Using Web services in the European Grid of Solar Observations (EGSO)

Simon Martin and Dave Pike

Space Science and Technology Department, Rutherford Appleton Laboratory, Chilton, OX11 0QX, UK
{simon.martin, c.d.pike}@rl.ac.uk

Abstract. The European Grid of Solar Observations (EGSO) [1] is employing Grid computing concepts to federate heterogeneous solar data archives into a single 'virtual' archive, allowing scientists to easily locate and retrieve particular data sets from multiple sources. EGSO will also offer facilities for the processing of data within the Grid, reducing the volume of data to be transferred to the user. In this paper, we examine the use of Web services in EGSO as a means of communicating between the various roles in the system.

1 Introduction

To understand the Sun, solar physicists need access to data from a variety of instruments scattered across the globe. Data are stored in archives with varying degrees of accessibility. Even if easily accessible via the Internet, these archives are heterogeneous, with the metadata catalogues describing the data varying widely between archives [1]; hence obtaining the desired data can often be difficult. Additionally, the volumes of data involved are very large. Current archives may have accumulated as much as 1TB of data, whilst future missions may produce this quantity of data in a day. As well as the problems associated with searching large archives, transferring vast amounts of data across networks is undesirable.

EGSO is a Grid test-bed whose main aim is to improve access to solar data. This will be achieved by federating distributed data archives, creating standardised meta-data catalogues of the data available and providing users with tools to search these catalogues for specific data sets and retrieve them, whilst insulating the user from the details of data access [2]. EGSO will reduce the amount of data transfer required by providing data processing facilities (e.g. to calibrate datasets) within the Grid; hence EGSO is both a data and service Grid [3].

In this paper we briefly outline the EGSO functional architecture (section 2) and describe how Web services are being employed in the current phase of the EGSO project (section 3). Section 4 assesses the suitability of Web services for this purpose.

2 The EGSO Functional Architecture

The functional architecture for EGSO defines three separate roles [4]: consumers, providers and brokers (an organisation may play multiple roles). In simple terms, a consumer represents the user interaction with EGSO, and interacts initially with a broker to discover which provider(s) may hold the desired data (or service). The consumer then contacts the relevant provider(s) to obtain the requested data (or service). Providers are usually linked to data centres and offer data access facilities, but may offer services such as data processing. Brokers collect information from providers, such as metadata catalogues or details of their services, which can then be used by consumers to perform data searches; the system has multiple brokers which can behave as a single 'virtual broker', although this multiplicity is invisible to consumers. Typical information to be exchanged between roles includes data files, images, fragments of metadata catalogues, and details such as authentication data or session IDs.

Roles interact with each exclusively via an *external interaction subsystem*, which must support passing messages of the types listed above, whilst being loosely coupled to the rest of the role to allow possible replacement as Grid technologies mature. The subsystem contains components which allow the consumer to interact with the broker and provider, the provider to interact with the broker and consumer, and the broker to interact not only with consumers and providers, but also with other brokers.

3 Using Web services in EGSO

Web services represent a service-oriented approach to distributed computing, with services accessed via XML messaging over Internet-based protocols for platform-independence [5]. Standards [6] such as XML and SOAP ensure interoperability, whilst UDDI and WSDL allow the discovery and description of Web services. Although Grid middleware is available (e.g. the Globus toolkit [7]), at this time we have decided to use Web services for inter-role communications (i.e. in the external interaction subsystem) in EGSO for several reasons. Web services are compliant with direction of W3C and industry, they are platform independent, they are lightweight, and can be easily replaced and deployed on systems. They are also loosely coupled, and enable remote procedure call (RPC) and document exchange type Web services to be implemented, synchronously and asynchronously [8]. Furthermore, the Globus toolkit is starting to implement the Open Grid Services Architecture (OGSA) [9], which integrates Web services and Grid technologies and concepts; OGSA is not yet a mature technology, but we are then well positioned to implement it if necessary in place of Web services.

3.1 Implementation

Document exchange and RPC-type Web services were investigated to determine their suitability for use in EGSO. Sun's Java Web Services Developers Pack (JWSDP) v1.1

[10] was used to implement both types of Web services. RPC-type Web services were also developed using Apache Axis, a SOAP implementation [11].

To develop RPC-type Web services and clients, the JWSDP provides the Java API for XML-based RPC (JAX-RPC); the current Reference Implementation uses SOAP as the application protocol and HTTP as the communication protocol. The API hides much of the complexity from the developer, representing method calls and responses as SOAP messages. On the server side, the developer specifies remote procedures by defining these methods in a Java interface, and codes the relevant classes that implement those methods. On the client side, the remote method is called on a stub object, which acts as a proxy for the remote service. The JWSDP tools create any required classes (e.g. stubs) and deploy the Web service in a Web container (Tomcat).

Axis can create Web services in a similar manner, but also allows for very simple deployment of RPC-type Web services; Java classes with public methods can be exposed as Web services by simply placing them in a target directory. Clients can then be created using a simple Axis API to access the service. Both Axis and JAX-RPC also allow the use of dynamic proxies or dynamic invocation interfaces to access Web services whose WSDL descriptions are only known at runtime, although this method was not tested.

Document exchange-type Web services were developed with the JWSDP using the Java API for XML Messaging (JAXM) and SOAP with Attachments API for Java (SAAJ). JAXM provides classes and interfaces for creating a special type of servlet (JAXMServlet) which can send and receive SOAP messages, and for using messaging providers, discussed below. SAAJ is used for creating SOAP messages (with optional attachments) and sending them synchronously without using a provider. XML messages can be sent between applications with or without the use of a messaging provider. In the former case, a standalone JAXM client can run independently, or within a Web container. It sends a SOAP message synchronously over a connection to a listening JAXM servlet; this is known as request-response messaging.

Alternatively, JAXM applications can use messaging providers (they are then peers). A messaging provider is a service hidden from the developer that handles the transmission and routing of messages. Very simply, the client sends the SOAP message to its messaging provider with the details of the recipient(s) in the SOAP Header. The messaging provider then forwards the message to the servlet's provider, which then sends the SOAP message to the servlet. There are several advantages to using messaging providers including the fact that they are continuously active, and so a JAXM application can close its connections after sending a message and the provider will still send the message; the provider can also be configured to resend messages until they are successfully delivered, and will store incoming messages for the application ready for delivery upon reconnection. A significant advantage of messaging providers in the context of EGSO is the ability to send a message to multiple intermediate destinations before the message is delivered to its final recipient. The intermediate destinations, or actors, are specified in the header of the SOAP message. Providers can also incorporate profiles which are implemented on top of SOAP; these are specifications that tell providers how to route messages.

4 Assessment of Web services for use in EGSO

In developing both types of Web service, the largest barrier to progress was found to be incomplete documentation. RPC-type Web services were quite easy to deploy using JAX-RPC and Axis, with Axis being the simpler of the two. Document exchange Web services were also found to be easy to employ using JAXM; creating and sending simple SOAP messages synchronously was quite straightforward. However, the real strengths of using JAXM were found to be the ability to add attachments of any type (e.g. images, text files) to the SOAP message, and the ability to use messaging providers to not only ensure delivery of messages, but to send the message to multiple recipients.

There are several issues to be considered before committing to use Web services in such a large scale project. Security is a prime concern when using Web services e.g. [13]; issues include authentication (verifying the identity of the message sender), authorisation (determining whether the sender has permission to perform the requested operation), integrity (verifying that the message received is unmodified), and confidentiality (keeping the message private from unauthorised users).

There are also several quality of service (QoS) and performance issues which need to be addressed [14]. In terms of reliability, SOAP messages are transmitted using HTTP; hence there is no guarantee of packets being delivered to their destination. In terms of RPC and synchronous document exchange services, this is a problem as the SOAP message will need to be resent. However, this can be overcome with JAXM since messaging providers will re-send a message until it is delivered. The use of SOAP (XML) can cause performance problems, both in terms of applications parsing the XML and the fact that XML messages tend to be substantially larger than equivalent binary data, increasing bandwidth usage. Furthermore, network latency, web server/container performance under load and back end systems can also affect performance. However, given the vast amounts of data which need to be searched in order to locate a particular data set for a user, along with processing of this data, then many of these performance issues may be inconsequential.

5 Conclusions

Web services are relatively easy to develop and deploy. RPC-type Web services can easily implement simple method calls, or could be used to initiate more complex tasks through a composition of method calls. Document exchange using JAXM appears to be very well suited to use in EGSO, with its ability to send messages reliably, to multiple recipients, and to send non-XML content as attachments.

Some further investigations need to be carried out regarding issues such as scalability, optimisations, and security, although these are not barriers to implementing Web services in EGSO. Web services appear to be a viable method for communicating between the roles in EGSO, particularly in the early stages of production to allow integration testing of various components under relatively light load levels. The lightweight nature and loose coupling of Web services means that it

should be relatively easy to add new roles into EGSO, and the external interaction subsystem can also be easily replaced with Grid middleware if required.

References

1. http://www.egso.org
2. Csillaghy, A., Zarro, D. M., and Freeland, S. L.: Steps towards a virtual solar observatory. IEEE Signal Processing Magazine, N.18/2 (2001) 41-48
3. Foster, I., and Kesselman, C. (eds.): The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers (1999)
4. Piccinelli, G. (ed.): EGSO Architecture. EGSO Report EGSO-WP1-D4 (2003)
5. Vasudevan, V.: A Web Services Primer http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/index.html (April 2001)
6. http://www.webservices.org/index.php/article/archive/3/
7. http://www.globus.org
8. Chappell, D. A., and Jewell, T.: Java Web Services. O'Reilly and Associates (March 2002)
9. Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum (June 22, 2002)
10. http://java.sun.com/webservices/webservicespack.html
11. http://ws.apache.org/axis/
13. Deitel, H. M., Deitel, P. J., Gadzik, J. P., Lomelí, K., Santry, S. E., and Zhang, S.: Java Web Services for Experienced Programmers. Prentice Hall (2003).
14. Mani, A., and Nagarajan, A.: Understanding quality of service for Web services. IBM Developer Works, http://www-106.ibm.com/developerworks/library/ws-quality.html, (January 2002)