Combining AJAX and WSRF for Web-browser based Grid clients

Arjun Sen, John Brooke, Bruno Harbulot, Mark Mc Keown,
Stephen Pickles and Andrew Porter
sena AT cs.man.ac.uk and {john.brooke, bruno.harbulot, mark.mckeown,
stephen.pickles, andrew.porter} AT manchester.ac.uk

The University of Manchester, Oxford Road, Manchester M13 9PL, United Kingdom

November 2006

Abstract

This paper demonstrates the effectiveness of combining *Asynchronous Javascript and XML (AJAX)* and the *Representational State Transfer (REST)* architectural style with the *Web Service Resource Framework (WSRF)* standard. Accessing *WS-Resources* (according to WSRF) in a manner compatible with the REST principles, using a Web browser, makes it possible to provide a light-weight, portable client for interacting with scientific simulations in distributed computing environments, with similar functionality to pre-existing platform-specific clients that required a full installation procedure. This paper describes this client and emphasises which standards and principles of distributed computing have played a key role in putting this application together.

1 Introduction

This paper demonstrates the effectiveness of combining AJAX and REST with WSRF by looking at their use within the RealityGrid[1] project. RealityGrid facilitates the study of complex condensed-matter systems through the use of Grid technology. The majority of the physical scientists involved in the project use computer simulations of one form or another in order to investigate the behaviour of different systems. The size of these simulations frequently necessitates the use of High Performance Computing resources which are increasingly being made available as a part of Grids such as the National Grid Service,[2] in the United Kingdom, and the TeraGrid,[3] in the United States.

The project described in this paper provides a light-weight client for interacting with WS-Resources (see Section 2.1) using only a Web browser, via the use of AJAX techniques (see Section 2.4) and REST principles (see Section 2.2). The RealityGrid use-case and reference implementations are presented in Section 3. Section 4 presents in detail the AJAX-based RealityGrid client, before Section 5 concludes.

2 Underlying technologies and concepts

This section presents the technologies and concepts that have been utilised throughout this project. First, the Web Service Resource Framework, which has been used to implement computational steering in RealityGrid, is presented in Section 2.1. Then, the REST architectural style is presented in Section 2.2. Finally, two sets of

[1] http://www.realitygrid.org/

[2] http://www.grid-support.ac.uk/

[3] http://www.teragrid.org/

technologies used in Web browsers are described: XSLT and AJAX, in Sections 2.3 and 2.4, respectively.

2.1 Web Service Resource Framework (WSRF)

The *Web Service Resource Framework (WSRF)* is a set of OASIS standards[4] for modelling stateful entities using Web services [FFG+04]. Within WSRF, a *WS-Resource* is defined as "*the composition of a resource and a Web service through which the resource can be accessed*" [GKM+06]. A *resource* is defined in WSRF as a logical entity that has the following characteristics: it must be identifiable, it must have a set of zero or more properties, which are expressible in XML infoset, and it may have a lifecycle [GKM+06]. Unfortunately, as presented in Section 2.2, the word "resource" has a different definition in REST.

The concept of a resource within WSRF is an abstract one and does not necessarily have to map to a real physical resource. For example, a WS-Resource could be used to model a physical object such as a printer or something more abstract like a computer simulation. The set of properties of a WS-Resource is expressed in the *ResourcePropertyDocument*, which is a projection of the state of the resource (written in XML). These properties, called *ResourceProperties*, can be accessed via Web service operations defined by the WSRF specifications [GT06]; in particular, a property can be queried and, when the WS-Resource permits it, inserted or updated. WS-Addressing[5] *EndpointReference*s *(EPR)*, are used for addressing WS-Resources.

The implementation of WSRF that has been used throughout this project is WSRF::Lite,[6] which is written in Perl. The main particularity of WSRF::Lite is that, whilst conforming to the WSRF standard, it is also designed to be usable according to the principles of REST, as described in the following section.

2.2 Representational State Transfer (REST)

Representational State Transfer (*REST*) [Fie00, Chap. 5] is an architectural style for building large-scale distributed systems. It consists of a set of principles and design constraints which were used in designing the protocols that make up the World Wide Web [JW04].

REST is based on a client-server model which supports caching and where interactions between client and server are stateless; all interaction state is stored on the client for server scalability. The concept of a resource is central to REST. Resources have an identity and anything that can have an identity can be a resource. Resources are manipulated through their representations and are networked together through linking — hypermedia is the engine of application state. Together, the last set of constraints combine to make up the principle of uniform interface. REST also has an optional constraint for the support of mobile code.

HTTP [FGM+99] is an example of a protocol that has been designed according to REST. On the World Wide Web a resource is identified by a URI [BLFM05] and clients can retrieve a representation of the resource using an HTTP GET.[7] HTTP can also supply caching information along with the representation to allow intermediaries to cache the representation. The representation may contain links to other resources creating a network of resources. Clients can change the representation of a resource by replacing the existing representation with a new one, for example using an HTTP PUT. All resources on the World Wide Web have a uniform interface allowing generic pieces of software such as Web browsers to interact with them. Web servers are also able to send code, for example Javascript

[4] `http://www.oasis-open.org/committees/wsrf`

[5] `http://www.w3.org/Submission/ws-addressing/`

[6] `http://www.sve.man.ac.uk/Research/AtoZ/ILCT`

[7] HTTP GET is the default action performed by Web browsers when a URI is entered.

(see Section 2.4), to the client to be executed within the browser.

WSRF::Lite supports REST in that: (a) it uses only the address part of an EPR for identifying a WS-Resource — there is a direct mapping between EPRs in WSRF and URIs in the REST style — and (b) it maps certain WSRF operations to HTTP methods. *GetResourcePropertyDocument* maps to HTTP GET, *PutResourcePropertyDocument* maps to HTTP PUT and *Destroy* maps to HTTP DELETE. For example, an HTTP GET on an HTTP URI from the address element of an EPR of a WS-Resource in WSRF::Lite returns the ResourcePropertyDocument for the WS-Resource. Similarly an HTTP PUT on the same URI will replace the ResourcePropertyDocument and an HTTP DELETE will destroy the WS-Resource. The ResourcePropertyDocument returned in response to an HTTP GET will not be wrapped in SOAP and, if the developer includes an XSLT transformation for the WS-Resource, WSRF::Lite will automatically include a link to the XSLT in the ResourcePropertyDocument.

2.3 Extensible Stylesheet Language Transformations (XSLT)

Extensible Stylesheet Language Transformations (*XSLT*) [ABC+01, XSL99] make it possible to express the rules for the transformation of one XML document format into another. Modern Web browsers include XSLT engines to perform these transformations upon receipt of an XML document. Typically, this is specified in the original XML document by including an XML processing instruction containing a reference to an XSL stylesheet (of the form "`<?xml-stylesheet type="text/xsl" href="URI-of-stylesheet"?>`").

XSLT is used in the context of this project to transform XML documents as returned by WS-Resources into XHTML documents, which are Web-pages ready to be rendered by Web browsers.

2.4 Asynchronous Javascript and XML (AJAX)

Asynchronous Javascript and XML (*AJAX*) is a technique for developing interactive Web sites. AJAX is not a technology in itself, rather it is the combination of a number of technologies: DOM, HTTP and Javascript.[8] AJAX allows a Web browser to update parts of a Web page asynchronously by communicating with a Web server using Javascript. Javascript is a client-side scripting language for Web browsers. Most modern graphical Web browsers are capable of executing Javascript scripts that are embedded into webpages. Example applications of Javascript are: raising and re-dimensioning pop-up windows, changing an image when pointing the mouse cursor over a zone and validating the content of a form before submission.

In the traditional processing model of the Web, entering a URI into a Web browser causes it to execute HTTP GET [FGM+99] request on the URI. The Web server hosting the URI sends a response to the Web browser, usually an HTML document, which the browser renders and displays for the user. The content of that HTML document is analysed and elements of the page that refer to other URIs are retrieved in the same manner (for example, images). This process finishes once all the elements required to render the page have been obtained.

The AJAX model differs from this processing model in that it makes it possible to update a page or some of its elements asynchronously. An HTML document can contain Javascript, or a link to a Javascript document defining various routines, which the browser may execute. The

[8]The term *Javascript* is used in this article to encompass all the variants of the language: the original Netscape JavaScript (which has now become Mozilla JavaScript [moz06]), Microsoft JScript and the standardised ECMAScript [ECM99]. Most differences between these dialects are minor; they may cause implementation issues for compatibility between browsers, thus developers are encouraged to test their Javascript code on multiple platforms. The main browser used throughout this project has been Firefox.

Javascript can use `XMLHTTPRequest`[9] to communicate with a Web server, for example to retrieve data to update part of the Web page. Because only a part of the page is updated the whole page does not have to be re-rendered, improving the user experience. The Javascript does not block when sending or receiving data using `XMLHTTPRequest`, allowing the browser to continue processing the Web page; therefore, AJAX is useful for creating Web pages that need to poll servers. `XMLHTTPRequest` can also be used to send data to the Web server.

3 Use-case: RealityGrid Computational Steering System

This section presents the use-case which has motivated the RealityGrid project. Traditionally, a scientist runs large-scale simulations non-interactively. A text file describing the initial conditions and parameters for the simulation is prepared. Then, the simulation is submitted to a batch queue, waiting until there are sufficient resources available for it to be executed. The simulation runs entirely according to the prepared input file and outputs the results to disk for the user to examine later.

This technique is suitable for some forms of investigation, but for others it can lead to a very inefficient use of resources. A solution to this problem is to provide the scientist with a way to interact with the simulation while it is running — a process that is called "computational steering" [MvWvL99]. This may be as simple as allowing the user to monitor the values of some parameters in the simulation and, if necessary, to edit the values of other parameters. However, to aid the scientist in making informed decisions, it is often necessary to make it possible to see a visualisation of some aspect of the simulated system as it evolves.

The RealityGrid Computational Steering System (RCSS) [PHPP05] provides a scientist with tools for performing steering on both local and remote simulations, potentially in collaboration with others. In a Grid context, it is the ability to steer remote simulations that is of importance. The RCSS implements this functionality using WSRF as implemented in WSRF::Lite.

The architecture of the system is shown in Figure 1 and consists of four principal components: the steered simulation built against the RealityGrid steering library, the Steering Web Service (SWS), the Registry and the Steering Client. The Registry is implemented as a WS-ServiceGroup (part of the WSRF standard) and is hosted in a WSRF::Lite Container. The SWS is implemented as a WS-Resource and is also hosted in a WSRF::Lite Container (which may be the same as that of the Registry). The simulation communicates with the SWS using the RealityGrid steering library which in turn uses the gSOAP toolkit[10] to perform SOAP messaging.

The SWS effectively provides a Web service interface to the steerable simulation with the majority of the functionality provided through standard WSRF operations such as GetResourceProperty and SetResourceProperty (Section 2.1). This Web service interface has made it possible to construct a variety of steering clients. These include a desktop steering client which enables the user to connect to multiple simulations simultaneously, as well as displaying (continuously updating) plots of parameter values. A client based upon the Microsoft .NET framework has also been developed for PDA and Smartphone platforms [HK06]. Finally, a Web-portal based steering client [BEG+05] has also previously been developed using the GridSphere[11] framework. This provides some of the functionality of the desktop client but does not refresh the values automatically. Of all the steering clients mentioned so far, this is perhaps the most convenient from a user perspective due to the removal of the need to install any software —

[9] `http://www.w3.org/TR/XMLHttpRequest/`

[10] `http://gsoap2.sourceforge.net/`

[11] `http://www.gridsphere.org/`

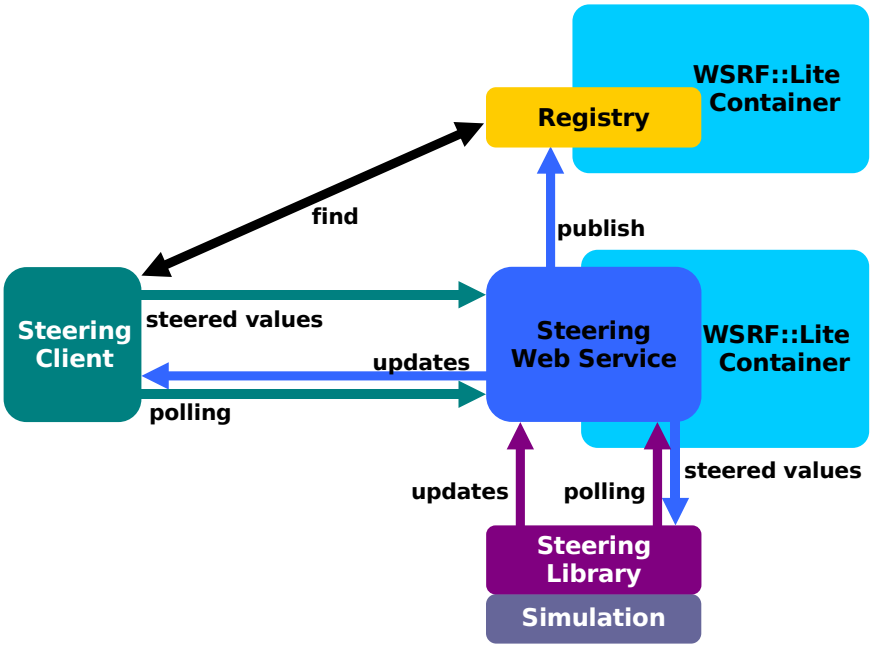

Figure 1: Architecture of the WSRF-based RealityGrid Computational Steering System.

a Web browser is all that is required. However, it requires user interaction for updating and, from a server administration point of view, the installation of the required GridSphere environment is quite an overhead to deployment.

The next section presents the AJAX-based client that circumvents these limitations.

4 The RealityGrid AJAX client

This section provides a detailed description of the RealityGrid AJAX steering client to show how AJAX, REST and WSRF can be combined. The overall architecture of the system is illustrated in Figure 2.

A RealityGrid user wishing to attach to a running simulation enters the URI from the address element of the EPR of the SWS (which may be obtained by querying the Registry — Figure 1) in the location field of the Web browser. This triggers an HTTP GET on this URI, performed by the Web browser. The WSRF::Lite Container hosting the SWS responds by returning the ResourcePropertyDocument of the SWS. This XML document includes a processing instruction containing a link to the XSLT stylesheet (see Section 2.3) for the SWS ResourcePropertyDocument.

Once the XSLT stylesheet is also retrieved, the Web browser uses it to transform the XML of the ResourcePropertyDocument into an XHTML document which is rendered by the Web browser (as shown in Figure 3, for example). Clients that are not Web browsers, or clients that do not understand XSLT, may ignore the XSLT transformation.

Embedding the XSLT stylesheet in the retrieved ResourcePropertyDocument causes the transformation to happen on the client-side. However, the same result could be achieved by applying the transformation on the server-side, and HTTP is able to provide support for this through content negotiation.

The XSL transformation is sufficient to extract parameter values from the ResourcePropertyDocument and display a representation of them in an attractive fashion, however it is only a static representation. AJAX enables the Web

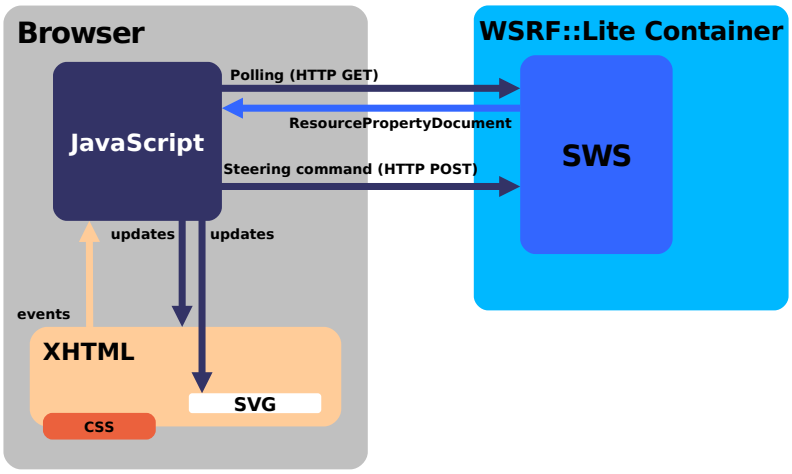

Figure 2: Architecture of the AJAX-based steering client and Steering Web Service.

Figure 3: Screenshot of the Control Panel of the Web Steering Client displayed in Firefox.

browser to modify parts of the XHTML document without reloading the whole page. As indicated in Figure 2, the XHTML document contains a reference to a Javascript file and makes use of the functions defined therein to provide functionality beyond that of displaying the values of key ResourceProperties. This includes the ability to perform automatic polling of the WS-Resource for updating the Web page and partial user-triggered updates of the WS-Resource's properties.

The polling is performed by a Javascript function invoked at regular intervals (the duration of the interval can be controlled by the user). This function performs an HTTP GET request on the WS-Resource, but the processing of the result is performed within the function, so as to update only the relevant parts of the XHTML document being displayed by the browser.

In addition to the automatic updates, Javascript enables the implementation of graphical widgets such as panels and buttons that can be used to build a Graphical User Interface (GUI). This GUI enables the user to interact with and change the values of properties of the WS-Resource.

The ResourceProperties of an SWS can be divided into two categories: (1) meta-data describing the SWS and the simulation including information such as the TerminationTime of the SWS and the name of the running application, and (2) data and parameters pertaining to the simulation being steered. From these two categories, the user interface was designed to have four separate "panels" and a navigation bar:

- Control Panel: this panel enables the user to control the Web Steering Client itself as well as to send generic commands (such as *detach* or *stop*) to the steered simulation. It displays the meta-data derived from the ResourcePropertyDocument of the SWS — see Figure 3.

- Steering Panel: this panel is similar to that of our original desktop steering client. It has four sections: Monitored (read-only) Parameters, Steered (writable) Parameters, Data IO and Checkpoint Types.

- Plot Panel: this panel is managed almost entirely by Javascript methods for drawing plots based on the data displayed in the Steering Panel (particularly "Monitored Parameters"). This is discussed below.

- Debug Panel: developing a relatively complex application in Javascript is difficult due to the lack of debugging tools; this panel allows debugging output to be displayed and also incorporates a text area in which the most recent XML exchanged between `XMLHTTPRequest` and the SWS may be viewed.

- Navigation bar: this is a part of the interface with buttons to switch between each of the panels mentioned above.

In addition to the ability to poll the SWS and thus monitor the parameter values of the simulation, a steering client must also permit the user to change the values of (steerable) parameters. In this steering client, a SOAP message containing a SetResourceProperty call is constructed and sent using the Javascript `XMLHTTPRequest` via an HTTP POST. It is therefore possible to update the ResourceProperties of the SWS which in turn results in the steered simulation picking up new parameter values from it.

This approach was taken here because the SWS does not yet implement the SetResourcePropertyDocument method. Once that method is implemented, it will be possible for the steering client to simply do an HTTP PUT of the XML containing the updated ResourceProperty — there will be no need to construct a SOAP message.

Previous experience with other steering clients has shown that users find the ability to view plots of parameter histories very useful, particularly due to the feedback they can provide on steering activity. This function is provided in this

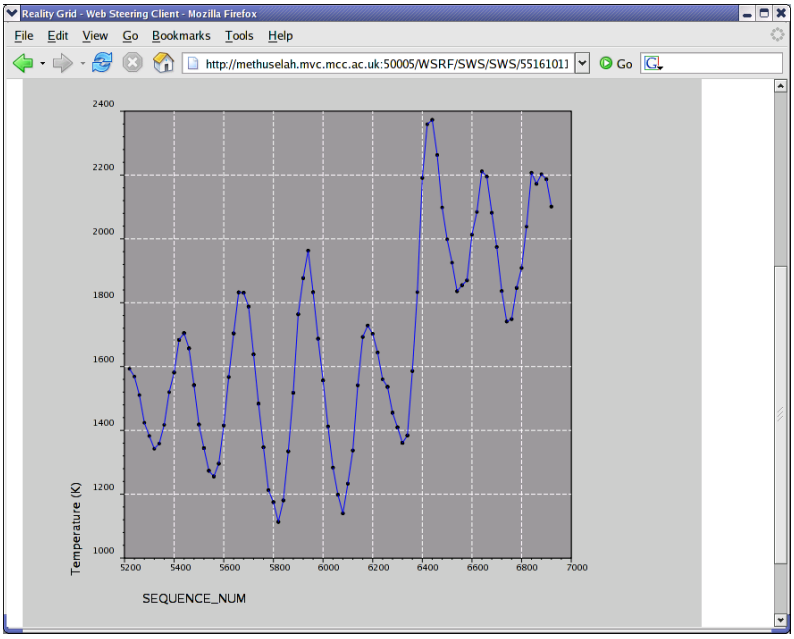

Figure 4: Screenshot of the Plot Panel displayed in Firefox.

client through a combination of Javascript and Scalable Vector Graphics (SVG).[12] Javascript code logs the parameter values obtained through polling the SWS and constructs a plot using SVG. Figure 4 shows an example obtained by monitoring the system temperature in a Molecular Dynamics simulation of crystalline Silicon. At a SEQUENCE_NUM of approximately 6400 the target temperature is steered from a value of 1500K up to a value of 2000K. The response in the system temperature can be seen in the plot.

Security is supported using X509 [HFPS99] certificates and Transport Layer Security [DA99]. The user imports his X509 certificate into the Web browser and the browser uses it for mutual authentication with the WSRF::Lite Container over a secure HTTPS [Res00] connection.

5 Conclusions

The paper has illustrated that combining AJAX and REST with WSRF can be a very effective approach to creating Grid clients. It provides a bridge between the world of Grid computing that extensively uses WSRF and the World Wide Web. For the RealityGrid project the advantages of the AJAX client are clear: users require only a Web browser and no longer need to install client software, clients and services can be updated without danger of version mismatch between server and client, and the RealityGrid steering client can be integrated into *Mashups*.[13] Another important factor for RealityGrid was that it did not require any modifications to the existing code base implemented for WSRF::Lite; all that was required was the creation of XSLT and Javascript files to be included in the deployed installation of WSRF::Lite.

Future work will include supporting the caching facilities of HTTP. This will allow the client to use conditional GET so that it will only receive a new set of ResourceProperties if they have changed since the last retrieval, thus reduc-

[12] `http://www.w3.org/Graphics/SVG/`

[13] `http://www-128.ibm.com/developerworks/library/x-mashups.html`

ing the network load and the client processing overhead.

References

[ABC+01] Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, and Steve Zilles. *Extensible Stylesheet Language (XSL) Version 1.0*. October 2001. http://www.w3.org/TR/xsl/.

[BEG+05] Mark G. Beckett, Matthew D. Egbert, Paul J. Graham, Kevin Stratford, and Jean-Christophe Desplat. The realitygrid steering portal. In *Proceedings of the UK e-Science All Hands Conference, Nottingham*, 2005. http://www.allhands.org.uk/2005/proceedings/papers/484.pdf.

[BLFM05] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. January 2005. http://www.ietf.org/rfc/rfc3986.txt.

[DA99] T. Dierks and C. Allen. *The TLS Protocol*. January 1999. http://www.ietf.org/rfc/rfc2246.txt.

[ECM99] ECMA. *ECMA-262: ECMAScript Language Specification*. ECMA (European Association for Standardizing Information and Communication Systems), third edition, December 1999. http://www.ecma-international.org/publications/standards/Ecma-262.htm.

[FFG+04] Ian Foster, Jeffrey Frey, Steve Graham, Steve Tuecke, Karl Czajkowski, Don Ferguson, Frank Leymann, Martin Nally, Igor Sedukhin, David Snelling, Tony Storey, William Vambenepe, and Sanjiva Weerawarana. Modeling stateful resources with web services, May 2004. http://www-128.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf.

[FGM+99] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. June 1999. http://www.ietf.org/rfc/rfc2616.txt.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[GKM+06] Steve Graham, Anish Karmarkar, Jeff Mischkinsky, Ian Robinson, and Igor Sedukhin, editors. *Web Services Resource 1.2 (WS-Resource)*. Organization for the Advancement of Structured Information Standards (OASIS), April 2006. http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf.

[GT06] Steve Graham and Jem Treadwell, editors. *Web Services Resource Properties 1.2 (WS-ResourceProperties)*. Organization for the Advancement of Structured Information Standards (OASIS), April 2006. http://docs.oasis-open.

org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf.

[HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo. *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*. January 1999. http://www.ietf.org/rfc/rfc2459.txt.

[HK06] Ian R. Holmes and Roy S. Kalawsky. The realitygrid PDA and smartphone clients: Developing effective handheld user interfaces for e-science. In *Proceedings of the UK e-Science All Hands Conference, Nottingham*, 2006.

[JW04] Ian Jacobs and Norman Walsh, editors. *Architecture of the World Wide Web, Volume One*. December 2004. http://www.w3.org/TR/2004/REC-webarch-20041215/.

[moz06] *Core JavaScript 1.5 Reference*. Mozilla, August 2006. http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference.

[MvWvL99] J. D. Mulder, J. J. van Wijk, and R. van Liere. A survey of computational steering environments. *Future Generation Computer Systems*, 15:119–129, 1999.

[PHPP05] S. M. Pickles, R. Haines, R. L. Pinning, and A. R. Porter. A practical toolkit for computational steering. *Philosophical Transactions of the Royal Society*, A363:1843–1853, 2005.

[Res00] E. Rescorla. *HTTP Over TLS*. May 2000. http://www.ietf.org/rfc/rfc2818.txt.

[XSL99] XSL transformations (XSLT) version 1.0, November 1999. http://www.w3.org/TR/xslt.