Central Agency for Information Technology

KGO Portal Implementation Project

KGO Connected eServices Standards DOCUMENT & SDK

KGO Connected eServices Standards Document & SDK
Version V1.5

August 12, 2015

Table of Contents

Executive Summary

The State of Kuwait represented by The Central Agency for Information Technology (CAIT) has embarked on an ambitious project to deliver high quality eServices to citizens, residents, businesses and visitors through the nation's official portal – The Kuwait Government Online (KGO) Portal. And since CAIT is responsible for developing and maintaining the official Portal for the State of Kuwait (KGO) and setting the standards for integrating different agencies eServices into it. CAIT has developed a set of recommendations for standards, guidelines, policies, procedures, and best practices in order to help each agency in their integration efforts with the Portal, which is included in this kit in hand.

This kit includes the complete set of standards along with an SDK that will help agencies connect their eServices to the portal in the best effective way. This can summarized as:

- Overview of the strategies and procedures to be adopted while integrating eServices
- Detailed Standards for implementing KGO Compliant Embedding and Web Services based eServices.
- Cascading Style Sheet for implementing KGO Compliant Embedding and Web Services based eServices. (Softcopy & Hardcopy) as part of the SDK
- Agency Self-Assessment Questionnaire*
- Overview of the Service Delivery Framework based on ITIL Standards.
- Code samples for .NET and Java based Web services according to W3C standards, as part of the SDK.

* This Questionnaire will assist agencies in defining the challenges and choosing the level of delivery and rendering of their eServices

KGO Connected eServices Standards – An Introduction

PURPOSE

The purpose of the set of standards given in this document is to encourage compliance with eGovernment eServices Standards and Good Practices in general and in specific the ones related to the KGO.

In particular, it is intended to ensure that eServices developed/restructured by Government agencies with the KGO connectivity in mind must comply with the given standards as a minimum.

SCOPE

These standards shall apply to all eServices created to be accessible to the general public via the KGO.

INTENDED AUDIENCE

Professional developers working for or on behalf of any government agency. Other IT Staff and IT Managers are strongly encouraged to have an overview about the document.

It is assumed that readers already have a general working software development knowledge and skills, such as might be acquired through appropriate training or experience.

ASSUMPTIONS

For any agency to publish an eService on the KGO portal, the following assumptions have been made:

- The agency has analysed, designed, developed, and tested the eService using some software development technology(ies) and/or tools.
- The agency has verified and approved the outcome of the eService.
- The agency is responsible for maintaining the eServices from an operational point of view.
- The agency has a capable staff to support and maintain the service.
- The agency is responsible for providing the infrastructure that will host the eService online.
- The agency ensures high-availability (24X7) operation of the eService.
- Any performance shortcoming/limitation/inadequacy shall be handled by the agency.
- The agency has published the eService on the internet for the public.
- The agency has capable staff/third parties to make the required changes to the eService to comply with the approved CAIT KGO eServices methods and standards (Embedded eServices & Web Services Based eServices) as explained in the next sections.

KGO Connected eServices

CAIT has taken initiatives in enhancing the life of the Kuwaiti Citizens and Residents through its ambitious plan to connect the existing eServices provided by different agencies to the KGO - the National Portal for Kuwait. CAIT has approved two types of connections for the eServices; namely; embedded eServices and Web Services based eServices.

Implementing eServices in such a way will positively affect the overall international position of the State of Kuwait in the eGovernment Portals Arena.

KGO Embedded Linking eServices

DESCRIPTION

Client (or User) is able to access existing eService(s) provided by agency(ies) through KGO portal by:

User's action	Response
• Select the requested service	• KGO portal will retrieve and display the portal Service Page
• At the Service Page, click "Do It Online" link to initiate the online service	• Based on the embedded agency's web page URL, KGO portal will re-direct user to the agency's website page to execute the eService.

BENEFITS

- Quick-win strategy by gaining user's early adoption of a single entry (KGO portal) to request information and eServices (aggregated from multiple agencies)
- Quick deployment, where no additional effort required by agency except to provide the URL of the eService

LIMITATIONS

- Different look-and-feel interfaces from KGO portal
  - Each agency's website has its own identity (e.g. branding, colour scheme)
  - Each agency presents the eService content differently
- Different levels of Service Agreement
  - Each agency is responsible for the availability of service, data security and integrity, decision making and data processing.

TECHNICAL SPECIFICATIONS

- For each eService
  - Provide the existing URL of the eService
  - If there is more than one language version exists in KGO (e.g. 2 languages: Arabic and English), provide the 2 URLs (URL for Arabic version of eService page, URL for English version of eService page)
  - Availability should be at least 99.95% measured on a calendar month, and operable 24x7
  - Must support the latest browsers versions including Internet Explorer, Mozilla Firefox, Google Chrome, and Safari
  - Persistent cookies should not be used
- For each web page
  - Screen size should not exceed 1024x768 (recommended 800x600)
  - Should adopt W3C guidelines
- Response with error
  - In the event of system downtime (e.g. system maintenance), agency will provide a standard statement to inform the non-availability of the requested eService.

KGO Embedded eServices

DESCRIPTION

Client (or User) is able to access existing eService(s) provided by agency(ies) through KGO portal by:

User's action	Response
• Select the requested service	• KGO portal will retrieve and display the portal Service Page • A designated area will display only the contents of the requested eService (from the respective agency).

BENEFITS

- Quick-win strategy by gaining user's early adoption of a single entry (KGO portal) to request information and eServices (aggregated from multiple agencies)
- Promote branding of KGO
  - Unified look-and-feel of the graphical user interfaces
  - Consistent branding of the eService content and branch-off strategy (i.e. click "Do It Online" link) to execute eService
- Standardization
  - Agency will provide only the content of the eService (to be displayed within a frame of the standard KGO template for service page or through a pop-up dialog)

    Note: The presentation layer (content of the eService page(s)) will be handled by the Agency. There is no web service at this stage of maturity.

LIMITATIONS

- Different levels of Service Agreement
    - Each agency is responsible for the availability of service, data security and integrity, decision making and data processing.

TECHNICAL SPECIFICATIONS

- For each eService
    - Provide the eService after applying KGO Cascading Style Sheet styling standards, for more information, please refer to Appendix I "KGO eServices Cascading Style Sheet".
    - Must be responsive and tested against different screen sizes (Mobile Devices – Tablets – Different desktop screens)
    - Provide 2 URLs (URL for Arabic version of eService page, URL for English version of eService page)
    - Availability should be at least 99.95% measured on a calendar month, and operable 24x7
    - Must support the latest browsers versions including Internet Explorer, Mozilla Firefox, Google Chrome, and Safari
    - Must demonstrate proper exception handling for various errors and system exceptions
    - Persistent cookies should not be used
    - Should adopt W3C guidelines

- Response with error
    - In the event of system downtime (e.g. system maintenance), agency will provide a standard statement to inform the non-availability of the requested eService.

ARCHITECTURE

The conceptual architectural diagram illustrates how user can access an eService embedded in KGO portal.

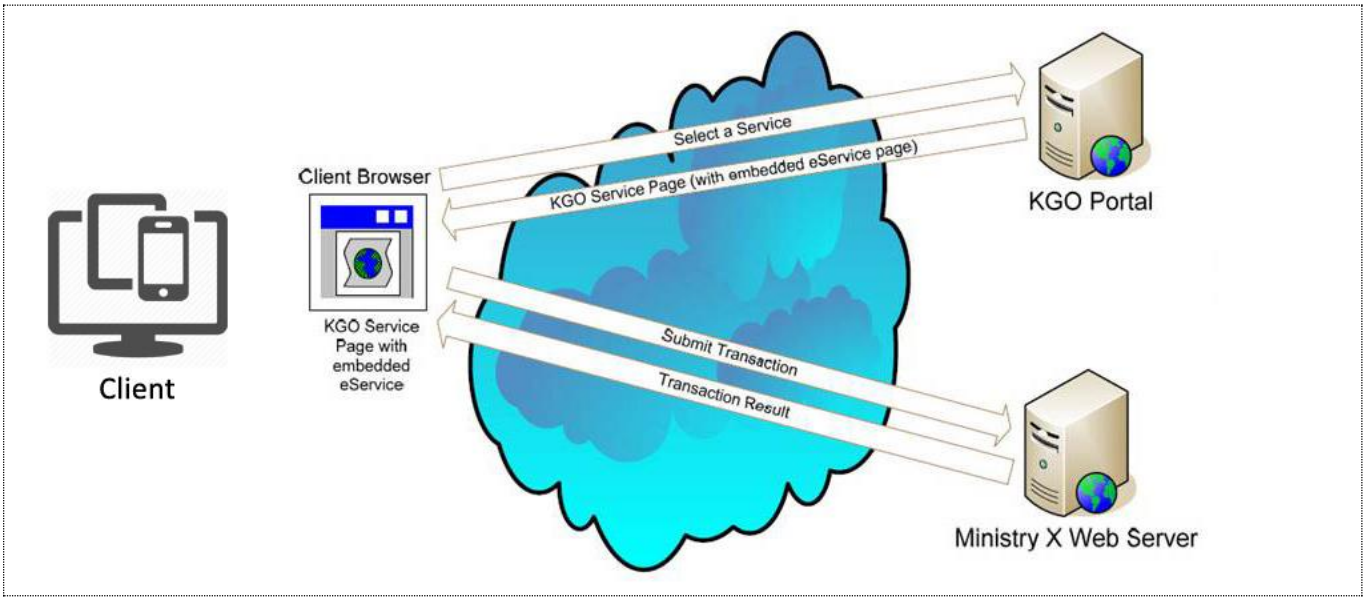

Figure 1
<Source: Diyar United >

EXAMPLES

KGO portal will display a list of the available eServices as shown in the following image

Figure 2

<Source: Diyar United, KGO Portal Landing Page >

When the user clicks on an embedded eService, the portal opens (within the same portal page/through a linked popup) the eService enhanced page on the agency's website as shown in the following image

Figure 3

<Source: Diyar United, courtesy of PACI – The eService enquiry page resides on PACI server but is embedded in KGO Page>

Figure 4

<Source: Diyar United, courtesy of PACI – The eService result page resides on PACI server but is embedded in KGO Page >

SERVICE MANAGEMENT COMPLIANCE

The eService should comply with the *it*SMF standard. For more information, please refer to Appendix IV "Service Management Compliance assessment for eServices".

Web Services based eServices

The Web Services based eServices is considered to be the ultimate goal in integrating eServices within the KGO Portal, for that we have given much attention to this section and more details will be given in Appendix V “Service Oriented Architecture (SOA)”.

DESCRIPTION

As a definition, A Web service (WS) is a software system to support interoperable machine-to-machine interaction over a network. It has an interface via the machine-process able format using Web Services Description Language (WSDL). Communication between the client and Web service is performed using Simple Object Access Protocol (SOAP) messages that are represented in an XML format.

Typically, Web services use HTTP as their transport; however, a number of different transports may be used such as MQ Series, SMTP, FTP, etc. A simple definition of a Service-Oriented Architecture (SOA) is a collection of software building blocks implemented as services and made available to consumers.

What is different with Web Services is that the call interface is based on standards that have been broadly accepted across the industry, and none are owned or controlled by a single company, they're all under the control of independent standards bodies, such as the (W3C) and (ECMA). These standards include...

- **XML** as the document formatting standard,
- **SOAP** (Simple Object Access Protocol) for the document 'wrapper'
- **WSDL** (Web Service Definition Language)
- **UDDI** (Universal Discovery and Description and Integration) the Yellow Pages which allows you to find and understand web services

This means that, it no longer matters whether you are interfacing completely heterogeneous and desperate systems.

BENEFITS

Technical motives for pursuing a standard approach to adopting a SOA via Web Services throughout the enterprise include:

- Increasing **code reuse**
- Enabling **interoperability** across heterogeneous environments and operating environments (J2EE, PowerBuilder, and Microsoft .NET Framework)
- Providing a **single repository** for storing Web service definitions discoverable by internal application development teams

- Facilitating a consistent cross-operating environment **security authentication** and **authorization** mechanism for deployed services
- Ensuring access to Web services are secured and monitored
- Establishing a **standards-based SOA framework** before application teams begin deploying services in an ad-hoc manner

DIFFICULTIES

Some specific issues and difficulties may be encountered like:

- A common approach toward implementing Web service technologies has not been established.
- Application teams have a difficult time upgrading their applications to newer operating environments and releases because of the interdependence of the client and server side of the applications.
- A common message format between different vendor systems does not exist.
- An enterprise-wide repository for code reuse does not exist.
- An end-to-end approach on how developers should write Web services has not been established.
- The duplication of data to support various applications frequently occurs.

ARCHITECTURE

The aim of Web Services is to enforce loose coupling between large pieces of functionality, while still allowing tight coupling at the fine-grained level. In this model, business functionality in the enterprise is broken up into units called services. Each service encapsulates business logic and data, and exposes an interface with which external applications can make requests and receive responses.
Applications are not permitted to access business logic or data directly – all requests must be made through an interface. This approach allows developers to later rewrite business or data access and manipulation logic (even moving to a different platform) without worrying about the potential impact on other applications. While SOA is most often associated with Web services technologies such as HTTP, SOAP, WSDL and UDDI, it is possible to implement a SOA using a variety of technologies. However, the platform-independent nature of Web services makes them an ideal choice for implementing a SOA in a heterogeneous environment.

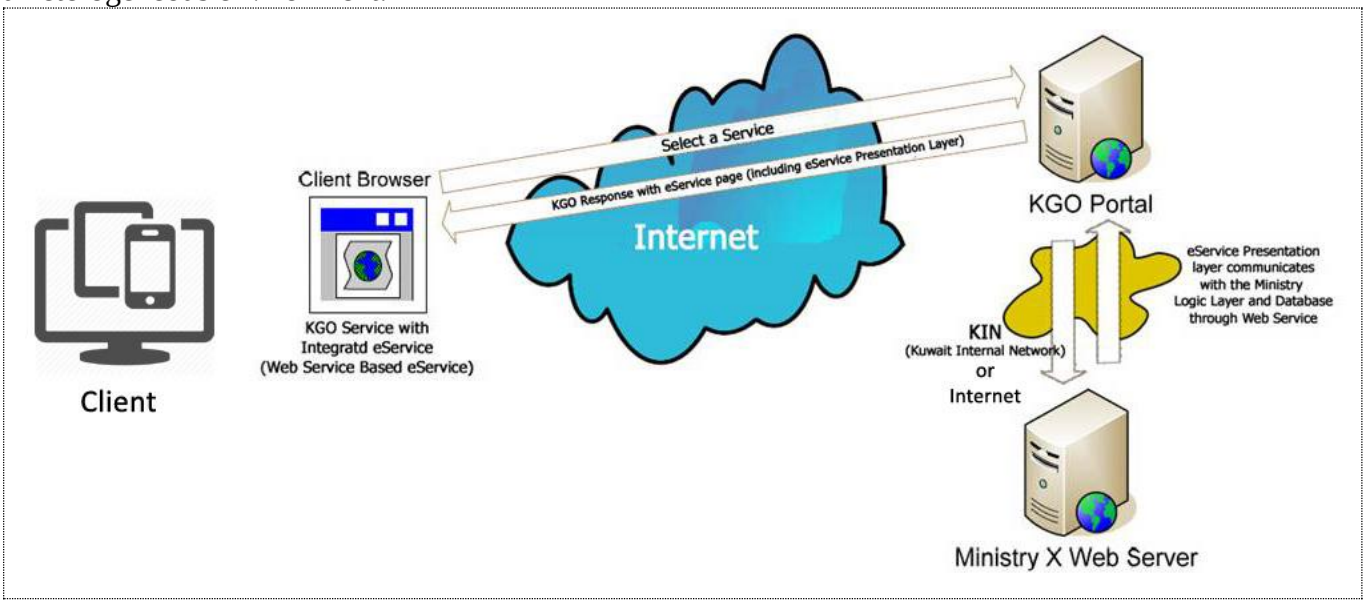

Figure 5
<Source: Diyar United >

TECHNICAL SPECIFICATIONS

- ***Service Interface***

  The service interface provides the sole mechanism by which external services or applications can access a service's functionality. The interface contains one or more *operations* or *methods* that developers program against to consume the service.

  Each operation on the interface specifies the required inputs, and corresponding outputs, all of which may be a combination of primitive data types and complex structures. Because cross-service calls are more resource-intensive than standard procedure or method calls in a single application, service operations should be designed to do the most possible useful work with the smallest number of calls. This result in fewer, chunky calls, as opposed to the frequent chatty calls often found in Object Oriented programs.

  Although they cannot generally be expressed in the service's interface, each operation will also have pre-conditions, post-conditions and exceptions. These, along with the order that operations must be called in order to yield a sensible result, constitute the service's *contract*.

  Many messaging protocols, including Web services technologies, also allow out of band information to be included as a part of the messages sent between services. While this information is not a part of the service's interface, it allows for more flexibility in service invocation by allowing common information such as security credentials and routing information to be accessed by messaging infrastructure or by the service itself.

- ***Black Box Implementation***

  The purpose of all services is to enforce business rules and to maintain some kind of data, which is normally stored in a database. The code, data, products and middleware that provide the service's functionality are collectively called the ***implementation***. The service encapsulates all of this and hides it behind the interface. The service's consumers know about the interface, but have no idea about the implementation – which is why the service's implementation is sometimes called a ***black box***.

- ***Stateless Programming Model***

  In Object Oriented programming, a program is able to create an object, after which it 'owns' the instance of the object until it falls out of scope or is destroyed. With web service, the relationship between services is quite different. There is no concept of 'ownership' or 'instances' – instead, there is (logically) a single running instance of the service which is able to service requests from multiple clients. As a consequence of this, a service cannot keep a memory of previous requests from a single client. This means that a client must pass enough contexts to be identifiable by the service with every request (if it needs to be identified). Note that the service is able to store shared state, so the client does not need to send *everything* with each request – the context could be as simple as an ID number which the service can use to retrieve client-specific data out of Agency database.

- ***Healthy Distrust***

  Services should have a healthy distrust with the outside world, which is everything not part of the service's own implementation. This distrust is necessary to achieve the goals of loose coupling and reusability. *A service should assume that any data it receives could be incorrect or malicious*. Consequently *all services should include mechanisms such as validation and authorization to ensure their data is kept consistent*. A service should not be limited to certain 'trusted' consumers, as the service's owners will rarely have complete visibility of the consumers (which is necessary to ensure their trustworthiness). In addition, restricting the use

of a service will preclude future reuse scenarios, for example exposing the service over the Internet to another organization.

EXAMPLES

From a look and feel point of view the Web Service based eServices will look exactly the same as the embedded eServices in KGO. Ideally the KGO portal will display a list of the available eServices as shown in the following image

Figure 6

<Source: Diyar United, KGO Portal Landing Page >

When the user clicks on a Web Service based eService the portal opens the integrated presentation layer of that eService as part of KGO page, as shown in the following image

Figure 7

<Source: Diyar United, courtesy of PACI – The eService presentation layer resides on KGO servers>

The agency's presentation layer (hosted in KGO page) will integrate with the Agency backend through web services to facilitate communication and (Submit Entries\Show Results), yet the look and feel will exactly be like the embedded eService.

Figure 8

<Source: Diyar United, courtesy of PACI – Web services facilitate the communication between presentation layer and Agency backend>

For developing web services using Microsoft .Net technology please refer to Appendix III .NET Web Service Development Guidelines

For developing web services using JAX-WS technology please refer to Appendix IV Web Service with JAX-WS Development Guidelines

SERVICE MANAGEMENT COMPLIANCE

The eService must comply with the *it*SMF standard. For more information, please refer to Appendix IV "Service Management Compliance assessment for eServices".

Electronic Forms (e-Forms)

eForms are the electronic version of the paper-based forms that we use in our everyday life. The need to use eForms is becoming vital in every aspect, especially in Government interactions with Citizens and Residents. Agencies can design, fill, save, print, email and track these forms in desktop and Web applications and easily move them through Web. eForms are equipped with all the features and technologies that bridge the gap between the Government Agency and the end user in a quick and efficient matter. eForms have a lot of benefits and advantages over the regular paper-based forms. With eForms, Agencies are able to manage, maintain, create, validate, and extract the information filled-in by users electronically.

As opposed to paper-based forms, eForms have a numbers of features that enables better interaction with users and ensures accurate extraction of information. eForms allows organizations to maintain the same look and feel of the paper-based document and to embed business logic, validations into the forms.
eForms are presented to end users in different format such as:

- Static forms

  In these forms, the paper-based form is converted as-is into an electronic form to be published online over the internet
  - Can download forms in PDF format, fill it off line
  - Can be printed, filled manually and submitted manually or via Fax machine
  - Can be filled online, printed and submitted manually or via Fax machine

- Dynamic forms

  In dynamic forms, the form embeds dynamic features such as the ability to fill-in fields, validating the data entered, generating error messages, utilizing dynamic tables, using dialog boxes and messages, etc.
  - Can be filled online, validate the input data, pre-populate some data with online submission

- Advanced Forms

  Advanced Forms includes very rich features, and provide engaging experience for non-sophisticated users, taking them through the steps of a process and reduces frustration and abandonment. This kind of eForms takes the user through sequential steps of questions and answers to retrieve the requested information.
  - Can be filled online, validate the input data, pre-populate data with online submission and be part of business process workflow where it also it can be kept as online documents in the government shared repository for later use.

  The printed version of the e-From can embed a 2-dimensional barcode that stores all filled-in information and as a result, the organization is able to extract the data without performing any rekeying of data.

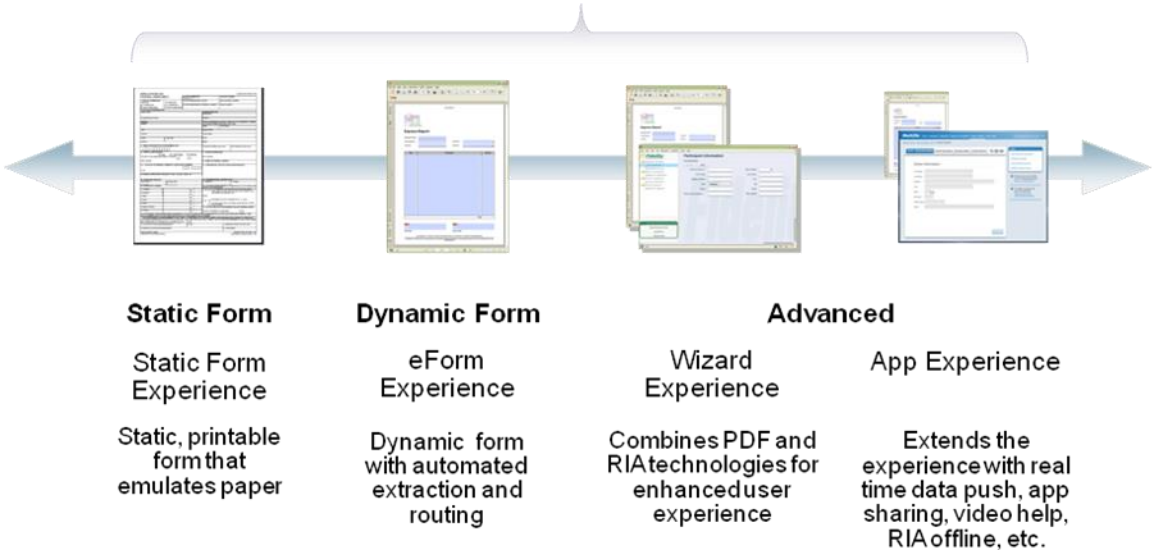

Figure 9
<Source: Adobe>

In addition, the organization ensures the completion and accuracy of the submitted form since eForms employ all recognized logic and business validation that the organization utilizes.

BENEFITS

- Create and generate smart dynamic forms that have the same look and feel of the original physical form and embed all business logic and validations.
- Improve Citizen & Resident satisfaction through online forms instead of agency visits, phone calls and paperwork
- Reduce costs through improved automation of information collection, retrieval, and storage
- Reduce errors introduced through hand-processing of paper forms
- Automate the extraction of data from paper-based forms using 2-dimensional barcode technology and deliver data to backend systems seamlessly and efficiently.
- Eliminate errors occurring due to user ignorance of rules while filling the form
- Eliminate errors occurring due to employee rekeying filled-in information
- Fully secure, if PDF forms were used with digital signatures and encryption capabilities
- Deploy and distribute smart forms in online (as XML and HTML) or offline (rich PDF forms) operations.
- Automate complex business processes by using smart forms, integrating with backend system, and using an advanced workflow system.
- Easily create, manage, and publish eForms to large number of users.
- Quickly update look & feel and content of existing eForms
- Enhance access to key, relevant government agency information

Compared to paper forms, e-Forms allow more focus on the business process or underlying problem for which they are designed (for example, expense reporting, purchasing, or time reporting).

e-Forms are intelligent enough to understand the roles and responsibilities of the different participants of the process and, in turn, automate routing and much of the decision making necessary to process the form.

Some e-form products now support many of the well-established Internet/intranet protocols. Organizations that rely on intranets and the Internet for internal, public, and business-to-business communications can further benefit by integrating intelligent e-forms for data collection and process automation. Many software programs include e-forms as an integral part of the application.

Support Information

For any further clarification regarding this document, please refer to the below link.

www.cait.gov.kw

This link includes FAQ, Sample Codes and much more other rich resources.

References

http://www.w3.org/
http://quickstarts.asp.net/QuickStartv20/webservices/
http://www.w3schools.com/webservices/
http://java.sun.com/webservices/docs/1.6/tutorial/doc/index.html
http://support.microsoft.com/kb/323752/EN-US
https://support.microsoft.com/en-us/kb/324013

Appendices

Appendix I

KGO eServices Cascading Style Sheet

KGO Eservices Cascading Style Sheet and JavaScript

Cascading Style Sheets simplify development and maintenance of websites. Pages within websites typically follow a standard format and layout. Common elements such as branding and navigation appear consistently across pages. Text has consistent style and size from page to page. There are common elements that appear on every page. A single style sheet can be developed and applied to every page that should have a similar design. When changes are needed, they can be made once to the single style sheet, and the changes are automatically applied to every page referencing that style sheet. As new content is created, the style sheet can be applied to the new content to create a new page. This is faster, less labor-intensive, and more accurate than making the changes to each affected page. In KGO eServices integration, the agency should refer to the KGO Cascading Style Sheets and JavaScript's files from the following URLs:

- English CSS File URL: https://www.e.gov.kw/css/en/KGOeServices.css
- Arabic CSS File URL: https://www.e.gov.kw/css/ar/KGOeServices.css
- JQuery Files URLs: https://www.e.gov.kw/js/jquery-ui-1.10.4.custom.js

  https://www.e.gov.kw/js/jquery-ui-1.10.4.custom.min.js

  https://www.e.gov.kw/js/footable.js

- Code Snippet to be added to the eService page to assure responsive gridviews:

```
<script type="text/javascript">
$(function () {
    $('.footable').footable();
});
</script>
```

The cascading style sheet, JavaScript files, along with samples of usage, can be found on the attached CD

Appendix II

Web Service Recommendations

Web Service Recommendations

Web Service Interoperability

Web services technology offers the promise and hope of integrating disparate applications in a seamless fashion. But enterprise applications are built around different technologies and platforms, and integration across these is never a trivial task. There are several issues which arise when web services on disparate platform interoperate:

- Most often developers use tools to derive the Web services semantics in WSDL from implementation code. This is convenient, but this approach ignores the design of the message schemas which is central to Web services interoperability in heterogeneous environments.
- Developers often use *RPC/encoded* style for developing web services because of their ease of use, flexibility, and familiarity. However, the implementation of the SOAP encoding data model among various vendors is not consistent and this leads to interoperability problems.
- Weakly-typed collection objects, arrays containing null elements, and certain native data types all pose special problems for interoperability. Specifically:
    - It is impossible for vendor tools to accurately interpret XML Schemas representing weakly-typed collection objects and map them to the correct native data types.
    - The XML representations of an array with null elements differ between platforms
    - Because native data types and XSD data types do not share a one-to-one mapping, information or precision can be lost during the translation.

IBM's Web Sphere Application Developer IDE is a popular development tool for building web services on the J2EE platform. On the other hand, Microsoft's Visual Studio is the tool of choice for building web services on the .Net platform. This document attempts to identify the most common problems which arise when developers try to interoperate between web services developed using one platform with clients developed using the other platform.

Design the XSD and WSDL first.

WSDL is the interface definition language (IDL) of Web services and the contract between the client and the server. It defines the services semantics, namely the message types, data types, and interaction styles. These are the key to building loosely-coupled systems. Ideally, just like the IDLs for COM and CORBA, create and edit the WSDL first -- define the interfaces, messages, and data types before building the Web services and clients in specific implementation languages based on the service semantics in the WSDL.

XSD offers a wide range of built-in primitive types and further allows service providers to define custom complex types. The XSD-type system is quite sophisticated and more importantly it is language-neutral. Always start with a XSD and then define language and platform specific data types using tools like XSD.exe which convert schemas into class representations. This ensures that the wire representations of data types are compatible across platforms.

Avoid using the RPC/encoded style. Use Document/Literal.

SOAP encoding is recognized as one of the major sources of the Web services interoperability problems. ASP.NET defaults to *Document/literal*, while most of the J2EE Web services toolkits default to *RPC/encoded*.
The *RPC/encoded* style was popular in the J2EE world mainly because of its simple programming model for developers who are accustomed to the remote procedure calling convention. The RPC binding style uses the names of the method and its parameters to generate structures that represent a method's call stack, so it makes the Web services look like a single logical component with encapsulated objects, all handled in the SOAP RPC stack.
WS-I Basic Profile 1.0 promotes the use of literal XML for interoperability. It prohibits the use of soap:encodingStyle attributes to soap:Envelope or descendents of the soap:Body elements. Therefore, *RPC/literal* and *Document/literal* are the only two formats supported by the WS-I standards. But not many Web services toolkits support *RPC/literal*, so it leaves the *Document/literal* as the only actual interoperability standard.

Avoid weakly-typed collection objects

Collection objects might contain elements of any data types. Thus, they are considered as weakly-typed data structures. If exposed across Web services, these collection types can cause problems. The problem lies in how the receiving side is able to understand the serialized SOAP messages that contain the weakly-typed object elements and native data types.
Even though some collection types look extremely similar between languages, such as System.Collections.ArrayList in C# and java.util.ArrayList in Java, the elements in the collections are generic references. To accurately unmarshall the XML representation of a collection, consumers must have prior knowledge of the original concrete types. The burden is on the toolkit developers to interpret the XML Schemas published by the Web services providers and map the SOAP messages to the native data - - not an easy task for the weakly-typed collections.
Example: Consider a *InventoryService* Web service deployed on the Microsoft .NET framework. This web service accepts a *System.Collections.ArrayList* of *Product* as arguments, sets the new price by increasing 10 percent for each product in the *ArrayList*, and returns the new object of *System.Collections.ArrayList* type.

Listing 1. An Inventory Web service in C#

```
namespace Inventory
{
 [WebService(Namespace="http://services.inventory")]
 public class InventoryService: WebService
 {
        //increase the product price by 10 percent
        private static float inc_rate = 0.10F;
        public struct Product {
                public string name;
                public int      qty;
                public float price;
        }
        [WebMethod]
        [XmlInclude(typeof(Product))]
        public ArrayList updateProductPrice(ArrayList products)
```

```
        {
                ArrayList newList = new ArrayList();
                IEnumerator eList = products.GetEnumerator();
                while(eList.MoveNext())
                {
                  Product item = (Product)(eList.Current);
                  item.price = item.price * (1 + inc_rate);
                  newList.Add(item);
                }
                return newList;
        }
  }
}
```

The WSDL engine in the .NET framework generates the following XML Schema for the Collection type, ArrayList, and the Product complex type :

Listing 2. The XML Schema for the ArrayList and Product

```
1.      <types>
2.      <s:schema xmlns:s="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
targetNamespace="http://services.inventory">
3.      <s:element name="updateProductPrice">
4.      <s:complexType>
5.      <s:sequence>
<s:element maxOccurs="1" minOccurs="0" name="products"
type="s0:ArrayOfAnyType"/>
6.      </s:sequence>
7.      </s:complexType>
8.      </s:element>
9.      <s:complexType name="ArrayOfAnyType">
10.     <s:sequence>
11.     <s:element maxOccurs="unbounded" minOccurs="0" name="anyType" nillable="true"/>
12.     </s:sequence>
13.     </s:complexType>
14.     <s:complexType name="Product">
15.     <s:sequence>
16.     <s:element maxOccurs="1" minOccurs="0" name="name" type="s:string"/>
17.     <s:element maxOccurs="1" minOccurs="1" name="qty" type="s:int"/>
18.     <s:element maxOccurs="1" minOccurs="1" name="price" type="s:float"/>
19.     </s:sequence>
20.     </s:complexType>
21.     <s:element name="updateProductPriceResponse">
22.     <s:complexType>
23.     <s:sequence>
<s:element maxOccurs="1" minOccurs="0" name="updateProductPriceResult"
type="s0:ArrayOfAnyType"/>
24.     </s:sequence>
25.     </s:complexType>
26.     </s:element>
```

```
27.     </s:schema>
28.     </types>
```

Lines 9 through 13 (see Listing 2) define a complex type, xsd:ArrayOfAnyType, with an unbounded sequence of elements *anyType*. The ArrayList of Products has been translated into a sequence of anonymous elements in the XML Schema definition. This is expected; however, it poses two problems. First, other Collection types will also be translated into xsd:ArrayOfAnyType. Therefore, how does the SOAP toolkit on another platform decide which Collection type to map it to?
Secondly, the xsd:anyType is the default type when the type is not specified. Line 11 in Listing 2 is expected because the objects in a Collection are generic references -- the types are not known until run-time. The problem occurs when the SOAP toolkit in another platform receives the serialized objects. How can the right serializer to de-serialize the XML payload back to the concrete objects be found?
In fact, JAX-RPC generates the following helper class from the xsd:ArrayOfAnyType schema in Listing 2.

Listing 3. The resulting helper class for the xsd:ArrayOfAnyType schema

```
public class ArrayOfAnyType  implements java.io.Serializable {
   private java.lang.Object[] anyType;
 <!-- The setter, getter, equals() and hashCode() methods -->
}
```

From Listing 3, it can be seen that the ambiguities in the xsd:ArrayOfAnyType schema have caused the JAX-RPC tool to generate the helper class with an generic java.lang.Object[] array as its private field instead of the concrete Product array.
To resolve this ambiguity, use *ArrayOfRealType* instead of the xsd:ArrayOfAnyType. Expose only the simple array of concrete types (that is, Product[]) as the signature of Web service methods.
For the Web service in Listing 1, a facade method can be exposed:

Listing 4. Facade to expose the simple array Product[]

```
        [WebMethod]
        [XmlInclude(typeof(Product))]
        public Product[] updateProductPriceFacade(Product[] products)
        {
                ArrayList alist = new ArrayList();
                IEnumerator it = products.GetEnumerator();
                while (it.MoveNext())
                        alist.Add((Product)(it.Current));
                alist = updateProductPrice(alist);
                Product[] outArray = (Product[])alist.ToArray(typeof(Product));
                return outArray;
        }
```

The new schemas for the input and output message parts are:

Listing 5. The XML Schema for the new Web service in Listing 4

```
1.      <s:element name="updateProductPriceFacade">
2.      <s:complexType>
3.      <s:sequence>
4.      <s:element minOccurs="0" maxOccurs="1" name="products"
type="s0:ArrayOfProduct" />
```

```
5.      </s:sequence>
6.      </s:complexType>
7.      </s:element>
8.      <s:complexType name="ArrayOfProduct">
9.      <s:sequence>
10.     <s:element minOccurs="0" maxOccurs="unbounded" name="Product"
type="s0:Product" />
11.     </s:sequence>
12.     </s:complexType>
13.     <s:element name="updateProductPriceFacadeResponse">
14.     <s:complexType>
15.     <s:sequence>
16.     <s:element minOccurs="0" maxOccurs="1"
name="updateProductPriceFacadeResult" type="s0:ArrayOfProduct" />
17.     </s:sequence>
18.     </s:complexType>
19.     </s:element>
```

From Line 8 to Line 12, the xsd:ArrayOfProduct schema is created to represent the concrete Product array. No ambiguity is present in the schema. As a result, the Web service client will have no problem in de-serializing the array of Products.

Avoid passing an array with null elements.

The XML representations of an array with null elements are different between .NET and J2EE. Consider the Java Web service method in Listing 6.

Listing 6. A Java method returning an array with a null element

```
public String[] returnArrayWithNull() {
        String[] s = new String[3];
        s[0] = "ABC";
        s[1] = null;
        s[2] = "XYZ";
        return s;
}
```

The String element, s[1], is assigned a null value. When a .NET client invokes this Web service method hosted on the J2EE platform, the String array is serialized as:

Listing 7. The Web service response message from WebSphere

```
<soapenv:Body>
  <returnArrayWithNullResponse xmlns="http://array.test">
  <returnArrayWithNullReturn>ABC</returnArrayWithNullReturn>
  <returnArrayWithNullReturn xsi:nil="true"/>
  <returnArrayWithNullReturn>XYZ</returnArrayWithNullReturn>
  </returnEmptyStringResponse>
</soapenv:Body>
```

The second element in the array is set to xsi:nil="true". That's fine in Java; a Java client would have correctly de-serialized it back to a null String for the second element in the array. However, the .NET client de-serializes it into a zero length string instead of a null string.
Now consider another Web service method hosted on WebSphere, as shown in Listing 8.

Listing 8. A Java method with arrays and its input and output signatures

```
public String[] processArray(String[] args) {
        //do something to the input array and return it back to the client
        return args;
}
```

This time, the Web service method takes an array as input, processes it, and returns the array back to the client. Suppose a .NET client sends out an array with a null element as shown in the code in Listing 9.

Listing 9. A .NET client sends an array with a null element

```
TestArrayService proxy = new TestArrayService();
string[] s = new string[3];
s[0] = "abc";
s[1] = null;
s[2] = "xyz";
// Console.WriteLine("the length of the input array = " + s.GetLength(0));
string[] ret = proxy.processArray(s);
// Console.WriteLine("the length of the output array = " +
ret.GetLength(0));
```

Listing 10 shows the SOAP request message from the .NET client.

Listing 10. The SOAP request message sent by the .NET client

```
<soap:Body>
  <processArray xmlns="http://array.test">
    <args>abc</args>
    <args>xyz</args>
  </processArray>
</soap:Body>
```

The null element, s[1], is omitted from the SOAP request sent by the .NET client. As a result, the length of the returned array is no longer the same as the length of the original array. If this array's length or element index is important to the client's logic, then the client will fail.
The best practice is not to pass an array with null elements between Web service clients and servers.

Avoid unsigned numerical data types.

A one-to-one mapping is not available between native data types and XSD data types. Therefore, information can be lost during the translation, or the receiver may simply not be able to do the mappings for certain native data types. Unsigned numerical types, such as xsd:unsignedInt, xsd:unsignedLong, xsd:unsignedShort, and xsd:unsignedByte, are the typical examples. In .NET, the uint, ulong, ushort, and ubyte types map directly to those xsd types, but the Java language does not have unsigned numerical types.

For interoperability, do not expose those numerical data types in the Web service methods. Instead, create wrapper methods to expose and transmit those numerical types as xsd:string (using System.Convert.ToString in C#).

Avoid using XSD types which map to a value type in one language and to a reference type in another.

Passing matching data types which are reference types on one platform and value type on the other can also cause problem. The object of a value type is in the stack, but the object of a reference type is in the heap. That means a reference type can have a null pointer, but a value type cannot have a null value. This can lead to a problem if the XSD type is mapped to a value type in one language but mapped to a reference type in another. For example, the xsd:dateTime is mapped to System.DateTime, which is a value type in C#. It is also mapped to java.util.Calendar, which is a reference type in Java. In fact, both java.util.Date and java.util.Calendar are reference types. In Java, it is a common practice to assign a null value to a reference type when it is not referencing any object. However, .NET Web services will throw a System.FormatException if it receives a null value to its value type of data from a Java client.
To avoid this problem, define a complex type to wrap the value type and set the complex type to be null to indicate a null reference.

Test Conformance using the WS-I Conformance Testing Tool

The WS-I Working Group has developed a WS-I Conformance Testing Tool to help developers determine whether their Web services are conformant with the WS-I basic Profile guidelines. A Profile Conformance Report can be created from the WS-I Testing Tool Analyser to document a conformance claim in your WSDL documents. Currently the C# and Java versions of the testing tool are available for download. By running the testing tool, much of the obvious violations of the WS-I Basic Profile 1.0 are reported.
Although the WS-I Conformance Testing Tool does not capture everything that might cause the Web services interoperability issues in the field, it is nonetheless a powerful tool. In parallel with the Web services development, incrementally developing a comprehensive test suite to capture potential interoperability issues is a best practice.

Appendix III

.NET Web Service Development Guidelines

.NET Web Service Development Guidelines

Creating XML Web Services with Visual Studio 2005

To create XML Web services in managed code using Visual Studio, you need access to a Web server configured for developing ASP.NET applications. There are several types of Web projects; when you want to work with XML Web services using managed code in Visual Studio .NET, you use the ASP.NET Web Service Application project template. After you create an XML Web service project in Visual Studio, you see the Component Designer. The Component Designer is the design surface for your XML Web service. You can use the Design view to add components to your XML Web service, and the Code view to view and edit the code associated with your XML Web service.
When you create an ASP.NET Web Service project in Visual Studio, it constructs a Web application project structure on the Web server and a Visual Studio solution file on your local computer. The solution file (.sln) contains the configuration and builds settings and keeps a list of files associated with the project. In addition, Visual Studio automatically creates the necessary files and references to support an XML Web service. When completed, the Visual Studio integrated development environment (IDE) displays the .ASMX file in Design view.
By default, Visual Studio uses code-behind files, such as Service1.asmx.vb or Service1.asmx.cs, when you create an XML Web service with the ASP.NET Web Service project template. The code-behind file contains the code that implements the functionality of the XML Web service. By default, this code-behind file is hidden in Solution Explorer. When viewing the Code view of the. ASMX file, you are actually seeing the contents of this code-behind file. The. ASMX file itself contains a processing directive, Web Service, which indicates where to find the implementation of the XML Web service. When you build an XML Web service in managed code, ASP.NET automatically provides the infrastructure and handles the processing of XML Web service requests and responses, including the parsing and creation of SOAP messages. The compiled output is a .DLL file that is placed in the project's bin folder.
The following walkthrough describes the process for creating a project for an XML Web service that converts temperatures measured in Fahrenheit to Celsius using Visual Basic or Visual C#.
Visual Studio provides an ASP.NET Web Service project template to help you create XML Web services in Visual Basic and Visual C#.

To create an ASP.NET Web Service Project

- On the **File** menu, point to **New**, and then click **Project**.
- In the **New Project** dialog box, select either the **Visual Basic Projects** or **the Visual C# Projects** folder.
- Click the **ASP.NET Web Service Application** icon.
- Enter the **Web Service Name** and **Location** for the project files. By default, the project uses your local machine, http://localhost as Web server name.

Note: You develop XML Web services on a development server. By default, the development server is your local machine. Typically, you develop and build the project on a development server, and then you deploy it to another server (the deployment server) that will host the XML Web service using a deployment project. However, if you are developing directly on the server that will host the XML Web service, the development server and deployment server are the same.

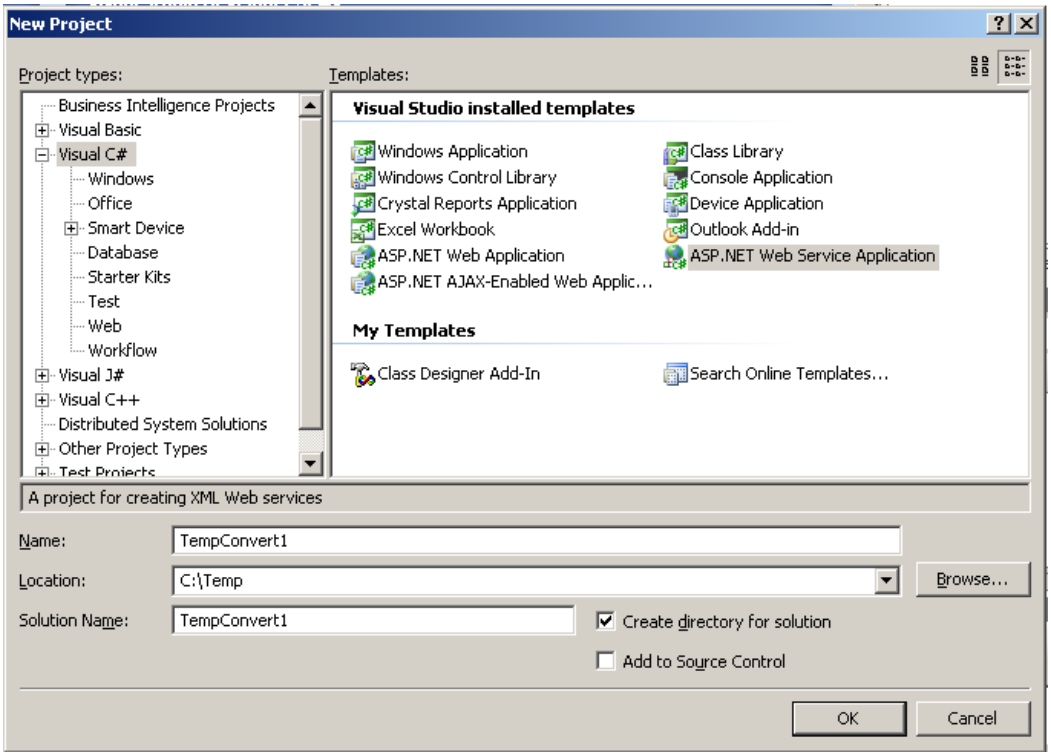

Figure III-1

Click OK to create the project.

Visual Studio automatically creates the necessary files and includes the needed references to support an XML Web service. When you create an XML Web service project in Visual Studio, you see the Component Designer for Service1.asmx.

Web Service Processing Directive

The **WebService** processing directive provides necessary information to the ASP.NET environment, such as which class implements the XML Web service functionality. An example of the **WebService** processing directive in an .asmx file follows:

```
// C#
<%@ WebService Language="c#" Codebehind="Service1.asmx.cs"
   Class="TempConvert1.Service1" %>
```

Note: To view the contents of the .asmx file, in **Solution Explorer**, right-click the .asmx file and click **Open With** on the shortcut menu. In the Open With dialog box, select **Source Code (Text) Editor**, and then click **Open**.
The Language attribute indicates the programming language used to develop the XML Web service. You can create XML Web services in any .NET-compatible language, such as Visual Basic .NET or Visual C#. The code-behind file associated with the .asmx page is indicated using the Codebehind attribute. The Class attribute indicates which class in the code-behind file implements the functionality of the XML Web service.

System.Web.Services.WebService Class

The **System.Web.Services.WebService** class, which defines the optional base class for XML Web services, provides direct access to common ASP.NET objects, such as those for application and session state. By default, XML Web services created in managed code using Visual Studio inherit from this class. The XML Web service can inherit from this class to gain access to ASP.NET's intrinsic objects, such as **Request** and **Session**.
If the XML Web service does not inherit from the **System.Web.Services.WebService** class, it can still access the ASP.NET intrinsic objects from **System.Web.HttpContext.Current**. The class implementing the XML Web service must be public and must have a public default constructor (a constructor without parameters). This makes it possible for ASP.NET to create an instance of the XML Web service class to process incoming XML Web service requests.

WebService Attribute

Each XML Web service requires a unique namespace, which makes it possible for client applications to differentiate among XML Web services that might use the same method name. The default namespace for XML Web services created in Visual Studio .NET is "http://tempuri.org/". Although the namespace resembles a typical URL, you should not assume that it is viewable in a Web browser, it is merely a unique identifier.

Note: You may want to provide a Web page at that location that contains information about XML Web services you provide.

The **WebService** attribute provides the following properties:

- **Description** – The value of this property contains a descriptive message that is displayed to prospective consumers of the XML Web service when description documents for the XML Web service are generated, such as the service description and the service help page.
- **Name** – The value of this property contains the name for the XML Web service. By default, the value is the name of the class implementing the XML Web service. The **Name** property

identifies the local part of the XML qualified name for the XML Web service. The **Name** property is also used to display the name of the XML Web service on the service help page.

- **Namespace** – The value of this property contains the default namespace for the XML Web service. XML namespaces offer a way to create names in an XML document that are identified by a Uniform Resource Identifier (URI). By using XML namespaces you can uniquely identify elements or attributes in a XML document. So, within the service description for an XML Web service, **Namespace** is used as the default namespace for XML elements directly pertaining to that XML Web service. If not specified, the default namespace is used, http://tempuri.org/.

The following code demonstrates the use of the **WebService** attribute:

```
//C#
[System.Web.Services.WebService(
   Namespace="http://Walkthrough/XmlWebServices/",
   Description="A temperature conversion service.")]
public class Service1 : System.Web.Services.WebService
{
    // Implementation code.
}
```

WebMethod Attribute

When you create an XML Web service in managed code, you indicate the methods that are available through that XML Web service by placing the **WebMethod** attribute before the method declaration of a **Public** method. **Private** methods cannot serve as the entry point for an XML Web service, although they can be in the same class and the XML Web service code can call them. The **WebMethod** attribute must be applied to each public method that is available as part of the XML Web service.
The **WebMethod** attribute contains several properties for configuring the behaviour of an XML Web service. For example, you can use this attribute to provide a brief description that will appear on the associated service help page.
The WebMethod attribute provides the following properties:

- **BufferResponse** – When set to **true**, the default setting, ASP.NET buffers the entire response before sending it to the client. The buffering is very efficient and helps improve performance by minimizing communication between the worker process and the Internet Information Services (IIS) process. When set to **false**, ASP.NET buffers the response in chunks of 16 KB. Typically, you set this property to **false** only if you do not want the entire contents of the response in memory at one time. For example, you are writing back a collection that is streaming its items out of a database.
- **CacheDuration** – Specifies for how many seconds ASP.NET should cache the results for each unique parameter set. The default value is zero, which disables the caching of results.
- **Description** – Supplies a description for an XML Web service method, which appears on the service help page. The default value is an empty string.
- **EnableSession** – When set to **false**, the default setting, ASP.NET cannot access the session state for an XML Web service method. When set to **true**, the XML Web service can access the session state collection directly from **HttpContext.Current.Session** or with the **WebService.Session** property if it inherits from the **WebService** base class.

- **MessageName** – Enables the XML Web service to uniquely identify overloaded methods using an alias. Unless otherwise specified, the default value is the method name. When specifying a value for MessageName, the resulting SOAP messages will reflect this name instead of the actual method name.
- **TransactionOption** – Specifies whether the XML Web service method can participate as the root object of a transaction. Even though you can set the TransactionOption property to any of the values of the TransactionOption enumeration, an XML Web service method only has two possible behaviours; it does not participate in a transaction (**Disabled, NotSupported, Supported**), or it creates a new transaction (**Required, RequiresNew**). Unless otherwise specified, the default value is **TransactionOption.Disabled**. To use this property, you need to add a reference to **System.EnterpriseServices.dll**. This namespace contains methods and properties that expose the distributed transaction model found in COM+ services. The **System.EnterpriseServices.ContextUtil** class lets you vote on the transaction using the **SetAbort** or **SetComplete** methods.

The following code demonstrates the use of the **WebMethod** attribute:

```
//C#
public class Service1 : System.Web.Services.WebService
{
   [WebMethod(Description="This method converts a temperature in " +
      "degrees Fahrenheit to a temperature in degrees Celsius.")]
   public double ConvertTemperature(double dFahrenheit)
   {
      return ((dFahrenheit - 32) * 5) / 9;
   }
}
```

Deploying XML Web Services with Visual Studio

When using Visual Studio .NET to create XML Web services in managed code, you use a standard deployment model: You compile your project and then you deploy the resulting files to a production server. The project .dll file contains the compiled code from the XML Web services code-behind class file (.asmx.vb or .asmx.cs) along with all other class files included in your project, but not the .asmx file itself. You then deploy this single project .dll file to the production server without any source code. When the XML Web service receives a request, the project .dll file is loaded and executed.

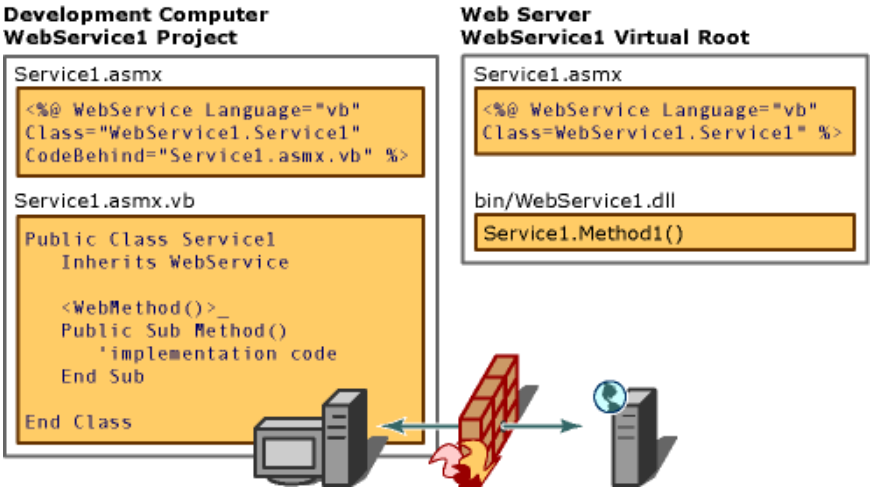

Figure III-2

To deploy the XML Web service to a server other than the development server, you can add a Web Setup project or you can copy the required files to the destination server. To make your XML Web service available to others, you will need to deploy it to a Web server that is accessible to the clients you wish to support.

Service Help Page

When called from a Web browser without supplying a recognized query string, the .asmx file returns an automatically generated service help page for the XML Web service.
For example, to access the service help page for an XML Web service named Service1.asmx that is part of a project named WebService1 on your local machine, you use the following URL:
http://localhost/TempConvert1/Service1.asmx
This service help page provides a list of the methods the XML Web service provides and that you can access programmatically. Each method has a link that takes you to additional information about that method. In addition, this page contains a link to the XML Web service description document.

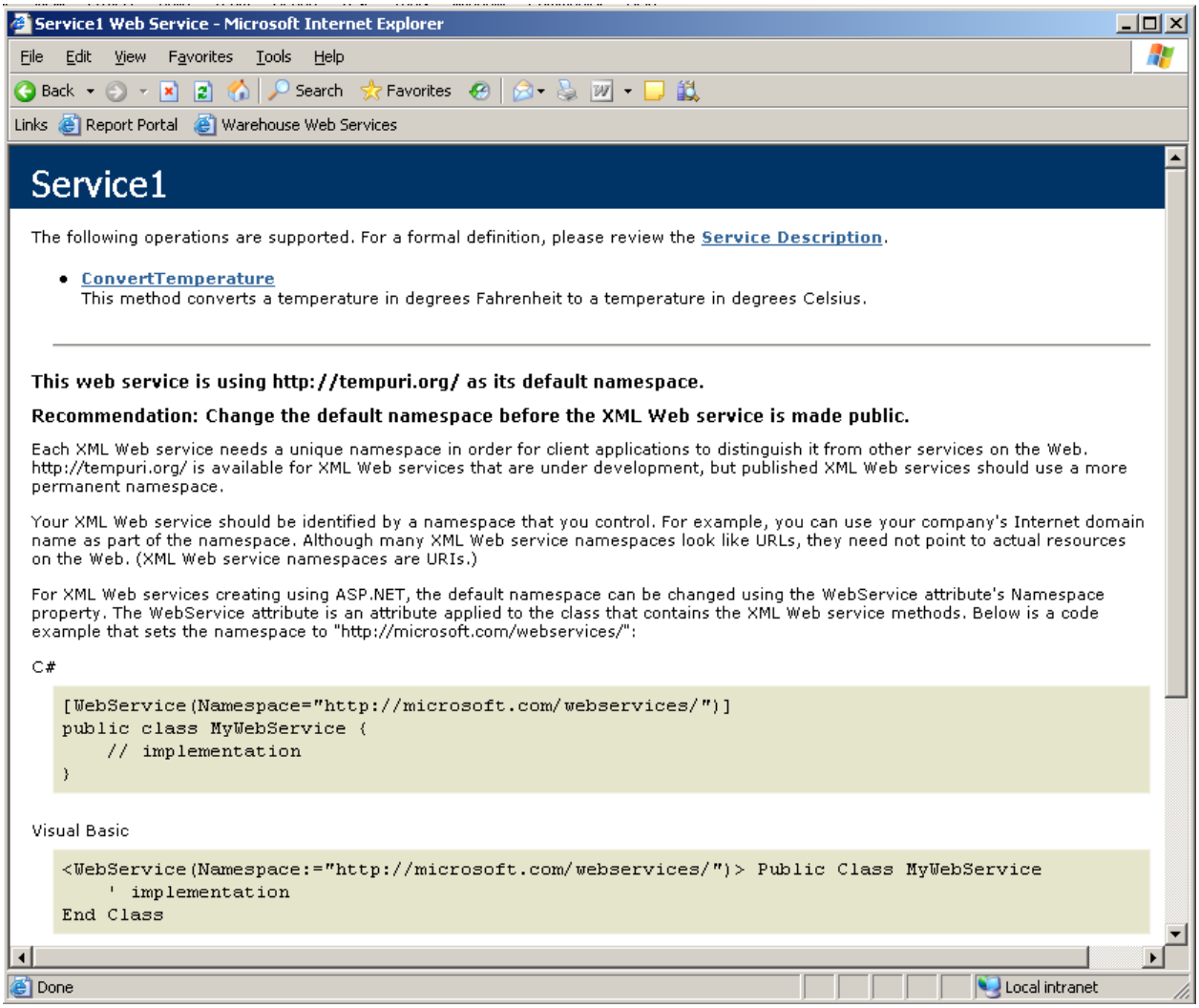
Service1 Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Links Report Portal Warehouse Web Services
Service1
The following operations are supported. For a formal definition, please review the Service Description.
ConvertTemperature
This method converts a temperature in degrees Fahrenheit to a temperature in degrees Celsius.
This web service is using http://tempuri.org/ as its default namespace.
Recommendation: Change the default namespace before the XML Web service is made public.
Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. http://tempuri.org/ is available for XML Web services that are under development, but published XML Web services should use a more permanent namespace.
Your XML Web service should be identified by a namespace that you control. For example, you can use your company's Internet domain name as part of the namespace. Although many XML Web service namespaces look like URLs, they need not point to actual resources on the Web. (XML Web service namespaces are URIs.)
For XML Web services creating using ASP.NET, the default namespace can be changed using the WebService attribute's Namespace property. The WebService attribute is an attribute applied to the class that contains the XML Web service methods. Below is a code example that sets the namespace to "http://microsoft.com/webservices/":
C#
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
// implementation
}
Visual Basic
<WebService(Namespace:="http://microsoft.com/webservices/")> Public Class MyWebService
' implementation
End Class
Done
Local intranet

Figure III-3

The file name for the service help page is specified in the <wsdlHelpGenerator> XML element of a configuration file with a default setting of **DefaultWsdlHelpGenerator.aspx**. By default, this particular ASP.NET Web Form page is common to all XML Web services on that machine:

```
\%WINDOWS%
  \Microsoft.NET
    \Framework
      \[version]
        \CONFIG
          \DefaultWsdlHelpGenerator.aspx
```

Because the service help page is simply an ASP.NET Web Form, it can be replaced or modified to include items, such as a company logo. Alternatively, you can modify the <wsdlHelpGenerator> element of the associated **web.config** file to specify a custom service help page.

Service Method Help Page

The service method help page provides additional information that relates to a particular XML Web service method. The page provides the ability to invoke the method using the HTTP-POST protocol. Subsequently, you cannot invoke methods that require complex types for input parameters, such as a dataset. In addition, the XML Web service method must support the HTTP-POST protocol to enable this functionality. At the bottom of the page, the service method help page provides sample request and response messages for the protocols the XML Web service method supports.

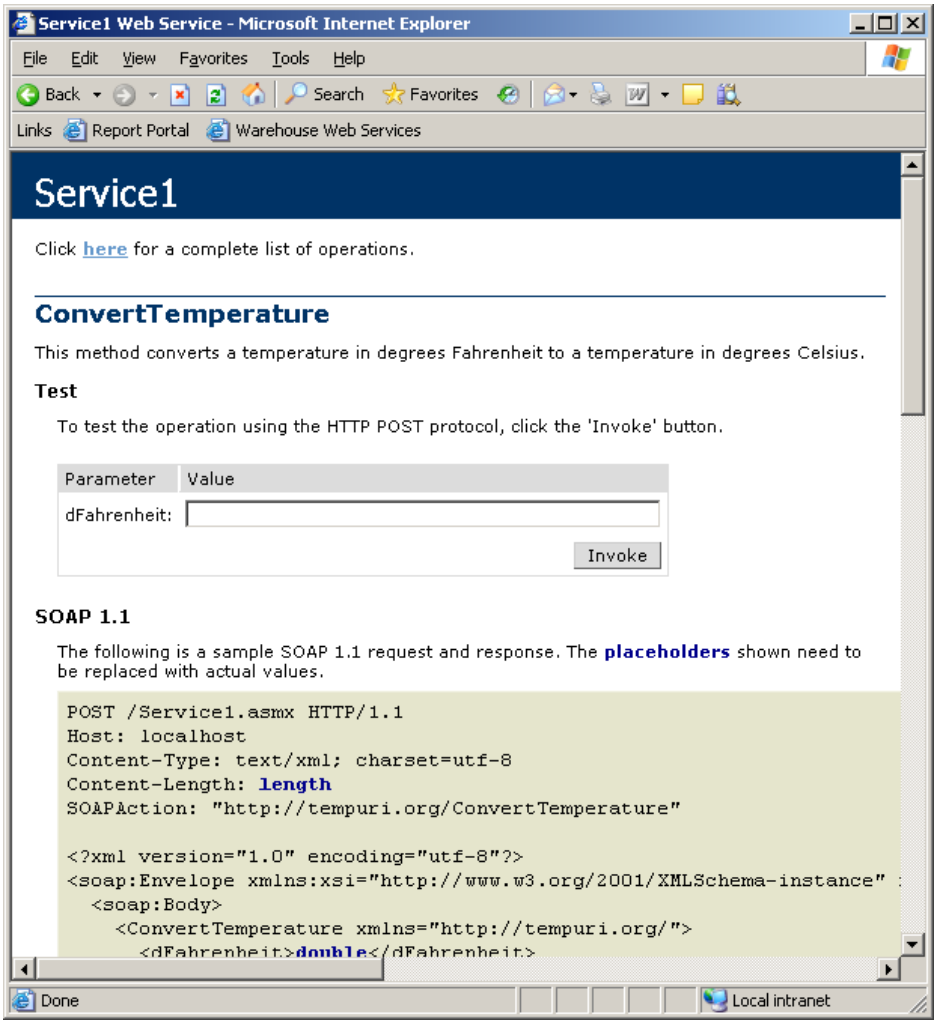

Figure III-4

For example, to access an XML Web service method named MyWebMethod on Service1.asmx that is part of a project named WebService1 on your local machine, you would use the following URL:
http://localhost/TempConvert1/Service1.asmx?op=MyWebMethod

Web Service Description (.wsdl)

The service help page also provides a link to the XML Web service's service description, which is a formal definition of the XML Web service's capabilities. The service description is a document that uses the Web Services Description Language (WSDL) grammar. The service description defines the contract for the message formats that clients need to follow when exchanging messages with the XML Web service.

XML Web Service Discovery

XML Web service discovery is the process of locating and interrogating XML Web service descriptions, which is a preliminary step for accessing an XML Web service. Through the discovery process, XML Web service clients can learn at design time that an XML Web service exists, what its capabilities are, and how to properly interact with it.
However, a Web site that implements an XML Web service need not support the discovery process. Instead, another site could be responsible for describing the service, such as an XML Web services

directory. Alternatively, there may not be a public means of finding the service, such as in the case of a web service developed for use by an internal application.

Static Discovery (.disco)

You can enable programmatic discovery of an XML Web service by publishing a .disco file, which is an XML document that contains links to other discovery documents, XML Schemas, and service descriptions. XML Web services created using ASP.NET automatically can provide a generated discovery document.

Web Service Directories

After deploying your XML Web service, you need to consider how to make it possible for developers to locate it if you intend for others to use it. An established method for advertising the availability of an XML Web service is to register it in a directory of XML Web services. The Universal Description, Discovery and Integration (UDDI) project provides a directory of businesses and the services they provide.

Appendix IV

Web Service with JAX-WS development Guidelines

Web Service with JAX-WS development Guidelines

JAX-WS stands for Java API for XML Web Services. JAX-WS is a technology for building web services and clients that communicate using XML. JAX-WS allows developers to write message-oriented as well as RPC-oriented web services.

In JAX-WS, a remote procedure call is represented by an XML-based protocol such as SOAP. The SOAP specification defines the envelope structure, encoding rules, and conventions for representing remote procedure calls and responses. These calls and responses are transmitted as SOAP messages (XML files) over HTTP.

Although SOAP messages are complex, the JAX-WS API hides this complexity from the application developer. On the server side, the developer specifies the remote procedures by defining methods in an interface written in the Java programming language. The developer also codes one or more classes that implement those methods. Client programs are also easy to code. A client creates a proxy (a local object representing the service) and then simply invokes methods on the proxy. With JAX-WS, the developer does not generate or parse SOAP messages. It is the JAX-WS runtime system that converts the API calls and responses to and from SOAP messages.

With JAX-WS, clients and web services have a big advantage: the platform independence of the Java programming language. In addition, JAX-WS is not restrictive: a JAX-WS client can access a web service that is not running on the Java platform, and vice versa. This flexibility is possible because JAX-WS uses technologies defined by the World Wide Web Consortium (W3C): HTTP, SOAP, and the Web Service Description Language (WSDL). WSDL specifies an XML format for describing a service as a set of endpoints operating on messages.

Creating a Simple Web Service and Client with JAX-WS

Figure IV-1 illustrates how JAX-WS technology manages communication between a web service and client.

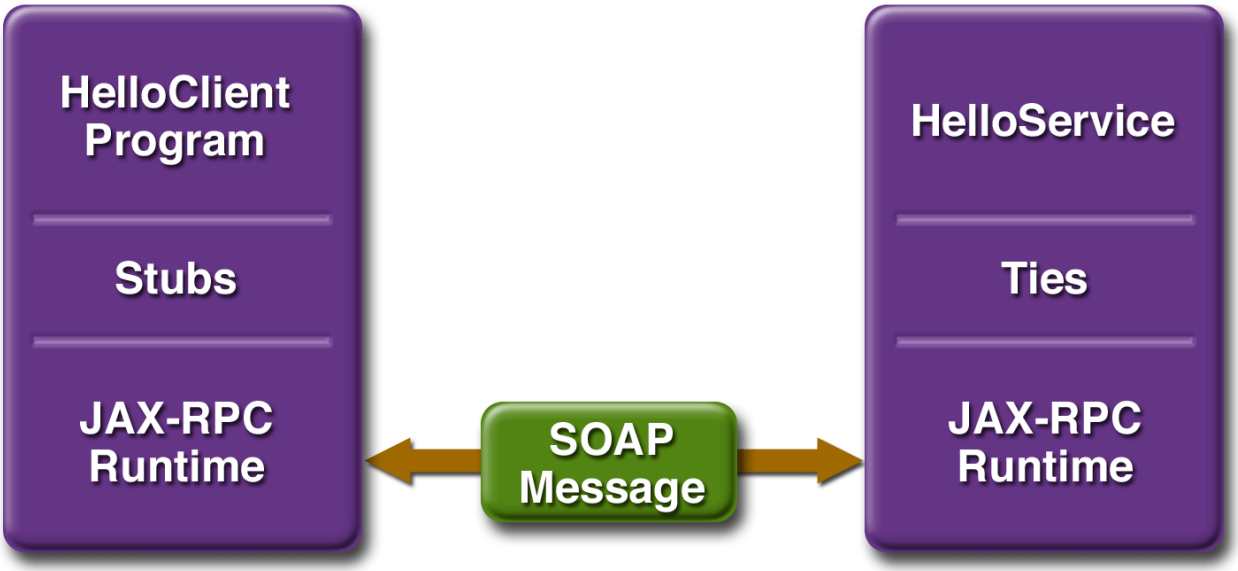

Figure IV-1

The starting point for developing a JAX-WS web service is a Java class annotated with the **javax.jws.WebService** annotation. The **WebService** annotation defines the class as a web service endpoint.

A service endpoint interface (SEI) is a Java interface that declares the methods that a client can invoke on the service. An SEI is not required when building a JAX-WS endpoint. The web service implementation class implicitly defines a SEI. You may specify an explicit SEI by adding the **endpointInterface** element to the **WebService** annotation in the implementation class. You must then provide a SEI that defines the public methods made available in the endpoint implementation class.

You use the endpoint implementation class and the **wsgen** tool to generate the web service artifacts and the stubs that connect a web service client to the JAXWS runtime. For reference documentation on **wsgen**, see the Application Server man pages at http://java.sun.com/j2ee/1.4/docs/tutorial/doc/About.html Together, the **wsgen** tool and the Application Server provide the Application Server's implementation of JAX-WS. These are the basic steps for creating the web service and client:

- Code the implementation class.
- Compile the implementation class.
- Use **wsgen** to generate the artifacts required to deploy the service.
- Package the files into a WAR file.
- Deploy the WAR file. The tie classes (which are used to communicate with clients) are generated by the Application Server during deployment.
- Code the client class.
- Use **wsimport** to generate and compile the stub files.
- Compile the client class.
- Run the client.

Requirements of a JAX-WS Endpoint

JAX-WS endpoints must follow these requirements:

- The implementing class must be annotated with either the **javax.jws.WebService** or **javax.jws.WebServiceProvider** annotation.
- The implementing class may explicitly reference an SEI through the **endpointInterface** element of the @**WebService** annotation, but is not required to do so. If no endpointInterface is not specified in @**WebService**, an SEI is implicitly defined for the implementing class.
- The business methods of the implementing class must be public, and must not be declared static or final.
- Business methods that are exposed to web service clients must be annotated with **javax.jws.WebMethod**.
- Business methods that are exposed to web service clients must have JAXB- compatible parameters and return types. See Default Data Type Bindings.
- The implementing class must not be declared final and must not be abstract.
- The implementing class must have a default public constructor.
- The implementing class must not define the finalize method.

- The implementing class may use the **javax.annotation.PostConstruct** or **javax.annotation.PreDestroy** annotations on its methods for lifecycle event callbacks. The @PostConstruct method is called by the container before the implementing class begins responding to web service clients.The @PreDestroy method is called by the container before the endpoint is removed from operation.

Coding the Service Endpoint Implementation Class

In this example, the implementation class, Hello, is annotated as a web service endpoint using the @WebService annotation. Hello declares a single method named sayHello, annotated with the @WebMethod annotation. @WebMethod exposes the annotated method to web service clients. sayHello returns a greeting to the client, using the name passed to sayHello to compose the greeting.

The implementation class also must define a default, public, no-argument constructor.

```
package helloservice.endpoint;

import javax.jws.WebService;

@WebService()
public class Hello {
private String message = new String("Hello, ");
public void Hello() {}
@WebMethod()
public String sayHello(String name) {
return message + name + ".";
}
}
```

Web Services Interoperability and JAXWS

JAX-WS 2.0 supports the Web Services Interoperability (WS-I) Basic Profile Version 1.1. The WS-I Basic Profile is a document that clarifies the SOAP 1.1 and WSDL 1.1 specifications in order to promote SOAP interoperability. For links related to WS-I, see Further Information (page xxiv).

To support WS-I Basic Profile Version 1.1, JAX-WS has the following features:

- The JAX-WS runtime supports doc/literal and RPC/literal encodings for services, static stubs, dynamic proxies, and DII.

Developing Web Services with Java 2 Platform

The Java 2 Platform, Enterprise Edition (J2EE) version 1.4 has evolved to integrate web services. Web services are now one of the many service delivery channels of the J2EE platform; existing J2EE components can be easily exposed as web services. Many benefits of the J2EE platform are available for web services, including portability, scalability, reliability, and no single-vendor locks-in. For example, J2EE containers provide transaction support, database connections, life cycle management, and other services that are scalable and require no code from application developers.

The J2EE 1.4 platform provides comprehensive support for web services through the JAX-RPC 1.1 API, which can be used to develop service endpoints based on SOAP. JAX-RPC 1.1 provides interoperability with web services based on the Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP). The J2EE 1.4 platform also supports JSR 109 that builds upon JAX-RPC and focuses on the programming model for implementing web services, as well as deploying web services in the J2EE 1.4 platform. In addition, J2EE 1.4 supports the WS-I Basic Profile to ensure that web services developed using the J2EE platform are portable not only across J2EE implementations, but are also interoperable with any web service developed, using any platform that conforms to the WS-I standards.

To develop web services, a developer usually needs extensive knowledge of XML-based standards and protocols (such as WSDL and SOAP), as well as a fair amount of programming experience. In the J2EE 1.4 platform, however, developing web services is seamless, since you don't really need to know about WSDL and SOAP; the mapping between the Java language and such XML-based standards is handled by the web service runtime system, thus freeing the developer from such low-level programming details.

Overview of Web Services

Web services are application components that are designed to support interoperable machine-to-machine interaction over a network. This interoperability is gained through a set of XML-based open standards, such as the Web Services Description Language (WSDL), the Simple Object Access Protocol (SOAP), and Universal Description, Discovery, and Integration (UDDI). These standards provide a common and interoperable approach for defining, publishing, and using web services. If you don't know anything about these technologies yet, no worries: you can develop web services in the J2EE 1.4 platform without knowing a thing about these XML-based standards and protocols. Figure III-2 shows how the Java APIs for XML Registries (JAXR) and Java APIs for XML Remote Procedure Calls (JAX-RPC) play a role in publishing, discovering, and using web services.

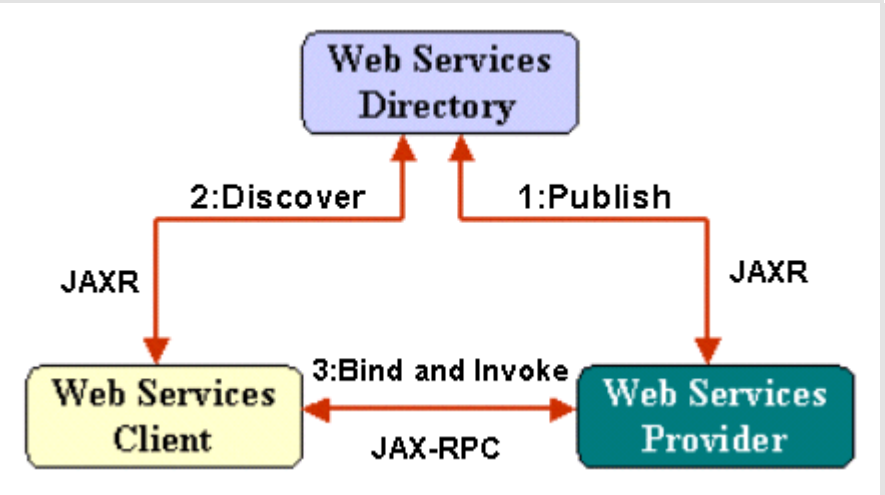

Figure IV-2 : J2EE 1.4 Publish-Discover-Invoke model

From a software architect's point of view, a web service can be considered as a service-oriented architecture, which consists of a collection of services that communicate with each other (and end-user clients) through well-defined interfaces. One advantage of service-oriented architecture is that it allows the development of loosely coupled applications that can be distributed and accessed, from any client, across the network.

The J2EE 1.4 SDK provides the tools you need to quickly build, test, and deploy web services and clients that interoperate with other web services and clients running on Java technology-based or non-Java

technology-based platforms. In addition, it enables businesses to expose their existing J2EE applications as web services. Servlets and Enterprise JavaBeans (EJBs) components can be exposed as web services that can be accessed by Java technology-based or non-Java technology-based web service clients. J2EE applications can act as web service clients themselves, and they can communicate with other web services, regardless of how they are implemented.

J2EE 1.4 SDK

The J2EE 1.4 SDK includes the following:

- J2EE 1.4 Application Server
- Java 2 Platform, Standard Edition (J2SE) 1.4.2_01
- J2EE Samples (Java Pet Store, Java Adventure Builder, Smart Ticket, and others)
- Sun ONE Message Queue
- PointBase Database Server

By default, the J2EE 1.4 SDK will be installed (on Microsoft Windows) at *c:\sun\AppServer*, and the JDK 1.4.2_01 would be installed at c:\sun\AppServer\jdk.

To configure your installation, just edit your path to include c:\sun\AppServer\bin and *c:\sun\AppServer\jdk\bin*. The *c:\sun\AppServer\bin* gives you access to several tools, including wscompile, which takes the service definition interface and generates the client-side stubs or server-side skeletons, or a WSDL description for the provided interface.

JSR 109

The process of developing and deploying web services is coupled with the runtime system. For example, deploying a web service on Apache Axis is different from deploying the same web service on Apache SOAP or any other platform. The Java Community Process (JCP) specification JSR 109 (Implementing Enterprise Web Services) promotes building portable and interoperable web services in the J2EE 1.4 environment. JSR 109 leverages J2EE technologies to provide an industry standard for developing and deploying web services on the J2EE platform, and it provides a service architecture that is familiar to J2EE developers. This specification outlines the lifecycle of web services to include:

- **Development**: Standardizes the web services programming model as well as the deployment descriptors
- **Deployment**: Describes the deployment actions expected of a J2EE 1.4 container
- **Service publication**: Specifies how the WSDL is made available to clients
- **Service consumption**: Standardizes the client deployment descriptors and a JNDI lookup model

J2EE Web Services

JAX-RPC is a Java API for XML-based Remote Procedure Calls (RPC). You can use it to build web services and clients that use RPC and XML. An RPC is represented using an XML-based protocol such as SOAP, which defines an envelope structure, encoding rules, and convention for representing RPC calls and responses, which are transmitted as SOAP messages over HTTP. The advantage of JAX-RPC is that it hides the complexity of SOAP messages from the developer. Here is how it works:

The developer specifies the remote procedures (web services) that can be invoked by remote clients in a Java programming language interface; the developer implements the interface. The client view of a web

service is a set of methods that perform business logic on behalf of the client. A client accesses a web service using a Service Endpoint Interface as defined by JAX-RPC. Client developers create the client -- a proxy (or a local object that represents the remote service) that is automatically generated -- and then simply invoke the methods on the proxy. The developer doesn't need to worry about generating or parsing SOAP messages; this is all taken care of by the JAX-RPC runtime system. Note that J2EE web services can be invoked by any web service client, and any J2EE web service client can invoke any web service.

To get a feeling for what happens behind the scenes, consider Figure IV-3, which shows how a Java client communicates with a Java web service in the J2EE 1.4 platform. Note that J2EE applications can use web services published by other providers, regardless of how they are implemented. In the case of non-Java technology-based clients and services, the figure would change slightly. As mentioned earlier, all the details between the request and the response happen behind the scenes. You only deal with typical Java programming language semantics, such as method calls and data types. You need not worry about mapping Java to XML and vice-versa, or constructing SOAP messages. All this low-level work is done behind the scenes, allowing you to focus on the high-level issues.

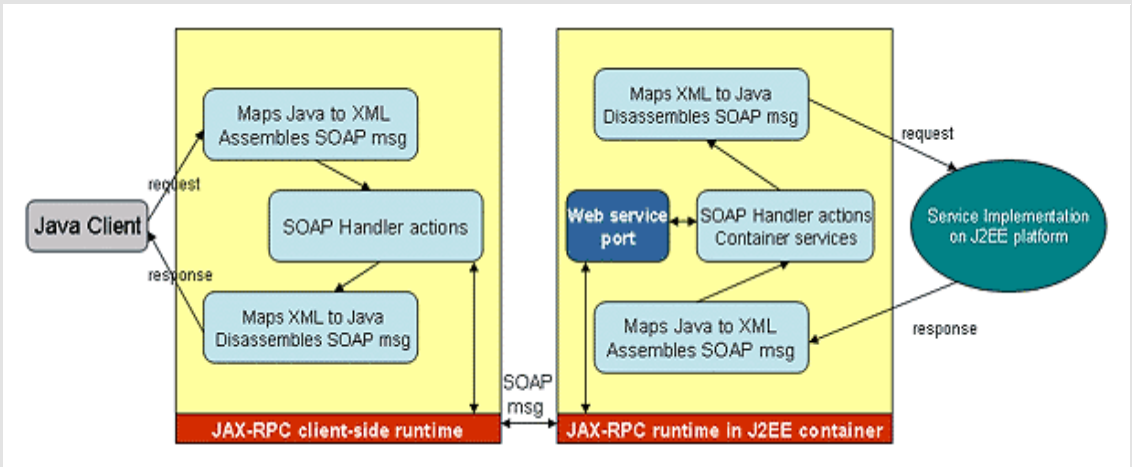

Figure IV-3: A Java client calling a J2EE web service

Note that a web service client never accesses a service directly; it does so through the container. This is a good thing, since it allows a web service to benefit from the added functionality that the container provides -- such as security, enhanced logging, and quality-of-service guarantees. This means that the container frees the developer from worrying about such low-level details.

Working with JAX-RPC

When working with JAX-RPC, remember that it maps Java types to XML/WSDL definitions. The good news is that you don't need to know the details of these mappings, but you should be aware that not all J2SE classes can be used as method parameters or return types in JAX-RPC. JAX-RPC supports the following primitive data types: *Boolean*, *byte*, double, *floats*, *int*, *long*, *short*, and *arrays*. In addition, it supports the following wrapper and utility classes:

java.lang.Boolean
java.lang.Byte
java.lang.Double
java.lang.Float
java.lang.Integer
java.lang.Long
java.lang.Short
java.lang.String
java.math.BigDecimal
java.math.BigInteger
java.net.URI
java.util.Calendar
java.util.Date

JAX-RPC also supports something called a value type, which is a class that can be passed between a client and a service as a parameter or a return value. A value type must follow these rules:

- It must have a public default constructor.
- It must not implement java.rmi.Remote.
- Its fields must be JAX-RPC supported types. Also, a public field cannot be final or transient, and a non-public field must have the corresponding getter and setter methods.

Creating Web Services - Sample

Building an XML-RPC style web service using the J2EE 1.4 platform involves five steps:

- Design and code the web service endpoint interface.
- Implement the interface.
- Write a configuration file.
- Generate the necessary files.
- Use the deploytool to package the service in a WAR file and deploy it.

Design and Code the Service Endpoint Interface

The first step in creating a web service is to design and code its endpoint interface, in which you declare the methods that a web service remote client may invoke on the service. When developing such an interface, ensure that:

- It extends the java.rmi.Remote interface
- It does not have constant declarations such as public static final
- Its methods throw the java.rmi.RemoteException (or one of its subclasses)
- Its method parameters and return data types are supported JAX-RPC types

To get started, create a directory of your choice. For the following example, I created an apps directory under c:\sun\AppServer and a subdirectory of apps called build. The apps directory contains the .java files, and the build directory contains the .class, as well as other files that will be automatically generated.

The web service I will be developing for the rest of this article provides a summation service for adding two numbers. Nothing fancy, but as you will see, it will demonstrate how to develop, deploy, and use web services. Code Sample 1 shows the service endpoint interface, which is a regular Java language interface that extends the java.rmi.Remote interface. The MathFace interface declares one method, add, which takes two integer values and returns an integer value representing the sum of the two integer parameters:

Code Sample 1: MathFace.java

```
package math;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface MathFace extends Remote {
  public int add(int a, int b) throws RemoteException;
}
```

Implement the Service Endpoint Interface

The next step is to implement the MathFace interface defined in Code Sample 1, which is quite straightforward, as shown in Code Sample 2.

Code Sample 2: MathImpl.java

```
package math;

import java.rmi.RemoteException;

public class MathImpl implements MathFace {
  public int add(int a, int b) throws RemoteException {
    return a + b;
  }
}
```

Now, compile the .java files specifying the .class files to be written to the build directory created above. The -d option instructs the compiler to write the output .class files into the build directory:

prompt> javac -d build Math*.java

Write a Configuration File

The next step is to define a configuration file to be passed to the wscompile tool. Here I call the file config.xml (and save it under the apps directory). In this configuration file, I describe the name of the service, its namespace, the package name (math in this case) and the name of the interface (MathFace). Code Sample 3 shows the configuration file:

Code Sample 3: config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service
     name="MyFirstService"
     targetNamespace="urn:Foo"
     typeNamespace="urn:Foo"
     packageName="math">
     <interface name="math.MathFace"/>
  </service>
</configuration>
```

This file tells wscompile to create a WSDL file with the following information:

- The service name is MyFirstService.
- The WSDL namespace is urn:Foo.
- The classes for the service are in the math package under the build directory.
- The service endpoint interface is math.MathFace.

Generate the Necessary Mapping Files

Now, use the wscompile tool to generate the necessary files. Consider the following command executed from the apps directory:

prompt> wscompile -define -mapping build/mapping.xml -d build -nd build -classpath build config.xml

This command, which reads the config.xml file created earlier, creates the MyFirstService.wsdl file and mapping.xml in the build directory. The command line options or flags are:

- -define: instructs the tool to read the service endpoint interface and create a WSDL file.
- -mapping: specifies the mapping file and where it should be written.
- -d and -nd: specifies where to place generated output files and non-class output files, respectively.

Believe it or not, you have now built a web service that is ready to be packaged and deployed.

The WSDL file MyFirstService.wsdl, generated by the wscompile tool, is shown in Code Sample 4. This file provides an XML description (based on WSDL) of the service that clients can invoke. To understand the details of the file you need some knowledge of WSDL -- but you don't need to understand all the details, so don't worry if you have no knowledge of WSDL.

Code Sample 4: MyFirstService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="MyFirstService" targetNamespace="urn:Foo" xmlns:tns="urn:Foo"
  xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
 <types/>
 <message name="MathFace_add">
  <part name="int_1" type="xsd:int"/>
  <part name="int_2" type="xsd:int"/></message>
 <message name="MathFace_addResponse">
  <part name="result" type="xsd:int"/></message>
 <portType name="MathFace">
  <operation name="add" parameterOrder="int_1 int_2">
   <input message="tns:MathFace_add"/>
   <output message="tns:MathFace_addResponse"/></operation></portType>
 <binding name="MathFaceBinding" type="tns:MathFace">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
  <operation name="add">
   <soap:operation soapAction=""/>
   <input>
    <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               use="encoded" namespace="urn:Foo"/>
   </input>
   <output>
    <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               use="encoded" namespace="urn:Foo"/>
   </output>
  </operation></binding>
 <service name="MyFirstService">
  <port name="MathFacePort" binding="tns:MathFaceBinding">
   <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
  </port>
 </service>
</definitions>
```

The mapping file, mapping.xml, generated by the wscompile tool is shown in the below table. This file follows the JSR 109 standard for Java <-> WSDL mappings. As you can see, the structure of the JAX-RPC mapping file matches closely with the structure of a WSDL file -- note the relationship between Java packages and XML namespaces. Each service offered is represented as a service-interface-mapping element. This element contains the mapping for the fully qualified class name of the service interface, WSDL service names, and WSDL port names. In addition, the JAX-RPC mapping file provides

mappings for WSDL bindings, WSDL port types, WSDL messages, and so forth. And once again, the good news is that you needn't worry about the WSDL file (Code Sample 4) of the JAX-RPC mapping file (Code Sample 5) in order to develop, deploy, and use web services.

Code Sample 5: mapping.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<java-wsdl-mapping version="1.1" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://www.ibm.com/webservices/xsd/j2ee_jaxrpc_mapping_1_1.xsd">
 <package-mapping>
  <package-type>math</package-type>
  <namespaceURI>urn:Foo</namespaceURI>
 </package-mapping>
 <package-mapping>
  <package-type>math</package-type>
  <namespaceURI>urn:Foo</namespaceURI>
 </package-mapping>
 <service-interface-mapping>
  <service-interface>math.MyFirstService</service-interface>
  <wsdl-service-name
       xmlns:serviceNS="urn:Foo">serviceNS:MyFirstService</wsdl-service-name>
  <port-mapping>
   <port-name>MathFacePort</port-name>
   <java-port-name>MathFacePort</java-port-name>
  </port-mapping>
 </service-interface-mapping>
 <service-endpoint-interface-mapping>
  <service-endpoint-interface>math.MathFace</service-endpoint-interface>
  <wsdl-port-type xmlns:portTypeNS="urn:Foo">portTypeNS:MathFace</wsdl-port-type>
  <wsdl-binding xmlns:bindingNS="urn:Foo">bindingNS:MathFaceBinding</wsdl-binding>
  <service-endpoint-method-mapping>
   <java-method-name>add</java-method-name>
   <wsdl-operation>add</wsdl-operation>
   <method-param-parts-mapping>
    <param-position>0</param-position>
    <param-type>int</param-type>
    <wsdl-message-mapping>
     <wsdl-message
            xmlns:wsdlMsgNS="urn:Foo">wsdlMsgNS:MathFace_add</wsdl-message>
     <wsdl-message-part-name>int_1</wsdl-message-part-name>
     <parameter-mode>IN</parameter-mode>
```

```
      </wsdl-message-mapping>
    </method-param-parts-mapping>
    <method-param-parts-mapping>
      <param-position>1</param-position>
      <param-type>int</param-type>
      <wsdl-message-mapping>
        <wsdl-message
                    xmlns:wsdlMsgNS="urn:Foo">wsdlMsgNS:MathFace_add</wsdl-message>
        <wsdl-message-part-name>int_2</wsdl-message-part-name>
        <parameter-mode>IN</parameter-mode>
      </wsdl-message-mapping>
    </method-param-parts-mapping>
    <wsdl-return-value-mapping>
      <method-return-value>int</method-return-value>
      <wsdl-message
                  xmlns:wsdlMsgNS="urn:Foo">wsdlMsgNS:MathFace_addResponse</wsdl-message>
      <wsdl-message-part-name>result</wsdl-message-part-name>
    </wsdl-return-value-mapping>
  </service-endpoint-method-mapping>
 </service-endpoint-interface-mapping>
</java-wsdl-mapping>
```

Packaging and Deploying the Service

A JAX-RPC web service is really a servlet (or a web component, in J2EE terminology), and hence you can use deploytool to package and generate all the necessary configuration files, and deploy the service. If you haven't yet used the deploytool to package and deploy applications, start the J2EE application server (or default domain) and then follow these instructions to package and deploy the math service. For the rest of the article, I will assume that the service can be accessed using the URL http://localhost:8080/math-service/math.

Creating Web Service Clients

Now, let's create a client that accesses the math service you have just deployed. A client invokes a web service in the same way it invokes a method locally. There are three types of web service clients:

Static Stub: A Java class that is statically bound to a service endpoint interface. A stub, or a client proxy object, defines all the methods that the service endpoint interface defines. Therefore, the client can invoke methods of a web service directly via the stub. The advantage of this is that it is simple and easy to code. The disadvantage is that the slightest change of web service definition lead to the stub being useless... and this means the stub must be regenerated. Use the static stub technique if you know that the web service is stable and is not going to change its definition. Static stub is tied to the implementation. In other words, it is implementation-specific.

Dynamic Proxy: Supports a service endpoint interface dynamically at runtime. Here, no stub code generation is required. A proxy is obtained at runtime and requires a service endpoint interface to be instantiated. As for invocation, it is invoked in the same way as a stub. This is useful for testing web services that may change their definitions. The dynamic proxy needs to be re-instantiated but not re-generated as is the case with stub.

Dynamic Invocation Interface (DII): Defines javax.xml.rpc.Call object instance for dynamic invocation. Unlike stub and proxy, it must be configured before it can be used. A client needs to provide: operation name, parameter names, types, modes, and port type. As you can tell, much more coding is involved here. The major benefit is that since Call is not bound to anything, there is no impact of changes on the client side (whenever the web service definition changes). The DII client is outside the scope of this article.

Static Stub Client

Let's develop a stand-alone client that calls the add method of MyFirstService. It makes the call through a stub, or a local object that acts as a client proxy to the remote service. It is called a static stub because the stub is generated before runtime by the wscompile tool. Before developing the Java client itself, you need to write a configuration file (in XML) that describes the location of the WSDL file (MyFirstService.wsdl). The configuration file is shown in Code Sample 6. I suggest that you create a new subdirectory under apps call it static-stub (where you save the .xml and .java files) and under that a build subdirectory.

Code Sample 6: config-wsdl.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
      xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl location="http://localhost:8080/math-service/math?WSDL"
     packageName="sstub"/>
</configuration>
```

As you can see, the URL http://localhost:8080/math-service/math?WSDL identifies the location of the WSDL file for MyFirstService. If you try this URL, you'd see something similar to Figure IV-4 (assuming you have deployed the web service developed above).

Figure IV-4: MyFirstService.wsdl

Click to Enlarge

Once you have written the configuration file, you're ready to generate client stubs, using the following command:

prompt> wscompile -gen:client -d build -classpath build config-wsdl.xml

This commands reads the MyFirstService.wsdl (the location of which is specified in the config-wsdl.xml), then generates files based on the information in the WSDL file and on the command-line flags. The -gen:client instructs wscompile to generate the stubs, as well as other needed runtime files such as serializers and value types. The -d flags tells the tool to write the output to the build subdirectory.

Now, you're ready to develop the client. A sample implementation is shown in Code Sample 7. Note that the stub generated earlier is used, and a service endpoint address is used to locate the service http://localhost:8080/math-service/math. As you can see, a stub object is created using the MathFirstService_Impl object generated by the wscompile tool; the endpoint address that the stub uses to access the service is set; and then the stub is cast to the MathFace service endpoint interface; finally, the add method is invoked.

Code Sample 7: MathClient.java

```
package sstub;

import javax.xml.rpc.Stub;

public class MathClient {

  private String endpointAddress;

  public static void main(String argv[]) {
    try {
      // Invoke createProxy() to create a stub object
      Stub stub = createProxy();

      // Set the endpoint address the stub uses to access the service
      stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
                        "http://localhost:8080/math-service/math");

      // Cast the stub to the service endpoint interface (MathFace)
      MathFace math = (MathFace) stub;

      // Invoke the add method
      System.out.println(math.add(12, 24));
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }

  private static Stub createProxy() {
    // Create a stub object
    // Note that MyFirstService_Impl, generated by wscompile, is implementation-specific
    return (Stub) (new MyFirstService_Impl().getMathFacePort());
  }
}
```

You can now compile MathClient.java and write the output to the build directory:

C:\sun\APPSER~1\apps\static-stub> javac -classpath build -d build MathClient.java

Run the client:

C:\sun\APPSER~1\apps\static-stub> java -classpath build sstub.MathClient

Dynamic Proxy

The client in this case calls a remote procedure through a dynamic proxy or a class that is created at runtime. Note that in the case of the static stub, the code relied on an implementation-specific class, but here (dynamic proxy) the code doesn't have this limitation. The first step is to create a configuration file. You can use the one in Code Sample 6 but make sure you change the packageName to dynamicproxy or whatever you like. Again, create a directory dynamic-proxy under apps, and build subdirectory under dynamic-proxy.

Now, use the wscompile to generate the needed interfaces:

C:\Sun\APPSER~1\apps\dynamic-proxy> wscompile -import -d build -nd build -f:norpc structures -classpath build config-wsdl.xml

Now, develop the client. Code Sample 8 provides a sample implementation. Here, an instance of the service factory is created. The service factory is used to create a service object that acts as a factory for proxies. As you can see, the createService method takes two parameters: a URL of the WSDL files and a QName object, which is a tuple that represents an XML qualified name -- the namespace URI and the local part of the qualified name (the service name). A proxy object is then created, and finally the add method is invoked on that object.

Code Sample 8: MathClient.java

```
package dynamicproxy;

import java.net.URL;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;
import dynamicproxy.FirstIF;


public class MathClient {

  public static void main(String[] args) {
    try {
      String nameSpaceUri = "urn:Foo";
      String serviceName = "MyFirstService";
      String portName = "MathFacePort";

      // Specify the location of the WSDL file
      URL url = new URL("http://localhost:8080/math-service/math?WSDL");

      // Create an instance of service factory
      ServiceFactory serviceFactory = ServiceFactory.newInstance();

      // Create a service object to act as a factory for proxies.
      Service mathService = serviceFactory.createService(url,
              new QName(nameSpaceUri, serviceName));

      // Create a proxy
      dynamicproxy.MathFace
              myProxy = (dynamicproxy.MathFace) mathService.getPort(new
                                      QName(nameSpaceUri,
              portName), dynamicproxy.MathFace.class);

      // Invoke the add method
      System.out.println(myProxy.add(23, 12));
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

Compile the client:

c:\Sun\APPSER~1\apps\dynamic-proxy> javac -classpath build -d build MathClient.java

And, now run the client:

C:\Sun\APPSER~1\apps\dynamic-proxy> java -classpath build dynamicproxy.MathClient

Command-Line J2EE Application Client

Unlike a stand-alone client, a J2EE application client can obtain a service interface using JNDI lookup. In this case, the client developer defines a logical JNDI name (service reference) for the web service. The container binds the service interface implementation under the client's environment context java:comp/env using the logical name of the service reference. The following snippet of code shows an example:

```
try {
  Context ic = new InitialContext();
  MyFirstService myFirstService = (MyFirstService)
     ic.lookup("java:comp/env/service/MyFirst");
  j2ee.MathFace mathPort = myFirstService.getMathFacePort();
  ((Stub) mathPort)._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY,
     "http://localhost:8080/math-service/math");
  System.out.println(mathPort.add(22, 1));
} catch (Exception ex) {
  ex.printStackTrace();
  System.exit(1);
}
```

Browser-Based Client

Finally, let's see how to develop a web client in which the web service is invoked from a browser-based form (here we use static stub). To start, you need a configuration file similar to the one in Code Sample 6. Just change the packageName to whatever you like; here, it's webclient. Once you have that, generate the client stub using the following command:

c:\Sun\APPSER~1\aps\web\>wscompile -gen:client -d build -classpath build config-wsdl.xml

The next step is to write the web client as a servlet or a JavaServer Pages technology page (JSP). Code Samples 9 and 10 show my JSP implementation of the web client. Code Sample 9 presents the form to the user and retrieves the parameters, and Code Sample 10 (based on static stub) is similar to the client in Code Sample 7.

Code Sample 9: math.jsp

```
<html>
<head><title>Hello</title></head>
<body bgcolor="#ffcccc">
<h2>Welcome to the Math Web Service</h2>
<form method="get">
<input type="text" name="num1" size="25">
<p></p>
<input type="text" name="num2" size="25">
<p></p>
<input type="submit" value="AddNumbers">
<input type="reset" value="Reset">
</form>

<%
   String str1 = request.getParameter("num1");
   String str2 = request.getParameter("num2");
   if ( str1 != null && str2 != null ) {
%>
   <%@include file="add.jsp" %>
<%
   }
%>
</body>
</html>
```

Code Sample 10: add.jsp

```
<%@ page import="javax.xml.rpc.Stub,javax.naming.*,webclient.*" %>
<%
   String resp = null;
   int result = 0;

   try {
     // Create a stub object
     Stub stub = (Stub)(new MyFirstService_Impl().getMathFacePort());

     // Set the endpoint address the stub uses to access the service
     stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
           "http://localhost:8080/math-service/math");
     // Cast the stub to MathFace
     MathFace math = (MathFace)stub;
     // Retrieve and parse the parameters from the request message
     int x = Integer.parseInt(request.getParameter("num1"));
     int y = Integer.parseInt(request.getParameter("num2"));
     // Invoke the method
     result = math.add(x, y);

   } catch (Exception ex) {
   }
%>
<h2>The sum is: <%=result%></h2>
```

The next step is to deploy the web client as a JSP web component using the deploytool. Again, if you haven't used deploytool, starts the J2EE application server (if it is not already running) and follow these instructions to package and deploy the web client as a JSP web component. During deployment, I specified this URL http://localhost:8080/math-webservice/add to be used to access the service.

Further Information

For more information and related technologies, refer to the following:
• Simple Object Access Protocol (SOAP) 1.2 W3C Note
http://www.w3.org/TR/SOAP/
• Web Services Description Language (WSDL) 1.1 W3C Note
http://www.w3.org/TR/wsdl
• WS-I Basic Profile 1.1
http://www.ws-i.org

Appendix V

Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA)

Effective integration of a customer-facing Portal with a constantly growing range of service specific back-end systems is probably the biggest challenge for e-Government initiatives. While even the most complex commercial systems need to integrate with a finite and typically well-known from the start set of systems, e-Government solutions usually face a bigger challenge. They must provide a platform for integrating a larger, and to some degree unknown set of current and future services. Making the integration easier and avoiding disruption to existing services is a major factor for the successful adoption of an e-Government solution.

Various options for integrating separate systems (which could be running on different platforms and software stacks) are possible. Choosing the most appropriate option depends on the specific constraints, type of integration and interoperability required, and other factors.

A relatively recent concept in the world of interoperability is that of the service oriented architecture (SOA). The underlying principle behind the concept of an SOA is the idea that IT systems, software, devices and services will integrate and "talk" to each other—even if they were never specifically designed for each other in the first place.

An SOA is implemented using Web services, and applications are constructed as sets of re-useable, co-operating services with each being responsible for one or more clearly identified and bounded user tasks, business processes or information services.

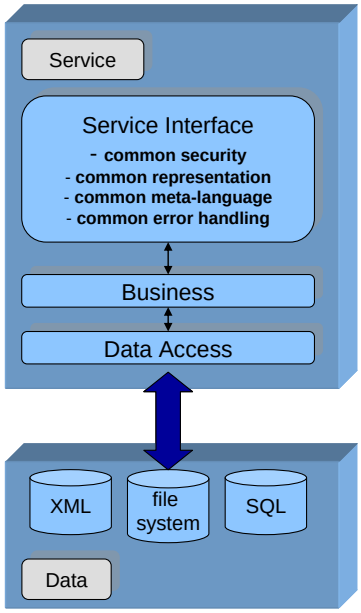

Figure V-1
<Source: Microsoft>

The service end-points that are made available in an SOA use Web services, which in turn are exposed using standards such as:

- **XML**: the eXtensible Markup Language, which provides vendor-independent data interoperability between systems
- **SOAP**: the Simple Object Access Protocol, which provides the syntax for accessing services
- **WSDL**: the Web Services Description Language, which effectively provides the contract for Web services, setting out what inputs are expected and what outputs will be supplied

The SOA model relies upon industry standards, which enable services to run on many different platforms and ensure that they can still interoperate and communicate with each other. Services can be delivered to user interfaces that run on any platform or device capable of talking to the underlying service interfaces.

The only requirement of the consuming application is its ability to comply with and use the standards involved.

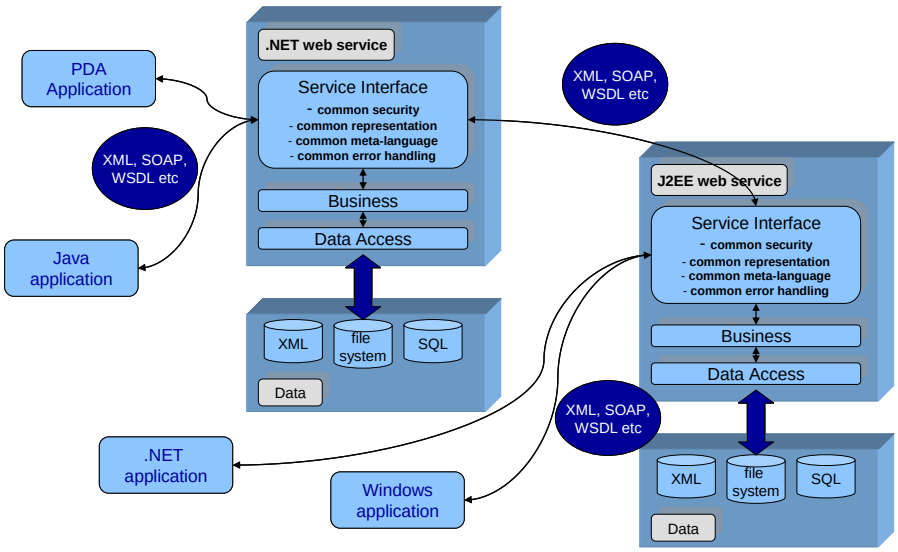

Figure V-2
<Source: Microsoft>

Agency Web Services must be implemented as SOAP 1.1 Web Services, using XML to interchange data, and expose its interface using WSDL.

Adopting a Service Oriented Architecture (SOA) approach to the implementation of large, complex software systems like e-Government integration solutions, allows effective addressing of the fundamental challenge of ensuring that these systems are adaptable, flexible, and reliable in the face of change.

Service Oriented Architecture (SOA) is a framework, and a set of policies and practices and that enable application functionality implemented as published services at a granularity relevant to the service requestor. They abstract the implementation by using a single, standards-based form of interface.

The functionality of the e-Government integration platform should be available as a set of generic services, available for use independently and as required.

The most important goal of the SOA is to encourage ***loose coupling*** between services.
This means that two services should have the minimum possible knowledge of each other while still being able to communicate. For example, to call a service it is always necessary to know Agency interface and contract. However, in a loosely-coupled architecture, the consumer should not have any knowledge of the service's location, business rules, programming language, database or operating system. As a result, it is possible to change any of these aspects of a service without impacting Agency consumers. This provides substantial flexibility and can dramatically lower maintenance costs.
The separation of the service's interface from the black box implementation provides the basis for loose coupling between services. Some additional considerations for loose coupling are as follows:

- A service must only call another service via Agency interface. Specifically, a service cannot directly access another service's state (for example, from a database or through global variables) or business logic.
- A service should not rely on the existence of a third party, such as a transaction coordinator or authentication service, for a call to succeed. A major implication of this is that ACID transactions

cannot be implemented across service boundaries. In a service boundary, relying on external infrastructure is acceptable.
- A service must not require that Agency caller also calls other known services, in order for the call to succeed – for example, Service A must not insist that the caller calls service B first.
- A service should not know or care which applications or services are consuming it.

Granularity

In order for the SOA to be effective, services should be defined at a relatively coarse level of granularity. At the very fine-grained level, loose coupling is unhelpful, and using services would give an unacceptable performance impact. At the extremely coarse-grained level, services would be too specific to a single context to be reusable. The best approach is somewhere in the middle, where a complex system can be broken up into reusable, loosely-coupled services. Of course, this is a very subjective statement, and extensive experience and analysis may be required to 'get it right'.

- ***Benefits of the SOA***

Some of the main benefits arising from the SOA are as follows:

- *REUSE*

  Reuse has long been considered one of the 'holy grails' of software engineering, as it has the potential to substantially lower development and testing costs by leveraging code from existing systems. However, most approaches to date, such as Component-Based Design (CBD), have failed to deliver the promised amount of reuse. While it would be premature to claim that SOA finally delivers on this promise, the more coarsely-grained approach of defining services makes reuse more achievable.

- *AGILITY*

  Agility is all about being able to respond to changing business requirements and a changing environment. The loose coupling and the separation of a service's interface from Agency implementation makes the SOA particularly conducive to supporting an agile enterprise. In a well-designed SOA, it is possible to change business rules, make major implementation changes or even move a service from one platform to another, with minimal or no changes to other systems depending on the service.

- *HETEROGENEOUS ENVIRONMENT FRIENDLY*

  It is considered inevitable that large enterprises will end up with a heterogeneous environment, with a variety of hardware platforms, operating systems, development platforms or application servers. These differences become irrelevant in a SOA, when used with Web services protocols such as HTTP, SOAP and WSDL. The end result is that services can be built using the most needed platform for the job, rather than being forced to use a particular platform to ensure interoperability.

Appendix VI

Service Management Compliance For eServices

Service Management Compliance For eServcies

Executive Summary

The State of Kuwait represented by The Central Agency for Information Technology (CAIT) has embarked on an ambitious project to deliver high quality eServices to citizens, residents, businesses and visitors. In order to guarantee reliability, security, availability, and the continuity (among others) of such eServices, and since such eServices are created by agencies, CAIT is planning to start an eServices Compliance Assessment program as part of its overall e-government initiative, in order to extend those eServices to the public.

Objectives

In order to ensure that such services are adhering to high international standards in terms of reliability, availability, security and performance just to name a few requirements, and because of the high number of eServices that need to be assessed for compliance, CAIT has chosen to base the Compliance Assessment on the de facto standard for ITSM and its leading framework which is ITIL.
The following processes should be evaluated and scored according to the Capability Maturity Model Integrated (CMMi) recommended by *it*SMF:

- Incident Management
- Problem Management
- Change Management
- Configuration Management
- Release Management
- Service Level Management
- Availability Management
- Capacity Management
- IT Service Continuity Management

eServices Compliance Assessment program boundaries and guide lines

At this stage of the Project, the eService compliance assessment program is considered as OPTIONAL but Highly Recommended. This will be a mandate at later stages of the maturity time line of the KGO eServices Integration program.
Agencies seeking integration of their eServices to the KGO are required to engage with itSMF and obtain compliance report that the eServices are managed according to International standards and best practices. The *it*SMF is the international, unbiased body that is responsible for the ITSM standards and has already a Chapter in the Gulf Region through *it*SMF Gulf.
This will ensure enforcing international common standards and best practices for service availability and service management for the delivery and compliance of such eServices .
Agencies will subsidize this activity on their own sole discretion and provide CAIT with the results of the Compliance Assessment Reports.
By creating such a model, current as well as future services can be aligned with CAIT's requirements at an early stage, thus shortening delivery times, and meeting the required level of service excellence that everyone wants to achieve.
The suggested compliance assessment stages should be:

- Readiness Assessment
- Service Compliance Assessment
- Continual Improvement

Approach

CAIT is suggesting a phased approach based on proven methodologies, and global standards, taking into account required degrees of customization and localization.

Phase 1 – Readiness Assessment

Agencies must be ready and able to manage the services, this assessment will cover:

- Process maturity rating against ITIL best practices
- Service Level Management
- Assessment of available competencies
- Tools and Technology

Deliverables:

- Gap analysis
- Communication Plan Outline
- Roles and Responsibilities
- Estimated resources and anticipated allocations
- Deliverables for each selected process to achieve specific maturity characteristics
- ITIL Awareness and Executive workshops

In addition to the above, itSMF as an International compliance assessment body will help agencies achieve the necessary state of readiness through the adoption of the relevant ITSM and ITIL processes.

Phase 2 – Service Compliance Assessment

Compliance assessment will cover the necessary activities and checklists to assess the compliance as stated in the ITSM standards. This process would entail the following steps:

- Assessment: creating a baseline
- Gap analysis: addressing what needs to be done
- Provisional Certification: Final audit to give preliminary certification
- Final Certification: will ensure that the processes have matured enough and have become embedded within the organization.

Deliverables:

- Compliance Assessment Reports

In addition to the service centric nature of this assessment, some other audits will be done on the organization/agency to ensure other requirements are met, as deemed necessary by the client.

Phase 3 – Post Implementation Support

The Post Implementation support can extend after successful project implementation to cover the following:

- Quarterly Health Checks
- Annual re-assessment
- Ongoing Education
- Continual Improvement Strategy
- Optionally ISO20000 Readiness Assessment

Appendix VII

eServices Questionnaire

eServices Questionnaire

This questionnaire has been designed in an easy way to assist agencies in deciding and choosing the best possible offering to comply with the KGO standards and to identify any challenges during the course of the implementation.
It is divided into three sections:

Section 1. Existing Service Details

Section 2. Existing Electronic Implementation of this service

Section 3. Implementation Challenges

Questionnaire (for each service)

Section 1. Existing Service Details

1. Please provide the name of the service

2. Please provide Service ID “If applicable”

3. Please identify the target users of this service.

Please select accordingly

- ☐ Citizens
- ☐ Residents
- ☐ Visitors
- ☐ Commercial Establishment
- ☐ Government Agency

4. Please identify the estimated number of transactions expected to be performed by this service within any one month period?

Please select only one option

- ☐ Less than 1,000
- ☐ Between 1,000 to 10,000
- ☐ Between 10,001 to 50,000
- ☐ Between 50,001 to 250,000
- ☐ More than 250,000

5. Please identify the periods (times) where there is high service usage

Please select all valid options

- ☐ Summer time (June – September)
- ☐ End of Month
- ☐ Beginning of Month
- ☐ New Calendar Year
- ☐ Month of Ramadan
- ☐ Other, Please specify

Section 2. Existing Electronic Implementation of this service

6. Does your organization offer its eService through an existing website/portal?

Please select only one option

☐ Yes

☐ No

7. Please provide the URL to this electronic implementation of the Service if available.

8. What is the current online maturity level of this service delivery?

Please select those options applicable

☐ Online information of e-service

☐ Online forms for download

☐ Online submission of forms

☐ Online payment of services

☐ Integration with Backend systems for processing request

☐ Integration with external agencies (if any) for processing request

9. Is authentication required to use the electronic service?

Please select only one option

☐ No

☐ Yes (Please complete the following)

a) Please identify the user authentication mechanism

Please select accordingly

☐ User account login

☐ Certificate based authentication

b) What User Directory Service does your organization use to maintain user credentials?

Please select accordingly

☐ LDAP

☐ Active Directory

☐ Database

☐ Others (Please indicate)

10. Please identify the application platform upon which this Service is implemented electronically or you would use to deliver this service electronically.

Please fill in the details for 10a to 10f.

a) Web Environment (Websphere, Weblogic, IIS, Cold Fusion, other…)

b) Web server (provide details of Hardware platform and Operating System)

c) Database used (Oracle, MS SQL, MySQL, DB2, etc…), operating system and Version no.

d) Do you use Package software to deliver the service? (If yes, provide the details of the package software and version no.)

e) Programming language (Java, C#, etc…)

f) Framework (J2EE, .NET, etc…)

11. Do you have in-house IT staff to provide support? *Please select only one option*

☐ No

☐ Yes (Please complete the following)

Please provide the approximate no. of IT staff (wherever applicable) for the following:

Production / Maintenance support for e-service delivery

Operations support

Network support

12. Please identify the network architecture upon which this Service is implemented electronically or you would use to deliver this service electronically. *Please fill in the details for 12a to 12d*

    a) Firewalls

    b) ISP hosting DMZ

    c) Proxy Server(s)

    d) Internet Connection speed

Section 3. Implementation Challenges

13. In which flavor does your organization/agency plan to deliver the eService to be integrated with the KGO? "To answer this section user MUST read and fully comprehend the "KGO Embedded eService & Web Services based eServices" section in the attached standard document"

- ☐ KGO Embedding
- ☐ Web Services based eServices "If your agency choose this option, then please answer the next question"

14. This question is only intended for answer by agencies that chose the option "Web Services based eServices" in the previous question. To implement an eService through the KGO Service Delivery Framework, Web-services exposed through SOAP over HTTPS is required. What is the estimated timeframe required for you to provide a Service Endpoint through a Web-service interface to support this Service?

Please select only one option

- ☐ Not applicable
- ☐ Web-Service interfaces are already available.
- ☐ [ ] Weeks (Please specify an estimate number of weeks)
- ☐ [ ] Months (Please specify an estimate number of months)
- ☐ Unable to give an estimate.

15. What are the difficulties/ challenges your Agency encounter or may expect to encounter in implementing the service through the KGO Portal?

Please select accordingly

- ☐ None
- ☐ Lack of staff with the ability or skills required to support the eService implementation, receptiveness i.e. amount of retraining required
- ☐ Requires scope of change in processes required
- ☐ Lack of technical staff to support the implementation of the eService through the KGO Portal
- ☐ Processes within the organization require users to sign the application submitted for processing
- ☐ Operational support, including call center
- ☐ Others (Please elaborate)

[ ]

16. What are the difficulties your Agency encounter / or may encounter in implementing technological components to support the implementation of the eService.

Please select accordingly

- ☐ None
- ☐ Some difficulty but still able to implement
- ☐ Lack of technological expertise within the organization
- ☐ Current state of computerization & IT infrastructure and the potential technology gap within the organization
- ☐ Others (Please elaborate)

•
•

17. What are the investments needed to implement this service online?

Please select accordingly

- ☐ *Not applicable*
- ☐ *Requirements are in the process of being implemented*
- ☐ *Need to invest in hardware and software*
- ☐ *Need to hire additional manpower*
- ☐ *Components required for integration with KGO Portal need to be developed*
- ☐ *No additional investments required Web pages are available for hyper-linking*
- ☐ *No additional investments required Web-Services are already available*
- ☐ *Other (Please elaborate)*

•
•

Appendix VIII

eServices Integration

Known Issues Resolution

Introduction

In some cases, Web Applications require storing and retrieving some session related data, using different techniques such as (Session State, Cookies, Application State, etc.).When embedding such applications in a FRAME (like the ones deployed for the KGO), and browsing it using Internet Explorer, the variables saved in sessions and cookies are blocked/lost, which gives incorrect response to the user. For that, we have included the issue and its resolution for future use and herein identified as a known issue. Details of the issue and the resolution are mentioned below;

Issue

If you implement a FRAMESET whose FRAMEs point to other Web sites on the networks of your partners or inside your network, but you use different top-level domain names, you may notice in Internet Explorer 6 that any cookies you try to set in those FRAMEs appear to be lost. This is most frequently experienced as a loss of session state in an Active Server Pages (ASP) or ASP.NET Web application. You try to access a variable in the **Session** object that you expect to exist, and a blank string is returned instead.

Resolution

You can add a P3P compact policy header to your child content, and you can declare that no malicious actions are performed with the data of the user. If Internet Explorer detects a satisfactory policy, then Internet Explorer permits the cookie to be set.

Definition of P3P

P3P is the Platform for Privacy Preferences Project (P3P) enables Websites to express their privacy practices in a standard format that can be retrieved automatically and interpreted easily by user agents. P3P user agents will allow users to be informed of site practices (in both machine- and human-readable formats) and to automate decision-making based on these practices when appropriate. Thus users need not read the privacy policies at every site they visit.

To Add P3P Compact Policy, one of the following methods can be followed:

1- Set the header by using the **Response.AddHeader** method in an ASP page. In ASP.NET, you can use the **Response.AppendHeader** method.

2- Use the IIS Management Snap-In (**inetmgr**) to add to a static file. By following the steps below:

a. Click **Start**, click **Run**, and then type **inetmgr**.

Figure VIII-1

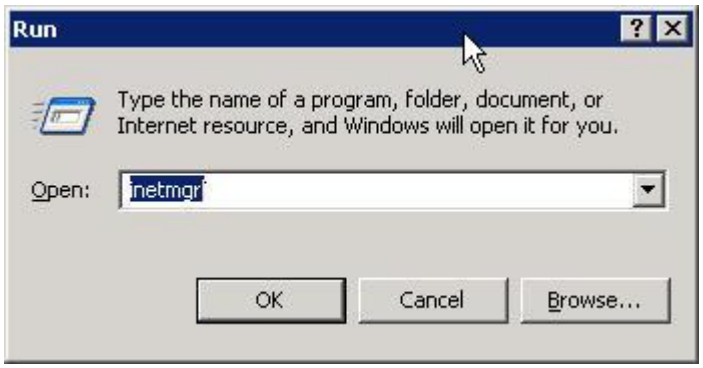

Figure VIII-2

b. In the left navigation page, click the appropriate file or directory in your Web site to which you want to add the header, right-click the file, and then click **Properties**.

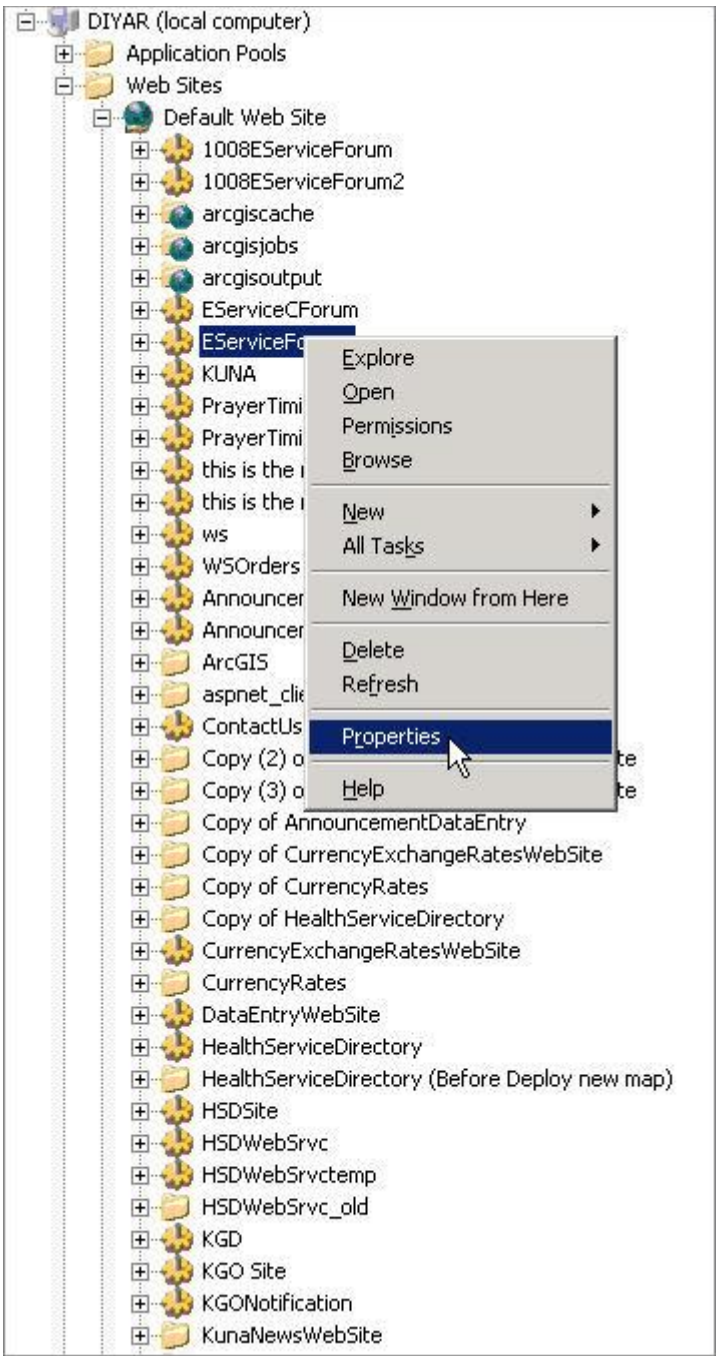

Figure VIII-3

c. Click the HTTP Headers tab.

d. In the **Custom HTTP Headers** group box, click **Add**.

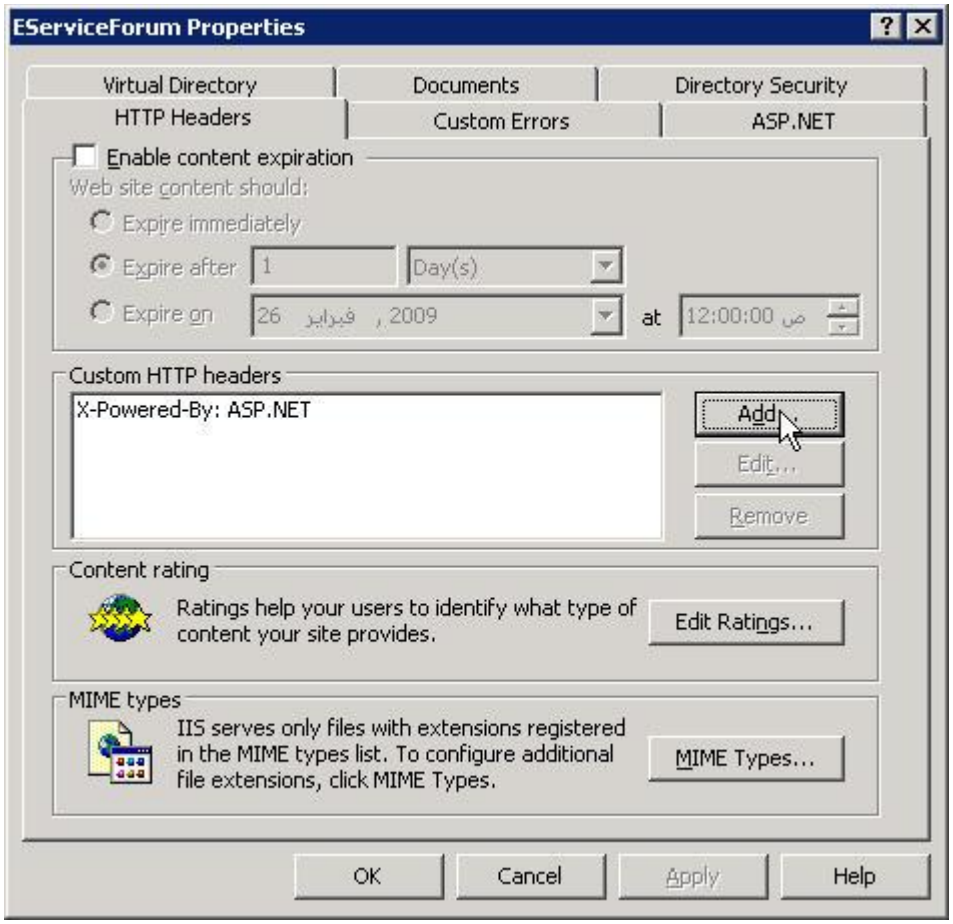

Figure VIII-4

e. Type **P3P** for the header name, and then for the compact policy string, type the appropriate code for your compact policy , in our case **CP = "CAO PSA OUR"** , then click **OK**

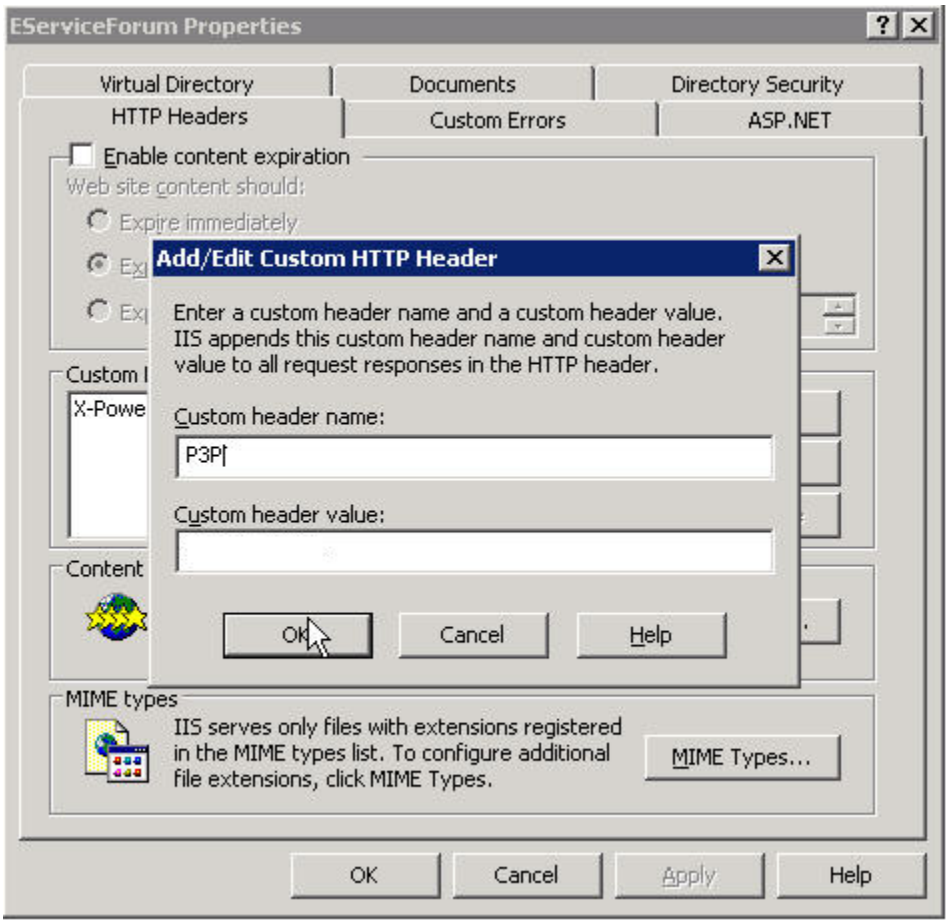

Figure VIII-5