Internationalization and Web Services

25th Internationalization and Unicode Conference
Presented by Addison P. Phillips
Director, Globalization Architecture
webMethods, Inc.
About This Presentation

- Duration: 40 minutes
- Audience: Internationalization professionals

The future of distributed computing is in Web services. How well do these new technologies support internationalization? What are the challenges with internationalizing Web services?
What are Web Services?

- A Web service is:
  “a software system designed to support interoperable machine-to-machine interaction over a network.”

- Interface defined in a machine readable format (WSDL)
- XML-based data interchange (SOAP)
- Standardized transports (HTTP)

- The key point:
  - Machine-to-machine interaction

*This definition is taken from http://www.w3.org/TR/ws-gloss/#webservice

Web services provide a way for systems to “componentize” functionality and to provide transparent access to diverse systems using open, standardized technologies. The core of Web services is the use of common transports (such as HTTP), data formats (XML, XML Schema), and protocols to create an interoperable way to collaborate, integrate, or choreograph applications.

At their core, Web services are the machine-to-machine version of the Web.
Web Services and Internationalization

Setting Sail for the New World

- Web services are an evolution of distributed software systems.
  - Client-Server
  - Web application
  - CORBA
  - Etc.

For all their apparent novelty, Web services are really just a new package for old ideas about building distributed systems. Many technologies have promised to deliver a heterogeneous system that can assemble virtual applications from “building blocks”, components, or disparate parts of a large enterprise network. These technologies have had limited success, generally restricted to large enterprise implementations, because the software has required a commitment to proprietary data formats, large scale planning and cooperative development, and other significant investments that put such integration beyond the reach of “average” users or systems.

Web services are interesting and successful because they are standardized, open, and based on technologies that are accessible. Interoperability has been a key focus of Web services early development, and this gives us the promise of distributed systems that can be secure, transactional, managed, scaled, and, yes, internationalized on an enterprise scale, while integrating software and solutions from nearly any vendor, no matter how small.
So what is a Web service? We need some terminology to have a useful discussion.

A Web service is really a method or function call. You can think of a Web service as a method on steroids. Just as with any method, a Web service has a parameter list (which can be empty) and a return value (which may also be empty or may be a document or object).

What makes the method a “Web service” is that you can invoke it by sending its Web service host, called a provider, a SOAP document (a specially formatted XML file) containing the information about which method to invoke and the data needed to fill in the parameter list. The provider decodes these values, calls the service, assembles any results, and can return another SOAP document as a response. It may also send an exception message (a fault) if something is wrong with the SOAP document or if something goes wrong during the invocation of the service. The exact pattern of inputs, outputs, and faults is called the message exchange pattern or MEP. There are many different MEPs that correspond the various different things a method call might do.

The client for this transaction is called the requester. The requester needs information about how to invoke the service: the URI where the service lives, the document (parameter list) to send, what document (if any) to expect in return, what errors might be returned instead, and so forth. These are described in the Web service description, which is what a WSDL file contains.

The WSDL file is exchanged through a process called discovery, which may be fully, partially, or not at all automatic. Automatic discovery mechanisms (pointers to Web service descriptions) sometimes use UDDI.

The nature of Web services is such that a provider can act as a proxy for another provider. This type of provider is called an intermediary.
Web Services and Internationalization

- What problem?
  - Locale-neutral representation (XML Schema)
  - No user interface (machine-to-machine)
  - Inherits XML’s rich support for Unicode, language tags, and so forth
  - “Internationalization is the problem of the service author, not the provider.”

So now we know what Web services are. What about internationalization?

Since a Web service is just a method invoked over the Web, the problems of Web services are very much the same as you would encounter with any remote invocation or distributed system. Web services do all sorts of things, from the trivial to the complex. They can be chained together into transactions or used to provide integration points between different pieces of software.

At first glance, Web services are pretty international to start with: they use a locale-neutral data representation (XML Schema) and they are involved in passing objects in XML documents between functions in software. They don’t have a direct user interface—humans aren’t really reading the SOAP documents directly.

So the first reaction most Web services folks have is that internationalization of a Web service is the problem of the actual service (method) writer and that Web services neither inhibits nor especially encourages any particular good or bad internationalization behavior.
Hogwash!

- Web service providers are the operating environment of the underlying service.
- Questions are familiar:
  - How do I build a locale affected service?
  - How do I get the user’s locale preferences?
  - How do I build internationalized Web services?
- Answer:
  - I have to write it all myself...

The argument that internationalization is not the problem of the Web service provider seems perfectly reasonable and logical. But it is still wrong.

Web services are supposed expose APIs in a platform and operating environment neutral way so that any compliant requester can invoke a service successfully, given the Web service description. Experience with app server technologies (such as .NET or J2EE) have demonstrated the need for locale models, language or content negotiation, and other infrastructural support for international operation. Since Web service technologies do not provide any of this capability in the base standards, creating an internationalized Web service means building and defining your own model.

This is a recipe for non-interoperable solutions or just outright non-internationalized services.
Web Services and Internationalization

Demo: webMethods Glue

- Web services providers generally don’t provide an internationalization infrastructure. Developers must write all of the supporting code, data structures, and logic for internationalization.

I’m going to demonstrate typical Web service internationalization as it exists today.

webMethods Glue is a Web service provider engine and is fairly typical as a Web service provider, in that it does nothing egregiously wrong with regard to internationalization. It has users all over the world who are using it successfully.

I’m going to write a Web service and progressively internationalize it during this presentation. As you’ll see, the Glue server allows me to do all kinds of useful things: I can name the service anything, I can pass all kinds of locale-affected data in or out successfully, and I can actually build a service that is completely internationalized. But look at all the code I had to write to do it!

A little later we’ll see a different webMethods product do the same thing with no lines of code.
The W3C Internationalization Working Group has been active in the area of Web services. The Web Services Task Force (WSTF) has completed both a Usage Scenarios document to illustrate the issues we’ve found and a set of Requirements for Web services technologies to address these problems.

The W3C I18N WG is currently re-chartering to work on developing standards to address some of these requirements on the W3C Recommendation track.
Issues We Found

- Identifying preferences: what is a locale, anyway?
  - Locales
  - Collation identifiers
  - Timezones
  - Other international preferences

- Identifying what a service does:
  - What locale will actually be used for the processing?

- Communicating preferences:
  - Language negotiation
  - Locale negotiation

- Policies and Patterns

Let’s look at the requirements we uncovered.

We need to be able to identify locales and international preferences in a platform neutral, consistent manner. Although de facto standards exist, rules and guidelines are really needed to standardize the interchange for all Web interactions.

Web service descriptions can then use these identifiers to describe how a particular instance of a service will run. Most services will probably be locale neutral, but locale-affected services need to be able to indicate what results to expect. This allows the client to select the endpoint (binding) that with the right language, collation, or other locale-affected attributes. WSTF calls this the “locale policy” of the service.

Services that are locale affected may also need the provider to do locale and language negotiation with the client. There needs to be a way for a SOAP document to carry these preferences to the provider. We’ll look in a minute at why the provider needs this information.

Finally, there needs to be a way for services to express what locale actually was used for a particular process in a response.
There is a de facto standard for locale identifiers on the Web. Most systems use the RFC 3066 (formerly RFC 1766) language tags as a proxy for locale identifiers. The similarity between RFC 3066 and POSIX-style locale identifiers is difficult to ignore. As a result, these tags, which were \textit{explicitly} meant for language identification only have become the standard for locale interchange.

They have some problems. Chief among them is that there are no rules for interpreting the tags as locales. In addition, there are some gaps in what can be conveyed with the existing tags.
What does that service do?

- Services generally run in the locale of the server where they are installed
  - May not be the same as the WS Provider
  - May not give the results the user expects
  - No way for the user to control it
- Developers must program services to provide international capabilities
  - Provide locale model
  - Provide localization model and capabilities
  - Define multiple endpoints for different locales
  - “Providers” do nothing for you.

As we saw in the Glue demo, services generally run in the locale of the operating system where they are installed. This locale may not be the same as that of the provider, the requester, or the service’s developer (since the same code might be distributed to many installations). The results you get now depend on something external to the service, the provider and the Web services infrastructure. And these results are not described by the Web service description.

Developers who overcome this issue, as noted, must do all the work to provide both design time and runtime differentiation of services. Web service descriptions might need to be set up for each different locale that is to be supported and different code provided to enable the service to provide this capability.
Web Services and Internationalization

Web Service Descriptions

- Exchange a locale that is explicitly in the service signature.
  - No standards exist for doing this
  - Strong platform and programming language dependency

- Exchange a locale that is implied in the service’s operation.
  - Web service descriptions don’t convey this information.

- Describe how a particular endpoint will work.
  - There may be multiple endpoints in multiple geographies.

WSDL provides the glue that holds the Web services world together. Using a WSDL document, a requester has all the information necessary to invoke the service. If we are to address the problems noted earlier, we need to be able to describe the locales and policies applied to a particular instance of a service.

One way to do this is to have different bindings for different locales. For example, the same method might be invoked from different URIs, allowing multiple instances to serve different locale settings. This is a convenient way to provide a small number of preconfigured services.

Another way to accomplish this is to provide a field in the service’s parameter list that can be used by code within the service to create a locale object. Since this is tied to the locale model for the service’s operating environment, programming language, and other information that really should be hidden inside from the requester, this may require extensive coding to convert (for example) and xsd:language field to a reasonable locale object in the service provider.

In some other cases, the service does not necessarily have control over the locale. For example, a database might return result sets in a particular collation sequence. A Web service that retrieves these values might not control the collation sequence used. The locale is implied in how the service actually works, but it isn’t something in the code of the service—it is more of a configuration time option.

Finally, intermediary providers might be used to do routing, sending requests to the “right” server, based on information in the request or based on other preferences in the SOAP document. One thing the intermediary might need is information on the capabilities or settings of a particular endpoint available.
Invocation

- Language negotiation
  - Services still need human readable messages.
  - Faults (exceptions) need human readable messages.
  - Service may retrieve, process, store, or otherwise access text.

- Locale negotiation
  - Making the service do what the user wants.
    - Collation, calendar, text processing, currency, routing, addressing, formatting, business rules, tax authority, legal requirements, etc.

Another point to address is why machine-to-machine processes need to be locale-affected at all. A purist might say that a design that is locale-affected is a poor choice, since pure machine-to-machine processes should be locale-neutral to the extent possible.

Indeed, most transactions, including most business transactions, can be designed to be locale-neutral. The use of Unicode in XML frees us from the “code page hell” that dominates older attempts at distributed systems. We don’t need to worry about losing characters. A locale-neutral process can receive, store, and process most kinds of character data and not need some of the meta data about language or code set previously necessary (since language was often used in legacy systems to approximate the encoding to use for data).

But “most” processes does not translate to “all” processes. There are many uses of Web services that deal with data that ultimately is human readable. Some of the applications that use Web services and also process text include:

- Messaging Systems, including IM and email
- Portals
- Webapp backend provisioning

Human readable text is a necessary part of any complete system. While the Web service might be equally capable of transferring any language of data, the underlying method call needs to select text for the end user of the Web service, not its current runtime environment. Burdening the service developer with this task by forcing them to include language preference information in every parameter list seems a steep penalty, especially since most runtime environments (like J2EE or .NET) provide a way to set a thread or context locale.
International Policies: What are they?

- Three Patterns:
  - Locale Neutral
  - Service Determined
  - Client Influenced

When we considered the ways in which a Web service can be locale affected, we found three basic patterns:

Locale Neutral services do the same thing, regardless of locale. This is typical of many Web services and it doesn’t require additional infrastructure to achieve this level of internationalization. A simple example of such a service would be “addInts”, which adds two integers together.

Service Determined services run in a specific locale, usually determined by the environment where the service is running or something inherent to the design of the service. This is the current state of affairs for most service providers: the service runs in the default locale where it is executing. But this can also include services that are programmed to do a specific task in a specific way. For example, a service that spell checks a document using a U.S. English dictionary is service determined.

Client Influenced is the final pattern. In this pattern, the service tries to meet the end user’s preference or preferences. This is typical of mutli-lingual or multi-locale (globalized) software. We called this pattern “client influenced”, since the service may not have resources for every possible value available to it. For example, there may different end-points for specific locales or there might be something in the configuration of the service that limits its locale setting capabilities.
webMethods Developer Demo

- Each service has a locale policy which has scope.
- Provides each of the patterns.
- (But no way to interoperate these capabilities)

Here is another webMethods product that implements all three patterns by providing a “locale policy” for each service. A quick demonstration shows that the services provide a concrete implementation that does everything we’ve been talking about.

The problem here is that, since there are no standards, I can’t expect a non-webMethods client to invoke the services correctly. We need a standard that describes these policies as SOAP headers and ensures that Web service requesters (clients) and providers are implemented in an interoperable way.

The development of these features will help ensure that this kind of interoperability is available in next generation Web services.
Requirements for Internationalization

- Standardized locale identifiers.
  - Possibly with standardized locale data?
- Standardized language and locale negotiation.
- Locale policy exchange (“quality of service”).
- Searching and discovery of services.

What’s needed to do this?

First, we need standardized locale and international preference identifiers which are platform neutral and have some level of interoperability in their own right. Standards in this area will ensure that systems have a mutually understood way of exchanging preferences. This applies to more than just Web services: it is a general need, long identified, for other Web applications.

For Web services in particular, there needs to be a set of SOAP and WSDL features that describe language and locale negotiation and the “quality of service” (policy, etc.) that apply to a particular service and its invocation. By making use of SOAP’s header capabilities, providers can be created that set the locale in the environment of well-written services, reducing or eliminating reprogramming of existing code to make it internationalized.

Finally, some more thought needs to be made about searching for and discovering services.
Who’s doing what...

- LDML and OpenLocale repository (locale data)
- IETF
  - RFC3066 replacement (draft-phillips-langtags-xx)
  - Collation identifiers (draft-newman-collation-xx)
- W3C I18N WG:
  - Locale identifier extension
  - SOAP I18N Feature
  - WSDL I18N Feature
- You:
  - What are you doing?

Some of the groups working in this area include:

Work on open locale data repositories.
Work on extensions to language and collation identifiers proceeding at the IETF.
The work I’ve described here today related to the W3C I18N WG.

This is a good time to become involved in one or more of these activities, which offer the hope of greater globalization for Web services and the Internet in general.
Summary

- 2004 will be a critical year in the development of internationalized Web services and systems.
- W3C, IETF, and other’s activities may result in powerful, open, standardized international capabilities.
  
  Or

- Too many implementations are created and too much time goes by and we spend our lives writing tutorials.

In summary, 2004 is a critical year for developing internationalization capabilities for Web services. Standards organizations are active in this area, but need support from the internationalization community.
Q&A