Enabling Access to WSRF from Mobile Devices

Jan Christian Mangs[1], Shihong Huang[2], Junwei Cao[3]

Department of Computer Science, Florida Atlantic University

777 Glades Road, Boca Raton, FL, USA

[1]jmangs@fau.edu

[2]shihong@cse.fau.edu

[3]*Research Institute of Information Technology, Tsinghua University*

Information Science & Technology Building 100084, Beijing, China

jcao@mail.tsinghua.edu.cn

Abstract—**The increasing availability of Web services and grid computing has made easier the access and reuse of different kind of services. Web services provide network accessible interfaces to application functionality, while grid computing enables the efficient distribution of computing resources and power. Mobile devices as a platform of computing have become a ubiquitous, inexpensive, and powerful computing resource. Currently, there are a few complete implementations that leverage mobile devices as a member of a grid environment. This paper presents a framework that enables the use of mobile devices to access stateful Web services on a Globus-based grid. To illustrate the presented framework, a user-friendly application has been created that uses the framework libraries to demonstrate the various functionalities that are accessible from any Nokia S60 phone.**

Keywords: grid computing, mobile devices, Web services, software reuse

I. Introduction

Web services are network-accessible interfaces to application functionality. Although there have been some discussions of the numerous technical challenges to fully utilize their potentials [11], web services have become part of mainstream computing.

A web service is defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network" [17]. An individual who wishes to utilize a web service only needs to know what a service does and not how it is implemented on the server side. Specifically, that individual needs to know where they can find a specific service, what input, if any, is required by an invocation to a web service and what information is returned by that web service. Web services describe themselves using the Web Service Description Language (WSDL)[16] and developers utilize this description to automatically generate client stubs that use Simple Object Access Protocol [14] invocations to access services.

This material is based upon work supported by the National Science Foundation under Grant No. OISE-0730065.

The convergence of grid computing and web services has lead to the concept of stateful web services. While first generation grids were capable of massive computational power, there were drawbacks due to the overhead required in managing and configuring available computing resources and the relatively lack of reusable end-user applications [4]. An application was tightly-coupled to the platform it was developed for and reusing previous work required additional effort. The latest trend has involved movement towards applying web services in grid computing and has led to development of "stateful" web services that are geared for use in grid computing environments.

Because of the lack of support for maintaining state information across web service invocations, the current standard for web services was insufficient for the grid computing environment. In order to effectively support the collaboration required by grid computing, the Open Grid Services Infrastructure (OGSI) standard was proposed [1]. OGSI was originally intended to provide the infrastructure to adding stateful resources to web services. However, the advent of new Web Service standards such as WSDL 2.0 [15] and WS-Addressing [6] necessitated the creation of Web Services Resource Framework (WSRF). The WSRF standard is relatively similar to OGSI with the exception of syntax and naming convention changes; it support is more robust for other Web Service standards.

A. Utilizing Mobile Devices in Grid Computing

In regards to grid computing, the mobile phones are in a position similar to personal computers: it is relatively inexpensive, available to many, and continuing to grow in computing power. Although still lagging behind traditional computers in technical aspects, the potential of mobile devices cannot be easily dismissed. The current generation of cell phone technology, for example, is suited for tasks which involve remote management of tasks running in a grid environment. Because of their inherent mobile nature, these devices are able to stay with a user regardless of their location. A person is not limited to sitting by a desktop machine. Although they aren't well suited yet for offering computational and storage, the processor speed and storage capacity of mobiles continues to grow rapidly with the advent of low-power processors and flash memory [12]. In the future, it may even be possible that mobile devices are powerful enough to perform some level of computation for grids through CPU cycle scavenging or voluntary participation in grids.

Because of the lack of notable solutions to the problem mentioned, this paper proposes to create a new and complete solution that will finally make it possible to readily utilize WSRF on a mobile device. It presents a framework created to address the issues that occur when trying to utilize WSRF from a mobile device. The purpose of this paper is threefold:

(1) To solve the complications when communicating to a stateful web service from a mobile device;

(2) To allow more complicated use of stateless web services simultaneously, and

(3) To lay the foundation for incorporating mobile devices into the grid environment.

II. Implementation Infrastructure

The objective of this framework is to enable mobile devices to connect to WSRF-enabled web services and lay the foundation for future work on incorporating mobiles into the grid computing environment. This section describes the details of the server-side component that implements WSRF and the client-side software platform on which the framework is built upon, the reasons that why each was chosen, and the difficulties in establishing successful communication.

A. Grid Component - Globus Toolkit

The open source Globus Toolkit [4] is a toolkit used for building distributed computing grids. It allows organizations to bring numerous independent computers together into a single virtual "supercomputer" otherwise known as a computing grid. The Globus Toolkit was created by the Globus Alliance some of whose members include the University of Chicago, the U.S. Argonne National Laboratory, and the University of South Carolina. Globus has widespread industry adoption with many companies such as IBM, Sun, Oracle, and Hewlet-Packard have pursued Globus-based Grid strategies [5].

One of the main reasons the Globus Toolkit was chosen as the platform to develop our framework in conjunction with was its implementation of WSRF [8]. As mentioned in the introduction, WSRF is an evolution of OGSI which is another standard which Globus also implemented. In Globus, stateful web services are used to expose grid computing services to the internet in a platform-independent manner. The services provided by Globus are well-suited for testing the framework proposed in this paper. Rather than relying on mock-up web services, the framework has been built in conjunction with testing on robust web services in Globus such as job submission and management and file transfers. The case study in Section 4 demonstrates the use of these services in an application built upon the framework.

B. Software Platform - Java ME

The target platform on which our framework would be developed also plays a crucial role. Because of the wide variety of cell phones, handsets, and PDAs available, we decided to develop our framework for the Java ME programming language. The reason for this decision is fairly simple. Because Java ME is not limited to a single manufacturers' platform, the framework which is built upon it will be applicable to as many phones as possible. For example, if we had to chosen to develop in .NET Mobile, it would only be applicable to devices which can run Windows Mobile.

The JSR-172 Web Services API [9][10] is a subset of the Java API for XML-based Remote Procedure Calls (JAX-RPC) [7]. JAX-RPC provides asynchronous RPC to web services using Extensible Markup Language (XML) and works over a wide variety of protocols including HTTP. In both JSR-172 and JAX-RPC, client-side stubs are used to hide the implementation details of the web service being called. The client-side stub includes the operation name, the input and output types, and miscellaneous parameters. In short, JSR-172 provides basic support for web service invocations and not much else. The lack of full support for complex data types results in an inability to communicate with any web services inside Globus that require more complicated input.

III. The Framework

Because of the limitations mentioned in the section above, the use of the standard JSR-172 API was not possible when communicating to Globus. The framework described in this section was created to fix the problem of communicating to stateful web services by implementing standards required by WSRF and allowing the use of complex data types.

A. Architecture of the Framework

The basic approach to this framework was to emulate the approach implemented in the JSR-172 API. The client creates stubs manually or automatically from a WSDL file of a WSRF-based service. The client stubs invoke the framework run-time. The framework's run-time consists of a connection manager which handles setting up the connection to a web service and the processing of the request and response commands, and a SOAP encoder and decoder. Since the SOAP encoder and decoder used by JSR-172 were not available publicly, the framework required the use of a custom encoder and decoder. This allowed us to add additional property fields to the stub such as those defined by WSRF or any of its related standards. It also allowed us to add an object-oriented method of constructing the numerous different SOAP messages required to communicate successfully. It allows a developer utilizing the framework to perform custom serialization of complex data types. This method was derived from Apache Axis as it is used in Globus [3].

B. Description of the Framework

In JSR-172, when the developer utilizes a method defined by the portType that requires complex input that is not a simple data type, there was no clear way to submit the information required. There is no method provided for custom serialization or custom message properties in the SOAP message. Rather than only accepting simple data types, such as in JSR-172, the framework's encoder was modified to accept classes that implement an interface called `GlobusObject`. This simple interface consisted of a single method called `GenerateMessage()`, which when called, would create an SOAP message representation of the object. This allows the developer to define the specific construction of the object as it would appear in the SOAP message. This method was applied to most of the default web services in Globus that required complex input. It has an added benefit when dealing with the more complicated XSD definitions that could have inheritance and contain arrays of objects; the developer simply has to create a Java representation of arrays and inheritance instead of trying to compose an enormous SOAP message in one single Java class.

C. Features of the Framework

The framework provides full support for the `WS-Addressing`, `WS-ResourceProperties`, and `WS-ResourceLifetime` standards as well as partial support for WS-Trust, WS-Security, and WS-BaseFaults. Due to the complexity in running a mobile phone as a server, there is no current support for the `WS-Notification` standard as it is implemented in the Globus Toolkit [13]. There is also limited inherent support for WS-BaseFault and its sub-faults; the framework parses errors returned by the server and throws an exception with the server-side error and its details.

The framework's SOAP encoder is able to encode simple types, arrays of simple types, and objects that implement `GlobusObject`. When creating a stub, the user is able to specify which WS-Addressing tags to utilize in the SOAP envelope. The decoder is able to parse simple types, arrays of simple types, complex types, and error messages. Also because the framework is based partially upon JSR-172, there is inherent support for stateless web service invocations.

IV. CASE STUDY

In order to demonstrate the functionality of the framework and provide a user-friendly manner in which to utilize its functionalities, a sample application was created and built upon the framework libraries previously created. It demonstrates the use of the framework while creating a user-friendly interface to utilize its functionality.

A. Emulation Platform: Nokia S60

After some research, it was decided to restrict the case study implementation to Nokia's S60 platform. This allowed the case study to focus on one phone platform for deployment. The latest S60 SDK provided by Nokia comes with an emulator and integrates into Eclipse through a plug-in called EclipseME [2]. The emulator's debugger integrates into Eclipse and allows a developer to perform real-time debugging of the actual code as if it were running in a physical device. The emulator also provides numerous diagnostics such as CPU and memory usage statistics as well as tracking messages sent over HTTP/HTTPS.

B. Implementation Overview

The client application for the phone is able to submit jobs, perform third party files transfers, view/browse a shared server space, and obtain delegated credentials. The user interface is geared for use on mobile phones and to simplify functionality as much as possible. The application allows the user to connect to one or more servers at one time and keeps track of currently open connections. To keep navigation simple, the user can only see only one 'view' of the server connection at a time but is free to independently switch between at will. In order to connect to a server, the phone requires a valid user certificate from a certificate authority. This simply required conversion of a Globus user certificate to PCKS12 and, once imported into the phone, works automatically with the case study application.

In this case study, the application was built using Nokia's S60 3rd Edition FP 2 SDK and was developed in Eclipse using the EclipseME 1.7.9 plug-in. Globus Toolkit 4.0.7 (GT4) was utilized to host the stateful web services and simpleCA was used as the Globus certificate authority. Currently, the application includes support for the several services in Globus including: Web Services Grid Resource Allocation Management (WS GRAM), Reliable File Transfer (RFT), and the Delegation Service.

C. Emulation Results

This section illustrates an overview of the application developed through the course of the case study. The main features enabled in this application are reviewed including file browsing, remote file transfers, and job submission & management.

The main menu screen presents the user with several options. The user can choose to view current jobs submitted from the phone, view the shared space in the applications file browser, enable credential delegation for a specific server, open a new connection, or switch between connections. The main function of the main menu is to provide access to the other key features of the application. Users are able to save connection info to different servers and connect to them at will; the application manages each connection independently and allows the user to switch between servers at freely. The "View Folders" command shows the root directory of the shared space on the current view and provides a good majority of the functionality enabled by the framework. The "View Jobs" command is only used when jobs are submitted from file browser; it provides a list of all job submissions and allows the user to manage them. The "Delegate" command is simply used to activate credential delegation with the current server. The final two commands allow for switching between open server connections and creating and opening new server connections.

The file-browsing interface which is accessed from "View Folders" emulates a Windows-type file browsing user interface. The browser is the main point of interaction with the Globus server and its shared space. The shared space is a directory defined by a stateful web service called `DirectoryService` which lists the contents of this directory and provides a method for getting the contents of the files listed. The user can navigate up and down the shared space's folder hierarchy similarly as in desktop operating systems. When selecting a specific file, several options become available. The user can choose to view the contents of the file, submit the file as a RSL description to WS GRAM, mark it as a source or destination, or mark the file for a deletion request. For mobile job submission, the application allows the mobile device to get jobs through the "Submit Job". In a separate screen which can be accessed from the main menu through "View Jobs", users are able manage operations perform operations such as "Destroy" to remove a finished job and "Get Status" to query a job resource property for its state. In the future, other more advanced management options will be integrated into the case study application.

V. Conclusions and Future Work

This paper presented a framework for utilizing stateful web services from a mobile device. The framework is built upon the previous work done in JSR-172 and expands support for custom serialization and additional message properties. To demonstrate the functionality of this framework, the paper showcases an example application built on top of this framework which allows users to browse and view files on a Globus server, submit jobs, and transfer files between multiple servers.

The main limitations in our framework are those created by the restrictions imposed by Java ME. Many features that could have been salvaged from the APIs already created for Java SE could not be directly reused to the limitations in the Java ME environment. Also, it should be noted that because Java ME works in a limited environment, performance will not always be ideal; for example, a loss of connectivity will prevent a web service-based application from working properly. Another limitation involves utilizing the framework on non-Nokia devices. In order to utilize the framework on other phones, the manufacturer must support the same features in their MIDP implementation for Java ME. Because the Globus Toolkit utilizes Transport Layer Security (TLS) [18], the manufacturer must provide full support for TLS. For example, a reference implementation such as Sun's Java ME SDK does not support client certificates under TLS and such does not work with our framework.

In the future, the framework will address some of the challenges presented in this paper such as developing a method to implement WS-Notification. In addition work will be done in order to deploy the application to another manufacturer's phone platform to test compatibility and improve its user interface as much as possible. Other possible areas of work involve simplifying the framework itself and creating a tool to automatically generate Java ME client stubs from Globus WSDL files.

References

[1] Cjazkowski, K., et. al., "From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution" Online at: http://www.globus.org/wsrf/specs/ogsi_to_wsrf_1.0.pdf

[2] Craig Setera, "Eclipse ME: J2ME Development Using Eclipse" Online at: http://eclipseme.org/

[3] Feller M., Foster I., and Martin S., "GT4 GRAM: A Functionality and Performance Study", TeraGrid Conference 2007, Madison, WI, June 2007.

[4] Foster I., "Globus Toolkit Version 4: Software for Service-Oriented Systems", *Proceedings of IFIP International Conference on Network and Parallel Computing*, Beijing, China, Springer-Verlag LNCS 3779, December 2005, pp 2-13.

[5] Globus Alliance "About the Globus Toolkit", Online at: http://www.globus.org/toolkit/about.html"

[6] IBM, "Web Services Addressing" Online at: http://www.ibm.com/developerworks/library/specification/ws-add/

[7] Nokia "Java ME Developer's Library" Online at: http://www.forum.nokia.com/document/Java_Developers_Library_v2/

[8] Pu L., Lewis, M.J., "Uniform Dynamic Deployment of Web and Grid Services", *Proceedings of the IEEE International Conference on Web Services*, 2007. ICWS 2007. 9-13 July 2007, pp.26-34.

[9] Sun Microsystems, "JAXP Reference Implementation", Available at: https://jaxp.dev.java.net/

[10] Sun Systems, "JSR 172 J2ME Web Services" Online at: http://jcp.org/en/jsr/detail?id=172

[11] Tilley, S.; Gerdes, J.; Hamilton, T.; Huang, **S**.; Müller, H.; Smith, D.; and Wong, K. "On the Business Value and Technical Challenges of Adopting Web Services." *Journal of Software Maintenance and Evolution: Research and Practice*, 16(1-2):31-50. John Wiley & Sons, April 2004.

[12] Trevor M., "Power: A First-Class Architectural Design Constraint" *Computer*, v.34 n.4, p.52-58, April 2001.

[13] Vinoski, S., "More Web Services Notifications", Internet Computing, IEEE, vol.8, no.3, May-Jun 2004, pp. 90-93.

[14] W3C, "SOAP Specifications" Online at: http://www.w3.org/TR/soap/

[15] W3C, "Web Services Description Language (WSDL) Version 2.0 Part 0: Primer" Online at: http://www.w3.org/TR/wsdl20-primer/

[16] W3C, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language" Online at: http://www.w3.org/TR/wsdl20/

[17] W3C, "Web Services Glossary" Online at: http://www.w3.org/TR/ws-gloss/

[18] Welch, V., et al., "Security for Grid services", *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, 2003. 22-24 June 2003, pp. 48-57