Remote computing resource management from mobile devices by utilising WSRF

Shihong Huang* and Michael VanHilst

Dept. of Computer Science & Engineering,
Florida Atlantic University,
777 Glades Road, Boca Raton, FL 33431-0991, USA
E-mail: shihong@cse.fau.edu E-mail: mike@cse.fau.edu
*Corresponding author

Junwei Cao

Research Institute of Information Technology,
Tsinghua University,
Beijing 100084, China
E-mail: jcao@mail.tsinghua.edu.cn

Jan Mangs

Lockheed Martin,
5600 Sand Lake Road, Orlando Florida 32819, USA
E-mail: jmangs@gmail.com

Abstract: The increasing processing power and capabilities of mobile phones enable them to become the next generation of computing platform. Mobile devices provide ubiquitous, inexpensive and powerful computing resources that people could use wherever they are. The increasing availability of web services and grid computing has made easier the access and reuse of different kinds of services. Web services provide network accessible interfaces to application functionality, while grid computing enables the efficient distribution of computing resources and power. In the future, higher functioning smart homes will need these capabilities. As a step in bringing grid computing to smaller devices, this paper presents a framework that enables the use of small mobile devices to conduct computing resource management in a distributed environment. The mobile devices access stateful web services on a Globus-based grid environment. To illustrate the presented framework, a phone-based 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: mobile devices; smart home; resource management; grid computing; web services; WSRF, Globus.

Reference to this paper should be made as follows: Huang, S., VanHilst, M., Cao, J. and Mangs, J. (xxxx) 'Remote computing resource management from mobile devices by utilising WSRF', *Int. J. Computer Aided Engineering and Technology*, Vol. X, No. Y, pp.000–000.

Biographical notes: Shihong Huang is an Assistant Professor in the Department of Computer Science and Engineering at the Florida Atlantic University. Her research interests include program reverse engineering, comprehension and redocumenation, and software maintenance and evolution. She was the main Research Associate for the 'Software recycling' project joint work with BMW Group, and an investigator for 'One pass to production' project joint work with Motorola. She was General Chair of the *24th ACM International Conference on Design of Communication (SIGDOC 2006)* and Program Co-Chair of the *9th IEEE International Symposium on Web Site Evolution (WSE 2007)*. She received her PhD from the University of California, Riverside. She is a member of the IEEE Computer Society and a member of the ACM.

Michael VanHilst is an Assistant Professor at Florida Atlantic University. His research history includes software development methods, software security, mobile agent and personal assistant software, and data visualisation. He has 20 years of industry experience with HP Labs, IBM Research, CNRS (French National Centre for Scientific Research) and the Harvard Smithsonian Center for Astrophysics. He received his PhD from the University of Washington in Seattle and his Master and Bachelor's from the Massachusetts Institute of Technology.

Junwei Cao is currently a Professor and Assistant Dean of Research Institute of Information Technology, Tsinghua University, China. He has worked on advanced computing technologies and applications for more than ten years. Before joining Tsinghua in 2006, He was a Research Scientist at the LIGO Laboratory, Massachusetts Institute of Technology and as a research staff member at the C&C Research laboratories, NEC Europe Ltd., Germany. He received his PhD in Computer Science from University of Warwick, UK and his Master and Bachelor's from Tsinghua University. He has published over 80 academic papers and books. He is a Senior Member of the IEEE Computer Society and a member of the ACM and the CCF.

Jan Mangs is currently with Lockheed Martin in Orlando, Florida. Prior to working at Lockheed, he was funded by a National Science Foundation Partnership for International Research and Education Program (PIRE) grant to work on addressing the special issues and developing a methodology and framework for migrating web and grid applications to small mobile devices. He also worked on model driven engineering as a Research Assistant funded by Motorola. He earned his Bachelor and Master's from Florida Atlantic University.

1 Introduction

The smart home of the future will act in many ways like a friend. The level of service anticipated will require more computing power and different models of interaction than what we see today. To take one example, to achieve higher levels of communication than simple task-specific commands will require learning an individual's patterns, accumulating context, and a great deal of search and reasoning (Oh and Woo, 2005). The underlying mechanisms for interaction must enable components to work together seamlessly and maintain state. To achieve this potential, tools and techniques being developed to solve large science scale problems on a grid network will eventually

transfer to solve more personal problems with smaller devices in the networked home. We have already seen the mechanisms of distributed computing evolve from large scale distributed systems to sensor networks and networks of components on a chip, including the Common Object Request Broker Architecture (CORBA) remote procedure call (RPC) protocol (Gidding, 2006). Virtualisation is also coming to the smart home (Perumal, 2008). In this paper we describe web services in grid computing and report on work to migrate their capabilities to the scale needed for use in the smart home. To facilitate development at this initial stage, we used a cell phone and tasks involving manual interaction.

Web services are network-accessible interfaces to application functionality. Although there have been some discussions of the numerous technical challenges to fully utilise their potential (Tilley et al., 2004), web services have already 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' (Haas and Brown, 2004). An individual who wishes to utilise 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) (Chinnici et al., 2007) and developers utilise this description to automatically generate client stubs that use Simple Object Access Protocol 2001 (Gudgin et al., 2007) invocations to access services.

Otherwise known as SOAP, the Simple Object Access Protocol is responsible for allowing servers and clients to communicate with each other regardless of their platform-specific details. The most important point to remember is that web services as defined by the W3C do not save state information.

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 relative lack of reusable end-user applications (Foster, 2006). 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 in order to improve some of the drawbacks of the first generation grids 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 (Cjazkowski et al., 2004). OGSI was originally intended to provide the infrastructure to add stateful resources to web services. However, the advent of new web service standards such as WSDL 2.0 (Booth and Liu, 2007) and WS-Addressing (Box et al., 2004) necessitated the creation of the Web Services Resource Framework (WSRF). The WSRF standard is relatively similar to OGSI with the exception of syntax and naming convention changes. Unlike stateless Web services, WSRF allows web servers that implement it to create, maintain and manage resources. Instead of existing as a singular entity at one time or another, stateful web services exist for a finite (or infinite) period of

time. As long as they are not destroyed they are associated with a resource on the service provider. These resources can be a range of objects: an endpoint reference to a web service, a database connection, or simply an integer or string value. The ability to save state information is crucial to enabling grid computing through web services. Without states to maintain, collaboration in grid computing environments becomes very difficult to accomplish while utilising web services. Without the ability to save and effectively manage state information, stateless web services are ill-suited for grid computing.

1.1 The growth of mobile devices

The explosive growth of mobile devices continues unabated. Each new generation of cell phones and PDAs are smaller, faster and consume less power. Most mobile devices in existence today have the capability to handle both voice and data. Almost all models of cell phones can access the internet through cellular transmission and some have Wi-Fi capabilities. First generation cell phones were nothing more than glorified analogue FM receivers. They utilised the traditional FM spectrum to transmit cellular calls and were costly to both operators and consumers. The advent of second-generation cell phones was enabled by the transition to digital communications and the introduction of voice encoding and compression. The current third generation provides features such as text messaging, picture messaging and access to internet-related functionalities. Because mobile devices do not require extensive landline infrastructure, mobile devices are much easier for people in developing countries to acquire. For example, while the ratio of cell phones to PCs is 0.9 in the USA, the ratio of cell phones to PCs in China is an astounding 3.6 cell phones to every PC (Kanellos, 2005). Many developing nations are simply skipping over more traditional avenues of communication technology in favour of less expensive and more attractive solutions like cellular phones. The availability of mobile devices to people all over the world provides a new potential source of computing power to exploit.

Web services have existed on the internet for a relatively long time but support for them on mobile devices is still relatively limited. For stateful web services, there is little existing support among the major cell phone manufacturers. The platform independent nature of web services makes them an ideal match for mobile devices. Because mobile devices do not require extensive landline infrastructure, mobile devices are much easier for people in developing countries to acquire. The availability of mobile devices to people all over the world provides a new potential source of computing power to exploit.

1.2 Utilising mobile devices in grid computing

In regards to distributed computing, mobile phones are in a position similar to personal computers: they are 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 computation and storage, the processor speed and storage capacity of mobiles continues to grow rapidly with the advent of low-power processors and flash memory

(Mudge, 2001). 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.

Currently, because cell phones and other mobile devices have been seldom envisioned as being powerful enough for grid computing, there is little support for their use in grid environments. Attempting to develop a Java-based cell phone application to access stateful web services exposed by a grid toolkit such as Globus revealed a major problem. The software running on the cell phone cannot access the grid services offered by Globus because Globus implements the WSRF standard for stateful web services (Foster et al., 2005). The most notable attempt to solve this problem was a project called 'wsrf4j2me' which had full or partial support for WS-Addressing, WS-ResourceProperties, WS-ResourceLifetime and WS-BaseFaults (Knerr, 2009). While promising, the project has not seen any action since August 2006 and seems to have been abandoned by its creator. No other notable attempts to fully integrate WSRF into Java ME or other mobile platforms can be readily found.

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 utilise WSRF on a mobile device. It presents a framework created to address the issues that occur when trying to utilise WSRF from a mobile device. The purpose of the framework presented 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

3 to lay the foundation for incorporating mobile devices into the grid environment.

The rest of the paper is organised as follows: Section 2 presents the technology upon which the framework is initially based and details some of the difficulties in interactions between the environments of grid and mobile computing. Section 3 presents the proposed framework that allows mobile devices to access WSRF-enabled web services. Section 4 describes a case study done to demonstrate the functionalities of the proposed framework. Section 5 provides a summary of the paper, points out the limitations of the framework in its current state and describes the future work planned for the framework.

2 Background

This section presents the background upon which this research work is based. 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. In order to appeal to the many users of grid software as well as mobile device users, the framework had to be applicable to the largest number of users in both domains. Although the framework that is proposed in this paper is applicable to all web servers that implement WSRF, not every WSRF implementation is equal. At the same time, the framework had to be implemented on a specific software platform for use on mobile devices. This section describes the details of the server-side component that implements WSRF and the client-side software platform upon which the framework is

built, the reasons why each was chosen, and the difficulties in establishing successful communication.

2.1 Grid component – Globus Toolkit

The open source Globus Toolkit (Foster, 2006) is a toolkit used for building distributed computing grids. It allows organisations 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 US 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, pursuing Globus-based grid strategies (Globus Alliance, 2008).

One of the main reasons the Globus Toolkit was chosen as the platform to develop our framework was its implementation of WSRF (Pu and Lewis, 2007). 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.

The Globus Toolkit provides the most complete implementation of WSRF and has superior response times to invocation calls when compared to WSRF.NET, WSRF::Lite, and pyGridWare (Humphrey et al., 2005). Because the Globus Toolkit has a robust implementation of WSRF, it eliminates the problem of dealing with an incomplete server-side implementation of WSRF and concentrates work on the client-side issues of communicating with WSRF-based web services from a mobile phone. In addition, the Globus Toolkit offers a high-performance version written in the C language and a platform-agnostic version written in Java. Combined with the substantial support and use among both academic and professional institutions, including the University of Chicago, US Argonne National Laboratory, University of South Carolina and IBM, the Globus Toolkit was chosen as the WSRF implementation to test our framework development. Although the case study in Section 4 is run solely on the Globus Toolkit, the standardised nature of WSRF allows it to be ported to other implementations that adhere to the same standard.

2.2 Software platform – Java ME

The target platform on which our framework is 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 simply that 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 that can run Windows Mobile.

The JSR-172 web services package (Coward, 2008) is a subset of the Java API for XML-based remote procedure calls (JAX-RPC) (GlassFish, 2009). 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.

Although HTTPS support is stated to be included in the JSR-172 package (Coward, 2008), it was revealed through testing and examining source code that the package only creates unsecure HTTP connections and does not support TSL. Also, JSR-172 does not support complex data types that are required to communicate to Globus web services (Globus Alliance, 2008). The lack of support for complex data types results in an inability to communicate with any web services inside Globus that require more than a simple data type such as a string or integer when using JSR-172.

3 The framework

Because of the limitations mentioned in Section 2.2, the use of the JSR-172 standard was not possible when communicating to Globus. The JSR-172 standard is fairly basic and only provides support for simple XML RPC commands over Java ME. It is not adequate when used to communicate to WSRF-based web services because JSR-172 has no built-in support for the WSRF standard and does not allow the use of complex XSD data types which are required in order to communicate with Globus. The framework described in this section was created to fix the problem of communicating to stateful web services (such as those running in Globus) by implementing standards required by WSRF and allowing the use of complex data types.

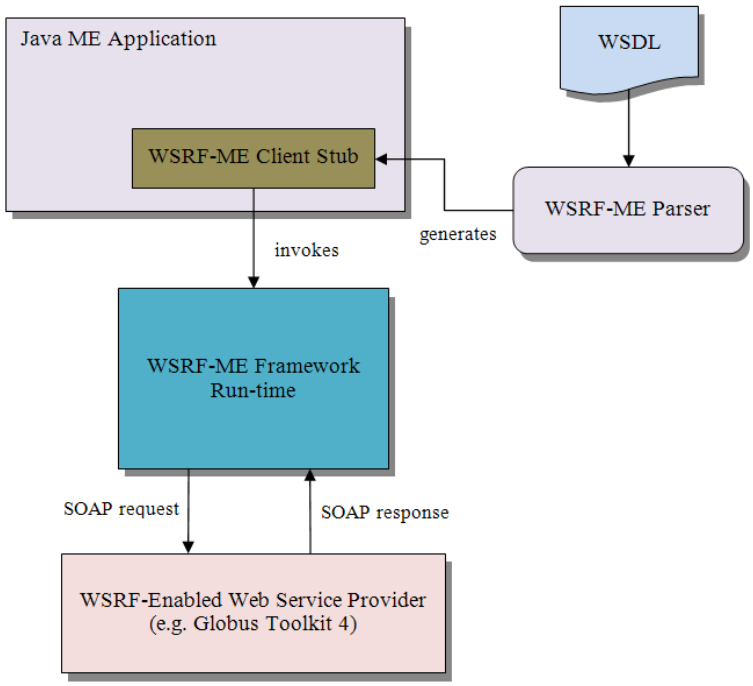
3.1 Architecture of the framework

The methodology applied to this framework was to emulate the strong points of the JSR-172 specification including its standardised approach and efficient support of web service invocations while addressing the problems of a lack of WSRF support, a non-WSRF-compatible parser and incomplete support for complex types. The idea is to improve in the areas where JSR-172 is lacking by leveraging concepts used in Apache Axis-based Java clients running in desktop environments. In these Axis-based clients, users do not need to deal with many of the stub properties present in JSR-172 and can simply utilise an addressing class to automatically set the various properties associated with an endpoint address. In the WSRF-ME framework, one goal was to support the concept of functionality with ease of use. Because the WSRF-ME framework operates in a limited Java ME environment, limitations had to be set as to how simplified the implementation of a client stub could be versus its ease of use. A parser is included in the framework to automatically generate code and complex types from a WSDL file, this allows developers to bypass the tedious portions of code generation and deal with any specific issues they may have with their WSRF-based web services.

Since the SOAP encoder and decoder used by JSR-172 were not available publicly, the framework required the use of a custom SOAP 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. This method was derived from Apache Axis as it is used in Globus (Feller et al., 2007).

In Figure 1 the architecture of the WSRF-ME framework and its relationship with the source WSDL files, the Java ME platform and the WSRF web service provider is detailed. The WSRF-ME framework as shown in the diagram consists of three parts: the WSDL parser, the client stubs and the framework run-time. The process begins with the developer creating or obtaining a WSDL file for a web service. This WSDL file is then input into the WSRF-ME parser to generate the client stubs required by a Java ME application to communicate with that web service. The developer then can utilise an instance of a client stub object and sets its associated stub properties such as the service endpoint, resource key and WS-Addressing messaging properties accordingly. In the same manner as which client stubs are used with JSR-172, the developer typecasts a binding stub to a portType class object and then invokes whichever method of the target web service that they require. The WSRF-ME run-time handles the invocation to the target web service and handles the encoding and decoding of the SOAP request and response respectively. The run-time returns any data from the web service according to its method invocation to the Java ME application as well. For example, a client portType could define an integer as its return value; if the invocation is successful, the run-time will return the integer value to the Java ME application running on the mobile device.

Figure 1 WSRF-ME framework architecture (see online version for colours)



3.2 Description of the framework

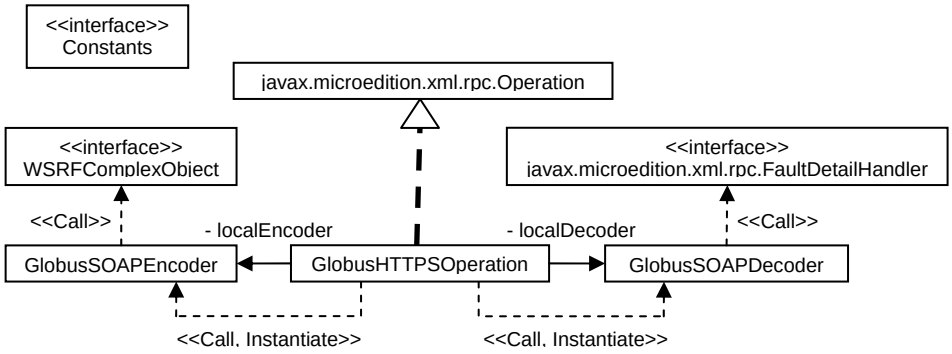
In JSR-172, when the developer utilises a method defined by the port type, it requires complex input that is not a simple data type. There was no clear way to submit the information required. There are no methods to define custom XML tags in the SOAP message such as those utilised by WS-Addressing. This meant that it was not possible to input enough data in the SOAP message to successfully communicate to the Globus server. The solution to this limitation required the aforementioned object-oriented approach. 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 a 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. For example, assume a complex data type called `PersonnelType` which consists of two complex data types called `PersonalInfoType` and `AddressType`. The developer would create a new `AddressType` and `PersonalInfoType` object and fill each with their respective information. `PersonalInfoType` and `AddressType` both implement `GlobusObject` and its `GenerateMessage()` method. The developer simply has to call the `GenerateMessage()` method of each object in `PersonnelType`'s `GenerateMessage()` method. When the `PersonnelType` is passed to an invocation, the encoder will automatically construct the SOAP message through the `GenerateMessage()` method. This 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.

Finally, one key benefit is the ability to directly control the size of the SOAP messages, it is possible to use all the features supported by WS-Addressing or just the bare minimum. As long as the message contains the very minimum required by that service, it would still function correctly. For example, instead of defining all the fields such as to, from, action and so on in the SOAP header, the framework can simply define the corresponding endpoint reference (EPR) and resource key and still successfully utilise a Globus service such as Web Services Grid Resource Allocation Management (WS GRAM). This can apply to the entire SOAP envelope and applies as long as the message satisfies the information required by the Globus web service container.

The high-level view of the WSRF-ME framework run-time is shown in Figure 2. It consists of the three main classes: GlobusHTTPSOperation, GlobusSOAPEncoder and GlobusSOAPDecoder. Each class has a specific role in the process of handling a remote web service invocation. The GlobusHTTPSOperation class is the main orchestrator for communicating with a web service. Since the SOAP encoder and decoder used by JSR-172 were not available publicly, the WSRF-ME framework required the implementation of a custom SOAP encoder and decoder. The GlobusSOAPEncoder and GlobusSOAPDecoder handle the request and response details related to the process of invoking a web service.

Figure 2 WSRF-ME framework runtime



3.3 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 (Vinoski, 2004). 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. Since WSRF is platform independent, any web service provider that implements WSRF should be able to utilise our framework to communicate with their respective web services. Although the WSRF-ME framework has primarily been tested with the Globus Toolkit, its functionality is applicable to any other implementations that correctly implement the WSRF.

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 stags to utilise 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. The WSRF-ME framework's support for the WSRF is summarised in Table 1.

Table 1 WSRF-ME framework's WSRF support

Specification	*Support*	*Description*
WS-Addressing	Full	Supports both EPRs and messaging properties
WS-Resource	Full	Supports WS-Resources as part of web service invocations
WS-ResourceProperties	Full	Pre-generated client stubs/types for all WS-RP methods
WS-ResourceLifetime	Full	Pre-generated client stubs/types for all WS-RL methods
WS-BaseFaults	Partial	Returned as an exception
WS-Notification	None	No support
Specification	Support	Description

4 Case study

In order to demonstrate the functionalities of the framework and provide a user-friendly manner in which to utilise its functionalities, an application is created and built upon the framework libraries previously created. The case study application demonstrates the effective use of the WSRF-ME framework to communicate with Globus web services that utilise WSRF. In addition, the application is intuitively geared towards a mobile experience; the assumption is made that data input on a cell phone is less efficient than in desktop environments. Instead of forcing a user to directly invoke the operations needed to utilise job submissions and reliable file transfers, the application streamlines the various steps into the mobile phone's user interface.

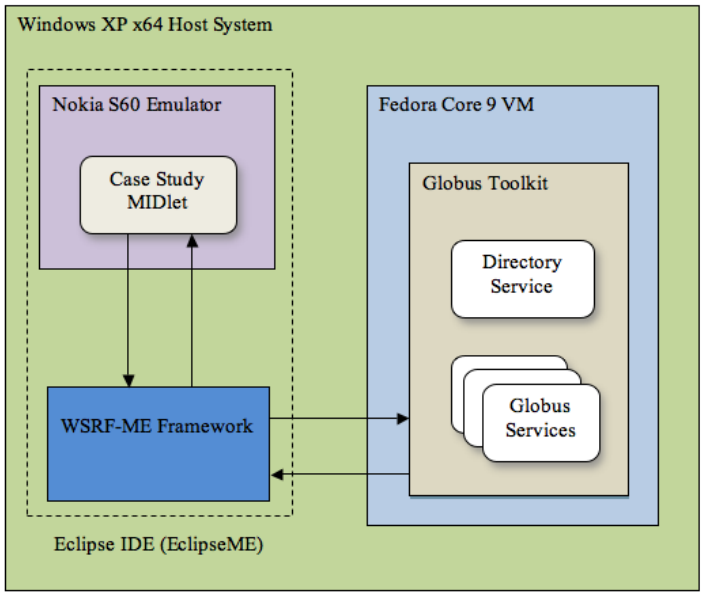
4.1 Simulation platform: Nokia S60

Although it was decided early on that the case study application would be written in Java ME, other questions remained such as on what phone to test the application, how to test the application and what environment would host the Globus services. Rather than choosing to concentrate solely on one single model of a phone created by a manufacturer or concentrating on the entire range of mobile devices available from a manufacturer, the decision was made to pick an intermediate point. It was decided to concentrate on a product line of phones to allow for the widest possible utilisation of our framework while ensuring the functionality remained identical on all devices. After some research, it was decided to restrict the case study implementation to Nokia's S60 platform. The reasoning for this choice is two-fold: Nokia supports a fully open-source initiative for its Symbian OS on which the S60 platform is based and the number of devices supported by the S60 platform is extensive.

The latest S60 SDK provided by Nokia comes with an emulator and integrates into Eclipse through a plug-in called EclipseME (Setera, 2009). 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.

Because the objective of this case study is to demonstrate the use of the WSRF-ME framework on a wide variety of devices, the application needed to be developed on a platform that supports a wide multitude of phones and not just a small subset of one manufacturer's product line. Nokia's S60 platform provides a wide range of support for various Nokia phones as well as other manufacturer's phones that utilise the S60, like Samsung, LG, and Panasonic (Nokia, 2007). In addition, Nokia's global market share for mobile devices is about 40% (Nokia, 2008). In the case of mobile grid computing where the quantity of computing platforms is as important as the impact, widespread use and acceptance of Nokia devices is important. Rather than forcing developers to possibly adopt new devices to test the case study application created in this work, the use of a standardised product line platform allows for simpler replication of these results by reducing interoperability issues between different phone platforms. In Figure 3, the architecture of the case study simulation environment is detailed.

Figure 3 Case study simulation environment (see online version for colours)



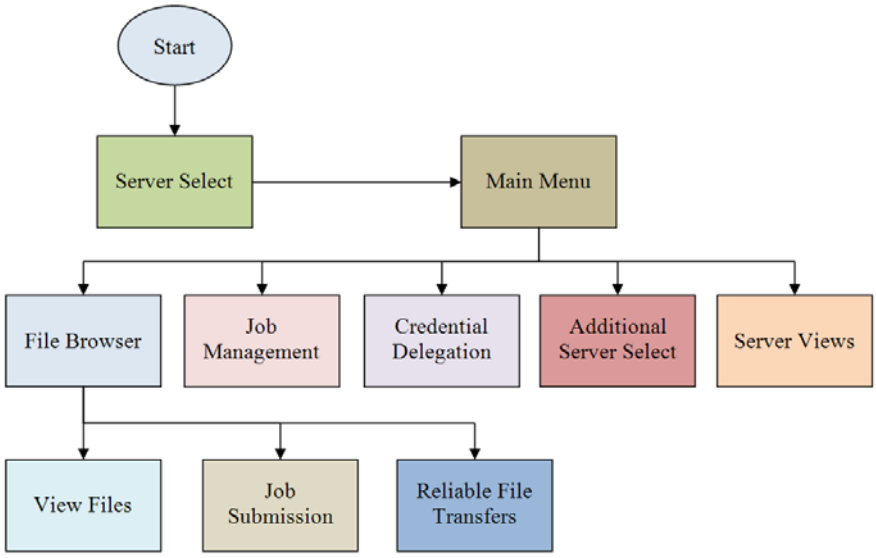
4.2 Implementation overview

The client application for the phone is able to submit jobs, perform third party file 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 see only one 'view' of the server connection at a time but is free to independently switch between them at will. In order to connect to a server, the phone requires a valid user certificate from a certificate authority.

Instead of creating a new certificate for the mobile phone, the application reused a certificate that was obtained beforehand. In order to do this, it required the conversion of the user certificate/key from the PEM format used by Globus to the PCKS12 format that is utilised by S60 platform phones. Fortunately, installation of certificates is straightforward and simply required transferring it to the phone's storage. Once installed, the phone will recognise user certificate requests from the server automatically and prompt the user to select from an installed certificate on the phone. The phone also prompts the user if they wish to temporarily or permanently trust the server's certificate when they connect to the server and receive its certificate.

In this case study, the application was built using Nokia's S60 3rd Edition Feature Pack 2 Software Development Kit and was developed in Eclipse using the EclipseME 1.7.9 plug-in. Globus Toolkit 4.0.7 (GT4) was utilised 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. These services are WS GRAM, reliable file transfer (RFT) and the delegation service.

Figure 4 Case study basic functions flow chart (see online version for colours)



4.3 Features of mobile application

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 and management etc., Figure 4 illustrates the basic functionalities that can be done by this framework. The next sections will describe some selected features in the framework, such as file browsing and remote job submission by using mobile devices.

Figure 5 Main menu screen (see online version for colours)

4.4 Main screen

The main menu screen presents the user with several options shown in Figure 5. 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 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 the 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.

Figure 6 File browser (see online version for colours)

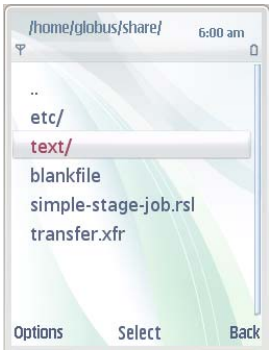
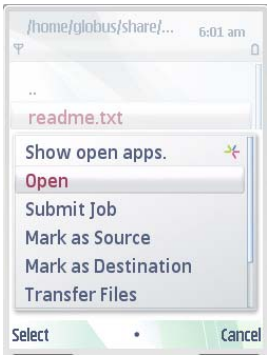


Figure 7 File browser option menu (see online version for colours)



4.5 File browser

To provide the functionalities of remote resource management, such as browsing files located on a different machine, conducting file transfer, submitting jobs remotely etc., a user friendly user interface, i.e., file-browsing interface, is created as shown in Figure 6. This 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, some of which are show in Figure 7. 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. The application is able to transfer files both locally and remotely using the file browser using the source and destination marking features. File deletion is also supported much like file transfers; users simply mark the file they wish to delete and then start the file deletion from the file browsers. The file deletion command is not shown in Figure 7 due to the limited resolution on the phone.

For mobile job submission, the application allows the mobile device to get jobs through the 'submit job' option as shown in Figure 7. In a separate screen which can be accessed from the main menu through 'view jobs', users are able to perform operations such as 'destroy' to remove a finished job and 'get status' to query a job resource property for its state. The resulting EPR of a job submission through "Submit Job" and the result of 'get status' on that same job are shown in Figure 8. The left screen shows the information returned by Globus when a job is created, the right screen shows the status of the job retrieved using the application's job manager.

4.6 Mobile job submission

The ability to submit jobs and manage their lifetimes demonstrates the possibility of leveraging stationary server resources for the benefit of a mobile user. Rather than perform intensive or repetitive tasks on their mobile device, a user can create a job description that allows them to allocate a workload to the Globus server. This functionality allows the mobile device to delegate computationally-heavy calculations to a remote server and retrieve the result without dealing with the issues normally involved with managing grid computing clusters. The end result is a savings in both battery power and time because the cluster grid environment can perform calculations much more efficiently than the single mobile device can. A single mobile device would require much more time to complete its calculations and also be limited by the amount of power it can consume in order to achieve the same results.

Figure 8 Mobile job submission process (see online version for colours)

Because the user interface design of cell phones is not normally well suited for efficiently inputting large amounts of textual information, applications developed on cell phones must take this limitation into account. In the context of the case study, requiring the user to input information required to execute a job on a Globus server manually would be too cumbersome. Not all mobile devices are equipped with a fully-fledged keyboard to provide input. Even with an onboard keyboard, the small size of both the keyboard itself and the screen requires the user to carefully input any information to the device. It is not feasible to expect users to remember the locations of files or directories mentally of one or more servers' shared spaces. To solve the problem of limited user input, the application provides functionality that simplifies job submission by allowing the DirectoryService to utilise server-side files to construct a Resource Specification Language (RSL) job description.

The diagram from Figure 8 presents the steps that a user would follow in order to remotely submit a job description called 'simple-stage-job.rsl' from the mobile device. The user selects a valid RSL job description file that has been saved on the server and chooses the 'submit job' command. This causes the application to retrieve the file's contents from the server through the directory service. Once successfully retrieved, the application displays the basic information about the job that is about to be submitted and allows the user to verify the correctness of the job submission. The user then chooses to finally submit the file and the job submission are sent to the ManagedJobFactory for execution. If the RSL job description file is correctly formatted and the job submission is successful, the application then displays the time of the job creation, the termination time of the job, and an EndpointReference to the ManagedJobExecutableJobService and the job's corresponding unique resource identifier.

5 Conclusions and future work

This paper presented a framework for utilising stateful web services from a mobile device. A case study is demonstrated as how this framework can be used in remote job management by using small and mobile devices. The JSR-172 package as implemented only provides support for simple RPC operations. Our framework takes the concept of this package and expands upon it to allow developers to connect not only WSRF-enabled web services but also stateless web services. It allows for the use of complex data types and arrays, something which was impossible to do under JSR-172. 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 APIs already created for Java SE could not be directly reused, due to the limitations in the Java ME environment. It required some effort in order to understand what libraries were and was not available, and find ways to work around problems that required features that were not available. 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 many web service-based application from working properly. A poor connection through wireless or cellular means can result in slow performance.

One final limitation involves utilising the framework on non-Nokia devices. In order to utilise the framework on other phones, the manufacturer must support the same features in their MIDP implementation for Java ME (Nokia, 2009). Because the Globus Toolkit utilises Transport Layer Security (TLS) (Welch et al., 2003) the manufacturer must provide full support for TLS. Otherwise they will only be able to connect a non-secure container (Feller et al., 2007). Message level security is still possible in a non-secure container but performance suffers greatly. A reference implementation such as Sun's Java ME SDK does not support client certificates under TLS and as such does not work with our framework. The framework itself does not support message level security. So any service that requires authentication, such as WS GRAM, will not be useable on the small device. Security is a major concern in smart homes and must be addressed before connecting small devices to web and grid services. To reduce the threat of man-in-the-middle attacks, future web services will require key management on both sides. Work has already been done to address this problem, but more work remains to be done (Al-Muhtadi, 2000).

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 the user interface. 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. Currently, the Globus Toolkit does not support the listing of all jobs available by default. In order to get a listing of jobs submitted by other users, the container running WS GRAM must have audit logging enabled and be configured to allow for dynamic listing of all jobs. The method described is not entirely secure but in the future this framework will possibly support a secure method of obtaining the status of all job submissions. The framework may also implement an optional service that securely maintains a list of these jobs and then use the mobile phone framework to retrieve that list.

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

References

Al-Muhtadi, J., Anand, M., Mickunas, M.D. and Campbell, R. (2000) 'Secure smart homes using Jini and UIUC SESAME', *Proceeding for the 16th Annual Conference on Computer Security Applications*, pp.77–85.

Booth, D. and Liu, C.K. (2007) *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*, available at http://www.w3.org/TR/wsdl20-primer/, (accessed 19 June 2009).

Box, D., Christensen, E., Curbera, E., Ferguson, D., Frey, J., Hadley, M., Kaler, C., Langworthy, D., Leymann, F., Lucco, S., Millet, S., Mukhi, N., Nottingham, M., Orchard, D., Shewchuk, Sindambiwe, E., Storey, T., Weerawarana, S. and Winkler, S. (2004) *Web Services Addressing (WS-Addressing)*, available at http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/, (accessed 19 June 2009).

Chinnici, R., Moreau, J-J., Ryman, A. and Weerawarana, S. (2007) *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, W3C, available at http://www.w3.org/TR/wsdl20/, (accessed 19 June 2009).

Cjazkowski, K., Ferguson, D., Foster, I., Frey, J., Graham, S., Maguire, T., Snelling, D. and Tuecke, S. (2004) *From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution*, Globus Alliance, available at http://www.globus.org/wsrf/specs/ogsi_to_wsrf_1.0.pdf, (accessed 19 June 2009).

Coward, D. (2008). *JSR 172: J2ME Web Services Specification*, Sun Microsystems, available at http://jcp.org/en/jsr/detail?id=172, (accessed 19 June 2009).

Feller M., Foster I. and Martin S. (2007) 'GT4 GRAM: a functionality and performance study', *TeraGrid Conference*.

Foster I. (2006) 'Globus toolkit version 4: software for service-oriented systems', *Journal of Computational Science and Technology*, Vol. 21 No. 4, pp.523–530.

Foster, I., Czajkowski, K., Ferguson, D.E., Frey, J., Graham, S., Maguire, T., Snelling, D. and Tuecke, S. (2005) 'Modeling and managing state in distributed systems: the role of OGSI and WSRF', *Proceedings of the IEEE*, Vol. 93 No. 3, pp.604–612.

Gidding, C. (2006) 'Not your father's CORBA – an architecture for real-time and embedded systems', *RTC Magazine*, available at http://www.rtcmagazine.com/magazine/articles/view/100752/, (accessed 19 June 2009).

GlassFish (2009) *JAXP Reference Implementation*, Sun Microsystems, available at https://jaxp.dev.java.net/, (accessed 19 June 2009).

Globus Alliance (2008) *About the Globus Toolkit*, available at http://code.google.com/p/wsrf4j2me/, (accessed 19 June 2009).

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, Nielsen, H.F., Karmarkar, A. and Lafon, Y. (2007) *SOAP Version 1.2 Part 1: Messaging Framework*, 2nd ed., W3C, available at http://www.w3.org/TR/soap12-part1/, (accessed 19 June 2009).

Haas, H. and Brown, A. (2004) *Web Services Glossary*, W3C, available at http://www.w3.org/TR/ws-gloss/, (accessed 19 June 2009).

Humphrey M., Wasson, G., Jackson, K. Boverhoh, J., Rodriguez, M., Gawor, J. Bester, J. Lang, S. Foster, I. Meder, S. Pickles, S. and McKeown, M. (2005) 'State and events for web services: a comparison of five WS-Resource framework and WS-Notification implementations', *Proceedings of the 4th IEEE International Symposium on High Performance Distributed Computing*, IEEE, pp.3–13.

Kanellos, M. (2005) 'Cell phones outnumber PCs in China', *CNET News*, available at http://news.cnet.com/Cell-phones-outnumber-PCs-in-China/2110-1039_3-5978594.html, (accessed 19 June 2009).

Knerr, T. (2009) *Implementation of the Web Services Resource Framework (WSRF) for J2ME*, available at http://code.google.com/p/wsrf4j2me/, (accessed 19 June 2009).

Mudge, T. (2001) 'Power: a first-class architectural design constraint', *Computer*, Vol. 34 No. 4, pp.52–58.

Nokia (2007) *S60 Devices*, available at http://www.s60.com/life/s60phones, (accessed 19 June 2009).

Nokia (2008) *Annual Information 2008*, Nokia Corporation, available at http://www.nokia.com/about-nokia/financials/quarterly-and-annual-information/quarterly-and-annual-information-2008, (accessed 19 June 2009).

Nokia (2009) *Nokia Java ME Developer's Library*, available at http://www.forum.nokia.com/document/Java_Developers_Library_v2/, (accessed 19 June 2009).

Oh, Y. and Woo, W. (2005) 'A unified application service model for ubiHome by exploiting intelligent context-awareness', *Springer Lecture Notes in Computer Science*, Vol. 3598/2005, pp.192–202.

Perumal, T. (2008) 'Virtualization for smart home technologies', *Home Technology eMagazine*, Vol. 13, No. 4, available at http://www.hometoys.com/ezine/08.08/perumal/virtualization.htm, (accessed 19 June 2009).

Pu, L. and Lewis, M.J. (2007) 'Uniform dynamic deployment of web and grid services', *Proceedings of the IEEE International Conference on Web Services*, pp.26–34.

Setera, C. (2009) *J2ME Development using Eclipse*, available at http://eclipseme.org/, (accessed 19 June 2009).

Tilley, S., Gerdes, J., Hamilton, T., Huang, S., Müller, H., Smith, D. and Wong, K. (2004) 'On the business value and technical challenges of adopting web services', *Journal of Software Maintenance and Evolution: Research and Practice*, John Wiley & Sons, Vol. 16, Nos. 1–2, pp.31–50.s

Vinoski, S. (2004) 'More web services notifications', *IEEE Internet Computing*, Vol. 8 No. 3, pp.90–93.

Welch, V., Siebenlist, F., Foster, I., Bresnahan, J. Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L. and Tuecke, S. (2003) 'Security for grid services', *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, pp.48–57.