Towards the Internationalization of Web Services in IBM WebSphere®

On the Development of Internationalized Web Service Applications in Distributed Heterogeneous Client-Server Environments

Debasish Banerjee, WebSphere Internationalization Architect, IBM Rochester, MN, USA
Casey A. Swenson, WebSphere Developer, IBM Rochester, MN, USA

Abstract. *Web services are rapidly emerging as the basis for the next generation of the eCommerce infrastructure, and SOAP is considered the de facto protocol for the invocation of Web services. In all the existing Web services implementations, a Web service provider may freely impose its own locale and time zone on all the locale- and time zone-sensitive computation requests from Web service requestors.*

The internationalization service in IBM WebSphere transparently propagates the 'internationalization context' consisting of the locale and time zone information. This provides a unique infrastructure for distributed internationalization in Java™ 2 Enterprise Edition (J2EE) environments. Typically, Web services components are implemented as wrappers over existing server-side business components.

The present paper proposes an architectural extension to Web services for distributed internationalization, where the wrapped business components use internationalization service for localization. The implementation is built on top of the existing internationalization service, and uses Axis, the Apache open group's emerging implementation of Web services, for transparently injecting internationalization contexts to and extracting the same from SOAP messages. The interaction with the backbone internationalization service is discussed in detail. The use of the proposed Web service infrastructure is illustrated with appropriate examples. The ongoing and future internationalization work in the domain of Web services is also highlighted.

1. Introduction Web services are rapidly emerging as the basis for the next generation of the collaborative eCommerce infrastructure, and SOAP [23] is considered to be the *de facto* protocol for invocation of Web services. Typically, Web services act as wrappers over server-side components, which allow clients to dynamically discover and use the desired services using RPC-style invocations. The server-side components can be developed using any established client-server technology, but their actual implementation mechanisms are abstracted out in their published WSDL [25] definitions, which should be available in globally accessible locations, like UDDI registries [22].

Discussions about internationalization in the domain of Internet and Web services are being vigorously pursued [9, 17, 24]. Barring a few exceptions [3, 15], the majority of the internationalization discussions and publications concentrate on XML document representation aspects, like character sets, proper creation and rendering of multilingual XML documents, the proper use of Unicode, multilingual names, etc. None of the Internet internationalization forums has architecturally addressed the rather fundamental issue of localization in distributed Internet environments. The present paper attempts to address the issue of distributed internationalization in the domain of Web services.

Section 2 points out the fundamental localization issues in Web services environments. As the background information, Section 3 briefly mentions the internationalization service [2, 8] available in IBM WebSphere, Enterprise Edition. Section 4 introduces the architectural extension to the Web services for distributed

internationalization. Section 4 also discusses IBM's implementation of Web services internationalization, which uses Axis [1], the Apache open group's emerging implementation of Web services, and is built on the top of the existing internationalization service. Section 5 discusses the programming model and interoperability issues. Section 6 illustrates Web services internationalization with appropriate examples. Finally, Section 7 draws the conclusion.

2. Distributed Internationalization Apart from the already well-explored issues of code set and endian mismatches [2], in real life situations, there can be locale and time zone mismatches between a Web service requester (client) and a Web service provider (server). In the existing Web service architectures and implementations, a service provider freely imposes its locale and time zone in all the locale- and time zone-sensitive operations requested by a client.

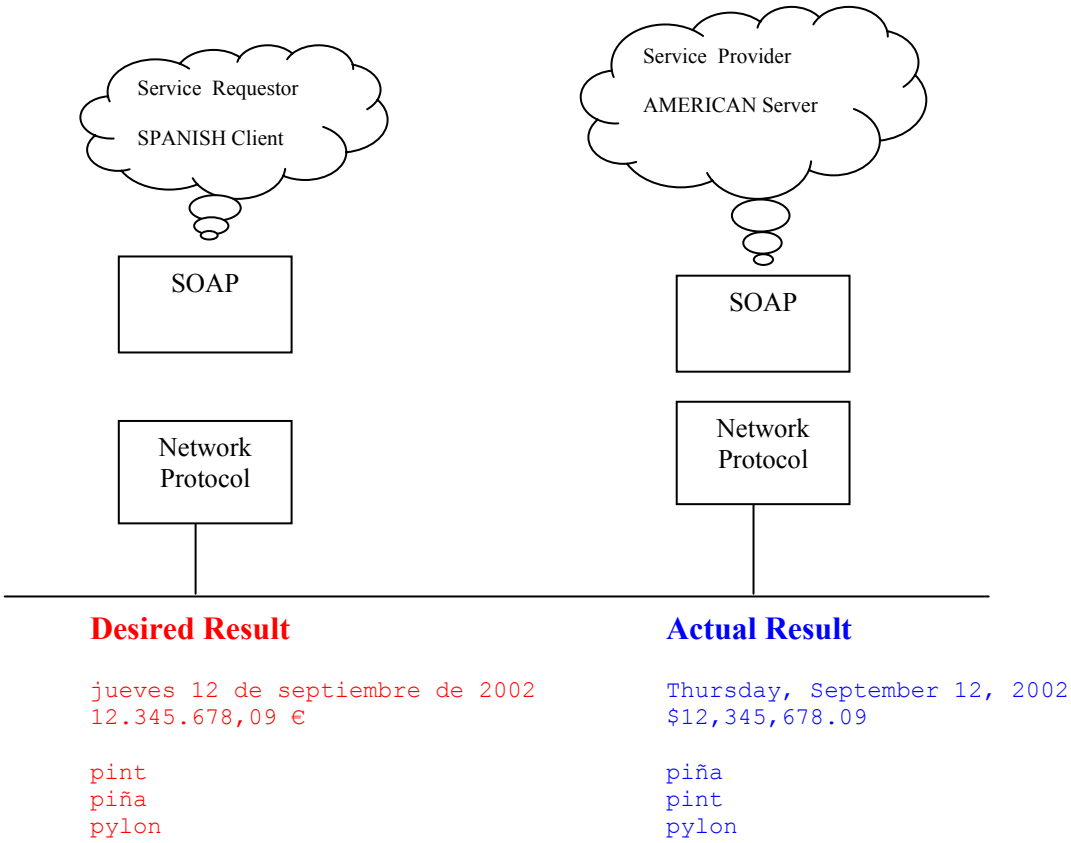

Figure 1

Figure 1 depicts a locale mismatch scenario between a Spanish requester and an American service provider implementing a Web service activity returning a formatted date, a formatted currency and a collated list. The existing Web services implementations will typically return all the results in American locale. Ideally, following the rule of 'local-remote transparency' in distributed environments, the client should expect the locale-sensitive results in it's (Spanish) locale and not in the service provider's (American) locale.

A Web service provider developed using modern programming languages and libraries [6, 13, 18] can easily localize relevant business computations in requester's locale, if the requester's locale information is available during actual computations. As an outwardly simple solution, service requestors can pass extra parameters carrying the locale information to locale-sensitive Web service methods. However, this approach is intrusive and error-prone. In a call chain of execution, even if the final method is locale-sensitive, the 'locale' parameter has to be passed all the way starting from the beginning of the call chain. Again, the business methods can generate faults containing text messages. Requestors should receive the

text messages formatted in their locales and not the service provider's locale. For generating localized fault messages, the 'locale' parameter needs to be added to all the Web service methods. This simple parameter addition approach also suffers from another serious disadvantage. The interfaces and hence the WSDL descriptions of most of the existing Web service applications can not easily be changed—any interface change necessitates a complex re-deployment.

Again, in modern eCommerce environments, the requester and the provider machines can be located across different time zones resulting in time zone mismatches. Some business method computations can be time zone-sensitive in nature. Financial transactions containing the requester's timestamp can be cited as an example. Quite similar to the case of locales, the present Web service providers impose their time zone on the client requested time zone-sensitive computations.

To avoid the time zone mismatch problem, some traditional applications always use GMT time zone. While GMT times may be suitable for machines, some users may consider it artificial for human interpretation. Application developers can also pass extra parameters carrying the requestor's time zone information to all the time zone-sensitive Web service methods. The parameter passing approach suffers from the same disadvantages as mentioned earlier in connection with locales.

3. Internationalization Service [2] introduced the concepts of 'internationalization context' consisting of an ordered chain of locales and a time zone, and 'internationalization service' for propagating and managing internationalization contexts in remote invocations. The locale chain of an internationalization context is ordered according to the requestor's preference. Inside any J2EE component, two categories of internationalization contexts are made available by the internationalization service: the caller internationalization context representing the requestor's internationalization information and the invocation internationalization context representing the internationalization environment under which a business method should execute.

The internationalization service works by associating the caller and invocation internationalization contexts with every thread of executions in an application. During a remote invocation, the internationalization service uses the J2EE Activity Service [16] and the underlying ORB to transparently intercept and propagate the invocation internationalization context along with the remote method invocation. On the receiving side, the internationalization service again, work harmoniously with the activity service and ORB, extracts the propagated internationalization context and attaches it to the thread of execution as the caller internationalization context. By default, for the server-side J2EE components, the internationalization service sets the invocation internationalization context on the receiving thread identical to the inbound caller internationalization context. However using proper deployment descriptors [4], the invocation internationalization contexts can be set to values which can potentially be different from the inbound caller internationalization contexts.

By means of a simple Java Naming and Directory Interface™ (JNDI) lookup, a J2EE programmer can obtain a reference to the `UserInternationalization` object containing references to both the internationalization contexts. The internationalization service provides a few APIs. An application can access both the categories of internationalization contexts, and the locales and time zones associated with the contexts using these APIs. The server side J2EE components can use the APIs to access the invocation locales and time zones, and use them for properly localizing the relevant locale and time zone-sensitive business computations. For a more complete description of internationalization service including its APIs and the suggested programming model the reader is referred to [2].

The present implementation of internationalization service in IBM WebSphere, Enterprise Edition is for J2EE environments. Internationalization service can be easily implemented in other managed environments like CORBA Component Model [14]. The present paper extends the applicability on internationalization service to the domain of Web services.

4. Internationalizing Web Services The proposed solution for the Web services internationalization is similar in spirit to the internationalization service—a Web service requester's internationalization information should be transparently extracted, propagated, and made available to the Web service providers. The existence of internationalization service for the underlying managed components implementing the WSDL `portType` definitions is assumed. Also, SOAP is assumed to be the XML-messaging protocol for all Web service invocations.

The Web services internationalization scheme can be divided into three logically separate strategies:

- Define internationalization contexts for SOAP requests.
- Propagate the requester's internationalization contexts in SOAP requests.
- Design and implement internationalization handler for interfacing between the Web service requesters, Web service providers, and the underlying internationalization service.

The following contains discussions about all the above mentioned strategies.

4.1 Internationalization Context The internationalization context defines an internationalization environment under which a business method may execute. It consists of a non-empty ordered chain of locales and a time zone identifier. The locales are spatially ordered according to the requester's preference. As an example, a requester may want to associate the following internationalization context along with a Web service invocation.

```
<InternationalizationContext>
      <Locales>
            <Locale>
                  <LanguageCode>ja</LanguageCode>
                  <CountryCode>JP</CountryCode>
            </Locale>
            <Locale>
                  <LanguageCode>fr</LanguageCode>
                  <CountryCode>FR</CountryCode>
            </Locale>
            <Locale>
                  <LanguageCode>en</LanguageCode>
                  <CountryCode>US</CountryCode>
            </Locale>
      </Locales>
      <TimeZoneID>JST</TimeZoneID>
</InternationalizationContext>
```

Figure 2

The internationalization context emanating from the requester is intended to convey the following information to the Web service provider.

4.1.1
```
if (Japanese resources are available)
      localize the requested computation in Japanese locale
else if (French resources are available)
      localize the requested computation in French locale
else
      try to localize the requested computation in American locale
```

4.1.2
```
Execute the requested computation under the Japanese Standard
Time (JST) time zone.
```

Appendix A contains a proposed XML schema for representing internationalization context in XML. The stringified representations of locale and time zone are modeled after Java™ [6] and ICU class library [7].

4. 2 Internationalization Context in SOAP The invocation internationalization context of a Web service requester is propagated from the requester to the Web service provider in a SOAP header block. The `mustUnderstand` SOAP attribute [23] is absent in the proposed header—non-internationalized Web service providers can simply ignore any propagated internationalization context in the SOAP headers without generating any fault. Figure 3 shows a SOAP header containing the sample internationalization context of Figure 2.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <SOAP-ENV:Header>
            <InternationalizationContext>
                  <Locales>
                        <Locale>
                              <LanguageCode>ja</LanguageCode>
                              <CountryCode>JP</CountryCode>
                        </Locale>
                        <Locale>
                              <LanguageCode>fr</LanguageCode>
                              <CountryCode>FR</CountryCode>
                        </Locale>
                        <Locale>
                              <LanguageCode>en</LanguageCode>
                              <CountryCode>US</CountryCode>
                        </Locale>
                  </Locales>
                  <TimeZoneID>JST</TimeZoneID>
            </InternationalizationContext>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
            <m:WebServiceMethod xmlns:m="Some-URI">
                  <parameter> … </parameter>
            </m:WebServiceMethod>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 3

4.3 Internationalization Handler IBM WebSphere uses the core J2EE internationalization service and Axis to provide the internationalization infrastructure for Web Services. The Axis framework provides a mechanism to transparently intercept inbound and outbound SOAP messages, and for pre- and post-processing of SOAP messages using pluggable handlers. Internationalization handlers installed on both the requestor and provider side of IBM Web services can access and modify the SOAP messages as well as the internal data structures of the underlying core internationalization service.

On the service requester side, the internationalization handler obtains the invocation internationalization context for the current thread of execution from internationalization service, creates the SOAP internationalization header from it, and attaches the newly created header to the outbound SOAP message. Similarly on the service provider side, the internationalization handler, on being invoked by Axis, looks for the internationalization header in the inbound SOAP message. If internationalization header is present, the handler uses the header information to create the caller internationalization context for the current thread of execution, and places the same in the internal data structure of the internationalization service. If the internationalization header is missing in the inbound SOAP message, the handler leaves the internationalization service's data structure intact, thereby creating a default caller internationalization context for the thread of execution. The default internationalization context comprises of the default locale and time zone of the underlying JVM.

It should be noted that the internationalization contexts are thread-scoped, and Axis ensures that a managed component wrapped as the Web service requestor or service provider and the corresponding handler run on

the same thread of execution. There are differences between the data types used to represent internationalization contexts by the internationalization service, and the SOAP internationalization header: internationalization service uses standard Java data types for locales and time zones, whereas SOAP internationalization header uses stringified XML representations of them as shown in Figure 3. The internationalization handler compensates the impedance mismatch between the internationalization service and SOAP header by performing the necessary data type transformations for both inbound and outbound SOAP messages.

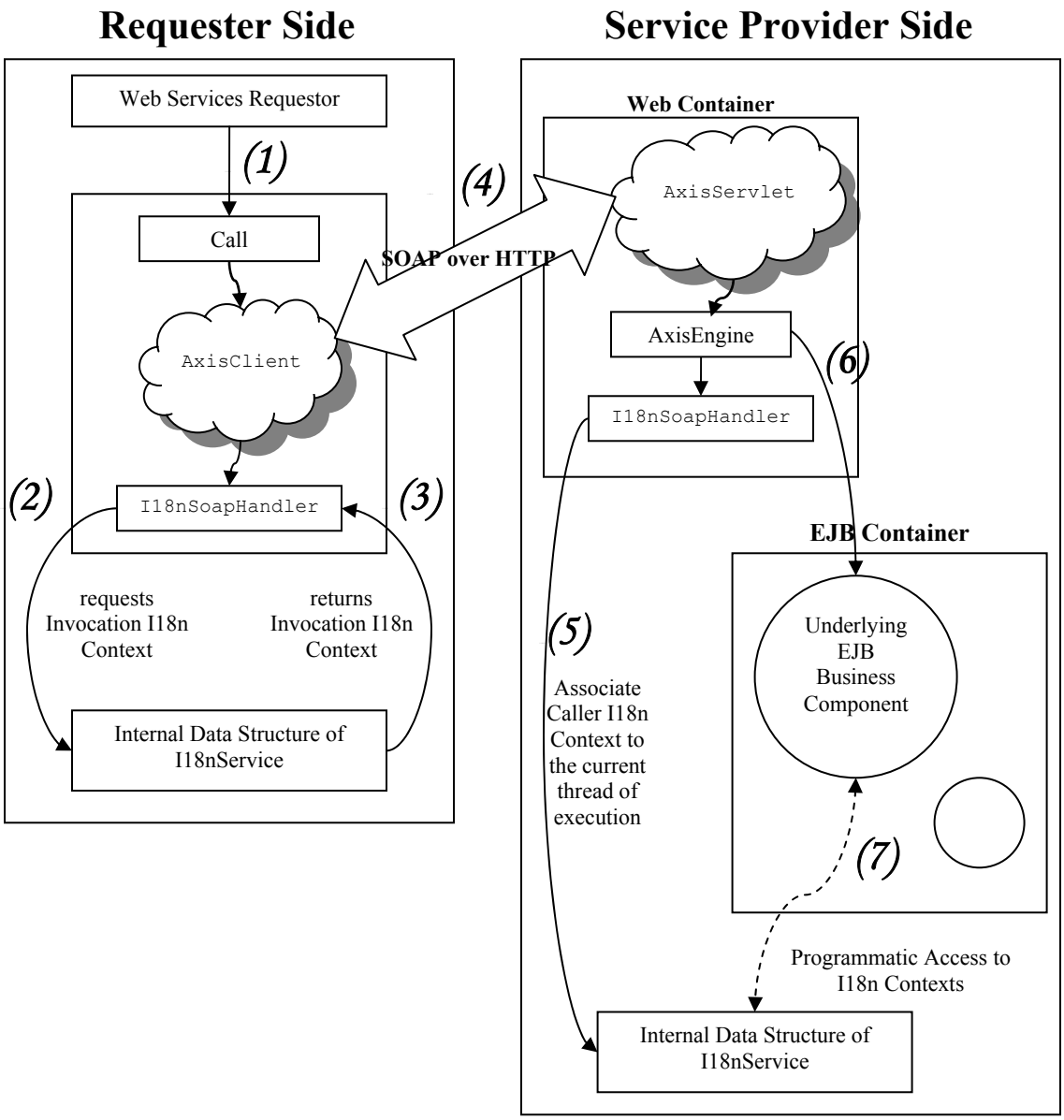

Figure 5

Figure 5 is a schematic diagram illustrating the interaction between SOAP internationalization handlers and the internationalization service, where `AxisClient` and `AxisServlet` represent the Axis objects for issuing and accepting Web service requests and responses. A Web service requestor creates a Web Service request at step *(1)*. `I18nSoapHandler`, the internationalization handler requests *(2)* and retrieves *(3)* the

invocation internationalization context for the current thread of execution. `I18nSoapHandler` creates the internationalization header from the retrieved context and attaches it to the outbound SOAP request. The SOAP request leaves the requestor site over HTTP *(4)*. On the service provider side, before the actual processing of the Web service request, the `I18nSoapHandler` creates an internationalization context from the SOAP internationalization header associated with the inbound request, and associates it *(4)* as the caller internationalization context to the current thread of execution. During the actual processing *(6)* of the service request, the wrapped J2EE component can access *(7)* the internationalization contexts using internationalization service APIs for performing locale- and time zone-sensitive computations.

5. Programming Model and Interoperability The present Web service internationalization scheme is built on the top of the internationalization service infrastructure, and hence it is applicable only for the J2EE components which can be wrapped as Web service requestors and providers and for which the support of internationalization service is assured by IBM WebSphere.

Internationalization service does not exist in applet containers or simple Web browser environments [2], and hence neither an applet nor an HTTP Web browser client can be wrapped as a Web service requester supporting the feature of automatic propagation of internationalization contexts in SOAP messages. Since the internationalization service is supported in J2EE application client container, Web container and EJB container, a J2EE application client, a servlet, a JavaServer Page™ (JSP), or an Enterprise JavaBean™ (EJB) can be wrapped as Web service components supporting distributed internationalization.

IBM WebSphere does not provide direct programmatic access to the SOAP internationalization contexts and headers. As explained in Section 4.3, the SOAP internationalization contexts and headers are created automatically by the SOAP internationalization handlers. Now depending on the internationalization context management policies and attributes [4] associated with the relevant J2EE components, application programmers may have update access to the invocation internationalization contexts. Any programmatic alteration of the invocation internationalization context made before the issuance of a Web service request will be reflected in the SOAP internationalization header. For example, a J2EE application client always allows programmatic alteration of the invocation internationalization context. For a J2EE application client wrapped as a Web service requester, a programmer can set the desired invocation internationalization context and hence the SOAP internationalization header. In the absence of any programmatic setting, the above mentioned Web service requester will propagate a default invocation internationalization comprising of the client JVM's default locale and time zone. Although by means of appropriate deployment, an application programmer can be granted update access, the more common and preferred default deployment of the server-side J2EE components permits a read-only access to the invocation internationalization contexts. The default deployment of the server-side J2EE components also sets the invocation internationalization contexts identical to the caller internationalization contexts. Hence, by default, in a call chain of Web service invocations, where internationalization service is available for all the underlying components wrapped as Web services, the invocation internationalization context will be identical in all the components and its value is set by the very first caller of the invocation chain.

The present Web service internationalization scheme provides easy interoperability with non-IBM Web service requesters and providers like Microsoft™ .NET [12], or TME's GLUE [21]. A non-IBM Web service requester can easily attach SOAP internationalization headers programmatically to the Web service requests directed towards an IBM WebSphere Web service provider. If the programmatically attached header is created according to the XML schema shown in Figure A.1 of Appendix A.1, the IBM Web service provider will utilize the header for properly localizing relevant business computations. On the other hand, if an IBM Web service requester transmits a SOAP request containing internationalization header to a non-IBM Web service provider, the service provider will simply ignore the transmitted header, since as mentioned in Section 4.2, the SOAP internationalization header does not contain the `mustUnderstand` attribute. The non-IBM service provider most likely will use its own locale and time zone to perform relevant locale- and time zone-sensitive computations, thereby maintaining status quo regarding distributed internationalization.

6. Examples This section presents a few code snippets to illustrate the hypothetical Web service internationalization scenario of Figure 1, where a Web service requester sends a SOAP request to a service for obtaining formatted versions of the current date, some currency value and a collated list.

`MyWebServiceRequester`, the Spanish J2EE application client of Figure 7 located in Pacific Standard Time (PST) time zone acts as a Web service requester and sends the SOAP request of Figure 8. It should be noted that in Figure 8 the SOAP internationalization handler working harmoniously with internationalization service has placed the default invocation locale and time zone of the requester as the internationalization context of the SOAP request.

```
// the necessary imports

public class MyWebServiceRequester
{
   public static void main(String[] args)
   {
      try
      {
         String endpoint =
                        "http://localhost:9080/axis/services/urn:MyWebService";
         Service service = new Service();
         Call call = (Call) service.createCall();
         call.setTargetEndpointAddress( new java.net.URL(endpoint) );
         call.setOperationName(new QName("http://soapinterop.org/",
                              "localize"));
         MyWebServiceResult result =
                        (MyWebServiceResult) call.invoke( new Object[] { } );
      } catch (Exception exception) {/* … */ }
   }
}
```

Figure 7

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <SOAP-ENV:Header>
            <InternationalizationContext>
                  <Locales>
                        <Locale>
                              <LanguageCode>es</LanguageCode>
                              <CountryCode>ES</CountryCode>
                        </Locale>
                  </Locales>
                  <TimeZoneID>PST</TimeZoneID>
            </InternationalizationContext>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
            <m:localize xmlns:m="Some-URI"/>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 8

Figure 9 is the code snippet of a session bean implementation of the requested Web service. For simplicity, some of the values are shown as hard coded. Internationalization service APIs are used to obtain the invocation locale.

```
// the necessary imports

public class MyWebService implements javax.ejb.SessionBean
{
   public MyWebServiceResult localize()
   {
      // obtain the initial context for JNDI lookup in IBM WebSphere
      InitialContext initialContext = null;
      try
      {  initialContext = new InitialContext();   }
      catch (NamingException namingException) { /* . . . */ }

      // obtain the a reference to the invocation locale
      try
      {
         userInternationalization = (UserInternationalization)
        initialContext.lookup("java:comp/webspehere/UserInternationalization");
      }
      catch (NamingException namingException) { /* . . . */ }

      invocationLocale = userInternationalization.
                              getInvocationInternationalization().getLocale();

      // compute and return the result to the requester
      MyWebServiceResult myWebServiceResult = new MyWebServiceResultType(
                                                    collate(list)
                                                  , toDaysDate()
                                        , formatCurrency(currencyValue) );
      return(myWebServiceResult);
   }

   private String[] collate(String[] list)
   {
      Arrays.sort(list, Collator.getInstance(invocationLocale));
      return list;
   }

   private String toDaysDate()
   {
      DateFormat dateFormat =
               DateFormat.getDateInstance(DateFormat.FULL, invocationLocale);
      return(dateFormat.format(new Date()));
   }

   private String formatCurrency(double currencyValue)
   {
      NumberFormat numberFormat =
                     NumberFormat.getCurrencyInstance(invocationLocale);
      return(numberFormat.format(currencyValue));

   }

   // ... other EJB methods
   ...
   UserInternationalization userInternationalization = null;
   InvocationInternationalization invocationInternationalization = null;
   InvocationLocale invocationLocale = null;

   String[] list = { "pylon", "piña", "pint" };
   double currencyValue = 12345678.09;
}
```

Figure 9

Figure 10 depicts a possible SOAP response generated by the Web service provider of Figure 9. The element `MyWebServiceResult` used in Figure 10 is an XML representation of the Java type `MyWebServiceResult` of Figure 9.

```
<SOAP-ENV:Envelope xmlns:xsd=http://www.w3.org/2001/XMLSchema"
                   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:SOAP-ENC="http://www.w3.org/2001/12/soap-encoding"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <SOAP-ENV:Body>
            <m:MyWebServiceResponse xmlns:m="Some-URI">
                  <MyWebServiceResult>
                        <date xsi:type=”xsd:string”>
                              jueves 12 de septiembre de 2002
                        </date>
                        <number xsi:type=”xsd:double”>
                              12.345.678,09 €
                        </number>
                        <words SOAP-ENC:arrayType="xsd:string[3]">
                              <word>pint</word>
                              <word>piña</word>
                              <word>pylon</word>
                        </words>
                  <MyWebServiceResult>
            </m:MyWebServiceResponse>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 10

7. Conclusions The internationalization infrastructure for Web services is available as a technical preview in IBM WebSphere, Enterprise Edition, release 5.0. The approach presented here should be considered as just the starting point for the complete internationalization of Web services. The paper extended the definition of internationalization context to XML and addressed the fundamental issue of the propagation of the requester’s internationalization information in all SOAP requests by defining a SOAP internationalization header. The orchestration of the internationalization handlers with the existing internationalization service enables a Web service requester to transparently propagate its invocation internationalization context to the Web service provider using which the Web service provider can localize relevant computations. Though the paper considered SOAP as the XML messaging protocol, in principle, the concepts of the paper are equally applicable to other messaging protocols, like HTTP, MIME as well. For non-SOAP protocols, one has to define a representation of internationalization context, and appropriate header(s) for carrying the internationalization contexts in remote invocations. The internationalization handlers for non-SOAP protocols have to provide the necessary data type transformations for communicating the internationalization context to and accessing the same from the underlying internationalization service.

In a recent document [15], Phillips also noted the absence of any architectural solution for distributed internationalization for Web services. In addition to the interesting notion of ULocale, [15] also contains the notion of internationalization context passed in SOAP envelope during remote invocations, and APIs somewhat similar in spirit to those of the internationalization service. However, the formalization of internationalization context and its propagation mechanism is not very clearly described in [15].

The notion of internationalization or any other context is missing in the present version of WSDL specifications [25]. W3C working group has already defined a requirement for supporting contexts in WSDL. Once this new requirement is addressed in an upcoming version of WSDL specifications, it will be possible to attach internationalization contexts in appropriate places in the WSDL descriptions of Web services.

A managed component or method can have internationalization context management policies and attributes [4] associated with them. Usually these management policies and attributes statically convey the internationalization capabilities and runtime behaviour of the managed components or methods. The managed components or methods retain their internationalization capabilities when wrapped as Web services. The internationalization context management policies and attributes can be considered 'qualities of services' (QOS) of the relevant Web service `portTypes` and `operations`, and should be available in UDDI registries. Web service requesters will then be able to scrutinize the internationalization capabilities of various service providers offering similar services before deciding to bind to specific providers.

The notion of internationalization context should also be incorporated in the real life business flow models, both for Web services [11, 20] and for other work flow domains. As part of its overall effort, [5] intends to formalize the interaction between J2EE components and Web services [10] regarding internationalization context and internationalization service.

Acknowledgements
David Zavala of IBM, Rochester, MN participated in many stimulating discussions, and provided the implementation of the core internationalization service in IBM WebSphere, Enterprise Edition. Martin Dürst of Keio University, Japan helped us in defining the XML schema. Betsy Baartman of IBM, Rochester, MN provided numerous suggestions for improving the quality of presentation.

References

1. Apache Group. Axis User's Guide, Oct. 2001, http://xml.apache.org/axis.
2. Banerjee D., Frey J. A., and High R. H., Jr., The Internationalization Service in IBM WebSphere: On the Development of Internationalized Applications in Distributed Heterogenous Client-Server Environments. 20th International Unicode Conference, Washington, DC, Jan./Feb.2002.
3. Banerjee D. Towards the Internationalization of Web Services. Position statement at W3C Internationalization Workshop, Washington D.C., Feb. 2002. http://www.w3.org/2002/02/01-i18n-workshop/Banerjee.html.
4. Banerjee D., et. al. On the Formal Management Policies of Internationalization Context. In preparation.
5. Banerjee D. Internationalization Service For J2EE, JSR 150. In preparation. http://jcp.org/jsr/details/150.jsp.
6. Chan P., Lee R., and Kramer D. *The Java Class Libraries, Second Edition, Volume 1*, Addison-Wesley, Reading, Mass., 1998.
7. IBM Corporation. International Components for Unicode, Version 2.1, Apr. 2002. http://oss.software.ibm.com/icu/download/2.1/.
8. IBM Corporation. *WebSphere Application Server Enterprise Edition 4.0: A Programmer's Guide*, RedBook SG24-6504-00, IBM Corporation, Armonk, NY, Feb. 2002. http://www.redbooks.ibm.com/.
9. ITU and WIPO. Multilingual Domain Names: Joint ITU/WIPO Symposium, Geneva, Dec. 2001, http://www.itu.int/mdns/presentations/index.html.
10. Knutson, J. , Kreger, H. *Web Services for J2EE, Version 1.0, Public Draft v0.3*, JSR 109, Apr. 2002. http://jcp.og/jsr/detail/109.jsp.
11. Leymann F. Web Services Flow Language (WSFL 1.0). IBM Corporation, May 2001. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
12. Microsoft Corporartion. http://www.microsoft.com/net.
13. Myers, N. C. The Standard C++ Locale. http://www.cantrip.org/locale.html.
14. OMG. CORBA Components Volumes 1, 2, and 3, Jul. 1999. http://www.omg.org/cgi-bin/doc?orbos/99-07-01, http://www.omg.org/cgi-bin/doc?orbos/99-07-02, and http://www.omg.org/cgi-bin/doc?orbos/99-07-03.
15. Phillips, A. P. ULocales: Towards a Locale Model for the Internet. webMethods Inc. CA, May 2002. http://www.inter-locale.com/whitepaper/ULocales.pdf.
16. Robinson, I. J2EE Activity Service Specification, JSR 095. http://jcp.org/jsr/detail/95.jsp.
17. Savourel Y. *XML Internationalization and Localization*. SAMS, Indianapolis, Indiana, 2001.

18. Schmitt, D. A. *International Programming for Microsoft Windows*, Microsoft Press, Redmond, WA, 2000.
19. Sun Microsystems. http://java.sun.com/j2se/1.4/docs/api/java/util/TimeZone.html 2002.
20. Thate, S. XLANG: Web Services for Business Process Design. Microsoft Corporation, 2001. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.
21. The Mind Electric. http://www.themindelectric.com/glue/index.html.
22. UDDI Org. UDDI Version 2.0 Data Structure Reference, Jun. 2001. http://uddi.org/pubs/DataStructure-V2.00-Open-20010608.doc, UDDI Version 2.0 API Specification, Jun. 2001. http://www.uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.pdf.
23. W3C Org. SOAP Version 1.2 Part 1: Messaging Framework, W3C Working Draft 17, Dec. 2001 http://www.w3.org/TR/SOAP12-part1, SOAP Version 1.2 Part 2: Adjuncts, W3C Working Draft 17, Dec. 2001. http://www.w3.org/TR/SOAP12-part2.
24. W3C Org. Character Model for the World Wide Web, W3C Working Draft, Sep. 2001. http://www.w3.org/TR/charmod.
25. W3C Org. Web Services Description Language (WSDL) 1.1, W3C Note, Mar. 2001. http://www.w3.org/TR/wsdl.

Appendix A Figure A.1 is the proposed XML schema for representing internationalization context. As mentioned earlier in Section 4.1, the XML representations of locale and time zone are modeled after Java and ICU class library.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">
      <xsd:annotation>
            <xsd:documentation>
                  <info>
                        The XML schema for the 'internationalization
                        context', a non-empty sequence of locales and a
                        time zone identifier
                  </info>
            </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType name="InternationalizationContextType"> <
            <xsd:sequence>
                  <xsd:element ref="Locales"/>
                  <xsd:element name="TimeZoneID" type="xsd:string"/>
            </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="Locale" type="LocaleType"/>
      <xsd:element name="Locales" type="LocalesType"/>
      <xsd:element name="LanguageCode" type="xsd:string"/>
      <xsd:element name="CountryCode" type="xsd:string"/>
      <xsd:element name="VariantCode" type="xsd:string"/>
      <xsd:complexType name="LocalesType">
            <xsd:sequence>
                  <xsd:element ref="Locale" maxOccurs="unbounded"/>
            </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="LocaleType">
            <xsd:choice>
                  <xsd:sequence>
                        <xsd:element ref="LanguageCode"/>
                        <xsd:element ref="CountryCode" minOccurs="0"/>
                        <xsd:element ref="VariantCode" minOccurs="0"/>
                  </xsd:sequence>
                  <xsd:sequence>
                        <xsd:element ref="CountryCode"/>
                        <xsd:element ref="VariantCode" minOccurs="0"/>
                  </xsd:sequence>
            </xsd:choice>
      </xsd:complexType>
      <xsd:element name="InternationalizationContext"
                  type="InternationalizationContextType"/>
</xsd:schema>
```

Figure A.1

A locale consists of three components: a language code, a country code and a variant, and a time zone is represented as a time zone identifier. Both the country code and the language code components of a valid locale may not be empty at the same time. Though the language code and the country code components of a locale should normally be selected from the ISO-639 two-character language codes and ISO-3166 two-character country codes, the XML schema of Figure A.1 allows one to place arbitrary strings in all the components of a locale. If a Web service provider is unable to interpret a propagated stringified locale, it may use the default locale of the server for performing the computations requested by the Web service requester.

A time zone id can be a time zone identifier derived from the Olson database, or it can be a customized time zone identifier, specifying an offset from GMT. For example, “GMT+10” or “GMT-1010” customized time zones indicate time zones ten hours ahead of GMT, or ten hours and ten minutes behind GMT respectively. A formal definition of the customized time zone id format is adopted from the `java.util.Timezone` [19] class and is in Figure A.2. Note that, no daylight savings transition time can be specified while creating customized time zone ids. Though the time zone id should be either a standard time zone id or a customized id constructed according to the rules mentioned in Figure A.2, the XML schema of Figure A.1 allows one to propagate an arbitrary string as a time zone id. If the Web service provider is unable to interpret a propagated time zone identifier, it may use the GMT time zone for performing relevant time-zone sensitive computations.

```
CustomTimeZoneID:
      GMT Sign Hours : Minutes
      GMT Sign Hours Minutes
      GMT Sign Hours
Sign: one of
      + -
Hours:
      Digit
      Digit Digit
Minutes:
      Digit Digit
Digit: one of
      0 1 2 3 4 5 6 7 8 9
```

Figure A.2

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:
IBM WebSphere

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information in this presentation concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information in this presentation addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Photographs shown are of engineering prototypes. Changes may be incorporated in production models.