CURSO

SERVICIOS WEB

Prólogo

La necesidad de reducir la brecha existente entre las etapas de aprendizaje en una institución educativa del nivel técnico, técnico superior universitario y superior y la de aplicación productiva en el mercado laboral, llevó a la ANIEI y al ILCE a diseñar el Modelo Paracurricular, en el que se procura la actualización inmediata y permanente de los profesionistas de las áreas de informática y computación. Motivo por el cual la tarea de "Formación de Entrenadores en Tecnologías de Información", se sustenta en el Modelo Paracurricular.

El modelo paracurricular es producto de diversos análisis y consultas, de las cuales surgieron las áreas de conocimiento a ser consideradas en la formación o entrenamiento del capital humano en nuestro país: Programación, Calidad, Sistemas Distribuidos y Desarrollo Empresarial, dichas áreas están incluidas en los cuatro perfiles del capital humano para la industria de Software, definidos en el modelo paracurricular: Emprendedor y Administrador de Proyectos de Software, Arquitecto de Software, Ingeniero de Software y Desarrollador de Software.

Para la generación de contenidos de los cursos se capacitó a los docentes para generar y revisar los contenidos de los cursos del modelo paracurricular y tutores de los mismos, en la modalidad e-learning.

En la primera fase de este subproyecto se inició con el perfil básico de la estructura de la industria del software, el *Desarrollador de Software*, y en la que se incluye el área de programación de servicios Web.

El presente material, ha sido diseñado para aprender Programación de Servicios Web, con un esquema de aprendizaje basado en problemas, en el que a través de escenarios, ejercicios prácticos y cuestionarios el participante se verá involucrado en construir y reflexionar las soluciones a partir de la colaboración e interacción con sus compañeros y el mismo tutor o profesor.

Los servicios Web ofrecen un nuevo paradigma de desarrollo para la construcción de aplicaciones distribuidas del Web. El curso trata las cuatro tecnologías base detrás de los servicios Web: XML, SOAP, WSDL y UDDI, denominado en conjunto comúnmente como la pila de interoperabilidad de los servicios Web; así como, su implementación y seguridad mediante el uso de plataformas de desarrollo como Microsoft .Net o Java de Sun, entre otras. Con la aplicación práctica de estos temas el participante estará en posibilidades de incursionar en las ofertas que demanda el proceso productivo de desarrollo de software en la actualidad.

Agradecimiento y Reconocimiento

Después de una ardua tarea de investigación se ha logrado la creación de una obra vasta en conocimiento en el desarrollo de las Tecnologías de la Información y Comunicación.

La presente obra no hubiera sido posible sin la valiosa aportación de destacados autores y especialistas en la materia. Es por ello que a manera de reconocimiento queremos agradecer su participación:

SERVICIOS WEB

Mtro. Sergio González Nava
Universidad La Salle

Lic. Lourdes Sánchez Guerrero
Universidad Autónoma Metropolitana

Intención Educativa

Contribuir al fortalecimiento de tu formación profesional en los temas relevantes de aplicaciones avanzadas del Web mundial, especialmente en el área de los servicios Web, los cuales se han popularizado por la facilidad de implementación en aplicaciones distribuidas. Con ello tendrás más y mejores oportunidades de incorporarte al mercado laboral.

Objetivos Generales

Al terminar el curso, el participante será capaz de:

- Entender y emplear los conceptos fundamentales de protocolos, arquitectura y seguridad de los servicios Web.
- Distinguir el funcionamiento de las tecnologías base, XML, SOAP, WSDL y UDDI, de los servicios Web.
- Diseñar e implementar una aplicación real, cliente Web, base de datos y servicios Web, mediante una plataforma de desarrollo conveniente.

- Apreciar y explotar las ventajas que ofrecen las plataformas de desarrollo dedicadas a los servicios Web.

Metodología

Método de Aprendizaje Basado en Problemas

El método se orienta a la adquisición de conocimientos como el desarrollo de tus habilidades y actitudes.

Se tiene como punto de partida la definición de problemas por resolver, recaudación de información, análisis de datos, la construcción de hipótesis y la evaluación así misma. Se promueve el análisis de problemas de la vida real, a fin de incrementar los niveles de comprensión, permitiendo utilizar tu conocimiento y habilidades.

En el recorrido que vivirás desde el planteamiento original del problema hasta su solución, trabajarás de manera cooperativa en pequeños grupos, compartiendo en esa experiencia de aprendizaje la posibilidad de practicar y desarrollar habilidades, de observar y reflexionar sobre actitudes y valores que en el método convencional expositivo difícilmente podrían ponerse en acción.
Los pasos que debes seguir son los siguientes:

1. Explora los temas:
   - El profesor o tutor te presenta un tema o escenario.
   - Discutan el trazado del problema y enlisten las partes significantes.
   - Quizás sientas que no sabes lo suficiente para resolver el problema pero ¡ese es el reto!, deberás reunir información y aprender nuevos conceptos, principios o habilidades a medida que avanzas en el proceso de resolución del problema.

2. Realiza una lista "¿Qué sabemos?":
   - ¿Qué sabes para resolver el problema?
   - Esto incluye tanto lo que sabes en realidad y que fortalezas y capacidades de cada miembro del equipo tiene.
   - Considera o apunta los aportes de todos, no importa que tan extraño pueda parecer: ¡Es una posibilidad!

3. Desarrolla y escribe, el trazado del problema con tus propias palabras:
   - Un trazo del problema debería salir del análisis de lo que saben y necesitan para resolverlo. Necesitarás:

  - El plan escrito.
  - El acuerdo de grupo sobre el plan.
- Retroalimentación del profesor acerca de este plan (puede ser opcional, pero es una buena idea).
- El trazo del problema es a menudo revisado y editado a medida que se descubre nueva información, o se descarta información “vieja”.

4. Realiza una lista de las posibles soluciones:
   - Realiza una lista de todas, luego ordénalas de la mayor a menor factible.
   - Elige la mejor, o la que es más factible que triunfe.

5. Realiza una lista de las acciones a ser tomadas con una línea del tiempo:
   - ¿Qué tenemos que saber y hacer para resolver el problema?
   - ¿Cómo ordenamos estas posibilidades?
   - ¿Cómo se relaciona esto con nuestra lista de soluciones?
   - ¿Estamos de acuerdo?

6. Realiza una lista "¿Qué necesitamos saber?"
   - Investiguen el conocimiento y los datos que respaldan su solución.
   - Necesitarán esta información para llenar los espacios faltantes.
     - Discutan posibles fuentes
     - Asignen y programen tareas de investigación, especialmente fechas límite.
     - Expertos, libros, sitios Web, etc.
   - Si la investigación respalda su solución y si hay acuerdo general, vaya a (7), sino vaya a (4).

7. Escriban su solución con su documentación que la respalda y entréguela:
   - Quizás necesiten presentar sus hallazgos y /o recomendaciones a un grupo de sus compañeros.
   - Estos deberían incluir el trazo del problema, preguntas, datos reunidos, análisis de datos, y respaldo para las soluciones o recomendaciones basadas en el análisis de datos: en breve, el proceso y los resultados.
   - Presentando y defendiendo sus conclusiones:
     - Establecer claramente tanto el problema como su conclusión.
     - Resuma el proceso que utilizo, las opciones consideradas y las dificultades que encontró.
     - Si es afrontado y tiene una respuesta, preséntala claramente y si no tiene una respuesta, reconózcalo y remítala a mayor consideración.
   - Compartir sus hallazgos con el profesor y compañeros es una oportunidad para demostrar lo que has aprendido. Si conoces bien el tema, será evidente.

Si se suscita una cuestión a la cual no puedes responder, acéptala como una oportunidad para ser explorada.

8. Reflexiona tu desempeño:
   - Este ejercicio de interrogación sirve tanto para el trabajo individual como para el grupo.
   - Siéntete orgulloso por lo que has hecho bien; aprende de lo que no has hecho bien.

Contenido

1. Introducción a los Servicios Web

1.1. Aplicación de servicios Web

El siempre cambiante escenario de los negocios depende cada día más del Web para las transacciones de datos y la comunicación entre aplicaciones. Debido a esta dependencia, en la visión de los programadores, además de crear aplicaciones para computadoras y aplicaciones Web, han puesto atención en crear aplicaciones distribuidas que permitan la integración de aplicaciones construidas en diferentes plataformas, así como la integración de datos de varias aplicaciones desarrolladas por distintas organizaciones o empresas. Esto es posible con la creación y uso de los servicios Web.

Para facilitar esta integración y el desarrollo de este tipo de aplicaciones, básicamente servicios Web, Microsoft ha introducido su plataforma .NET. Además de .NET existen otras arquitecturas que tienen el mismo objetivo como la arquitectura Java J2EE de Sun Microsystems y una serie de iniciativas para estandarizar esa integración, entre otras.

Los servicios Web apuntan a estar en la inmensa red global del Web, establecidos para una interacción humana y de software para realizar operaciones automáticas como:

- Buscar servicios para conseguir el mejor precio.
- Coordinar boletos de viaje y reservación de hotel para una fecha determinada.
- Procurar negociaciones dinámicas, de facturación y operaciones de envío.

En este curso, partiremos del escenario clásico de la “Agencia de Viajes” con objeto de realizar su implementación mediante infraestructura de servicios Web, empleando la plataforma .NET o alternativamente otras plataformas de desarrollo.

Aunque tratemos el escenario principal de la “Agencia de Viajes”, utilizaremos otros escenarios sencillos para reforzar la comprensión de los temas del contenido del curso.

1.1.1. Escenario Agencia de Viajes

Es una agencia de viajes que ha decidido ofrecer a sus clientes el beneficio de planear y reservar los arreglos de viaje a través de una aplicación basada en Web. Como parte de esta aplicación, la agencia pretende que la capacidad de la

aplicación permita a los socios de negocio (hoteles, aerolíneas y bancos) y clientes acceso programático a sus servicios mediante interfaz de servicios Web.

La agencia pretende lograr el desarrollo y despliegue de sus servicios Web, con el proceso básico siguiente (ver figura 1):

1. El cliente se conectará al servidor Web de la agencia de viajes, presentando su itinerario.

2. El servidor Web de la agencia de viajes se encargará de enviar la petición según el itinerario, a los servidores de los hoteles y aerolíneas, para consultar las posibilidades y disponibilidad de reservación.

3. Si la respuesta es exitosa se le presentan las ofertas al cliente.

4. Si el cliente acepta una de las ofertas, se le solicitará sus datos de tarjeta bancaria para la transacción con el banco identificado y se reservará automáticamente todo lo requerido.

5. Se le notifica al cliente de la realización exitosa del proceso y se le envía el número de confirmación de la reservación y los detalles últimos del itinerario.

6. Una vez que se notifica al cliente del éxito o fallo de su itinerario solicitado, puede presentar otra solicitud de viaje.

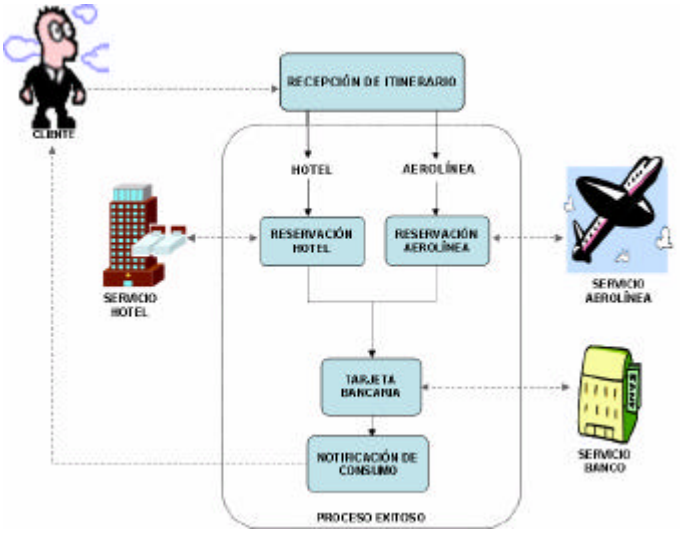


Figura 1. Proceso de Reservación

1.1.2. Actividades de aprendizaje

Implementación del escenario de la agencia de viajes

Con objeto de escenario de la agencia de viajes, se desarrollarán actividades relacionadas durante el curso con entregables al término de cada tema. Además, como parte de los entregables se incluyen las actividades resultantes de la aplicación de aprendizaje basado en problemas.

En seguida, se describen las actividades a realizar para el escenario de la agencia de viajes.

Instrucciones

1. Antes de iniciar el escenario, lea detenidamente el artículo del "Anexo A. Artículo sobre proactividad"

   - Tarea Individual: En base a esta lectura, describa que relación tiene con su participación en este curso.

2. Lea detenidamente el "Escenario Agencia de Viajes".

   - Tarea en Equipo: Determine un plan de trabajo para el análisis, diseño e implementación del escenario de la agencia de viajes con interfaz de servicios Web.

   - Dentro de las actividades del plan de trabajo debe considerar la dinámica del aprendizaje basado en problemas, descrito en la sección "Metodología".

   - Se recomiendan los enlaces de interés de la sección: "Anexo E. Fuentes de Información" así como la resolución de ejercicios y cuestionarios.

1.2. Las aplicaciones distribuidas y el Web

1.2.1. El World Wide Web: El Web

La aplicación distribuida mayor conocida es el World Wide Web (WWW), en corto denominado el Web. Técnicamente, el Web es un sistema distribuido de servidores y clientes HTTP (protocolo de transferencia de hipertexto), normalmente conocidos como los servidores Web y exploradores Web. La figura 2 muestra un esquema general de la arquitectura Web, en la que sus elementos se describen adelante.

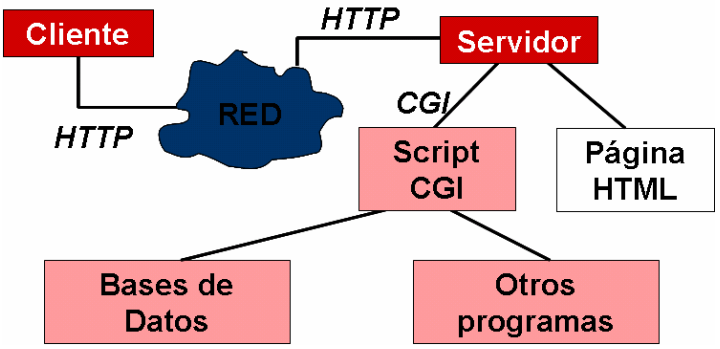


Figura 2: Esquema General de Arquitectura Web

Antes del suceso del Web, la comunidad de usuarios de Internet comprendida grandemente de investigadores y académicos usaron los servicios de red como correo electrónico y transferencia de archivo para intercambiar información.

El World Wide Web creado en 1990 en Ginebra Suiza, en el Laboratorio de Partículas Físicas Europeo. Una propuesta sometida por Tim Berners-Lee y Robert Cailliau para un "sistema de hipertexto universal."

Desde la propuesta original, el crecimiento del Web mundial ha sido extraordinario y se ha extendido más allá de la investigación y la comunidad académica en todos los sectores del mundo, incluso el comercio y casas privadas. El continuo desarrollo de la tecnología Web es actualmente coordinado por el World Wide Web Consortium, W3C.

La noción del Web mundial es que combina tres importantes y bien establecidas tecnologías de la computación:

- Documentos hipertexto: documentos en que palabras o frase elegidas, típicamente resaltadas, pueden marcarse como enlaces a otros documentos,

para que un usuario pueda tener acceso a los documentos vinculados haciendo clic con un ratón en el texto resaltado.

- Recuperación de información apoyada en la red: Mediante el protocolo de transferencia de archivos (FTP), que fue ampliamente usado.

- El lenguaje de marcado generalizado estándar (SGML), un estándar de ISO que permite el marcado de los documentos para que puedan desplegarse en un formato uniforme en cualquier plataforma, independiente de los mecanismos de la presentación.

El típico Web es la aplicación cliente-servidor, basada en HTTP:

- Un servidor Web es un servidor orientado a conexión que implementa el HTTP. Por omisión, un servidor HTTP se ejecuta en el muy conocido puerto 80.

- Un usuario ejecuta un cliente del Web (a veces llamado explorador) en una computadora local. El cliente interactúa con un servidor Web según el HTTP, especificando un documento para ser obtenido. Si el documento es localizado por el servidor en su directorio, los contenidos del documento se devuelven al cliente, el cual los presenta al usuario.

El Hypertext Markup Language: HTML

HTML es un lenguaje de marcado para crear documentos que pueden recuperarse usando el Web. HTML esta basado sobre SGML, con semánticas apropiadas para representar información de una amplia gama de tipos. La marca HTML puede representar hipertexto de noticias, correo, documentación e hipermedia; menús de opciones; resultados de consulta de base de datos; documentos estructurados simples con gráficos en línea, etc. Se muestra a continuación un ejemplo de código de HTML:

```
<HTML>
<HEAD>
<TITLE>A Sample Web Page</TITLE>
</HEAD>
<HR>
<BODY>
<center>
<H1>My Home Page</H1>
<IMG SRC="/images/myPhoto.gif">
<b>Welcome to Kelly's page!</b>
<p>
<! A list of hyperlinks follows.>
<a href="/doc/myResume.html"> My resume</a>.
```

```
<p>
<a href="http://www.someUniversity.edu/">My university<a>
</center>
<HR>
</BODY>
```

El eXtensible Markup Language: XML

Mientras HTML es un lenguaje que permite marcado de un documento para presentación o despliegue de su contenido, XML permite marcado del documento para información estructurada. También basado en SGML, XML usa las marcas para describir la información contenida en un documento. Se muestra a continuación un ejemplo de código XML:

```
<message>
    <to>you@yourAddress.com</to>
    <from>me@myAddress.com</from>
    <subject>This is a message</subject>
    <text>
       Hello world!
    </text>
</message>
```

El lenguaje XML será tratado a detalle en el tema 2, puesto que es elemental dentro del contexto de cómo se usa para definir e implementar los servicios Web.

1.2.2. Transferencia de contenidos Web

El Hipertext Transfer Protocol: HTTP

Originalmente concebido para obtener y desplegar archivos de texto, HTTP se ha extendido para permitir la transferencia de contenidos Web de tipos virtualmente casi ilimitados.

La primera versión de HTTP, HTTP/0.9, fue un protocolo simple para la transferencia datos en bruto. La versión de HTTP ampliamente usada es HTTP/1.0 que fue propuesta por Tim Berners Lee, aunque no tiene especificación formal.

Desde entonces, un protocolo mejorado, conocido como HTTP/1.1, se ha desarrollado y a menudo se ha adoptado. HTTP/1.1 es un protocolo más extenso que HTTP/1.0. Sin embargo, los fundamentos del protocolo son bien representados en el HTTP/1.0 más simple.

HTTP es orientado a conexión, sin estado, un protocolo de petición-respuesta. Un servidor HTTP o servidor Web, ejecuta por omisión en el puerto 80 de TCP. Los

clientes de HTTP, familiarmente llamados exploradores Web, son procesos a los que implementan interacción HTTP con un servidor Web para recuperar documentos expresados en HTML, cuyos contenidos se despliegan según el marcado de los documentos.

En HTTP/1.0, cada conexión permite sólo una ronda de petición-respuesta: un cliente obtiene una conexión, emite una petición; el servidor procesa la petición, emite una respuesta y cierra la conexión después de esto.

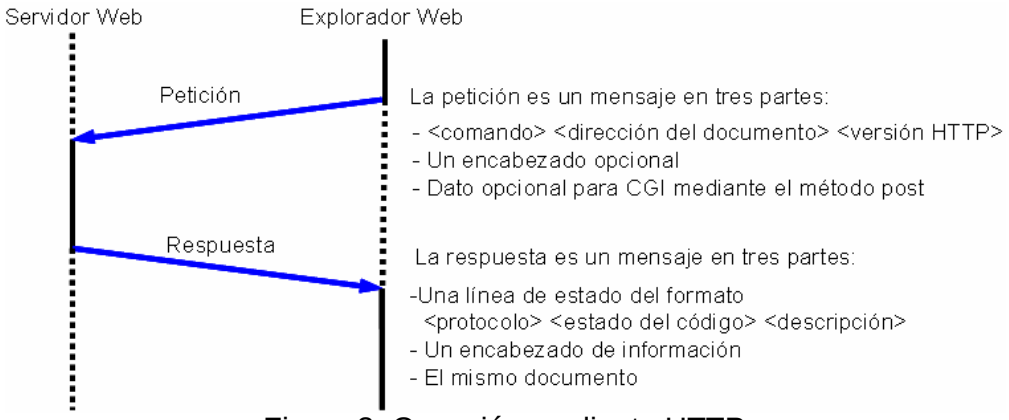


Figura 3: Conexión mediante HTTP

HTTP esta basado en texto: las peticiones y respuestas son cadenas de caracteres. Cada petición y respuesta está compuesta de las partes siguientes, en orden:

- Una línea de petición/respuesta
- Una sección de encabezado
- Una línea en blanco
- El cuerpo

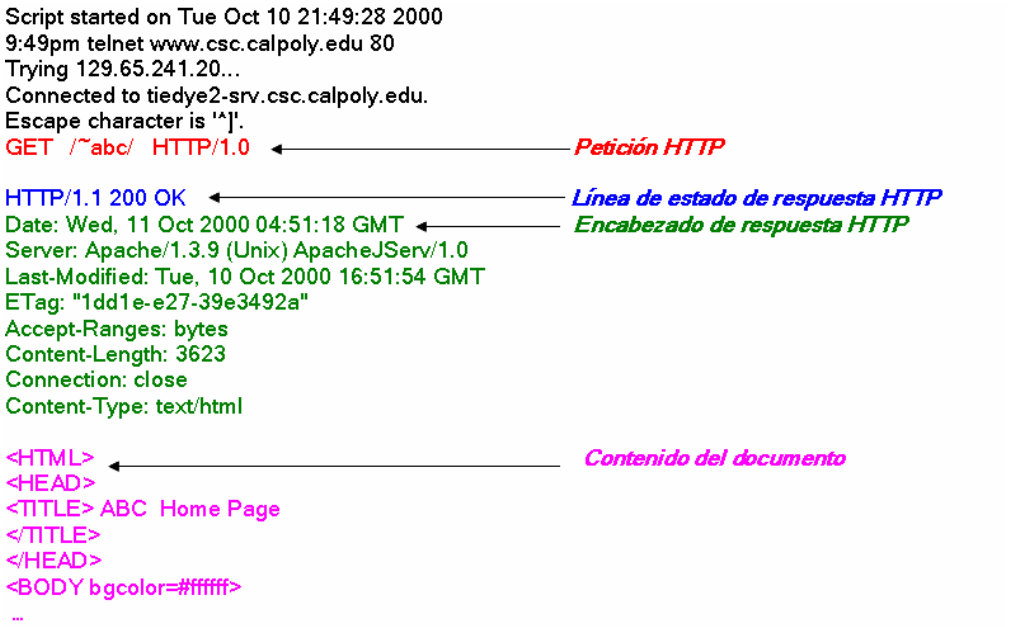


Figura 4: Un ejemplo de sesión HTTP

La petición HTTP

Una petición del cliente se envía al servidor después de que el cliente ha establecido una conexión al servidor.

Una línea de petición es de la forma siguiente:

```
<Método HTTP></><Petición-URI></><Especificación de protocolo>\r\n
```

Donde:

- <Método de HTTP> es el nombre de un método definido para el protocolo .
- <Petición-URI> es el URI de un documento Web o, más generalmente, un objeto del Web.
- <Especificación de protocolo > es una especificación del protocolo observada por el cliente.
- < / > es el carácter “/”.

Un ejemplo de petición del cliente es como sigue:

```
GET /index.html HTTP/1.0
```

El método HTTP

El método de HTTP en una petición del cliente es una palabra reservada (en mayúscula) que especifica una operación del servidor deseada por el cliente.

Algunos de los métodos de petición del cliente claves son los siguientes:

- GET: Recupera los contenidos del objeto Web referenciado por el URI especificado.
- HEAD: Recupera solo un encabezado del servidor, no el propio objeto.
- POST: Envía los datos a un proceso en el computador central del servidor.
- PUT: Solicita al servidor para almacenar los contenidos adjuntados con la petición para la máquina servidora en el directorio especificado por el URI.

El encabezado de petición

Los campos de encabezado de petición permiten al cliente pasar información adicional sobre la petición y sobre el propio cliente, al servidor. Estos campos actúan como modificadores de la petición, con semántica equivalente a los parámetros en una invocación de método (procedimiento) de un lenguaje de programación.

Un encabezado está compuesto de una o más líneas, cada línea en la forma de:

<Palabra clave>: <Valor>\r\n

Algunas de las palabras claves y valores que pueden aparecer en un encabezado de la petición son:

- Accept: tipos de contenido aceptable por el cliente.
- User-Agent: especifica el tipo de explorador.
- Connection: Puede especificarse para que el servidor no cierre una conexión inmediatamente después de enviar una respuesta.
- Host: nombre del sistema principal del servidor

Un ejemplo de encabezado de petición es como sigue:

```
Accept: */*
Connection: Keep-Alive
Host: www.someU.edu
User-Agent: Generic
```

El cuerpo de la petición

Una petición termina opcionalmente con un cuerpo de la petición que contiene datos que necesitan ser transferidos al servidor en asociación con la petición. Por ejemplo, si el método POST es especificado en la línea de petición, entonces el cuerpo contiene datos para ser pasados al proceso designado.

Ejemplos completos de la petición del cliente:

Ejemplo 1:
```
GET / HTTP/1.1
<blank line>
```

Ejemplo 2:
```
HEAD / HTTP/1.1
Accept: */*
Connection: Keep-Alive
Host: somehost.com
User-Agent: Generic
<blank line>
```

Ejemplo 3:
```
POST  /servlet/myServer.servlet HTTP/1.0
Content-type: application/x-www-form-urlencoded
Content-length: 34
Accept: */*
Connection: Keep-Alive
Host: somehost.com
User-Agent: Generic
<blank line>
Name=abc&email=abc@someU.edu
```

El Common Gateway Interface : CGI

En principio, HTTP fue empleado para transferir los contenidos estáticos, es decir, contenidos que existen en un estado constante, como un archivo de texto común o un archivo de imagen. Cuando el Web evolucionó, las aplicaciones empezaron a no usar HTTP para un propósito originalmente pensado: una aplicación que permite un usuario del navegador para recuperar datos se basa en información dinámica aplicada durante una sesión de HTTP.

Una aplicación Web típica, como el carrito de tienda, requiere la obtención de datos remotos basado en la elección del cliente, en tiempo de ejecución. Aplicado en el Web, es deseable permitir al cliente introducir datos durante una sesión del Web para recuperar los datos del servidor Web central y ser desplegado por el explorador Web (ver figura).

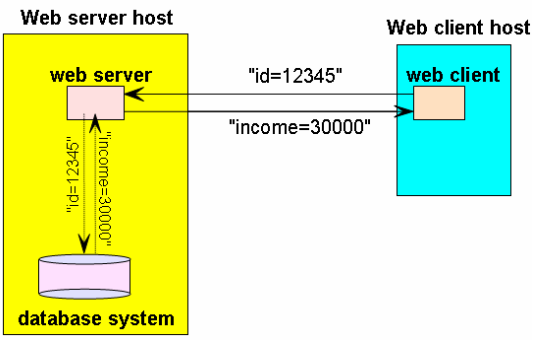


Figura 5: Web dinámico

Un servidor HTTP genérico no posee la lógica de aplicación para obtener datos de una base de datos. En cambio, un proceso externo que tiene la lógica de aplicación será un intermediario. El proceso externo ejecuta en el servidor central, acepta el dato de entrada desde el servidor Web, ejerce su lógica de aplicación para obtener los datos del origen de datos, devuelve el resultado al servidor Web que transmite el resultado al cliente.

El primer protocolo extensamente adoptado para aumentar a HTTP soporte para generar contenidos del Web en tiempo de ejecución, es el protocolo Common Gateway Interface (CGI). El CGI es un estándar para proporcionar una interfaz o una vía de acceso, entre un servidor de información y un proceso externo al servidor.

Con CGI, un cliente Web puede especificar un programa, guión CGI, como el objeto del Web destino en una petición de HTTP. El servidor Web obtiene el guión CGI, lo activa como un proceso, pasando al dato de entrada del proceso transmitido por el cliente Web. El guión Web ejecuta y transmite su salida al servidor del Web, el cual devuelve los datos generados del guión Web como el cuerpo de una respuesta al cliente Web.

En la práctica, un guión de CGI es invocado típicamente por un tipo especial de página Web conocida como Web Form, la cual acepta entradas en tiempo de ejecución e invocan un guión CGI que hace uso de lo que ha entrado.

Aunque rudimentario CGI es el predecesor de protocolos más sofisticados y medios que sirven para el propósito de acceder a programas o datos desde el servidor Web. En seguida se muestran varios de ellos:

- Microsoft Active Server Pages y ASPX.
- Java Servlets y Java Server Pages.

- Invocación de lenguajes interpretados (Perl, PHP, REXX, Python).

- API's propietarias de servidores Web (Microsoft ISAPI, Netscape NSAPI, etc.) y desarrollo de filtros, módulos, packages, cartdriges.

CGI y Microsoft ISAPI

Cada tipo de servidor proporciona diferentes métodos para ejecutar aplicaciones CGI. Los dos métodos soportados por IIS (Internet Information Services) de Microsoft son:

- Interfaz CGI.
  Una aplicación CGI es un ejecutable externo, que procesa las variables de entorno y la entrada estándar y genera los resultados enviándolos a la salida estándar. La programación de los CGI no es muy complicada, ya que con un compilador de C o similares y unas sencillas bibliotecas de funciones para procesar las variables de entorno se pueden crear CGI. Este método es el tradicional en UNIX.

- Interfaz ISAPI.
  Esta interfaz es una derivación del CGI pero que en vez de ejecutables externos utiliza bibliotecas dinámicas DLL para procesar la petición del usuario. Estas bibliotecas se registran en el servidor IIS y son cargadas permanentemente por el servidor al arrancar.

La interfaz ISAPI tiene la ventaja de que el procesamiento de las peticiones es mucho más rápido. Actualmente existen multitud de herramientas que nos permiten generar código ISAPI o CGI, e incluso ambos simultáneamente. Una limitación del servidor IIS es que los ejecutables CGI deben ser aplicaciones WIN32 de modo consola, aunque se pueden ejecutar todo tipo de aplicaciones con el procesador o intérprete adecuados. Así es posible instalar los intérpretes de Perl, Tcl, etc.

Un ejemplo de biblioteca ISAPI incluida con el servidor IIS es la herramienta de conectividad con el servidor SQL Server. Esta biblioteca permite un acceso sencillo a los datos del servidor SQL Server.

1.2.3. Actividades de aprendizaje

Elaboración de una aplicación Web

Instrucciones

1. Realizar el ejercicio: "1.8.1 Creación de una aplicación Web simple"

2. Contestar el cuestionario: "1.9.1 Preguntas aplicaciones distribuidas y el Web"

3. Implementar una aplicación Web que permita desplegar el plan de vuelos de una línea aérea para la ciudad destino que sea seleccionada.

    - Tomar como base el ejercicio 1.8.1 realizado con codificación C#.

Se recomienda la lectura del Material de Apoyo "1.2 Aplicaciones distribuidas y el Web" y/o "Anexo E. Fuentes de Información".

1.3. Tecnologías base de servicios Web

1.3.1. Introducción a las tecnologías básicas

Un servicio Web es una colección de funciones que son empaquetadas como una sola entidad y publicadas a la red para el uso a través de otros programas. Los servicios Web son bloques componentes para crear sistemas distribuidos abiertos y permiten a las compañías e individuos acceder rápidamente a los recursos digitales disponibles mundialmente.

La base de Servicios Web es la mensajería de XML sobre los protocolos del Web estándar como HTTP. Éste es un mecanismo de comunicación muy ligero en el que cualquier lenguaje de programación, middleware o plataforma pueden participar, suavizando la interoperabilidad grandemente.

Las tecnologías más populares que han ganado mayor aceptación de la industria y que son un camino posible de realizar los servicios Web son las siguientes:

- Un proveedor crea, ensambla y despliega un servicio del Web mediante un lenguaje de programación, middleware y plataforma de la propia opción del proveedor.

- El proveedor define el servicio del Web en WSDL (Web Services Description Language). Un documento de WSDL describe un servicio del Web al otro.

- El proveedor registra el servicio en registros de UDDI (Universal Description, Discovery and Integration). UDDI permite a los desarrolladores publicar los Servicios Web y eso permite a su software buscar los servicios ofrecidos por otros.

- Un usuario probable encuentra el servicio buscando un registro de UDDI.

- La aplicación del usuario liga al Servicio Web e invoca las operaciones del servicio mediante SOAP (Simple Object Access Protocol). SOAP ofrece un formato de XML para representar parámetros y valores devueltos sobre HTTP.

Con estas tecnologías es una manera de hacer trabajo de servicios Web. Hay otras opciones también, pero estas tecnologías son los más importantes y han logrado el acogimiento amplio en la industria. Aunque, no se ha perfilado completamente y todavía hay detalles para la construcción de servicios Web, en la actualidad se tienen los progresos siguientes:

- Generalmente hay acuerdos de que SOAP, WSDL y UDDI son entidades apropiadas y que (o sus estándares derivados) proveerán una base al futuro.

- Todos los vendedores están trabajando juntos para establecer estándares de los servicios Web y una base está emergiendo.

1.3.2. Descripción de las tecnologías básicas

XML

En el contexto de servicios Web, XML se usa no sólo como el formato de mensaje sino también como la manera en la que los servicios están definidos. Por consiguiente, es importante saber un poco sobre XML, sobre todo dentro del contexto de cómo se usa para definir e implementar los servicios Web.

Propósitos de XML

XML fue desarrollado para superar limitaciones de HTML, especialmente para soportar mejor la creación y manejo del contenido dinámico. HTML está bien para definir y mantener el contenido estático, pero cuando el Web evoluciona hacia una plataforma de software activado, en la que los datos tienen significado asociado, el contenido necesita ser generado y dirigido dinámicamente. Usando XML, pueden definir cualquier número de elementos para el significado asociado con datos; es

decir, describe los datos y qué realiza con ellos usando uno o más elementos creados para ese propósito. Por ejemplo:

```
<Company>
  <CompanyName region="US">
  Skateboots Manufacturing
  </CompanyName>
  <address>
   <line>
    200 High Street
   </line>
   <line>
   Springfield, MA 55555
   </line>
   <Country>
   USA
   </Country>
  </address>
  <phone>
  +1 781 555 5000
  </phone>
</Company>
```

En este ejemplo, XML permite no sólo definir elementos que describen los datos sino también las estructuras de grupo de datos relacionados. Es fácil de imaginar una búsqueda para elementos que emparejan cierto criterio, como <Country> y <phone> para una compañía dada o para todos los elementos <company> y regresar una lista de esas entidades que se identifican como compañías en el Web. Además, XML permite a esquemas asociados validar los datos separadamente y describir otros atributos y calidades de los datos, algo completamente imposible usando HTML.

Dos participantes que intercambian datos de XML pueden entender e interpretar los elementos en el mismo modo si solamente comparten las mismas definiciones. Si los dos participantes que comparten un documento XML también comparten el mismo esquema, pueden asegurarse de entender el significado de las mismas marcas de elementos en el mismo modo. Esto es exactamente cómo trabajan los servicios Web.

Tecnologías de XML

XML es una familia de tecnologías: un lenguaje de marcado de datos, varios modelos de contenido, un modelo de vinculación, un modelo de espacio de nombres y varios mecanismos de transformación. En seguida se describen los miembros característicos de la familia XML usados como la base de servicios Web:

- XML v1.0: Reglas que definen los elementos, atributos y marcas adjuntas dentro de un elemento raíz del documento, proporcionando un modelo de datos abstracto y serialización de formato.

- XML schema: Documentos XML que definen los tipos de datos, contenido, estructura y elementos permitidos en un documento XML asociado; también usado para la descripción de instrucciones de procesamiento semántico asociadas con elementos del documento.

- XML namespaces: Los nombres singularmente calificados para los elementos del documento XML y aplicaciones.

- XML Information Set: Una consistente representación abstracta de las partes de un documento del XML.

- XPointer: Un apuntador para una parte específica de un documento.

- XPath: Expresiones para la búsqueda en documentos de XML.

- XLink: Para la búsqueda múltiple de documentos XML.

- XSLT (Extensible Stylesheet Language Transformations): Transformación para documentos XML en otros formatos de documentos XML o para exportar en formatos de no-XML.

- DOM (Document Object Module) y SAX (Simple API for XML): Bibliotecas de programación y modelos para análisis de documentos XML, o creación de árbol completo para ser visitado con respuesta uno por uno de los elementos de XML.

Estas tecnologías y otras se describen en detalle en el tema 2.

SOAP

La especificación de SOAP define una estructura de la mensajería para intercambiar los datos de XML formateados en Internet. La estructura de la mensajería es simple, fácil de desarrollar y completamente neutral con respecto al sistema operativo, lenguaje de programación o la plataforma de computación distribuida.

SOAP es fundamentalmente un modelo de comunicación unidireccional que asegura que un mensaje coherente sea transferido desde un remitente a un destinatario, inclusive potencialmente intermediarios que pueden procesar parte

de o agregar a la unidad del mensaje. La especificación de SOAP contiene las convenciones de adaptar la mensajería unidireccional para el paradigma típico de petición/respuesta en comunicación estilo RPC y además define cómo transmitir los documentos XML completos.

SOAP define una regla de codificación optativa para los tipos de datos, pero en los puntos terminales en una comunicación de SOAP puede decidirse por sus propias reglas de codificación a través de acuerdo privado. La comunicación usa a menudo la codificación nativa XML.

SOAP esta diseñado para proporcionar un protocolo de comunicaciones independiente, abstracto capaz de pontear o conectando, dos o más negocios o dos o más sitios comerciales remotos. Los sistemas conectados pueden ser construidos usando cualquier combinación de hardware y software que soporten el acceso a Internet para los sistemas existentes, como .NET y J2EE.

Los sistemas existentes típicamente también representan infraestructuras múltiples y productos del paquete de software. SOAP y el resto de la estructura de XML proveen los medios a cualquier dos o más sitios comerciales, mercados o países que comercian entre sí para convenir en un contacto común para exponer sus servicios al Web.

SOAP tiene varias partes principales:

- Envelope: Define el inicio y el fin del mensaje.
- Header: Contiene cualquier atributo optativo del mensaje usado en el procesamiento del mensaje o a un punto intermediario o al punto final definitivo.
- Body: Contiene los datos de XML que comprenden el mensaje a enviar.
- Attachment: Consiste en uno o más documentos adjuntados al mensaje principal (sólo SOAP con ligaduras).
- RPC Interaction: Define cómo modelar las interacciones del estilo RPC con SOAP
- Encoding: Define cómo representar datos simples y complejos en el mensaje a transmitir.

SOAP se trata en detalle en el tema 3.

WSDL

El Lenguaje de descripción de servicios Web (WSDL) es un formato de esquema XML que define una estructura extensible por describir las interfaces de los servicios Web. WSDL fue desarrollado principalmente por Microsoft e IBM y fue sometido por compañías al W3C. WSDL tiene el núcleo de la estructura de los servicios Web, proporcionando un modo común para representar los tipos de datos proporcionados en los mensajes, las operaciones a ser realizadas en los mensajes y el mapeo de los mensajes sobre el transporte de la red.

WSDL es dividido en tres elementos mayores:

- Definiciones de tipo de datos.
- Definiciones abstractas.
- Servicio de ligaciones.

Cada elemento mayor puede especificarse en un fragmento de documento XML separado y puede importarse en varias combinaciones para crear una descripción final de los servicios Web o ellos pueden todos ser definidos juntos en un solo documento. Las definiciones del tipo de datos determinan la estructura y el contenido de los mensajes. Las definiciones abstractas determinan las operaciones realizadas en el contenido del mensaje y el servicio de ligaciones determina el transporte de la red que llevará el mensaje a su destino.

WSDL se trata en detalle en el tema 4.

UDDI

Después de que han definido los datos en los mensajes (XML), han descrito los servicios que recibirán y procesarán el mensaje (WSDL) y han identificado los medios de enviar y recibir los mensajes (SOAP), necesitan una manera para publicar el servicio que ofrecen y encontrar los servicios que otros ofrecen y que desean usar. Ésta es la función que proporciona UDDI (distribución universal, descubrimiento e integración).

La estructura de UDDI define un modelo de datos en XML e interfaces de programación de aplicaciones de SOAP (APIs) para registrar y descubrir la información comercial, inclusive los servicios Web en publicación de negocios. UDDI es producido por un consorcio independiente de vendedores, fundado por Microsoft, IBM y Ariba, para el desarrollo de un estándar de Internet en el registro de descripción y descubrimiento de servicios Web.

UDDI es similar en concepto a un directorio de las páginas amarillas. Los negocios registran su contacto de información, incluyendo cosas en detalle como números de teléfono y fax, código postal y sitio Web. El registro incluye la información de la categoría por buscar, como ubicación geográfica, razón social de la industria, tipo de negocio mercantil, y así sucesivamente. Otros negocios pueden buscar la información registrada en UDDI para encontrar suministro de partes, servicios de comida, compraventas o actividades comerciales. Un negocio también puede descubrir la información sobre el servicio Web específico en el registro, encontrando típicamente una URL para un archivo de WSDL que señala el servicio Web de un distribuidor.

Los negocios usan SOAP para registrarse u otros con UDDI; entonces los clientes del registro usan la consulta de las APIs para buscar la información registrada y descubrir socios comerciales. Una consulta inicial puede devolver varios juegos de los que una sola entrada es escogida. Una vez que una entrada comercial es escogida, una última llamada del API se hace para obtener la información del contacto específica para el negocio.

UDDI se trata en detalle en el tema 5.

1.3.3. Actividades de aprendizaje

Elaboración de una aplicación con servicio Web

Instrucciones

1. Realizar el ejercicio: “1.8.2 Creación de una aplicación cliente y servicio Web simple”

2. Contestar el cuestionario: “1.9.2 Preguntas tecnologías base de servicios Web”

3. Implementar la aplicación del ejercicio 1.8.2 pero modificando el acceso de base datos con SQL Server 2000.

Se recomienda la lectura del Material de Apoyo “1.3 Tecnologías base de servicios Web” y/o “Anexo E. Fuentes de Información”.

1.4. Arquitectura de los servicios Web

1.4.1. Roles, servicios y pila de interoperabilidad

Hay dos formas de ver la arquitectura de los servicio de Web. La primera es examinar los roles individuales de cada actor del servicio de Web, en la que se puede interpretar como una arquitectura orientada a servicios; la segunda es examinar el servicio Web surgiendo de la pila protocolar o de interoperabilidad.

Hay tres roles mayores dentro de la arquitectura del servicio Web:

- Proveedor de Servicios:
  Es el proveedor del servicio Web. El proveedor de servicios lleva a cabo el servicio y lo hace disponible en Internet.

- Solicitante de Servicios:
  Es cualquier consumidor del servicio Web. El solicitante utiliza un servicio del Web existente abriendo una conexión de red y enviando una petición XML.

- Registro de Servicios:
  Es un directorio lógicamente centralizado de servicios. El registro proporciona un lugar central dónde los desarrolladores pueden publicar los nuevos servicios o pueden encontrar existentes. Sirve por consiguiente como una cámara de prestación centralizada para las compañías y sus servicios.

La figura muestra los roles de servicios de servicio Web y cómo interactúan entre sí.

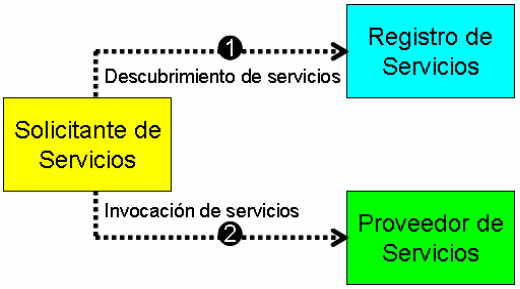


Figura 6. Roles en los servicios Web

Al asumir los roles de servicios, en la actualidad las aplicaciones Web son basadas en una arquitectura distribuida, las aplicaciones Web se han movido fuera de la naturaleza centralizada de datos de aplicaciones típicas del cliente-servidor y han cuidado más a una arquitectura orientada a servicios.

Las tecnologías distribuidas básicas RPC, RMI y CORBA son inherentemente orientadas a servicios. Las aplicaciones Web han evolucionado de aplicaciones simples accedidas y han desplegado en un Intranet a las aplicaciones de negocios y organizaciones diferentes "conversando" con nosotros e intercambiando los datos. El próximo paso para las aplicaciones de Web es proporcionar servicios comerciales a los que pueden ser tenidos acceso por otras aplicaciones de negocios y organizaciones diferentes que usan la arquitectura orientada a servicios de servicios Web.

Con el enfoque de una arquitectura orientada a servicios de servicios Web, la aplicación proporciona un servicio y la aplicación cliente usando el servicio de habla para ambos en un lenguaje común. Luego, un servicio que proporciona la aplicación y la aplicación cliente usando el servicio requerido en algún modo para localizarse antes de que ellos empiecen hablando entre sí. Esto es especialmente verdad para aplicaciones distribuidas dónde las aplicaciones no tienen conocimiento de la ubicación entre ellas.

Por lo que, se resume que una arquitectura orientada a servicio básica para Servicios Web tiene:

- Una manera estándar para la comunicación.
- Una representación de la información uniforme y mecanismos de intercambio.
- Un meta lenguaje estándar para describir los servicios ofrecidos.
- Un mecanismo para registrar y localizar aplicaciones basadas en servicio Web.

La segunda forma para ver la arquitectura de servicio Web es examinar el servicio Web emergiendo de la pila de interoperabilidad. La pila todavía está evolucionando, pero actualmente tiene cuatro capas principales. En seguida se da una descripción breve de cada capa.

- Transporte de Servicio:
  Esta capa es responsable para transportar los mensajes entre las aplicaciones. Actualmente, esta capa incluye el protocolo de transferencia de hipertexto (HTTP), el protocolo simple de transferencia de correo (SMTP), el protocolo de transferencia de archivo (FTP) y los más reciente protocolos, como el protocolo intercambio de bloques extensible (BEEP).

- Mensajería XML:
  Esta capa es responsable para codificar los mensajes en un XML común que da formato para que puedan entenderse mensajes a cualquier fin. Actualmente, esta capa incluye XML-RPC y SOAP.
- Descripción de Servicio:

Esta capa es responsable para describir la interfaz pública a un servicio Web específico. Actualmente la descripción de servicio es manejado por el lenguaje WSDL.

- Descubrimiento de servicio
  Esta capa es responsable para centralizar los servicios en un registro común y proporcionando fácil funcionalidad en la publicación y localización. Actualmente, el descubrimiento de servicio es manejado por UDDI.

Como los servicios Web evolucionan, pueden agregarse capas adicionales y pueden agregarse tecnologías adicionales a cada capa. En seguida la figura resume la pila de interoperabilidad del servicio Web en la actualidad. Cada capa ha sido tratada en general, pero seguirán tratándose a detalle sus características e implementación a través del curso.

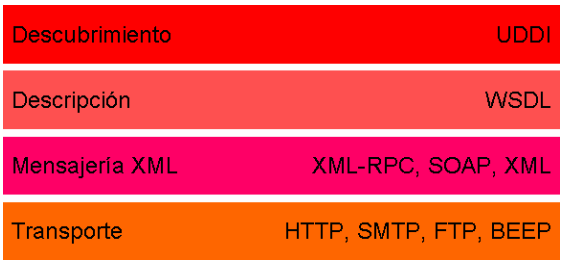


Figura 7. Pila de interoperabilidad de los servicios Web

1.4.2. Modalidades de la pila de interoperabilidad

La arquitectura de lo servicios Web es una infraestructura que permite que ciertos servicios de la red sean dinámicamente descritos, publicados, descubiertos e invocados en un entorno de computación distribuida. Comúnmente esa arquitectura es representada como una pila ilustrada de capas o un formato representativo, denominada pila de interoperabilidad de los servicios Web.

Cada vendedor, organización de estándares, o empresa de búsqueda de mercados define los servicios Web de una manera ligeramente diferente. Gartner, por ejemplo, define servicios Web como “componentes de software flojamente acoplados que interactúan entre si dinámicamente con vía las tecnologías estándares de Internet.” Forrester Research lleva un acercamiento más abierto a servicios Web como "conexiones automatizadas entre las personas, sistemas y aplicaciones que exponen elementos de funcionalidad comercial como un servicio del software y crean el nuevo valor comercial."

Por consiguiente, la arquitectura de la pila de los servicios Web varía de una organización a otro. El número y complejidad de capas para la pila dependen de la organización. Cada pila exige a las interfaces de servicios Web conseguir un cliente de servicios Web para conversar a una aplicación de Servidor o a un componente Middleware, tal como CORBA, Java 2EE o Microsoft .NET. Para habilitar la interfaz, necesitan SOAP, SOAP con ligaduras o RMI, entre otros protocolos de Internet.

Aunque hay una variedad de arquitecturas de servicios Web, a un nivel básico, pueden ser considerados como una arquitectura cliente-servidor global que permite a los sistemas dispares para comunicar entre sí sin usar las bibliotecas del cliente propietario, de acuerdo al informe de investigación de WebMethods, Implementing Enterprise WebServices with the WebMethods Integration Platform (diciembre de 2001). El informe de investigación señala que "esta arquitectura simplifica el proceso de desarrollo típicamente asociado con aplicaciones cliente/servidor por la eficaz eliminación de dependencias de código entre el cliente y servidor" y "la información de interfaz del servidor se descubre al cliente vía un archivo de configuración codificado en un formato estándar (WSDL)." Esto permite al servidor publicar un solo archivo para todas las plataformas del cliente destino.

Las pilas de arquitectura de los servicios Web que muestran los proveedores u organizaciones se presentan en orden de complejidad ascendente: WebServices.Org, Stencil Group, IBM y W3C. Además otras pilas de arquitectura son presentadas por Microsoft, Sun, Oracle, HP, BEA y Borland.

Basado en resultados iniciales o el estado actual de aplicaciones, la arquitectura de IBM es muy aceptable. En las versiones de IBM, con el navegador les permite que busquen los servicios existentes, ve sus descripciones de servicio y automáticamente invoca esos servicios. Esto permite ver cada capa dentro de la pila de interoperabilidad sin escribir cualquier código realmente. De manera similar Microsoft con la infraestructura .NET trata las capas estandarizadas en una simple arquitectura modular para construir, describir y localizar servicios Web.

1.4.3. Actividades de aprendizaje

Elaboración esquemática de la arquitectura de servicios Web

Instrucciones

1. Elaborar un mapa mental o cuadro sinóptico relativo a la arquitectura de los servicios Web de acuerdo a proveedores, organizaciones de normalización, o

empresas de búsqueda de mercados, en base a una investigación documental y lectura de fuentes de información, incluir las referencias.

2. Contestar el cuestionario: “1.9.3 Preguntas arquitectura de los servicios Web”

Se recomienda la lectura del Material de Apoyo “1.4 Arquitectura de los servicios Web” y/o “Anexo E. Fuentes de Información”.

1.5. Plataformas de desarrollo

1.5.1. Introducción a las plataformas

Los Servicios Web son componentes de software o “islas de funcionalidad” computablemente accesibles desde la intranet o desde Internet que, utilizando protocolos abiertos, incluyendo el HTTP y XML para el transporte y representación de información variada y flexible, permiten crear aplicaciones distribuidas centradas en el usuario. Ya que la interacción entre un servicio Web y su cliente se encuentra gobernada por estándares abiertos, cualquier dispositivo o plataforma que los soporte puede albergar o consumir estos servicios. El modelo de programación de Servicios Web es completamente independiente de la plataforma y se adapta de manera ideal a la naturaleza heterogénea de Internet.

Teóricamente, utilizando los servicios Web las posibilidades son ilimitadas. Los servidores que necesiten interactuar con otros servidores pueden hacerlo, los servidores pueden interactuar con los clientes y los clientes pueden interactuar entre sí. Por ejemplo, si te diriges a una sala de juntas y llevas en la mano un PDA y hay otras personas en la sala con dispositivos similares, deberías ser capaz de recuperar, proporcionar e intercambiar información con ellos. A este respecto, la solución requiere elementos de cliente-a-cliente, cliente-a-servidor y servidor-a-servidor, construidos alrededor de un estándar único y utilizando las características y la inteligencia con la que estén dotados estos dispositivos.

A medida que la importancia de Internet ha ido creciendo y las empresas buscan integrar su información entre límites departamentales y empresariales, la salida viable será el enfoque de desarrollo de soluciones basado en aplicaciones distribuidas y servicios Web. Desde el punto de vista del consumidor, estos servicios son conceptualmente similares a los componentes habituales de Visual Basic, salvo que encapsulan sus propios datos y no forman parte, estrictamente hablando, de la aplicación sino que ésta los utiliza. Esas aplicaciones y servicios que necesitan integrarse pueden ser desarrollados en distintas plataformas, por distintos equipos, en diferentes programas y se pueden mantener y actualizar de una forma independiente.

Enfoque de J2EE y .NET para servicios Web

Ante la creciente demanda de soluciones distribuidas, en el mercado han surgidos dos plataformas o enfoques tecnológicamente distintos e incluso enfrentados. Una de ellas, la pionera, ofrece la visión de la empresa Sun Microsystems (creadora del lenguaje Java) y recibe el nombre genérico de J2EE (Java2 Enterprise Edition). La otra, más reciente, es la perspectiva de Microsoft y recibe el nombre de Plataforma "dotNET" o, más concretamente, .NET Framework.

En este punto, para clarificar conceptos, añadir que J2EE es, ante todo, un conjunto de especificaciones de Sun que definen un posible estándar "de facto" para el desarrollo de aplicaciones empresariales multicapa, no un producto. Son varios los vendedores que las implementan en sus productos. Entre otros, los más importantes son IBM WebSphere, BEA WebLogic, Oracle9iAS o Sun ONE. Sun también, pero aunque sea la propietaria de la definición J2EE, su presencia en el mercado, aun siendo importante, es relativamente menor. A su vez, cada una de estas implementaciones proporciona servicios de valor añadido a los originalmente propuestos en las especificaciones J2EE.

La plataforma .NET en realidad no es algo radicalmente nuevo. Es un conjunto de tecnologías dispersas, que en muchos casos ya existían, que Microsoft ha integrado en una plataforma común con el objetivo de facilitar el desarrollo de las aplicaciones distribuidas.

J2EE y .NET son despliegues de tecnología de aplicación de servidor usadas para construir aplicaciones empresariales. Las versiones más tempranas de estas tecnologías no se han usado para construir los servicios Web históricamente. Ahora con el arribo de servicios Web, ambas infraestructuras están recalibrando sus soluciones como plataformas que también pueden usarse para construir los servicios Web.

1.5.2. Descripción de Plataformas

Plataforma J2EE

Argumentos de Sun, "J2EE define un estándar para el desarrollo de aplicaciones empresariales multicapa. J2EE simplifica las aplicaciones empresariales basándolas en componentes modulares y estandarizados, proveyendo un completo conjunto de servicios a estos componentes, y manejando muchos de las funciones de la aplicación de forma automática, sin necesidad de una programación compleja".

Las capas son simples agrupaciones lógicas de los componentes de software que conforman una aplicación o servicio. Ayudan a diferenciar entre los distintos tipos de tareas que realizan los componentes, facilitando el diseño de su reutilización en la solución. Cada capa lógica contiene un número de tipos de componentes discretos agrupados en subcapas, cada una de las cuales realiza el mismo tipo de tarea específica. Al identificar los tipos genéricos de componentes que existen en la mayoría de soluciones, se puede construir un mapa coherente de una aplicación o servicio y, a continuación, utilizar este mapa como plano técnico para el diseño.

La figura muestra una visión muy simplificada de las tres capas (presentación, lógica empresarial y acceso a datos) de una aplicación Web distribuida con arquitectura J2EE y los diferentes componentes de sus especificaciones: acceso a base de datos (JDBC), utilización de directorios distribuidos (JNDI), acceso a métodos remotos (RMI/IIOP), funciones de correo electrónico (JavaMail), aplicaciones Web (JSP y Servlets), etc.

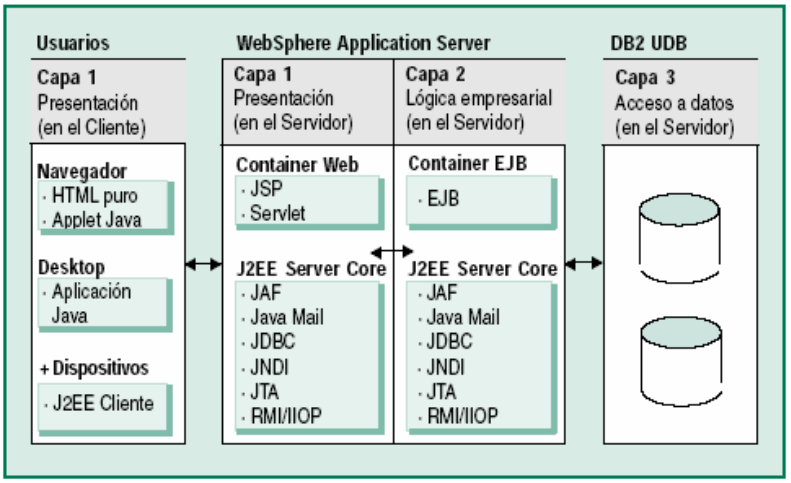


Figura: Plataforma J2EE en una aplicación multicapa

J2EE y servicios Web

J2EE ha sido históricamente una arquitectura para alojar despliegues del lado del servidor en el lenguaje de programación Java. Puede usarse para construir los sitios Web tradicionales, componentes software o aplicaciones empaquetadas. J2EE se ha extendido recientemente para incluir soporte en la construcción de servicios Web basados en XML. Estos Servicios Web pueden interoperar con otros servicios Web que pueden o no haber sido escritos en el estándar de J2EE. El modelo de desarrollo de J2EE Web Services se muestra en la figura siguiente.

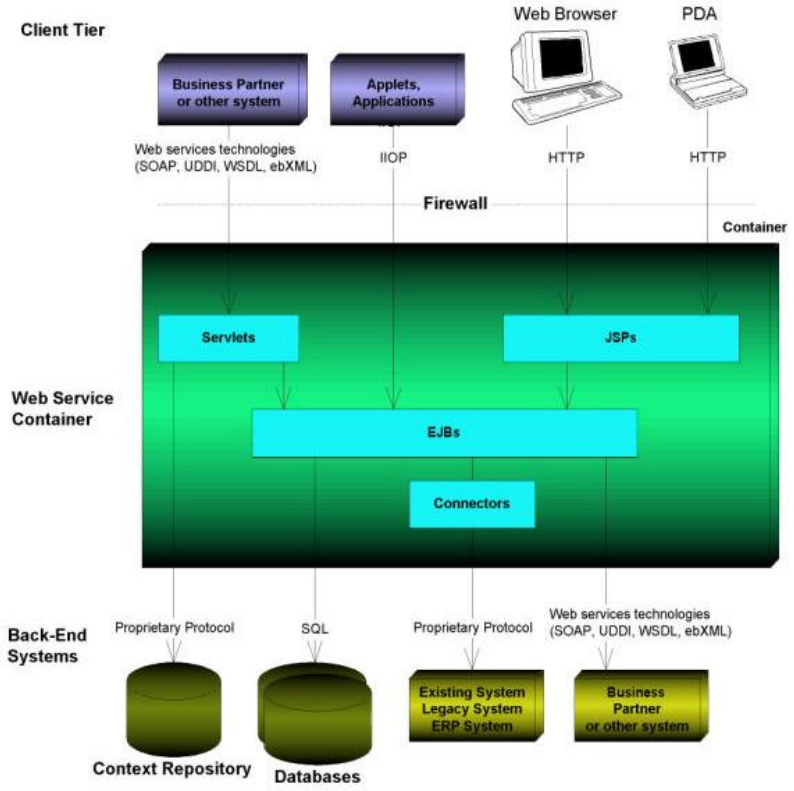


Figura: Desarrollo de servicios Web con J2EE

Enseguida, brevemente se explica lo que representa la figura:

- La aplicación de J2EE
  Se organiza dentro de un contenedor que provee calidades de servicio necesario a las aplicaciones de empresas, tal como transacciones, seguridad y servicios de persistencia.

- La capa de negocios
  Realiza proceso de negocios y lógica de datos. En aplicaciones de J2EE en gran escala, la lógica de negocios se construye usando los componentes Enterprise JavaBeans (EJB). Conecta a bases de datos que usando Java Database Connectivity (JDBC) o SQL/J o sistemas existentes que usan el Java Connector Architecture (JCA). También puede conectar a socios de negocios usando tecnologías de servicios Web (SOAP, UDDI, WSDL, ebXML) a través de los APIs de Java para XML (JAX APIs).

- Los socios de negocios
  Pueden conectar con aplicaciones de J2EE mediante tecnologías de servicios Web (SOAP, UDDI, WSDL, ebXML). Un servlet que es una petición/respuesta orientado al objeto de Java, puede aceptar las peticiones de servicio Web de los socios de negocios. El servlet usa los APIs de JAX para realizar las operaciones de servicios Web. Servicios de contexto compartido serán estandarizados en el futuro, los cuales serán incluidos con J2EE.

- Clientes tradicionales "densos"
  Los clientes tales como applets o aplicaciones conectadas directamente a la capa de EJB mediante Internet Inter-ORB Protocol (IIOP) en vez de servicios Web, dado que generalmente los clientes densos son escritos por la misma organización que creados por la aplicación J2EE y no hay necesidad en consecuencia de la colaboración del servicio Web basado en XML.

- Los exploradores Web y los dispositivos inalámbricos
  Conectan a JavaServer Pages (JSPs) que dan las interfaces del usuario en HTML, XHTML o WML.

Plataforma .NET

Microsoft .NET es una plataforma encaminada a la construcción y ejecución de aplicaciones distribuidas, que en lo general consiste de los medios siguientes:

- Un modelo de programación basado en XML.

- Un conjunto de servicios Web XML, como Microsoft .NET My Services para facilitar a los desarrolladores integrar servicios.

- Un conjunto de servidores que permiten ejecutar servicios, como .NET Enterprise Servers.

- Software en el cliente para poder utilizar servicios, como Windows XP, agendas electrónicas, etc.

- Herramientas para desarrollo, como Visual Studio .NET.

La tecnología .NET, es una estrategia de Microsoft que pretende homogeneizar y unificar sus APIs de programación en una sola que abstrae todo el sistema, simplificando el acceso a ella desde múltiples lenguajes, al mismo tiempo que sienta las bases de un rediseño estructural de Windows en favor de un modelo más uniforme de componentes orientados a objetos. Esta tecnología consta de dos partes:

- .NET Framework: un nuevo modelo de objetos COM+, un recompilador dinámico JIT (Just in time), un entorno de ejecución virtual y las propias APIs.

- .NET Services: Aplicaciones y librerías que pueden ser ejecutadas remotamente y utilizadas desde los programas.

El esquema de la figura muestra las partes fundamentales de la plataforma .NET, en seguida se describen estos elementos.

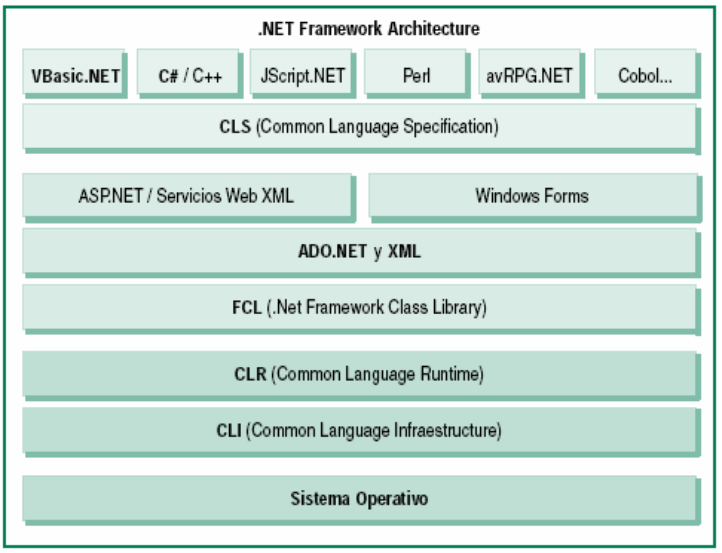


Figura: Arquitectura de .NET Framework

A un nivel muy elemental, .NET tiene bastantes similitudes con la tecnología Java: ambos compilan el código fuente a un código intermedio que, en el caso de Java se denomina Bytecode, y en .NET recibe el nombre de CIL (Common Intermediate Language). Para ejecutar este código intermedio es necesario un entorno que lo interprete para convertirlo al código de máquina del sistema donde se esté ejecutando.

Pero .NET va más allá de J2EE, su objetivo no es sólo la independencia de la compilación sino también la independencia del lenguaje de alto nivel que se utilice. Como veremos, CIL ha sido diseñado para proporcionar todo lo necesario para la mayoría de los lenguajes actuales. El que más aprovecha su potencia es C# un estándar diseñado por la propia Microsoft, pero nada impide que para formar parte de la plataforma .NET una empresa cree un compilador a código intermedio CIL para su lenguaje.

La arquitectura .NET (ver figura anterior) define las especificaciones de un lenguaje neutral denominado CLI (Common Language Infrastructure) que es el alma y corazón de lo que Microsoft denomina como ".NET Framework" y que a su

vez consta de un CTS (Common Type System) con soporte a los suficientes tipos de datos para cubrir las necesidades de cualquier lenguaje actual; y unas especificaciones CLS (Common Language Specification) que deben cumplir todos los lenguajes que pretendan beneficiarse de dicha interoperabilidad.

Este CLI proporciona, por ejemplo, la posibilidad de poder reutilizar clases programadas en un lenguaje de alto nivel como pueda ser C# desde Visual Basic.NET o desde cualquier otro lenguaje .NET de una forma muy sencilla. El lenguaje pasa a ser irrelevante, porque todas las clases son las mismas, y porque el entorno de desarrollo unificado (en el caso de Microsoft, Visual Studio .NET) brinda las mismas funcionalidades a todos.

El Common Language Runtime (CLR) es el entorno de ejecución que traduce el código intermedio CIL a código máquina y que, por tanto, permite ejecutar cualquier aplicación de la plataforma. La implementación del CLR de Microsoft incorpora tecnología JIT de forma que sólo se traducen a código máquina las partes necesarias y éstas se "recuerdan" por si vuelven a ser llamadas (por ejemplo, funciones) consiguiendo así un mayor rendimiento en la ejecución. En resumen, CLR actúa como:

- Un runtime para todos los lenguajes .NET
- Gestiona threads y memoria
- Recolector de basura (garbage collection)
- Fuerza la seguridad a nivel de código
- Elimina problemas de versiones de DLLs
- Podemos ejecutar simultáneamente varias versiones de un DLL
- Las aplicaciones pueden especificar la versión del DLL que van a usar.

El Framework Class Library (FCL) es la librería de clases responsable de proporcionar una gran cantidad de servicios: Entrada/Salida; XML; ADO.NET, para el acceso a Bases de datos; ASP.NET, como soporte para las aplicaciones Web (incluyendo "Web Forms" para la interfaz humana y "XML Web Services" para la interfaz de máquina); Windows.Forms, para el desarrollo de aplicaciones Windows tradicionales; sockets; colecciones; etc., etc. El FCL también presta parte de sus servicios a cualquier lenguaje que esté dentro de la plataforma .NET ya que éstos cumplen con CLI, minimizando así la incidencia de sus características propias.

En la tabla de la figura se relacionan las principales características técnicas de .NET y de J2EE.

Característica	.NET	Java2 Enterprise Edition
Acceso a bases de datos	ADO.NET	JDBC, SQL/J
Código máquina "ejecutable"	IL	Java Bytecode
Componentes	.NET Managed Components	EJB (Java Beans)
Librerías desarrollo - API	.NET Framework	Java API
Integración Web	ASP.NET	Servlets, JSP
Rercursos gráficos	WinForms y WebForms	Java Swing
IDE progamación	Visual Studio .NET	Según fabricante
Lenguajes	C#, VBasic, etc...	Java
Gestión mensajes	MSMQ	JMS
Runtime	CLR	JRE
Servicio de directorio	ADSI	JNDI
Servicios Web	SOAP, WDSL, UDDI	SOAP, WDSL, UDDI
Transacciones distribuidas	MS-DTC	JTS

Figura: Principales componentes de .NET y J2EE

.NET y servicios Web

El modelo de desarrollo para construir los servicios Web con Microsoft .NET se muestra en la figura siguiente.

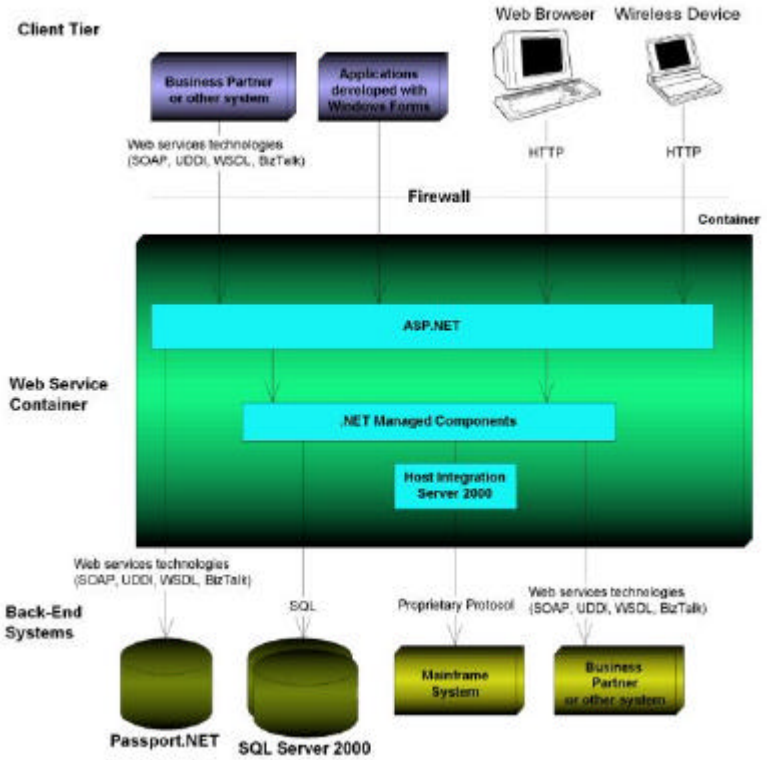


Figura: Desarrollo de servicios Web con .NET

Enseguida, brevemente se explica lo que representa la figura:

- La aplicación de .NET
  Se organiza dentro de un contenedor que provee calidades de servicio necesario a las aplicaciones de empresas como transacciones, seguridad y servicios de mensajería.

- La capa de negocios
  Es construida usando componentes administrados de .NET. Esta capa realiza proceso de negocios y lógica de datos. Conecta a bases de datos usando Active Data Objects (ADO.NET) y los sistemas existentes mediante los servicios proporcionadas por Microsoft Host Integration Server 2000, como el COM Transaction Integrator (COM TI). También puede conectar a socios de negocios que usan las tecnologías de servicios Web (SOAP, UDDI, WSDL).

- Los socios de negocios
  Pueden conectar con la aplicación de .NET a través de las tecnologías de servicios Web (SOAP, UDDI, WSDL, BizTalk).

- Clientes tradicionales "densos", exploradores Web y dispositivos inalámbricos
  Se conectan a Active Server Pages (ASP.NET) que dan las interfaces del usuario en HTML, XHTML o WML. Se construyen interfaces del usuario usando Windows Forms.

1.5.3. Actividades de aprendizaje

Resumen de la infraestructura .NET para servicios Web

Instrucciones

1. Elaborar una presentación sobre los aspectos más relevantes del .NET Framework para el desarrollo de aplicaciones con servicios Web, en base a una investigación documental y lectura de fuentes de información, incluir las referencias.

2. Contestar el cuestionario: "1.9.4 Preguntas plataformas de desarrollo"

Se recomienda la lectura del Material de Apoyo "1.5 Plataformas de desarrollo" y/o "Anexo E. Fuentes de Información".

1.6. Esquemas de implementación

1.6.1. Etapas de Implementación del servicio Web

Una vez comprendida cada estrato en el servicio Web, pila de interoperabilidad, el próximo paso importante es entender cómo esos estratos se acoplan juntos. Hay dos maneras de tratar el problema, desde el enfoque del solicitante de servicio o el enfoque del proveedor de servicios.

Enfoque solicitud de servicio

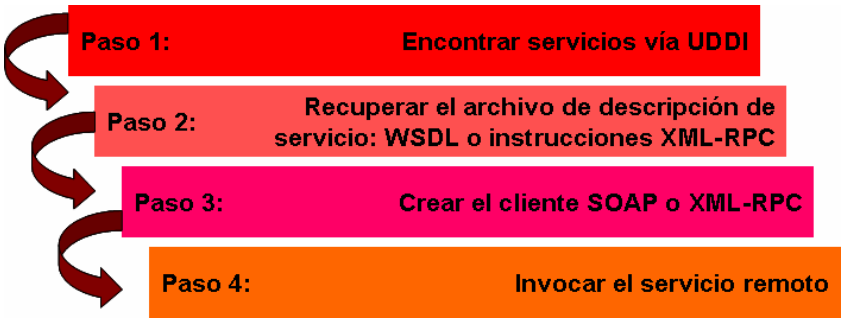
El solicitante de servicio es cualquier consumidor de servicios del Web. Un plan de desarrollo típico para un solicitante de servicio en orden es:

1. Debe identificar y descubrir esos servicios que son pertinentes para su aplicación. Este primer paso por consiguiente normalmente envuelve la búsqueda del directorio de negocios UDDI para los socios y servicios.

2. Una vez identificado el servicio que quiere, el próximo paso es localizar una descripción del servicio. Si éste es un servicio de SOAP, es probable que encuentre un documento de WSDL. Si esto es un servicio XML-RPC, es probable que encuentre algunas instrucciones humano-leíbles para la integración.

3. Debe crear una aplicación del cliente. Por ejemplo, puede crear un XML-RPC o un cliente SOAP en el lenguaje de su opción. Si el servicio tiene un archivo de WSDL, también tiene la opción de crear el código del cliente automáticamente vía una herramienta de invocación WSDL.

4. Finalmente, ejecute su aplicación del cliente para invocar el servicio Web actualmente.



Enfoque proveedor de servicios

El proveedor de servicios es cualquier proveedor de uno o más servicios del Web. Un plan de desarrollo típico para un proveedor de servicios en orden es:

1. Debe desarrollar la funcionalidad del núcleo de su servicio. Ésta normalmente es la parte más dura, cuando su aplicación puede conectar a las bases de datos, elementos EJB o COM+, o aplicaciones de herencia.

2. Debe desarrollar una envoltura de servicio a su funcionalidad del núcleo. Esto podría ser un XML-RPC o una envoltura de servicio SOAP. Éste normalmente es un paso relativamente simple, como está envolviendo la funcionalidad existente meramente en una plataforma mayor.

3. Luego, debe proporcionar una descripción de servicio. Si está creando una aplicación de SOAP, debe crear un archivo de WSDL. Si está creando un servicio XML-RPC, debe considerar la creación de algunas instrucciones humano-leíbles.

4. Necesita desplegar el servicio. Dependiendo de sus necesidades, esto representa instalarlo y ejecutarlo en un servidor autónomo o integrarlo con un servidor del Web existente.

5. Necesita publicar la existencia y característica técnicas de su nuevo servicio. Esto normalmente representa los datos de la publicación a un directorio de UDDI global o quizás un directorio de UDDI privado específico a su compañía.

1.6.2. Estructura y funcionamiento de la aplicación

Como hemos visto, un servicio Web puede ser una aplicación intermedia que permite que una aplicación cliente del servicio Web acceda a datos de una base de datos de apoyo. Para realizar esto podemos representar la estructura del servicio Web y su interacción con el cliente Web mediante las capas siguientes:

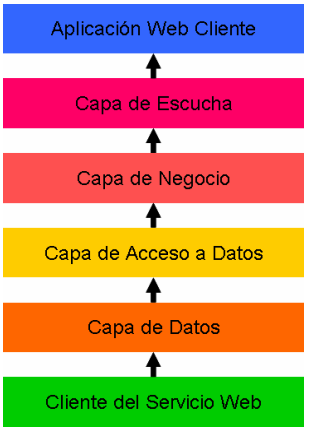
- **Capa de datos**. Esta capa es la primera de las capas del servicio Web y contiene los datos a los que debe acceder el servicio Web.

- **Capa de acceso a datos**. Esta capa está situada por encima de la capa de datos y contiene la lógica de negocio o el código que permite que la aplicación cliente del servicio Web acceda a los datos de la capa inferior. Además de almacenar datos, esta capa se usa para proteger los datos de la capa de datos.

- **Capa de negocio**. La tercera capa del servicio Web contiene el código necesario para implementar el servicio Web. La capa de negocio se divide a su vez en las capas de lógica de negocio y de fachada de negocio. La capa de lógica de negocio contiene todos los servicios que proporciona el servicio Web y la capa de fachada actúa como la interfaz del servicio Web.

- **Capa de escucha**. La capa más cercana al cliente del servicio Web se emplea para comunicar con el servicio Web. Cuando un cliente de servicio Web quiere acceder a un método Web presente en un servicio Web, el cliente envía una petición. Esta petición la recoge la capa de escucha, que la interpreta. Cuando se procesa la petición y el servicio Web devuelve la respuesta como un mensaje XML, la capa de escucha es quien se la reenvía al cliente del servicio Web.

En la figura, se muestra la estructura de un servicio Web.

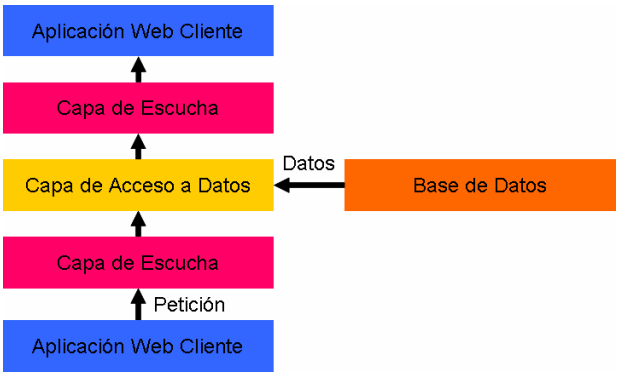


El funcionamiento de un servicio Web implica el envío de una petición por parte del cliente para obtener un servicio. Esta petición es un mensaje XML que se envía con un protocolo como HTTP.

Esta situación es parecida a la sentencia de llamada a un método que empleamos para llamar a un método concreto.

La petición del servicio se le pasa a la capa de escucha, que la reenvía al proveedor del servicio Web. Entonces, el proveedor de servicio Web procesa la petición.

El procesamiento de la petición incluye la capa de acceso para obtener los datos pedidos por la aplicación cliente. Estos datos se le pasan entonces a la capa de escucha, que a su vez se los reenvía a la aplicación cliente. La figura siguiente muestra el funcionamiento de un servicio Web.



Cuando una aplicación cliente envía la petición de un servicio, es posible que tengamos que pasarle argumentos.

Para enviar argumentos a través de una red, éstos se empaquetan como un mensaje SOAP y se le pasan al método Web con un protocolo de red.

Posteriormente, el servicio Web decodifica el mensaje SOAP para obtener los argumentos que le hemos pasado al método Web, se ejecuta el método y se le pasa el valor de vuelta a la aplicación Web cliente.

Podemos crear servicios Web ya sea con J2EE, o .NET, pero para que una aplicación pueda acceder a un servicio Web, esta aplicación debe de cumplir los requisitos que hemos mencionado sobre las estratos o pila de interoperabilidad. Estos requisitos incluyen un formato estándar para describir los servicios Web, un formato estándar para representar las transferencias de datos y un estándar para enviar los métodos y los resultados devueltos por estos métodos a través de la red. Además, para poder acceder a un servicio Web, la aplicación Web cliente

debe identificar un sistema para localizar el servicio Web y pasarle valores a sus métodos Web. Requisitos que son tratados ampliamente en la duración del curso.

En la actualidad para implementaciones mayores o menores de cliente y servicio Web, siempre será de gran utilidad aplicar técnicas orientadas a objetos en el desarrollo del sistema, para obtener una representación abstracta de las capas mencionadas arriba, así como las ventajas que ofrecen de reutilidad y escalabilidad.

1.6.3. Actividades de aprendizaje

Elaboración de servicios Web con tipos de datos complejos

Instrucciones

1. Realizar el ejercicio: "1.8.3 Creación de servicios Web con tipos de datos complejos"

2. Contestar el cuestionario: "1.9.5 Preguntas esquemas de implementación"

3. Implementar una aplicación que permita obtener un conjunto de datos del servicio Web para que sean desplegados en el cliente. Además que se permitan hacer actualizaciones en base de datos del lado del servicio Web.

    - Tomar como base los escenarios de línea aérea de las actividades de aprendizaje 1.3.2 y 1.3.3.
    - Utilizar técnicas de desarrollo orientado a objetos.
    - Obtener el conjunto de datos mediante el uso de ArrayList.

4. Realizar la misma implementación del inciso anterior (3) pero obteniendo el conjunto de datos con el uso de DataSet o DataReader.

Se recomienda la lectura del Material de Apoyo "1.6 Esquemas de implementación" y/o "Anexo E. Fuentes de Información".

1.7. Consideraciones de seguridad

1.7.1. Estándares de seguridad en servicios Web

Las tecnologías centrales de servicios Web, como SOAP, WSDL y UDDI, son útiles para enlazar dominios de tecnologías dispares y sus documentos sometidos a flujos de proceso de negocio. Sin embargo, para que sea de utilidad en más tipos de aplicaciones y cumplir una visión completa de los servicios Web, como habilitar el uso de bloques de componentes de la aplicación sobre Internet, las tecnologías de servicios Web tienen que ser extendidas para abarcar características adicionales, funciones y calidades de servicio. Estas son básicamente la seguridad, flujo de procesos, transacciones y mensajería.

Una de las tecnologías adicionales más importantes para los servicios Web incluye las tecnologías de seguridad.

La seguridad es importante para asegurar la confidencialidad e integridad de datos de servicios Web. La seguridad también es necesaria para controlar el acceso a servicios Web, sobre todo cuando se usan juntos servicios Web múltiples.

Se tienen estándares de seguridad propuestos para:

- La autentificación y autorización (SAML, Security Authorization Markup Language). SAML hace posible que los servicios Web intercambien información de autentificación y autorización entre ellos, de modo que un servicio Web confíe en un usuario autentificado por otro servicio Web.

- La administración de la llave pública para la encriptación (XKMS, XML Key Management Specification). XKMS define una serie de servicios para distribución y administración de llaves públicas y certificados. El propio protocolo se construye en SOAP y WSDL y es por consiguiente un ejemplo excelente de un servicio Web.

- La base de toda la seguridad de Internet el Secure Socket Layer (SSL). Ambos XML-RPC y SOAP trabajan principalmente sobre HTTP y puede ser cifrada la comunicación de XML vía SSL. SSL es una tecnología probada, se despliega ampliamente y es por consiguiente una opción viable para encriptación de mensajes.

- Los protocolos basados en HTTP, HTTPS (HTTP Secure) para el nivel básico de seguridad de encriptación.

- Firma digital XML. Asocia los datos del mensaje al usuario que emite la forma, de modo que este usuario es el único que puede modificar dichos datos.

- Encriptación XML. Evita que los datos se vean expuestos a lo largo de su recorrido.

- Validación de datos. Permite que lo servicios Web reciban datos dentro de los rangos esperados o tipos definidos.

- Además de HTTPS, firewalls, SAML, XKMS, el uso de firmas digitales y encriptación XML, Microsoft ha propuesto WS-License para la administración de credenciales y WS-Security para la propagación de credenciales de seguridad asociadas con interacciones de servicios Web.

Mucha de esta seguridad esta relacionada con SOAP, dado que cada mensaje que se intercambia realiza múltiples saltos y es encaminado a través de numerosos puntos antes de que alcance su destino final, por lo que la seguridad aumenta en consideración para los mensajes.

Además, hay técnicas que permiten mantener la seguridad a otros niveles. La seguridad en UDDI permite autentificar todas las entidades que toman parte en la publicación de un servicio Web; proveedor, agente y consumidor del servicio. De este modo, nadie podrá registrar servicios en el rol de un proveedor o hacer uso de ellos sin contar con los permisos adecuados.

1.7.2. Integración y calidad de servicios Web

Otras tecnologías adicionales relacionadas con la integridad, calidad y seguridad de los servicios Web son:

El flujo del proceso. Es crítico para automatizar las interacciones del proceso de negocios sobre el Web y dentro de una empresa. El flujo del proceso también se llama a menudo la orquestación porque define la relación entre una serie de interacciones necesarias para lograr un propósito dado, como completar un orden de compra, procesando una reservación de viaje o ejecutar un plan de manufactura. Un flujo se planea como una secuencia de pasos definida para un proceso de negocio dado. La serie de pasos crea una agregación de funciones para las que una interfaz de servicio Web pueda definirse.

Las transacciones. En el mundo de operaciones de negocios automatizadas, las transacciones han representado el papel de ejecutor por mucho tiempo, asegurando que las plataformas en la ejecución produzcan resultados consistentes de una serie de operaciones relacionadas en los datos, a pesar del software o errores de hardware. Estos protocolos tradicionales y técnicas no son directamente aplicables al Web, sin embargo, como ellos se diseña para un entorno herméticamente acoplado en el que es posible sostener los bloqueos de una base de datos la notificación pendiente del resultado del motivo transacción y

en que un protocolo orientado a conexión está disponible para detectar los fallos de comunicación automáticamente. La propuesta Business Transaction Protocol (BTP) de OASIS (Organization for the Advancement of Structured Information Standards) esta diseñada para resolverse este problema para servicios Web definiendo un protocolo débilmente acoplado que asegura que se propaguen los resultados de interacciones de servicios Web múltiples correctamente y compartidos.

La mensajería. Los protocolos de la mensajería ejecutan los patrones de comunicación definidos para las interacciones de servicio de Web, como asincrónico unidireccional, petición/respuesta, difusión, interactivo o punto a punto. Tecnologías adicionales de servicios Web también pueden sobre la capa de mensajería para ciertas calidades de servicio, como confiabilidad o garantía de entrega, propagación de seguridad y contextos de transacción, y enrutamiento correcto de los mensajes a lo largo de una ruta definida que incluye a uno o más intermediarios. IBM ha propuesto a HTTP Reliable (HTTPR) dirigido para requerimientos en esta área.

IBM y Microsoft han colaborado en la propuesta de WS-Inspection para descubrir la información sobre los servicios Web disponibles en un mensaje particular destino. Microsoft también ha propuesto a WS-Referral y a WS-Routing para definir una ruta del mensaje específica para un servicio Web, incluso cualquier número de intermediarios y cómo dirigir los mensajes remitir y hacia atrás a lo largo de la ruta especificada.

Los Blocks Extensible Exchange Protocol (BEEP) de IETF (Internet Engineering Task Force) definen un protocolo de Internet orientado a conexión. Un mapeo de SOAP para BEEP esta definido y en este caso, los mensajes de SOAP heredan las calidades adicionales de servicio de BEEP para mantener el contexto de la sesión al remitente y los nodos del destinatario. El contexto puede usarse para relacionar los mensajes múltiples en una unidad más grande de transferencia y relacionar los mensajes múltiples apareciendo del mismo origen o preparado para el mismo destino. También pueden asociarse seguridad y contexto de transacción con una conexión.

Si bien todavía se siguen proponiendo estándares adicionales por diferentes organizaciones. El trabajo de estas organizaciones a menudo con enfoque de promocionar la adopción de XML para los objetivos específicos de negocios, como construir sobre la base de los estándares la definición de formatos de documentos y protocolos para las industria electrónica, financiera, atención médica y otras. Debido a que los servicios Web están basados en XML, el trabajo de casi cualquier cuerpo de estándares o consorcio que promocionan el uso de tecnologías relacionadas a XML para negocio de Internet es conveniente. Algunos otros trabajos mencionados previamente, BTP y SAML, emergen como las

tecnologías candidatas para su adopción por el W3C dentro de su actividad de arquitectura de los servicios Web.

1.7.3. Actividades de aprendizaje

Elaboración de un servicio Web con medios de seguridad

Instrucciones

1. Realizar el ejercicio: “1.8.4 Creación de servicio Web con medios de seguridad”

2. Contestar el cuestionario: “1.9.6 Preguntas consideraciones de seguridad”

3. Implementar …

Se recomienda la lectura del Material de Apoyo “1.7 Consideraciones de seguridad” y/o “Anexo E. Fuentes de Información”.

1.8. Ejercicios prácticos

1.8.1. Creación de una aplicación Web simple

1. Revisar y estudiar la presentación “Implementación de aplicaciones Web” que se proporciona en el anexo B.

2. En base a la presentación, probar el código que se proporciona en el rubro “Código Aplicación Web Simple” del Anexo C.

   - Consiste en una aplicación que utiliza una página aspx (Web Form) para acceder a la base de datos Pubs (Access) y desplegar autores de publicaciones de libros de acuerdo a la elección previa de su estado o lugar de residencia.

3. Antes de crear la aplicación debe de verificar que su equipo este instalado adecuadamente el servidor IIS (Internet Information Services), el Visual Estudio .NET y el Microsoft Office Access, así como el SQL Server 2000 (cliente-servidor), el cual será utilizado en ejercicios posteriores.

4. Desde Visual Estudio .NET crear una aplicación Web ASP.NET con C# en el directorio \inetpub\wwwroot o desde un directorio que apunte desde IIS como virtual. El nombre de la aplicación debe ser SamplePage.aspx.

5. Agregar el código a la página e instalar la base de datos Pubs (Access), que se proporcionan en el rubro antes mencionado, "Código Aplicación Web Simple" del Anexo C.

6. Después de ejecutar satisfactoriamente la aplicación, el resultado del ejercicio debe realizar lo siguiente:

   - Mostrar la página que permita seleccionar los autores de publicaciones de la base de datos Pubs para el estado de residencia de los autores que deseen visualizarse.

   - Mostrar la información de los autores de acuerdo a la selección del estado de residencia.

1.8.2. Creación de una aplicación cliente y servicio Web simple

1. Revisar y estudiar la presentación "Introducción a los servicios Web .NET" que se proporciona en el anexo B.

2. En base a la presentación, probar el código que se proporciona en el rubro "Código Aplicación cliente y servicio Web simple" del Anexo C.

   - El escenario de la aplicación consiste en que un visitante solicita una reservación en una línea aérea, si existe lugar, el servicio Web actualiza el asiento y regresa al cliente éxito de lo contrario fracaso.

   - La aplicación tiene dos partes, el cliente Web (aspx) y el servicio Web (asmx), así como codificación en C#.

3. Los proyectos del cliente Web y servicio Web deberá de instalarlos en el directorio \inetpub\wwwroot o en un directorio que apunte desde IIS como virtual. Los nombres de los proyectos son MakeReservation, el cliente Web y AirlineReservation, el servicio Web.

4. Primero deberá probar el servicio Web, elaborado como ASP.NET Web Service, que realiza una consulta y actualización si es el caso, para la base de datos tickets (Access) y regresa un valor booleano.

5. En seguida deberá probar el cliente Web, elaborado como ASP.NET Web Application, que realiza el acceso al servicio Web. Es conveniente que elimine la referencia Web y vuelva a instalar la referencia del servicio Web ya probado.

6. Una vez efectuado lo anterior, el resultado del ejercicio debe de realizar lo siguiente:

   - Mostrar la página Cliente que permita seleccionar un lugar del avión de la línea aérea, ubicación y clase. Oprimir la opción de reservar para que se conecte al servicio Web y regrese éxito o fracaso en la reservación.

   - El servicio Web actualizará en base de datos o regresará fracaso si no hay cupo para la reservación.

1.8.3. Creación de servicios Web con tipos de datos complejos

1.8.4. Creación de servicio Web con medios de seguridad

1.9. Cuestionarios

1.9.1. Preguntas aplicaciones distribuidas y el Web

¿Cuáles son los parámetros estándar del manejador de eventos para las páginas ASP.NET?

void event_handler_name(Object Sender, EventArgs e)

Los argumentos o lista de parámetros distinguen al manejador del evento. El primer parámetro es un tipo de datos Object que representa el objeto que levanta al evento. El segundo parámetro, EventArgs, es un nuevo tipo que contiene cualquier información específica para el evento que ha ocurrido. Típicamente, esta variable es vacía. Aunque es estándar la lista de parámetros hay casos de excepción a la regla.

1.9.2. Preguntas tecnologías base de servicios Web

¿Un solo proyecto de servicio Web XML puede proporcionar servicios múltiples?

Sí, puede agregar los servicios múltiples a su proyecto de servicio Web XML. Estos servicios son todo compilados como archivos asmx diferentes y comparten el mismo archivo de Global.asax. Podríamos tener funciones en un segundo servicio y los clientes permitidos para tener acceso a ellos separadamente.

¿Tiene sentido agregar la descripción opcional para sus métodos?

Sí, es conveniente. Si sus métodos se exponen a los clientes externos, la vida es más simple si pueden ver una descripción pequeña de cada método cuando escriben el código para usar su servicio. Aun cuando sus métodos sólo son internamente expuestos y sólo usados por las personas que les escribieron, se olvidará de lo que algunas funciones hacen en el futuro; tener estos recordatorios pequeños puede salvarlo del problema de tener que buscar el material de referencia o leer las líneas de comentario del código fuente.

Ha creado un servicio Web XML que compila correctamente pero cuando lo ejecuta no expone ningún método. ¿Por qué?

Olvidó incluir la declaración <WebMethod()> en su código.

¿Cómo incluye una referencia Web a un proyecto en Visual Estudio .NET?

Usando el diálogo desplegado Add Web Reference bajo el Project menu.

¿Por qué es importante crear un nuevo archivo WSDL después del despliegue de su servicio Web XML?

El archivo original de WSDL apunta a su servidor de desarrollo, no a la ubicación dónde el servicio tiene que ser desplegado.

Necesita cerrar una conexión de una base de datos y poner algunas referencias de objetos. ¿Dónde debe realizar esto?

En el método Dispose() que es donde los servicios Web XML hacen su limpieza.

¿Será necesario crear una clase proxy para mis clientes del servicio Web XML?

Sí. La alternativa para crear una clase Proxy es para que usted escriba toda la funcionalidad del manejo de SOAP. Esto es interesante como un ejercicio de aprendizaje pero no puede ser práctico para más escenarios de desarrollo.

Cómo crear un nuevo objeto, llamado oServicio, basado en una clase llamada Balance de una DLL de servicio Web XML, llamada Presupuesto (en C#):

```
Presupuesto.Balance OServicio = new Presupuesto.Balance
```

¿Existirá una manera de crear una sola clase como intermediaria que selecciona referencias desde varios servicios Web XML?

Es posible crear una sola clase que se ocupa de los métodos de servicios Web XML múltiples. Esto se hace usando WSDL.exe. Selectivamente eliminando los métodos no deseados desde la clase resultante, borrando directamente del código fuente generado antes de compilar.

¿Existirá una razón para usar WSDL.exe si tiene el Visual Estudio .NET?

Sí, hay varias razones. La primera es que WSDL.exe le permite crear una clase una vez a partir de los servicios Web XML múltiples. Esto permite una clase para servir todo de sus funciones de los servicio Web XML o por lo menos agruparse los servicios juntos relacionados en varias clases. La segunda razón es que puede controlar la ubicación del DLL de salida, haciéndole más fácil crear uno DLL y reutilizarlo en proyectos múltiples. Debe esperar que Microsoft incluya muchas de estas características a los diálogos de Referencia Web en liberaciones del futuro.

¿Qué realiza la opción /language de WSDL.exe?

Controla el lenguaje en el que la clase proxy resultante es creada.

¿Cuál es el nombre de DLL que normalmente se agrega a los clientes de servicio Web XML para obtener la funcionalidad específica para los clientes de servicio Web XML?

Sistema.Web.Services.dll.

1.9.3. Preguntas arquitectura de los servicios Web

Actividad asignada en clase

1.9.4. Preguntas plataformas de desarrollo

Actividad asignada en clase

1.9.5. Preguntas esquemas de implementación

Actividad asignada en clase

1.9.6. Preguntas consideraciones de seguridad

Actividad asignada en clase

2. Descripción de Información: XML

2.1. Aplicación de XML – Servicios Web

En esencia, los servicios Web son aplicaciones XML (Extensible Markup Language) con mapeo a programas, objetos o bases de datos o a funciones de negocios. Usando un documento XML creado en la forma de un mensaje, un programa envía una petición a un servicio Web por la red y opcionalmente recibe una respuesta, también en la forma de un documento XML. Los estándares de servicios Web definen el formato del mensaje, específicamente la interfaz a la que se envía un mensaje, describen las convenciones para el mapeo de los contenidos del mensaje en y fuera de los programas que implementan el servicio y definen los mecanismos para publicar y descubrir las interfaces de servicios de Web.

El Lenguaje XML, como el Hypertext Markup Language, comparten una ascendencia común con el Standard Generalized Markup Language (SGML). Una de las características de SGML fue la separación de formato y contenido. Si un documento fue producido para A4 o en formato carta, por ejemplo, el formato fue descrito independientemente del contenido del documento. El mismo documento podría ser por consiguiente la salida en formatos múltiples sin cambiar el contenido. Este principio de lenguajes de marcado es aplicado a los servicios Web a través de la separación del caso del documento que contiene los datos y el esquema que describe las estructuras de datos y tipos, incluyendo la información semántica útil para el mapeo del documento a múltiples lenguajes de programación y sistemas del software.

XML representa un número grande de especificaciones, muchas de las cuales son más pertinentes al procesamiento del documento que para el procesamiento de la información. En este tema trataremos las especificaciones de XML y tecnologías más importante para servicios Web que en general puede decirse que más allá del marcado para proveer los medios de estructuración y serialización de datos.

Para iniciar el tema 2, en seguida se describe un escenario sobre actividades de una agenda de propósito general, en la que se utiliza un servicio Web con manejo de operaciones de XML.

2.1.1. Escenario Actividades de Agenda

Las actividades de una agenda son habituales en cualquier esfera, sea oficina, escuela, hogar, etc. Por consiguiente en este escenario planteamos la implementación de una agenda con un servicio Web, en la que:

- Las actividades deben de almacenarse en un archivo XML.
- Debe especificarse la ubicación de este archivo de trabajo.
- Al solicitar una actividad, un servicio Web abre el archivo y se devuelve la información.
- Además pueden agregarse nuevas actividades.

Con el desarrollo de este escenario, se experimentará como trabajan las técnicas de acceso a la información con XML.

2.1.2. Actividades de aprendizaje

Implementación del escenario de actividades de una agenda

Con este servicio Web se carga un archivo, lee, cambia y se contesta fuera. Se cubren técnicas de acceso a la información con XML.

Instrucciones

1. Lea detenidamente el "Escenario Actividades de Agenda".
2. Revise y compile el código de programa que se proporciona en el rubro "Código Ejemplo Servicio Web con XML" del Anexo B.
3. En base a las actividades previas, discutir en equipo la implementación del escenario en Visual C#, se recomienda el uso de las bibliotecas de XML del .NET Framework.
4. Recuerde que debe seguir los lineamientos del aprendizaje basado en problemas, descrito en la sección "Metodología".
5. Se recomiendan los enlaces de interés de la sección: "Anexo E. Fuentes de Información" así como la resolución de ejercicios y cuestionarios del subtema en curso.

2.2. Documentos XML

Un documento XML es realmente una colección de datos que consiste tanto de una estructura física como lógica. Físicamente, el documento consiste en información textual. Contiene entidades que pueden hacer referencia otras entidades que se localizan en otra parte en memoria, en un disco fijo, o, más pretenciosamente, en el Web. La estructura lógica de un documento XML incluye instrucciones de procesamiento, declaraciones, comentarios y elementos. Los documentos XML contienen texto ordinario que representa marcado o datos de caracteres.

La forma general de un documento XML se parece algo así:

```
<?xml version="1.0" ?>
<Car Year="2002">
   <Make>Chevrolet</Make>
   <Model>Corvette<Model>
   <Color>Gunmetal</Color>
</Car>
```

Conocemos que éste es un documento XML debido a la declaración XML en la primera línea que dice que este documento conforma a XML versión 1.0. En este ejemplo, <Car /> es el elemento raíz o elemento documento de este documento XML, <Make /> es simplemente uno de los elementos hijo contenido dentro de <Car /> y Year es un atributo del elemento <Car />.

Es común encontrar instrucciones de procesamiento incrustadas dentro del documento, pero no son consideradas parte del contenido del documento. Se usan para comunicar la información para el código a nivel de la aplicación sin cambiar el significado del contenido del documento XML. Como ejemplo la siguiente expresión:

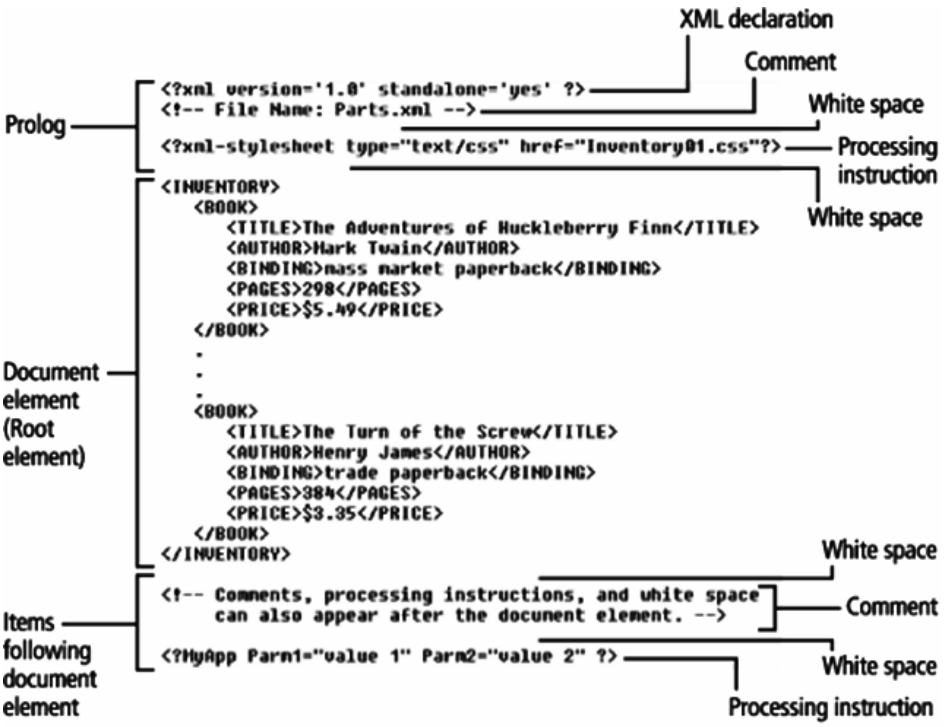
```
<?xml version="1.0" ?>
```

Cualquier documento de texto es considerado un documento XML bien-formado si conforma delante un conjunto de restricciones en la especificación XML. Por ejemplo, una restricción muy importante es la limitación a uno y sólo un elemento de la raíz en un documento. Un documento XML es considerado válido si está bien formado, si tiene asociado una definición de tipo de documento (DTD, Document Type Definition) o un Esquema XML y si la instancia del documento dado obedece esa definición. Los documentos DTD o Esquema XML actúan como una plantilla que el documento XML asociado precisamente debe emparejar. De otra forma, hay un problema con el formato de la instancia del documento y el documento

entero no es considerado válido. En seguida trataremos más a detalle documentos XML bien-formados y en subtemas por separado DTDs y esquemas XML.

2.2.1. Documentos XML bien formados

Un documento XML consiste en dos partes principales: el prólogo y el elemento documento (qué también es conocido como el elemento raíz). Además, siguiendo el elemento documento, un documento XML bien-formado puede incluir comentarios, instrucciones de procesamiento y el espacio blanco (espacios, tabuladores o interrupciones de línea). En seguida tenemos un ejemplo de un documento XML bien-formado que muestra las partes diferentes del documento y los elementos pueden agregarse a cada parte:




```
<?xml version='1.0' standalone='yes' ?>
<!-- File Name: Parts.xml -->
<?xml-stylesheet type="text/css" href="Inventory01.css"?>
<INVENTORY>
   <BOOK>
      <TITLE>The Adventures of Huckleberry Finn</TITLE>
      <AUTHOR>Mark Twain</AUTHOR>
      <BINDING>mass market paperback</BINDING>
      <PAGES>298</PAGES>
      <PRICE>$5.49</PRICE>
   </BOOK>
   .
   .
   .
   <BOOK>
      <TITLE>The Turn of the Screw</TITLE>
      <AUTHOR>Henry James</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>384</PAGES>
      <PRICE>$3.35</PRICE>
   </BOOK>
</INVENTORY>
<!-- Comments, processing instructions, and white space
     can also appear after the document element. -->
<?MyApp Parm1="value 1" Parm2="value 2" ?>
```

La primera línea del documento del ejemplo consiste en la declaración del XML, la cual es optativa aunque se recomienda incluirla. Identifica al documento como XML, especifica el número de versión del XML, además provee un lugar a incluir en dos piezas de información optativas: la declaración de la codificación y la declaración del documento standalone. La declaración de la codificación especifica el plan de la codificación usado para los caracteres en el documento. La declaración del documento standalone indica si el documento contiene declaraciones de marcado externas que afectan el contenido del documento. Si incluyen una declaración del XML, debe aparecer al mismo principio del

documento. (No les permiten incluir los caracteres de espacio en blanco incluso antes de la declaración del XML).

El documento del ejemplo incluye un comentario en el prólogo y otro comentario que sigue el elemento documento. Sobre estos comentarios se tratará más adelante.

El documento también contiene dos líneas del espacio en blanco en el prólogo y dos líneas más en blanco que siguen el elemento documento. El espacio blanco consiste en uno o más espacios, tabulador o carro de retorno. Para hacer más legible, pueden agregar libremente también el espacio blanco entre marcado XML, tal como etiquetas de comienzo, etiquetas de fin, comentarios e instrucciones de procesamiento. El documento del ejemplo tiene una instrucción de procesamiento en el prólogo y otro que sigue el elemento documento.

Finalmente, el documento del ejemplo incluye la parte trascendental de un documento XML: el elemento documento. Crear el elemento documento y los elementos anidados que contiene es el enfoque de este subtema.

Agregando elementos al documento

Los elementos en un documento XML contienen la información del documento real (ejemplo anterior: títulos, autores, precios y otra información del registro de inventario de libros) e indican la estructura lógica de esta información.

Los elementos se organizan en una jerarquía similar al árbol, con elementos anidados dentro de otros elementos. El documento debe tener un elemento documento exactamente uno al nivel de la cima (el elemento documento o elemento raíz) con todos los otros elementos anidados dentro de él. En seguida se muestra un documento XML bien-formado:

```
<?xml version="1.0"?>
<!-- A well-formed XML document. -->
<INVENTORY>
   <BOOK>
      <TITLE>The Adventures of Huckleberry Finn</TITLE>
      <AUTHOR>Mark Twain</AUTHOR>
      <BINDING>mass market paperback</BINDING>
      <PAGES>298</PAGES>
      <PRICE>$5.49</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>Leaves of Grass</TITLE>
      <AUTHOR>Walt Whitman</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>462</PAGES>
```

```
      <PRICE>$7.75</PRICE>
   </BOOK>
</INVENTORY>
```

El documento siguiente, no está bien-formado porque tiene dos elementos, en lugar de uno, al nivel de la cima:

```
<?xml version="1.0"?>
<!-- This document is NOT well-formed. -->
<BOOK>
   <TITLE>The Adventures of Huckleberry Finn</TITLE>
   <AUTHOR>Mark Twain</AUTHOR>
   <BINDING>mass market paperback</BINDING>
   <PAGES>298</PAGES>
   <PRICE>$5.49</PRICE>
</BOOK>
<BOOK>
   <TITLE>Leaves of Grass</TITLE>
   <AUTHOR>Walt Whitman</AUTHOR>
   <BINDING>hardcover</BINDING>
   <PAGES>462</PAGES>
   <PRICE>$7.75</PRICE>
</BOOK>
```

También deben anidarse propiamente elementos. Es decir, si un elemento (delimitado por una etiqueta de comienzo y una etiqueta de fin) empieza dentro de otro elemento, también debe terminar dentro de ese mismo elemento. En otros términos, no pueden solapar los elementos como a veces se hace para HTML.

Como se ha visto, un elemento normalmente consiste en una etiqueta de comienzo, contenido y una etiqueta de fin. El nombre que aparece al principio de la etiqueta de comienzo y en la etiqueta de fin (caso del ejemplo, TITLE) identifica el tipo de elementos, no el elemento específico. Un documento puede contener así más de un elemento que tiene el mismo nombre (como los elementos BOOK o TITLE del ejemplo) todos de los cuales se considera que son del mismo tipo.

Cuando agregan un elemento a su documento XML, pueden seleccionar cualquier nombre de tipo que quieran, con tal de que sigan estas pautas:

- El nombre debe empezar con una letra o subrayado (_), seguido por cero o más letras, dígitos, puntos (.), guiones (-) o subrayados.

- La especificación de XML establece que nombres al principio con las letras xml son "reservadas para la estandarización." Por tanto deben de omitirse para nombres al principio y evitar problemas futuros.

- Pueden asignar el nombre del elemento a un espacio de nombres poniendo un prefijo del espacio de nombres seguido por el signo dos puntos (:) frente del nombre del elemento.

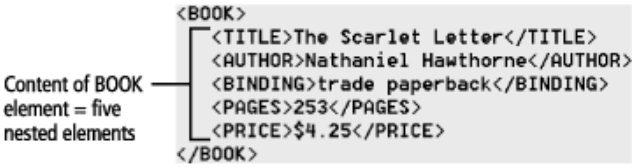
Los siguientes son nombres de tipos de elementos legales:

```
Part
_1stPlace
A
B-SECTION
Street.Address.1
books:TITLE <!-- Allowed only if the 'books' namespace prefix
                 has been properly declared. -->
```

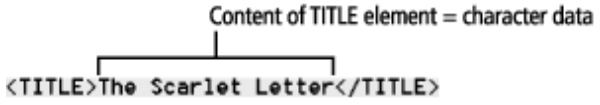
Tipos de contenido en un elemento

El contenido de un elemento es el texto entre la etiqueta de comienzo y la etiqueta de fin. Pueden incluir los tipos siguientes de elementos en el contenido de un elemento:

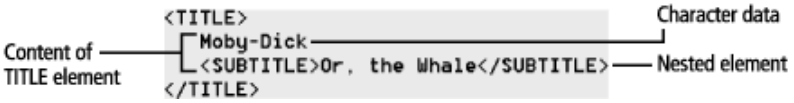
- **Elementos anidados**. En seguida, el elemento BOOK contienen los elementos anidados como su contenido:



- **Datos de caracteres**. Es texto que expresa el contenido de información de un elemento, como el título de un libro específico en un elemento TITLE:



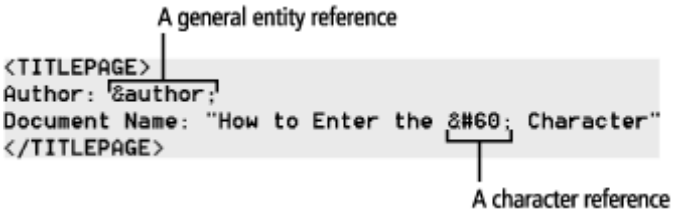
Otro ejemplo de contenido en un elemento que consiste en datos de caracteres y un elemento anidado:



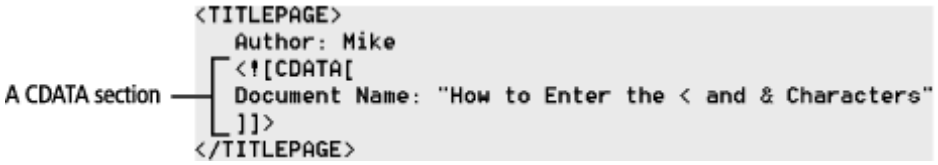
Al agregar los datos de caracteres a un elemento, pueden insertar cualquier carácter como parte de los datos del carácter excepto el corchete angular izquierdo (<), el signo ampersand (&) o la cadena]]>. Esto debido a que el

analizador sintáctico de XML busca los datos de caracteres de un elemento que buscando el marcado de XML. Si quieren insertar < o & como una parte integral de los datos de caracteres, pueden usar una sección CDATA. También pueden insertar <, & o cualquier otro, usando una referencia de carácter y pueden insertar ciertos caracteres usando referencias de entidad general predefinida (como < o & par insertar < o &).

- **Referencias de entidad general o referencias de carácter**. En seguida se muestra un elemento conteniendo uno de cada uno:



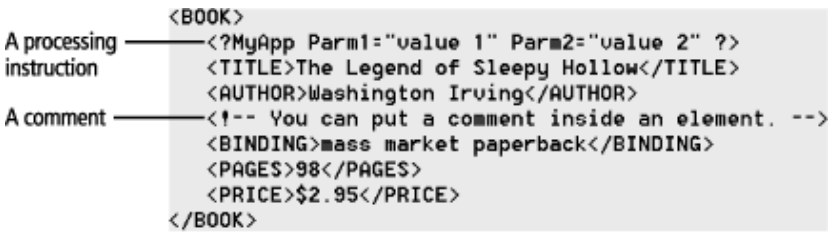
- **Secciones CDATA**. Una sección de CDATA es un bloque de texto en el que pueden insertar cualquier carácter libremente excepto la cadena]]>. En seguida un ejemplo de una sección CDATA en un elemento:



- **Instrucciones de procesamiento**. Una instrucción de procesamiento proporciona información a la aplicación de XML.

- **Comentarios**. Un comentario es una anotación a su documento XML que las personas pueden leer pero que el procesador del XML ignora y (opcionalmente) pasa sobre la aplicación.

  En seguida, un elemento conteniendo una instrucción de procesamiento y un comentario:



Agregando atributos a los elementos

En la etiqueta de comienzo de un elemento o en una marca de elemento vacío, pueden incluir una o más especificaciones de atributo. Una especificación de atributo es un par nombre-valor que es asociado con el elemento.

En el ejemplo siguiente, el elemento BOOK incluye dos atributos, Category y Display:

```
<BOOK Category="fiction" Display="emphasize">
   <TITLE>The Marble Faun</TITLE>
   <AUTHOR>Nathaniel Hawthorne</AUTHOR>
   <BINDING>trade paperback</BINDING>
   <PAGES>473</PAGES>
   <PRICE>$10.95</PRICE>
</BOOK>
```

El elemento vacío siguiente incluye un atributo llamado Source que indica el nombre del archivo que contiene una imagen a ser desplegada:

```
<COVER_IMAGE Source="Faun.gif" />
```

Reglas para crear los atributos

Como hemos visto, una especificación de atributo consiste en un nombre del atributo seguido por el carácter igual (=), luego seguido por un valor del atributo. Pueden seleccionar cualquier nombre del atributo que quieran, con tal de que siga estas reglas:

- El nombre debe empezar con una letra o un subrayado (_), seguido por cero o más letras, dígitos, dedos, puntos (.), guiones (-) o subrayado.

- La especificación del XML determina que nombres al principio con el prefijo xml son reservado para la estandarización. Por lo que debe de omitir este nombramiento.

- Pueden asignar el nombre del atributo a un espacio de nombres (namespace) poniendo un prefijo del espacio de nombres seguido por dos puntos (:) en seguida el nombre del atributo. El propio nombre del atributo todavía debe conformarse conforme las primeras dos reglas.

- Un nombre del atributo particular sólo puede aparecer una vez en la misma etiqueta de comienzo o marca de elemento vacío. Pueden, sin embargo, tener idénticamente nombrados los atributos en una etiqueta si están en espacios de nombres diferentes.

Ejemplo, los nombres de atributos en las etiquetas de inicio son legales:

```
<ANIMATION FileName="Waldo.ani">
<LIST _1stPlace="Sam">
<ENTRY Zip.Code="94941">
<STANZA xml:space="preserve"> <!-- 'xml:' is allowed because it
                                  designates a predefined
                                  namespace. -->
```

Ejemplo, los nombres de atributos en las etiquetas de inicio son ilegales:

```
<!-- Duplicated attribute name in same tag: -->
<ANIMATION FileName="Waldo1.ani" FileName="Waldo2.ani">
<LIST 1stPlace="Sam"> <!-- Digit not allowed as
                           first character. -->
```

Reglas para los valores del atributo

El valor que asignan a un atributo es una serie de caracteres delimitada con comillas, denominado como una cadena encomiada o literal. Pueden asignar cualquier valor literal a un atributo, con tal de que se observen estas reglas:

- La cadena puede delimitarse con comillas simples (') o comillas dobles (").
- La cadena no puede contener que el mismo carácter de comillas que la delimitan.
- La cadena puede contener referencias de entidad general o referencias de carácter.
- La cadena no puede incluir el carácter ampersand (&), excepto para iniciar un carácter o referencia de entidad.
- La cadena no puede incluir el carácter angular izquierdo (<).

Las especificaciones de atributo siguientes son ilegales:

```
<EMPLOYEE Status=""downsized""> <!-- Can't use delimiting quote
                                    within string. -->
<ALBUM Type="<CD>"> <!-- Can't use < within string. -->
<WEATHER Forecast="Cold & Windy"> <!-- Can't use & except to
                                      start a reference. -->
```

Si quieren incluir las comillas dobles (") dentro del valor del atributo, pueden usar las comillas simples (') para delimitar la cadena, como en este ejemplo:

```
<EMPLOYEE Status='"downsized"'> <!-- Legal attribute value. -->
```

Igualmente, incluir comilla simple dentro del valor, delimitando con comillas dobles:

```
<CANDIDATE name="W.T. 'Bill' Bagley"> <!-- Legal atrr. value. -->
```

2.2.2. Uso de espacios de nombres

Combinar datos de XML de varias fuentes puede producir conflictos en los nombres de elementos y atributos. Por ejemplo, el seguimiento de respaldo de sus BOOKs en un documento XML:

```
<?xml version="1.0"?>
<COLLECTION>
   <ITEM Status="in">
      <TITLE>The Adventures of Huckleberry Finn</TITLE>
      <AUTHOR>Mark Twain</AUTHOR>
      <PRICE>$5.49</PRICE>
   </ITEM>
   <ITEM Status="out">
      <TITLE>Leaves of Grass</TITLE>
      <AUTHOR>Walt Whitman</AUTHOR>
      <PRICE>$7.75</PRICE>
   </ITEM>
   <ITEM Status="out">
      <TITLE>The Legend of Sleepy Hollow</TITLE>
      <AUTHOR>Washington Irving</AUTHOR>
      <PRICE>$2.95</PRICE>
   </ITEM>
   <ITEM Status="in">
      <TITLE>The Marble Faun</TITLE>
      <AUTHOR>Nathaniel Hawthorne</AUTHOR>
      <PRICE>$10.95</PRICE>
   </ITEM>
</COLLECTION>
```

Que es el seguimiento de respaldo de sus CDs en otro documento XML:

```
<?xml version="1.0"?>
<COLLECTION>
   <ITEM>
      <TITLE>Violin Concerto in D</TITLE>
      <COMPOSER>Beethoven</COMPOSER>
      <PRICE>$14.95</PRICE>
   </ITEM>
   <ITEM>
```

```
      <TITLE>Violin Concertos Numbers 1, 2, and 3</TITLE>
      <COMPOSER>Mozart</COMPOSER>
      <PRICE>$16.49</PRICE>
   </ITEM>
</COLLECTION>
```

Ahora suponga que quieran combinar estos documentos en un solo documento XML que rastrea ambas colecciones y quieran combinar las aplicaciones (quizás los scripts de la página Web) que gestionan estos documentos en una sola aplicación (quizá un script combinado de la página Web). El problema es que en el documento agrupado la nueva aplicación no podría diferenciar los elementos libro de los elementos del CD (elementos ITEM), títulos del libro de los títulos del CD (elementos TITLE) y precios del libro de los precios del CD (elementos PRICE). Esto sería difícil, por ejemplo, para listar todos los elementos del CD, desplegar todos los títulos de libros o encontrar el precio promedio de sus CD. Aunque pudieran inventar una manera de distinguir los tipos diferentes de elementos ITEM, TITLE y PRICE, probablemente, la solución involucraría reescribiendo partes de o el documento XML o la aplicación.

El mecanismo de espacio de nombres de XML proporciona una manera más fácil. Les permite diferenciar dos o más elementos fácilmente o dos o más atributos que tienen el mismo nombre asignando cada uno a un espacio de nombres separado. Esto lo hacemos para los documentos previos en un documento combinado y fusionando. En seguida se muestra:

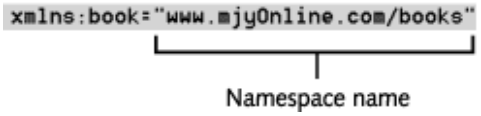
```
<?xml version="1.0"?>

<!-- File Name: Collection.xml -->

<COLLECTION
   xmlns:book="http://www.mjyOnline.com/books"
   xmlns:cd="http://www.mjyOnline.com/cds">

   <book:ITEM Status="in">
      <book:TITLE>The Adventures of Huckleberry Finn</book:TITLE>
      <book:AUTHOR>Mark Twain</book:AUTHOR>
      <book:PRICE>$5.49</book:PRICE>
   </book:ITEM>
   <cd:ITEM>
      <cd:TITLE>Violin Concerto in D</cd:TITLE>
      <cd:COMPOSER>Beethoven</cd:COMPOSER>
      <cd:PRICE>$14.95</cd:PRICE>
   </cd:ITEM>
   <book:ITEM Status="out">
      <book:TITLE>Leaves of Grass</book:TITLE>
      <book:AUTHOR>Walt Whitman</book:AUTHOR>
      <book:PRICE>$7.75</book:PRICE>
```

```
   </book:ITEM>
   <cd:ITEM>
      <cd:TITLE>Violin Concertos Numbers 1, 2, and 3</cd:TITLE>
      <cd:COMPOSER>Mozart</cd:COMPOSER>
      <cd:PRICE>$16.49</cd:PRICE>
   </cd:ITEM>
   <book:ITEM Status="out">
      <book:TITLE>The Legend of Sleepy Hollow</book:TITLE>
      <book:AUTHOR>Washington Irving</book:AUTHOR>
      <book:PRICE>$2.95</book:PRICE>
   </book:ITEM>
   <book:ITEM Status="in">
      <book:TITLE>The Marble Faun</book:TITLE>
      <book:AUTHOR>Nathaniel Hawthorne</book:AUTHOR>
      <book:PRICE>$10.95</book:PRICE>
   </book:ITEM>
</COLLECTION>
```

Para usar un espacio de nombres, deben declararlo dentro de una etiqueta de comienzo del elemento usando una especificación de atributo de propósito especial. En el código del documento previo, se declaran ambos espacios de nombres en la etiqueta de comienzo del elemento de raíz COLLECTION:

```
 <COLLECTION
   xmlns:book="http://www.mjyOnline.com/books"
   xmlns:cd="http://www.mjyOnline.com/cds">
```

En seguida se muestra la declaración de espacio de nombres book:



En la declaración, el nombre del espacio de nombres es el identificador real del espacio de nombres y debe ser un URI (Uniform Resource Identifier). Un espacio de nombres se identifica usando un URI, no porque el procesador del XML o la aplicación necesite tener acceso a cualquier información realmente en ese URI, sino simplemente porque los URIs son globalmente únicos. En las especificaciones del W3C usa el más nuevo y más ancho término URI en lugar del término URL más familiar pero más rígido. El término URI es exhaustivo, refiriéndose a Internet como cadenas de direccionamiento de recursos que utilizan cualquiera del presente o futuros esquemas de direccionamiento.

El xmlns en la declaración del espacio de nombres es un prefijo predefinido (no tienen que declararlo) que se usa específicamente para definir los espacios de nombres, book es el prefijo del espacio de nombres que es una notación escrita

corta para el nombre del espacio de nombres completo. Se usa el prefijo del espacio de nombres para calificar los nombres de elementos específicos o atributos, indicando que pertenecen al espacio de nombres. Un prefijo del espacio de nombres válido debe empezar con una letra o subrayado (_), en seguida por cero o más letras, dígitos, puntos (.), guiones (-) o subrayados, no debe usar el prefijo con las letras xml, que son para estandarización.

Pueden usar el prefijo en cualquier parte dentro del elemento en el que el espacio de nombres ha estado definido o dentro de cualquiera de sus elementos anidados; este conjunto de elementos es conocido como el alcance del espacio de nombres.

Resumiendo, el mecanismo del espacio de nombres hace XML más modular, permite unir documentos XML, así como las aplicaciones que los procesan, sin riesgo de contradicción de nombres de elementos y atributos. Además, los estándares del XML para mejorar a menudo requieren que usen elementos y atributos específicos para que las aplicaciones que se ocupan de los documentos puedan reconocerlos. Los nombres de estos elementos y atributos pertenecen a los únicos espacios de nombres para evitar chocar con nombres que ha decidido usar en sus documentos. En seguida se tienen ejemplos de esos estándares:

- La propia especificación del XML incluye los atributos reservados xml:space (espacio) y xml:lang (lenguaje). Porque pertenecen a un único espacio de nombres que se hace referencia usando el prefijo de espacio de nombres xml predefinido. Debido a que el prefijo de espacio de nombres xml es definido por la propia especificación del XML, no tienen que declararlo.

- Creando esquemas XML usando un conjunto estándar de elementos que pertenecen al espacio de nombres nombrado.

- Desplegando un documento mediante CSS (cascading style sheet), pueden insertar los elementos del HTML en su documento usando elementos del XML que representan los elementos del HTML y que se prologan con el prefijo de espacio de nombres de html que necesita declararse.

- Cuando crea una hoja de estilos de XSLT, usa un conjunto estándar de elementos que pertenecen al espacio de nombres nombrado.

2.2.3. Actividades de aprendizaje

Ejercicio sobre documentos XML bien formados

El escenario consiste de un documento XML sobre pedidos de productos de una empresa proveedora.

Instrucciones

1. En base al Material de Apoyo “2.2 Documentos XML”, utilice el archivo de documento XML bien formado order.xml que se proporciona en el rubro “Código documento XML bien formado” del Anexo C; para realizar las tareas siguientes:

Represente mediante un esquema jerárquico los elementos del documento XML, verificando porqué está bien formado.

Sugerencia. Al abrir un archivo XML mediante el explorador, este realiza análisis sintáctico.

Implemente un archivo nombrado como orderid.xml, considerando la información de order.xml, para formar un documento XML en donde se clasifique la información por clave de pedidos (order ID).

Sugerencia. Puede utilizar atributo de elementos.

2. Coloque su trabajo en “Tarea Individual Ejercicio (2a)”

2.3. Validación XML: DTD

XML es un lenguaje de meta-marcado que es totalmente extensible. Con tal de que se forme bien, pueden crear cualquier estructura de XML que desean para describir sus datos. Sin embargo, no pueden estar seguros que la estructura realizada con tanto tiempo y esfuerzo no será cambiada por otro. Necesita tratar una manera de asegurar que la estructura de XML no pueda cambiarse al azar. Esta seguridad para la estructura de documento XML es vital para las aplicaciones de comercio electrónico y procesamiento de negocio a negocio, entre otras cosas. Esto es donde la definición de tipo de documento (DTD, Document Type Definition) participa. Un DTD provee un mapa de carretera a describir y documentar la estructura que constituye un documento XML. Un DTD puede usarse para determinar la validez de un documento XML.

2.3.1. Introducción a DTD

Una definición del tipo de documento permite definir un conjunto de reglas para un documento XML para hacerlo válido. Un documento XML es considerado "bien formado" si ese documento es en sintaxis correcto según las reglas de XML 1.0. Sin embargo, eso no significa que el documento es necesariamente válido. Para ser considerado válido, un documento XML debe validarse o verificado, contra un DTD. El DTD definirá los elementos requeridos por un documento XML, los elementos que son optativos, el número de veces que un elemento debe o puede ocurrir y el orden en el que deben anidarse elementos. El marcado de DTD también define el tipo de datos que ocurrirán en un elemento XML y los atributos que pueden asociarse con esos elementos. Un documento, aun cuando este bien formado, no es considerado válido si no sigue las reglas definidas en el DTD.

Un DTD puede ser interno, residiendo dentro del cuerpo de un solo documento XML. También puede ser externo, referenciado por el documento XML. Un solo documento XML podría tener ambos una porción (o subconjunto) de sus DTD que es interna y una porción que es externa. Un solo DTD externo puede ser referenciado por muchos documentos XML, esto es un buen almacén para los tipos globales de definiciones (definiciones que aplican a todos los documentos). Un DTD interno es bueno para usar en reglas que sólo se aplican a un documento específico. Si un documento tiene los subconjuntos de DTD internos y externos, las reglas internas anulan las reglas externas en casos dónde el mismo elemento está definido en ambos subconjuntos.

En seguida mostramos un par de ejemplos de DTDs simples, interno y externo, para mejor comprensión hasta este punto.

Ejemplo de un DTD interno:

```
<?xml version="1.0"?>
<!DOCTYPE message [
<!ELEMENT message (#PCDATA)>
]>
<message>
Let the good times roll!
</message>
```

El DTD interno esta contenido dentro de la declaración de tipo de documento con la que empieza <! DOCTYPE y termina con]>. Una declaración del tipo de documento aparecerá entre la declaración de XML y el inicio del propio documento (el documento o elemento raíz) e identificará esa sección del documento XML como contener una Definición del Tipo de Documento. Siguiendo la declaración del tipo de documento (DOCTYPE), el elemento raíz del documento XML está

definido (en este caso, message). El DTD nos dice que este documento tendrá un solo elemento, message, que contendrá los datos de caracteres analizados (#PCDATA).

Ahora, veamos cómo se unirían este mismo DTD y documento XML si el DTD es externo.

Ejemplo de un DTD externo:

```
<?xml version="1.0"?>
<!DOCTYPE message SYSTEM "message.dtd">
<message>
Let the good times roll!
</message>
```

El DTD externo esta contenido en un archivo separado, message.dtd. Se supone que los contenidos de message.dtd son igual que los contenidos del DTD interno previamente visto. La palabra clave SYSTEM en la declaración del tipo de documento nos deja saber que el DTD va a ser encontrado en un archivo separado. Una URL podría usarse para definir la situación del DTD. Por ejemplo, en lugar de message.dtd, la declaración del tipo de documento podría especificar algo como ../DTD/message.dtd.

Ahora mostraremos un ejemplo de documento NO valido de acuerdo al DTD definido para los ejemplos previos.

Ejemplo de un documento no valido de acuerdo al DTD definido:

```
<?xml version="1.0"?>
<!DOCTYPE message SYSTEM "message.dtd">
<message>
<text>
Let the good times roll!
</text>
</message>
```

Aunque éste es un documento XML bien-formado, no es válido. Cuando este documento se valida contra message.dtd, una bandera se levantará porque message.dtd no define a un elemento nombrado como text.

2.3.2. Estructura de DTD

La estructura de un DTD consiste en una declaración del tipo de documento, elementos, atributos, entidades y varias otras palabras claves menores. Trataremos estos puntos en ese orden.

La declaración de tipo de documento

Para hacer referencia a un DTD desde un documento XML, una declaración de tipo de documento debe ser incluida en el documento XML. La sintaxis de la declaración de tipo de documento para el documento XML es como sigue:

```
<!DOCTYPE rootelement SYSTEM | PUBLIC DTDlocation [
internalDTDelements ] >
```

- El signo de admiración (!) se usa para significar el principio de la declaración.
- DOCTYPE es la palabra clave para denota que es una Definición del Tipo de Documento.
- rootelement es el nombre del elemento raíz o elemento documento del documento XML.
- SYSTEM y PUBLIC son palabras claves usadas para designar que el DTD esta incluido en un documento externo. Aunque el uso de estas palabras claves es opcional, hacer referencia a un DTD externo tendrían que usar uno o el otro. La palabra clave SYSTEM se usa en turno con una URL para localizar el DTD. La palabra clave PUBLIC especifica alguna ubicación pública que normalmente será alguna referencia del recurso de aplicación específica.
- internalDTDelements son las declaraciones de DTD internos. Estas declaraciones siempre se pondrán dentro de abrir ([) y cerrar (]) los corchetes.

Es posible para una declaración de tipo de documento contener ambos un subconjunto de DTD externo y un subconjunto de DTD interior. En esta situación, las declaraciones interiores tienen precedencia sobre los externos. Considere el fragmento de declaración de tipo de documento mostrado en seguida:

Listado de DTDs interiores y externos

```
<!DOCTYPE rootelement SYSTEM "http://www.myserver.com/mydtd.dtd"
[
<!ELEMENT element1 (element2,element3)>
<!ELEMENT element2 (#PCDATA)>
<!ELEMENT element3 (#PCDATA)>
]>
```

En el listado previo, vemos que la declaración de tipo de documento hace referencia a un DTD externo. Hay también un subconjunto interno del DTD incluido en la declaración de tipo de documento. Cualquier regla en el DTD externo que aplica a los elementos definidos en el DTD interno será anulada por las reglas del DTD interno.

Elementos DTD

Todos los elementos en un documento XML válido están definidos con una declaración de elemento en el DTD. Una declaración de elemento define el nombre y todos los contenidos permitidos de un elemento. Los nombres del elemento deben iniciar con una letra o un subrayado y pueden contener cualquier combinación de letras, números, subrayados, guiones y puntos. Los nombres del elemento nunca deben arrancar con la cadena "xml". No deben usarse el signo dos puntos (:) en nombres del elemento porque normalmente se usa para hacer referencia de los espacios de nombres.

Cada elemento en el DTD debe definirse con la sintaxis siguiente:

```
<!ELEMENT elementname rule >
```

- ELEMENT es el nombre de etiqueta que especifica que ésta es una definición de elemento.
- elementname es el nombre del elemento.
- rule es la definición con la que debe conformarse el contenido de datos del elemento.

En un DTD, los elementos se procesan de arriba abajo. La validación con el analizador sintáctico de XML esperará el orden de la aparición de elementos en el documento XML para emparejar el orden de elementos definido en el DTD. Por consiguiente, los elementos en un DTD deben aparecer en el orden que quieren que ellos aparezcan en un documento XML. Si los elementos en un documento XML no emparejan el orden del DTD, el documento XML no será considerado válido por la validación del analizador sintáctico.

En seguida el listado muestra un DTD que define la clasificación de elementos para referenciar los documentos XML.

```
<!ELEMENT contactlist (fullname, address, phone, email) >
<!ELEMENT fullname (#PCDATA)>
<!ELEMENT address (addressline1, addressline2)>
<!ELEMENT addressline1 (#PCDATA)>
<!ELEMENT addressline2 (#PCDATA)>
```

```
<!ELEMENT phone (#PCDATA)>
<! ELEMENT email (#PCDATA)>
```

El primer elemento en el DTD, contactlist, es el elemento documento. La regla para este elemento es que contiene (es el elemento padre de) los elementos fullname, address, phone y email. La regla para los elementos fullname, phone y email es que cada uno contiene datos de caracteres analizados (#PCDATA). El elemento address tiene dos elementos hijos: addressline1 y addressline2. Estos dos elementos hijos contienen #PCDATA. Este DTD define una estructura de XML que es en profundidad de dos niveles anidados. El elemento raíz, contactlist, tiene cuatro elementos hijos. El elemento address es, a su vez, padre para dos más elementos. Para que un documento XML que hace referencia a este DTD para que sea válido, debe ponerse en el mismo orden y debe tener la misma profundidad de anidación.

En seguida, el documento XML es un documento válido porque sigue las reglas puestas en el listado previo:

```
<?xml version="1.0"?>
<!DOCTYPE contactlist SYSTEM "contactlist.dtd">
<contactlist>
<fullname>Bobby Soninlaw</fullname>
<address>
<addressline1>101 South Street</addressline1>
<addressline2>Apartment #2</addressline2>
</address>
<phone>(405) 555-1234</phone>
<email>bs@mail.com</email>
</contactlist>
```

La segunda línea de este documento XML es la declaración de tipo de documento que hace referencia a contactlist.dtd. Éste es un documento XML válido porque se forma bien y obedece la definición estructural puesta en el DTD.

Reglas de elementos DTD

Hay dos tipos básicos de reglas en las que los elementos deben caer. El primer tipo trata con el contenido. El segundo tipo trata con la estructura.

Reglas de contenido

Las reglas de contenido para elementos tratan los datos reales que los elementos definidos pueden contener. Estas reglas incluyen la regla ANY, la regla EMPTY, y la regla #PCDATA.

Regla ANY

La regla ANY es lo como lo que parece: el elemento puede contener otros elementos y/o datos de caracteres normales (cualquier cosa con tal de que se forme bien). Un elemento que usa la regla ANY aparecería como sigue:

```
<!ELEMENT elementname ANY>
```

El fragmento siguiente de XML muestra que todo es válido dado la definición de elementname.

```
<elementname>
This is valid content
</elementname>

<elementname>
<anotherelement>
This is more valid content
</anotherelement>
This is still valid content
</elementname>

<elementname>
<emptyelement />
<yetanotherelement>
This is still valid content!
</yetanotherelement>
Here is more valid content
</elementname>
```

Deben ver desde el fragmento por qué no siempre es una gran idea para usar la regla ANY. Todos los tres fragmentos que contienen el elemento elementname son válidos. Hay, en efecto, ninguna validación para este elemento. El Uso de la regla ANY probablemente debe limitarse a casos dónde los datos de XML serán texto libre u otros tipos de datos que serán muy variables y tendrán dificultades que conforman a una estructura fija.

La regla EMPTY

Esta regla es lo contrario de la regla ANY. Un elemento que está definido con esta regla no contendrá ningún dato. Sin embargo, un elemento con la regla EMPTY todavía podría contener los atributos. El elemento siguiente es un ejemplo de la regla EMPTY:

```
<!ELEMENT elementname EMPTY>
```

Este concepto se ve mucho en HTML. Hay muchas marcas como la etiqueta break (
) y la etiqueta paragraph (<p />) que siguen esta regla. Ni uno de estas marcas no contiene cualquier datos, pero los dos son muy importantes en documentos HTML. El ejemplo mejor de una etiqueta vacía usado en HTML es la etiqueta image (<img>). Aunque la etiqueta image no contiene ningún datos, tiene atributos que describen la ubicación y pantalla de una imagen para un explorador Web.

En XML, la regla EMPTY podría usarse para definir elementos vacíos que contienen la información de diagnóstico para el procesamiento de datos. Los elementos vacíos también podrían crearse para sostener metadatos que describe los contenidos del documento XML para propósitos de indexación. Los elementos vacíos podrían usarse para proveer las pistas a aplicaciones que darán los datos para visualizar.

La regla #PCDATA

La regla #PCDATA indica datos de caracteres analizados que estarán incluidos en el elemento. El elemento siguiente muestra la regla #PCDATA:

```
<!ELEMENT elementname (#PCDATA)>
```

Un elemento en un documento XML al que liga la regla #PCDATA podría aparecer como sigue:

```
<data>
This is some parsed character data
</data>
```

Es posible en un elemento que usa la regla #PCDATA para utilizar la palabra clave CDATA para impedir que los datos de caracteres se analicen. Pueden ver un ejemplo de esto en seguida:

```
<sample>
<data>
<![CDATA[<tag>This will not be parsed</tag>]]>
</data>
</sample>
```

Todos los datos entre <! [CDATA [y]] > serán ignorados por el analizador sintáctico y se tratará como caracteres normales (la etiqueta es ignorada).

Reglas de estructura

Considerando que las reglas de contenido tratan el contenido real de los datos incluidos en los elementos definidos, las reglas de estructura tratan cómo esos datos pueden organizarse. Hay dos tipos de reglas de estructura. La primera es la regla “Element Only”. La segunda regla es la regla "mixed".

La regla “Element Only”

La regla “Element Only” especifica que sólo elementos pueden aparecer como hijos del elemento actual. Las secuencias de elemento hijo deben ser separadas por comas y deben listarse en el orden que deben aparecer. Si hay opciones para las que los elementos aparecerán, los elementos listados deben ser separados por el símbolo pipe (|). La definición del elemento siguiente muestra la regla “Element Only”:

```
<!ELEMENT elementname (element1, element2, element3)>
```

Pueden ver aquí que se espera que una lista de elementos aparezca como elementos hijo de elementname cuando el documento XML referenciado es analizado. Todos estos elementos hijo deben ser actuales y en el orden especificado. En seguida se muestra cómo un elemento está listando una serie de opciones de aparición:

```
<!ELEMENT elementname (element1 | element2)>
```

El elemento definido aquí tendrá un solo elemento hijo: element1 o element2.

La regla "mixed"

La regla "mixed" se usa para ayudar a definir los elementos que pueden tener datos de caracteres (#PCDATA) y elementos hijo en los datos que contienen. Una lista de opciones o una lista secuencial serán encerradas por paréntesis. Las opciones serán separadas por el símbolo pipe (|), considerando que las listas secuenciales serán separadas por comas. El elemento siguiente es un ejemplo de la regla "mixed":

```
<!ELEMENT elementname (#PCDATA | childelement1 | childelement2)*>
```

En este ejemplo, el elemento puede contener una mezcla de datos de caracteres y elementos hijo. El símbolo pipe se usa aquí para indicar que hay una opción entre #PCDATA y cada uno de los elementos hijo. Sin embargo, el símbolo asterisco (*) se agrega aquí para indicar que cada uno de los elementos dentro de los paréntesis puede parecer nulo o más veces. Esto puede ser útil para describir conjuntos de datos que tienen los valores opcionales. Considere la definición de elemento siguiente:

```
<!ELEMENT Son (#PCDATA | Name | Age)*>
```

Ahora un fragmento basada en esta definición con la regla “mixed”:

```
<Son>
N/A
</Son>
<Son>
Adopted Son
<Name>Bobby</Name>
<Age>12</Age>
</Son>
```

Esta definición define un elemento, Hijo para que puede haber datos de caracteres, elementos o ambos. Un hombre podría tener un hijo, pero no puede. Si no hay ningún hijo, entonces los datos de caracteres normales (como "N/A") podrían usarse para describir esta condición. Alternativamente, el hombre podría tener un hijo adoptivo y le gustaría indicar esto.

Símbolos de elemento

Además de las reglas normales que se aplican a las definiciones de elemento, los símbolos del elemento pueden usarse para controlar la ocurrencia de datos. En seguida se muestran los símbolos que están disponibles para el uso en DTDs.

- Asterisco (*): Los datos aparecerán cero o más veces (0, 1, 2,...).
- Coma (,): Proporciona separación de elementos en una secuencia.
- Paréntesis [()]: Los paréntesis se usan para contener la regla para un elemento. Los paréntesis también pueden usarse para agrupar una secuencia, subsecuencia o un conjunto de alternativas en una regla.
- Pipe (|): Separa las opciones en un conjunto de opciones.
- Signo mas (+): Significa que los datos deben aparecer uno o más veces (1, 2, 3,...).

- Signo de interrogación (?): Los datos aparecerán cero veces o una vez en el elemento.

- Ningún símbolo: Cuando ningún símbolo se usa (de otra manera que los paréntesis), esto significa que los datos deben aparecer una vez en el archivo de XML.

Pueden agregarse símbolos del elemento a las definiciones de elemento para otro nivel de control sobre los documentos XML que están validándose contra él. Considere los DTD en el listado siguiente que hace uso muy limitado de símbolos de XML:

```
<!ELEMENT contactlist (contact) >
<!ELEMENT contact (name, age, sex, address, city, state, zip,
children) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT age (#PCDATA) >
<!ELEMENT sex (#PCDATA) >
<!ELEMENT address (#PCDATA) >
<!ELEMENT city (#PCDATA) >
<!ELEMENT state (#PCDATA) >
<!ELEMENT zip (#PCDATA) >
<!ELEMENT children (child) >
<!ELEMENT child (childname, childage, childsex) >
<!ELEMENT childname (#PCDATA) >
<!ELEMENT childage (#PCDATA) >
<!ELEMENT childsex (#PCDATA) >
```

El uso de símbolos del elemento, pueden crear un DTD más flexibles que tendrán en cuenta la posibilidad que no siempre podrían saber toda la información de los contactos personales. Ahora considere un DTD similar al anterior pero con más símbolos:

```
<!ELEMENT contactlist (contact+) >
<!ELEMENT contact (name, age?, sex, address?, city?, state?, zip?,
children?) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT age (#PCDATA) >
<!ELEMENT sex (#PCDATA) >
<!ELEMENT address (#PCDATA) >
<!ELEMENT city (#PCDATA) >
<!ELEMENT state (#PCDATA) >
<!ELEMENT zip (#PCDATA) >
<!ELEMENT children (child*) >
<!ELEMENT child (childname, childage?, childsex) >
<!ELEMENT childname (#PCDATA) >
<!ELEMENT childage (#PCDATA) >
<!ELEMENT childsex (#PCDATA) >
```

Este listado es mucho más flexible que el anterior. Hay todavía un solo elemento raíz, contactlist que contendrá uno o más casos (+) del elemento contacto. Bajo cada elemento contacto es una lista de elementos hijo que constituyen la descripción del registro del contacto. Es supuesto aquí que se conocerán el nombre y sexo del contacto. Sin embargo, la definición indica que habrá cero o una ocurrencia (?) de la edad, dirección, ciudad, estado, código postal y elementos de hijos. También vemos que el elemento de los hijos se etiqueta para tener cero o más casos (*) del elemento hijo. Esto es porque una persona podría tener ningún hijo o muchos hijos (o no podríamos saber cuántos hijos la persona tiene). Bajo el elemento hijo, es supuesto que se conocerán (si hay un elemento hijo por lo menos) el nombre y sexo. Sin embargo, el elemento edad es marcado como cero o uno (?), simplemente en caso de que es desconocido cuántos años tiene el hijo.

Sobre el listado anterior con pocos símbolos todavía podría validarse y aceptado por un analizador sintáctico validando aunque no podría tener los datos generales de todo el contacto. Sin embargo, se rechazaría como inválido si no incluye el elemento de los hijos.

Atributos DTD

Hasta ahora hemos visto que es posible usar combinaciones intrincadas de elementos y símbolos para crear las definiciones de elemento complejas. Ahora echemos una mirada a cómo pueden agregarse definiciones de atributo de XML en esta miscelánea.

Los atributos de XML son pares de nombre/valor que se usan como metadatos para describir los elementos de XML. Los atributos de XML son muy similares a los atributos del HTML. En HTML, el src es un atributo de la etiqueta img, como se muestra en el ejemplo siguiente:

```
<img src="images/imagename.gif" width="10" height="20">
```

En este ejemplo, el ancho y altura son también atributos de la etiqueta img. En seguida se muestra el uso del atributo en XML:

```
<image src="images/" width="10" height="20">
imagename.gif
</image>
```

En este listado, se presentan src, ancho y altura como atributos del elemento image de XML. Esto es muy similar a la manera que estos atributos se usan en HTML. La única diferencia es que el atributo src meramente contiene la ruta relativa del directorio de las imágenes y no el nombre real del archivo de la imagen. Además, se usan los atributos como metadatos para describir ciertos

aspectos del contenido del elemento image. Esto es consistente con el uso normal de atributos. Los atributos también pueden usarse para proporcionar la información adicional para una extensión o indexación de un elemento o incluso la información de formateo.

Los atributos también están definidos en DTDs. Se declaran definiciones de atributo usando la declaración ATTLIST. Una declaración ATTLIST definirá uno o más atributos para el elemento que está haciendo referencia.

Las declaraciones de la lista de especificaciones de atributos en un DTD tendrán la sintaxis siguiente:

```
<!ATTLIST elementname attributename type defaultbehavior
defaultvalue>
```

- ATTLIST es el nombre de etiqueta que especifica que esta definición será para una lista de atributos.
- elementname es el nombre del elemento al que el atributo se unirá.
- attributename es el nombre real del atributo.
- type indica cual de los 10 tipos válidos de atributos de esta definición de atributo existirá.
- defaultbehavior dicta si el atributo será requerido, opcional o el valor fijo. Esta configuración determina cómo un analizador sintáctico validando debe relacionar a este atributo.
- defaultvalue es el valor del atributo si ningún valor es explícitamente fijo.

Veamos el listado siguiente, un ejemplo de cómo la declaración puede usarse.

```
<!ATTLIST name
sex CDATA #REQUIRED
age CDATA #IMPLIED
race CDATA #IMPLIED >
```

En el listado, una lista de especificaciones de atributos es declarado. El elemento nombre es al inicio referenciado por la declaración. Tres atributos están definidos; sexo, edad y raza. Los tres atributos son datos de caracteres (CDATA). Uno de los atributos, sexo, es requerido (#REQUIRED). Los otros dos atributos, edad y raza, son opcionales (#IMPLIED). Un elemento XML que usa la lista de especificaciones de los atributos declarados aparecería como sigue:

```
<name sex="male" age="30" race="Caucasian">Michael Qualls</name>
```

El elemento nombre contiene el valor "Michael Qualls." También tiene tres atributos de Michael Qualls: sexo, edad y raza. Los atributos son datos de caracteres (CDATA), pero, actualmente hay 10 posibles tipos datos.

Tipos del atributo

En seguida se describen los 10 tipos válidos de atributos que pueden usarse en un DTD.

- CDATA : Sólo datos de caracteres. Ejemplo,
  ```
  <ATTLIST box height CDATA "0">
  ```

- ENTITY : El nombre de una entidad general sin análisis que se declara en el DTD pero se refiere a algunos datos externos (como un archivo de la imagen). Ejemplo,
  ```
  <!ATTLIST img src ENTITY #REQUIRED>
  ```

- ENTITIES : Esto es igual que el tipo ENTITY pero representa valores múltiples listados en orden secuencial, separado por espacio blanco. Ejemplo,
  ```
  <!ATTLIST imgs srcs ENTITIES #REQUIRED>
  ```

- ID : Un atributo que singularmente identifica el elemento. El valor para este tipo de atributo debe ser único dentro del documento XML. Cada elemento puede tener sólo un solo atributo del identificador y el valor del atributo del identificador debe ser un nombre de XML válido. Ejemplo,
  ```
  <!ATTLIST cog serial ID #REQUIRED>
  ```

- IDREF : Éste es el valor de un atributo ID de otro elemento en el documento. Se usa para establecer una relación con otras etiquetas cuando no hay necesariamente una relación de padre / hijo. Ejemplo,
  ```
  <!ATTLIST person cousin IDREF #IMPLIED>
  ```

- IDREFS : Esto es igual que IDREF; sin embargo, representa valores múltiples listados en orden secuencial, separado por espacio blanco. Ejemplo,
  ```
  <!ATTLIST person cousins IDREFS #IMPLIED>
  ```

- NMTOKEN : Restringe el valor del atributo a un nombre de XML válido. Ejemplo,
  ```
  <!ATTLIST address country NMTOKEN "usa">
  ```

- NMTOKENS : Esto es igual que NMTOKENS; sin embargo, representa valores múltiples listados en orden secuencial, separado por espacio blanco. Ejemplo,
  ```
  <!ATTLIST region states NMTOKENS "KS OK" >
  ```

- NOTATION : Este tipo se refiere al nombre de un notación declarado en el DTD. Se usa para identificar el formato de datos no-XML. Un ejemplo estaría usando el tipo del NOTATION para referirse a una aplicación externa que interactuará con el documento. Ejemplo,
  ```
  <!ATTLIST music play NOTATION "mplayer2.exe ">
  ```

- Enumerado : Este tipo no es una palabra clave. Realmente es una lista de posibles valores para el atributo separado por el símbolo pipe (|). Ejemplo,
  ```
  <!ATTLIST college grad (1|0) "1">
  ```

Hay cuatro tipos de valor por defecto diferentes que pueden usarse en una definición de atributo, que ya hemos utilizado en los ejemplos. En seguida se describen:

- #REQUIRED : Indica que el valor del atributo debe especificarse. Ejemplo,
  ```
  <!ATTLIST season year CDATA #REQUIRED >
  ```

- #IMPLIED : Indica que el valor del atributo es opcional. Ejemplo,
  ```
  <!ATTLIST field size CDATA #IMPLIED >
  ```

- #FIXED : Indica que el atributo es opcional, pero si esta presente, debe tener un valor fijo especificado que no puede cambiarse. Ejemplo,
  ```
  <!ATTLIST bcc hidden #FIXED "true" >
  ```

- Valor predeterminado : Éste no es un tipo de comportamiento predeterminado actual. El valor del valor predeterminado se proporciona en el DTD. Ejemplo,
  ```
  <!ATTLIST children number CDATA "0">
  ```

Hasta ahora tenemos declaraciones de elemento (ELEMENT) y declaraciones de atributo (ATTLIST). Con ello se pueden crear estructuras jerárquicas complejas mediante los elementos y atributos. En seguida trataremos, las entidades de DTD que ofrecen una manera de almacenar pedazos cortos y gruesos repetitivos o grandes de datos para referencia rápida.

Entidades de DTD

Las entidades en DTDs son las unidades del almacenamiento. También pueden ser considerados los marcadores de posición. Las entidades son etiquetados especiales que incluyen el contenido para la inserción en el documento XML. Normalmente éste será algún tipo de información que es voluminosa o repetitiva.

Las entidades hacen de este tipo de información el manejo más fácilmente porque el autor del DTD puede usarlas para indicar donde la información debe insertarse en el documento XML. Esto es mucho mejor que teniendo que reescribir la misma información sobre de y sobre de.

El contenido de una entidad podría bien formarse en XML, texto normal, datos binarios, un registro de base de datos y así sucesivamente. El propósito principal de una entidad es sostener el contenido y no hay virtualmente ningún límite en el tipo de contenido que una entidad puede contener.

La sintaxis general de una entidad es como sigue:

```
<!ENTITY entityname [SYSTEM | PUBLIC] entitycontent>
```

- ENTITY es el nombre de etiqueta que especifica que esta definición será para una entidad.
- entityname es el nombre por el que la entidad será referida en el documento XML.
- entitycontent son los contenidos actuales de los datos de la entidad para los que la entidad está sirviendo como un marcador de ubicación.
- SYSTEM y PUBLIC son las palabras claves opcionales. Cualquiera de los dos puede agregarse a la definición de una entidad para indicar que la entidad se refiere al contenido externo.

Las entidades pueden apuntar a datos internos o a datos externos. Las entidades internas representan datos que se contienen completamente dentro del DTD. Las entidades externas apuntan al contenido en otra ubicación vía un URL.
Una entidad es referenciada en un documento XML insertando el nombre de la entidad prefijado por & y sufijado por ;. Cuando hace referencia de esta manera, el contenido de la entidad se pondrá en el documento XML cuando el documento se analiza y se valida. Veamos un ejemplo de cómo esto trabaja:

Uso de entidades internas

```
<?xml version="1.0"?>
<!DOCTYPE library [
<!ENTITY cpy "Copyright 2000">
<!ELEMENT library (book+)>
<!ELEMENT book (title,author,copyright)>
<!ELEMENT title (#PCDATA)>
```

```
<!ELEMENT author (#PCDATA)>
<!ELEMENT copyright (#PCDATA)>
]>
<library>
<book>
<title>How to Win Friends</title>
<author>Joe Charisma</author>
<copyright>&cpy;</copyright>
</book>
<book>
<title>Make Money Fast</title>
<author>Jimmy QuickBuck</author>
<copyright>&cpy;</copyright>
</book>
</library>
```

En el DTD, una entidad llamada cpy se declara que contiene el contenido "Copyright 2000". En el elemento copyrigth del documento XML, esta entidad es referenciada usando &cpy;. Cuando este documento se analiza, &cpy; se reemplazará con "Copyright 2000" en cada caso en el que se usa. Usar la entidad &cpy; salva al autor del documento XML de tener que teclear "Copyright 2000" una y otra vez. Éste es un ejemplo bastante simple, pero imagine si la entidad contuvo una cadena de datos que eran muchas veces varios cientos de caracteres.

Entidades predefinidas

Hay cinco entidades predefinidas, & < > " ' que al utilizarse se reemplazan los contenidos &, <, >, “, ‘, respectivamente. Estas entidades no tienen que ser declaradas en el DTD. Cuando un analizador sintáctico de XML encuentra estas entidades (a menos que estén contenidas en una sección de CDATA), se reemplazarán automáticamente con el contenido que representan.

El fragmento de XML siguiente, muestra el uso de una entidad predefinida.

```
<icecream>
<flavor>Cherry Garcia</flavor>
<vendor>Ben &amp; Jerry's</vendor>
</icecream>
```

En esta fragmento, el ampersand en "Ben & Jerry” es reemplazado con la entidad predefinida para un ampersand (&) .

Entidades externas

Las entidades externas se usan para hacer referencia al contenido externo. Como declarado previamente, las entidades externas consiguen su contenido haciéndolo la referencia vía un URL ubicada en la porción del entitycontent de la declaración

de entidad. La palabra clave SYSTEM o la palabra clave PUBLIC se usa aquí para permitir al analizador sintáctico de XML saber que el contenido es externo.

XML es prodigiosamente flexible. Las entidades externas pueden contener las referencias a casi cualquier tipo de dato incluso de otros documentos XML. Un documento XML bien formado puede contener otro documento XML bien formado a través del uso de una referencia de la entidad externa. O más lejos, puede extrapolarse fácilmente que un solo documento XML puede componerse de referencias a muchos pequeños documentos XML. Cuando el documento es analizado, el analizador sintáctico de XML recogerá que todos los pequeños documentos XML, fusionándolos en un todo. La aplicación de usuario final verá sólo un documento y nunca sabrá la diferencia. Esto se muestra enseguida, el uso de entidades externas:

```
<?xml version="1.0"?>
<!DOCTYPE employees [
<!ENTITY bob SYSTEM "http://srvr/emps/bob.xml">
<!ENTITY nancy SYSTEM "http://srvr/emps/nancy.xml">
<!ELEMENT employees (clerk)>
<!ELEMENT clerk (#PCDATA)>
]>
<employees>
<clerk>&bob;</clerk>
<clerk>&nancy;</clerk>
</employees>
```

En esta listado, dos referencias de la entidad externa se usan para referirse a los documentos XML fuera del documento actual que contiene los datos del empleado sobre "bob" (bob.xml) y "nancy" (nancy.xml). La palabra clave SYSTEM se usa aquí para permitir al analizador sintáctico de XML saber que éste es el contenido externo. Para insertar el contenido externo en el documento XML, las entidades &bob; y &nancy; son usadas. Es útil para poder contener la información del empleado en un archivo separado e "importarlo" usando una referencia de la entidad. Esto es porque esta misma información pudiera ser referenciada fácilmente por otros documentos XML, como un directorio del empleado y una aplicación de la nómina de sueldos.

Entidades externas no-texto y notaciones

Algunas entidades externas contendrán datos no-texto, como un archivo de imagen. No queremos que el analizador sintáctico de XML intentar analizar estos tipos de archivos. Para detener el analizador sintáctico de XML, se usa la palabra clave de NDATA. Veamos la declaración siguiente:

```
<!ENTITY myimage SYSTEM "myimage.gif" NDATA gif>
```

La palabra clave NDATA se usa para alertar el analizador sintáctico que el contenido de la entidad debe enviarse sin analizar al documento de salida.

La parte final de la declaración, gif, es una referencia a una notación. Una notación es una declaración especial que identifica el formato de no-texto de datos externos para que la aplicación de XML conozca cómo ocuparse de los datos. Notaciones son declaradas en el cuerpo del DTD y tienen la sintaxis siguiente:

```
<!NOTATION notationname [SYSTEM | PUBLIC ] dataformat>
```

- NOTATION es el nombre de etiqueta que especifica que esta definición será para una notación.

- notationname es el nombre por el que el notación será referida en el documento XML.

- SYSTEM es una palabra clave que se agrega a la definición de la notación para indicar que el formato de datos externos está en inicio definido. También podrían usar la palabra clave PUBLIC en lugar de SYSTEM. Sin embargo, usando PUBLIC les exige que proporcionen una URL a la definición de la ubicación de los datos.

- dataformat es una referencia a un tipo MIME, estándar de ISO o alguna otra ubicación que pueden proporcionar una definición de los datos que inicia la referencia.

En seguida un ejemplo del uso de declaraciones de notación para las entidades externas no-texto.

```
<!NOTATION gif SYSTEM "image/gif" >
<!ENTITY employeephoto SYSTEM "images/employees/MichaelQ.gif"
NDATA gif >
<!ELEMENT employee (name, sex, title, years) >
<!ATTLIST employee pic ENTITY #IMPLIED >
...
<employee pic="employeephoto">
...
</employee>
```

En este ejemplo, un tipo de entidad de atributo, pic, está definido para el elemento empleado. En el documento XML, el atributo pic se da el valor employeephoto que es una entidad externa que sirve como un marcador de ubicación para el archivo GIF, MichaelQ.gif. En orden apoya el proceso de la aplicación y despliega el

archivo GIF, la entidad externa (usando la palabra clave NDATA) hace referencia a la notación gif que apunta al tipo MIME para archivos GIF.

Entidades parámetro

El tipo final de entidad que veremos es la entidad parámetro que es muy similar a la entidad interna. La diferencia principal entre una entidad interna y una entidad parámetro es que una entidad parámetro sólo puede ser referida dentro del DTD. Las entidades parámetro son entidades específicamente para DTDs.

Las entidades parámetro pueden ser útiles cuando tienen que usar mucho texto repetitivo o largo en un DTD. Se usa la sintaxis siguiente para las entidades parámetro:

```
<!ENTITY % entityname entitycontent>
```

La sintaxis para una entidad parámetro es casi idéntica a la sintaxis para una entidad interna. Sin embargo, en la sintaxis, después de la declaración, hay un espacio, una señal por ciento y otro espacio antes del entityname. Esto alerta el analizador sintáctico de XML que ésta es una entidad parámetro y sólo se usará en el DTD. Estos tipos de entidades, cuando se hace referencia, deben empezar con % y terminar con;. Enseguida se muestra un ejemplo de esto:

```
<!ENTITY % pc "(#PCDATA)">
<!ELEMENT name %pc;>
<!ELEMENT age %pc;>
<!ELEMENT weight %pc;>
```

En este ejemplo, pc se usa como una entidad parámetro para referenciar (#PCDATA). Todas las entidades en el DTD sostienen el análisis de los datos de caracteres usando la referencia de entidad %pc;.

Otras declaraciones de DTD

- La declaración IGNORE : se usa para indicar bloques de declaraciones que no deben ser incluidas cuando el documento se procesa. Ejemplo,
  ```
  <![ IGNORE
  This is the part of the DTD ignored
  ]]>
  ```

- La declaración INCLUDE : se usa para indicar bloques de declaraciones que deben ser incluidas cuando el documento se procesa. Esta declaración es totalmente innecesaria para un procesamiento exitoso de un DTD. Ejemplo,
  ```
  <![ INCLUDE
  ```

```
This is the part of the DTD included
]]>
```

- Comentarios : pueden ser incluidos en DTDs. Se usan comentarios en DTDs exactamente de la misma manera que se usan en HTML. Ejemplo,
  ```
  <!-- Everything between the opening tag and closing tag is a comment -->
  ```

El DTD es una herramienta poderosa por definir las reglas para los documentos XML a seguir. DTDs han tenido y continuarán teniendo un lugar importante en el mundo de XML durante algún tiempo para manifestarse. Sin embargo, DTDs no son perfectos. Como XML se ha extendido más allá de un lenguaje de marcado de documento simple, estas limitaciones se han puesto más claras. XML está volviéndose el lenguaje de opción rápidamente para describir tipos más abstractos de datos. DTDs está presionando duro para mantenerse.

2.3.3 Actividades de aprendizaje

Ejercicio sobre validación de documento XML mediante DTD

El escenario consiste de un documento XML sobre vendedores. El documento incluye DTD y espacio de nombres.

Instrucciones

1. En base a el Material de Apoyo “2.2 Documentos XML” y Material de Apoyo “2.3 Validación XML: DTD”, utilice el archivo de documento XML vendor.xml que se proporciona en el rubro “Código documento XML validación DTD” del Anexo C; para realizar la tarea siguiente:

Identifique las inconsistencias del documento y modifique para que este sea validado y bien formado.
2. Coloque su trabajo en “Tarea Individual Ejercicio (2b)”

Se recomienda:

- La lectura de ligas de interés del “Anexo E. Fuentes de Información”.
- El uso del foro de discusión “Foro Descripción de Información: XML”

2.4. Esquemas y Consultas XML

Como es sabido, todos los documentos XML deben formarse bien. Por ejemplo, las marcas no pueden mezclarse. Debe especificarse algún orden de jerarquía. Pero a menudo los beneficios de producir un documento bien-formado no serán suficientes. Decir que los elementos XML no pueden mezclarse no es tan útil como decir que los elementos XML no pueden mezclarse y deben seguir un orden

específico o usar ciertos nombres de marcas. La especificación del Esquema XML identifica un vocabulario XML que puede usarse para crear otros vocabularios XML. Con esto, se comunica a consumidores de cómo los esquemas XML deben construirse para que sean válidos en su diseño.

Además de conocer que los documentos XML se forman bien y son validados, también es conveniente conocer tecnologías que nos permitan consultar y manejar valores de elementos detrás de los documentos XML, esto a través de XPath y con XLink apuntar o identificar elementos XML que no necesariamente tienen un comportamiento jerárquico, esto conduce a la serialización en servicios Web.

2.4.1. Comprensión de esquemas XML

Muchas veces cuando desarrolla un documento XML, necesita a menudo poner restricciones sobre la forma de representar los datos en el documento. Podría implicar que un conjunto particular de elementos XML sigue un orden específico o podría querer identificar un elemento XML como contener texto que actualmente representa un tipo de datos específico, como un punto flotante.

Para poner las restricciones en datos, debe construir Definiciones de Tipo Documento (DTDs) o Esquemas XML para proporcionar datos alrededor de los datos, también conocido como metadatos. Los DTDs son un mecanismo temprano de restricción de XML. Aunque DTDs son beneficiosos a muchas aplicaciones de XML, no tienen las características necesarias para describir estructuras como herencia o tipos de datos complejos. Para superar estas limitaciones, un grupo activo fue formado para producir Esquemas XML basados en un tratado original de Microsoft. La especificación de Esquema XML es dividida en dos partes.

1. Estructuras, propone a una forma de estructurar y restringir el contenido del documento.

2. Tipos de datos, proporciona una forma de describir tipos de datos, tanto primitiva como compleja, dentro de un documento.

La especificación de Esquema XML establece medios para los que el lenguaje de Esquema XML describe la estructura y contenido de documentos XML. Una característica deseable de Esquemas XML es el hecho que son representados en XML los analizadores sintácticos de XML estándares para que puedan usarse como navegables.

A estas alturas, ya está familiarizado con XML y muchos de los términos usados para identificar los conceptos detrás de XML. El tratado de Esquema XML define

varias nuevos términos que apoyan a describir la semántica de uso y la comprensión de esquemas.

Instancias y esquema

Una instancia de documento XML se refiere al elemento documento, incluso los elementos, los atributos y contenido dentro del documento que conforma a un Esquema XML. Las instancias en el sentido más general puede referirse a cualquier elemento (incluyendo sus atributos y contenido) que conforma a un Esquema XML. Se considera que una instancia que conforma a un esquema es esquema válido.

Los esquemas pueden ser los documentos XML independientes o ellos pueden acoplarse dentro de otro XML con referencias al esquema. Los esquemas toman esta forma:

```
<xsd:schema xmlns:xsd="http://www.w3.org/TR/xmlschema-1/">
   <!--type definitions, element declarations, etc. -->
</xsd:schema>
```

El fragmento muestra una declaración del espacio de nombres para el URI http://www.w3.org/TR/xmlschema-1/, este espacio de nombres se mapea tradicionalmente al prefijo xs o xsd como el estandar. El elemento Documento de un Esquema XML (XSD, XML Schema Document) es el xsd:schema.

Definiciones y declaraciones

Una gran ventaja de esquemas es que permiten crear tipos simples o complejos para aplicación de clasificaciones de elementos. Como en la mayoría de los lenguajes de programación, esto se llama la definición del tipo y se muestra como sigue:

```
<xsd:schema xmlns:xsd="http://www.w3.org/TR/xmlschema-1/">
   <xsd:complexType name="Person">
      <xsd:element name="FirstName" type="xsd:string" />
      <xsd:attribute name="Age" type="xsd:integer" />
   </xsd:complexType>
</xsd:schema>
```

El ejemplo precedente también muestra que una declaración de elemento (para el elemento <FirstName/>) y una declaración del atributo (para el atributo Age) que son locales a un tipo particular nombrado Person.

Más allá de la definición del tipo, pueden declararse también elementos como elementos a un nivel cima de un tipo particular, como es mostrado en el ejemplo siguiente dónde BaseballPlayer es un tipo de Person:

```
<xsd:schema xmlns:xsd="http://www.w3.org/TR/xmlschema-1/">
   <!-- ...Type definition... -->

   <xsd:element name="BaseballPlayer" type="Person" />
</schema>
```

Los atributos, sin embargo, pueden ser de sólo tipos simples, definidos como string, boolean o float.

Espacio de nombres destino

Debido a que las declaraciones de elemento y atributo se usan para validar las instancias, es necesario para ellos para emparejar el característico espacio de nombres de una instancia en particular. Esto implica que las declaraciones tienen una asociación en absoluto con un espacio de nombres destino URI o ningún espacio de nombres, dependiendo adelante si la instancia tiene un nombre calificado. Para especificar un espacio de nombres destino, debe usar para un esquema el atributo targetNamespace, como sigue:

```
<xsd:schema xmlns:xsd="http://www.w3.org/TR/xmlschema-1/"
 targetNamespace="SomeNamespaceURI">
   <xsd:element name="ElementInNS" type="xsd:string" />
   <xsd:complexType name="TypeInNS">
      <xsd:element name="LocalElementInNS" type="xsd:integer" />
      <xsd:attribute name="LocalAttrInNS" type="xsd:string" />
   </xsd:complexType>
</xsd:schema>
```

Como puede ver, todos los elementos globales y locales son asociados con SomeNamespaceURI.

Tipos de datos y restricciones del esquema

Los tipos de datos consisten de un espacio de valor, espacio léxico y facetas. El espacio de valor es el conjunto permitido de tipo de datos de valores y puede tener varias propiedades asociadas con él. Un conjunto de literales válido para un tipo de datos constituye el espacio léxico de ese tipo de datos. Finalmente, una faceta es una sola dimensión de un concepto que le permite que distinga entre los tipos de datos diferentes. Dos tipos de facetas se usan para describir los tipos de datos, fundamental y condicional.

Las facetas fundamentales le permiten que describa el orden, límites, cardinalidad, exactitud y las propiedades numéricas de un espacio de valor de tipo de datos dado.

Las facetas condicionales le permiten que describa las restricciones en un espacio de valor de tipo de datos. Las posibles restricciones incluyen el mínimo y longitud máxima, correspondencia de patrones, límites superiores e inferiores y enumeración de valores válidos.

En seguida es el fragmento de una definición de tipo simple:

```
<xsd:simpleType name="HourType">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="1" />
      <xsd:maxInclusive value="12" />
   </xsd:restriction>
</datatype>
```

En este caso, HourType se define para ser del tipo de datos entero predefinido y adicionalmente se restringe a los valores entre 1 y 12. Este nuevo tipo puede usarse entonces en otras definiciones del tipo como en el atributo Hour siguiente:

```
<xsd:complexType name="Time">
   <xsd:attribute name="Hour" type="HourType" />
   <xsd:attribute name="Minute" type="MinuteType" />
</xsd:complexType>
```

La instancia para este tipo podría ser así:

```
<Time Hour="7" Minute="30" />
```

Este ejemplo es también un ejemplo de una definición de tipo compleja. La definición de tipo compleja combina uno o los tipos más simples para formar algo nuevo. En seguida otro ejemplo de tipo complejo:

```
<xsd:element name="cars" type="CarsType"/>

<xsd:complexType name="CarsType">
   <xsd:element name="car" type="CarType"
     minoccurs="0" maxoccurs="unbounded"/>
</xsd:complexType>

<xsd:complexType name="CarType">
   <xsd:element name="make" type="xsd:string"/>
   <xsd:element name="model" type="xsd:string"/>
</xsd:complexType>
```

Este tipo puede ser representado por una instancia como sigue:

```
<cars xmlns:xsi="http://www.w3.org/TR/xmlschema-1/"
  xsi:noNamespaceSchemaLocation="CarSchema.xsd">
   <car>
      <make>Cheverolet</make>
      <model>Corvette</model>
   </car>
</cars>
```

minOccurs y maxOccurs

Los elementos y atributos le permiten que especifique el número mínimo y máximo de veces que pueden aparecer en la instancia. El ejemplo siguiente muestra cómo puede obligar a un atributo aparecer una y sóla una vez:

```
<xsd:element name="Book">
    <attribute name="Author" type="A" minOccurs="1" maxOccurs="1" />
    <attribute name="Title" type="T" minOccurs="1" maxOccurs="1" />
</xsd:element>
```

El atributo maxOccurs también puede establecerse a ilimitado, denotando que el elemento o atributo pueden aparecer muchas veces. También puede impedir a un valor aparecer, estableciendo el atributo maxOccurs igual a 0.

Derivación de definiciones del tipo

Similar a los lenguajes de programación orientados a objetos de la manera que trabajan, los esquemas le permiten que derive los tipos de otros tipos de una manera controlada. Al definir un nuevo tipo, puede decidir extender o restringir la otra definición del tipo.

Al extender otra definición del tipo, puede introducir elementos adicionales y atributos, como es mostrado en el ejemplo siguiente:

```
<xsd:complexType name="Book">
   <xsd:element name="Title" type="xsd:string" />
   <xsd:element name="Author" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="ElectronicBook">
   <xsd:complexContent>
      <xsd:extension base="Book">
          <xsd:sequence>
             <element name="URL" type="xsd:string" />
          </xsd:sequence>
      </xsd:extension>
```

```
   </xsd:complexContent>
</xsd:complexType>
```

A veces una instancia requiere indicar su tipo explícitamente. Para hacer esto, la instancia puede usar la definición de la instancia del espacio de nombres del Esquema XML, xsi:type, como sigue:

```
<Car xsi:type="SportsCar">
    <Driver>Me</Driver>
</Car>
```

Resumiendo esquemas, se presentan los siguientes puntos relevantes:

- Puede delimitar sus documentos XML y sus contenidos usando Esquemas XML.
- La especificación de Esquema XML le proporciona un conjunto de tipos de datos predefinidos.
- Puede usar los tipos de datos predefinidos o puede crear sus propios tipos de datos.
- Los esquemas pueden ser documentos autónomos o pueden combinarse dentro de otros documentos XML.

Estos puntos relevantes son de suma importancia para su posterior uso con servicios Web XML.

2.4.2. Tecnologías XPath y XLink

Es conveniente insistir en la importancia de XML y los esquemas dentro de las tecnologías subyacentes de los servicios Web. Sobre esto daremos un vistazo y luego resaltar otros aspectos de XML, como las tecnologías XPath y XLink.

La especificación de SOAP con XML, además de permitir codificar parámetros de métodos, se pueden validar los paquetes de SOAP entrantes, extraer los datos del parámetro del método y la invocación del mismo. Pero más relevante, mediante WSDL se cuenta con un documento de descripción de interfaz que puede usar para determinar qué información de XML el servicio Web aceptará. En otros términos, puede cambiar el XML que da formato para su servicio Web cambiando la forma como describa el Servicio Web en su archivo de WSDL. Ello es posible, debido a que hay un esquema XML alojado dentro del archivo de WSDL.

Esto tiene mucho sentido realmente. Sea el caso que proporcione un documento XML arbitrariamente a alguien y espera que esa persona deduzca por inspección sólo lo que está pidiéndole que haga es una tarea muy compleja, si esa persona incluso tiene bastante información para tomar una decisión informada. Por otro lado, si maneja a la misma persona un documento que perfila los tipos de datos de sus parámetros del método y el orden en los que pueden encontrarse, le ha dado bastante información a esa persona para descifrar los documentos de instancia XML que piensa transmitir.

Por esta razón, encontrará un esquema XML alojado dentro del documento de WSDL que describe su servicio Web. Esencialmente, con su documento de WSDL, está diciendo al otro lado qué tipos de datos espera y cómo los quiere ordenados y cómo deben aparecer dentro del paquete de SOAP. Servicios Web son ahora significativamente más flexibles.

Ahora que sabe cómo se forman documentos XML y cómo se validan, ¿cómo consigue los valores asociados con los elementos XML detrás del documento XML?,éste es el trabajo de XPath.

Discutiremos XPath en algún detalle, principalmente porque muchas personas que han trabajado hasta cierto punto todavía con XML podrían requerir un poco de pincelada XPath. Y es que XPath hace su trabajo más fácil si necesita meter la mano en un paquete de SOAP y modificar lo que encuentra, como podría hacerlo con un SoapExtension de .NET. Esto por razones de interoperabilidad o dependencia en sus requisitos individuales de sistema. Por ejemplo, podría querer recuperar un valor de parámetro de SOAP y encriptarlo.

XPath

Considere este documento XML, ficticio, creado para ocurrirse cómo tener acceso a un par de sus servicios Web favoritos:

```
<?xml version="1.0"?>
<Servers>
   <Server name="Gumby">
      <WebService wsdl="?wsdl">
         <Family>Calculators</Family>
         <EndpointURL>http://www.myurl.com/calc</EndpointURL>
      </WebService>
   </Server>
   <Server name="Pokey">
      <WebService wsdl=".wsdl">
         <Family>Time</Family>
         <EndpointURL>http://www.myurl.com/time</EndpointURL>
      </WebService>
   </Server>
</Servers>
```

En este documento XML totalmente ficticio se describen dos servidores de servicios Web. El primero proporciona alguna clase de servicio de la calculadora, basado en la familia del servicio y el segundo proporciona el Time. El servicio de la calculadora envía su WSDL agregando ?WSDL al punto terminal URL, que es cómo trabaja .NET. El segundo hace lo mismo concatenando .WSDL, que es cómo trabaja SOAP Toolkit. Se muestran sólo dos servidores en este caso. Podría tener centenares o más, para que el documento XML correspondiente pudiera crecer más grande. ¿Cómo encontrará la información de un servidor particular que sea de interés para usted?

Ahora digamos que quiere recuperar todos los servidores que son parte de la familia de Time. Podría usar la cadena de consulta XPath:

```
/Servers/Server/WebService[./Family="Time"]
```

El resultado de esta consulta es un nodeset que es un conjunto de elementos XML que emparejan la consulta. La propia consulta puede llamarse “camino de la ubicación” que puede dividirse en tres partes:

- El eje
- La prueba del nodo
- El predicado

Veamos cada uno en más detalle.

El eje XPath

De hecho el eje es opcional, es el camino de la ubicación particular que no tiene eje identificado. Usa el eje para moverse a través del documento XML de alguna otra manera que de arriba abajo. Esto es porque el camino de la ubicación por defecto es el hijo::, para que regrese las consultas XPath, por defecto, un nodeset que contiene a los hijos del nodo del contexto actual. El nodo del contexto es el elemento XML actual que sucede para ser examinando, qué, en este caso, es la raíz o elemento de documento. Otros posibles ejes incluyen el ascendiente::, padre::, siguiendo:: y un sinnúmero de otros posibles valores, en seguida se describen.

- Ascendiente: Ascendencia del nodo del contexto, incluso el nodo de la raíz si el nodo del contexto ya no es el nodo de la raíz.

- Ascendiente propio: Mismo que ascendiente, pero incluye el nodo del contexto.

- Atributo: Atributos del nodo del contexto (el nodo del contexto debe ser un elemento).

- Hijo: Todos (inmediatos) los hijos del nodo del contexto.
- Descendiente: Todos los hijos del nodo del contexto, sin tener en cuenta la profundidad.
- Descendiente propio: Mismo que el descendiente, pero incluye el nodo del contexto.
- Siguiente: Todos los nodos que siguen el nodo del contexto, en orden del documento.
- Siguiente hermano: Sólo nodos del hermano que siguen el nodo del contexto, en orden del documento.
- Espacio de nombres: Espacio de nombres del nodo del contexto (el nodo del contexto debe ser un elemento).
- Padre: Padre inmediato del nodo del contexto, si cualquiera (es decir, no el nodo de la raíz).
- Precedente: Similar a siguiente, excepto que regresa los nodos precedentes en orden del documento.
- Precedente hermano: Mismo que precedente, pero sólo para los nodos del hermano.
- Propio: El propio nodo del contexto.

El orden del documento no se refiere a la clasificación y jerarquía de elementos XML, sino en cambio al orden literal en el que el elemento se encuentra en el documento, si es un padre, hermano, o cualquier cosa. Esencialmente, cuando tiene acceso a los nodos en orden del documento, está aplanando cualquier jerarquía que podría estar presente.

La prueba del nodo XPath

La prueba del nodo es eficazmente un mapa del camino que muestra los nombres del elemento en progreso, del inicio del documento al elemento particular en cuestión. Es literalmente una ruta del elemento de documento (el elemento de XML de raíz) a los datos que está probando para la inclusión en el resultado de nodeset. XPath, como actualmente es implementado, es a menudo más eficaz si especifica la ruta del elemento completa. Sin embargo, podría escribir el camino de ubicación como este ejemplo:

```
//WebService[./Family="Time"]
```

El símbolo de división doble, / /, dice al procesador de XPath iniciar en el elemento de documento, busca de forma recursiva el elemento <WebService/> y después de encontrarlo, ejecuta el predicado.

El predicado XPath

El predicado, a veces llamado filtro, es una prueba Booleana que aplica para tomar una última decisión sobre el elemento XML particular que XPath está examinando. Si el predicado devuelve un resultado verdadero, el elemento XML se agrega al resultado de nodeset. Si no, el elemento está descartado del nodeset. Básicamente, está afinando un filtro de elemento de XML. Por ejemplo, dado los dos servidores mostrados en el documento XML de ejemplo, el nodeset inicial devuelto del eje llega a ambos servidores, que hace el resultado de la prueba del nodo. Es decir, ambos elementos <Server/> tienen los elementos hijos <WebService/>. Es el predicado que los distingue, en este caso, porque sólo al segundo servidor, Pokey, le expone el servicio Web Family de Time.

La prueba del nodo, siendo un camino en el documento XML, es sensible al caso de alfabeto y el espacio de nombres del elemento XML particular mostrados en la ruta. Es decir, suponga que ensaya el camino de ubicación de la forma siguiente:

```
/servers/server/webservice[./family="Time"]
```

El nodeset resultante estaría vacío, considerando que antes de que contuviera el elemento para el servidor Pokey. Nótese que en la consulta de XPath que todo el texto es letra minúscula que es por lo qué fallaría.

Operadores de XPath

XPath propiamente es un lenguaje. Como cualquier lenguaje de programación, XPath tiene un conjunto de operadores. Los operadores representan que capacidades intrínsecas que XPath puede realizar, en seguida se describen:

Operadores intrínsecos de XPath

- ***/*** : Operador de hijo que selecciona los nodos del hijo o especifica el nodo de la raíz.
- ***//*** : Descenso recursivo que mira para un elemento especificado a cualquier profundidad.

- **.** : Nodo del contexto actual (semejante a “this” de C# o “me” de VB)
- **..** : Notación para el padre de nodo del contexto actual (semejante a moverse a un directorio de archivos)
- ***** : Comodín que selecciona todos los elementos sin tener en cuenta su nombre del elemento.
- **:** : Operador del espacio de nombres (mismo uso como en XML)
- @ : Operador de atributo que prefija un nombre del atributo
- @ : Comodín de atributo (cuando es usado exclusivamente), es semánticamente equivalente *.
- **+** : Indicador de suma.
- **-** : Indicador de substracción.
- ***** : Indicador de multiplicación.
- **div** : Indicador de división de punto flotante.
- **mod** : Módulo (resto de una operación de la división truncanda).
- **( )** : Operador de precedencia.
- **[ ]** : Operador que aplica un filtro (semejante a una prueba Booleana)
- **[ ]** : Operador del subíndice fijo (semejante a una especificación del índice de un arreglo).

Regresando al ejemplo anterior, podría localizar todos los servicios Web del SOAP toolkit almacenados en el documento XML que usa este camino de ubicación XPath:

```
/Servers/Server/WebService[./@wsdl=".wsdl"]
```

Podría lograr la misma tarea con estos dos caminos de la ubicación:

```
//WebService[./@wsdl=".wsdl"]
```

```
/Servers/*/*[./@wsdl=".wsdl"]
```

Podría probar para la presencia del atributo wsdl, como:

```
/Servers/Server/WebService[./@wsdl]
```

En este caso, los nodeset contendrían ambos servidores mostrados en el documento XML de ejemplo, pero esto sólo es porque ambos servidores tienen un atributo del wsdl. Si un servidor no tuviera ningún atributo wsdl, el predicado fallaría para ese nodo particular y ese elemento XML se borraría del nodeset resultante.

Funciones intrínseca de XPath

Además de operadores, XPath tiene un conjunto completo de funciones intrínsecas que expone para ayudar con sus consultas. Hay muchas de ellas, las más representativas o de uso común son las siguientes:

Funciones intrínsecas principales de XPath

ceiling () : Es el entero más pequeño no menor que el argumento.

count (nodeset) : Da el número de nodos en el argumento del nodeset.

contains (string, string) : Regresa verdadero si el primer argumento contiene el segundo.

false () : Siempre regresa un Booleano falso.

floor () : Es el entero más grande no mayor que el argumento.

last () : Da el tamaño del contexto (el número de nodos).

local-name () : Devuelve el nombre local del primer nodo (orden del documento).

name () : Devuelve el QName del primer nodo (orden del documento).

node () : Regresa verdadero para cualquier tipo de nodo.

not () : Indica una negación lógica.

number (object) : Convierte el objeto a un número.

position () : Da el número del índice del nodo dentro del padre.

starts-with (string, string) : Regresa verdadero si el primer argumento inicia con el segundo.

string (object) : Convierte el objeto en una cadena.

sum (nodeset) : Convierte el nodeset a los valores numéricos y los suma.

true () : Siempre regresa un Booleano verdadero.

Usar el ejemplo de servidor de Servicio de Web, podría identificar el primer servidor o un servidor arbitrario, usando este camino de la ubicación:

```
/Servers/Server[position()="1"]
```

Semejantemente, encuentra el último servidor:

```
/Servers/Server[last()]
```

Si quiere todos los servidores pero el último uno, pregunta el documento de esta manera:

```
/Servers/Server[not(position()=last())]
```

Muchas personas quieren localizar la información del XML dentro de un documento basado en valores de la cadena o búsquedas de la cadena. Por ejemplo, todos los servidores que tienen el arranque de los nombres con la letra G:

```
/Servers/Server[starts-with(string(./@name),"G")]
```

Claro, esto devolverá la información del XML del servidor Gumby.

Como puede ver, puede producir una variedad amplia de consultas, sobre todo si combina un eje con una prueba del nodo y un filtro. Esto se pone importante después si necesita seccionarse abriendo un paquete de SOAP para examinar y modificar los contenidos a mano.

SOAP usa otra tecnología del XML llamada XLink. Veamos que ofrece esa tecnología y para servicios Web.

Identificación de elementos XML usando XLink

Cuando gana experiencia con XML, encontrarán relaciones jerárquicas entre los elementos de datos que facilita ello a la serialización del XML. Aunque XML está

bastante bien con relaciones padre/hijo o hermano/hermano entre elementos de datos. ¿Qué hace si los datos no son jerárquicos por naturaleza?

Un ejemplo clásico de esto es la lista ligada. Los nodos en una lista ligada, por definición, no tienen ninguna relación jerárquica. Podría ser a favor de una relación de hermano/hermano, pero en realidad XML no especifica clasificación de elementos. Para XML, es un arreglo de documento:

```
<element1/>
<element2/>
```

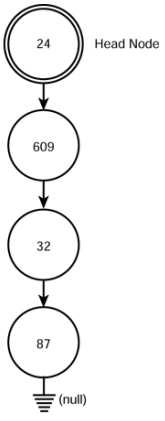
Es semánticamente igual que este arreglo:

```
<element2/>
<element1 />
```

XML no distingue entre los dos a menos que considere su documento en orden y esté orden es considerado tenue. Porque estamos hablando de una lista ligada que tiene una clasificación definida en memoria que necesita tener un bit más en concreto. En este caso, XLink es empleado.

XLink usa dos atributos, href e id, para identificar los enlaces semánticos entre los elementos. En realidad, XLink es un tratado mayor. Sólo discutiremos XLink con respecto a su uso en el protocolo de SOAP. Podría usar XLink para vincular los elementos en documentos XML completamente diferentes.

Veamos XLink en ejemplo. Suponga que tiene la lista ligada siguiente:



Es importante notar que tiene cuatro nodos, con los componentes usuales cabeza nodo, nodo de la cola y nodos intermedios. El nodo cola tiene su "siguiente" enlace a nulo o nada, indica que no hay más datos.

En código, esto lista ligada es realizada con nodos:

```
class Node
{
   int iValue;
   Node pNext;
}
```

En XML, creamos elementos XML que representan los nodos de la lista y los vinculamos para que el nodo actual "apunte" al próximo nodo usando el atributo href:

```
<node head="true">
   <iValue>24</iValue>
   <pNext href="#node2" />
</node>
<node root="true" id="node2">
   <iValue>609</iValue>
   <pNext href="#node3" />
</node>
<node root="true" id="node3">
   <iValue>32</iValue>
   <pNext href="#node4" />
</node>
<node id="node4">
   <iValue>87</iValue>
   <pNext xsi:null="true" />
</node>
```

Cada "siguiente" nodo correspondiente tiene un atributo id que es idéntico al href del nodo previo. Note que el atributo href precede el valor actual id con un símbolo #. Esto indica que el enlace se contiene dentro de este documento en lugar de alguna referencia externa.

Un atributo head también fue creada para este ejemplo. Esto identifica el nodo como el apuntador de inicio sin que necesariamente se conozca cuál nodo fue el punto de partida. Es verdad que pudieran parecer todos los elementos <pNext/> y deducidos, pero agregando el atributo constituye más rápidamente proceso. SOAP tiene un concepto similar, que sera tratado en el tema 3.

Para ligar la lista, agregamos el atributo xsi:null. Esto llega directamente del esquema XML dónde puede identificar un valor como nulo o nada que usa esta estructura. En curso, debe ligar la lista que es requerida como parte de la lista ligada básica del tipo de datos abstracto. Hacer su código por otra parte es atravesar la memoria al azar hasta que se vulnere. El mismo requisito aplica a la serialización de la lista del XML, del atributo.

Como la representación de lista de XML indica, XLink es una tecnología necesaria para serializar servicios Web. Después de todo, si repasa en eso, los métodos que envía en el método del servicio Web remoto son aproximadamente equivalentes a una lista ligada. No son vinculados en el sentido tradicional, pero tienen alguna asociación, si sólo porque pertenecen al mismo método remoto. SOAP mira sus parámetros del método y, en algunos casos, los serializa usando XLink.

El .NET tiene apoyo especial para XLink, pero requiere un bit adicional de encabezado.

XPathNavigator tiene un método, MoveToId() que se diseña para trabajar con atributos id. El encabezado adicional mencionado es que el atributo id debe definirse explícitamente en el XML DTD o esquema que son asociado con su documento XML. Si no tiene DTD o esquema, MoveToId () no trabajará. Sin embargo, todavía puede usar XPath para buscar todos los elementos con un atributo id o incluso un atributo id que contiene un valor de identificación específico.

2.4.3. Actividades de aprendizaje

Ejercicios sobre creación y uso de esquemas

Instrucciones
1. Revisar y estudiar la presentación “El rol de los mensajes XML y esquemas XSD” que se proporciona en el anexo B.

- Se recomienda la lectura del Material de Apoyo “2.4 Esquemas y Consultas XML”

2. En base a la presentación previa, implementar los ejercicios siguientes sobre creación y uso de esquemas:

- Para los archivos que se hace referencia, se proporcionan en el rubro “Código creación y uso de esquemas” del Anexo C.
- Ejercicio A. Generar un esquema desde un documento XML mediante la herramienta xsd.exe. El documento XML contiene información sobre “Purchase Orders” y utiliza un DTD para validación.
    - Desde el command Prompt de VS .NET, generar el archivo XSD del archivo XML po.xml que usa el archivo de validación po.dtd, mediante el comando siguiente:
      > xsd po.xml
    - Abrir el archivo resultante po.xsd con el notepad y explicar brevemente para este ejemplo el uso de tipos, compositores y restricciones.

- Ejercicio B. Generar un esquema XML para clases en una biblioteca DLL mediante xsd.exe. El archivo fuente es product.cs, “tipo de producto”.
    - Compilar el archivo product.cs en una biblioteca product.dll
    - Ejecutar el comando: > xsd product.dll
    - Revisar el archivo product.xsd resultante. ¿Cómo fué implementada la restricción para contar con varios elementos “Street”?
- Ejercicio C. Crear una instancia XSD sobre información de clientes (customer.xsd) mediante programación y utilizando el XML Designer de VS .NET.
    - Compilar el programa en visual C#, CreateSchema.cs, para obtener el archivo customer.xsd.
    - En base al archivo customer.xsd, desde el XML Designer de VS .NET, crear el mismo esquema con el Schema Designer.

3. Coloque el resultado de los ejercicios A, B y C en “Tarea Individual Ejercicio (2c)”

4. Contestar el cuestionario: “Preguntas esquemas y consultas XML”
- Coloque sus respuestas en “Tarea Individual Cuestionario (2a)”.

Se recomienda:
- La lectura de ligas de interés del “Anexo E. Fuentes de Información”.
- El uso del foro de discusión “Foro Descripción de Información: XML”

2.5. Análisis y Acceso XML

Los procesadores (amalizadores sintácticos) del XML son componentes de software para que un documento XML, exponga una interfaz programable que le permite que trabaje con el documento XML de una forma automatizada. Típicamente los procesadores del XML exponen el modelo de objeto de documento (DOM) o quizás el API simple para XML (SAX). El DOM le permite que trate con el documento XML como si estuviera compuesto de nodos de árbol. SAX, por otro lado, ofrece una aproximación basada en flujo para tener acceso a la información del XML, toma lo que necesita del flujo y desecha el resto. Debido a que este curso se centra a servicios Web, no entraremos en detalle sobre DOM o SAX, pero si aspectos esenciales de ellos para el acceso y manejo de datos XML.

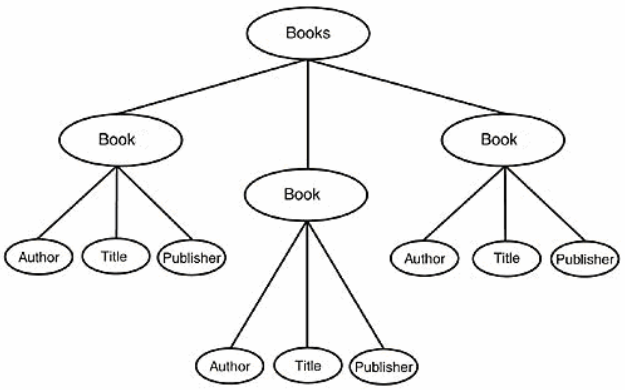
2.5.1. DOM y el núcleo XML

El Document Object Model (DOM) es un estándar de Internet por representar la información contenida en un HTML o documento XML como un árbol de nodos. Como muchos otros estándares de Internet, el DOM es un estándar oficial del W3C, ver en http://www.w3.org/DOM.

En seguida se muestra un documento XML que representa tres libros de una librería de computación con elementos y atributos:

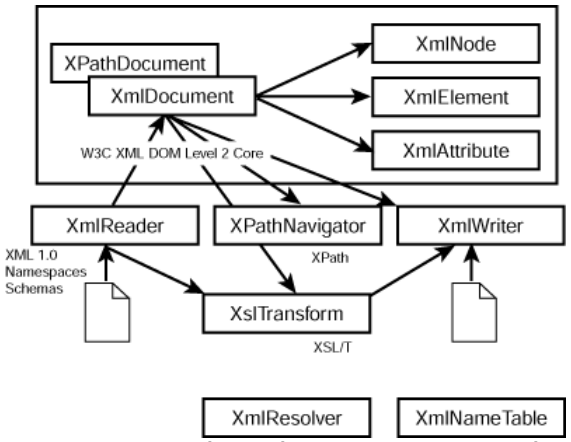
```
<?xml version="1.0" encoding="UTF-8"?>
<Books>
  <Book Pages="1088">
    <Author>Delaney, Kalen</Author>
    <Title>Inside Microsoft SQL Server 2000</Title>
    <Publisher>Microsoft Press</Publisher>
  </Book>
  <Book Pages="997">
    <Author>Burton, Kevin</Author>
    <Title>.NET Common Language Runtime</Title>
    <Publisher>Sams</Publisher>
  </Book>
  <Book Pages="392">
    <Author>Cooper, James W.</Author>
    <Title>C# Design Patterns</Title>
    <Publisher>Addison Wesley</Publisher>
  </Book>
</Books>
```

Estructuralmente, un documento XML es una serie de elementos anidados, incluso los elementos y atributos. Cualquier estructura anidada puede transformarse a una estructura del árbol equivalente si el elemento anidado extremo es hecho la raíz del árbol, el siguiente en elementos son los hijos de la raíz, y así sucesivamente. El DOM provee el estándar para construir este árbol, incluso una clasificación para los nodos individuales y reglas para las que los nodos puedan tener los hijos. En seguida se ilustra la representación como un árbol XML del documento XML de los tres libros de una librería de computación.



En el DOM, no se representan atributos como nodos dentro del árbol. Más bien, se considera que los atributos son propiedades de sus elementos padre.

Aunque hay un estándar de DOM, no todos los vendedores implementan el DOM exactamente de la misma manera. El problema mayor es que hay actualmente varios estándares diferentes que están agrupados bajo el nombre general de DOM. También, vendedores escogen y seleccionan cuáles partes de estos estándares para implementaciones. La infraestructura .NET incluye apoyo para el Nivel 1 de DOM el núcleo y Nivel 2 de DOM las especificaciones del núcleo, pero también extiende el DOM agregando objetos adicionales, métodos y propiedades a la especificación.La figura siguiente muestra la infraestructura de .NET para la arquitectura núcleo de XML.



La principal clase de XML es XPathNavigator. Con esta clase, puede moverse a través del documento XML, usando los comandos de movimiento como MoveToNext() y MoveToFirstChild(). De otro modo, como el nombre sugiere, puede proporcionar el XPathNavigator una consulta de XPath y le devolverá una iteración que representa el nodeset devuelto de la consulta. El Nivel 2 DOM para XML es soportado mediante XmlDocument. Pero para tratar principalmente consultas, XPathDocument está perfeccionado para búsquedas rápidas. Aunque es útil saber que XmlDocument esta dado cuando se trabaja con XML DOM, con XML del .NET se amplia el manejo con su conjunto de clases.

2.5.2. Acceso y manejo de datos XML

Lectura de datos XML

Si tiene un archivo XML en disco, XPathNavigator aceptará el nombre del archivo, ruta y cargará el archivo para ser procesado. Con la clase XmlReader será leído y gestionado el movimiento a través de flujo en el archivo XML.

Para ver XmlReader en acción, simulemos el archivo leído que XPathNavigator realiza cuando carga los datos del XML. Suponga que tiene un archivo XML llamado Activities.xml. Para cargar este archivo en .NET y para mostrar XmlReader, usaría este código:

```
FileStream fstmActivities =
                  new FileStream("Activities.xml",FileMode.Open);
XmlTextReader xrdActivities = new XmlTextReader(fstmActivities);
```

Note que XmlReader es una clase abstracta; no lo usa directamente. En cambio, usa una clase derivada de XmlReader, como XmlTextReader.

Aunque pudiera cargar el archivo directamente en XmlTextReader, el constructor basado en flujo es usado en este ejemplo breve para que vea la forma de cómo con .NET, la información de SOAP será aceptada cuando se procesa un SoapExtension.

XmlReader le da un seguro flujo de datos de XML sólo hacia adelante. XmlTextReader es ideal para determinar rápidamente si su documento XML esta bien formado y permitiéndole progresar a través del documento. En seguida trataremos la escritura de datos XML y conjuntamente en un ejemplo con la acción de XmlTextReader.

Escritura de datos XML

Si hay un XmlReader .NET, debe haber escritura equivalente, el nombre de la clase abstracta es XmlWriter y del mismo modo una clase derivada que es XmlTextWriter.

XmlTextWriter también acepta un constructor de flujo y es para este flujo que XmlTextWriter escribirá el XML que se crea o por otra parte se pone en curso para la salida. El listado siguiente muestra XmlTextReader y XmlTextWriter en acción.

Listado de lectura y escritura de datos XML con clases .NET:

```
using System;
using System.IO;
using System.Text;
using System.Xml;

namespace RdrWtr
{
   /// <summary>
   /// Summary description for Class1.
   /// </summary>
   class Class1
   {
      static void Main(string[] args)
      {
         try
         {
            // Create a file stream
            FileStream fstmXmlOut = new FileStream("MyXML.xml",
                                                   FileMode.Create);
```

```
// Create an encoding
UTF8Encoding objEncoding = new UTF8Encoding();

// Create an XML text writer
XmlTextWriter objXmlWriter =
               new XmlTextWriter(fstmXmlOut,objEncoding);

// Create some XML
objXmlWriter.WriteStartDocument();
objXmlWriter.WriteStartElement("m",
                               "Employees",
                               "http://www.myurl.com");
objXmlWriter.WriteAttributeString("xmlns",
                                  "m",
                                  null,
                                  "http://www.myurl.com");

// Write an employee element
objXmlWriter.WriteStartElement("m",
                               "Employee",
                               "http://www.myurl.com");
objXmlWriter.WriteStartAttribute("m",
                                 "id",
                                 "http://www.myurl.com");
objXmlWriter.WriteString("175-A15");
objXmlWriter.WriteEndAttribute();
objXmlWriter.WriteStartElement("m",
                               "Name",
                               "http://www.myurl.com");
objXmlWriter.WriteString("Kenn Scribner");
objXmlWriter.WriteEndElement(); // Name
objXmlWriter.WriteStartElement("m",
                               "Title",
                               "http://www.myurl.com");
objXmlWriter.WriteString("Code Gecko");
objXmlWriter.WriteEndElement(); // Title
objXmlWriter.WriteEndElement(); // Employee

// Write another employee element
objXmlWriter.WriteStartElement("m",
                               "Employee",
                               "http://www.myurl.com");
objXmlWriter.WriteStartAttribute("m",
                                 "id",
                                 "http://www.myurl.com");
objXmlWriter.WriteString("129-B68");
objXmlWriter.WriteEndAttribute();
objXmlWriter.WriteStartElement("m",
                               "Name",
                               "http://www.myurl.com");
objXmlWriter.WriteString("Mark Stiver");
objXmlWriter.WriteEndElement(); // Name
objXmlWriter.WriteStartElement("m",
                               "Title",
                               "http://www.myurl.com");
objXmlWriter.WriteString("Code Godzilla");
objXmlWriter.WriteEndElement(); // Title
objXmlWriter.WriteEndElement(); // Employee

// Finish off the document
```

```
   objXmlWriter.WriteEndElement(); // Employees

   // Flush it to the file
   objXmlWriter.Flush();
   objXmlWriter.Close();
   fstmXmlOut.Close();

   // Create a file stream
   FileStream fstmXmlIn = new FileStream("MyXML.xml",
                                         FileMode.Open);

   // Create an XML text writer
   XmlTextReader objXmlReader = new XmlTextReader(fstmXmlIn);

   while (objXmlReader.Read())
   {
      switch (objXmlReader.NodeType)
      {
         case XmlNodeType.XmlDeclaration:
            Console.WriteLine("<?xml version=\"1.0\"?>");
            break;

         case XmlNodeType.Element:
            Pad(objXmlReader.Depth);
            Console.Write("<{0}", objXmlReader.Name);
            if ( objXmlReader.HasAttributes )
            {
               bool bAttr =
                        objXmlReader.MoveToFirstAttribute();
               while ( bAttr )
               {
                  Console.Write(" {0}=\"{1}\"",
                                objXmlReader.Name,
                                objXmlReader.Value);
                  bAttr = objXmlReader.MoveToNextAttribute();
               } // while
            } // if
            Console.WriteLine(">");
            break;

            case XmlNodeType.Text:
               Pad(objXmlReader.Depth);
               Console.WriteLine(objXmlReader.Value);
               break;

            case XmlNodeType.EndElement:
               Pad(objXmlReader.Depth);
               Console.WriteLine("</{0}>", objXmlReader.Name);
               break;

            default:
               break;
      } // switch
   } // while

   // Close the file
   objXmlReader.Close();
   fstmXmlIn.Close();
} // try
catch (Exception ex)
```

```
         {
            Console.WriteLine("Exception: {0}\n",ex.Message);
         } // catch
      }

      static void Pad(int iDepth)
      {
         for ( int i = 0; i < iDepth; i++ )
         {
            Console.Write("   ");
         } // for
      }
   }
}
```

Como se ve en el listado, se proporcionan los flujos para XmlTextReader y XmlTextWriter en lugar de solicitar la apertura de los archivos directamente. Esto es conveniente dando los flujos de XML como también se trata con SOAP para .NET.

Creando elementos XML con XmlTextWriter son una cuestión simple de decidir que tipo de elemento para crear y escribir la información del elemento al flujo que usa el método XmlTextWriter adecuadamente. Se usa la combinación de WriteStartElement(), WriteString() y WriteEndElement(), pero también podría usar WriteElementString() o podría crear completamente diferente clases de elementos, como aquéllas dedicadas a un DTD o comentario.

Después de que es creado el archivo XML, es leído detrás de la memoria y desplegado en la pantalla de la consola usando XmlTextReader. Para dar formato a la salida en condiciones más familiares, se determina en este caso el tipo de elemento XmlTextReader. Se trata el inicio de un elemento diferentemente como se trata el contenido o la terminación de la marca del elemento.

El XmlTextReader y XmlTextWriter son buenos, pero trabajan a un nivel bastante bajo cuando tratar con XML. Luego verá cómo subir la jerarquía de abstracción y trabajar con XML a un nivel ligeramente más alto.

Navegación XML

Cuando las personas que conocen el .NET piensan en navegar un documento XML, probablemente piensan en XPathDocument y XPathNavigator ante XmlReader y XmlWriter. Esto es porque esas dos clases de XPath le proporcionan la capacidad para tratar con el documento XML en una pareja de maneras diferentes:

- Primero, puede tener acceso a la información del XML que usa un modelo de extracción. Es decir, obtiene la información del XML como la solicita.

- Segundo, tiene la opción de proporcionar un camino de ubicación XPath para extraer un nodeset con acceso usando XPathNodeIterator.

En la mayoría de los casos, usará una combinación de los dos probablemente.

Simulación de SAX

El API simple para XML fue diseñado para proporcionar un mayor rendimiento y un modelo de programación más simple que XML DOM. Usa un modelo de programación diferente, en lugar de leer en el documento entero inmediatamente y exponer los elementos del documento como nodos, SAX provee un modelo basado en flujo para leer la información de XML.

SAX crea funciones de llamada atrás que el analizador sintáctico de SAX invocará cuando se da cuenta de la información que está interesado en aceptar. Es decir, crea una función de llamada atrás para el inicio de una marca del elemento, SAX invocará esa función siempre que lea una etiqueta inicial. Éste es un modelo de empujón porque SAX empuja XML en su forma cuando encuentra la información dentro del documento.

SAX todavía no es soportado en .NET. De hecho, incluso no es un estándar de Internet oficial. Es una interfaz de programación para XML que fue creado por desarrolladores que quisieron un analizador sintáctico de XML con mayor rendimiento y una traza de la memoria más pequeña, sobre todo al analizar los documentos muy grandes.

Sin embargo, con XPathDocument y XPathNavigator relacionados permiten visitar el contenido XML mediante o consultas XPath o técnicas similares del modelo del empujón como MoveToFirstAttribute(). Otro modelo similar, pero con detalles de implementación distintas es el DocumentNavigator.

2.5.3. Actividades de aprendizaje

Ejercicios sobre acceso de datos en XML

Instrucciones

1. El propósito de esta sección de ejercicios es comprender la infraestructura .NET de la arquitectura núcleo de XML y su relación con los analizadores DOM y SAX. Para lo cual se recomienda la lectura de las secciones siguientes:

- Material de Apoyo “2.4 Esquemas y Consultas XML”, sección tecnologías XPath y XLink.

- Material de Apoyo “2.5 Análisis y Acceso XML”.

2. Implementar los ejercicios siguientes sobre acceso de datos XML:

- Para los archivos que se hace referencia, se proporcionan en el rubro “Código acceso de datos XML” del Anexo C.
- Ejercicio A. Compile el código de programa RdrWtr que utiliza API de flujo XmlReader/XmlWriter.
    - ¿Qué clases son heredadas desde XmlReader y XmlWriter para leer y escribir documentos XML desde el flujo, respectivamente?
    - Investigue qué clases son utilizadas para API DOM y cómo trabajarían en conjunto con las clases de API XmlReader/XmlWriter o en que casos.
- Ejercicio B. Compile el código de programa XPathNav que utiliza API XPathNavigator.
    - ¿Qué diferencia representa API XPathNavigator para el acceso a datos XML con respecto a API XmlReader y API DOM (XmlDocument)?
- Ejercicio C. Compile el código de programa XPathDemo que usa el soporte de API XPathNavigator con la clase XPathNodeIterator.
    - Describa la funcionalidad de XPath.
    - Por último, describa que relación tiene XLink y SAX con las APIs tratadas en estos tres ejercicios.

3. Coloque los resultados de los ejercicios A, B y C en “Tarea Individual Ejercicio (2d)”

2.6. Transformación XML

El XLST (eXtensible Stylesheet Language with Transforms) es un lenguaje diseñado para transformar XML de un vocabulario en otro. Es decir, puede cambiar un formato de documento de XML en otro. XLST utiliza XPath para cambiar XML de un forma en otra.

La importancia de tratar XLST en el contenido de servicios Web, es porque podrían presentarse situaciones en las que se requiera modificar paquetes de SOAP, debido a cambios de versiones o uso frecuente de servicios Web, etc. XSLT es una alternativa excelente para hacerlo. Por ejemplo podría interceptar un paquete que usa un .NET SoapExtension del que .NET no puede ocuparse (quizá por la versión) y a continuación, puede transformarlo en algo que el .NET puede manejar.

En este apartado trataremos aspectos básicos de XSL / XSLT con XPath, debido a que son temas grandes de los que merecen propias referencias individuales.

2.6.1. Comprensión de XSLT

XSLT combina las expresiones de XPath con plantillas. Una plantilla combina un nodeset devuelto desde el camino de una ubicación específica XPath y produce una salida basada en los contenidos de la plantilla. La mayoría de las veces, la plantilla usa los resultados obtenidos del nodeset como entrada.

Digamos que se dio este documento XML:

```
<?xml version="1.0"?>
<Parts>
   <Part vehicle="Corvette" manufacturer="GM">
      <Number>3972178</Number>
      <Desc>Special Hi-Perf Camshaft</Desc>
      <MinYear>1970</MinYear>
      <MaxYear>1972</MaxYear>
      <Comment>LT-1, Solid Lifters</Comment>
   </Part>
   (More part elements...)
</Parts>
```

En la vista del documento tiene centrado las “partes”, pero si deseará verlo centrado como “vehículos” conteniendo las “partes”. El documento XML necesitaría parecerse a esto:

```
<?xml version="1.0"?>
<Vehicles>
   <Vehicle>
      <Make/>
      <Model/>
      <Parts>
         <Part>
            <Number/>
            <Description/>
            <ModelYears/>
         </Part>
      <Parts>
      <Additional parts elements here)
   </Vehicle>
</Vehicles>
```

Para conseguir esto, se muestra la hoja de estilos XSL en el listado siguiente:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
<xsl:output method="xml"/>
```

```
<xsl:template match="/">
   <Vehicles>
      <xsl:apply-templates select="Parts"/>
   </Vehicles>
</xsl:template>

<xsl:template match="Parts">
   <Vehicle>
      <Make><xsl:value-of select="Part/@manufacturer"/></Make>
      <Model><xsl:value-of select="Part/@vehicle"/></Model>
      <Parts>
         <xsl:for-each select="Part">
            <Part>
               <Number>
                  <xsl:value-of select="Number"/>
               </Number>
               <Description>
                  <xsl:value-of select="Desc"/>
                  <xsl:text>: </xsl:text>
                  <xsl:value-of select="Comment"/>
               </Description>
               <ModelYears>
                  <xsl:value-of select="MinYear"/>
                  <xsl:text> to </xsl:text>
                  <xsl:value-of select="MaxYear"/>
               </ModelYears>
            </Part>
         </xsl:for-each>
      </Parts>
   </Vehicle>
</xsl:template>

</xsl:stylesheet>
```

Si corre el archivo del XML y hoja de estilos a través de Internet Explorer, no podría conseguir la salida que espera. Internet Explorer ejecutará la transformación, pero no desplegará el resultado como XML. Si está interesado en ver la salida transformada real, es mejor ejecutar el documento XML y hoja de estilos a través de un programa diseñado para ejecutar la hoja de estilos y guardar la salida resultante a un archivo para una llamada posterior. Este programa lo trataremos más adelante.

XSL es un vocabulario del XML, es decir, sus propias hojas de estilos de XSL serán los documentos del XML. Varios elementos son asociados con XSL, la tabla siguiente muestra las más habitualmente instrucciones XSL usadas.

Operador	Propósito
<apply-templates/>	Comunica al procesador XSL aplicar todas las plantillas adecuadas al nodeset
<call-template/>	Invoca una plantilla por nombre
<choose/>	Usa la comprobación condicional múltiple (uso con when y otherwise)
<for-each/>	Aplica una plantilla repetidamente a todos los nodos en el nodeset
<if/>	Es una estructura condicional (no tiene la claúsula else, en su lugar use < choose / >)
<number/>	Inserta un entero formateado en el documento de la salida
<otherwise/>	Es una expresión condicional predeterminada usada junto con < choose / >
<output/>	Especifica serialización del árbol resultante
<sort/>	Ordena los nodos de la salida (reorganiza el documento de salida)
<stylesheet/>	Es la hoja de estilos XSL de elemento documento (raíz)
<template/>	Es la marca de la plantilla (identifica y encapsula la plantilla)
<text/>	Inserta texto literal en el documento de la salida
<value-of/>	Copia el nodo del documento de la entrada al documento de la salida
<when/>	Es una expresión condicional optativa usada junto con < choose / >

Aquí ve el elemento documento de XSL, <xsl:stylesheet/>, así como el soporte de XSL, <xsl:template/>. El elemento de la hoja de estilos encapsula la propia hoja de estilos, mientras el elemento de la plantilla identifica lo que podría estar llamando de subrutinas XSL. De hecho, no son subrutinas, pero podría verlo de esa manera. Aunque no mostramos los elementos de XSL relacionados a las variables, XSL tiene la capacidad de pasar variables y parámetros entre las plantillas. Porque son tan poderosos, miremos las plantillas de XSL en poco más detalle.

Plantillas de XSL

En el listado de ejemplo de hojas de estilo XSL anterior, proporciona dos plantillas, una para el elemento documento y una para el elemento <Parts/> . Podríamos crear más plantillas, uno para cada nodo <Part/> e hijos, pero en cambio elegimos tratar más profundamente en el documento XML original el uso de <xsl:for-each/>. Esto simplemente hizo más fácil leer la plantilla.

La propia plantilla es el cookie cortador del que XSL usará para crear el documento nuevamente formateado. Para cada plantilla que incluye dentro de su hoja de estilos, recibirá la salida formateada. La clave es preparar las plantillas para que operen en el nodeset devuelto del atributo en juego. Si conoce varios de los nodos XML que emparejarán la plantilla, pero quiere reprimir el uso de algunos de esos nodos, también puede aplicar modo atributo a la plantilla:

```
<xsl:template match="/">
   ...
   <xsl:apply-templates select="Parts" mode="Engine" />
   <xsl:apply-templates select="Parts" mode="Body" />
   ...
</xsl:template>

<xsl:template match="Parts" mode="Engine">
   (Something to do with engine parts...)
</xsl:template>

<xsl:template match="Parts" mode="Body">
   (Something to do with body parts...)
</xsl:template>
```

Esencialmente, el modo le permite procesar los nodos del documento original muchas veces, con cada iteración de procesamiento produciendo un resultado diferente.

Note que también usamos la instrucción XSL <xsl:apply-templates/>. Esto le comunica a XSL que pase por su colección de la plantilla y para cualquiera de los nodos de XML de entrada que emparejan las plantillas, produzca la salida. La clave es crear expresiones de XPath que deducen del documento XML el nodeset preciso que quiera trabajar con él.

La plantilla es la clave para transformar el documento XML original. Después de que establece la expresión de XPath que tirará el conjunto de nodos de interés, inserta la nueva estructura de documento XML dentro de la plantilla. Para cada nodo del XML XSL encuentra que empareja la plantilla, arroja una copia de la plantilla resultante en el nuevo documento XML.

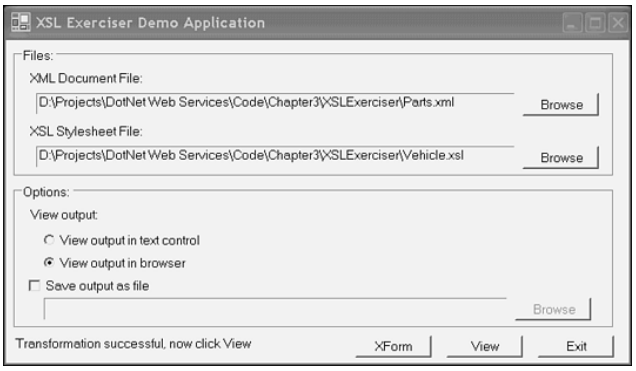
2.6.2. Uso de XSL

La tecnología de uso general XSL es soportada por .NET a través de su clase XSLTransform. Si carga un documento XML en una instancia de XPathDocument y a continuación, carga una hoja de estilos de XSL en XSLTransform, puede ejecutar la hoja de estilos de esta manera:

```
objXSL.Transform(objXMLDoc,null,objWriter);
```

Aquí, el objXSL es la instancia de XSLTransform, el objXMLDoc es el XPathDocument y el objWriter es el XmlTextWriter que usará para persistencia de los cambios menores.

Para proporcionar una demostración útil de transformaciones dentro de .NET, veamos una aplicación creada y nombrada XSLExerciser, cuya interfaz de usuario se muestra en la figura siguiente:



Si navega para un archivo del XML y una hoja de estilos asociada, puede hacer clic en el botón XForm para ejecutar la transformación. Si ha elegido para guardar los resultados, el nuevo documento del XML se grabará dentro del archivo que especificó. También puede ver o la salida como texto o cuando XML contuvo dentro del control del Explorador Web.

No necesita cargar cualquier archivo realmente hasta usted quiere realizar la transformación, para que si necesita ajustar la hoja de estilos, como a menudo pasa, puede hacer tan fácilmente y volver a transformar el documento. En seguida se muestra el código de programa de la acción de transformar, cuando hace clic al bóton XForm.

```
private void cmdXForm_Click(object sender, System.EventArgs e)
{
   try
   {
      // Open the original XML document
      m_objXMLDoc = new XPathDocument(m_strXMLPath);

      // Open the stylesheet
      XslTransform objXSL = new XslTransform();
      objXSL.Load(m_strXSLPath);

      // Create a stream to contain the results
```

```
      m_stmOutput = new MemoryStream();

      // Create and overlay an XML writer
      ASCIIEncoding objEncoding = new ASCIIEncoding();
      XmlTextWriter objWriter = new XmlTextWriter(m_stmOutput,
                                                  objEncoding);

      // Do the transform
      objXSL.Transform(m_xpdXMLDoc,null,objWriter);

      // Save output as a string
      byte[] bytes = m_stmOutput.ToArray();
      m_strXMLOutput = objEncoding.GetString(bytes);

      // Check to see if we need to save the file...
      if ( optSave.Checked && m_strOutPath.Length > 0 )
      {
         FileStream fstmXMLOut = new FileStream(m_strOutPath,
                                                FileMode.Create);
         m_stmOutput.Position = 0;
         TextReader objTRFrom = new StreamReader(m_stmOutput);
         TextWriter pbjTWTo = new StreamWriter(fstmXMLOut);
         twTo.WriteLine(trFrom.ReadToEnd());
         twTo.Flush();
         fstmXMLOut.Close();
      } // if

      // Close the stream
      m_stmOutput.Close();

      // Enable the UI
      cmdView.Enabled = true;

      // Show results
      lblResults.ForeColor = SystemColors.ControlText;
      lblResults.Text = "Transformation successful, now click View";
   } // try
   catch (Exception ex)
   {
      // Show results
      lblResults.ForeColor = Color.Red;
      lblResults.Text = "Transformation Unsuccessful";

      MessageBox.Show(ex.Message,
                      "XSLExerciser Error",
                      MessageBoxButtons.OK,
                      MessageBoxIcon.Error);
   } // catch
}
```

Este código fundamentalmente lo que hace es crear un flujo basado en memoria en el que se empuja el XML transformado:

```
// Create a stream to contain the results
m_stmOutput = new MemoryStream();
```

Hacemos esto porque tenemos el XML transformado entonces en una situación, prepare desplegar o almacenar en un archivo.

2.6.3. Actividades de aprendizaje

Ejercicio sobre transformación XML: XSLT

Instrucciones

1. El propósito de este ejercicio es visualizar el alcance de transformación de documentos XML con XSLT que puede tomar una implementación de interfaz API XPathNavigator sobre las demás APIs de XML .NET.

No obstante que en los ejercicios anteriores ya hemos trabajado con algunas implementaciones API XPathNavigator, en este ejercicio trataremos la transformación usando XPathDocument (XPath/XSLT). Para lo cual se recomienda la lectura de la sección siguiente:

- Material de Apoyo “2.5 Transformación XML”

2. Implementar el ejercicio siguiente sobre transformación XML:

- Para los archivos que se hace referencia, se proporcionan en el rubro “Código transformación XML” del Anexo C.
- Ejercicio. Compile el código de programa XSLTDemo que utiliza la clase XslTransform de XSLT.
    - Describa la funcionalidad del programa XSLTDemo.

3. Coloque el resultado del ejercicio en “Tarea Individual Ejercicio (2e)”

2.7. Ejercicios prácticos

Ejercicios de clase asignados por el profesor

2.8. Cuestionarios

Tarea asignada por el profesor

3. Invocación de Servicios Web: SOAP

SOAP (Simple Object Access Protocol) es un protocolo basado en XML para intercambiar información estructurada y tipificada entre socios en un descentralizado entorno distribuido. El W3C empezó trabajando con SOAP en 1999, la recomendación actual de W3C es la versión 1.2.

SOAP cubre las cuatro partes principales siguientes:

- Un formato de mensaje para comunicaciones en una dirección, describiendo como se organiza la información en un documento XML.

- Un conjunto de reglas que cualquier entidad que procesa un mensaje SOAP debe seguir. Definen que elementos debería leer y comprender, y las acciones que deberían realizar sino entienden el contenido.

- Un conjunto de normas para implementar interacciones estilo RPC mediante mensajes SOAP, definiendo como los clientes pueden invocar un procedimiento remoto enviando un mensaje SOAP y como los servicios pueden replicar enviando otro mensaje al cliente.

- Una descripción de cómo su mensaje SOAP se debería transportar sobre HTTP y SMTP.

Los mensajes SOAP son eficazmente solicitudes de servicio enviadas para algún punto terminal en una red. Ese punto terminal puede implementarse en cualquier número de formas: servidor RPC, objeto de COM, servlet de Java, script de Perl. Estos pueden estar ejecutando en cualquier plataforma. Así, SOAP está sobre la interoperabilidad entre aplicaciones que ejecutan en plataformas potencialmente dispares que usan varias tecnologías de aplicación en varios lenguajes de programación.

3.1. El mensaje SOAP

Aunque originalmente concebido como una tecnología para puentear el hueco entre plataformas de comunicación dispares basado en RPC, SOAP ha evolucionado en el más amplio soporte de formato de mensaje y protocolo para el uso con servicios Web XML.

La especificación SOAP establece un formato de mensajes estándar que consiste de un documento XML capaz de organizar RPC y datos basados en documento (ver figura). Esto facilita modelos síncronos (petición y respuesta) así como modelos de intercambio asincrónico de datos (proceso-manejado). Junto con

WSDL establecen un formato estándar de descripción del punto terminal para las aplicaciones, el formato de mensaje basado en documento es por mucho el más común.

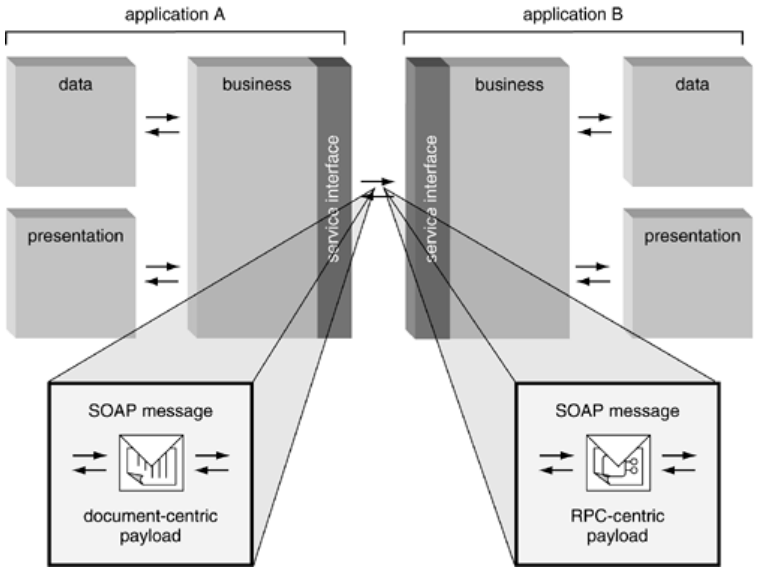


Figura. Establece dos formatos estándar primarios

3.1.1. Bases del mensaje SOAP

El documento o mensaje SOAP se transmite entre aplicaciones y pueden atravesar a varios intermediarios cuando viajan del remitente inicial al destinatario definitivo. El documento SOAP está compuesto de tres secciones: Envelope (sobre), Header (encabezado) y Body (cuerpo). La figura ilustra la estructura de documento SOAP.

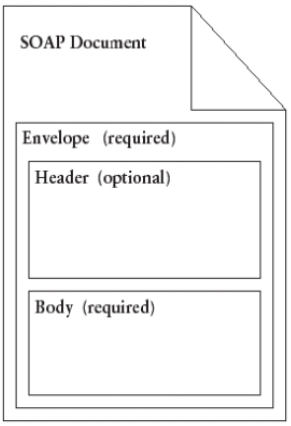


Figura. Estructura de un documento SOAP

Los estándares de SOAP definen tres parámetros principales:

- **Estructura del contenido Envelope.** Contiene información como que métodos a invocar, parámetros optativos, valores devueltos, y dónde algo no se ejecutó con éxito, las excepciones optativas (conocidas como faults SOAP).

- **Reglas de codificación de datos.** Puesto que SOAP tiene que soportar lenguajes múltiples y sistemas operativos, tiene que definir una representación universalmente aceptada para los tipos de datos diferentes como punto flotante, entero y arreglo; así como tipos de datos complejos con codificación personalizada.

- **Convenciones uso.** Puede usarse SOAP en una multitud de maneras, pero son todas variaciones de las mismas acciones: un remitente envía un documento del XML y el destinatario, opcionalmente, devuelve una contestación en la forma de un documento XML. Convenciones también para el documento XML que puede tener "faults" si errores ocurrieron durante el procesamiento.

Para permitir a los destinatarios sean enlazados, las arquitecturas basadas en SOAP pueden ser sofisticadas. La figura muestra cinco arquitecturas comunes que son usadas con sistemas basados en SOAP: Envía y Olvida, Solicita Respuesta, Notificación, Difusión y Flujo de Trabajo/Orquestación.

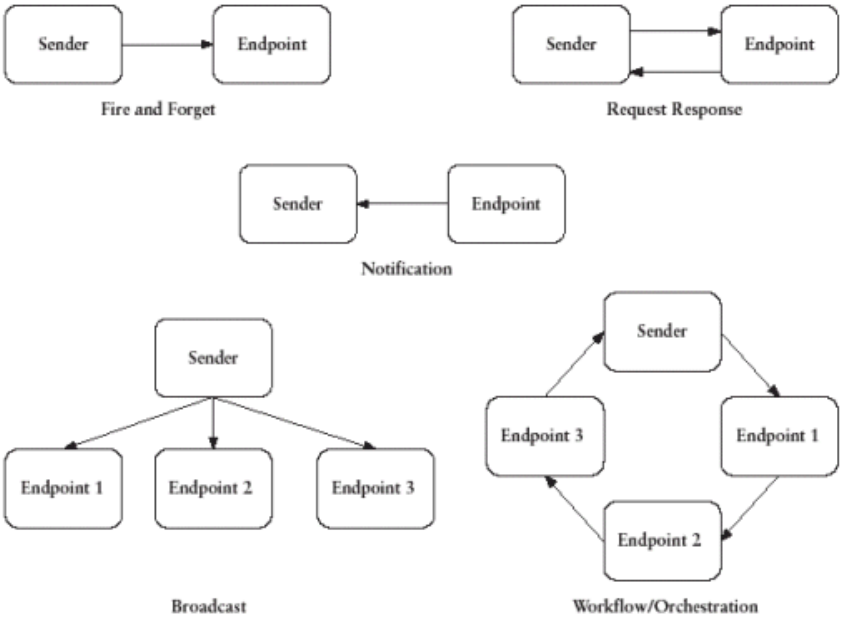


Figura. Arquitecturas comunes SOAP

Cualquier enlace en la cadena de procesamiento que no es punto terminal es llamado un intermediario. Las especificaciones de SOAP le permiten a un intermediario procesar un mensaje de SOAP parcialmente antes de pasarlo al próximo enlace en la cadena de procesamiento, qué puede ser otro intermediario o punto terminal. Estos también se les conoce como nodos SOAP, la figura siguiente ilustra un nodo SOAP, actuando como un intermediario, durante las transiciones como remitente y destinatario SOAP en el procesamiento de un mensaje SOAP.

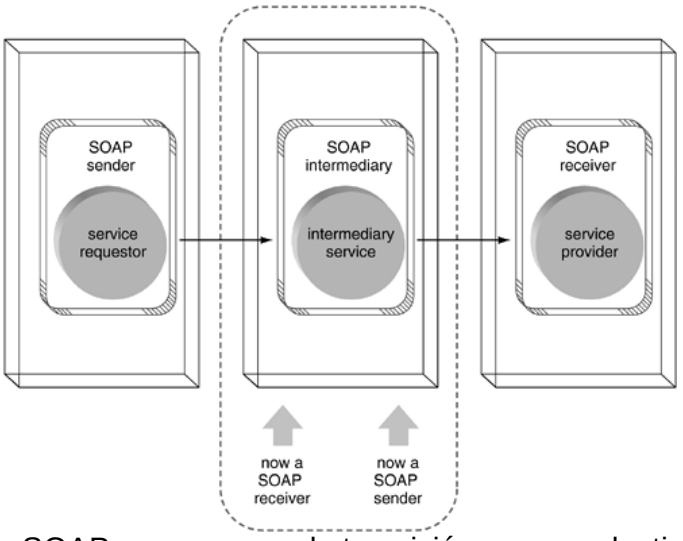


Figura. Un nodo SOAP que pasa por la transición a ser un destinatario SOAP y remitente durante el procesamiento de un mensaje de SOAP

La migración desde XML para SOAP resulta ser un procedimiento simple, que incluye los pasos siguientes:

1. Agregar elementos Header optativos.
2. Envolver el cuerpo del documento XML en el body de SOAP que a su vez es incluido en Envelope SOAP.
3. Declarar los espacios de nombres de SOAP adecuados.
4. Agregar manejador de excepción optativo.
5. Especificar el protocolo que debe usarse.

El ejemplo siguiente muestra la migración de un documento XML para ser envuelto por SOAP.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
        xmnls:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
        xmnls:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xmnls:xsi="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Header>
        . . . [optional header information]
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<Order>
    <Customer>
        <name>John Doe</name>
        <street>1111 AnyStreet<street>
        <city>AnyTown</city>
        <state>GA<state>
        <zip>10000</zip>
    </Customer>
</Order>
<SOAP-ENV:Body>
<SOAP-ENV:Envelope>
```

Envelope SOAP

Envelope SOAP es el recipiente para los otros elementos en el mensaje de SOAP, es el elemento raíz que representa el documento del mensaje.

Un proceso del lado del servidor llamado manejador SOAP puede usar la disponibilidad de Envelope SOAP con la declaración de espacio de nombres, para determinar si el documento XML entrante es un mensaje de SOAP o no. El manejador puede ser parte de la aplicación del servidor o puede ser un producto externo.

Header SOAP

Como parte de su meta de diseño de extensibilidad, los arquitectos de SOAP proporcionaron el elemento Header para permitir extender los mensajes de SOAP genéricamente todavía mientras conformándose las especificaciones de SOAP.

Si un elemento de Header SOAP es actual (puede haber más de un elemento Header presente), tiene que ser el primer hijo del elemento de Envelope. Cada elemento Header puede tener elementos hijo a su vez.

Usos comunes de información Header incluyen:

- Implementación de extensiones SOAP que son aplicaciones predefinidas o específicas.

- Identificación de destinatarios intermediarios de SOAP.

- Provisión de información de la meta suplementaria sobre el mensaje de SOAP.

Mientras un mensaje de SOAP progresa a lo largo de una ruta del mensaje, los intermediarios pueden agregar, borrar o procesar información en los Header SOAP. Otras opciones en las especificaciones para Header SOAP serán tratadas más adelante, como el caso de las extensiones SOAP.

Dos ejemplos que usan información Header para proporcionar la extensibilidad incluyen:

- Información de autenticación alojada.

- Especificación de un número de cuenta para el uso con un servicio de SOAP de pago-por-uso.

Un intermediario de SOAP puede usar esta información del Header para determinar si el mensaje entrante es autorizado propiamente antes de remitirlo a otro intermediario o punto terminal.

Manejo de excepción

En casos dónde un manejador de SOAP no puede descifrar un mensaje, una falla SOAP se genera, identificado por el elemento Fault. Su elemento hijo, faultcode, identifica la categoría de errores que pueden suceder. En seguida se describen valores definidos para el elemento faultcode:

- **VersionMismatch.** El destinatario del mensaje encontró un espacio de nombres no válido para el elemento de Envelope SOAP.

- **MustUnderstand.** El destinatario encontró un elemento del Header predefinido como obligatorio que no podía entender.

- **Client.** La falla estaba en el mensaje que se recibía. Posibles causas: elementos perdidos, elementos malformados, y similares.

- **Servidor.** La falla ocurrió en el lado del destinatario, es decir, un error del servidor.

Considere que una aplicación es libre para extender esos valores usando la notación (.). Por ejemplo, un valor Client.Login puede usarse para especificar que había un problema con un inicio de sesión del cliente.

Además del elemento faultcode, hay otros dos elementos que pueden usarse para proporcionar mayor claridad sobre la falla:

- **faultstring.** Este elemento proporciona una explicación legible adelante por qué la falla ocurrió.

- **details.** El valor del elemento details, indica que el problema ocurrió mientras procesaba el elemento body. Si el elemento detalle no es actual, entonces la falla ocurrió fuera del body del mensaje.

Otros elementos o atributos de información Header son:

relay. Incluido en SOAP 1.2, indica si información Header apuntará a un intermediario SOAP deba liberarse si no se procesa.

role. Atributo que identifica elementos Header para indicar los tipos específicos de destinatarios SOAP que efectúan el proceso. El nodo SOAP que procesa el mensaje puede jugar el papel de none, next, ultimateReceiver. None es para propagar información que no debe ser procesada, Next indica que un nodo que reciba el mensaje puede procesar ese elemento Header y ultimateReceiver indica que el Header va dirigido al destinatario final del mensaje (ver figura).

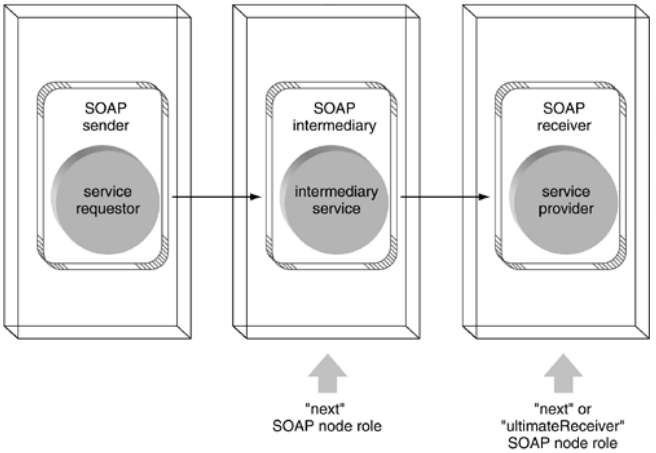


Figura. Roles que pueden asumir los nodos destinatarios.

Body SOAP

La parte de un mensaje de SOAP que no es optativo es el Body. Está sección actúa como un recipiente de los datos que serán entregados por el mensaje SOAP. Los datos dentro Body SOAP a menudo se les refiere como la carga útil o datos de carga.

El body SOAP también puede usarse para organizar la información de la excepción dentro de los elementos Fault anidados. Aunque las secciones de Fault pueden residir junto a las cargas útiles de los datos estándares, este tipo de información se envía a menudo separadamente en mensajes de respuesta que comunican las condiciones de error.

Procesamiento del mensaje

El Listado 1 muestra un mensaje SOAP hipotético que podría usarse para comunicar un pedido entre la compañía Angus Hardware y un abastecedor. Usaremos este ejemplo durante la discusión de esta sección.

Listado 1. Un mensaje SOAP para un pedido del producto al por mayor.

```
<?xml version ="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <SOAP-ENV:Header>
    <ns1:terms xmlns:ns1="urn:angushardware"
      SOAP-ENV:MustUnderstand="1">Net 30</ns1:terms>
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
    <ns1:placeOrder
      SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <productCode xsi:type="xsd:string">99HGTY</productCode>
      <quantity xsi:type="xsd:int">300</quantity>
    </ns1:placeOrder>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Hay cuatro espacios de nombres incluidos en el mensaje SOAP. El primero, con el prefijo SOAP-ENV, se refiere a Envelope SOAP:

```
xmlns:SOAP-ENV = "http://schemas.xmlsoap.org/soap/envelope/"
```

El elemento SOAP-ENV:Envelope tiene dos subelementos, SOAP-ENV:Header y SOAP-ENV:Body. El elemento SOAP-ENV:Header (optativo) proporciona la información a la aplicación que procesa el mensaje.

El elemento dentro del elemento SOAP-ENV:Header puede ser cualquier elemento de cualquier espacio de nombres de otra manera que SOAP-ENV y puede tener cualquier atributo sin restricción. Aunque está abierto para permitir la flexibilidad máxima, los sub-elementos de SOAP-ENV:Header tienen los dos atributos optativos específicos, MustUnderstand y Role.

El atributo MustUnderstand tiene un valor Booleano, indicando a la aplicación que si no sabe procesar el mensaje, debe desechar el mensaje entero. El atributo Role cuyo valor es un URI, puede usarse cuando un mensaje SOAP se envía a varias aplicaciones en una ruta del mensaje, e indica cuál aplicación el mensaje esta para ser considerado:

```
<SOAP-ENV:Header>
  <ns1:terms xmlns:ns1="urn:angushardware"
    SOAP-ENV:MustUnderstand="1">Net 30</ns1:terms>
</SOAP-ENV:Header>
```

En el listado el elemento ns1:terms indica las condiciones que Angus Hardware está deseoso para dar a su vendedor de la compra de mayor inventario. Las condiciones son Net 30 y la aplicación que procesa el pedido debe entender el elemento ns1:terms para continuar el procesamiento.

Igual que SOAP-ENV:Header, el elemento SOAP-ENV:Body puede tener cualquier subelemento. Sin embargo, a diferencia de SOAP-ENV:Header, SOAP-ENV:Body no puede tener ningún atributo. El contenido del elemento SOAP-ENV:Body es una secuencia de acciones a ser procesadas. En el listado 1, la acción es una llamada al método placeOrder:

```
<SOAP-ENV:Body>
  <ns1:placeOrder
    SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <productCode xsi:type="xsd:string">99HGTY</productCode>
    <quantity xsi:type="xsd:int">300</quantity>
  </ns1:getInventory>
</SOAP-ENV:Body>
```

El elemento ns1:placeOrder tiene un atributo, SOAP-ENV:encodingStyle. Este atributo indica las reglas de codificación para el mensaje del que hablaremos un poco más.

Un Envelope SOAP también puede tener una respuesta, el listado 2 muestra un ejemplo.

Listado 2. Un mensaje de respuesta de SOAP para un pedido del producto al por mayor.

```
<?xml version ="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <SOAP-ENV:Body>
    <ns1:placeOrderResponse xmlns:ns1="urn:angushardware"
      SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <ns1:deliveryDate xsi:type="xsd:date">2002-09-
04</deliveryDate>
    </ns1:placeOrderResponse>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

El Listado 2 muestra una posible respuesta al mensaje en el Listado 1. En este caso, el servidor responde con un elemento ns1:placeOrderResponse, conteniendo un elemento ns1:deliveryDate que indica una fecha de entrega esperada 2002-09-04. Sin embargo, una respuesta de SOAP también puede incluir el elemento SOAP-ENV:Fault, indicando un error. El Listado 3 muestra un mensaje Fault SOAP.

Listado 3. Un mensaje Fault SOAP para un pedido del producto al por mayor

```
<?xml version ="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <SOAP-ENV:faultCode xsi:type="xsd:string">
        SOAP-ENV:MustUnderstand
      </SOAP-ENV:faultCode>
      <SOAP-ENV:faultString xsi:type="xsd:string">
        The server did not understand the header element ns1:terms
      </SOAP-ENV:faultString>
      <SOAP-ENV:faultActor xsi:type="xsd:string">
        Smith's Sprocket Company
      </SOAP-ENV:faultActor>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Como puede ver, este mensaje SOAP es muy similar en estructura a ambos los ejemplos anteriores, sólo que incluye el elemento SOAP-ENV:Fault.

El elemento SOAP-ENV:faultCode contiene un código de falla que indica el tipo de error que ocurrió. SOAP-ENV:faultString proporciona la explicación humano-legible de la falla; también opcionalmente el elemento SOAP-ENV:detail podría proporcionar más información sobre el error. Finalmente, el SOAP-ENV:faultActor indica cuál actor tenía la falla. En este caso, el servidor de SOAP en la compañía Smith's Sprocket tiene indicado que no entiende el elemento ns1:terms en el Header de la petición.

Los elementos SOAP-ENV:Header y SOAP-ENV:Fault son optativos en cualquier mensaje SOAP. Los dos mensajes de SOAP, petición y respuesta, tienen la estructura idéntica realmente; la única diferencia es en que de los dos elementos optativos es incluido y en la sintaxis específica de los elementos del Body.

La estructura total del mensaje SOAP es sumamente flexible que lo hace importante para el cliente y servidor de reglas bien establecidas para la sintaxis de sus comunicaciones.

Reglas de codificación

Algunos de los elementos en los mensajes de SOAP discutidos brevemente incluyen el atributo SOAP-ENV:encodingStyle:

```
SOAP-ENV:encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
```

El atributo SOAP-ENV:encodingStyle especifica al sistema de codificación a ser usada. El sistema de codificación es mutuamente convenido sobre la manera de representar los datos en un mensaje SOAP. Aunque los ejemplos usan las reglas de codificación SOAP 1.1, puede actualmente usar cualquier codificación que desee o ninguna, en su Envelope SOAP. Para usar el sistema de codificación SOAP 1.2, por ejemplo, especificaría http://www.w3.org/2003/05/soap-encoding. El URI, igual que los espacios de nombres URIs, se usa como un único nombre en lugar de un recurso actual de Internet.

El estilo de la codificación aplica al alcance entero del elemento en el que el atributo aparece. En los listados 1 y 2, los atributos SOAP-ENV:encodingStyle aparecen en los elementos ns1:placeOrder y ns1:placeOrderResponse, respectivamente.

3.1.2. Ligaciones SOAP

Ligación SOAP para HTTP

SOAP define una ligación al protocolo de HTTP. Esta ligación describe la relación entre las partes del mensaje de petición de SOAP y varios Headers HTTP. Todas las peticiones SOAP usan el método HTTP POST y específicamente tres Headers HTTP por lo menos: Content-Type, Content-Length y un Header personalizado SOAPAction. El mensaje SOAP actual se pasa como el Body de la petición o respuesta.

Content-Type

```
Content-Type: text/xml; charset=character encoding
```

El Header de Content-Type para las peticiones de SOAP y las respuestas especifican el tipo MIME para el mensaje y siempre es text/xml. También puede especificar la codificación de carácter usada para el Body XML de la petición de HTTP o respuesta. Esto sigue la parte text/xml de los valores del Header.

Ejemplo de uso de Content-Type

```
POST /endpoint.pl HTTP/1.1
Content-Type: text/xml
```

Un ejemplo Header de Content-Type en una petición de HTTP

Content-Length

El Header de Content-Length para las peticiones y respuestas de SOAP se establecen en el número de bytes en el Body de la petición o respuesta.

Ejemplos de uso de Content-Length

```
POST /endpoint.pl HTTP/1.1
Content-Type: text/xml
Content-Length: 167
SOAPAction: urn:example-org:demos#Method

<s:Envelope
  xmlns:s='http://schemas.xmlsoap.org/soap/envelope/' >
 <s:Body>
  <m:Method xmlns:m='urn:example-org:demos' />
 </s:Body>
</s:Envelope>
```

Un ejemplo Header de Content-Length en una petición de HTTP. La petición es codificada usando un formato de codificación de 8 bits.

Uso de Content-Length con conjunto de caracteres

```
POST /endpoint.pl HTTP/1.1
Content-Type: text/xml; charset=UTF-16
Content-Length: 167
SOAPAction: urn:example-org:demos#Method

<s:Envelope
  xmlns:s='http://schemas.xmlsoap.org/soap/envelope/' >
 <s:Body>
  <m:Method xmlns:m='urn:example-org:demos' />
 </s:Body>
</s:Envelope>
```

Ejemplo Header de Content-Length de en una petición de HTTP. La petición es codificada usando un formato de codificación de 16 bits.

SOAPAction

El Header de SOAPAction indica al servidor HTTP que la petición es una petición de SOAP. El valor del Header es un URI. Más allá de eso, su valor es indefinido.

Ejemplo de uso de SOAPAction

```
POST /endpoint.pl HTTP/1.1
Content-Type: text/xml; charset=UTF-16
Content-Length: 167
SOAPAction: urn:example-org:demos#Method

<s:Envelope
  xmlns:s='http://schemas.xmlsoap.org/soap/envelope/' >
 <s:Body>
  <m:Method xmlns:m='urn:example-org:demos' />
 </s:Body>
</s:Envelope>
```

Ejemplo Header de SOAPAction en una petición de HTTP. La cadena que precede el # es el nombre del espacio de nombres del primer hijo del elemento del Body considerando que la cadena que sigue el # es el nombre local de ese elemento.

Representación de RPC

Aunque Envelope SOAP proporciona la información sobre el método remoto a ser llamado y la codificación de los datos que se pasan al objeto remoto, no contiene ninguna información inherentemente sobre el programa remoto o sistema. SOAP depende del protocolo de transporte para proporcionar esto. Mientras HTTP trabaja muy bien como un transporte, y SOAP tiene las ligaciones específicas para HTTP, no hay nada en la especificación que limite su opción de protocolos de transporte.

3.1.3. Actividades de aprendizaje

Ejercicio sobre peticiones de servicios Web

Instrucciones

1. Revisar y estudiar la presentación “Manejo de petición de servicios Web con .NET” que se proporciona en el anexo B.

2. Con base a la presentación previa, implementar el manejo de petición de servicios Web para los métodos Web de la aplicación elaborada en la sección “Actividades de aprendizaje 1.6”:

- Programa en C# de .NET para acceder los métodos Web vía HTTP GET
- Programa en C# de .NET para acceder los métodos Web vía HTTP POST
- Programa en C# de .NET para acceder los métodos Web vía SOAP

Coloque sus programas en “Tarea Individual Ejercicio (3a)”

3. Contestar el cuestionario “Preguntas mensaje SOAP”

- Coloque sus respuestas en “Tarea Individual Cuestionario (3a)”

3.2. Implementaciones SOAP

3.2.1. Serialización y extensiones SOAP

Reglas de serialización SOAP

La serialización se refiere al proceso de transformar los datos de una instancia de objeto en memoria en una representación estructurada de esos datos en el flujo. La serialización permite preservar el estado de los objetos de una aplicación, si simplemente guarda los datos que está trabajando o transmite los datos a otra aplicación. El proceso de leer el flujo de datos a una nueva instancia de objeto en memoria es la deserialización que es lo contrario de serialización.

SOAP define un conjunto de reglas de la serialización para codificar los tipos de datos en XML. Todos los datos son serializados como elementos en lugar de atributos. Sólo se usan atributos para la estructura de metadatos; por ejemplo, cuando se necesitan referencias.

Para los tipos simples como cadenas, números, fechas y sucesivamente, se usan los tipos de datos definidos para Esquemas XML.

Para los tipos como clases o estructuras, cada campo en el tpo es serializado usando un elemento con el mismo nombre como el campo.

Para los tipos arreglos, cada elemento del arreglo es serializado usando un elemento típicamente con el mismo nombre como el tipo, aunque pueden usarse otros nombres del elemento. En ambos casos, si el campo en la serialización sea una estructura o arreglo, se usan elementos anidados.

El elemento a nivel cima en el caso de la estructura y el caso del arreglo es el espacio de nombres calificado. Los elementos descendientes no deben estar calificados.

Las reglas de la serialización se aplican a los hijos del elemento Header así como los hijos del elemento Body. Tales hijos son tipos serializados justo como cualquier otro tipo. También se trata una petición y cualquier respuesta asociada como tipos y son serializadas según las mismas reglas.

La especificación de SOAP determina reglas de serialización para los tipos siguientes:

1. Simples estructuras de datos.
2. Estructuras de datos con múltiples referencias.
3. Referencias null estructuras de datos complejas.

4. Datos dinámicamente tipificados.
5. Arreglos.
6. Arreglos multidimensionales.
7. Transmisión parcial de arreglos.
8. Arreglos esparcidos.
9. Arreglos de arreglos.

La infraestructura .NET vislumbra varios esquemas de serialización:

Serialización XML

La representación estructurada está definida vía la sintaxis de XML. Una simple forma de serialización XML es usando la clase XmlSerializer para controlar la transformación que especifica un XSD.

En general, el uso de serialización XML se exige cuando su aplicación requiere intercambiar datos entre sistemas posiblemente dispares cuya sola riqueza podría ser la capacidad de lectura y escritura de XML.

La serialización XML simple es adecuada cuando tiene un esquema XML existente y desea leer los datos en un objeto; o cuando tiene los objetos existentes y desea producir una representación de sus datos en un formato XML. Estos casos normalmente envuelven el intercambio de datos no interactivo; es decir, los datos están intercambiados, pero no en un contexto de servicios Web.

Serialización en tiempo de ejecución

Utiliza una clase formatter para crear una versión serializada de un objeto, usando la información disponible sobre el objeto desde reflection. Reflection es el mecanismo por el que los objetos en memoria en tiempo de ejecución pueden ser examinados sobre información de sus campos, propiedades, métodos y atributos.

La serialización en tiempo de ejecución es considerada cuando la comunicación está pasando entre dos aplicaciones del .NET.

Serialización SOAP

Hay otra forma de serialización XML que puede parecer redundante en principio. Retoma la serialización en tiempo de ejecución para codificar un objeto mediante SOAP. El SoapFormatter produce un flujo de SOAP que esta perfeccionado para recrear el objeto original en otra aplicación .NET; específicamente, se codifica el objeto y todos sus miembros usando los tipos de CLR. Una aplicación no .NET que lee ese flujo de SOAP no tendría ninguna idea probablemente qué ver con los datos.

Sin embargo, el XmlSerializer también puede serializar un objeto para SOAP, con un énfasis en las codificaciones de SOAP estándares. Con serialización de SOAP, consigue toda la interoperabilidad de XML, con conocimiento de CLR adicional. La clave para la serialización de SOAP de conformidad estándar es la clase SoapReflectionImporter.

La documentación SDK .NET le dirá que SoapReflectionImporter es reservado para uso interno y no debe ser usado por su aplicación. Sin embargo, tiene un constructor y un método que puede usar para serializar los objetos para SOAP.

La serialización SOAP puede usarse cuando sabe que su socio del intercambio de datos lo soporta, o cuando está diseñando una nueva aplicación distribuida que requiere el intercambio de datos interactivo.

Extensiones SOAP

Como hemos visto, cuando un cliente de servicio Web invoca un método Web que está en un servidor de servicio Web, el cliente regresa los objetos y parámetros involucrados en un mensaje XML: la petición de SOAP.

Además, este proceso de convertir los objetos a XML se llama serialización. En la terminación del servidor, el proceso inverso (llamado deserialización) se usa para retornar el XML en los objetos. El servidor entonces regresa la respuesta de SOAP que pasa por un proceso similar de serialización y deserialización.

Las extensiones SOAP le permiten que ejecute su propio código como parte de los procesos de serialización o deserialización.

Una extensión SOAP puede realizar tareas como la encriptación, comprobación de firma, traducción o cualquier otra modificación de los mensajes de SOAP que están pasando.

En .NET las extensiones SOAP se derivan de la clase SoapExtension. En esencia, una clase SoapExtension le permite profundizar en el XML de SOAP que el .NET ha producido o simplemente ha recibido para que pueda alterar contenidos o de otra forma poder realizar procesamiento necesario.

Ejemplos de uso con SOAPExtension:

- Podría usar XPath para extraer un elemento de SOAP XML, encriptarlo y devolverlo al flujo del XML antes de que deje su computadora.

- Cuando recibe un paquete de SOAP, podría buscar los contenidos para ver si los recursos adicionales se demandarán para procesar la petición y en ese caso, decide esos recursos por adelantado.

Para crear una SoapExtension, derive una nueva clase de SoapExtension y revoque los métodos virtuales necesarios. Para captar la extensión en sus métodos Web, puede crear también un atributo personalizado basado en SoapExtensionAttribute.

Cuando el atributo de extensión personalizada es agregado al método del servicio Web o a una clase cliente proxy, el servicio Web invoca la extensión asociada en tiempo apropiado. El atributo de extensión es una clase de atributo personalizado derivando de la clase SoapExtensionAttribute.

Otro método importante de extensiones SOAP es ProcessMessage, utililizado en las etapas de definición de enumeración SoapMessageStage.

En los ejercicios prácticos para esta sección trataremos algunos ejemplos con extensiones SOAP.

3.2.2. Control de formato y llamadas asíncronas

Control de formato SOAP

El .NET tiene herramientas para modificar la representación de SOAP en la red. Proporciona éstas para que pueda personalizar el SOAP por cualquier número de razones, la más importante para la interoperabilidad. El mundo de servicios Web está encontrando que los estándares generalmente se están solidificando, pero algunas cosas todavía están en flujo. A veces podría necesitar entallar SOAP para encajar un cierto molde, como cuando quiere crear un Servicio Web que se encuentra preexistiendo WSDL. Otras veces, podría querer crear SOAP que especifique los clientes existentes esperando, quizás minimizar los efectos laterales de migración de .NET.

Un par de mecanismos primarios existen para modificar el flujo de SOAP. Por ejemplo, podría agregar el SoapMethodAttribute o el SoapRpcMethodAttribute relacionados para modificar el flujo de SOAP explícitamente. O, podría emplear técnicas de formación de datos para cambiar la representación en red implícitamente.

Control de formato XML

Hasta ahora se ha tratado el formato de SOAP de acuerdo al estándar de SOAP. Pero, de hecho, el estándar permite un buen trato de flexibilidad convirtiendo los

objetos a los mensajes del XML. Hay variaciones dando formato ambos para los parámetros dentro del cuerpo de un mensaje de SOAP y para el formato del propio cuerpo:

- **Formateo de parámetro literal**. Da formato a los parámetros de acuerdo a un esquema de XSD. Éste es el parámetro predeterminado que da formato para los servicios Web de .NET.

- **Formateo de parámetro codificado**. Da formato a los parámetros de a cuerdo a las denominadas reglas de codificación Sección 5 SOAP. Estas reglas codifican la información del tipo de datos directamente en el mensaje de SOAP.

- **Formateo del cuerpo basado en documento.** Es el predeterminado para formateo de cuerpo en .NET. En este método de dar formato al cuerpo, bs parámetros pueden irrumpirse en elementos múltiples dentro del cuerpo del mensaje. Pueden envolverse elementos dentro de un elemento total o pueden ponerse en el cuerpo como elementos desprovistos.

- **Formateo del cuerpo basado en RPC.** A veces está conocido como formateo Sección 7 SOAP. En este estilo de dar formato al cuerpo, todos los parámetros se contienen dentro de un solo elemento en el cuerpo del mensaje de SOAP.

En los ejercicios prácticos para esta sección trataremos algunos ejemplos de control de formato.

Métodos Web asíncronos

Si llama simplemente un método de servicio Web desde un programa cliente (vía un proxy de servicio Web), la llamada es síncrona. Es decir, su aplicación del cliente no procesará ningún otro comando hasta que la respuesta de SOAP regrese del servidor.

Las clases del proxy también generadas por Visual Estudio Visual .NET (o por wsdl.exe) le permiten que haga llamadas asincrónicas a un servicio Web. Cuando hace una llamada asincrónica, el cliente envía la petición de SOAP, pero no se bloquea mientras esperando por la respuesta de SOAP. El servidor de servicio Web no exige a ninguna configuración especial soportar las llamadas asincrónicas. De hecho, hasta donde el servidor sabe, la llamada es síncrona. Todo el procesamiento adicional se hace en el cliente. El CLR envía la petición de SOAP y entonces los monitores del puerto de realización de E/S del sistema esperan por la respuesta de SOAP.

En los ejercicios prácticos para esta sección trataremos algunos ejemplos de llamadas asíncronas.

3.2.3. Actividades de aprendizaje

Ejercicios sobre implementaciones SOAP

Instrucciones

1. Realizar el ejercicio “3.3.1 Reglas y medios de serialización”

2. Realizar el ejercicio “3.3.2 Uso de extensiones SOAP”

3. Realizar el ejercicio “3.3.3 Control de formato de flujo y cuerpo”

4. Realizar el ejercicio “3.3.4 Invocación o llamada asíncrona”

5. Con base a los cuatro ejercicios previos, sobre la aplicación elaborada en la sección “Actividades de aprendizaje 1.6”, realizar las tareas siguientes:

- Represente un mecanismo de serialización.
- Utilice por lo menos un manejo de extensión SOAP.
- Implemente por lo menos un manejo del flujo SOAP y formateo del cuerpo.
- Realice la aplicación para que sea invocada asíncronamente.

Coloque el resultado de estas tareas en “Tarea Individual Ejercicio (3b)”

3.3. Ejercicios prácticos

Ejercicio 1.

SOAP define un conjunto de reglas de serialización para codificación de tipos de datos en XML. Con objeto de que revise estas reglas, represente en código C# los tipos de datos siguientes, tomando como base el “Código reglas de serialización SOAP” del anexo C:

A. Simples estructuras de datos.
B. Estructuras de datos con múltiples referencias.
C. Referencias null estructuras de datos complejas.
D. Datos dinámicamente tipificados.
E. Arreglos.
F. Arreglos multidimensionales.

G. Transmisión parcial de arreglos.
H. Arreglos esparcidos.
I. Arreglos de arreglos.

Ejercicio 2.

Implemente el "Código Serialización XML y SOAP en .NET" proporcionado en el anexo C, para que compare las representaciones de serialización de la infraestructura .NET siguientes:

A. Serialización XML.
B. Serialización Runtime.
C. Serialización SOAP.

3.4. Cuestionarios

Tareas asignadas por el profesor

4. Descripción de Servicios Web: WSDL

WSDL (Web Services Description Language) es una especificación para describir y publicar los formatos y protocolos de un servicio Web de manera estandarizada, usualmente basada en gramática XML. WSDL fue desarrollado por Microsoft, Ariba e IBM y fue aceptada la v1.1 de la especificación WSDL por el W3C y publicada en su sitio Web.

WSDL describe todas las operaciones que permitan la interacción de un cliente con el servicio Web:

- Parámetros de entrada y salida junto con los tipos asociados. Para invocar cada operación, el cliente necesita conocer los parámetros esperados por cada operación de la entrada y la salida esperada de cada operación.

- Ligación e información de la dirección para cada servicio del Web. Para permitir el acoplamiento suelto entre un solicitante y un servicio del Web, WSDL específica donde el servicio puede encontrarse y el protocolo de transporte que debe usarse para invocar el servicio (pueden ser múltiples protocolos).

En esencia, WSDL define un contrato que compromete al proveedor soportar la separación de la aplicación de la interfaz, WSDL no especifica como cada servicio del Web se implementa. WSDL proporciona el formato común para codificar y

descifrar mensajes a y de virtualmente de cualquier aplicación back-end, como CORBA, COM, EJB, JMS, MQ Series, sistemas ERP y así sucesivamente. Las aplicaciones detrás de los servicios Web pueden ser cualquier cosa.

4.1. Comprensión de WSDL

En la capa de integración estandar para la infraestructura de servicios Web, reconocida universalmente y con soporte de interfaz programática, WSDL habilita la comunicación entre estas capas proporcionando descripciones estandarizadas en los puntos terminales (ver figura 1).

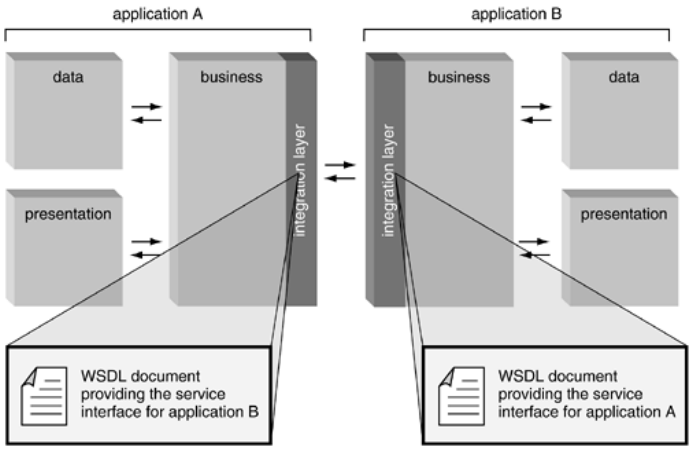


Figura 1. WSDL representando los servicios Web para las aplicaciones.

Los elementos que forman parte de WSDL contienen una descripción de los datos, típicamente usando uno o más esquemas XML, pasados al servicio Web para que el remitente y el destinatario entiendan los datos a intercambiar.

Los elementos de WSDL también contienen una descripción de las operaciones a ser ejecutadas en los datos, para que el destinatario de un mensaje sepa procesarlas y una ligación (binding) a un protocolo o transporte, para que el remitente sepa enviarlos. Básicamente, WSDL se usa con SOAP y la especificación de WSDL incluye una ligación de SOAP.

Análogo a SOAP y las otras tecnologías de infraestructura de integración de XML, WSDL es una infraestructura extensible. Por ejemplo, las ligaciones para SOAP se extienden a WSDL, como se realizan las ligaciones para HTTP GET y POST y MIME.

De esta manera, WSDL separa la definición abstracta de puntos terminales y mensajes de sus despliegues concretos de la red o ligaciones de formato de datos, que permite reutilizar de las definiciones abstractas. WSDL es completamente extensible a los formatos múltiples y protocolos.

WSDL tiene siete partes, pero se contienen dentro de tres elementos principales que pueden desarrollarse como documentos separados y, entonces, pueden combinarse o pueden reutilizarse para formar los archivos WSDL completos. Los elementos principales son divididos según su nivel de abstracción en la pila que representa un servicio Web.

Las partes o subelementos que conforman WSDL son los siguientes:

1. **Tipos de datos (types)**: los tipos de datos en la forma de esquemas XML o posiblemente algún otro mecanismo usado en los mensajes.

2. **Mensaje (message)**: una definición abstracta de los datos, en la forma de un mensaje o presentado como un documento entero o como argumentos con mapeo a una invocación del método.

3. **Operación (operation)**: la definición abstracta de la operación para un mensaje, como nombrar un método, cola de mensajes o el proceso de negocio que aceptarán y procesarán el mensaje.

4. **Tipo de puerto (portType)**: un conjunto abstracto de operaciones con mapeo a uno o más puntos finales, definiendo la colección de operaciones para una ligación; la colección de operaciones, porque es abstracta, pueden mapearse a transportes múltiples a través de las varias ligaciones.

5. **Ligación (binding)**: el protocolo concreto y disposiciones de los datos para las operaciones y mensajes definidos para un tipo de puerto particular.

6. **Puerto (port)**: una combinación de una ligación y una dirección de la red, proporcionando la dirección de destino de la comunicación de servicio.

7. **Servicio (service)**: una colección de puntos finales relacionados que abarcan las definiciones de servicio en el archivo; los servicios mapean la ligación al puerto e incluyen cualquier definición de extensibilidad.

Los tres elementos principales de WSDL que puede definirse separadamente son:

1. **Tipos**
2. **Operaciones**
3. **Ligación**

Las declaraciones de tipo de datos son el elemento más abstracto de un servicio, definiendo los elementos de datos de documentos XML, para ser intercambiados

por un servicio Web. Los tipos de datos pueden definirse una vez, incluidos en archivos y referenciados dentro de cualquiera de las operaciones.

Las operaciones en los tipos de datos representan el siguiente nivel de abstracción, definiendo la forma en que los datos se pasan de un lado a otro.

La ligación, el nivel final de abstracción, define que transporte es usado para pasar el mensaje.

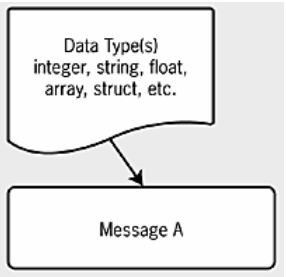
4.1.1. Definición de tipos de datos del mensaje

Un servicio Web necesita definir sus entradas y salidas y cómo será el mapeo dentro y fuera de los servicios. Los tipos WSDL cuidan esto. Los tipos son documentos XML o partes del documento. WSDL permite definir los tipos en elementos separados para que los tipos sean reutilizables con servicios Web múltiples.

Los tipos son comúnmente definidos usando los esquemas XML; como otras partes de WSDL, sin embargo, la porción de los tipos es completamente extensible y otro tipo de sistemas pueden usarse en cambio. Por ejemplo, si saben que el destino de un caso particular de un servicio Web es un objeto del CORBA, pueden usar el tipo de sistema de CORBA en lugar del sistema de tipo de esquema de XML.

Un servicio Web incluye una fase de mapeo durante la cual los datos del XML son establecidos al programa del software, así como una fase de ejecución durante la que el propio programa se ejecuta.

Con el mapeo, uno o más tipos de datos individuales pueden establecerse en los mensajes y el mismo mensaje puede establecerse en múltiples operaciones. Los tipos son propiamente cualquiera de los tipos de datos soportados en los esquemas XML, tal como enteros, cadenas, booleanos, fechas e inclusive tipos complejos, como estructuras y arreglos, incluyendo aquéllos definidos para SOAP.



Los tipos de datos son por lo tanto cualquier tipo de esquema simple o esquemas que definen los tipos complejos. Como en las otras áreas de WSDL, los tipos no son restringidos a los esquemas XML, porque nadie espera un tipo único de sistema para ser capaz de describir todos los posibles formatos de mensaje para el Web.

El ejemplo siguiente ilustra el tipo y definiciones del mensaje para un servicio de orden de compra que devuelve el valor total de uno o más órdenes de compra. Los tipos de datos del esquema XML usados en el archivo WSDL son mapeados para mensajes usando los nombres de elemento del esquema.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="PurchaseOrderService"
    targetNamespace="PurchaseOrderService"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="PurchaseOrderService"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd1="PurchaseOrderService-xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <types>
        <schema targetNamespace="PurchaseOrderService-xsd"
             xmlns="http://www.w3.org/2001/XMLSchema"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
            <complexType name="PurchaseOrder">
                <all>
                    <element name="CompanyName" type="xsd:string"/>
                    <element name="Items" type="xsd1:ArrayOfItem"/>
                    <element name="Address" type="xsd1:Address"/>
                </all>
            </complexType>
            <complexType name="Item">
                <all>
                    <element name="Price" type="xsd:float"/>
                    <element name="PartID" type="xsd:string"/>
                    <element name="Description" type="xsd:string"/>
                    <element name="Quantity" type="xsd:int"/>
                </all>
            </complexType>
            <complexType name="ArrayOfItem">
                <complexContent>
                    <restriction base="SOAP-ENC:Array">
                        <attribute ref="SOAP-ENC:arrayType"
                              wsdl:arrayType="xsd1:Item[]"/>
                    </restriction>
                </complexContent>
            </complexType>
            <complexType name="Address">
                <all>
                    <element name="State" type="xsd:string"/>
```

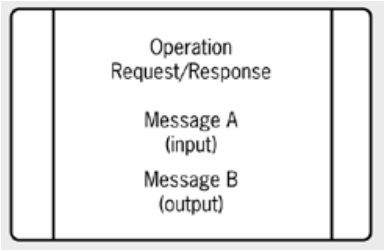
```
                <element name="PostalCode" type="xsd:string"/>
                <element name="City" type="xsd:string"/>
                <element name="Line2" type="xsd:string"/>
                <element name="Country" type="xsd:string"/>
                <element name="Line1" type="xsd:string"/>
            </all>
        </complexType>
        <complexType name="ArrayOfPurchaseOrder">
            <complexContent>
                <restriction base="SOAP-ENC:Array">
                    <attribute ref="SOAP-ENC:arrayType"
                        wsdl:arrayType="xsd1:PurchaseOrder[]"/>
                </restriction>
            </complexContent>
        </complexType>
    </schema>
</types>
<message name="postPurchaseOrderRequest">
    <part name="order" type="xsd1:PurchaseOrder"/>
</message>
<message name="postPurchaseOrderResult">
    <part name="return" type="xsd:float"/>
</message>
<message name="postPurchaseOrdersRequest">
    <part name="orders" type="xsd1:ArrayOfPurchaseOrder"/>
</message>
<message name="postPurchaseOrdersResult">
    <part name="return" type="xsd:float"/>
</message>
```

Al usar SOAP, un mensaje se lleva como la carga útil de la petición o respuesta de SOAP. Es decir, la definición de mensaje WSDL no incluye cualquier información que sea mapeada a la envoltura, encabezado o código de falla de SOAP. En otros términos, pueden decir que WSDL apunta una capa de abstracción completamente sobre el de SOAP.

4.1.2. Definición de operaciones sobre mensajes

El siguiente nivel de abstracción, las operaciones, dirige el requisito de un servicio Web para identificar el tipo de operaciones que se realizan a favor de un mensaje dado o conjunto de mensajes. La operación está definida para que el servicio Web sepa como interpretar los datos y los datos que serán devueltos en la respuesta.

Las operaciones están definidas en correspondencia al los patrones comunes del mensaje, como unidireccional y petición/respuesta. WSDL no incluye las definiciones específicas para otras operaciones, pero las interacciones más complicadas pueden ser construidas combinando estos tipos básicos.



WSDL tiene cuatro tipos de operaciones:

- **Unidireccional**: El mensaje se envía sin un requisito para devolver una respuesta.

- **Petición/respuesta**: Similar a una interacción estilo RPC; el remitente envía un mensaje y el destinatario envía una respuesta correspondiente (algunos protocolos no pueden garantizar que una respuesta se devuelve para cada petición).

- **Solicita respuesta**: Una petición simple para una respuesta sin el dato de entrada. Es una petición para conseguir un mensaje y no incluye el envío del mensaje, en el sentido de un mensaje de WSDL consistente en uno o más tipos definidos. Es la marcha atrás de la operación unidireccional.

- **Notificación**: Este tipo de operación define a múltiples destinatarios para un mensaje, similar a una difusión y a menudo incluye un mecanismo de suscripción, como en la preparación de publicar/suscribir.

Las operaciones permiten poner en correspondencia las secuencias de mensajes en los patrones específicos sin tener que introducir una especificación de flujo más compleja. Las operaciones no son lo mismo que los métodos en objetos, aunque ciertamente los parámetros de la entrada y salida definidos para las operaciones normalmente mapearán a los argumentos de entrada y salida del método cuando los servicios que son implementados usan una tecnología basada en objetos como .NET, EJB o CORBA.

El ejemplo siguiente ilustra una definición de la operación de petición/respuesta para el servicio de orden de compra:

```
<portType name="PurchaseOrderPortType">
    <operation name="postPurchaseOrder">
        <input message="tns:postPurchaseOrderRequest"
            name="postPurchaseOrder"/>
        <output message="tns:postPurchaseOrderResult"
            name="postPurchaseOrderResult"/>
    </operation>
```

```
    <operation name="postPurchaseOrders">
        <input message="tns:postPurchaseOrdersRequest"
            name="postPurchaseOrders"/>
        <output message="tns:postPurchaseOrdersResult"
            name="postPurchaseOrdersResult"/>
    </operation>
</portType>
```

Los mensajes de la entrada y salida están definidos para la petición y operaciones de la respuesta, usando las definiciones del mensaje creadas en los elementos previamente mostrados del archivo de WSDL.

Operaciones de petición/respuesta no requieren uso del atributo de RPC en la ligación de SOAP, aunque es una buena práctica de programación preservarlo dentro la ligación de SOAP, por ejemplo, la firma de un objeto al que el servicio Web es mapeado. Los nombres de la parte del listado del atributo parameterOrder del mensaje y el orden de los mensajes debe emparejar aquéllos de la firma de RPC.

No hay ninguna sintaxis para definir un valor devuelto, pero si una parte que el nombre aparece en la entrada y los mensajes de salida, es un parámetro in/out; si sólo está en entrada, es un parámetro de entrada. Esta convención le hace más fácil mapear las operaciones de WSDL hacia ligaciones RPC, por ejemplo la ligación RPC para SOAP.

SOAP también tiene una ligación del documento y las interacciones de servicios Web incluirán ambos probablemente. Por ejemplo, una orden de compra es compartida entre el comprador y el vendedor. El comprador y el vendedor pueden primero intercambiar una copia del mismo documento. Pueden intercambiar la información dinámicamente sobre el Web para rellenar los campos en la forma, como inventario disponible, fecha de envío, descuento por cantidad, y así sucesivamente. Esto permitirá una negociación más dinámica, de tiempo real entre el comprador y vendedor estableciendo los términos y condición de una venta, basado en un documento compartido.

Las operaciones ponen los mensajes de entrada/salida en correspondencia, aunque varía por el transporte sobre que tipo de correspondencia garantizada está disponible; SOAP no requiere la correspondencia, aunque HTTP GET y POST lo hacen. En la actualidad, los servicios separados tienen que ser definidos si quieren anunciar ambos un servicio orientado a documento y un servicio orientado a procedimiento, pero probablemente parece que éstos pueden combinarse en una versión futura de WSDL.

Más adelante trataremos el tercer elemento principal, la ligación, y su relación con las demás partes de WSDL.

4.1.3. Actividades de aprendizaje

Análisis y aspecto de un documento WSDL

Instrucciones

1. En base al Material de Apoyo “4.Descripción de Servicios Web: WSDL” analice el documento WSDLTester que se proporciona en el rubro “Código ejercicios WSDL” del anexo C. para realizar la tarea siguiente:

- El documento WSDLTester describe un servicio, que contiene un método TestMethod1. Este método acepta como sus argumentos un entero nombrado iNum1 y un Booleano nombrado fBool1 y devuelve una cadena.
- Identifique los bloques o líneas de código de los varios elementos siguientes y cuál es la funcionalidad de ellos:
    - Types.
    - Messages.
    - PortTypes.
    - Bindings. S
    - Services.

2. Implementar el programa WsdlViewer para que visualice el programa WSDLTester del inciso anterior.

- El programa WsdlViewer se proporciona en el rubro “Código ejercicios WSDL” del anexo C.

3. Coloque los resultados de los incisos 1 y 2 previos en “Tarea Individual Ejercicio (4a)”

4. Contestar el cuestionario “Preguntas comprensión de WSDL”

- Coloque sus respuestas en “Tarea Individual Cuestionario (4a)”.

4.2. Mapeo de mensajes a protocolos

Después de que los tipos de datos y los tipos de operación están definidos, tienen que ser mapeados hacia un protocolo de transporte específico y un punto terminal o dirigir para qué datos se envía y para cual es posible encontrar e invocar la operación particular, debe identificarse. Esta operación se requiere porque pueden mapearse los mismos tipos de datos y operaciones hacia transportes múltiples, como SOAP, BEEP, IIOP, JMS, y otros y un servicio Web pueda coexistir a las direcciones potencialmente múltiples de punto final.

Esta parte del WSDL resuelve el problema de cómo un transporte espera entender los datos inicialmente pasados, por ejemplo pueden ser serializados según la especificación SOAP y dónde encontrar el servicio, en una dirección IP de Internet, intranet, LAN, y así sucesivamente.

Primero, se agrupan las operaciones en los tipos de puerto. Entonces cada tipo de puerto es mapeado a uno o más puertos específicos que representan varios transportes sobre de los que un servicio podría estar disponible; por ejemplo, un tipo de puerto podría mapearse a los puertos específicos para HTTP GET y POST, SOAP o MIME. Las extensiones de ligación de transporte entonces son mapeadas a cada puerto específico para definir la información necesaria para ofrecer un servicio dado sobre de un transporte específico.

Las extensiones de ligación de transporte debajo de los tipos de datos, el tipo de operación y los tipos del puerto identifican al destinatario de los datos y la operación a ser realizada. Así para cualquier servicio Web dado, es necesario y posible definir los datos, la operación en los datos y el lugar y cómo se envían los datos.

4.2.1. Tipos de puertos y servicios

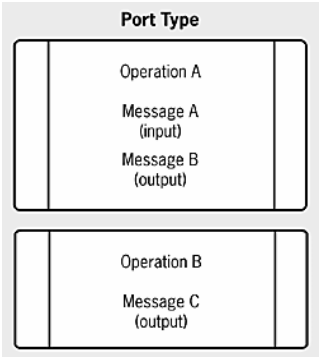
Un tipo de puerto es una agrupación lógica de operaciones, similar a los tipos de bibliotecas en .NET, clases en Java, o el IDL de un objeto (lenguaje de definición de interfaces) en CORBA. Si una operación es análoga a un método en un objeto y si los mensajes son análogos a los argumentos de la entrada y salida, un tipo de puerto es análogo a una definición del objeto que potencialmente contiene los métodos múltiples. Pero estas analogías no son exactas, porque WSDL es extensible y se piensa que proporciona un nivel de abstracción mayor que lo que es proporcionado por cualquiera de estos sistemas basados en objetos.

En otros términos, WSDL es una abstracción de datos independiente que puede usarse para mucho más que el mapeo dentro y fuera de .NET, EJB u objetos del CORBA. Pero al trabajar con estos sistemas basados en objetos, ayuda a entender las partes de WSDL que corresponde a las partes de estos sistemas.

El estilo de petición/respuesta usa las definiciones del mensaje diferentes para los mensajes de la entrada y salida; como con “paso del documento”, el estilo unidireccional usa un solo tipo como un documento completo. Ambos tipos de interacciones puede definirse dentro de un tipo de puerto dado.

Un tipo de puerto representa una colección de operaciones, del mismo modo que un objeto representa una colección de métodos. Para los servicios Web, el mapeo entre un tipo de puerto y un tipo de objeto o una clase es bastante natural. Pero

debido a que los servicios Web están definidos en un nivel alto de abstracción, también pueden hacerse mapeos a los documentos y las tecnologías orientadas a procedimiento.



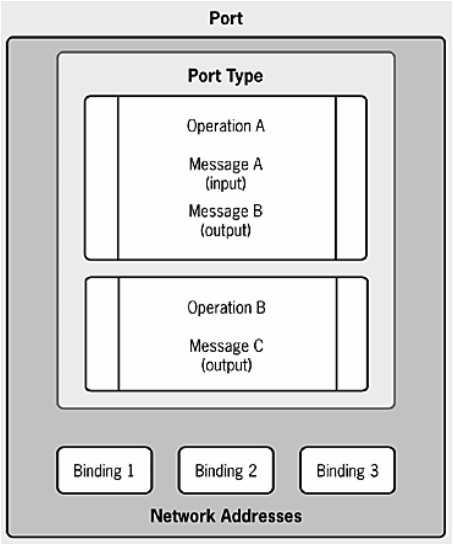
Puertos

Un puerto se usa para exponer un conjunto de operaciones o tipos de puerto, sobre un transporte dado. Los tipos de puerto que pueden agruparse para una o más ligaciones, las cuales son cómo los servicios se conectan sobre la red. Un puerto identifica uno o más ligaciones de transporte para un tipo de puerto dado. Para continuar la comparación con sistemas basados en objetos, un puerto es análogo al transporte. Por ejemplo, un transporte común para CORBA es IIOP; Para EJB, es RMI o RMI/IIOP; y para COM, es DCOM que es basado en DCE RPC.

Estos sistemas no proporcionan un mecanismo de la definición verdaderamente equivalente con que definir un transporte particular, aunque EJB y sistemas de CORBA pueden ser ciertamente y se han mapeado a una variedad de transportes. Pero el mapeo no es considerado parte de la definición del objeto.

Con WSDL, sin embargo, es necesario para anunciar dentro de la definición de servicio que las ligaciones de transporte están disponibles para una colección dada de operaciones. El sistema enviando o descubre a un punto final de la red remoto tendrá acceso a un archivo de WSDL publicado típicamente vía HTTP como una operación típica de documento GET pero puede desear negociar con el destinatario o publicador del servicio Web para interactuar en un transporte diferente que proporcione la funcionalidad adicional.

Entonces, un puerto en WSDL reúne las operaciones, la ligación y un URL que definen una dirección IP específica a la que la aplicación del servicio Web pueda encontrarse. El ejemplo siguiente ilustra la ligación SOAP para las operaciones para la orden de compra:




```
<binding name="PurchaseOrderBinding" type="tns:PurchaseOrderPortType">
    <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http/"/>
    <operation name="postPurchaseOrder">
        <soap:operation
            soapAction="PurchaseOrderService/postPurchaseOrder"
                style="rpc"/>
        <input name="postPurchaseOrder">
            <soap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/
                encoding/"namespace="PurchaseOrderService"
                use="encoded"/>
        </input>
        <output name="postPurchaseOrderResult">
            <soap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/
                encoding/"namespace="PurchaseOrderService"
                use="encoded"/>
            </output>
    </operation>
    <operation name="postPurchaseOrders">
        <soap:operation
            soapAction="PurchaseOrderService/postPurchaseOrders"
                style="rpc"/>
        <input name="postPurchaseOrders">
            <soap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/
                encoding/"namespace="PurchaseOrderService"
                use="encoded"/>
        </input>
        <output name="postPurchaseOrdersResult">
            <soap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/
```

```
                encoding/"namespace="PurchaseOrderService"
                use="encoded"/>
        </output>
    </operation>
</binding>
```

Noten el uso de Acción SOAP, codificación SOAP y estilo de interacción RPC.

La ligación de transporte puede hacerse por operación. Los puertos por definición incluyen la ligación de transporte o ligaciones que en el caso de la ligación de SOAP incluyen una declaración de si la interacción es petición/respuesta (RPC) o paso del documento (DOCUMENT). Éste es el estilo de ligación SOAP. Varias otras extensiones para WSDL están específicamente definidas para el uso con SOAP, como una manera de definir Acción SOAP y mensajes de la entrada y salida.

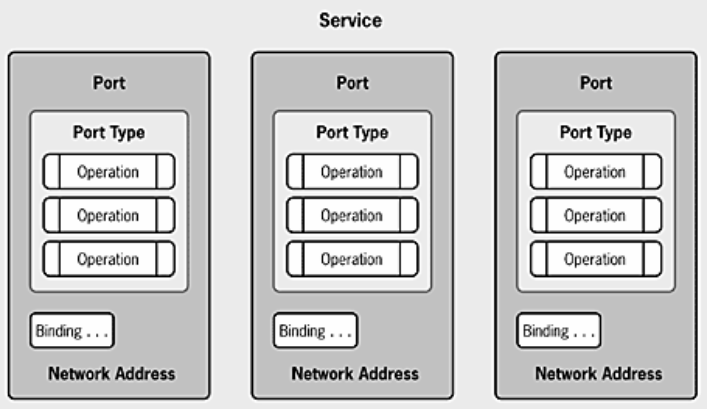
Codificación SOAP puede usarse opcionalmente con WSDL, como en el ejemplo siguiente y ambas opciones son explícitamente soportadas.

```
<soap:operation
    soapAction=http://www.iona.com/GetLastTradePrice"/>

    <input>
    <soap:body use="literal"
        namespace="http://www.iona.com/stockquotes.xsd"/>
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>

    <output>
    <soap:body use="literal"
        namespace="http://www.iona.com/stockquotes.xsd/>
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>

</operation>
```

Pueden definirse ligaciones para otros transportes, como SMTP, y las extensiones incluidas específicamente para ellos. La separación de las extensiones de ligación de transporte de la definición del tipo de puerto permite un servicio Web para estar disponible sobre transportes múltiples sin tener que redefinir el archivo de WSDL entero. De nuevo, porque se diseña para ser completamente extensible, WSDL permite usar otras extensiones de ligación, como por ejemplo para IIOP, .NET, JMS, Series de MQ, y así sucesivamente.

Servicios

La parte de servicio de WSDL adjunta uno o más tipos de puerto, similar a la manera en la que una clase de objeto puede contener los objetos múltiples. El ejemplo siguiente ilustra el servicio que liga para el orden de compra que integra el servicio:




```
<service name="PurchaseOrderService">
    <port binding="tns:PurchaseOrderBinding"
        name="PurchaseOrderPort">
        <soap:address
            location="http://localhost:9000/
            xmlbus/container/PurchaseOrder/
            PurchaseOrderService/
            PurchaseOrderPort"/>
    </port>
</service>
```

Noten la definición de la dirección de servicio (en este caso una dirección organizada localmente).

El servicio permite un punto final dado en una aplicación remota decidir exponer múltiples categorías de operaciones para varios tipos de interacciones. Por ejemplo, una categoría podría contener un conjunto de interacciones orientadas a documento para intercambiar asincrónicamente y completar un orden de compra para un envío futuro. Otra categoría podría contener un conjunto de interacciones orientadas a RPC para interactuar síncronamente en una orden para el envío inmediato. En el primer caso, el acceso a los datos del inventario en tiempo real no se requiere; pero en el segundo caso si está requerido.

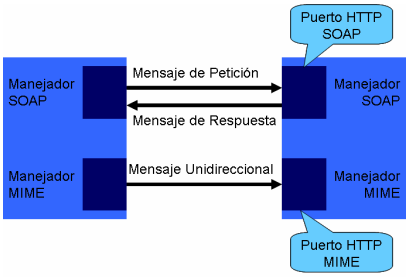
4.2.2. Reunión de todos los elementos

Una vez que todas las partes y elementos de WSDL están definidos, el archivo de WSDL está completo y puede ponerse en un directorio accesible al Web vía un URL. Se generan archivos de WSDL típicamente, para que no se preocupe demasiado por su complejidad individual; el punto principal es entender cómo WSDL resuelve los problemas particulares.

La figura siguiente ilustra los dos tipos de operaciones o patrones de uso, para servicios Web definidos en la especificación de WSDL:

- Mensajes petición/respuesta
- Mensaje unidireccional

Para la operación de petición/respuesta, un mensaje de entrada se envía al manejador receptor SOAP como parte de una petición de HTTP y un mensaje de salida se devuelve vía la respuesta de HTTP. Para la operación unidireccional, un mensaje de entrada se envía a un manejador de MIME en un puerto diferente identificado por el mismo servicio.



La figura representa que WSDL define mensajes, operaciones, puertos y transportes para SOAP y MIME.

Importación de elementos WSDL y espacios de nombres

Otro elemento import permite elementos de WSDL que estaban separado en documentos independientes sean importados, como sea requerido, para crear un documento completo. Espacios de nombres diferentes se usan para calificar los nombres en los tres elementos principales.

Como mencionamos previamente, pueden desarrollarse tipos, operaciones abstractas y ligaciones independientemente y pueden combinarse después para crear el archivo de WSDL completo usado para describir un caso de servicio Web particular. Así WSDL permite combinar los tipos de datos diferentes con operaciones diferentes y las ligaciones diferentes. Los espacios de nombres están

definidos para cada elemento y la parte sutil es asegurar que los espacios de nombres no se solapen.

WSDL hace uso extensivo de espacios de nombre XML.

Extensiones para ligación a SOAP

Debido a que SOAP es el más claro transporte de WSDL, este utiliza extensiones de ligación SOAP. La especificación de SOAP contiene las reglas predefinidas por representar los tipos de datos físicamente como booleanos, enteros y arreglos. Ligación para SOAP por consiguiente exige la ligación de los tipos abstractos de datos, mensajes y operaciones para solidificar las representaciones físicas en la red. El elemento <binding> de WSDL asocia un tipo de puerto con un puerto para definir los aspectos concretos de operaciones. Por ejemplo:

```
  <binding name='InSeasonOrderBinding'
      type='wsdlns:InSeasonOrderPort' >
....
  </binding>
```

El nombrado atributo de ligación también se requiere para el elemento WSDL <port>. Dentro del elemento <binding> esta un elemento extensión SOAP WSDL llamado <soap:binding > que especifica el protocolo de transporte físico y el estilo de petición: RPC o documento. Por ejemplo:

```
<soap:binding style='rpc'
    transport='http://schemas.xmlsoap.org/soap/
    http' />
```

Este ejemplo define que el estilo de interacción RPC para un SOAP mapeando a HTTP. Para cada operación dentro del servicio, el valor de SOAPAction del encabezado http tiene que ser incluido. SOAPAction es un encabezado HTTP que el sistema solicitando envía cuando invoca el servicio. El procesador de SOAP en el sistema receptor puede usar el encabezado para determinar el destino definitivo del servicio. Por ejemplo:

```
<binding name='InSeasonOrderBinding'
    type='wsdlns:InSeasonOrderPort' >
    <soap:binding style='rpc'
        transport='http://schemas.xmlsoap.org/
        soap/http' />
    <operation name='OrderEntry' >
      <soap:operation
          soapAction='http://www.skateboots.com/
          order/OrderManagement.OrderEntry' />
              ....
```

```
    </operation>
</binding>
```

El elemento <operation> contiene el mismo nombre como la operación definida antes. Un <soap:operation> con el atributo soapAction es incluido dentro de este <operation> para mostrar que el destino definitivo del mensaje es el servicio OrderEntry reparan en la carpeta /order.

Luego, el mecanismo de la codificación para la entrada y los mensajes de la salida deben especificarse, como se muestra en seguida.

```
<binding name='InSeasonOrderBinding'
type='wsdlns:InSeasonOrderPort' >
    <soap:binding style='rpc'
        transport='http://schemas.xmlsoap.org/soap/http' />
    <operation name='OrderEntry' >
        <soap:operation
            soapAction='http://www.skateboots.com/action/
            OrderManagement.OrderEntry' />
        <input>
          <soap:body use='encoded'
              namespace='http://www.skateboots.com/typesIn/'
              encodingStyle='http://schemas.xmlsoap.org/soap/
              encoding/' />
        </input>
        <output>
          <soap:body use='encoded'
              namespace='http://www.skateboots.com/typesOut/'
              encodingStyle='http://schemas.xmlsoap.org/soap/
              encoding/' />
        </output>
    </operation>
</binding>
```

Ambos los elementos <input> y <output> usan un elemento <soap:body> para especificar reglas de codificación de tipo de datos. El URI hace referencia el estilo de codificación de SOAP optativo, significando que es usado para la serialización, o codificación de tipo de datos, de los mensajes de entrada y salida.

4.2.3. Actividades de aprendizaje

Ejercicios sobre manejo de WSDL

Instrucciones

1. Realizar el ejercicio “4.3 Ejercicio manejo y uso de documentos WSDL”
2. En base al ejercicio previo, sobre la aplicación elaborada en la sección “Actividades de Aprendizaje 1.6”, realice las tareas siguientes:

- Implemente los documentos WSDL: único, con esquema XSD y con elemento Imports.
- Implemente la interfaz única y múltiple.

Coloque el resultado de estas tareas en “Tarea Individual Ejercicio (4b)”

3. Contestar el cuestionario “Preguntas mapeo de mensajes a protocolos”

- Coloque sus respuestas en “Tarea Individual Cuestionario (4b)”

4.3. Ejercicios prácticos

En base a las referencias siguientes:

- Presentación “El rol de los mensajes XML y esquemas XSD” del anexo B.
- Presentación “El rol del archivo de clase de definición de interfaz” del anexo B.
- Material de Apoyo “4.Descripción de Servicios Web: WSDL”.

1. Analice y compile los programas del proyecto StockTrader que se proporcionan en “Código modos de documentos WSDL” del anexo C. Ellos ilustran documentos WSDL con XSD, WSDL único y WSDL con Imports.

2. Analice y compile los programas del proyecto LinkAdminInterface que se proporcionan en “Código implementación de interfaz WSDL” del anexo C.

4.4. Cuestionarios

Tarea Asignada por el profesor.

5. Localización de Servicios Web: UDDI

UDDI (Universal Description, Discovery, and Integration) es una especificación técnica para describir, descubrir e integrar servicios Web. UDDI es en consecuencia una parte crítica de la pila protocolar del servicio Web, habilitando a las compañías tanto publicar como encontrar servicios Web. Originalmente emitida por Microsoft, IBM y Ariba en septiembre del 2000, el consorcio de UDDI ha crecido incluyendo centenares de miembros.

Básicamente UDDI esta concebida como:

- Una especificación técnica para construir un directorio distribuido de negocios y servicios Web. Datos que son almacenados dentro de un formato de XML específico y detalles de APIs para buscar datos existentes y publicación de nuevos datos.

- Un registro de negocios que es una implementación totalmente operacional de la especificación de UDDI. El registro de UDDI permite ahora a cualquiera buscar los datos de UDDI existentes. También permite a cualquier compañía registrarse y sus servicios.

5.1. El registro UDDI

5.1.1. Comprensión del registro UDDI

Una parte clave de UDDI es la estandarización de archivos del perfil almacenado dentro del directorio, también conocido como registro. Esto permite decidir en implementaciones dependiendo de quién este en el registro.

El registro UDDI puede clasificarse en dos grupos:

- **Públicos.** Es un directorio global de descripciones de servicios de negocios internacional. Las instancias de este registro son organizadas por operadores de nodo (son grandes corporaciones como IBM y Microsoft) en una serie de servidores de UDDI dedicados. Se reproducen archivos de UDDI automáticamente entre las instancias del registro. Algunas compañías también actúan como archivistas de UDDI, permitiendo a otros agregar y editar sus perfiles de descripción de servicio Web. El registro público de negocios es complementado por varios mercados de servicio que ofrecen servicios Web genéricos para venta o renta.

- **Privados.** Son almacenes de descripciones de servicio organizados dentro de una corporación o empresa (ver figura 1). Los que autorizan el acceso a este directorio pueden incluir la selección de socios de negocio externos. Un registro restringido sólo para usuarios internos puede ser referido como un registro interno.

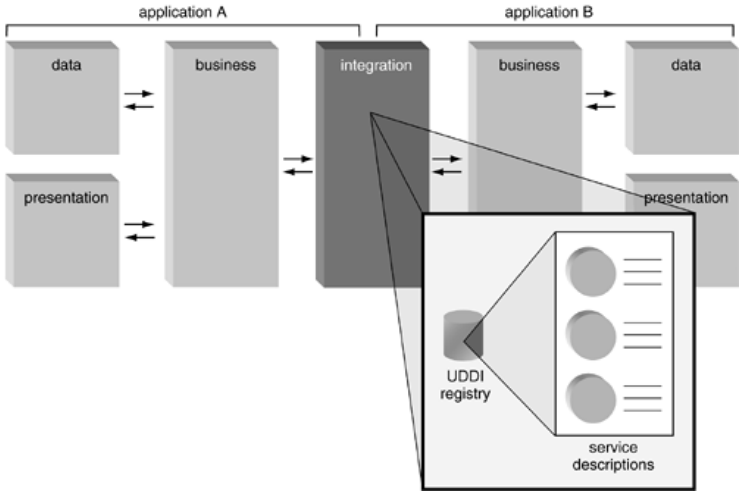


Figura 1. Descripciones de servicio centralizadas en un registro privado UDDI

Categorías UDDI

Un registro de UDDI contiene las entradas sobre los negocios, los servicios que estos negocios proporcionan, e información en cómo a esos servicios pueden tenerse acceso. El directorio de UDDI tiene tres categorías:

- **Páginas blancas.** Contienen información básica sobre un proveedor de servicios, incluso el nombre del proveedor, descripción textual del negocio información del contacto y otros identificadores exclusivos.

- **Páginas amarillas.** Incluyen la clasificación basada en taxonomías estándares del proveedor de servicio o compañía registrada.

- **Páginas Verdes.** Contienen las entradas técnicas para los servicios Web. Generalmente esto incluye un apuntador a la especificación externa y una dirección en la que invocar el servicio.

Las entradas en un directorio UDDI no se limita a servicios Web, pueden ser también para servicios basados en correo electrónico, FTP, CORBA, RMI, o incluso el teléfono.

Modelo de datos UDDI

El modelo de datos UDDI incluye un esquema XML que proporciona cuatro elementos mayores:

- **businessEntity.** Representa al propietario de los servicios e incluye el nombre del negocio, descripción, dirección, categorías de información de contacto e identificadores. En el registro, cada negocio recibe un único valor businessKey que se usa para poner en correspondencia con el servicio publicado de negocio. Las categorías e identificadores pueden usarse para especificar los detalles sobre un negocio que pueden ser útiles al realizar búsquedas.

- **businessService.** Tiene la información sobre un solo servicio Web o un grupo de relacionados, incluso el nombre, descripción, propietario (referencia cruzada con un único valor businessKey del elemento businessEntity asociado) y una lista de elementos bindingTemplate optativos. Cada servicio es identificado singularmente por el valor serviceKey.

- **bindingTemplate.** Representa un solo servicio y contiene toda la información requerida sobre cómo y dónde tener acceso al servicio (por ejemplo, el URL si es un servicio Web). Cada plantilla de ligación es identificada singularmente por un valor bindingKey. El servicio no tiene que ser un servicio del Web, puede ser basado en correo electrónico (SMTP), FTP, o incluso el fax.

- **tModel.** Acortado de "technical Model " y también conocido como el tipo de servicio, se usa principalmente para apuntar a la especificación externa del servicio inicialmente proporcionado. Para un servicio del Web, este elemento (más específicamente, el elemento hijo overviewURL) debe idealmente apuntar al documento WSDL que proporciona toda la información necesitada para describir el servicio inequívocamente y cómo invocarlo. Si dos servicios tienen el mismo valor clave tModel, entonces los servicios pueden ser considerados equivalentes. Esto permite potencialmente al solicitante cambiar de un proveedor de servicio a otro.

Estos elementos del modelo de datos estan relacionados y representan estructuras de datos con sus correspondientes elementos. La figura 2 ilustra las estructuras de datos de UDDI:

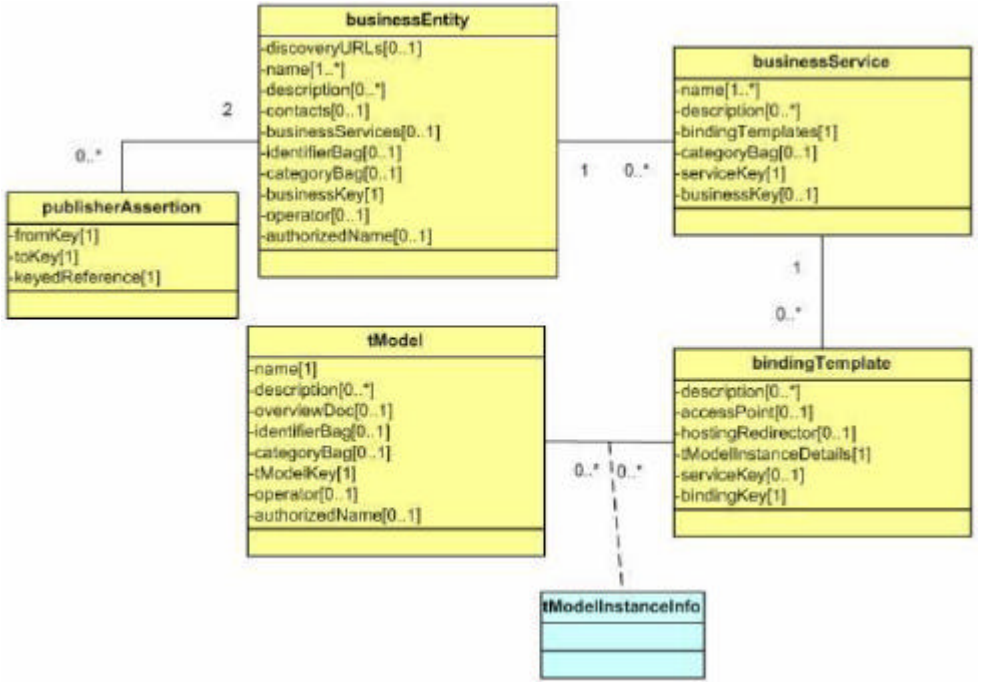


Figura 2. Estructuras, elementos y relaciones del modelo de datos UDDI

Serán referidos estos elementos y estructuras en lo sucesivo.

5.1.2. Uso del modelo de datos UDDI

Siguiendo con el modelo de datos, para las entradas del registro UDDI se pueden organizan de acuerdo a los tipos de datos primarios siguientes:

- Entidades de negocios
- Servicios de negocios
- Apuntadores de especificación
- Tipos de servicio
- Relaciones de negocios
- Suscripciones

Los datos de la entidad de negocios, son representados por el elemento businessEntity, que proporciona la información del perfil sobre el negocio registrado, incluyendo su nombre, descripción y un identificador exclusivo.

En seguida el listado 1 muestra un documento XML que contiene una estructura de businessEntity.

Listado 1. Un documento de la entidad de negocio actual es recuperada desde un registro de servicio público:

```
<businessEntity xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  businessKey="e9355d51-32ca-49cf-8eb4-1ce59afbf4a7"
  operator="Microsoft Corporation"
  authorizedName="Thomas Erl"
  xmlns="urn:uddi-org:api_v2">
  <discoveryURLs>
    <discoveryURL useType=
      "businessEntity">http://test.uddi.microsoft.com/discovery
      ?businesskey=e9355d51-32ca-49cf-8eb4-1ce59afbf4a7
    </discoveryURL>
  </discoveryURLs>
  <name xml:lang="en">
    XMLTC Consulting Inc
  </name>
  <description xml:lang="en">
    XMLTC has been building end-to-end enterprise
    eBusiness solutions for corporations and
    government agencies since 1996. We offer a
    wide range of design, development and
    integration services
  </description>
  <businessServices>
    <businessService
      serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a"
      businessKey="e9355d51-32ca-49cf-8eb4-1ce59afbf4a7">
      <name xml:lang="en-us">
        Corporate Home Page
      </name>
      <bindingTemplates>
        <bindingTemplate
          bindingKey="48b02d40-0312-4293-a7f5-4449ca190984"
          serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a">
          <description xml:lang="en">
            Entry point into the XMLTC Web site
            through which a number of resource
            sites can be accessed
          </description>
          <accessPoint URLType="http">
            http://www.xmltc.com/
          </accessPoint>
          <tModelInstanceDetails />
        </bindingTemplate>
      </bindingTemplates>
      <categoryBag>
        <keyedReference
```

```
          tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"
          keyName="Namespace" keyValue="namespace" />
      </categoryBag>
    </businessService>
  </businessServices>
</businessEntity>
```

En base a esta muestra de documento, en seguida se describen las demás estructuras individuales.

Listado 2. El elemento padre businessEntity con varios atributos:

```
<businessEntity xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  businessKey="e9355d51-32ca-49cf-8eb4-1ce59afbf4a7"
  operator="Microsoft Corporation"
  authorizedName="Thomas Erl"
  xmlns="urn:uddi-org:api_v2">
```

Cuando se registró XMLTC Consulting Inc. se dio un identificador exclusivo e9355d51-32ca-49cf-8eb4-1ce59afbf4a7 que se asignó entonces al atributo businessKey del elemento padre businessEntity. Puesto que Microsoft actuó como el operador del nodo que proporciona una instancia del registro UDDI, su nombre se despliega en el atributo del operador del elemento businessEntity.

El elemento discoveryURL identifica la dirección usada para localizar este documento XML.

Listado 3. Las estructuras discoveryURLs contienen el URL original:

```
<discoveryURLs>
  <discoveryURL useType="businessEntity">
    http://test.uddi.microsoft.com/discovery
    ?businesskey=e9355d51-32ca-49cf8eb4-1ce59afbf4a7
  </discoveryURL>
</discoveryURLs>
```

El elemento name simplemente contiene el nombre del negocio oficial.

Listado 4. El elemento name proporciona el nombre del negocio:

```
<name xml:lang="en">
  XMLTC Consulting Inc
</name>
```

Los registros del servicio de negocios representan los servicios actuales ofrecidos por el negocio registrado, estando anidados dentro de la estructura del businessEntity.

Listado 5. Estructura de businessServices:

```
<businessServices>
  <businessService
    serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a"
    businessKey="e9355d51-32ca-49cf-8eb4-1ce59afbf4a7">
    <name xml:lang="en-us">
      Corporate Home Page
    </name>
    <bindingTemplates>
      <bindingTemplate
        bindingKey="48b02d40-0312-4293-a7f5-4449ca190984"
        serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a">
        <description xml:lang="en">
          Entry point into the XMLTC Web site
          through which a number of resource
          sites can be accessed
        </description>
        <accessPoint URLType="http">
          http://www.xmltc.com/
        </accessPoint>
        <tModelInstanceDetails />
      </bindingTemplate>
    </bindingTemplates>
    <categoryBag>
      <keyedReference
        tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"
        keyName="Namespace" keyValue="namespace" />
    </categoryBag>
  </businessService>
</businessServices>
```

Un servicio de negocios se identifica con un valor único asignado al atributo serviceKey. Su elemento padre businessEntity es referenciado por el atributo businessKey.

Listado 6. Atributos serviceKey de businessService y businessKey

```
<businessService
  serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a"
  businessKey="e9355d51-32ca-49cf-8eb4-1ce59afbf4a7">
  ...
</businessService>
```

El único servicio de negocios asociado con esa entidad de negocios es la página inicial del sitio Web de negocio, identificado por el elemento name.

Listado 7. Elemento name con el nombre del servicio

```
<name xml:lang="en-us">
  Corporate Home Page
</name>
```

Cada servicio de negocios proporciona los apuntadores de la especificación. También conocido como las plantillas de ligación, estos registros consisten en direcciones que vinculan el servicio de negocios a la información de la implementación. Con el uso de apuntadores de servicio, un desarrollador puede aprender cómo y donde ligar físicamente a un servicio Web.

Listado 8. Estructura bindingTemplates que alberga la información de la ubicación concreta:

```
<bindingTemplates>
  <bindingTemplate
    bindingKey="48b02d40-0312-4293-a7f5-4449ca190984"
    serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a">
    <description xml:lang="en">
      Entry point into the XMLTC Web site
      through which a number of resource
      sites can be accessed.
    </description>
    <accessPoint URLType="http">
      http://www.xmltc.com/
    </accessPoint>
    <tModelInstanceDetails />
  </bindingTemplate>
</bindingTemplates>
```

La estructura bindingTemplate mostrada en el listado previo establece la ubicación y descripción del servicio que usando los elementos accessPoint y description.

Pueden asignarse varias categorías a los servicios de negocios. En nuestro ejemplo, el URL que identificamos ha sido clasificado como un espacio de nombres que usa el elemento hijo keyedReference de la estructura categoryBag.

Listado 9. Elemento categoryBag que proporciona una categorización mediante el elemento anidado keyedReference:

```
<categoryBag>
  <keyedReference
```

```
    tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"
    keyName="Namespace" keyValue="namespace" />
</categoryBag>
```

No hay ninguna relación formal entre UDDI y WSDL. Un registro UDDI proporciona mecanismos de apuntadores para definiciones de interfaz de servicio mediante el uso de tModel. Aunque probablemente sería un documento de WSDL, no tiene que ser. El tModel representa la definición del tipo de servicio UDDI y también puede proporcionar información que relaciona a los formatos de mensaje, así como el mensaje y protocolos de seguridad.

Finalmente, la relación de negocios y datos de la suscripción son representados por los elementos publisherAssertion y subscription, respectivamente. Las estructuras publisherAssertion proporcionan medios de establecer la relación del businessEntity actual con otro. El elemento subscription permite a los subscriptores ser notificados cuando la información del perfil de la entidad de negocios se actualiza.

Puede programar la interfaz con un registro de UDDI. La especificación de UDDI proporciona varios APIs que pueden agruparse en dos categorías generales: Consulta y Publicación. Por ejemplo, podría emitir un mensaje de SOAP para buscar una compañía por nombre con la carga útil siguiente:

Listado 10. Estructura find_business envolviendo un comando para el API UDDI de consulta.

```
<find_business xmlns="urn:uddi-org:api_v3">
  <findQualifiers>
    <findQualifier>
       uddi:uddi.org:findQualifier:exactMatch
    </findQualifier>
  </findQualifiers>
  <name>
    XMLTC Consulting Inc
  </name>
</find_business>
```

Aunque hemos visto una apreciación global de las bases de UDDI, enfocada a la estructura de entidades de negocios, el paso siguiente es tratar más a detalle el núcleo de un registro de UDDI: el tModel. Esta estructura importante proporciona el acceso a los detalles técnicos requeridos para los solicitantes articular e interactuar con servicios Web disponibles.

5.1.3. Actividades de aprendizaje

Análisis e implementación con el registro UDDI

Instrucciones

1. En base al Material de Apoyo “5. Localización de Servicios Web: UDDI”, realice las tareas siguientes:

- implemente los programas TryUddi y UddiViewer que se proporciona en el rubro “Código ejercicios UDDI” del anexo C.
- Investigue para que puede utilzarse la herramienta DISCO del .NET.

Coloque sus tareas en “Tarea Individual Ejercicio (5a)”

2. En base a la presentación “Patrones de invocación UDDI” del anexo B, implemente el registro con UDDI como tModel de la aplicación elaborada en la sección “Actividades de aprendizaje 1.6”

- Coloque su implementación en “Tarea Individual Ejercicio (5b)”

3. Contestar el cuestionario “Preguntas comprensión de UDDI”

- Coloque sus respuestas en “Tarea Individual Cuestionario (5a)”.

5.2. Escenarios UDDI

5.2.1. Uso de escenarios UDDI

Para ilustrar el uso de situaciones o escenarios de UDDI, trataremos el ejemplo del consorcio MegaBucks.

El consorcio MegaBucks (una consorcio financiero) quiere crear una manera estándar de valorar los negocios pequeños al por menor y entonces publicar esa información para que sus miembros de empresas puedan implementar servicios Web para conformar a ese estándar. En este ejemplo, el consorcio opera un registro privado. Para permitirles a sus miembros tener acceso a ese estándar, el consorcio necesita:

- Producir un archivo WSDL para definir las especificaciones a las que debe adherir un servicio de valuación. El proveedor del servicio de valuación probablemente sería un miembro del consorcio financiero.

- Publicar el archivo WSDL a una ubicación pública (por ejemplo, www.megabucks.org/valuation.wsdl en su servidor o cualquier ubicación públicamente accesible).

- Crear un tModel para representar las especificaciones de servicio de valuación (estas característica técnicas se describen en el archivo de WSDL mencionado en el primer punto).

- Publicar el tModel en su registro. Como parte del proceso de publicación, el registro emite una única clave para el tModel (por ejemplo, 5000X) y almacena el URL del archivo WSDL en el elemento hijo de overviewURL del elemento del tModel.

En seguida, la figura 3 ilustra esa secuencia de eventos.

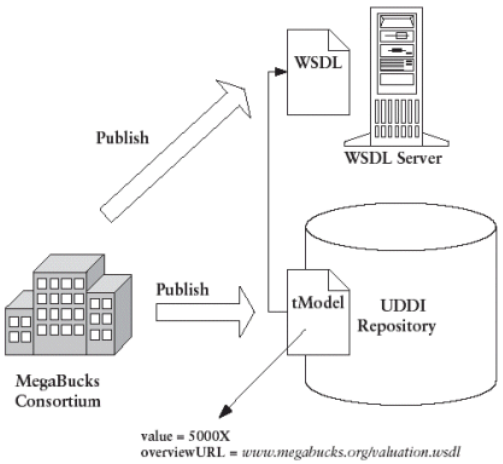


Figura 3. Creación de un tModel

Una empresa que quiera publicar un servicio Web para cumplir ese estándar (probablemente un miembro del consorcio MegaBucks) necesita hacer lo siguiente (ver figura 4):

- Publicar su negocio al registro privado operado por consorcio MegaBucks.

- Publicar el servicio Web con un bindingTemplate, punto de acceso de que es el URL de la implementación de servicios Web de las empresas y de quien los valores tModel (5000X) es ese del tModel publicado por el consorcio MegaBucks. En esencia, este miembro de empresa está anunciando que su servicio de valuación obedece a un conjunto de especificaciones (capturado en el tModel) establecido por un consorcio (en este caso, Consorcio MegaBucks).

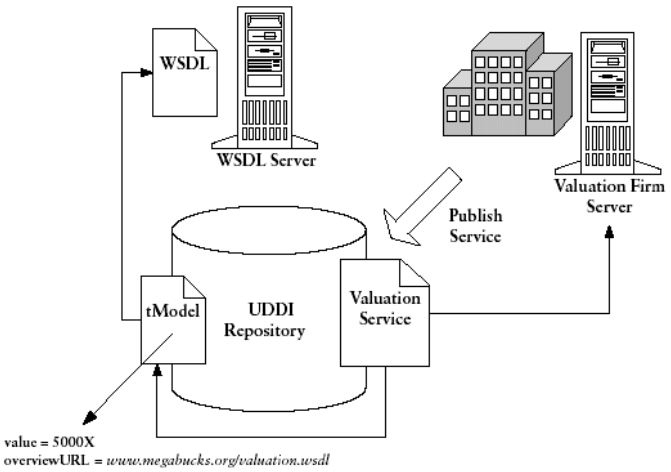


Figura 4. Usando el tModel

Como se menciono previamente, múltiples empresas pueden publicar servicios Web que implementan el mismo tModel.

Una empresa que realmente quiere invocar el servicio de valuación (por ejemplo, una compañía propietaria que compra otros negocios) primero tiene que saber el valor que representa el servicio de tasación (en este caso, 5000X) y debe usar este valor para localizar el servicio del Web estáticamente (navegando los nodos del operador vía el Web) o de manera programada. Si los resultados de búsqueda en más de un servicio, entonces el cliente puede usar otro criterio (precio, ubicación, etc.) antes de seleccionar un proveedor.

Este ejemplo:

- El consorcio MegaBucks es un publicador (porque está publicando un servicio, el tModel) y un intermediario de servicio (porque está organizando un registro que las empresas del solicitante están buscando).

- El miembro de empresa, el que está proporcionando el servicio de valuación actual, es el proveedor.

- La compañía propietaria es el cliente.

En seguida la figura 5 ilustra al solicitante localizando el servicio de valuación e invocándolo.

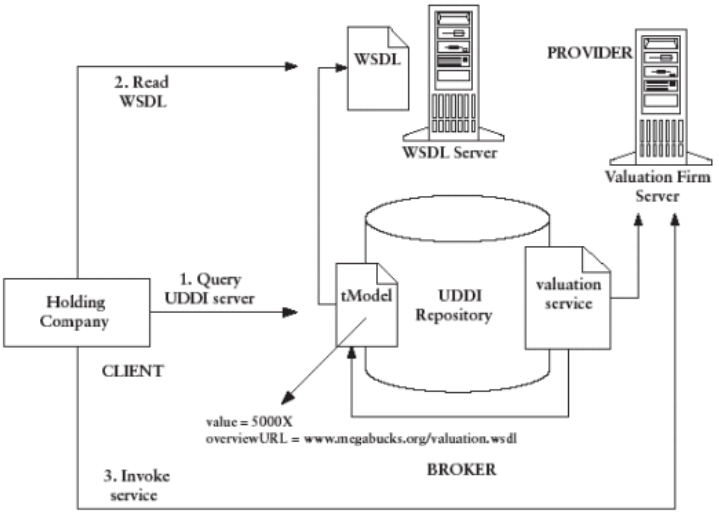


Figura 5. Localización e invocación del servicio Web.

5.2.2. Publicación y consulta UDDI

La publicación de un servicio Web para un registro UDDI y la búsqueda de un registro UDDI para descubrir un servicio Web pueden realizarse de dos maneras alternativas:

- Con medios de interfaz basada en Web, los cuales son proporcionados por proveedores como IBM y Microsoft. Estas interfases basadas en Web habilitan trabajar de forma interactiva siguiendo los pasos necesarios para la publicación y búsqueda de servicios Web.

- Con medios de programación usando las APIs (Application Programming Interfaces) de UDDI, las cuales se dividen en dos tipos, APIs de publicación (publish) y APIs de consulta (inquiry).

Usando medios de interfaz basada en Web simplifica mucho el trabajo y generalmente los proveedores proporcionan guías para su manejo. Sin embargo con el uso de las APIs de la especificación de UDDI permite al programador implementar mayores capacidades de publicación y búsqueda de servicios Web.

En esta sección trataremos las APIs involucradas en el proceso de registro (llamado publicación) con un registro UDDI, así como las APIs involucradas en el proceso de descubrimiento (llamado consulta) con un registro UDDI. Utilizaremos ejemplos que usan el SDK UDDI .NET en C#.

APIs de UDDI

Para facilitar la interacción entre la interfaz de usuario y registro, necesita un conjunto de APIs. Para fácil interoperabilidad, estas APIs deben escribirse en el mismo lenguaje de programación usado para la interfaz de usuario o lógica de negocio. Esto ha dado lugar para varias APIs de UDDI de lenguajes específicos (ver figura 6). Los APIs de lenguajes específicos manejan las comunicaciones del XML para la capa de aplicaciones.

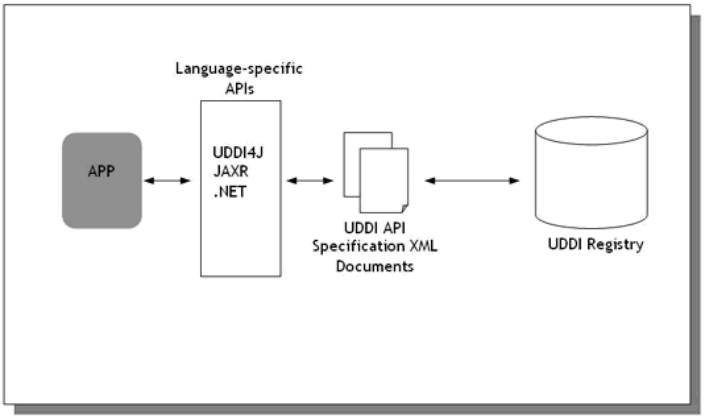


Figura 6. Relaciones entre APIs de lenguajes específicos APIs UDDI

Para entender las APIs, es esencial conocer las posibles interacciones con un registro UDDI.

Patrones de interacción UDDI

Para mostrar el uso de los APIs de C# y Java, pasemos el caso de negocios de un proveedor de servicios que quiere publicar su servicio dentro del Registro de Negocios Universal (UBR), un consorcio global de registros de UDDI.

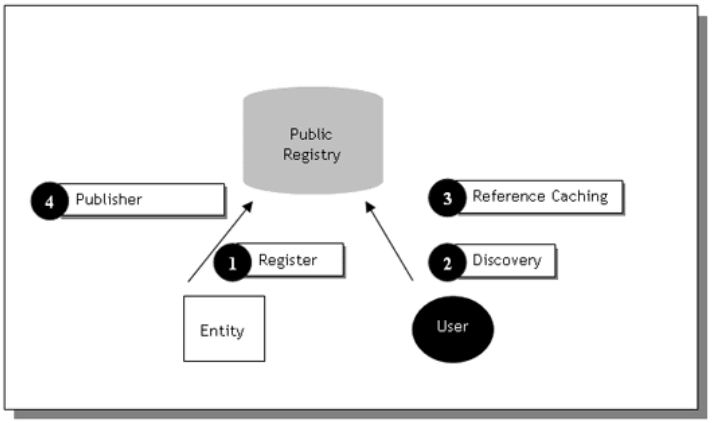


Figura 7. Típicos patrones de interacción con el registro UDDI

La interacción con un registro UDDI vía sus puntos de comunicación caen en dos categorías: publicación y consulta. Todos los APIs de UDDI pueden ser clasificados en estas categorías. Los puntos de comunicación del registro UDDI también son asociadas con estas dos categorías: URL Publicación y URL consulta. Estas URLs reciben y procesan las correspondientes llamadas UDDI API.

APIs de publicación

API de publicación recibe la información de publicación sobre una entidad de negocio y los servicios y especificaciones de la interacción que ofrece. La publicación API usa un localizador de recurso uniforme dedicado (URL) llamado URL Publicación. URL Publicación acepta mensajes del XML que realizan las tareas, como lo siguiente (todos de los cuales también incluye la eliminación):

- Registro de la entidad de negocio
- Registro de servicio
- Servicio de definiciones de ligaciones
- Registro de taxonomía
- Registro de especificación de interacción
- Confirmación de las relaciones de negocios

URL Publicación para los operadores de UBR normalmente tomaría la forma siguiente:

```
https://uddi.mycompany.com/publish
```

Note que estos URLs proporcionan comunicación segura porque aceptan los datos sensibles como nombre de usuario y contraseña. Este tipo de información debe ser encriptada cuando viaja sobre de una red abierta como Internet.

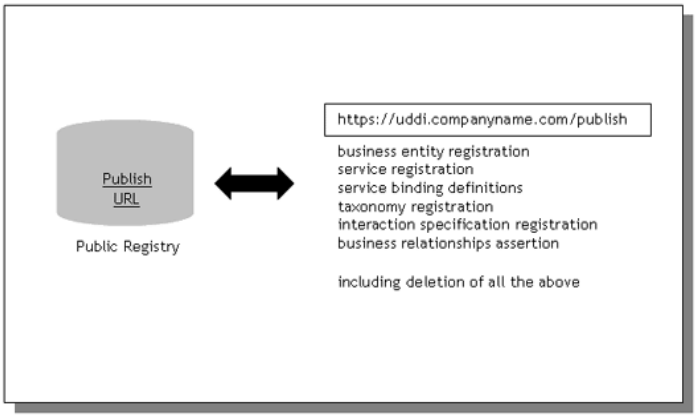


Figura 8. Interacciones UDDI soportadas sobre una URL Publicación

Establecimiento de conexión

Antes de interactuar con un registro UDDI para publicar los propósitos, es esencial establecer las credenciales correctas con el formador de UDDI. Para el registro de Microsoft, esto significa consiguiendo una cuenta Passport. Una vez esta cuenta se establece, puede usarse para interactuar con el registro de Microsoft.

Por la especificación de UDDI, un usuario registrado y validado (o programa) puede publicar la información sobre un registro UDDI. Registro y validación entran en forma de una autenticación muestra. Esta muestra tiene una vida específica durante la que puede presentarse al registro por publicar los propósitos. Un segmento de programa de C# usado para este propósito podría parecerse:

```
Publish.AuthenticationMode =
  AuthenticationMode.UddiAuthentication;
Publish.Url = "URL of the UDDI registry";
Publish.User = "Passport login name";
Publish.Password = "Passport login password";
GetAuthToken gat = new GetAuthToken();
gat.UserID = Publish.User;
gat.Credentials = Publish.Password;
AuthToken at = gat.Send();
```

Cuando el método Send del objeto GetAuthToken se invoca, los APIs SDK UDDI .NET interactúan con el registro UDDI en formato XML y reciben una autenticación muestra en la autenticación exitosa. Esta muestra puede usarse entonces al publicar los datos sobre un servicio o proveedor de servicios. Consideremos un ejemplo.

Ejemplo: Prudentially 401(k), Inc.

Prudentially 401(k), Inc. Es una subsidiaria de un conglomerado de servicios financieros global, especializada en proporcionar administración de servicios de fondos de pensiones para sus clientes. La compañía ha establecido una clientela grande a través de los años y ha sido uno de los miembros fundadores de Interacciones Financieras y Organización de Estandarización de Transacciones (FITSO).

FITSO es un consorcio establecido no lucrativo de instituciones financieras grandes en América del Norte, incluso los proveedores del fondo de pensiones. FITSO ha embarcado en un esfuerzo de estandarización para interaccionar con organizaciones de servicio financiero más simple y más predecible. FITSO está centrándose actualmente en proveedores del fondo de pensiones y espera seguir después de esto a otras áreas de servicio financiero. Después de investigar los

estándares de servicios Web y patrones de interacción, FITSO decide que necesita adoptar y fortalecer dos tipos de estándares:

- Esquema de clasificación para entidades de negocio
- Conjunto de patrones de interacción básicos

Dado la familiaridad del Sistema de Clasificación de Industria Norteamericana (NAICS) en la industria financiera, FITSO decide recomendar el esquema de clasificación de NAICS para clasificar las entidades de negocios detrás de los servicios de pensiones. Esta opción de taxonomía traduce a usar el tModel de NAICS cargado previamente en el registro UDDI.

FITSO también ha decidido estandarizar en interacciones básicas. Los consumidores de servicio podrían desarrollar aplicaciones que interactúan con los servicios; la visión es que las aplicaciones diseñadas para trabajar con un servicio intercambiarían fácilmente con otro. Basado en los tipos de interacciones que son bastante simples para estandarizar, FITSO diseña un conjunto de interfaces de servicio para los servicios del fondo de pensiones.

Para este ejemplo, nos centramos en la especificación de servicios de información 401(k). Las funciones que deben ser soportadas por un fondo de pensiones de FITSO son como sigue:

1. Obtener lista de fondos.
2. Obtener el rendimiento del fondo.
3. Agregar al empleado a un plan.
4. Obtener los datos de contribución del empleado.

FITSO necesita registrar un tModel asociado con esta interfaz de interacción de servicio, como se muestra más adelante. Este tModel se llama TM401k. Prudentially 401(k), Inc. necesita usar esta información al registrar como una compañía que proporciona un servicio que cumple el tModel TM401k.

Suponga que los datos del registro para Prudencial 401(k) son como se muestra en la tabla.

Modelo	**Representación de UDDI**
Negocio	Entity: Prudentially 401(k), Inc.
Clasificación	NAICS
Subclasificación	524292
Servicio	Prudentially401kInformationService
Especificación de interfaz	TM401k

Publicación de la entidad de negocio

Antes de que sus servicios puedan publicarse, Prudentially 401(k), Inc. necesita registrar sus entidades de negocio con el registro UDDI. Esta entidad registrada supone la titularidad de los servicios registrada después de esto. Los consumidores de servicio que descubren los servicios aplicables a sus necesidades pueden buscar la entidad responsable para conseguir más información sobre el proveedor de servicios.

En C#, la clase principal que maneja el registro de la entidad de negocio es SaveBusiness. Esta clase actúa como el recipiente que contiene la información necesaria para registrar el negocio. El segmento del código siguiente puede usarse para anunciar las entidades de negocio para Prudentially 401(k), Inc.

```
SaveBusiness sb = new SaveBusiness();

// Add business information
sb.BusinessEntities.Add();
sb.BusinessEntities[0].Names.Add("en", "Prudentially401k, Inc.");
sb.BusinessEntities[0].Descriptions.Add
  ("en", "Prudentially401k, Inc. is known for outstanding
      service to its customers for their 401(k) needs.");
sb.BusinessEntities[0].DiscoveryUrls.Add
  (http://www.tasmanave.com/Prudentially401k, "http");

//Add contact
Contact c = new Contact("John Doe", "Contact");
c.Phones.Add("111-222-3333", "");
sb.BusinessEntities[0].Contacts.Add(c);
```

Para la descripción humano-legible asociada con cualquier atributo, como la descripción de la entidad de negocio encima, la especificación de UDDI permite una descripción por lenguaje de código. Los lenguajes de códigos aplicables son definidos por el estándar del Internet Engineering Task Force (IETF) conocido como Request For Comment (RFC).

Clasificación de la entidad de negocio

En el momento de registro, una entidad de negocio debe clasificarse también debidamente para que pueda ser descubierta por el cliente adecuado, aunque esto no es determinado por la especificación de UDDI. Prudentially 401(k), Inc. necesita clasificarlas bajo la taxonomía de NAICS y el código de clasificación del fondo de pensiones, 524292, debido a la especificación de FITSO. Para cada clasificación, se requieren tres elementos de datos:

- La clave tModel asociada con el esquema de clasificación

- Nombre de clave bajo el que la entidad es clasificada
- Valor o código asociado con la clave

Una clase keyedReference contiene estos tres elementos juntos:

```
//Add taxonomy tModel
KeyedReference kr = new KeyedReference();
//Use UDDI-defined tModel key for NAICS
kr.TModelKey = "uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2";
kr.KeyName = "Pension fund, third-party administrative services";
kr.KeyValue = "524292";
sb.BusinessEntities[0].CategoryBag.Add(kr);
```

Aunque no es usado en este ejemplo, una entidad de negocio también debe especificar el conjunto de identificadores, como identificadores DUNS (Dun & Bradstreer Number) que pueden identificar la entidad singularmente. Aunque proporcionando estos identificadores es optativo, ayudan en la precisión de una entidad descubierta. Como el esquema de clasificación, la especificación de UDDI no establece ningún esquema de identificación.

Una vez que el recipiente objeto SaveBusiness está listo con toda la información necesaria, la entidad puede registrarse usando el método Send:

```
BusinessDetail bd = sb.Send();
```

El objeto BusinessDetail devuelto puede usarse para conseguir la clave de UUID (Universally Unique Identifier) asignada al recurso registrado.

Publicación del tModel

Recuerde que FITSO desarrolló una especificación de interacción de servicio, TM401k, para el fondo de pensiones 401(k) servicios de información. Esta especificación de la interacción necesita ser registrada en UDDI como un tModel.

En el SDK UDDI .NET, la clase principal a usar para el registro del tModel es SaveTModel. Esta clase se usa por registrar (y modificar) de un tModel. Como con la clase de la entidad de negocio, el modelo de objetos tModel necesita ser anunciado antes de ser guardado en el registro UDDI:

```
SaveTModel stm = new SaveTModel();

stm.TModels.Add();
stm.TModels[0].Name = "TM401k";
stm.TModels[0].Descriptions.Add
  ("en", "tModel defined by FITSO for 401(k) interactions");
stm.TModels[0].OverviewDoc.OverviewURL =
```

```
  "http://www.tasmanAve.com/FITSO/TM401k.wsdl";

//Add category bag conditionally
stm.TModels[0].CategoryBag.Add();
//Use built-in uddi-org:general_keywords tModel UUID
//to classify this tModel
stm.TModels[0].CategoryBag[0].TModelKey =
  "uuid:A035A07C-F362-44dd-8F95-E2B134BF43B4";
stm.TModels[0].CategoryBag[0].KeyName = "KEYWORD";
stm.TModels[0].CategoryBag[0].KeyValue = "401(k)";

TModelDetail tmd = stm.Send();
```

Los elementos de datos asociados con registro del tModel son similares a aquéllos del registro de una entidad de negocio, nombre, descripción, categoría, y así sucesivamente. El registro UDDI provee una única clave de UUID al registro del tModel. Los tModel que usan esta clave entonces pueden identificarse.

Publicación del servicio

Seguir un patrón similar, el API principal para publicar un servicio al registro UDDI es SaveService mediante el SDK UDDI .NET. Usar los métodos de la clase SaveService, el objeto de servicio necesita ser anunciado con los detalles de servicio. Un detalle de servicio adicional para notar es el campo BusinessKey al que vincula el servicio para el responsable o publicación de la entidad de negocio. En este caso, vinculamos el P401Kservice a la entidad de negocio Prudentially401(k), Inc. El BusinessKey devuelto cuando la entidad de negocio fue registrada debe almacenarse:

```
SaveService ss = new SaveService();
ss.BusinessServices.Add();
ss.BusinessServices[0].Names.Add("en", P401kService);
ss.BusinessServices[0].Descriptions.Add
  ("en", "401(k) management service by
     Prudentially401(k), Inc. abiding by the FITSO
     TM401k tModel");
ss.BusinessServices[0].BusinessKey = "business key
                  from SaveBusiness call";
```

Además de esta información, el servicio de información de ligación también debe proporcionarse. Esta información de ligación incluye los datos sobre un caso específico del servicio como el punto de acceso de servicio y el tModel que el servicio cumple:

```
BindingTemplate bt = new BindingTemplate();
bt.AccessPoint = new
```

```
  AccessPoint(Microsoft.Uddi.Api.URLType.Http,
   "service access point");
TModelInstanceInfo tmii = new TModelInstanceInfo();
tmii.TModelKey = "TM401k registration key";
bt.TModelInstanceDetail.TModelInstanceInfos.Add(tmii);
ss.BusinessServices[0].BindingTemplates.Add(bt);
```

Un servicio también debe ser clasificado, como otros recursos como entidades de negocio y tModels.

Eliminación desde el registro

La especificación de UDDI también proporciona funcionalidad de borrado que puede usarse para retirar cualquier recurso que ya no necesita ser anunciado. Los API correspondientes en .NET exigen a la autenticación adecuada borrar un recurso y la clave del identificador exclusivo. Con suerte, el publicador tendría una lista de todas las claves del identificador de recursos. Si no, el publicador pudiera encontrar el recurso, podría obtener su clave del identificador y, a continuación, podría borrar del registro. Obteniendo la clave del identificador de un recurso simplemente no proporciona la titularidad sobre del recurso; la clave del identificador es meramente una única referencia al recurso. El registro UDDI establece las relaciones de la titularidad dentro del propio registro. Por consiguiente, sólo el usuario de UDDI que publicó un recurso al registro puede borrarlo. El segmento del código siguiente muestra la eliminación de un servicio:

```
DeleteService ds = new DeleteService();
ds.ServiceKeys.Add("UUID key of the service to be deleted");
DispositionReport dr = ds.Send();
```

Otros recursos UDDI-registrados como plantillas de ligaciones, tModels y entidades de negocio pueden ser borrados de una manera similar usando las APIs correspondientes y clases. Debido a que tModels son extensivamente referenciados en el modelo de datos UDDI, simplemente no pueden borrarse del registro. Cuando un API delete_tModel se invoca, el tModel especificado está "oculto." Esto significa que los tModel borrados que usan el find_tModel no pueden descubrirse, pero los detalles que todavía usan el get_tModel pueden obtenerse. Descubrimiento se borra, pero los usuarios existentes pueden continuar usando el tModel. La única manera de borrar un tModel completamente es solicitar al operador dónde el tModel fue guardado.

APIs Consulta

APIs Publicación se usan por proveedores de servicios para anunciar y ofrecer sus servicios a los usuarios potenciales. Los usuarios de servicio, por otro lado, deben explorar un registro de UDDI para proveedores de servicios potenciales que

pueden satisfacer sus necesidades. UDDI provee APIs Consulta para este propósito que serán tratadas en seguida en esta sección.

Como APIs Publicación, APIs Consulta usan un localizador de recurso uniforme dedicado (URL) a un registro. Este URL Consulta puede usarse para recuperar la información sobre las entidades registradas como negocios, servicios y tModels. El URL Consulta acepta mensajes del XML que realizan las tareas, como lo siguiente (todos pero el último que también incluye obtención de detalles del registro):

- Descubrimiento de la entidad de negocios
- Descubrimiento de servicio
- Descubrimiento de servicio de ligación
- Descubrimiento de taxonomía
- Descubrimiento de especificación de interacción
- Descubrimiento de relaciones de negocios

La figura 9 ilustra el papel del URL Consulta en un registro de UDDI. Las URLs Consulta para el UBR normalmente tomarían la forma siguiente:

```
http://uddi.tasmanAve.com/inquiry
```

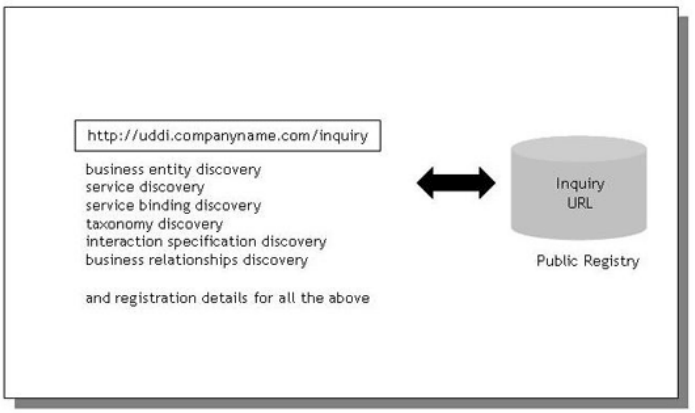


Figura 9. Interacciones de UDDI soportadas en URL Consulta

Al contrario del URL Publicación, el URL Consulta no usa un modo seguro de comunicación, porque APIs Consulta no necesitan los datos sensibles como nombres de usuario y contraseñas. La información recuperada también es considerada de dominio público; por consiguiente, el publicador debe tener cuidado sobre el tipo de datos que se publican a un registro de UDDI.

Establecimiento de la conexión

Al contrario de la publicación, la consulta es una interacción del registro más simple y no requiere el registro a priori con el proveedor del registro. Esto también significa que ninguna muestra de la autorización se necesita al usar APIs Consulta.

El SDK UDDI .NET provee APIs para descubrir recursos (servicios, negocios y tModels) que satisfacen cierto criterio. Para demostrar el uso de API de UDDI Consulta, consideremos el servicio de ecosistema de 401(k) de los ejemplos de publicación UDDI.

Este ejemplo prueba el URL Consulta para descubrir el tModel uddi-org:inquiry_v 2, usando la clase FindTModel. Un segmento de programa de C# usado para establecer la conexión para la consulta podría parecerse a lo siguiente:

```
Inquire.Url = "URL of UDDI registry";
FindTModel ftm = new FindTModel();
Console.Writeln("Searching for tModel
  \"uddi-org:inquiry_v2\"");
ftm.Name = "uddi-org:inquiry_v2";
TModelList tml = ftm.Send();

Console.Writeln("\r\nThe TModel Name: " +
  tml.TModelInfos[0].Name);
Console.Writeln("\r\nThe TModel Key: " +
  tml.TModelInfos[0].TModelKey);
```

Cuando el método Send de la clase de FindTModel se invoca, APIs SDK UDDI .NET interactúan con el registro de UDDI en formato XML y reciben la información del tModel sobre conexión exitosa.

El ecosistema de los ejemplos consiste de los participantes Prudentially 401(k), Inc. y FITSO.

La última categoría de participantes en este ecosistema consiste en las compañías asociadas que usan el servicio Prudentially 401(k) para sus planes de jubilaciones. Ahora veamos cómo el cliente de Prudentially usará APIs Consulta de UDDI para usar el servicio 401(k) eficazmente.

El proceso Consulta

Para descubrir un servicio particular con intervención manual mínima, una estructura debe estar en lugar que soportar la ruta al servicio deseado para una aplicación. La clasificación y característica técnicas de interacción de servicio

proporcionan simplemente esa estructura. Las entidades de negocios y servicios son clasificados dentro de esa infraestructura (ver Figura 10) y el proceso de consulta navega esa estructura.

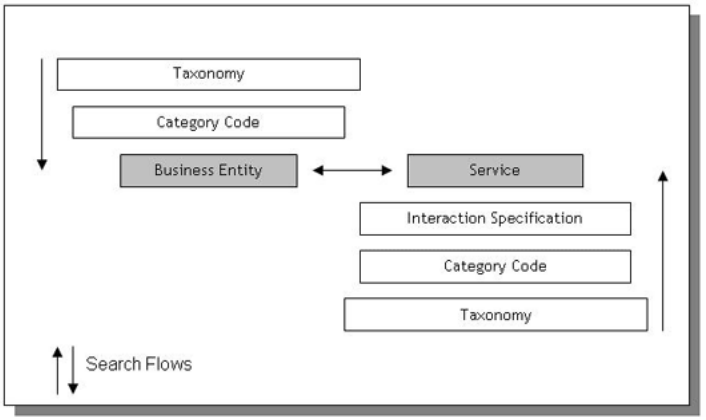


Figura 10. Estructura de consulta en el registro UDDI

El proceso de encontrar un servicio depende de la información disponible sobre el tipo o clase de servicios a ser descubierto. La especificación de UDDI proporciona varias APIs para descubrir los diversos recursos en el registro. Solo nos centraremos en los tipos siguientes de consultas:

- Encontrar la interfaz de servicio (tModel)
- Encontrar el servicio
- Encontrar la entidad de negocios

Normalmente, muchos detalles se publican a lo largo de con el recurso principal (si un servicio, entidad de negocios o un tModel). Varios de estos detalles buscables son:

- Nombre
- Clave de la categoría
- Clave del identificador exclusivo
- TModel aplicable
- Recurso padre

Una aplicación de consumo de servicio puede diseñarse para buscar en cualquier número o combinación de estos tipos de detalles. Sin embargo, el registro de UDDI se diseña para el uso en un paradigma del ecosistema. En seguida tenemos el guión típico para el que una aplicación de consumo de servicio se diseña:

1. Encontrar todas las especificaciones de interacción de servicio pertinentes.
2. Seleccionar uno que parece satisfacer sus necesidades.

3. Encontrar todos los servicios que obedecen esa especificación de interacción de servicio particular.
4. Seleccionar el servicio que es bastante conveniente.
5. Encuentrar las entidades de negocios responsables para los servicios descubiertos.

Un consumidor de servicio puede tener una relación existente con un proveedor de servicios y puede querer buscar todos sus servicios. En este caso, el proceso de consulta puede incluir buscando todos los servicios simplemente proporcionados por esa entidad de negocios.

Consideremos un ejemplo. American Corporation, Inc. (ACI) es una compañía con una mano de obra regular. Ha decidido ofrecer recientemente planes 401(k) a sus empleados, ACI investiga la industria del fondo de pensiones y llega a la conclusión que la compatibilidad de FITSO es un criterio importante en selección; por consiguiente, ACI ha determinado que sólo funcionará con 401(k) proveedores de servicios FITSO-compatibles. En lo sucesivo se explican explica el proceso de consulta electrónica de ACI soportado por el registro de UDDI.

Descubrimiento de la especificación de interacción de servicio

La figura 11 ilustra una secuencia de la consulta típica para un consumidor de servicio. Al principio del proceso de la consulta, el consumidor normalmente sabe sólo que los servicios deseados cumplen una cierta especificación de la interacción. En este ejemplo, ACI está buscando servicios que obedecen la especificación TM401k que FITSO publicó al registro de UDDI; esta especificación proporciona la guía en qué características de 401(k) proveedores de servicios FITSO-compatibles deben incluir.

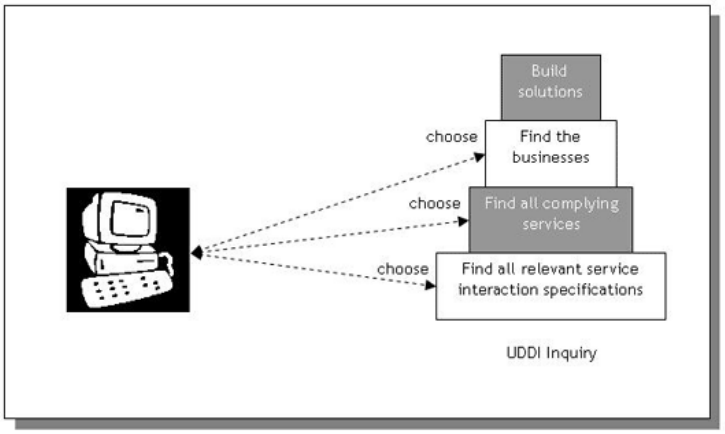


Figura 11. Consulta Ecosistema-basado

Recuerde que las llamadas al URL Consulta no son seguras porque ninguna autenticación se requiere (sobre todo en el caso del UBR usado en estos ejemplos). La configuración principal está denotando el URL Consulta con el que el consumidor de servicio quiere comunicar simplemente.

La clase FindTModel es el punto de partida para encontrar un tModel. FindTModel proporciona varios métodos para construir las consultas del registro para buscar el tModels. Las consultas pueden diseñarse para ocuparse de una búsqueda basada en amplitud de especificaciones de interacción y esquemas de clasificación o una búsqueda más refinada que usa más información sobre la especificación de la interacción. En nuestro ejemplo, el caso más amplio usa una búsqueda del comodín simplemente para encontrar los caracteres particulares en el name de la especificación tModels publicada en el registro. La búsqueda más refinada será basada en categoría.

Búsqueda basada en amplitud

La búsqueda basada en amplitud no siempre es suficiente. Una de las metas de registros XML-basado es ayudar en el proceso de decisión electrónica. Las entidades como consorcios y los cuerpos de gobierno deben proporcionar más detalles acerca de cómo alcanzar la especificación de la interacción exacta. Idealmente, proporcionarían los detalles siguientes:

- Nombre de especificación de interacción
- Esquema de clasificación y detalles

Búsqueda refinada

FITSO proporciona la información de la interacción detallada en su especificación. Usar estas pautas, ACI puede refinar su búsqueda para localizar las especificaciones de la interacción con un nombre específico y tipo de clasificación: La especificación TM401k se publicó dentro de la taxonomía del general_keyword, con la palabra clave 401k. (Detrás de los escenarios, una estructura CategoryBag se construye para almacenar la información adicional y preparar la consulta XML adecuada para ejecutar en el registro de UDDI).

El método Send comunica la consulta al registro de UDDI:

```
FindTModel ftm = new FindTModel();
...
ftm.CategoryBag.Add();
ftm.CategoryBag[0].TModelKey = "tModel key from publish";
ftm.CategoryBag[0].KeyName = "KEYWORD";
ftm.CategoryBag[0].KeyValue = "401(k)";
```

```
TModelList tml = ftm.Send();
foreach( TModelInfo tmi in tml.TModelInfos )
{
  Console.WriteLine(tmi.Name);
  Console.WriteLine(tmi.TmodelKey);
}
```

Usando los datos del tModel encontrado, el segmento del código siguiente usa el identificador exclusivo y la clase GetTModelDetail para desplegar la descripción y los documentos de vista general asociados con cada tModel:

```
GetTModelDetail gtmd = new GetTModelDetail();
gtmd.TModelKeys.Add("Put a specific tModel key here");
TModelDetail tmd = gtmd.Send();
...
foreach(TModel tm in tmd.TModels)
{
  // Show overview document descriptions
  foreach( Description d in tm.OverviewDoc.Descriptions )
}
```

Descubrimiento del servicio

Después de obtener la clave del tModel adecuado para la especificación de interfaz de servicio, ACI puede encontrar los servicios que obedecen ese tModel específico. Llamaremos este el tModel de complacencia.

Descubriendo un servicio es muy similar al proceso por descubrir un tModel (o incluso un negocio). Usar el nombre de servicio y la clave del tModel compatible, ACI puede localizar uno o más de los servicios que obedecen al tModel de FITSO:

```
FindService fs = new FindService();
fs.Names.Add("%401%k%");
fs.TModelKeys.Add("Put compliance tModel key here");
ServiceList sl = fs.Send();
```

Los resultados de retorno de la consulta, el nombre de servicio así como las claves de identificación de servicio, son claves útiles porque proveen el acceso directo al servicio en uso futuro.

```
GetServiceDetail gsd = new GetServiceDetail();
gsd.ServiceKeys.Add("Put service UUID key here");
ServiceDetail sd = gsd.Send();
...
Console.WriteLine("Responsible Business");
GetBusinessDetail gbd = new GetBusinessDetail();
gbd.BusinessKeys.Add(bs.BusinessKey);
```

```
BusinessDetail bd = gbd.Send();

foreach(BusinessEntity be in bd.BusinessEntities)
Console.WriteLine(be.Names[0].Text);
```

Pueden obtenerse detalles de servicio adicionales para los servicios que son el resultado de la consulta propuesta, con el registro que usa la clase GetServiceDetail.

Descubrimiento de la entidad de negocios

A estas alturas, hemos cruzado un diseño del descubrimiento completo dónde descubrimos un tModel basado en ciertas palabras claves, hemos descubierto un conjunto de servicios que obedecieron al tModel específico y hemos obtenido más información sobre la entidad de negocios que tuvo el servicio.

Otro patrón del descubrimiento también es posible cuando un usuario de servicio potencial esta enterado de entidades existentes que podrían ser los socios de negocios. En esta situación, el descubrimiento pasaría en orden inverso: La entidad de negocios se descubriría primero, seguido por los servicios que ofrece.

Para descubrir una entidad de negocios, la clase FindBusiness actúa como el punto central dónde el criterio de búsqueda se proporciona. En este ejemplo, incluye una taxonomía bajo la que la entidad de negocios es clasificada:

```
FindBusiness fb = new FindBusiness();
//Add taxonomy
KeyedReference kr = new KeyedReference();
kr.TModelKey = "Put taxonomy tModel key here";
kr.KeyName = "Pension fund, third party administrative services";
kr.KeyValue = "524292";
fb.CategoryBag.Add(kr);
...
BusinessList bl = fb.Send();
```

La clase GetBusinessDetail obtiene información detallada sobre la entidad de negocios escogida:

```
GetBusinessDetail gbd = new GetBusinessDetail();
gbd.BusinessKeys.Add("Put specific business key here");
BusinessDetail bd = gbd.Send();

foreach(BusinessEntity be in bd.BusinessEntities)
{
  // Code to obtain specific business entity information
}
```

Calificadores de búsqueda

El comportamiento predeterminado para APIs Consulta, incluso aquéllos para encontrar negocios, servicios y tModels, ejecutan consultas con un conjunto de operadores (AND) de criterios de búsqueda. Sin embargo, este comportamiento puede modificarse usando el elemento opcional findQualifier. Esto es especialmente útil en despliegue de registro privado, dónde búsquedas especializadas podrían desearse.

Uso de servicios descubiertos

Cuando publicó al registro, un servicio incluye su información de servicio de ligación que proporciona el punto de entrada de servicio y lista los protocolos soportados. Usar esta información, un consumidor puede escribir una aplicación que usa el protocolo adecuado para comunicar al punto de entrada de servicio.

En el entorno de SDK UDDI .NET, la clase GetBindingDetail puede usarse para recuperar la información sobre el servicio, incluso el punto de acceso para el servicio que usa un protocolo específico. Después de que el punto de acceso se recupera, pueden intercambiarse mensajes entre el usuario y la aplicación que descubre el servicio. Se aloja conocimiento del dominio proporcionado en la aplicación, este proceso puede ser automático.

5.2.3. Actividades de aprendizaje

Implementación de escenario UDDI

Instrucciones

1. Realizar el ejercicio “5.3 Ejercicio acceso de UDDI programáticamente”

2. En base al ejercicio previo, implemente el acceso UDDI en código C#, sobre la aplicación elaborada en la sección “Actividades de aprendizaje 1.6”

- Coloque su implementación en “Tarea Individual Ejercicio (5c)”

3. Realice una síntesis sobre los procesos de publicación y localización de servicios Web.

- Coloque su síntesis en “Tarea Individual Síntesis (5a)”

5.3. Ejercicios prácticos

Ejercicio acceso de UDDI programáticamente

Antes de iniciar este ejercicio revisar las "Actividades de Aprendizaje 5.2"

En base al Material de Apoyo “5. Localización de Servicios Web: UDDI”, analice y compile el proyecto UddiExplorer proporcionado en “Código acceso UDDI programáticamente” del anexo C.

5.4. Cuestionarios

Actividad asignada por el profesor.

6. Seguridad en Servicios Web

Aunque servicios Web representan una manera conveniente y poderosa para las compañías desplegar los nuevos servicios de negocios e integrar las aplicaciones de negocios existentes, la naturaleza abierta y flexible de la infraestructura de SOAP expone a las corporaciones potencialmente a los riesgos de seguridad. Con su confianza en HTTP como el protocolo de transporte subyacente de los mensajes de SOAP, maliciosos pueden atravesar los cortafuegos corporativos, evadir las tecnologías de paquetes de filtración, etc. Es más, los mensajes de SOAP están expuestos a la falsificación, interceptación, corrupción y otras formas de mal uso. Los riesgos de seguridad no son únicos a servicios Web, pero protegiendo servicios Web de la exposición de datos y maltrato de aplicación requiere una solución comprensiva que asegure un alto grado de fiabilidad y confianza.

La seguridad de servicios Web y mensajes de SOAP, por consiguiente, se dirige a cuatro tipos principales de desafíos:

- Integridad. ¿Los contenidos de un mensaje de SOAP han sido modificados en el paso?
- Confidencialidad. ¿Alguien se ha enterado del mensaje durante el paso? ¿La privacidad se ha mantenido?
- Autorización y Autenticación. ¿El mensaje viene de un origen justo? ¿Hay permisos para tener acceso a un servicio Web particular?
- Inspección de Contenido. ¿Los contenidos del mensaje son dañinos o perjudiciales?

Las tecnologías de seguridad para red, transporte y aplicación, cada una de ellas con su enfoque, deben trabajar juntas para establecer mecanismos efectivos de protección en servicios Web.

6.1. Enfoque de niveles de seguridad

6.1.1. Comprensión de los niveles de seguridad

Cada tecnología de seguridad es extensiva pero trataremos los aspectos más característicos para servicios Web. En seguida se muestra una tabla comparativa en la que se resumen ventajas y desventajas de tecnologías de seguridad para red, transporte y aplicación:

Enfoque	Ejemplo	Ventajas	Desventajas
Red	▪ Ruteador ▪ Cortafuegos ▪ Paquete de filtración	▪ Limitan el acceso a máquinas que son autorizadas para operar dentro de un perímetro particular de la red. ▪ Bloques de tráfico basados sobre dirección IP, protocolos y asignaciones de puertos. ▪ Es transparente para las aplicaciones.	▪ No pueden inspeccionar el tráfico de la aplicación. ▪ No limita a furtivos. ▪ Sólo proporcionan autenticación y autorización de máquinas huésped.
Transporte	▪ Secure Sockets Layer (SSL) ▪ Transport Layer Security (TLS)	▪ Limitan el acceso a recursos que son autorizados para usar un servicio. ▪ Bloques de tráfico basados en claves de certificados. ▪ Digitalmente encriptan la transmisión de datos. ▪ Ofrecen seguridad punto a punto.	▪ No proporcionan seguridad terminal a terminal. ▪ No realizan credenciales de seguridad disponibles para las aplicaciones. ▪ Sólo proporcionan todo o nada de control de acceso.
Aplicación	▪ Módulos de software de aplicaciones personalizadas	▪ Limitan el acceso a recursos que son autorizados para usar un servicio. ▪ Bloques de tráfico basados en contenidos de mensajes. ▪ Digitalmente firman mensajes. ▪ Digitalmente encriptan mensajes. ▪ Ofrecen seguridad terminal a terminal. ▪ Realizan credenciales disponibles para las aplicaciones.	▪ Requieren conocimiento de los protocolos de la aplicación. ▪ Deben ser individualmente construidos o personalizados para cada tipo de aplicación.

Comparando con la seguridad de red y transporte, la seguridad de aplicación proporciona el nivel más fuerte de seguridad para los mensajes de SOAP. La seguridad de la aplicación garantiza integridad de mensaje y confidencialidad así como la autenticación y autorización de servicios Web, aunque alojando las funciones de seguridad en las aplicaciones.

En seguida trataremos cómo desarrollar y aplicar autenticación, autorización y técnicas de comunicaciones seguras, para seguridad de servicios Web y mensajes con la plataforma .NET; considerando el enfoque de niveles de seguridad:

- Seguridad punto a punto que incluye el nivel de transporte y plataforma.
- Seguridad personalizada que incluye el nivel de aplicación.
- Seguridad terminal a terminal que incluye el nivel de mensaje.

Cada enfoque tiene fuerzas y debilidades diferentes, la opción del enfoque es principalmente dependiente en las características de la arquitectura y plataformas involucradas en el intercambio del mensaje.

Seguridad punto a punto

El canal de transporte entre dos puntos terminales, el cliente y el servicio Web, puede usarse para proporcionar la seguridad de punto a punto. Esto se ilustra en la figura 1.

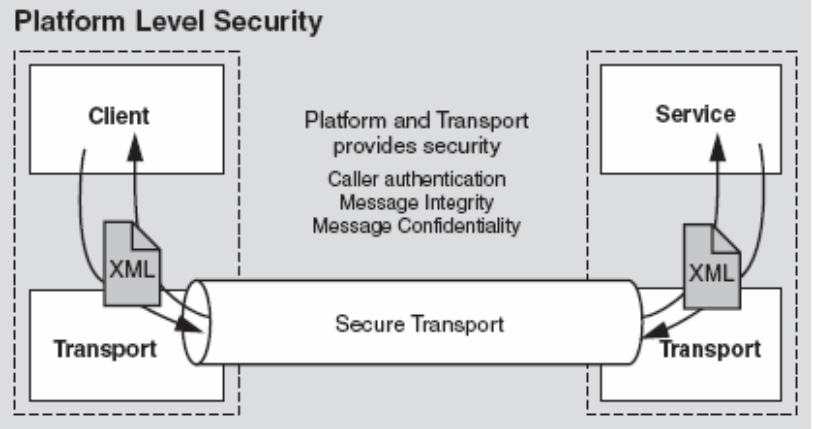


Figura 1. Seguridad a nivel de plataforma / transporte

Cuando usa seguridad de la plataforma, supone un entorno de sistema operativo Microsoft Windows herméticamente acoplado, por ejemplo, una intranet corporativa:

- El servidor Web (IIS) proporciona autenticación Básica, Digerida, e Integrada.
- Servicios Web XML hereda algunas características de autenticación y autorización de ASP.NET.
- SSL y/o IPSec pueden usarse para proporcionar integridad de mensaje y confidencialidad.

6.1.2. Uso de autenticación básica y digerida

Autenticación HTTP básica y digerida se usan para seguridad en el tráfico de HTTP. Con ambas formas de autenticación, las credenciales del usuario se pasan en el campo de encabezado de autorización HTTP. Las credenciales del usuario siempre incluyen un nombre de usuario y contraseña y pueden incluir la información con respecto al dominio al que pertenece el usuario.

Cuando estas credenciales son pasadas usando autenticación Básica, el nombre de usuario y la contraseña viajan como texto codificado en base64. Para un usuario nombrado TestUser que tiene una contraseña password, se representarán el nombre de usuario y contraseña en los encabezados HTTP como sigue:

```
Authorization: Basic VGVzdFVzZXI6cGFzc3dvcmQ=
```

Los estados del encabezado con la autenticación Básica usada y la codificación base64 copian el nombre de usuario y contraseña como VGVzdFVzZXI6cGFzc3dvcmQ =. Descifrando esta información se descubre que dice TestUser:password.

Nota. Una manera simple de transformar un valor entre el base64 y su representación de texto regular es haciendo uso de clases XML. En particular, el uso de XmlTextReader.

Lo primero que tiene que hacerse es capturar la conversación de HTTP. Esto puede hacerse usando el Toolkit SOAP y capturando una "traza sin formato". Mientras una "traza formateada" despliega sólo el XML intercambiado entre el cliente y servidor, la "traza sin formato" muestra cada byte que se presenta. Las terminaciones de la cadena de codificación base64 se representan con un “=”. Con la capacidad para localizar la cadena fácilmente, las líneas siguientes descifrarán e imprimirán la cadena en consola:

```
Dim str As String = "<ROOT>VGVzdFVzZXI6cGFzc3dvcmQ=</ROOT>"
Dim reader As New _
  System.Xml.XmlTextReader(New System.IO.StringReader(str))

reader.Read()
Dim buffer(1) As Byte
Dim AE As New Text.ASCIIEncoding()

' When done, will write out TestUser:password
While reader.ReadBase64(buffer, 0, 1) <> 0
  Console.Write(AE.GetChars(buffer)(0))
End While
```

La autenticación básica funciona bien sobre un canal de SSL debido a que el mensaje completo HTTP intercambiado es encriptado.

Veamos ahora el bosquejo de desarrollo de un servicio Web XML simple que utiliza autenticación Básica, más adelante mencionamos la autenticación Digerida.

El método Web es el siguiente:

```
<WebMethod()> Public Function HelloWorld() As String
  HelloWorld = "Hello " & Me.User.Identity.Name
End Function
```

Esta función toma el nombre ligado a la identidad del que invoca y le dice "Hola".

El tipo de la autenticación se establece como Windows en el archivo de web.config.

```
<authentication mode="Windows" />
```

Desde IIS se aplica la autentificación Básica HTTP. Dentro de IIS en el nodo de la aplicación del servicio Web:

- Haga clic derecho para propiedades.
- Haga clic en la etiqueta de directorio seguridad y haga clic en el botón editar.
- Desactive la casilla de verificación de acceso anónimo y asigne la casilla de verificación de autenticación Básica. Si quiere permitir sólo usuarios de Windows válidos, active la casilla de verificación de Windows autenticación Integrada también. Esto se muestra en la figura siguiente:

- Haga clic en OK dos veces y regresará a la consola de gestión de IIS. La aplicación aceptará ahora sólo usuarios autenticados que tienen las cuentas de Windows.

Ahora que el servicio Web es bloqueado y sólo los usuarios autenticados tendrán acceso, el paso siguiente es cómo tendrán el acceso. Para hacer esto, se necesitan obtener las condiciones de conversación, con dos clases: CredentialCache y NetworkCredential. Ambas están en el espacio de nombres System.Net y trabajan juntas.

Para nuestros propósitos, CredentialCache es usada para hacer las asociaciones entre los servidores que se quieren acceder y el nombre de usuario/contraseña usado para autenticarse a los servidores. Entonces se comunica al Proxy para el uso de CredentialCache cuando se necesita una combinación de nombre de usuario/contraseña. El listado 1 describe un cliente muy simple que utiliza la autenticación Básica HTTP para validarse para utilizar el Servicio Web.

Listado 1. Un cliente muy simple que utiliza la autenticación Básica HTTP

```
Sub Main()
  Dim svc As New localhost.Service1()

  ' Tell the proxy to use the current user's credentials
  ' whenever it needs to authenticate this client.
  svc.Credentials = System.Net.CredentialCache.DefaultCredentials
  svc.AllowAutoRedirect = True
  svc.PreAuthenticate = True

  ' Show the results and exit
  ' The expected output is "Hello [user name]"
  System.Console.WriteLine(svc.HelloWorld())
  System.Console.WriteLine("Press return to exit")
  System.Console.ReadLine()
End Sub
```

Para cambiar esto sobre una autenticación Digerida HTTP, el cliente permanece igual. Simplemente que en los pasos previos de IIS para la aplicación, se selecciona dominio de servidores Windows en lugar de autenticación Básica.

La autenticación Digerida funciona encriptando el encabezado de la autenticación HTTP. La información de la identificación del usuario viaja dentro de un Hash MD5. En una red de Windows, esta información puede ser autenticada sólo por un controlador de dominio.

6.1.3. Actividades de aprendizaje

Implementación de autenticación cliente y servicio

Instrucciones

1. En base al Material de Apoyo “6. Seguridad en Servicios Web”, realice las tareas siguientes:

- Implemente el proyecto de autenticación Autentica1 que se proporciona en el rubro “Código ejercicios Seguridad” del anexo C.
- Implemente el proyecto de autenticación Autentica2 que se proporciona en el rubro “Código ejercicios Seguridad” del anexo C.
- Compare los proyectos Autentica1 y Autentica2, y describa las características de implementación de seguridad que utilizan.

Coloque sus tareas en “Tarea Individual Ejercicio (6a)”

6.2. Seguridad de la aplicación y mensaje

6.2.1. Seguridad en el nivel de la aplicación

Con el enfoque de seguridad a nivel de la aplicación, la aplicación toma la seguridad y usa las opciones de seguridad personalizadas. Por ejemplo:

- Una aplicación puede usar un encabezado de SOAP personalizado para pasar las credenciales del usuario para autenticar el usuario con cada petición de servicio Web. Un enfoque común es pasar un ticket (o nombre de usuario o licencia) en el Header SOAP.

- La aplicación tiene la flexibilidad para generar su propio objeto principal que contiene los roles. Ésta podría ser una clase personalizada o clase GenericPrincipal proporcionada por el .NET.

- La aplicación puede encriptar selectivamente lo que ella necesita, aunque esto requiere que el almacenamiento de la clave segura y los desarrolladores deben tener conocimiento de las API de criptografía pertinente .

- Una técnica alternativa es usar SSL para proporcionar confidencialidad e integridad y combinarlo con Header SOAP personalizados para realizar la autenticación.

Uso de autorización a nivel de la aplicación

A veces, puede tener sentido proporcionar sus propios mecanismos de autenticación. Algunas de las razones para incluirlo son:

- Su servicio Web está ejecutando en una intranet corporativa y quiere conceder el acceso a los departamentos, no los individuos. También puede suponer que todos los usuarios son legítimos.

- Los clientes ya están usando credenciales que gestiona n para otros elementos (por eje mplo, cliente ID y contraseña).

- La integración con otros sistemas de la autenticación no tiene sentido para su aplicación.

Cualquier razón, creando su propio mecanismo de autenticación es posible.

Las credenciales son útiles para establecer titularidad de datos, derechos para ver o manipular los datos y para rastrear el curso del servicio Web. Sin tener en cuenta cómo se usan, las credenciales es la información sobre la entidad que llama un método Web particular que es extra. En un API programático típico, conseguiría la identidad del usuario llamando algunas funciones específicas del sistema operativo y nunca harían esa parte de datos de la firma de la función. Con un servicio Web, puede hacer no siempre esto. Sin embargo, puede guardar fuera las credenciales de la firma de la función actual. Para ello, requiere que el cliente envíe las credenciales en el Header SOAP.

La forma más común de autenticación a nivel de la aplicación incluye el uso de un nombre de usuario y contraseña. Típicamente, este inte rcambio inicial debe pasar encima de una conexión segura, como SSL. Esta operación debe devolver una muestra de alguna clase para el visitante que usarán en llamadas subsecuentes. Esta muestra son las credenciales que identifican al visitante. Debe dar las muestras en breve existencia para que tenga sentido. Para los datos muy seguros, la muestra puede permanecer sólo unos segundos. Para los datos menos seguros, la muestra puede permanecer durante una hora o más.

Como el proveedor de un servicio Web, debe decir a los clientes qué esperar para un error cuando sus muestras so n puestas invalidas. Cuando la vida de la muestra se acaba, el usuario tendrá que iniciar sesión para conseguir una nueva, muestra válida nueva para que pueda continuar usando el servicio Web.

El listado 2 describe cómo podría lograr esta tarea. La muestra usa un URL de HTTP sencillo y no revisa SSL. Si éste fuera un sistema de producción, realizaría pasos extras para suprimir el acceso anónimo y activar SSL para IIS. Para usar

SSL, necesitaría un certificado desde una autoridad de certificados en la que usted y sus usuarios podrían confiar. Para aplicaciones internas, ésta podría ser una máquina que ejecute Microsoft Certificate Server. Para aplicaciones externas, tendría que adquirir un certificado desde una autoridad de certificados pública confiable, como VeriSign.

El escenario rutinario para el ejemplo es simple:

- Inicio de sesión y adquiera la muestra.
- Usar la muestra, llamada HelloWorld.

El método del inicio de sesión usa un nombre de usuario duro-codificado y contraseña.

Listado 2. Método de inicio de sesión de autenticación personalizado

```
<WebMethod()> Public Function Login(ByVal userName As String, _
  ByVal password As String) As String

  If (userName = "Admin") And _
    (password = "simplePW") Then
    Login = theToken
  Else
    Throw New System.Web.Services.Protocols.SoapException( _
      "Invalid username/password combination", _
      System.Web.Services.Protocols.SoapException. _
        ClientFaultCode)
  End If
End Function
```

La muestra es una cadena constante. Usando esa muestra, el usuario llamaría HelloWorld pasando la muestra en el Header SOAP. El listado 3 describe cómo el propio servicio pudiera verificar la muestra y hacerlo seguro emparejando Header de las credenciales requeridas.

Listado 3. La clase muestra y la declaración HelloWorld que utiliza esa muestra.

```
Public Class TokenHeader
  Inherits System.Web.Services.Protocols.SoapHeader

  Public theToken As String

End Class

Public m_tokenHeader As TokenHeader
```

```
Private Const theToken As String = "this_is_the_token"

<WebMethod(), _
  System.Web.Services.Protocols.SoapHeader("m_tokenHeader", _
  Direction:=System.Web.Services.Protocols.SoapHeaderDirection.In,
_
  Required:=True)> Public Function HelloWorld() As String

  If (m_tokenHeader.theToken = theToken) Then
    HelloWorld = "Hello World!"
  Else
    Throw New System.Web.Services.Protocols.SoapException( _
      "You must login first", _
      System.Web.Services.Protocols.SoapException.ClientFaultCode)
  End If
End Function
```

Para ver esto en acción, podemos usar una aplicación de consola simple, como describe en el listado 4. Esta aplicación apenas iniciará sesión y, con la muestra devuelta, llamada HelloWorld. Utiliza el URI

http://schemas.xmlsoap.org/soap/actor/next

Este, especial para indicar que el mustUnderstand se refiere al destinatario del mensaje.

Listado 4. Un cliente simple mediante la autenticación de la aplicación

```
Sub Main()
  Dim svc As New localhost.Authenticate()

  Dim theToken As String
  Try
    Dim userName As String
    Dim password As String

    ' For this sample, we already know
    ' the username and password.
    userName = "Admin"
    password = "simplePW"

    theToken = svc.Login(username, password)
    If (theToken.Length > 0) Then
      ' We were authenticated. Call HelloWorld

      ' First, setup the token header
      Dim theHeader As New localhost.TokenHeader()
      theHeader.theToken = theToken
```

```
      theHeader.MustUnderstand = True
      ' Set the actor to say who must understand.
      ' Yes, it's the one who receives the message.
        theHeader.Actor = _
          "http://schemas.xmlsoap.org/soap/actor/next"
      svc.TokenHeaderValue = theHeader
      System.Console.WriteLine(svc.HelloWorld())
    End If
  Catch ex As System.Web.Services.Protocols.SoapException
    System.Console.WriteLine(ex.Detail)
  End Try
  System.Console.WriteLine("Press return to exit")
  System.Console.ReadLine()
End Sub
```

Si todo trabaja correctamente, la consola debe tener el texto "Hola Mundo!" en ella. Para escribir una versión vivida más larga del cliente del listado 4, almacenaría la muestra y sólo la cambia cuando un método de servicio Web indicando que la muestra no fue más suelta válida. La tendencia baja de este enfoque es que alguien con un rastreador de paquete y la habilidad para mirar una parte de la red pudieran capturar la muestra e inicio que la utiliza. Si cuida esto, podría correr la comunicación entera sobre SSL para que la muestra sea encriptada. Todo esto depende de que tan seguros necesita tener los datos.

6.2.2. Seguridad en el nivel del mensaje

La seguridad a nivel de mensaje (terminal a terminal) representa el enfoque más flexible y poderoso y es usado por la iniciativa de GXA (Global XML Web Services Architecture), específicamente dentro de la especificación de WS-Security. Seguridad a nivel de mensaje se ilustra en la figura.

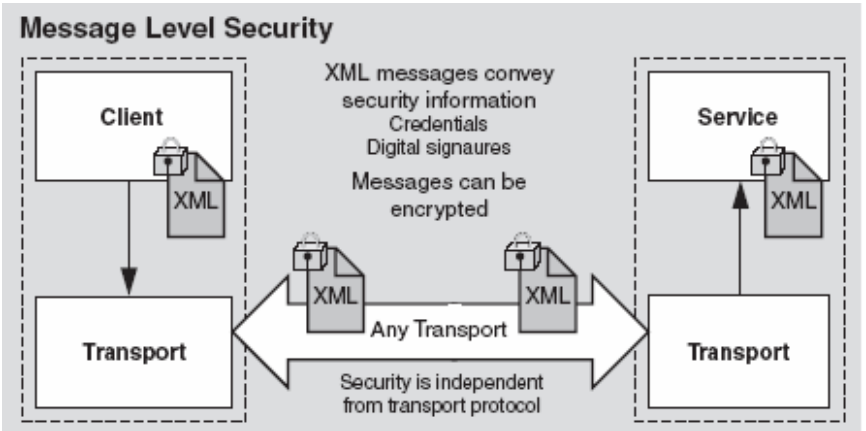


Figura. Seguridad a nivel de mensaje

Las características técnicas del WS-Security describen las mejoras para mensajería SOAP que proporciona integridad de mensaje, confidencialidad del mensaje y la única autenticación del mensaje.

- La autenticación es proporcionada por muestras de seguridad que fluyen en el encabezado de SOAP. Ningún tipo específico de muestra es requerido a través de WS-Security. Las muestras de seguridad pueden incluir tickets Kerberos, certificados X.509 o una muestra binaria personalizada.

- La comunicación segura es proporcionada por firmas digitales para asegurar integridad de mensaje y encriptación de XML para la confidencialidad del mensaje.

Uso de certificados cliente X509

Los certificados cliente X509 presenta aún otra manera de autenticar los usuarios. Típicamente, un certificado es emitido por una entidad llamada Autoridad de Certificados (CA). Un ejemplo de una CA pública es Verisign.

Muchas compañías usan Microsoft Certificate Server como una CA interna. Certificados hacen uso de infraestructura de clave pública (PKI, Public Key Infrastructure) para encriptar los datos. Usan la clave privada/pública para que empate asegurando los datos. El remitente de los datos encripta los datos mediante una clave privada que sólo el remitente conoce. Los datos pueden descifrarse utilizando su clave pública. Este arreglo le permite al destinatario de los datos averiguar que los datos fueron enviados por un participante conocido. Si la clave privada se otorga, el dueño del certificado necesita invalidar el certificado y conseguir un nuevo. Esto tiene un manejo para servicios Web.

Como sabe, ASP.NET transmite peticiones de SOAP mediante HTTP. SSL, una tecnología de HTTP extensamente adoptada, usa típicamente certificados de servidor para garantizar que un cliente esté conversando con el exigido. Los clientes averiguan esto solicitando el certificado del servidor y asegurando que pueden descifrarse los datos encriptados por el servidor con la clave pública. Si es requerido, el servidor puede solicitar un certificado del cliente que verifique la identidad del cliente mediante las mismas técnicas.

Para preparar esto en el servidor, la primera situación que querrá hacer es activar SSL en el servidor. Sin SSL, el intercambio del certificado no sucederá. Dos manejos deben realizarse:

1. La preparación de la autoridad del certificado.

Para ejecutar SSL en el servidor, necesitará tener un certificado para demostrar la identidad del servidor. Si quiere experimentar con SSL sin pagar una autoridad de certificados, como Verisign para un certificado, que puede preparar su propia autoridad de certificados.

Después de que SSL está establecido, se realizan acciones en el servidor y cliente. Normalmente, cuando está en un cliente, como Internet Explorer, que instalará un certificado del cliente para que IE pueda enviarlo a voluntad. Al desarrollar una aplicación del cliente, no tiene el acceso a los certificados que IE sabe. En cambio, el certificado necesita ser almacenado en un archivo. Después de que esto se hace, se carga el archivo en tiempo de ejecución y lo agrega a la lista de certificados que son usados por el proxy. El listado 5 describe la modificación del listado 1 visto anteriormente, para usarse con los certificados del cliente.

Listado 5. Uso de certificados cliente para la autenticación de un servicio Web simple, con configuración para SSL.

```
Sub Main()
  Dim svc As New localhost.Service1()
  ' Load the certificate from a file.
  Dim x509 As X509Certificate = _
    X509Certificate.CreateFromCertFile("c:\example.cer")

  ' Add the certificate to the service cache
  svc.ClientCertificates.Add(x509)

  ' Show the results and exit
  Try
    System.Console.WriteLine(svc.HelloWorld())
  Catch ex As Exception
    System.Console.WriteLine(ex.ToString())
  End Try
  System.Console.WriteLine("Press return to exit")
  System.Console.ReadLine()
End Sub
```

Los requisitos para que esto trabaje es simple, una autoridad de certificados que proporcione las confianzas al servidor para emitir el certificado del cliente. Éste es el mismo requisito que el cliente pone en el servidor al decidir confiar en el servidor. Como en el lado del servidor la autenticación es toda manejada por IIS, el propio Servicio Web no cambia. Las cosas que cambian es el cliente y la manera como él se despliega.

2. La preparación de IIS para los clientes certificados.

Se tiene que preparar IIS para aceptar un certificado del cliente o solicitado un certificado para la autenticación del cliente. Para que trabajen los certificados X509, tiene que preparar el cliente y el servidor de IIS correctamente. Haciendo esto consiste en dos pasos más grandes obtener un certificado del cliente y mapear ese certificado a una cuenta de Windows.

6.2.3. Actividades de aprendizaje

Aplicación con seguridad SOAP personalizado

Instrucciones

1. Realizar el ejercicio “6.3 Ejercicio aplicación con seguridad SOAP”

2. Similar al ejercicio previo, implemente los mismos mecanismos de seguridad para la aplicación elaborada en la sección “Actividades de aprendizaje 1.6”.

- Coloque su implementación en “Tarea Individual Ejercicio (6b)”

3. Contestar el cuestionario “Preguntas seguridad y servicios Web”

- Coloque sus respuestas en “Tarea Individual Cuestionario (6a)”.

6.3. Ejercicios prácticos

Ejercicio aplicación con seguridad SOAP

Antes de realizar este ejercicio revisar las "Actividades de Aprendizaje 6.2"

En base al Material de Apoyo “6. Seguridad en Servicios Web” , analice y compile el proyecto EmployeesService con implementaciones de seguridad, proporcionado en “Código aplicación con medios de seguridad” del anexo C

6.4. Cuestionarios

Tarea Asignada por el profesor.

Anexos

A. Artículo sobre proactividad

Introducción

Esta pequeña narración, "Un Mensaje a García", fue escrita en una sola hora, por la tarde después de la comida. Esto sucedió el 22 de febrero de 1899, día en que se conmemora el natalicio de Washington. La edición correspondiente al mes de marzo de la revista "Philistine" iba a entrar en prensa.

Nació como brote entusiasta de mi corazón, escrito después de un día en que había agotado mis fuerzas tratando de convencer a algunos aldeanos indolentes, para que abandonasen su estado comatoso, por una actividad radial.

Pero la verdadera inspiración brotó al calor de la discusión, mientras bebía una taza de té, con mi hijo Bert, quien sostenía que el verdadero héroe de la Guerra de Cuba había sido Rowan, quien, por sí solo, había realizado la más importante hazaña: había llevado El Mensaje a García.

Fue una idea inspiradora. Mi hijo tenía razón, porque efectivamente había sido un verdadero héroe el realizador de aquella obra, el que había llevado el mensaje a García. Me levanté y escribí el relato.

Tan poco importante me pareció el artículo así realizado, que lo publiqué sin título. Salió la edición y en breve vinieron peticiones por mayor número de ejemplares de la edición de marzo de "Philistine", una docena, cincuenta, cien. Cuando la Compañía de Noticias Americanas pidió mil ejemplares, pregunté a mis ayudantes cuál era el artículo que había conmovido en tal forma al público. Este era el artículo sobre García.

Al día siguiente George H. Daniels, del Ferrocarril Central de Nueva York, nos mandó el siguiente telegrama: "Coticen precio cien mil ejemplares de artículo Rowan en forma de panfleto, con aviso del Empire State Express al final y digan en qué fecha pueden entregarlos".

Contesté dando el precio y añadí que entregaríamos los folletos en dos años. Nuestros talleres eran entonces muy pequeños y cien mil folletos nos parecían una enormidad.

El resultado fue que hube de autorizar al señor Daniels para que reimprimiera el artículo como quisiera. Así salió medio millón de ejemplares, en forma de folleto.

Por dos o tres veces más lo reprodujo el señor Daniels, en cantidad de medio millón y más de doscientos periódicos y revistas lo reprodujeron también. Posteriormente fue traducido a todas las lenguas.

Cuando el señor Daniels distribuía el "Mensaje a García", estaba aquí el Príncipe Hilakoff, Director los Ferrocarriles de Rusia. Era huésped del Ferrocarril Central de Nueva y ork y el señor Daniels lo acompañó en su viaje a través del país. El Príncipe vio el artículo y se interesó por él, probablemente no por otra cosa que por estarlo distribuyendo tan en grande el señor Daniels. Sea de ello lo que se quiera, cuando regresó a su país, lo hizo traducir al ruso y dio un ejemplar a cada empleado de los ferrocarriles de Rusia.

Otros países siguieron el ejemplo y de Rusia pasó a Alemania, a Francia, a España, a Turquía, al Indostán y a China.

Durante la guerra entre Rusia y el Japón, cada soldado llevaba consigo un ejemplar del "Mensaje a García". Los japoneses encontraron estos folletos en manos de los prisioneros y, pensando que tendrían algún mérito, los tradujeron al japonés. Y por orden del Mikado se dio un ejemplar a cada empleado del gobierno japonés, civil o militar.

"Un Mensaje a García" ha sido impreso, pues, en más de cuarenta millones de ejemplares, suma que jamás ha alcanzado publicación alguna, quizá gracias a una serie de incidentes afortunados.

Un Mensaje a García

Hay en la historia de Cuba un hombre que destaca en mi memoria como Marte en Perihelio.

Al estallar la guerra entre los Estados Unidos y España, era necesario entenderse con toda rapidez con el jefe de los revolucionarios de Cuba.

En aquellos momentos este jefe, el general García, estaba emboscado en las asperezas de las montañas: nadie sabía dónde. Ninguna comunicación le podía llegar ni por correo ni por telégrafo. No obstante, era preciso que el presidente de los Estados Unidos se comunicara con él. ¿Qué debería hacerse?

Alguien aconsejó al Presidente: "Conozco a un tal Rowan que, si es posible encontrar a García, lo encontrará".

Buscaron a Rowan y se le entregó la carta para García.

Rowan tomó la carta y la guardó en una bolsa impermeable, sobre su pecho, cerca del corazón.

Después de cuatro días de navegación dejó la pequeña canoa que le había conducido a la costa de Cuba. Desapareció por entre los juncales y después de tres semanas se presentó al otro lado de la isla: había atravesado a pie un país hostil y había cumplido su misión de entregar a García el mensaje de que era portador.

No es el objeto de este artículo narrar detalladamente el episodio que he descrito a grandes rasgos. Lo que quiero hacer notar es lo siguiente: Mc Kinley le dio a Rowan una carta para que la entregara a García, y Rowan no preguntó: "¿En dónde lo encuentro?"

Verdaderamente aquí hay un hombre que debe ser inmortalizado en bronce y su estatua colocada en todos los colegios del país.

Porque no es erudición lo que necesita la juventud, ni enseñanza de talo cual cosa, sino la inculcación del amor al deber, de la fidelidad a la confianza que en ella se deposita, del obrar con prontitud, del concentrar todas sus energías; hacer bien lo que se tiene que hacer. "Llevar un Mensaje a García".

El general García ha muerto; pero hay muchos otros Garcías en todas partes.

Todo hombre que ha tratado de llevar a cabo una empresa para la que necesita la ayuda de otros, se ha quedado frecuentemente sorprendido por la estupidez de la generalidad de los hombres, por su incapacidad o falta de voluntad para concentrar sus facultadas en una idea y ejecutarla.

Ayuda torpe, craso descuido, despreciable indiferencia y apatía por el cumplimiento de sus deberes: tal es y ha sido siempre la rutina. Así, ningún hombre sale avante, ni se logra ningún éxito si no es con amenazas o sobornando de cualquier otra manera a aquellos cuya ayuda es necesaria.

Lector amigo, tú mismo puedes hacer la prueba.

Te supongo muy tranquilo, sentado en tu despacho y alrededor seis empleados dispuestos todos a servirte. Llama a uno de ellos y hazle este encargo: "Busque, por favor, en la enciclopedia y hágame un breve memorándum acerca de la vida del Correggio".

¿Esperas que tu empleado con toda calma te conteste: "Sí, señor", y vaya tranquilamente a poner manos a la obra?

¡Desde luego que no! Abrirá desmesuradamente los ojos, te mirará sorprendido y te dirigirá una o más de las siguientes preguntas:

¿Quién fue? ¿Cuál enciclopedia?
¿Eso me corresponde a mí?
Usted quiere decir Bismark, ¿no es cierto?
¿No sería mejor que lo hiciera Carlos? ¿Murió ya?
¿No sería mejor que le trajera el libro para que usted mismo lo buscara? ¿Para qué lo quiere usted saber?

Apuesto diez contra uno, a que después de haber contestado a tales preguntas y explicado cómo hallar la información que deseas y para qué la quieres, tu dependiente se marchará confuso e irá a solicitar la ayuda de sus compañeros para "encontrar a García". Y todavía regresará después para decirte que no existe tal hombre. Puedo, por excepción, perder la apuesta; pero en la generalidad de los casos, tengo muchas probabilidades de ganarla.

Si conoces la ineptitud de tus empleados no te molestarás en explicar a tu "ayudante", que Correggio se encuentra en la letra C y no en la K. Te limitarás a sonreír e irás a buscarlos tú mismo.

No parece sino que es indispensable el nudoso garrote y el temor a ser despedido el sábado más próximo, para retener a muchos empleados en sus puestos. Cuando se solicita un taquígrafo, de cada diez que ofrezcan sus servicios, nueve no sabrán escribir con ortografía y algunos de ellos considerarán este conocimiento como muy secundario.

¿Podrá tal persona redactar una carta a García?

-¿Ve usted este tenedor de libros?

Me decía el administrador de una gran fábrica.

-Sí, ¿por qué?

-Es un gran contador, pero si le confío una comisión, sólo por casualidad la desempeñará con acierto. Siempre tendré el temor de que en el camino se detenga en cada cantina que encuentre y cuando llegue a la Calle Real, haya olvidado completamente lo que tenía que hacer.

¿Crees, querido lector, que a tal hombre se le puede confiar Un Mensaje para García?

A últimas fechas es frecuente escuchar que se excita nuestra compasión para los enternecedores lamentos de los desheredados, esclavos del salario, que van en busca de un empleo. Y esas voces a menudo van acompañadas de maldiciones por los que están "arriba".

Nadie compadece al patrón que envejece antes de tiempo, por esforzarse inútilmente para conseguir que el aprendiz chambón ejecute bien un trabajo. Ni nos ocupamos del tiempo y paciencia que pierde en educar a sus empleados para que estén en aptitud de realizar su trabajo, empleados que flojean en cuanto vuelve la espalda.

En todo almacén o fábrica se encuentran muchos zánganos, y el patrón se ve obligado a despedir a sus empleados todos los días, por su ineptitud para defender los intereses de la negociación. Y a cada despedido siguen y seguirán muchos iguales.

Esta es invariablemente la historia que se repite en tiempos de abundancia. Pero cuando, por efecto de las circunstancias, escasea el trabajo, el jefe tiene oportunidad de escoger cuidadosamente y de señalar la puerta a los ineptos y a los holgazanes.

Por propio interés, cada patrón procura conservar lo mejor que encuentra; es decir, a aquellos que pueden llevar Un Mensaje a García.

.

Conozco un individuo que se halla dotado de cualidades y aptitudes verdaderamente sorprendentes; pero carece de la habilidad necesaria para manejar sus propios negocios y que es absolutamente inservible para los demás. Sufre la monomanía de que sus jefes lo tiranizan y tratan de oprimirlo. No sabe dar órdenes ni quiere recibirlas.

Si se le confía Un Mensaje a García probablemente contestará "llévelo usted mismo".

Actualmente este individuo recorre las calles en busca de trabajo, sin más abrigo que un deshilachado saco por donde el aire se cuela silbando. Nadie que lo conozca accederá a darle empleo. A la menor observación que se le hace monta en cólera y no admite razones: sería preciso tratarlo a puntapiés, para sacar de él algún partido.

Convengo de buen grado en que un ser tan deforme, bajo el punto de vista moral es digno cuando menos de la misma compasión que nos inspira un lisiado físicamente. Pero en medio de nuestro filantrópico enternecimiento, no debemos olvidar derramar una lágrima por aquellos que se afanan en llevar a cabo una gran empresa; por aquellos cuyas horas de trabajo son ilimitadas, pues para ellos no

existe el silbato; por aquellos que a toda prisa encanecen, a causa de la lucha constante que se ven obligados a sostener contra la mugrienta indiferencia, la andrajosa estupidez y la negra ingratitud de los empleados que, si no fuera por el espíritu emprendedor de estos hombres, se verían sin hogar y acosados por el hambre.

¿Son demasiados severos los términos en que acabo de expresarme? Tal vez sí. Pero cuando todo mundo ha prodigado su compasión por el proletario inepto, yo quiero decir una palabra de simpatía hacia el hombre que ha triunfado, hacia el hombre que, luchando con grandes obstáculos, ha sabido dirigir los esfuerzos de otros, y, después de haber vencido, se encuentra con que lo que ha hecho no vale nada; sólo la satisfacción de haber ganado su pan.

Yo mismo he cargado el portaviandas y trabajado por el jornal diario; y también he sido patrón de empresa, empleado "ayuda" de la misma clase a que me he referido, y sé bien que hay argumentos por los dos lados.

La pobreza en sí, no reviste excelencia alguna. Los harapos no son recomendables ni recomiendan por ningún motivo. No son todos los patrones rapaces y tiranos, ni tampoco todos los pobres son virtuosos.

Admiro de todo corazón al hombre que cumple con su deber, tanto cuando está ausente el jefe, como cuando está presente. Y el hombre que con toda calma toma el mensaje que se le entrega para García, sin hacer tantas preguntas, ni abrigar la aviesa intención de arrojarlo en la primera atarjea que encuentre, o de hacer cualquier otra cosa que no sea entregarlo, jamás encontrará cerrada la puerta, ni necesitará armar huelgas para obtener un aumento de sueldo.

Ésta es la clase de hombres que se necesitan y a la cual nada puede negarse. Son tan escasos y tan valiosos, que ningún patrón consentirá en dejarlos ir.

A un hombre así, se le necesita en todas las ciudades, pueblos y aldeas, en todas las oficinas, talleres, fábricas y almacenes. El mundo entero clama por él, se necesita, ¡¡urge... el hombre que pueda llevar un mensaje a García!!

Helbert Hubbard.