WS-I18N: Why Aren’t Web Services Internationalized (yet)?

26th 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.
Are Web services internationalized?
What does “internationalization” mean for Web services?
If it is broken, how do we fix it?
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)
- ...at the end of a URI

- 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.
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.
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 block” components or disparate parts of an enterprise. These technologies have had limited success, generally restricted to large, centrally-managed implementations, because the solutions require a commitment to proprietary data formats, large scale planning, 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.
Web Services and Internationalization

- WS-* extensions provide two basic types of capability:
  - Quality-of-Service
  - Execution State

- Web Services are internationalized?
  - 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.”

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. Since a Web service is basically a method invoked over the Web, the problems of Web services are similar to those in any remote invocation or distributed process.

If you look at the WS-* stack of standards, you’ll see that the mostly concern two things:
- Quality of Service. Ensuring that services are reliable and transactional.
- Execution state. Ensuring that data (such as user identity or process state) associated with a service invocation is available to the service.

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. Internationalization isn’t traditional “quality of service” and most developers don’t think of internationalization as a “state”.
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 work a lot like method calls in an API. They provide compartmentalized logic that can be accessed in a generic manner—without having to know about the implementation details. This works in a platform and operating environment neutral way because providers have agreed to use XML Schema to transport the data (so data formats can be decoupled from their object representations in a particular platform). Although a service can end up being tied to a specific programming model, most business objects can be represented as data structures not reliant on a particular technology.

But internationalization is not typically about the data structures. Internationalization generally affects how processing takes place: we expect our systems to perform differently depending on the cultural or linguistic preferences of the end user. In traditional computing environments (Win32 API, MacOS, X Windows, etc.), the information about the end-user’s preferences is stored in the environment in a structure called a “locale”. (On some systems there is quite detailed information and customization available, of course).

Experience with app server and other distributed technologies (such as .NET or J2EE) have demonstrated the need for locale models, language or content negotiation, and other infrastructure on the Web. These technologies have developed proprietary schemes for negotiating and caching the user’s preferences for a session and providing programmers access to this information.

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 and getting requesters to send you the information you expect.

This is a recipe for non-interoperable solutions or just outright non-internationalized services.
The W3C’s Internationalization Working Group has a Task Force dedicated to studying this problem. Chartered in 2002, the group has produced two documents, which are now W3C Notes.

The **Web Services Internationalization Usage Scenarios** spells out how Web services and international usage interact. This lengthy document covers many of the items I’ll talk about today and is the source for:

The **Web Services Internationalization Requirements** document is a Note spelling out the specific things that the Task Force felt that the W3C or its members should undertake in the area of Web service internationalization.

A key third document is the *draft W3C Internationalization Core Working Group Charter*, which renews this groups activities and adds specific “work items” (new deliverables from the Working Group) related to Web services, directly or indirectly.
The task force considered the problem of creating internationalized Web services: what is required to do this today and how should it work? In considering these issues, the group came to four basic conclusions.

- Web services need a locale model. Other operating environments provide developers with the infrastructure to manage user’s international preferences and to provide access to underlying functionality. Web services are no different.
- Web service descriptions need to describe services accurately. This includes internationalization issues.
- Locale identifiers are needed. For the first two items to work, there has to be a way to provide information about a user’s international preferences and this mechanism should be platform neutral and standardized.
- Finally, Web service discovery mechanisms need to be smarter about handling language and describing a service’s capabilities.
International Preferences

- Existing Web technologies use proprietary locale models
  - J2EE
  - .NET
  - Etc.

- Web services need locale identifiers that are:
  - Open
  - Platform neutral
  - “Consistent enough” results

Existing Web technologies are generally proprietary in nature and use proprietary mechanisms to deal with the internationalization problem. This works because generally the systems are two-tiered, with a client and a server, and most (if not all) of the logic is on the server side.

With Web services, the requester and the service (and sometimes the provider) must be able to specify the settings from their proprietary locale model to systems that may have a maximally different locale model.

This requires identifiers that are open, shared, platform-neutral, and produce “consistent-enough” results.
Locale Model for the World Wide Web?

- This does NOT require that operating environments, languages, etc. share locale data or a locale model
  - Identifiers solve 9/10 of the problem
  - Services that need finer grained control should expose this in their service signatures
- This does not mean that (for example) CLDR is a bad idea. It just isn’t required to achieve Web service interoperability.

If locale identifiers are needed, does that mean operating environments, programming languages, service providers, and so forth need to adopt the same locale model?

I don’t believe it does. Locale identifiers can generally be harmonized, identifying locales across platforms with a remarkable degree of consistency. Where results diverge is where specific locale models are richer or poorer by comparison. But the overall results are identifiably correct.

Services that need more fine grained control may rely on either proprietary extensions that extend their provider’s native environment. For example (and not to pick on Microsoft) I can imagine that a .NET extension to WS-International would provide the requester’s specific Regional Options control panel settings to the provider (which, if it were a .NET-based provider, could set the service’s thread locale appropriately). By the same token, invoking a service written in Java would get only marginally different results, given that most users do not personalize their settings that much.

That said, a service written to produce a highly tailored result that relies on something as ephemeral as a specific date format, probably should expose that control in the service signature (the parameter list for invoking the service) and not rely on the presence of a header or extension. When details matter to the service, they are exposed at that level and don’t require contextual help.
Locale Identifier Sources

- RFC 3066 (bis) provides the de facto standard for this today. RFC 3066 provides **language tags** which are emphatically not supposed to be locale identifiers...
  - Abused in Web services already
  - Abused by Java and .NET
  - Abused by most programmers
  - Supported through firewalls (HTTP), in XML, in XML Schema, and so forth
- Ideally we want both language tag based identifiers and URI based identifiers.

The IETF standard RFC 3066 (which replaced RFC 1766, a reference you may still see in various places) and its proposed successor (draft-langtags-phillips-04*, colloquially called RFC 3066bis) define tags for identifying the language of content. These tags are in the familiar “en-US” (language-country) form. This standard has historically been used to infer the locale from HTTP headers or XML content, even though that is frowned upon by the language tagging community.

Ideally we would like to see URI based identifiers and it isn’t difficult to envision a bridge between the two.

So what do we need these identifiers for? Why are they required?

* Yes, the author of this presentation is the author of the Internet-Draft with Mark Davis of IBM. See http://www.inter-locale.com for links to the HTML version of the draft.
Choreography and Orchestration

- Web services are becoming a programming model unto themselves.
  - Choreographing services into a service
  - Orchestrating services to form business processes
- The distributed nature of Web services allows organizations to tap their existing infrastructure in powerful ways.

The lack of a locale model becomes more obvious and more problematic when one considers the growth of Web services technologies as the leading method of creating integrated business logic in the enterprise. Where previously integration required large scale solutions, Web services allow an organization to tap its existing resources in a consistent manner with low overhead.

The simplest form of doing this is called “Choreography” and amounts to tying a few services together to perform a specific task. On a larger scale, “Orchestration” allows services to be tied together to form complete business processes, with all the hallmarks of traditional, transaction-oriented, managed business processes.

When multiple services are tied together in this way, the ability to model the linguistic, regional, cultural, and legal differences between different markets becomes critical. In a connected enterprise, there may be thousands of nodes available for a given task (so discovery management is important), which may hold differing capabilities (which should be described in the WSDL), and which may be filling client requests in a variety of languages and regions (runtime negotiation in SOAP).
When the Task Force considered the ways in which a Web service can be locale affected, they found four basic patterns:

1. **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.

2. **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.

3. **Client Influenced** is the third pattern. In this pattern, the service tries to meet the end user’s preference or preferences. This is typical of multi-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.

4. **Data or Resource Driven** is the final pattern. In this pattern, the back-end resource’s configuration, set-up, or capabilities determine the outcome of a particular transaction.
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.

Services generally run in the locale that their host environment is running in. If they run in the same process or thread as the provider, then this may be the provider’s locale. But note that 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), since the service can be hosted anywhere. Add intermediaries to the equation and the locale of the service can be anything. The results the requester receives 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 need to 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. And the service has to expose this (today) in the service’s signature. This is tedious and doesn’t provide for interoperability, since the temptation is to expose your native locale identifiers or to collect this information naïvely.

If an internationalization infrastructure, such as the one demonstrated in this presentation, is to emerge, it requires general agreement on the format and content of the metadata being exchanged. And services must be able to describe the policy necessary to either set the service’s performance or the way in which they are configured to run (so that users can select the correct endpoint or correctly configured service).
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.
Who’s doing what...

- **W3C I18N WG:**
  - New Charter includes “WS-International”, “Locale Identifiers”, and “Collation Identifiers”
  - Need members to join and participate
  - Need industry support for WS-International functionality.

- **You:**
  - What are you doing?

Some of the groups working in this area include:

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 is 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.