Translation Web Services — an Implementation for the IGNITE Project

Kevin Bargary

Kevin Bargary describes the Localisation Research Centre's (LRC) involvement in the Translation Web Services specification implementation, which is part of the IGNITE project currently being undertaken by the LRC and its partners; VeriTest, Archeptypon, PASS Engineering and Vivendi Universal Games.

IGNITE will pool together linguistic infrastructure resources and provide convenient access and a marketplace for them. This will satisfy a pre-requisite for the success of the European digital content industries.

Keywords: Translation Web Services, OASIS, Localisation, Standards, SOAP, WSDL.

Introduction

IGNITE is divided into three phases, the third of which is the 'Performance Analysis' stage of the project. This third phase aims to firstly verify standards and their use and implementation in tools and source material, and secondly, enhance the standards themselves by providing feedback from the project findings to the various standards committees. With this in mind the LRC became actively involved in the Translation Web Services (TWS) Technical Committee (TC) which operates under the umbrella of OASIS. Our main function as members of the TC was to provide an independent implementation of the draft specification that was already in place. As this was the first draft of the specification, and it had not been previously implemented, some issues were expected — it was our job to identify these and see how the specification functioned in a practical working scenario.

Web Services

The basic premise of TWS — and indeed any web services implementation — is that you have a *client machine* and a *server machine*. The server machine will contain a pre-programmed set of methods or functions that the client machine will access using web services. An example of this would be an online credit card validation system. One website (the client) connects to a remote service (the server) with the details; the service validates the credit card and returns the result to the website.

Implementation Work

The first stage was to decide on an implementation platform. The underlying technology in web services is SOAP (Simple Object Access Protocol). SOAP can be thought of as a message-passing system between two computers using the Hypertext Transfer Protocol (HTTP) over the Internet. The TC decided to use the J2EE development platform and the Java programming language. The rationale behind this decision was that the open source 'Apache Project' had a Java-based implementation of SOAP called AXIS. The AXIS implementation is a reliable and stable base on which to implement Java Web Services. This implementation provides an Application Programming Interface (API) into the SOAP actions that are required for implementing the TWS specification. The logical choice of web server to complement the use of Apache AXIS was Apache Tomcat — so this was used to host the web services on our internal LRC server.

The development process was based on the prototype model of software development. The first stage was to start with one of the 18 services currently available in the TWS specification, and from there to develop one service at a time and report back to the TC on any issues or suggestions for improvements that arose as we progressed. Table 1 shows a list of all of the services available in the current TWS specification. The services are divided into three categories; namely 'Required Services', 'Optional Services' and 'Recommended Services'.

Required Services	Optional Services	Recommended Services
submitJob	retrieveServiceList	rejectJob
retrieveJobInformation	requestQuote	associateResource
retrieveJob	acceptQuote	disassociateResource
retrieveActiveJobsList	retrieveQuote	retrieveResource Information
suspendJob	retrieveFullJobsList	retrieveFullResourceList
resumeJob		UploadFile
cancelJob		

Table 1: *List of services in the current Translation Web Services (TWS) specification.*

As previously mentioned, AXIS provides an API to the SOAP functionality. It also provides two command line utilities that further aid the implementation of web services. The 'Java2WSDL' utility takes pre-existing Java code and creates a WSDL file for that code. A WSDL (Web Services Description Language) file indicates how the client can access the service, i.e. what parameters need to be passed to the service in order to evoke a response. The TWS specification already has a WSDL, so this utility was useless for our purposes. However, the second utility, 'WSDL2Java', creates the Java stubs (Java files that contain the code needed to use SOAP) required by

(a) the server to write and deploy the service and

(b) the client to access the service through its own code.

Figures 1 and 2 show this process.

Firstly the Java stubs are created:

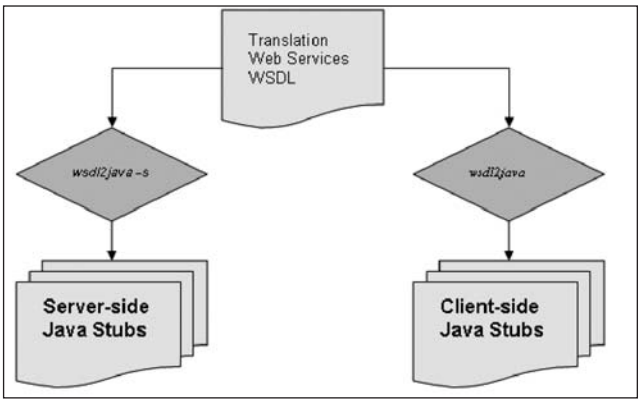

Figure 1: *Creating the Java stubs from the WSDL.*

Then the Java stubs are used by the client application to access the services and by the server to deploy the services.

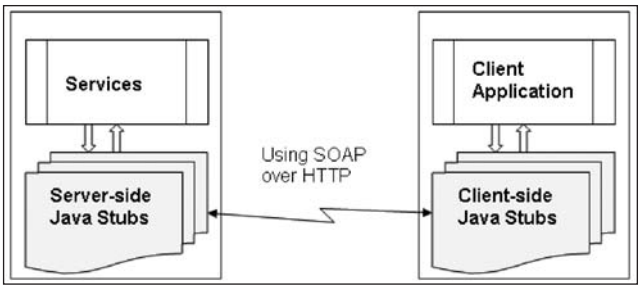

Figure 2: *Connecting client and server (through the Java stubs) over HTTP using SOAP.*

The first service that we implemented was the *retrieveServiceList* service. This service was chosen because there were no input parameters required for it. All that was required to invoke the service was an instance of the *retrieveServiceListRequest* class. The *retrieveServiceList* service returns "*a complete list of services offered by a particular vendor. This will include the languages dealt with and services offered by a particular vendor*" (Translation Web Services Specification Draft 1.0). After writing the server-side code to handle a *retrieveServiceListRequest*, i.e. return all of the appropriate values, the next stage was to create a simple test class that could instantiate a *retrieveServiceListRequest* and handle the results received back from the server in a *retrieveServiceListResponse*. With both classes now ready, we needed to deploy the services to the Apache web server. The WSDL file also contains the location of the service, i.e. where it can be accessed from. Deployment is necessary to ensure the service is in the location as defined in the WSDL. When the utility 'WSDL2Java' creates the Java stubs needed for the server-side machine, it also creates two other files that are used to deploy and '*un-deploy*' the service to a web server (Apache Tomcat). These files are called Web Service Deployment Descriptors ('deploy.wsdd' and 'undeploy.wsdd').

The implementation is currently on www.electonline.org:8080/index.html

The next stage in the development of the implementation was to write the code for the rest of the services. While writing the code we encountered some issues with the specification (including inconsistencies between the schema and the specification document). These issues were quickly amended by the TC. During the process of coding we also made some suggestions to the TC about possible improvements to the specification and we were actively involved in applying these changes. For example, the service 'retrieveQuote' in the original specification did not return any information about the location of the actual quote. This was deemed to be an important piece of information for this service and was promptly included in the specification.

With the code for the implementation of the services now written, the initial service deployed (*retrieveServiceList*) was undeployed and the full list of services was deployed to the web server. A JSP (Java Server Pages) client interface was developed to allow for the input of the parameters required for each service and also to show the responses from the server — see Figure 3.

Conclusions

Working on the implementation of the Translation Web Services specification has been very beneficial for the IGNITE project. One valuable lesson learnt during this implementation was the importance of having a reference implementation of a standard. This is vital to ensure the standard works in a practical environment. Becoming involved in the development of the TWS specification draft has allowed the IGNITE team to see how a standard comes together from initial design specification to an industry-accepted standard. The real benefit of the TWS standard will be seen when an implementation exists that incorporates the XLIFF standard for the exchange of localisable content through the localisation process. XLIFF was designed as an interchange file format for the loss-less exchange of data for localisation. If TWS is used as a medium for the transport of XLIFF, then a whole new localisation process in which automation is a key factor could evolve. Our work within the TWS TC has allowed us to be ingrained at each stage in the development of this standard. With the knowledge gained from this work we in the IGNITE project hope to research the possible marriage of these two important localisation standards in a working automated environment.

For more information on Translation Web Services visit http://www.igniteweb.org/documents.php to see a presentation by Peter Reynolds entitled *Web Services for Translation*, given at the IGNITE Working Conference in Dublin on 14 December 2005.

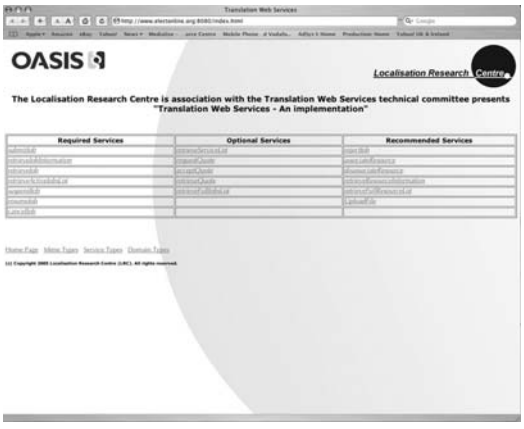

Figure 3: *JSP client interface used for inputting service parameters and showing server responses.*

The implementation is currently on www.electonline.org:8080/index.html

Kevin Bargary graduated from the University of Limerick in 2004 with a B.Sc. in Computer Systems. As well as working as a researcher on the LRC-coordinated IGNITE project, he is currently pursuing an M.Sc. in the area of open source localisation — also at the University of Limerick. He can be reached at Kevin.Bargary@ul.ie