[*Translating and the Computer 25*, November 2003 [London: Aslib, 2003]

Translation Web Services - A Reality

Ian Harris
Chief Technical Officer
thebigword
Belmont House, Wood Lane
Headingley, Leeds, UK
LS6 2AE

ianh@thebigword. com

Introduction

What is a web service

In simple terms, a web service is very similar to a simple web site. This simple web site does not contain any graphics or user interface, it simply returns text when it is contacted. Like a web site, different pages, or different information can be requested, and information can be passed to the web site, like fields on a form, that the web site then processes in some way to return some results.

With the absence of a user interface, web services are designed to be called from programs. In other words, instead of a person requesting a page (as a normal web site is used) web services are requested by programs on computers over the internet.

A simple program that tells you the latest weather on your desktop is probably linking to a web service somewhere on the internet, in the background, in order to get its information. The program will be sending some data to the web site to request the forecast, and the web service, at the Met Office or other, is returning a description of the weather to the program on your desktop. That desktop program is then formatting the data and displaying it to you in a user friendly form. This simple use for a web service is fairly trivial, but it does show some important things about web services:

1. They are used to get information from a site on the internet.

2. They have no user interface, so the calling program must read the response and do something appropriate with it.
3. They are invaluable when you need to use information in a program that it is impossible to get locally on your PC. The weather is an excellent example. If the program needs the weather forecast, this cannot be retrieved from anything Microsoft ship with PCs (yet), so the program must look externally for this information.

More about Web Services

When programmers create programs, they tend not to write the entire program themselves. Instead, they access code that someone else has already written. This is generally called the use of pre-written 'objects'. If the program that they are writing uses the time and date, they do not need to create their own counters and clocks. Instead, they simply access the system clock. The system clock is presented to the programmer in an easy to use way, as an object that they can call at any time they need it.

Other examples of objects a programmer may need to make use of are:

1. The spellchecker in Microsoft Word
2. Email
3. Graph drawing utilities

The list is actually endless, but of these three examples above, there is a common thread. They can all be contained on the local machine. For example, if I need to use spellchecking n my program, I simply need to ensure my users have Microsoft Word installed, and then I can access this spellchecker with little problem. Again with e mail, I will probably an object from Outlook to send a mail using the default account on that machine.

Objects such as those above are generally easy to use for programmers. The object can be selected with the mouse, and it then exposes itself and its requirements, such as, "if you pass me a word, I'll return 'yes' if I can find it in my spell check dictionary, and 'no' if not". This is obviously an over simplification, but this is the principle.

The problem comes when the programmer needs to access an object that is not on the local machine. This is where web services come in. The concept of web services is then simply accessing objects or services on remote machines over the web.

What is SOAP?

SOAP is a protocol for accessing web services. A protocol is a set of rules, and in the case of SOAP, it is a set of rules that define how the remote object will be accessed. The SOAP rules state that fields of data must be passed in a certain way (actually using XML,

for those familiar), and that the object or service will then respond in a certain way. SOAP doesn't specify the data that will be sent; this is individual to the particular web service, but it specifies the way it will be packaged up. It's like defining that when we speak today, the protocol will be English. It is up to the individual what they say, but it must be packaged and sent as English. As long as we all have this protocol installed, we will all understand.

Remote objects are kept fairly simple. They perform fairly simple functions with a question-and-answer type mechanism.

Now, hopefully, you can see how the name was devised. It is a protocol for accessing simple objects over the internet. Simple Object Access Protocol.

Life before SOAP

Text Interfaces

The idea of linking to remote machines for information that you cannot get locally is not new. This has been done in a number of ways in that past. One simple method of requesting and returning data is to use text files.

The programmer of one of the systems that needed to be linked (normally the one supplying the service) would define layouts for text messages to be sent to his system, and would define the layout of the information to be returned.

For example, if my program supplies weather forecasts, I may define that if you want a weather forecast from me, you need to send me a text file in the form below:

Field Name	Length
ClientAccessCode	3
DateForForecast	8

Example File Sent

0112102003

Where the ClientAccessCode is 011, and the date forecast is required for the date 21/02/2003

I may then define that if you send me the above data, I will return the following:

Field Name	Length
DateForForecast	8
Morning Weather	255

AfternoonWeather 255

Example File Sent

21102003Bright with sunny spells Pouring down

This, on the face of it, was not a bad way of passing information around. It could be passed over the internet, and could be used by programs and web servers, much the same as the web services above. However, there are some important drawbacks to this method:

1. The files are hard to read, and therefore the programs are hard to debug. In order to read (or parse) these files, you need to write some code that says, for example, read three characters starting at position 1 and put this into a variable called 'ClientCode', then read the next ten starting at position 4, convert it to a date, and put it into a variable called 'Date'. If your reading program gets one character wrong, the rest of the file is wrong. For complex files, this becomes unwieldy.
2. In such files, data is hard to organise. If you wanted to send a set of records, for example, a set of personnel records, these had to be organised into, perhaps lines of the file. Each person's record would go on a line of the file, with each line having defined character lengths for fields such as name, employee number, salary, etc. The problem is, the line feed or carriage return is the only record separator that you have at your easy disposal. This means that if you need a further level of records, for example, for each employee you want to send their salary history, you have a problem.
3. Such files are not extensible. If you use the example of sending personnel records as above, and you are sending records containing the person's name, employee number and salary, if you wanted at some stage to add a new field (because your personnel program has advanced and now stores 'Maiden Name', to change this interface file is a problem. You can either add the new field on the end of the line, or you can put it in a more sensible place, after Name. In either case, you need to tell everyone, wherever they are on the internet, who calls your program, and time the upgrade with them, so that the change does not break their programs.

Text files worked, but were unwieldy and not easily extensible. The programming of the interface using a text file took heavy developer resource. As a benchmark, let us say that a typical project, for the interfacing part alone, took three months programming (we will use this benchmark later on to compare with SOAP).

XML

In interfacing terms, the answer to all problems arrived in the form of XML. XML is still sent as a text file over the internet, but the data is packaged up and enclosed in 'tags'. These tags not only make the file easier for a human to read, but they also make the files easier to process, easier to organise (in records, sub records etc.), and they make the files extensible (hence the name eXtensible Markup Language).

The weather example above would probably be sent in XML in the following form:

```
<ClientAccessCode>011 </ClientAccessCode>
<DateForForecast>22/10/2003</DateForForecast>
```

With the returned file as below:

```
<DateForForecast>22/10/2003</DateForForecast>
<MorningWeather>Bright with sunny spells</MorningWeather>
<AfternoonWeather>Pouringdown</AfternoonWeather>
```

You will note that like HTML, the tags are descriptive, enclosed in '<' and '>' signs, and the closing tag leads with the '/' character.

Some points to note:

1. The file is easily readable by a human, and therefore easier for a programmer to deal with, process, and debug should anything not work correctly first time.
2. Data is easier to organise into records and sub records by adding fields within other fields.
3. The structure is extensible. I can add fields at any point, at any level, without breaking the original structure. As long as the programmers using this file have read it in a sensible way, I need not even tell them when I add new fields since their existing programs will still work.

XML became so popular that even from the outset, pre-written XML readers or parsers were available for programmers to use so that they did not have to get involved in the detail of reading in the individual fields for their programs to use. This was all done for them.

In interfacing terms, XML was a huge leap forward, and development times for interfaces, and the maintenance of interfaces was cut dramatically. In our benchmark example, interface times were cut from three months to one month. With no real problems, how could things possible improve?

SOAP

SOAP is built on XML, so does not fundamentally change the way the raw interfacing works from that of XML above. However, the important leap forward is that more of the

programming of the linking to the server, file reading, and grabbing of all the fields of data is pre-done for the programmer.

Using XML, the programmer must study the XML layout so he knows the fields to use, then must link to the server, download the XML file, parse the fields into variables within the program, and then start work doing whatever his program does with these fields.

Using SOAP, the programmer simply clicks on the web service he is trying to access, and this service exposes itself to him, showing all the fields that can be sent and returned. This is organised in the familiar form of an object so the programmer can use it like any other object. Although the data is actually sent as XML, the programmer does not see this. Instead, he sees an object that he can easily pass data and get answers in return; as easy as using the system clock to display the time.

Using our benchmark, development time on the interface is cut to a matter of days.

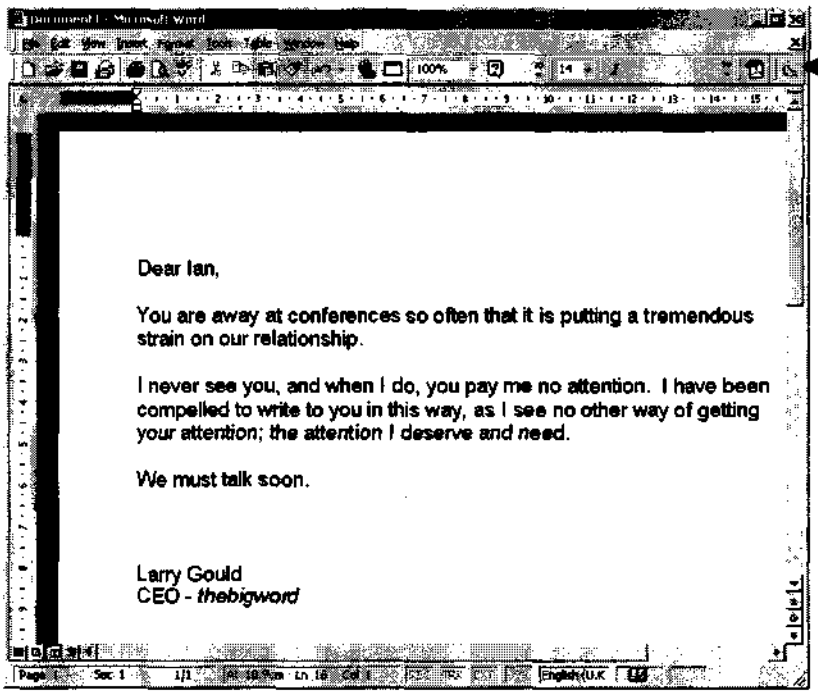
A Translation Web Service

Translation is the perfect use for a web service. Human translation can never be a resource that is local to an individual machine, and it always must be retrieved from elsewhere.

Imagine if we had a translation web service, what would this service do?

Text or a file could be passed into the service, along with a list of required languages, and then the service would return (maybe not instantly) the translated versions of that text or file. This would mean that a programmer could build language support into his program with ease. Any time a translation was required by the user, the program would go out to the internet and get it. All this would be managed by the program itself, not the user. The program would track outstanding translation requests, and organise the storage or display of the file on its return.

At *thebigword*, we have developed such a web service. One example of how this can be used is when linking to MS Word.



When in a document, the user can simply click the icon to send this document off for translation. The document will be encrypted before transfer over the internet (that is built-in to the web service).

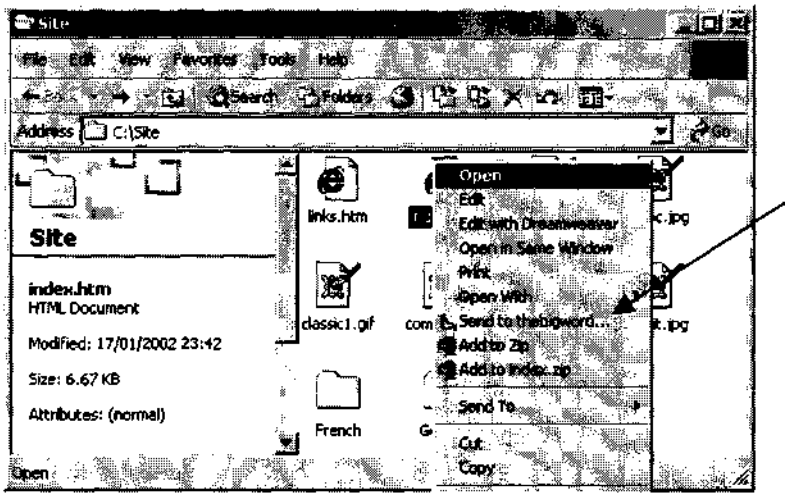
When translated, the document will be returned directly to the user's PC, again, over the web service, and again, encrypted for security.

What is happening?

When the button is pressed, this launches a program within MS Word that packages up the file and communicates with the web service over the internet. This program simply calls the web service, with all translation being done remotely.

This MS Word example is an example of a front end that has been placed on the web service. The web service does all the communication and organisation of translation, but nothing more. It needs the front end to communicate with the user, link to MS Word etc.

Another example is below. The front end here is the right-click within Windows.



When the user right-clicks a file, a new option shows ... to send the file for translation. Any file or group of files can b selected in this way.

Again, when translated, the document will be returned directly to the user's PC.

The two examples above have proved very popular, and extremely useful. However, for the real power of the translation web service, we have to look at how it is used to translate fast moving web sites built on Content Management Systems (CMS).

Multilingual Content Management

Content Management Systems are generally built on the idea of a workflow.

1. An author authors a document.
2. The CMS recognises this, and automatically e mails an editor to check the document.
3. Editor Edits the document.
4. The CMS recognises that this has happened and automatically e mails an approver, or whoever is next in the chain.
5. Approver approves.
6. CMS recognises this and the page goes live.

The workflows are completely user configurable, so do not need to follow the example above, with author → edit → approve.

Most large sites have multiple authors, multiple editors, multiple approvers. This is complex in one language, but imagine the site needed to be translated. Who will manage the translations? Will translators log in as another step in the workflow? What if the authors author in different languages? Do we need to copy and paste text out for

translation? How do we ensure nothing is missed, and nothing is translated twice? Can we leverage translation memory?

The answer lies in SOAP. Some important points:

1. Most Content Management Systems are built on a SOAP compliant platform, so are ready to link to SOAP web services.
2. All Content Management Systems have change detection built in. When an author authors a document, the CMS spots this change and sends an e mail. Then when the editor edits, the CMS spots this change and e mails an approver.

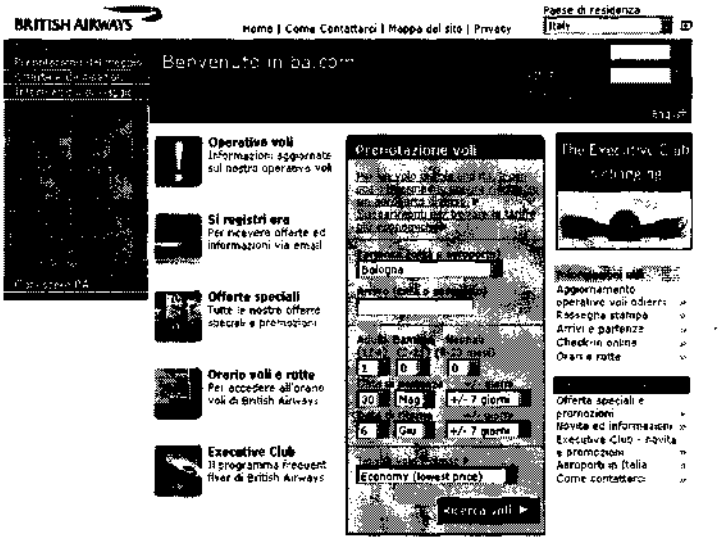
When the approver approves, all we have to do is get the CMS to fire a SOAP message to the translation web service (which most can do), and we have built-in translation management. There is no expensive 'Connector' technology or GMS system to install and own (at very high expense). Instead, we leave the CMS to do what it is built for. Detect changes and inform the next in the chain.

The translations, once complete, are simply returned to the CMS workflow and the CMS files them away or launches them, or passes to in-country reviewers, or whatever the next step is, as defined by the web manager.

We have now integrated with a number of Content Management Systems in this way. The case study we will investigate further is British Airways' web site, BA.com.

Case Study Localisation of BA.com

BA.com is built on a Content Management System called Interwoven Teamsite. This is an enterprise CMS, that runs many large and complex web sites.



BA.com has many types of content, such as navigational elements (menus), Functional elements (flight booking system) static content (marketing and flight information), and each of these can be local to one language, global to all languages, or global to a subset of languages. Content can also be authored in any language, and may need translating to the rest of the languages (or some, or none). The rules as to what needs translating change regularly.

Such a complex set up, with complex rules, needs to be managed by the web team at BA using Interwoven. BA do not need a complete reconfiguration or re-set up of any translation technology just because they decide to have a new section of the site translated.

The web team at BA simply configured Teamsite to send out SOAP based translation requests to *thebigword*'s SOAP web service any time it needed translation. The translation requests are then processed, the files are run against memory, translated, checked, and set for return by *thebigword*'s project manager for BA, then the web service returns them directly into the Teamsite workflow ready for the next stage of the process, what ever BA determine that might be for this type of content.

Key benefits of this set up are as follows:

1. The BA web team are in total control of the rules for translation.
2. No-one manages the translation process at BA. This would be an impossible job with the number of languages and level of changes. Even on small sites this becomes impossible. From BA's perspective, the translation is managed by the system.
3. Translators can translate in their CAT tools in an environment of their choice.

4. There is no expensive software set up, or cost of ownership of any connector technology. BA will never have to contact their translation supplier because of a technical problem on their web server.
5. The translation suffers no delays. All interfacing is automatic and instant.
6. All content is encrypted for security and confidentiality prior to launch.

You do not need a web site the size of BA.com to justify translating using web services. Any web site that changes frequently, and has multiple languages very quickly becomes impossible to manage in any other way. Once the language versions get out of step, it is almost impossible to recover.