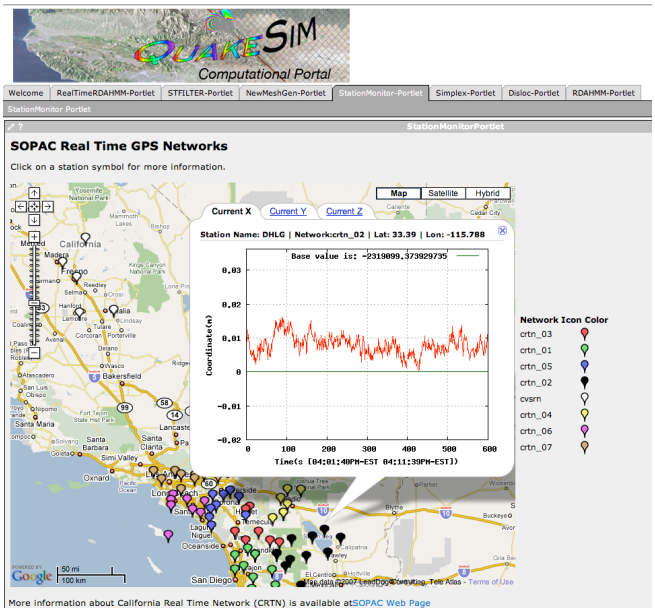

Figure 1: Portlets provide browser-based user interfaces to remote web services. The screen shot shows a Google Map interface to real-time GPS station monitoring services.

The current set of QuakeSim portlets is listed below:

(1) Disloc: a simple Okada [7]-based application for determining surface stress from a given fault model;

(2) Simplex: an inversion of Disloc that uses surface deformation information to determine the best fit fault model;

(3) GeoFEST Suite: a set of codes for modeling faults in realistic materials with finite element meshes;

(4) RDAHMM: a data mining application that automatically classifies modes and patterns in time series data, such as Global Positioning System (GPS) positions;

(5) Analyze_tseri: part of the ST-Filter suite of codes that is used for filtering GPS signals to help identify anomalies; and

(6) Station Monitoring: a portlet that provides access to the latest real-time GPS data from the California Real Time Network [8] (see also Figure 1).

These portlets correspond to horizontal tabs along the top of Figure 1 (beneath the logo), although this particular layout is optional. Future work will include the addition of VirtualCalifornia (VC), which simulates large interacting fault systems. VC, GeoFEST, Disloc, and Simplex all use a variation of the same basic geometric model for representing faults. We are currently working to standardize this object representation within the portal and with the QuakeTables fault database [9] developers.

As we discuss in Section 3, all communications between the portal and remote services use SOAP-based Web services. We develop these using the Java programming language, although the interactions are programming language-independent. Messages enclosed in the SOAP envelope are mapped on both the portal and server side to simple JavaBeans. These data objects are kept purposefully simple: the beans consist of only simple types (strings, integers, double precision numbers). Input, output, and other data files are represented using URLs, and it is up to the implementation to retrieve the file via HTTP GET as a separate step; that is, we do not use SOAP attachments to move files or encode files as XML SOAP body elements.

Most portal applications are designed to support a project and session model: users create new projects or else load older projects through web interfaces. These projects maintain all metadata (parameters used, location of remote input and output files, etc), which the user may edit or delete. The projects are assigned unique IDs, which are used to maintain multiple versions of the same basic project. All beans created by the portal through interactions with remote services are stored in an open source object database, DB4O (see www.db4o.com). This replaces our previous Context Manager service (see [2]) and provides improved performance and scalability. The use of beans also provides a useful integration with built-in Java Server Faces functionality.

3. WEB SERVICES

The portlets listed in Section 2 are clients to one or more supporting, remote Web Services that actually access or generate data. Data access services include fault models from the QuakeTables service [9] and GPS data the GRWS service [10]. We have also, in previous work, built Web Service versions of the Open Geospatial Consortium's Web Feature and Web Map services. Our focus in this section will be on revisions to the execution services that manage invocations of remote geophysical applications.

In the process of upgrading the portal front end, we determined that the application services that provide access to executables were in need of upgrading. Earlier versions of the QuakeSim system were built around the concept of a generic Application Web Service that could wrap any executable. This service was built on generic Web services

for running remote commands and interacting with remote file systems. For related work in this field, see [11] and [12].

We found ultimately that our Application Web Service approach tied services too closely to the portal environment. Web Services ideally should be self-contained, stateless (or nearly so), and completely self-describing. This allows them to bind easily to any client environments such as workflow engines, portals, and desktop user interfaces. It also enables other development groups to use the services with minimal guidance from the developer: the service's WSDL provides sufficient information for invocation.

Our original services did not meet these criteria. Designing for generality, we embedded too many specific steps, such as creating application-specific input files, into the portal instead of the service. Thus the semantics of invoking the Disloc code, for example, were not encapsulated in a single WSDL file but rather in Application Web Service metadata.

Our revised service interfaces have been more an effort of code refactoring and reuse than a complete re-write. Our core job management service, which extends Apache Ant, is still in place but no longer exposed directly as a service. Instead, all new versions of execution services (Disloc, Simplex, etc) extend this parent service (in the object oriented sense) in their implementation.

We have likewise attempted to design the WSDL for these services to capture all information needed to invoke a particular code. A full listing of the service WSDL descriptions is available from the project code repository [3], but we do not include it here since it is generally unreadable by non-experts. Instead, we summarize the Disloc service's XML messages below as a simple example (other applications follow the same general pattern):

(1) Fault Model: this provides a description of major fault geometric and material properties, including latitude and longitude of starting and ending points, length, width, strike, slip, dip, and Lame parameters. As mentioned previously, this model is reusable across many QuakeSim applications and is being adopted by the revised QuakeTables data service.

(2) Input Parameters Model: Disloc calculates surface displacements associated with the fault at either grid or scatter points provided by the user, so we must send this information as well as the fault model parameters. The current version of the service supports the grid input model. The input parameters model contains all information needed to create this part of the input file.

(3) Output Model: the Disloc service creates input, output, and standard output files. The Output Model XML message contains the URLs for these files on the service. In contrast to our earlier execution services, we do not have a specific "file management" service that manages uploads, downloads, and crossloads between services. These are instead done with simple HTTP GET operations.

These messages are XML encodings that correspond to simple JavaBeans in our implementation. They may also be bound to C structs or other simple data structures. These messages are used to communicate the details of the desired invocation and its results over the wire. We use Apache Axis 1.4 for implementing our services.

All QuakeSim execution services implement blocking and non-blocking invocations of the executable, which are inherited from their generic parent class. As the names imply, blocking executions keep the connection between the client and service open until the invocation is complete. This is useful for codes such as Disloc, Simplex, Analyze_tseri and RDAHMM, which take typically only a few seconds (to tens of seconds) to execute.

Longer running codes (GeoFEST and VirtualCalifornia) use non-blocking invocations. This allows the service to return immediately, but the application will continue to run. The user can monitor the application by examining messages from the output files and logs (returned as URLs to the portal). We also provide a call-back mechanism: a simple Event Service stores and retrieves messages. The running application's Ant wrapper monitors the major stages of the application (*started* or *complete,* for example) and posts state changes to the Event Service. Clients, such as the portal, can query the Event Service to determine the current state. This service is admittedly simplistic compared to Web Service standards such as WS-Notification and WS-Eventing, but sophistication has often proven to not be virtue. A more serious criticism would be our lack of use of general-purpose distributed event systems. We do make extensive use of this in our real-time GPS system, described in the next section.

Output files must be usually plotted to be comprehensible. Earlier versions of the portal wrapped Generic Map Tool (GMT) commands with the generic execution service, and we also have implemented the Web Map Service for creating plots. More recently, however, we have found tools such as Google Maps and the associated Keyhole Markup Language (KML) to be a much better mixture of simplicity, interactivity, and power. We have developed KML services that generate grid and arrow plots useful for visualizing Disloc and Simplex output. These services consume Disloc and Simplex output (which are passed to the service via URL), although future versions of these execution services may incorporate these internally, providing the URL of for the generated KML as another part of the execution services' return message.

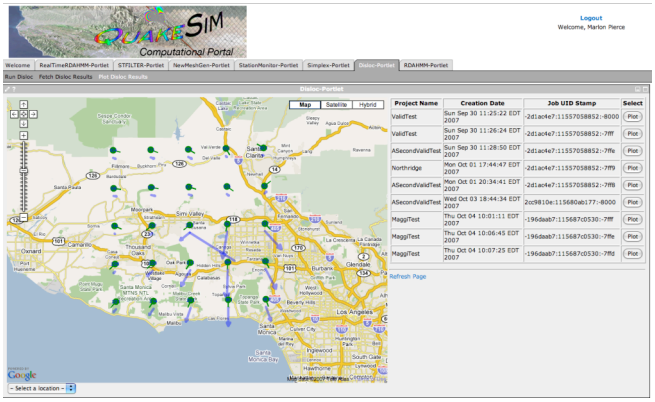

Figure 2: QuakeSim plotting services create KML representations of Disloc output that can be displayed in Google Maps embedded into portlets.

The current portal includes Google Maps portlets that plot the KML files, although these are memory limited. We use the EGeoXML class, a small JavaScript extension to the Google Map API for working with KML, which we have found superior to Google Map's GGeoXML class. We also provide the KML files for direct download, which allows them to be included in Google Earth. These are illustrated in Figures 2 and 3.

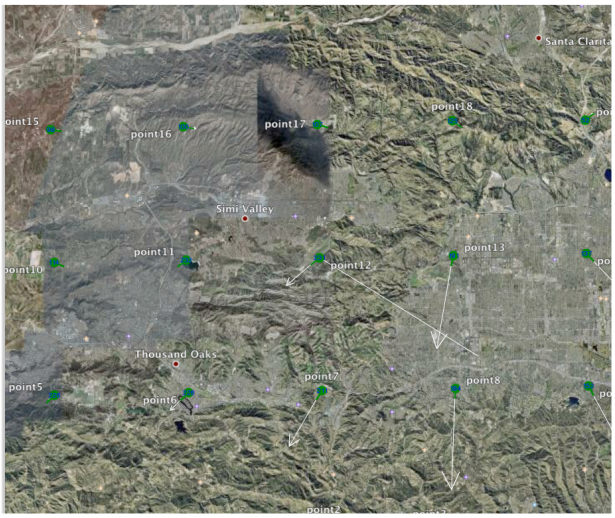

Figure 3: Generated KML files can also be downloaded and plotted with Google Earth outside the portal.

4. Real-Time Sensor Support

In addition to fault models, the other major data types that QuakeSim applications must ingest are seismic events and GPS data. Archival GPS data services such as GRWS provide daily position data, but there is also interest in real-time and custom data products. The California Real Time Network (CRTN) provides online access to GPS position data at 1 Hz, and higher rates will be available in the future.

We have developed a system using topic-based publish/subscribe distributed computing techniques to manage this data. This system is described in greater detail in [13] [14] and summarized here. The basic components are shown in Figure 4. Raw GPS data is passed through a number of filters, which act as subscribers to particular input topics and publishers to output topics. An example topic is a path-like name that is used to indicate the network or station source and the data format. For example, the topic name /SOPAC/GPS/PARKFIELD/ASCII is used to publish or subscribe to the ASCII formatted data from the Parkfield network.

The simplest filters are used to de-multiplex the incoming binary GPS stream. The CRTN consists of eight sub-networks, each with one or more individual GPS stations. The de-multiplexing filters translate this binary formatted signal and extract the individual stations' data. The filters than publishes the data for each GPS station to a new topic. Downstream filters can thus receive individual station data for further processing. See for example Figure 1. We can also provide more sophisticated filters. For example, have developed RDAHMM filters that perform mode classification on the real-time positions.

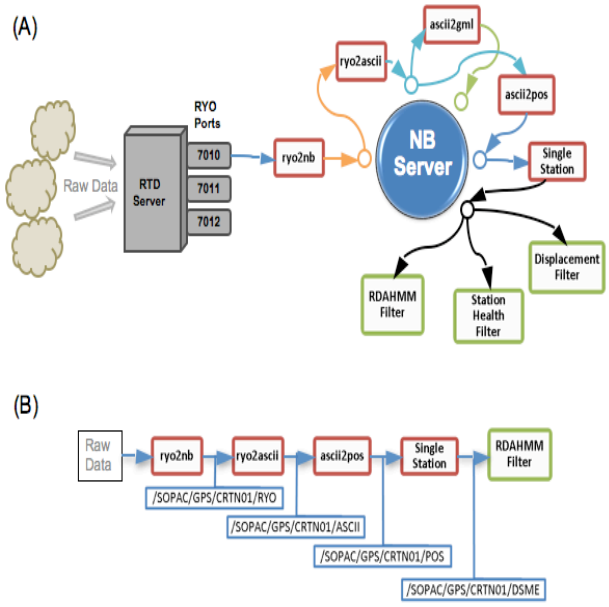

Figure 4: (A) QuakeSim's Sensor Grid system successively filters the incoming, binary GPS data from the California Real Time Network. Arrows indicate network connections. (B) Filter chains are associated with structured topics so that associated with specific data products.

The system is implemented using the NaradaBrokering messaging software (www.naradabrokering.org). Extensive network performance and scaling analyses are available from [14]. To summarize those results, we have demonstrated that we are able to (with simple filters) deliver data with overhead far below the 1 Hz frequency, and our system with a single broker scales to 1000 publishers or subscribers. With networks of brokers the system will scale to larger numbers. Our tests with two coupled brokers acting as a single logical broker showed scaling to 1500 connections.

5. SUMMARY AND FUTURE WORK

In this paper, we have summarized extensive revisions and enhancements to the QuakeSim infrastructure that supersede the earlier work described in [2]. These include complete revisions of the portal and the execution Web services. The portal has been updated to meet current component standards, and the Web Services have been revised to provide simple, self-contained WSDL interfaces. Prominent new capabilities include access to archival and real-time GPS data (in collaboration with the SCIGN team [4]) and KML generating Web services for plotting vector output. We have two categories of future work, summarized below.

Support for Grid Computing: QuakeSim applications typically run on small computers and clusters in which batch scheduling and related issues are not important. This is adequate for codes such as Disloc, RDAHMM, Simplex, and Analyze_tseri. However, only small GeoFEST and VirtualCalifornia simulations can be done this way. Rather than revise our Web Services to address issues such as interactions with scheduling and queuing systems, we are adapting our infrastructure to work with the Globus toolkit. Our Grid Tag Libraries and Beans project extends Java Server Faces to provide components for invoking Grid services. This work is part of our broader Grid portal software efforts, the Open Grid Computing Environments project [15].

QuakeSim Portal and Web 2.0: As described in more detail in [16], we have analyzed the so-called Web 2.0 trend in Web programming and have concluded it is architecturally nearly identical to science gateways such as QuakeSim, although implementations may be very different. For example, most Web 2.0 services use simple, stateless invocations known as Representational State Transfer (REST) instead of WSDL. Message encoding formats used by these services include (simple) XML, RSS, ATOM, and JSON in place of SOAP. These are notably easier to parse and manipulate by JavaScript libraries than the more complicated messages typically used by Web Services. The re-emergence of JavaScript (following the standardization of the XmlHttpRequest class in most major browsers) has driven this simplification (if not standardization) of network messages.

The impact of Web 2.0 on science gateways such as QuakeSim is just beginning to be felt. The QuakeSim portal is primarily based on server-side Web technologies (portlets and Java Server Faces). These are compatible with some prominent Web 2.0 examples (particularly Google Maps, as we have shown in Figures 1 and 2). However, the more serious incompatibility is that Web 2.0 applications use client-side integration. See for example "Start Pages" such as iGoogle and NetVibes, which aggregate content in a manner superficially similar to JSR 168 portlets but which put the user much more in control of his or her content. This similarly simplifies the process of developing new content when compared to the portlet model. On the server-side, QuakeSim services will need to be reconsidered to see if they can be made compatible with Web 2.0's favored message formats.

Most interesting of all is the investigation of the overlap between Web 2.0 and Grids. We will use the QuakeSim project as our laboratory for this research. Important areas to be addressed include scalable event management, single-sign on security, service interoperability, and authorization. All of these are important to Grid computing but are noticeably lacking in Web 2.0.

ACKNOWLEDGEMENTS

We acknowledge the collaborations of the QuakeSim team: John Rundle (UC-Davis), Lisa Grant (UC-Irvine), and Dennis McLeod (USC). We also thank the members of the NASA REASoN project for assistance and collaboration using their GPS services. The QuakeSim project is supported by the NASA Earth Science Technology Office.

REFERENCES

[1] Nancy Wilkins-Diehr, "Special Issue: Science Gateways - Common Community Interfaces to Grid Resources," Concurrency and Computation: Practice and Experience Volume 19, Issue 6, Date: 25 April 2007, Pages: 743-749

[2] Marlon E. Pierce, Geoffrey C. Fox, Mehmet S. Aktas, Galip Aydin, Harshawardhan Gadgil, Zhigang Qi, and Ahmet Sayar, "The Solid Earth Research Virtual Observatory: Web Services for Managing Geophysical Data and Applications," accepted for publication in PAGEOPH Special Issue for 5th ACES International Workshop. Available from http://grids.ucs.indiana.edu/ptliupages/publications/SERVOAces2006.pdf.

[3] QuakeSim Web Site: http://www.quakesim.org; QuakeSim at SourceForge: http://sourceforge.net/projects/crisisgrid.

[4] SCIGN Data Portal: http://reason.scign.org/scignDataPortal/

[5] GPS Explorer: http://geodemo-c.ucsd.edu/gridsphere/gridsphere.

[6] Jason Novotny, Michael Russell, Oliver Wehrens: GridSphere: a portal framework for building collaborations. Concurrency - Practice and Experience 16(5): 503-513 (2004).

[7] Yoshimitsu Okada, "Surface Deformation Due to Shear and Tensile Faults in a Half-Space," 1985, BSSA, vol 75, no. 4, pp 1135-1154.

[8] California Real Time Network: http://sopac.ucsd.edu/projects/realtime/.

[9] Chen, A., Donnellan, A., McLeod, D., Fox, G., Parker, J., Rundle, J., Grant, L., Pierce, M., Gould, M., Chung, S., and Gao, S., Interoperability and Semantics for Heterogeneous Earthquake Science Data, International Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data, Sanibel Island, FL, October 2003.

[10] Geophysical Resource Web Services (GRWS): http://reason.scign.org/scignDataPortal/grwsSummary.jsp.

[11] Gopi Kandaswamy, Dennis Gannon: A Mechanism for Creating Scientific Application Services On-demand from Workflows. ICPP Workshops 2006: 25-32.

[12] Sriram Krishnan, Brent Stearn, Karan Bhatia, Kim Baldridge, Wilfred W. Li, Peter W. Arzberger: Opal: SimpleWeb Services Wrappers for Scientific Applications. ICWS 2006: 823-832.

[13] Galip Aydin, Zhigang Qi, Marlon E. Pierce, Yehuda Bock, and Geoffrey C. Fox Building a Sensor Grid for Real Time Global Positioning System Data Proceedings of Workshop on Principles of Pervasive Information Systems Design Sunday, May 13, 2007 In conjunction with Pervasive 2007, Toronto, Ontario, Canada.

[14] Galip Aydin, Zhigang Qi, Marlon E. Pierce, Geoffrey C. Fox, Yehuda Bock, "Architecture, Performance, and Scalability of a Real-Time Global Positioning System Data Grid," Special issue on Computational Challenges in Geosciences in PEPI Physics of the Earth and Planetary Interiors.

[15] Jay Alameda, Marcus Christie, Geoffrey Fox, Joe Futrelle, Dennis Gannon, Mihael Hategan, Gopi Kandaswamy, Gregor von Laszewski, Mehmet A. Nacar, Marlon Pierce, Eric Roberts, Charles Severance, Mary Thomas, " The Open Grid Computing Environments collaboration: portlets and services for science gateways," Concurrency and Computation: Practice and Experience Volume 19, Issue 6, Date: 25 April 2007, Pages: 921-942.

[16] Marlon E. Pierce, Geoffrey Fox, Huapeng Yuan, and Yu Deng, "Cyberinfrastructure and Web 2.0," Proceedings of HPC2006, July 4 2006 Cetraro Italy. Available from http://grids.ucs.indiana.edu/ptliupages/publications/Web20ChapterFinal.pdf.

BIOGRAPHY

Marlon Pierce is assistant director of the Community Grids Laboratory at Indiana University and leads the portal and services development for the QuakeSim project. His research interests include the application of service-oriented architectures and other distributed systems to problems in applied science. Pierce has a Ph. D. in computational condensed matter physics from Florida State University (1998).