

**College of  
William & Mary**  
Department of Computer Science

WM-CS-2005-11

**Building an Open Grid Service Proxy**

Suzanna Schmeelk

August 12, 2005

## Abstract

Scalable Grid Computing allows distributed, heterogeneous computational resources to be shared and coordinated. Recent standardizations by the Global Grid Forum (GGF), such as the Open Grid Service Architecture (OGSA), introduce standardized, portable, and interoperable computing to the grid environment by extending the web service paradigm. The extension of the web service paradigm to the grid computing environment is known as *Grid Services*. Grid services enable remote clients to access services deployed on a grid in a standardized format. Current grid service clients select static locations for grid service execution. In this paper, I build an open grid service proxy to dynamically and transparently select grid service execution locations on behalf of clients.

## 1 Introduction

Scalable Grid Computing, explained in “The Anatomy of the Grid” [1], permits distributed, heterogeneous computational resources to be shared and coordinated. Grid Computing standardization became apparent by the late 1990s, and the Global Grid Forum (GGF) [2] emerged as a standards body to develop and achieve interoperable grid computing environments [3]. The Open Grid Service Architecture (OGSA) [4] is one fundamental GGF standard for creating interoperable grid computing environments.

Current grid service clients select static locations for grid service execution. In this paper, I build an open grid service proxy to dynamically and transparently select grid service execution locations on behalf of clients. Section II focuses on grid service background and proxy motivation. The grid service proxy architecture is given in Section III. Section IV characterizes the performance of the grid service proxy. Lastly, the conclusions and directions for future work are discussed in Section V.

## 2 Grid Services Background

OGSA, introduced in “Physiology of the Grid [5],” is a standardized service-oriented framework that presents interoperable and portable services within the Grid Computing environment by extending the web service paradigm. A fundamental concept of OGSA is that of Grid Services. Foster and Kesselman define a Grid Service as “a web service that implements standard interfaces, behaviors, and conventions that collectively allow for services that can be transient, and stateful [3].” There are numerous OGSA implementations available, including the Globus Toolkit 3 and, more recently, Globus Toolkit 4.

The GGF and the Organization for the Advancement of Structured Information Standards (OASIS) [6] have developed grid service standards that are base components of OGSA. These standards mirror the changing trends in web services and specify grid service interfaces, behaviors, and conventions. The specifications include the Open Grid Service Infrastructure version 1.0 (OGSI) [7] and the more recent Web Service Reference Framework (WSRF) [8]. OGSI and WSRF specifications are defined using Web Service Definition Language (WSDL) [9, 10].

Areas interested in interoperable grid technology include: Government, Media, Bioinformatics, Electronics, and Life sciences. Grid service examples include: DNA Sequence Analysis, Neuroimaging, Mesh Generation, Drug Discovery, Storm Analysis, and Earthquake Engineering [11, 12].

### 2.1 Standard Web Service Paradigm

Standard Web Services have no single definition. Graham et. al. define a web service as “a platform and implementation independent software component that can be: described using a service description language,

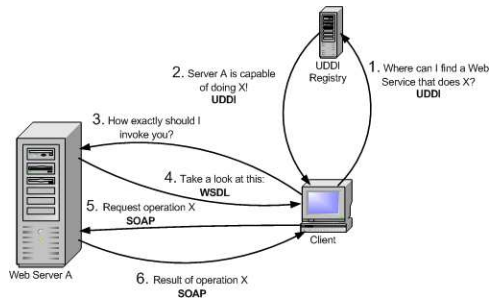


Figure 1: Standard Web Service Process [14]

published to a registry of services, discovered through a standard mechanism (at runtime or design time), invoked through a declared Application Program Interface (API) (usually over a network) and composed with other services [13].” Standard web services include search engines such as Google, product searches such as that used by Amazon and any service offered on a remote machine providing the results to clients over a network.

Standard Web Services are implemented using a variety of communication protocols. Web service communication protocols traditionally use the eXtensible Markup Language (XML) and its extensions: Universal Description, Discovery and Integration (UDDI), Web Service Definition Language (WSDL), and Standard Object Access Protocol (SOAP). The communication protocols serialize information to be conveyed to the remote machine in a standardized format. The communication protocols are then typically bound to an application protocol such as the HyperText Transfer Protocol (HTTP) or the Simple Mail Transfer Protocol (SMTP) and passed through the Open Systems Interconnection (OSI) stacked architecture for execution on the remote machine.

*Figure 1* represents the standard web service process. First, a client sends a UDDI request to a web service directory to locate a server offering a web service. The web service directory sends a UDDI response including the service location. Then, the client sends a request to the web service server requesting information on how to invoke the service. The web service server responds with an invocation guide described using WSDL. The client then uses the WSDL guide to send a SOAP request for an operation. The web service server performs the job and sends a SOAP response with the result to the client.

## 2.2 Open Proxy Security

Security is one important aspect of open proxy technology. The Network System Group from Princeton University published their results of using Open Proxies on PlanetLab’s CoDeeN Content Distribution Network. Malicious users quickly took over their research-based proxies for spamming, content theft, and password cracking. The malicious traffic escalated, requiring the CoDeeN CDN system to be shut down, secured, and relaunched. The security mechanisms that the Network System Group added to the CoDeeN system, including rate limiting and privilege separation, have thus far protected the system from most attacks [15]. The experiences of the Princeton Network System Group research-based open proxies with undesirable traffic, will likely extend to the open grid service proxy. As a result, I am developing rate limiting into the grid service proxy security. Rate limiting will bound malicious use of the open proxy for attacks such as Denial of Service.

## 2.3 Interoperable Grid Computing Environment

The grid computing environment architecture follows a layered paradigm. It has four layers including: grid applications, a middleware, virtualization software, and the physical resources. Similar to the top layer of the OSI paradigm, the grid applications are at the top of the grid layered paradigm. Below the application layer is the middleware layer. The middleware contains the user-focused tools, including the portals to connect to the grid applications. Below the grid middleware is the virtualization software. The virtualization software conceals the complexity of the system activities and permits access to the underlying heterogeneous and dynamic resources found at the bottom layer of the paradigm, the physical resources [11].

There are many grid middlewares available, proprietary and open source, including: Globus, Cactus, Legion, Avaki, EcoGRID, Polder, Condor, MOL, Entropia, and Gridware [14]. I focus on Globus since it is the predominate open source grid middleware that includes an integrated toolkit for grid services.

## 3 Grid Service Proxy Architecture

The goal of the grid service proxy is dynamically and transparently select grid service execution locations on behalf of clients. Currently, the grid service proxy acts as a forwarding agent and is being developed to provide transparent and dynamic grid selection on behalf of client grid service requests. In this section, I present the grid service proxy design rationale.

### 3.1 Grid Services and the Globus Toolkit

The Globus Toolkit (GT) is supported by the Globus Alliance which receives patronage from organizations including: DARPA, DOE, NASA, NSF, IBM, and Microsoft [14]. Globus, developed by Ian Foster and Carl Kesselman, provides standard middleware services including: system monitoring, remote data access, authentication, and communication services.

The Globus Toolkit version 3 (GT3) grid middleware implements the OGSA and OGSF framework to support grid services. Grid services extend the web service paradigm in five ways management of state, identification, sessions, life cycles and a notification mechanism used with grid service data elements [11]. Grid services augment standard web services to provide grid computing power to remote clients. Grid services reduce both end user reliance on local grid computing resources and data flow over the Internet.

### 3.2 Parsing XML

XML standardizes the format of data and is accessible by any application via a parser. The grid service proxy parses XML-based messages and extracts specific elements and their attributes. The selected elements and attributes are processed for transparent grid selection.

Parser selection is the first step towards parsing XML and XML-based languages. The XML parser reads an XML message and divides it into elements, attributes and data for analysis. There are two designated parser types: *validating* and *non-validating*. *Validating* parsers validate XML messages for well-formedness and ensures messages conform to given Document Type Definitions (DTD). *Non-validating* Parsers inspect XML message solely for well-formedness. Companies that develop XML processors include Apache, IBM, Oracle, Sun, and Microsoft. I used a Sun XML processor in the grid service proxy. Currently, the proxy does not use the parser validation feature since I have not yet developed DTDs for XML-based messages parsed at the proxy to conform. This will be a direction for future work.

There are two predominate APIs for parsing XML: Document Object Modeler (DOM) and Simple API for XML (SAX). The APIs are used by applications to obtain elements, attributes and data.

### 3.2.1 SAX API

SAX is an event-based parsing API that uses callbacks for events. SAX callback events are established by the application through handlers prior to parsing. Then as the parser works, it calls back to the application as events occur. All events occur in one-pass of the parser. SAX parser callbacks allow applications to implant actions into the parser flow without modifying the given input source [16].

### 3.2.2 DOM API

The DOM XML parsing API originated in the World Wide Web Consortium (W3C) [17]. The DOM parser represents a document completely in memory and follows a nested tree format [18]. A DOM nested tree is structured by making XML elements and data tree nodes of the document. DOM is customizable, supremely organized and can be traversed multiple times.

### 3.2.3 XML Processing Performance

Maruyama et. al. [16] show that DOM and SAX differ in memory and speed performance. SAX shows higher speed performance than DOM processing since SAX does not maintain a tree representation in memory. However, DOM is easier to customize than SAX. Maruyama et. al. [16] suggest using DOM and SAX for different situations. They recommend using DOM for the following four situations: structurally modifying an XML document, sharing a document in memory with other applications, small XML documents, and parsing after validation. Maruyama et. al. suggest using the SAX API for the following two scenarios: an applications has performance and memory constraints, and an application does not need simultaneous access to multiple elements in the document.

## 3.3 SOAP over HTTP

The Simple Object Access Protocol (SOAP) is a lightweight, XML-based protocol standardized by the W3C. SOAP is the most frequently used communication protocol for web services [16] and can be used as a remote procedure call system. A SOAP message path can involve multiple intermediary and multiple transport protocols before arriving at a given final destination.

SOAP has three main elements: *Envelope*, *Header*, *Body*. The SOAP *Envelope* is the root of the XML-based SOAP message and defines the message framework for content and processing. The *Envelope* has two types of children elements: *Header* and *Body*. The *Header* element is optional and includes application-independent information. The *Body* element is required and includes application-dependent information including encountered error problems. The *Body* element houses the remote procedure call (RPC) request and response data. Multiple *Header* and *Body* entries are allowed within the *Envelope* element.

Two of the most popular transport protocols for SOAP are HTTP and HTTPS [19]. The SOAP message is passed in the entity-body of these transport protocols.

There are multiple SOAP toolkits available in many languages including Java and C++. Java SOAP toolkits examples include: Axis, Sun Java Web Service Developer Pack (JWSDP), SoapFabric, Systinet WASP Server and .NET. SOAP toolkits for C++ include: gSOAP, Axis C++, Rogue Wave LEIF, and Systinet WASP Server. Govindaraju et. al. [19] characterize the performance of five SOAP toolkits for varying

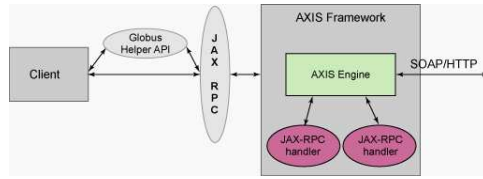


Figure 2: GT3 Grid Service Client Side Framework [20]

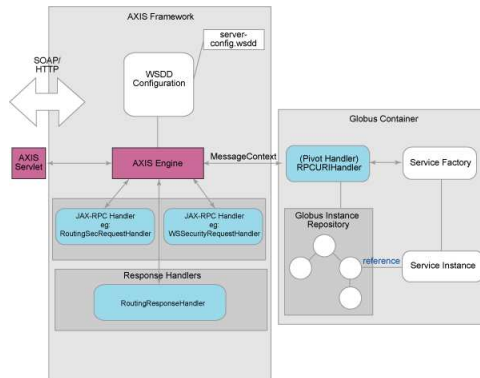


Figure 3: GT3 Grid Service Server Side Framework [20]

workloads used in scientific computing such as varying sizes of double, integer, and string, arrays. They find that SOAP toolkit performance is dependent on application requirements.

### 3.4 GT3 and Axis

GT3 relies on the open source SOAP toolkit Axis (Apache eXtensible Interaction System) to provide the client and the server web service endpoints and SOAP message processing for GT3. *Figure 2* shows the grid service client-side framework and *Figure 3* shows the grid service server-side framework [13, 20].

As can be seen in *Figure 2*, Globus uses SOAP as a remote procedure call system. First, Globus libraries use the Java API for XML-based Remote Procedure Calls (JAX-RPC) [21] programming model to pass a client remote method call to the local Axis Engine. The Axis Engine serializes the call into SOAP, binds the SOAP message to the body of an HTTP POST message, and sends the request to the machine running the deployed grid service.

At the server side, shown in *Figure 3*, there are two main architecture components: the web service engine Axis and the Globus container. The Axis servlet accepts a client's HTTP request. If the message contains a *SOAPAction* header field, the HTTP body is passed to the Axis SOAP engine for deserialization and XML parsing. Then, the engine performs the following: passes the message to the Globus container where the specified service is found and invoked, receives the result, serializes the result in SOAP, and sends the SOAP bound HTTP response to the client [22].

### 3.5 An Open Grid Service Proxy

The design of the proxy includes five main components: transparent grid selection, XML processor configuration, security, forwarding agent and multiple transport protocol compatibilities. The initial grid service implementation acts as a forwarding agent in a static environment. It accepts requests for grid services from

clients and forwards the request to a machine running the deployed grid service. At the deployed grid service machine, the request is accepted, fulfilled and a response returned to the client via the grid service proxy. The exchanged messages are bound to HTTP and sent between web service engine endpoints running a grid service client and a deployed grid service. Currently, I am developing the grid service proxy to implement the four features described in the below subsections.

### **3.5.1 Transparent Grid Selection**

Transparent Grid Selection is the predominate feature of the grid service proxy. The current grid service proxy is configured during initialization to forward grid service requests to a machine running the client grid services. However, proxy configuration during initialization is limited to a static environment. I am currently developing the proxy to function in a dynamic environment by implementing grid service discovery and selection feature. I am developing transparent grid selection based on trusted grid service location, grid load, and network conditions.

### **3.5.2 Security Features**

Two important security features being incorporated into the grid service proxy are rate limiting and compatibility with the HyperText Transfer Protocol Secure (HTTPS).

HTTPS secures HTTP application data by using the Transport Layer Security (TLS) of the OSI architecture. The TLS runs on top of the Transport layer, which includes TCP and UDP, and below the Application layer of the OSI architecture. HTTPS does not modify application data. Instead, HTTPS adds privacy, integrity and authentication to application data.

The grid service proxy will also incorporate rate limiting. I determined rate limiting to be an important feature for open proxy security due to the results of the research-based open proxies of Pai et. al. [15] discussed in Section II.B. The integration of rate limiting into the proxy will bound both incoming requests from overactive client-based machines and outgoing requests to highly utilized grid point of access machines.

### **3.5.3 XML Processor Configuration**

The XML processor is currently fixed in the grid service proxy to Sun's Java DOM parser. I selected the DOM parser since initial XML messages passed through the proxy were small. However, as the grid service proxy capabilities grow so will the need for users to select the XML processor. I am developing the grid service proxy to use a user-selected XML vendor parser class during proxy initialization.

### **3.5.4 Transport Protocol Compatibility**

Web services can be bound to a variety of transport protocols including: HTTP 1.0, HTTP 1.1, HTTPS, SMTP. Currently, the grid service proxy is compatible with HTTP 1.0. I am developing the proxy to be compatible with HTTP 1.1 and HTTPS.

## **4 Proxy Performance**

The following experiments were performed using the Globus Toolkit version 3 (GT3). GT3 was installed on a hyper-threaded 2.8Ghz P4, 800MHz memory, Gb CSA LAN machine. I deployed the grid service and

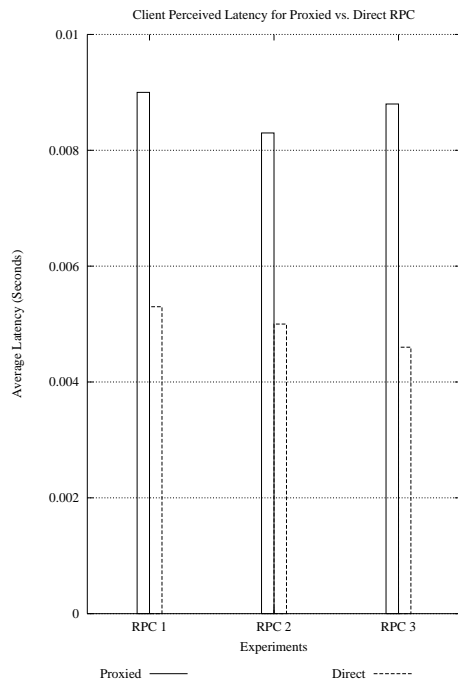


Figure 4: Average Client Perceived Latency

ran the grid service client on the GT3 installed machine since both service and client relied on GT3 libraries for compilation and execution.

I implemented the grid service *Math Service* described in Borja Sotomayor’s GT3 Programmers Tutorial [23]. The actual *Math Service* service is implemented using Java 1.4.2 since GT3 is dependent on this Java version for grid services. The client for the deployed *Math Service*, however, is implemented using Java 1.5.0. I ran three experiments. The first, second and third RPC experiments were grid service client calls to three remote methods of *Math Service*: *add*, *getValue*, and *subtract* methods, respectively.

I define *latency* as the time from when a grid service client initiates a call to a remote method until the time at which the grid service client’s remote method call completes. I compared the latency for both direct and proxied request and response. The Apache Axis web service engine was configured to use the transport protocol HTTP 1.0.

*Figure 4* shows the average client perceived latency for fifty runs of each experiment. As expected, the proxy slightly increases latency. The increased overhead perceived by the client is small compared with the features being developing into the proxy to optimize grid service performance based on grid selection. *Figure 5* shows the standard deviation for the fifty runs of each experiment and mirrors the consistency of the collected latency data.

## 5 Conclusion and Future Work

Current grid services follow a static model with grid service clients statically selecting a location for grid service execution prior to execution. In this paper I implemented a grid service proxy. The proxy mediates grid service endpoint communications. The grid service proxy acts as a forwarding agent and is being developed to provide transparent and dynamic grid selection on behalf of client grid service requests. The

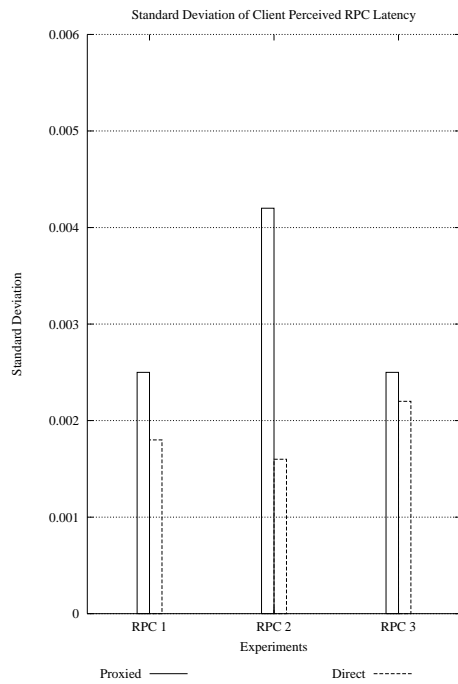


Figure 5: Standard Deviation of fifty Client Perceived Latency Simulations

proxy will direct client grid service requests according to trusted grid service location, grid load, and network conditions. I foresee the grid service proxy as an invaluable resource for clients to adapt to a dynamic environment. Transparent grid selection provided by a proxy offloads client burden and promotes grid service development. Other developing proxy functionality includes rate limiting, XML parser selection, and compatibility with multiple transport protocols.

I have shown that the base grid service proxy only slightly increases service overhead. The increased service overhead does not outweigh the dynamic client benefits being developed into the proxy. It is a natural extension and enhancement to the OGSA paradigm.

## References

- [1] I. Foster, C. Kesselman, and S. Tuecke, “The anatomy of the Grid: Enabling scalable virtual organizations,” *Lecture Notes in Computer Science*, vol. 2150, pp. 1–25, 2001. [Online]. Available: [citeseer.ist.psu.edu/foster01anatomy.html](http://citeseer.ist.psu.edu/foster01anatomy.html)
- [2] (2004) Global grid forum. [Online]. Available: <http://www.gridforum.org/>
- [3] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 2004.
- [4] *Open Grid Service Architecture*, <http://forge.gridforum.org/projects/ogsa-wg>, Global Grid Forum Std. 1.0, 2004.
- [5] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, “The physiology of the grid: An open grid services architecture for distributed systems integration,” 2002. [Online]. Available: [citeseer.ist.psu.edu/foster02physiology.html](http://citeseer.ist.psu.edu/foster02physiology.html)
- [6] (2005) Oasis. [Online]. Available: <http://www.oasis-open.org>
- [7] J. Joseph, “A developer’s overview of ogsi and ogsi-based grid computing,” 2003. [Online]. Available: <http://www-128.ibm.com/developerworks/library/gr-ogsi/index.html>
- [8] (2005) The globus alliance: The ws-resource framework. [Online]. Available: <http://www.globus.org/wsrf>
- [9] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, “W3c: Web services description language (wsdl) 1.1,” <http://www.w3.org/TR/wsdl>, 2001.
- [10] (2005) Ogsi.net. [Online]. Available: <http://www.cs.virginia.edu/~humphrey/GCG/ogsi.net.html>
- [11] F. Berman, A. Hey, and G. Fox, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley and Sons Ltd., 2003.
- [12] L. Ferreira, B. Jacob, S. Slevin, S. Sundararajan, M. Brown, J. Lepsant, and J. Bank, “Globus toolkit 3.0 quick start,” Redpaper, 2004.
- [13] S. Graham, D. Davis, S. Simeonov, G. Daniels, P. Brittenham, Y. Nakamura, P. Fremantle, D. Konig, and C. Zentner, *Building Web Services with Java*. Indianapolis, Indiana: Sams Publishing, 2005, ch. 5.
- [14] (2004) The globus alliance. [Online]. Available: <http://www.globus.org>
- [15] V. Pai, L. Wang, K. Park, R. Pang, and L. Peterson, “The dark side of the web: An open proxy’s view,” *ACM SIGCOMM Computer Communications Review*, vol. 34, no. 1, pp. 57–62, 2004.
- [16] H. Maruyama, K. Tamura, N. Uramoto, M. Murata, and A. Clark, *XML and Java: Developing Web Applications*. Boston, MA: Addison-Wesley, 2002, ch. 12.
- [17] w3org, “World wide web consortium (w3c),” <http://www.w3.org/>, 2005.
- [18] B. McLaughlin, *Java and XML*. Sebastopol, CA: O’Reilly and Associates, Inc., 2001.

- [19] M. Govindaraju, Slominski, Chiu, Engelen, and Lewis, "Toward characterizing the performance of soap toolkits," *Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pp. 365–372, 2004.
- [20] J. Joseph, "Globus toolkit 3.0 and the ogis architecture," 2003. [Online]. Available: <http://www-106.ibm.com/developerworks/grid/library/gr-gt3/>
- [21] Sun, "Java api for xml-based rpc (jax-rpc) overview," <http://java.sun.com/xml/jaxrpc/overview.html>, 2005.
- [22] G. Glass, "The web services (r)evolution: Part 3," 2003. [Online]. Available: <http://www-106.ibm.com/developerworks/webservices/library/ws-peer3>
- [23] B. Sotomayor. (2004) Globus toolkit version 3 tutorial. <http://gdp.globus.org>.