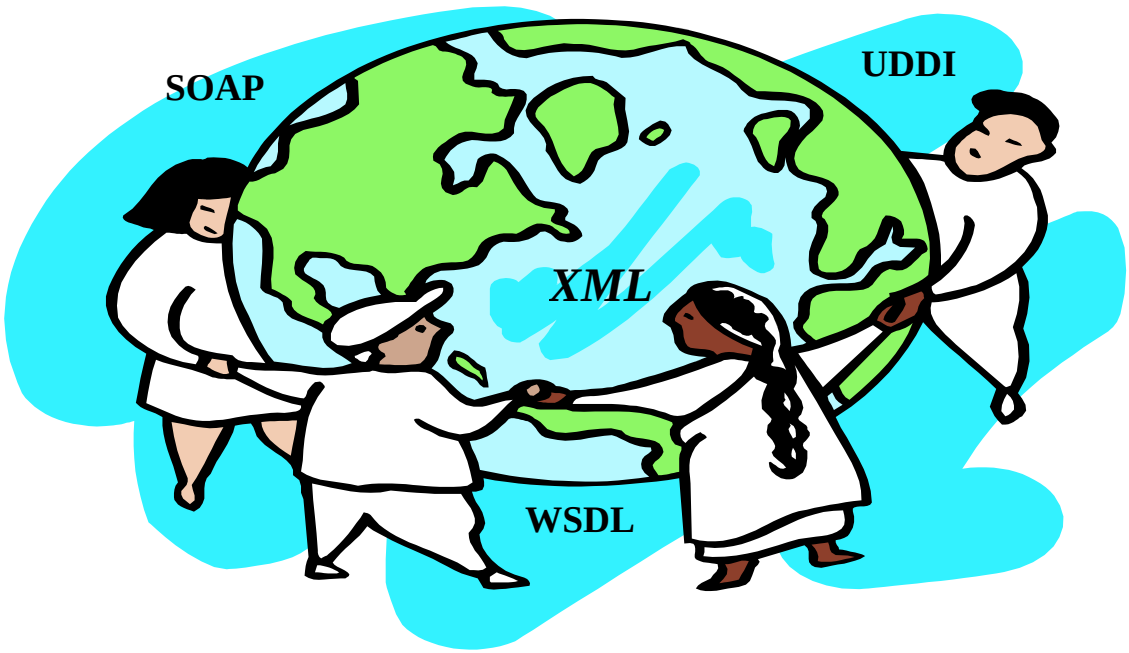
ARQUITECTURAS ORIENTADAS A WEB SERVICES



Director **Lic. Francisco Javier Díaz**
Alumna: **C.C. Lía Molinari**

Abril 2004

Indice

Las tendencias que se concretan son aquellas que surgen en respuesta a necesidades. Pueden ser necesidades futuras, previstas por aquéllos que son capaces de ver el próximo paso, o las que plantea hoy y ahora un usuario cada vez más exigente e insatisfecho.

Los que tenemos muchos años en Informática hemos asistido al surgimiento de distintos modelos. En algunos casos, quedaron en el recuerdo como “modas” que algunas empresas intentaron imponer.

Pero todos contribuyeron a formar la base para construir la innovación tecnológica a la que hoy asistimos, y disfrutamos, a través de la Red.

En invierno del 2003, tuve la oportunidad de concurrir a la conferencia que ofreció Microsoft sobre .NET con la participación cuasi estelar de Miguel de Icaza (XIMIAN-Mono) en la Universidad CAECE, Capital Federal, Buenos Aires.

Cuando dos mundos tan disímiles convergen en una tecnología, y exponen esa posibilidad de interoperabilidad, se hace imprescindible “husmear” por allí, porque algo está pasando. Y en ese momento, me alegré de haber aceptado la sugerencia del Lic. Francisco Javier Díaz de tratar este tema en la tesis. Por que, más allá del título elegido, esta tesis trata sobre la interoperabilidad.

Por otra parte, si bien hay mucho material sobre el tema, gran parte de él trata sobre la implementación de Web Services en distintas plataformas o productos, lo que a veces confunde sobre cuáles usan protocolos standards y cuáles no lo son.

Esta tesis trata sobre qué son los Web Services, la tecnología que es necesaria para su implementación y el uso de protocolos standards. Se incluyen algunos capítulos para ver como enfrentan el tema grandes empresas y sus productos asociados (Microsoft .NET, IBM Websphere, SUN ONE y Mono, como alternativa de implementación sobre LINUX).

El desarrollo de esta tesis me exigió revisar conceptos y aprender muchos otros. Y agradezco como esa exigencia contribuyó y enriqueció mi actividad docente.

En la Cátedra Sistemas Operativos de la Maestría de Redes de Datos donde colaboro como docente, tengo la posibilidad de interactuar con profesionales que provienen de distintas áreas de la Informática. Tienen diferente formación, y en la mayoría de los casos, Web services y middleware es un concepto novedoso. Ojalá que mi trabajo contribuya a generar futuros trabajos.

Aclaración sobre la Bibliografía

La bibliografía y referencias se encuentran al final de documento (ver 15).

La forma de individualizarla es un código con la forma **XXXnnn**, donde,

- XXX, son las tres primeras letras del apellido del autor, o empresa que publica
- nnn, año de publicación.

Al final de cada capítulo, en *Notas*, se indica la bibliografía y referencias utilizadas.

A G R A D E C I M I E N T O S

- al Lic. Francisco Javier Díaz, por su guía permanente que me ha permitido crecer profesionalmente. Gracias a su apoyo, me impuse metas que hasta ahora me parecían imposibles;
- a mis colegas, amigos y alumnos, que interesándose por este trabajo, me infundaron el ánimo para concretarlo;
- a mi familia (Sofía , Mara, Franco y Mandy), que asistió a mis horas de enajenación ante la PC, y que comprendió que el tiempo que les robaba, me ayudaba a crecer.

Capítulo I

El surgimiento de un nuevo modelo

1 El surgimiento de un nuevo modelo

La Arquitectura de los sistemas de cómputo evolucionaron para adaptarse a las nuevas formas de trabajo.

Hace unos años las grandes corporaciones y organismos públicos contaban con mainframes, grandes computadoras con gran capacidad de computo y proceso centralizado, donde la interacción con el usuario se daba a través de las llamadas terminales bobas, sin capacidad propia de computo ni almacenamiento secundario.

Con el abaratamiento de las PC's las organizaciones tuvieron la posibilidad de contar con varias de ellas. El inconveniente que surgió fue la duplicación de la información (se replicaban archivos y bases) con el consecuente problema de inconsistencia. Además, el alto costo de otros dispositivos como las impresoras Láser o grabadoras DAT, etc., no permitían asociar una de estas a cada PC y por lo tanto se hacia necesario transferir la información a imprimir o grabar a través de un diskette a la maquina que contaba con estos periféricos.

La interconexión física de estas PC's fue el próximo paso. Pero... cómo era la visión del usuario en esta interconexión? Podía él acceder a los archivos ubicados en otra máquina como si fueran locales? Qué capacidad de abstracción de origen le permitía?

No bastaba la interconexión física.

Es importante además, citando la definición de sistemas operativos que hace Silberschatz (SIL003), no olvidar que el objetivo de un sistema operativo es el manejo eficiente de los recursos, de una manera cómoda para el usuario.

En las empresas que contaban con mainframes, la llegada de las PCs permitieron que cada usuario cuente con una de ellas aprovechando, por un lado, su capacidad de computo y almacenamiento secundario y, a la vez, utilizarla como terminal del sistema central, donde residen las bases de datos corporativas, sistemas de gestión, etc. Cada PC tiene su propio sistema operativo. El próximo paso fueron los sistemas cliente servidor. Y comenzó el camino hacia la interoperabilidad.

Ya no podía ignorarse a Internet y su potencial. La tendencia se orientó al modelo distribuido sobre la red.

Un ***sistema distribuido*** es un sistema donde sus componentes, ubicados en distintas computadora conectadas en red, se comunican y coordinan a través de mensajes. Hay concurrencia de componentes, carencia de reloj global y fallos independientes de los componentes.

En la referencia COU001, Coulouris explica de una forma ágil y clara estos conceptos, y los diferentes modelos y tendencias.

Ejemplos: Internet, Intranet, computación móvil.

Básicamente, el objetivo es compartir recursos.

Los recursos pueden ser administrados por servidores y accedidos por clientes o encapsularse como objetos y accedidos por objetos clientes.

Un ***servicio*** es una parte del sistema de computadoras que gestiona una colección de recursos relacionados presentando su funcionalidad a usuarios y aplicaciones.

Un navegador (browser) es un ejemplo de cliente: se comunica con el servicio WEB para solicitarle páginas.

El concepto de distribución sobre la red, implica el concepto de ***heterogeneidad***. La heterogeneidad (variedad y diferencia) en estos sistemas se aplica a :

- Redes
- HW de computadoras
- SO
- Lenguajes
- Implementación de desarrolladores

Hoy, gran parte de las organizaciones el ambiente es multiplataforma, es decir, conviven distintas arquitecturas con distintos sistemas operativos. Cómo ofrecer un ambiente integrado?

1.1 Conceptos relacionados

Extensibilidad: determina si el sistema puede ser extendido y reimplementado, como añadir nuevos servicios para compartir recursos.

Escalabilidad: es aquel sistema que mantiene su funcionalidad cuando aumentan el número de recursos y usuarios significativamente.

Tratamiento de fallos: En un SD hay componentes que fallan y otros que siguen trabajando. Es importante proveer *detección de fallos, enmascaramiento de fallos, tolerancia, recuperación, redundancia*.

- ✓ *Enmascaramiento* (ocultar o atenuar) pueden lograrse retransmitiendo mensajes o a través de espejados de discos.
- ✓ *Tolerancia*: es que, a veces no se pueden detectar y ocultar todos los fallos. Por ejemplo: no se puede mostrar una página.
- ✓ *Recuperación:* hacer software que permita hacer roll back ante la caída del servidor (llevarse a un estado anterior).
- ✓ *Redundancia:* poner más componentes (al menos dos rutas diferentes entre 2 routers, tablas de nombres copiadas en 2 servdiores DNS diferentes.
- ✓ *Transparencia:* Es clave en un ambiente distribuido que el usuario tenga la "sensación" de estar trabajando en un único sistema, abstrayendo se de la cantidad de computadoras, los vinculo s y la ubicación de los archivos. Cuando se logra este objetivo, se dice que el sistema es *transparente.* Transparencia es, entonces, la ocultación al usuario y al programador de la separación de componentes en un sistema distribuido. De esa manera, se ve al sistema como un todo más que como una colección de componentes independientes.

1.2 Modelos. Concepto y alternativas

Modelo de arquitectura define la forma en que los componentes de un sistema interactúan uno con otro y se vinculan con la red subyacente. El modelo debe tener en cuenta la ubicación de las componentes y la interrelación entre ellos (como funcionan y comunican).

De acuerdo a la ubicación definiré como distribuyo datos y carga de trabajo.

Arquitectura de soft se refiere a la estructuración del soft en capas o módulos dentro de una computadora. Hoy hablamos de servicios ofrecidos y solicitados entre procesos dentro o en diferentes computadoras. De allí que se hable de capas de servicio.

La plataforma es el nivel de HW y capas más bajas de soft. Estas capas dan servicio a las que están por encima de ellos.

El *middleware* es la capa de soft que enmascara la heterogeneidad ofreciendo al programador de aplicaciones un ambiente unificado. El middleware ofrece un modelo uniforme al programador de servidor y aplicaciones distribuidas.

Sobre el concepto de middleware volveremos más adelante.

1.3 Arquitecturas de sistemas

La responsabilidad de cada componente y su ubicación es uno de los aspectos más importantes en el diseño de los sistemas distribuidos.

Veamos como interactúan esas componentes, ubicados en distintas computadoras para realizar una tarea útil.

- Modelo cliente servidor
- Servicios proporcionados por múltiples servidores
- Servidores Proxys y cachés
- Procesos de igual a igual

En el caso del **modelo cliente servidor** se definen entidades clientes, servidoras y servicios.

Veamos un ejemplo para aclarar el concepto de servicio remoto.

Puede ser que un usuario necesite hacer un procesamiento de un archivo que está en otra computadora. Por ejemplo, puede ser un archivo que contiene datos que son números y el usuario necesita estadísticas de ese conjunto.

Una alternativa sería solicitar que se transfiera el archivo a la computadora y hacer la estadística de manera local (migración de datos). La otra es solicitar el procesamiento a la computadora remota, diciendo que se quiere hacer y sobre que archivo. La computadora remota procesa y envía la respuesta (migración de procesamiento).

Una forma de implementar esta última alternativa es a través de *servicios remotos.*

Hay una maquina remota con una aplicación servidor que atiende los requerimientos de acceso provenientes de otras maquinas. La aplicación servidor atiende el pedido del usuario remoto, lo procesa y le devuelve las resultados.

Consideremos que si esta aplicación servidor tiene muchos clientes, esta interacción generará trafico en la red, desde y hacia el servidor.

Ejemplo

1. Un servidor WEB puede ser cliente de servidor, por ejemplo, un servidor de archivos local que administra los archivos donde se almacenan las páginas WEB.
2. Un servidor WEB es cliente del servicio de DNS, que traduce nombres de dominio en direcciones de Internet.
3. Los buscadores que permiten buscar información de resumen de página WEB en sitios de Internet.

Una máquina de búsqueda es tanto cliente como servidor. Como servidor, atiende los requerimientos que hacen los clientes a través del navegador, y a su vez, ejecuta web crawlers que actúan como clientes de otros servidores.

Los web crawlers (o escaladores web) son programas que realizan los resúmenes que son presentados a los buscadores.

De acuerdo a este ejemplo vemos que los requerimientos de los clientes, si bien van dirigidos a un servidor, existen distintos procesos de servidor interaccionando para proporcionar ese servicio.

Este sería el modelo de **servicios proporcionados por múltiples servidores**.

Ahora hablemos sobre el uso de la cache que necesitan estos procesos.

Una caché es un almacenamiento de objetos de datos que se usaron recientemente y que se encuentra más próximo que los objetos en sí.

Cuando un proceso cliente necesita un objeto, el servicio caché comprueba en la caché y si está, le da una copia actualizada. Si no, se busca.

Las cachés pueden ubicarse en cada cliente o en un servidor proxy compartido por varios clientes.

Los navegadores web mantienen una cache de las páginas visitadas recientemente y realizan una petición especial para comprobar si han sido actualizados en el servicio antes de mostrarlos en la pantalla.

Los **servidores proxy** para WEB proporcionan una caché compartida de recursos WEB para las máquinas cliente de 1 o más sistemas. Incrementan la disponibilidad y prestaciones del servicio. También se pueden usar para acceder a servidores WEB remotos a través de firewall.

Pero, las necesidades de comunicación actuales, exigen además un modelo que contemple la comunicación entre procesos sin pasar por un servidor.

Hay algunas aplicaciones donde el uso de un servidor aumentaría retardos para acceder a objetos locales.

Cada uno tiene aplicación y código de coordinación.

Supongamos un pizarrón interactivo donde todos los usuarios van modificando el dibujo.

Hay un proceso en cada sitio para notificar eventos y comunicar a los grupos los cambios en el dibujo. Hay una mejor respuesta interactiva que si residiera en un servidor

También, el surgimiento de los dispositivos móviles y la necesidad de comunicarse y trabajar a través de ellos, hizo surgir **variaciones del modelo C/S**.

Hoy en día, la necesidad de comunicarse y trabajar a través de dispositivos móviles, o de computadoras de bajo costo, hizo surgir variaciones del modelo C/S.

Ante toda esta tendencia, vale definir conceptos tales como código móvil y agentes móviles.

El **código móvil** es aquel que reside en un servidor WEB y ante la solicitud que realiza un browser (navegador) de un usuario es descargado desde el servidor al navegador y allí se ejecuta, en el cliente. Es el caso del applet de Java.

Obtener un servicio significa ejecutar código que es necesario invocar a través de las operaciones definidas.

La solicitud de servicios a través del navegador a un servidor a veces necesita descargar código adicional previamente desde el servidor para ciertas funciones que no son provistas por el navegador.

Este modelo, al que se llama push, exige que el servidor, no el cliente, inicie interacciones.

Un **agente móvil** es un proceso (tanto código como datos) que toma información desde una computadora según el requerimiento de alguien, pasando de una computadora a otra, para obtener un resultado.

Un uso común es la instalación y mantenimiento de soft de las computadoras de una organización.

Por ejemplo, se hizo un programa que aprovechaba las computadoras ociosas para usar su capacidad de cálculo.

Si bien son útiles y necesarios, los códigos y agentes móviles pueden ser una amenaza a la seguridad

1.4 Notas

Para este capítulo, me fueron útiles las lecturas a [COU001], un libro al que consulto asiduamente, junto con [SIL001] y [STA002], por mi actividad docente.

También [NUT992], que introdujo, en su momento, conceptos relacionados con los sistemas abiertos.

Capítulo II

El modelo distribuido

2 El modelo distribuido

2.1 Funcionalidad. Enfoques

A continuación expongo algunos ejemplos sobre lo que puede llegar a necesitar implementar el usuario, hoy, a través de la Red.

- Supongamos que un usuario cuenta con un disco de 40 GB. Necesita más capacidad de memoria secundaria. Decide, entonces comprar un disco de mayor capacidad o comprar uno adicional, con los costos (y riesgos) de instalación y configuración (aún mayor si se necesita resguardar la información almacenada hasta el momento...). En un nuevo modelo distribuido sobre la Red, a través de Internet, el usuario no tendría necesidad de mantener toda la información en su PC. Sus datos o las aplicaciones pueden estar en servidores a través de la red, con un costo mínimo de mantenimiento, y, por supuesto, acuerdos de seguridad. El usuario de esta manera, puede desinteresarse de la tarea de administración, actualizaciones del software, configuración de dispositivos, etc.

- Supongamos que un usuario desea hacer una compra importante de un producto. Desde la PC de su oficina, comienza a visitar diferentes sites para comparar precios. Más tarde, ya en otro ámbito, decide concretar la compra a través de su teléfono celular. Un modelo que prevea esta facilidad debe permitir que el cliente acceda al site con aplicaciones e-commerce través de diferentes dispositivos, como es este caso.

- Un desarrollador debe escribir una aplicación para Tareas de Investigación relacionadas específicamente con la Biotecnología. Se entera que existe una base de datos en el NCBI (National Center for Biotechnology) con artículos relacionados con este campo. No obstante no quiere que el usuario acceda al site, si no que sea su aplicación quien dialogue con la aplicación que ofrece este servicio (se pueden ver ejemplos específicos en www.alphaworks.ibm.com/tech/ws4LS). Si además, como se mostró la utilidad en el ejemplo anterior, el desarrollador quiere que su aplicación sea soportada por dispositivos móviles, como por ejemplo, palms, puede hacerlo a través de herramientas que le permitan trabajar en un ambiente interoperable.

En estos ejemplos, la Red está al servicio del usuario.

El modelo distribuido sobre la Red es hoy una necesidad. Pero para implementar ese modelo distribuido es imprescindible acordar como trabajar entre distinto hardware, distintas plataformas, integrando servicios, pero sin perder la transparencia para el usuario.

Quizás el usuario corriente, no desarrollador, o que no trabaja a partir de datos que ofrecen aplicaciones en la Red, no llegue a vislumbrar que los cambios o los

beneficios, pues la transparencia es uno de los conceptos más importantes de este modelo distribuido.

Pero desde el punto de vista del desarrollador, este modelo le ofrece la posibilidad de:

- ✓ Escribir aplicaciones que interactuarán con distintos dispositivos (aún con los por venir) sin tener que modificar su aplicación, permitiendo que su aplicación e-commerce sea accedida sin restricciones de este tipo.
- ✓ Reutilizar software escritos por otros desarrolladores
- ✓ Y, lo más importante, reutilizar ese software desinteresándose por cómo y en qué lenguaje está escrito.

Si hablamos de interacción, no podemos obviar que la solución que se presente debe considerar la heterogeneidad antes mencionada. Esta solución debe prever su implementación transparente sobre ambientes multiplataforma.

Analicemos, por ejemplo, el tema de los dispositivos.

La innovación frecuente (consideremos si no el progreso de la última década en cuanto a teléfonos inalámbricos, teléfonos inteligentes, nuevos modelos de laptops, etc.) exige que el modelo actual tanto de aplicación como de comunicación, no sea suficiente.

Hasta ahora, con cada nuevo dispositivo, era necesaria la adaptación de la aplicación para integrarlo. Las soluciones que se brinden desde el servidor deben tener en cuenta a estos nuevos dispositivos del cliente.

Una alternativa es desarrollar una solución para cada dispositivo, lo cual, obviamente, es ineficiente, considerando que la falta de escalabilidad no permitirá una rápida adaptación de la aplicación a nuevos dispositivos que puedan surgir (y surgirán) en el futuro.

Los dispositivos tienen, por ejemplo, diferentes resoluciones de pantalla. Además, en la mayoría de los casos, deben integrar datos provenientes de diferentes servidores.

El desafío es, entonces, desarrollar una aplicación que deba tener en cuenta esta variedad, y que sea reusable para nuevos dispositivos que surjan (lo cual es lo esperado).

Este objetivo está dentro de lo que se llama ***ubiquitous connectivity*** (algo así, como conectividad para todos, entre todos). En GLA000, se hace una referencia interesante de los web services como evolución y revolución. Por ejemplo, en un futuro, podemos pensar que una máquina expendedora de gaseosa, wireless, pueda contactar al proveedor para ordenar un pedido sobre una marca de gaseosa faltante. El “diálogo” se establecería entre dispositivos.

Tratando de explotar la ley de Moore (que enuncia que cada nuevo microchip aumenta al doble la capacidad de procesamiento del anterior) y el abaratamiento del ancho de banda, pensar en un modelo distribuido basado en Internet, es hoy una posibilidad cierta.

Este nuevo modelo incluye componentes de software que usan protocolos abiertos de Internet, incluyendo HTTP para transporte y XML para la representación de la información. Es un modelo orientado a servicios WEB

Cualquier dispositivo y plataforma hoy puede trabajar con estos protocolos. De allí que la adaptación al nuevo modelo sea factible a corto plazo.

Y en cuanto a la seguridad, al usar HTTP siempre se puede usar SSL, para obtener un nivel básico, que puede complementarse con servicios de autenticación.

Para adaptarse a este modelo, la nueva generación de lenguajes de computadoras debe integrar Internet de una manera transparente.

Las plataformas de lenguajes tipo Java, usan interpretadores en el momento de la ejecución para soportar las diferencias en el hardware cliente. Sus características de orientación a objetos, no logra la independencia esperada del hardware ni los niveles de performance, ni la interoperabilidad.

La Web ha evolucionado desde contenidos estáticos, como documentos, a contenidos dinámicos, usando servidores de aplicaciones usando business logic, como programas CGI o JAVA y transacciones.

También usan nuevos protocolos como WAP (Wireless Access Control) y WMP (Wireless Markup Language).

Dentro del modelo de procesos de igual a igual podemos considerar el **modelo P2P**.

El modelo cliente-cliente (o peer to peer) trata sobre la comunicación entre clientes, sin usar, imprescindiblemente, un servidor.

Como ocurre con los modelos, no es aplicable a todas las situaciones. Si el cliente debe ser autenticado o si los clientes quieren trabajar asincrónicamente, se hace necesario el uso de un servidor. También se debe tener en cuenta la capacidad de proceso del cliente: si no es suficiente, deberá recurrir a otras máquinas para el procesamiento de la información.

Para este modelo se debe garantizar que se cumplen las exigencias de un modelo integrado de distribución sobre Internet, se debe analizar:

- Manejo de identidad y autenticación del usuario y los servicios asociados
- Capacidad de trabajar en línea y fuera de línea
- Optimizar la forma de mostrar la información
- Libre uso de dispositivos
- Interacción con otros usuarios
- Servicios disponibles para el usuario de acuerdo a su personalización

Veamos una proyección futura, lo que podría ser posible. Imaginemos una versión P2P de eBay, el sitio de remates online. Un usuario podría instalar un servicio de subasta en su PC, o en su celular. Configurando este servicio indicando qué se quiere comprar o vender y las condiciones, el dispositivo se comunicaría con otros lugares en

el mundo, contactando los compradores o vendedores adecuados. No se trabajaría a través de un servidor, si no que se comunican los dispositivos de los consumidores.

O pensemos un la posibilidad de establecer una red celular sin estación base, sino que los mismo teléfonos de consumidores sean los que intermedian.

2.2 Objetos en el mundo distribuido

2.2.1 Objetos remotos

La interfaz de un proceso es la especificación del conjunto de funciones que pueden invocarse sobre él. Esto es válido tanto para C/S como para comunicación entre iguales.

En Cliente-servidor, cada proceso servidor es una entidad con una interfase definida.

Cuando tratamos con lenguajes OO, un proceso puede encapsular varios objetos.

En el modelo OO los nuevos tipos de objetos deben ser instanciados y estar disponibles rápidamente para su invocación.

Cuando el lenguaje lo permite, los procesos distribuidos se organizan como objetos. Se pueden encapsular muchos objetos y las referencias a estos objetos se van pasando a otros procesos para que puedan invocar sus métodos. JAVA, con su RMI (invocación remota de objetos) usa este mecanismo.

Es muy importante en cuanto al diseño, como se distribuyen las responsabilidades en el modelo distribuido. Esta distribución puede ser estática (por ejemplo, un servidor de archivos se responsabiliza de los archivos no de proxys, mails, o páginas WEB) o más dinámica como es el modelo orientado a objetos donde se pueden instanciar nuevos objetos y nuevos servicios inmediatamente disponibles para ser invocados.

Las referencias a esos procesos deben armarse de tal forma que puedan invocarse sus métodos desde otros procesos. Es lo que llamamos *invocación remota*.

Esa invocación es remota cuando ocurre entre objetos en diferentes procesos, estén estos en la misma computadora o en computadoras interconectadas, se llaman invocación a objetos remotos. Cuando se invocan métodos entre objetos que están en el mismo proceso, son **invocaciones locales**.

Los objetos que pueden recibir invocaciones remotas, son objetos remotos.

Para poder invocar un objeto, se debe tener una referencia a él. Y, un objeto remoto debe tener una interfaz remota que diga cuál de sus métodos puede invocarse remotamente.

Cuando se trabaja con este tipo de objetos, se debe tener en cuenta que, por ejemplo, una vez que el objeto fue borrado, no deben resolverse futuras referencias, ni derivarlas a otros objetos.

2.2.2 Interfases remotas y referencias a objetos remotos

En el ambiente C/S se habla de interfaz de servicio pues los servidores proporcionan servicios y para invocarlos debe ofrecer una especificación de los procedimientos disponibles. Define allí también los argumentos de E y de S.

Veamos como se sería la Interfase de un File Server.

A través de esta interfase se ofrecen los servicios que presta el File Server.
Es decir, los clientes solicitan los servicios invocando read, write, create o delete, con los argumentos que se especifican.

#include <header. h>

specification of file_server, version 3.1:

long read (in char name[MAX_PATH], out char buf[BUF_SIZE], in long bytes, in long position);
long write (in char name[MAX_PATH], in char buf[BUF_SIZE], in long bytes, in long position);

int create (in char[MAX_PATH], in int mode);

int delete (in char[MAX_PATH]);

end;

En el modelo de objetos distribuidos, la interfaz remota especifica los métodos de un objeto disponible a ser invocadas por otros procesos. Define también argumentos de entrada y de salida.
Pero lo que hay que tener en cuenta, y es lo que lo diferencia del otro modelo, es que los métodos de interfases remotas pueden tener objetos como argumento y como resultado. También se pueden pasar referencias a objetos remotos.

Cuando un proceso cliente quiere invocar un método de un objeto remoto, debe asegurarse de estar invocando el método deseado. El proceso invocante envía un mensaje de invocación al servidor donde está el proceso que alberga el objeto remoto.

El mensaje especifica el objeto remoto de una manera que debe ser válida para el sistema distribuido. En un ambiente así, hay distintos procesos que alojan objetos factibles de ser invocados remotamente. La forma de referenciarlos debe asegurar la unicidad.

Una referencia a un objeto remoto es un identificador que es válido dentro del sistema distribuido.

Las referencias pueden pasarse como argumentos y pueden devolverse como resultado de la invocación a métodos remotos. Incluso las referencias pueden compararse para saber si se refieren al mismo objeto.

No sólo la referencia debe asegurar la unicidad con respecto a la ubicación: también es importante considerar el tiempo, es decir, la posibilidad de estar invocando una versión que no sea la más reciente.

Una forma estática de construir una referencia, sería incluir la dirección de Internet de la computadora donde reside, numero de puerto del proceso que lo creó y momento de creación y un número de objeto local. Este último número es actualizado cada vez que el proceso crea un nuevo objeto.

El problema es la falta de flexibilidad de este esquema. Qué pasa si los objetos son "reubicados"? Qué pasa con la deseada transparencia de ubicación o de migración?

Se pueden utilizar un servicio de localización, que utiliza una base de datos que relaciona objetos remotos con sus direcciones actuales, o por lo menos, las últimas direcciones reportadas.

Se llama *acción* a una cadena de invocaciones de métodos relacionados, cada uno de los cuales retornara en algún momento.

2.2.3 Activación/desactivación de objetos. Garbage collection.

Un objeto remoto está activo cuando está disponible para su invocación dentro de un proceso en ejecución.

Está en estado pasivo, o inactivo, si no está activo en ese momento, pero que se puede llegar a activar.

Un objeto pasivo contiene la implementación de sus métodos y su estado.

Cuando un objeto pasivo se "activa", crea una nueva instancia de su clase y se inicializan las variables de instancia desde el estado que está almacenado.

Pueden activarse bajo demanda como cuando son invocados por otros procesos.

Los objetos invocados remotamente, cuando no están siendo utilizados pero tienen información, gastan recursos. Por lo tanto, en los servidores hay actividades que se arrancan bajo demanda para activar los procesos.

Estos activadores registran los objetos pasivos que se pueden activar (nombre de los servidores, URL o nombre de archivo), activan los objetos y mantienen información sobre los servidores de objetos remotos ya activados.

El activador de CORBA (ver 3.2.1) recibe el nombre de ***repositorio de implementación***.

En JAVA RMI usa su activador en cada máquina que actúa como servidor, activando los objetos sobre esa máquina.

La persistencia de un objeto está dada por contar con esos procesos de activación. Esos objetos se almacenan en “almacenes de objetos persistentes” , guardando su estado en forma empaquetada.

Cuando un cliente activa un objeto y pierde la conexión, el cliente y el servidor quedan huérfanos. Si se puede conocer el estado de una entidad remota se puede tratar con los huérfanos, para poder finalizarlos y reclamar los recursos que libera.

Frecuentemente, un thread del sistema analiza la cantidad de referencias que tiene el objeto y si no tiene, es removido.

Cuando un objeto remoto no es ya referenciado por ningún objeto, es deseable que la memoria utilizada por ese objeto sea recuperada. Para “limpiar” estas memorias se usan técnicas de **garbage collection**.

En Java, este mecanismo se basa en un recuento de referencias, usando los proxys de los objetos remotos que existen en el cliente.

El garbage collector se debe ejecutar automáticamente, con cierta frecuencia, o se puede forzar manualmente.

2.2.4 Serialización de objetos

Serializar un objeto significa “aplanarlo”, es decir, obtener una forma lineal adecuada para ser almacenado en disco, o para transmitirlo en un mensaje.

La deserialización consiste en volver a llevar esa forma lineal a la forma original del objeto.

Si asumimos que el proceso de deserialización (también llamado *rehydrating*) no tienen conocimiento del objeto original y que debe reconstruirlo desde la forma lineal, es necesario incluir en ella, la información necesaria para la reconstrucción.

Se debe tener en cuenta que los objetos tienen un estado. Ese estado está dado por los valores que pueden tener uno o más campos, al instanciarse la clase.

Al serializar un objeto debe incluirse información de la clase, tipos y nombres de campos. Si el objeto a serializar tiene referencias a otros objetos, también deben serializarse éstos para su reconstrucción exacta en el servidor.

Dentro de la forma lineal se incluye el objeto que se referencia y la referencia dentro del objeto se indica como un apuntador o handler.

Aunque el objeto original referencie más de una vez a un objeto (que se incluirá en la forma lineal), este objeto estará sólo una vez en la serialización: las demás referencias se implementarán como handlers.

La información sobre la clase incluye nombre y versión. El número de versión es para no invocar una versión “vieja”.

En el caso de Java, para los tipos primitivos, en la serialización se usan métodos de la clase ObjectOutputStream para escribirlos en un binario transportable.

Caracteres en general se escriben en UTF (Universal Transfer Format) usando

los métodos writeUTF. En el caso de cadenas de caracteres, se preceden por el número de bytes que ocupan.

Dado el objeto alumno a serializar, en Java se crea una instancia de la clase *ObjectOutputStream* y se invoca el método *writeObject* con el objeto alumno como argumento.

Para deserializar desde el stream, se abre *ObjectInputStream* y se utiliza *readObject* para la reconstrucción del original.

Serialización y deserialización es realizada, normalmente, por el middleware.

Hay variables que no deben ser serializadas, como referencias locales a archivos o sockets. Cuando es así, esas variables se declaran como *transient*.

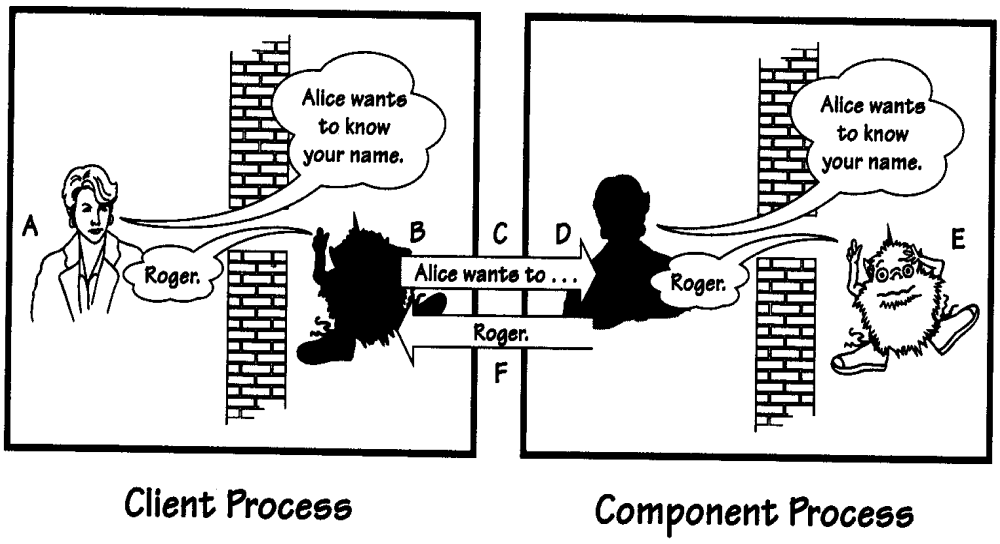
2.3 De los objetos a las componentes

En SES000, Roger Sessions analiza la evolución desde la programación Orientada a Objetos al uso de Componentes.

Hace especial hincapié en la dificultad de implementar las tres características del modelo POO (polimorfismo, herencia y encapsulación).

Con respecto al polimorfismo, utiliza un ejemplo donde construye un objeto empleado, y dos métodos: uno para asignarle un nombre y otro para responder el nombre, en caso que "alguien" lo pregunte.

Pero luego, va haciendo más compleja la relación, considerando la alternativa que el objeto conteste según quien pregunte (si es jefe, contesta; si no lo es, no contesta).



Así define dos tipos de empleados: uno que contesta siempre, pregunte quien pregunte, y otro que responde sólo cuando pregunta el jefe. Al que contesta siempre, lo llama LaidBackEmployee, y el que contesta sólo cuando pregunta el jefe, ParanoidEmployee. O sea: el objeto empleado tiene diferentes comportamientos, de acuerdo a quien lo invoca.

El punto de instanciación, cuando se crea el empleado, sería el momento donde el empleado asume uno de los dos comportamientos.

Employee empx = create ParanoidEmployee
o, en el otro caso
Employee empx = create LaidBackEmployee

De esta manera se asume que, en primer lugar, es un empleado, con el comportamiento inherente a su condición de tal.

Pero además, podemos hacer una mayor especificación, de acuerdo al comportamiento particular, en cuanto a contestar el nombre.

El punto de instanciación es en el único lugar donde se hace la distinción sobre qué tipo de empleado es.
El código, de allí en adelante tratará con empx, como una implementación del objeto empleado.

Sessions describe a este comportamiento callee makes right, pues cae en el objeto la decisión sobre qué tipo de empleado se está invocando. Lo distingue de un modelo no orientado a objetos, donde el que invoca (el código del cliente) debe determinar a qué tipo de empleado invoca, existiendo dos procedimientos diferentes. Llama a este comportamiento caller makes right.

A través de este ejemplo, define polimorfismo como la posibilidad de tener distintas implementaciones del comportamiento de un objeto, siendo la responsabilidad del objeto cuál de ellas usará.

Según Sessions, ni la encapsulación ni la herencia han logrado la trascendencia esperada: la encapsulación, porque no la soportan los lenguajes que no son OO; la herencia, porque no siempre puede usarse exitosamente (Roger Sessions es, en realidad, bastante más drástico: “...Nobody could figure out how to use it sucessfully, even in those limited cases when they could agree on what it was...”)

También, cita los inconvenientes que tiene el programador del cliente para interactuar con el objeto, a causa de dependencias del lenguaje: que es dificultoso reusar objetos desarrollados para un lenguaje desde otro. Y aún muchos desarrolladores trabajan en lenguajes que no son OO.

La conclusión sería que sólo la comunidad que trabaja en POO podría establecer comunicación con objetos construidos en ese modelo.

De esta manera, explica que el advenimiento de las componentes es un intento que solucionar algunos problemas con los objetos.

Las componentes difieren de los objetos en que:

es una forma de tecnología de packaging. Puede verse como un paquete de software que puede usar cualquier cliente. No es la forma de implementar una tecnología. Se pueden implementar de acuerdo a POO, o no.

Se ven como si estuvieran en un lenguaje neutral: el programador cliente no tiene porqué saber qué lenguaje se usó para implementar la componente

Están "más" encapsuladas que los objetos. A través de un lenguaje de definición de interfases (IDL) se describe el comportamiento de la componente. Sólo comportamiento: no describe la implementación de ese comportamiento.

En los 90´s, los dos modelos de componentes dominantes fueron CORBA (de OMG) y COM/DCOM (de Microsoft).

Según Sessions, a partir de 1998 comienza la influencia de Sun con la tecnología Java. Desde su punto de vista, las componentes de Sun están escritas en Java, transformándose en un lenguaje de implementación de componentes, de desarrollo del cliente, del IDL. O sea, que se apartaría del concepto inicial (y esencial) sobre lo que debe ser un componente.

2.3.1 Las componentes distribuidas

El problema de distribuir los procesos a lo largo de la red, implica un análisis profundo de cómo se relacionan.

Se debe pensar, por ejemplo, en los cambios necesarios y recurrentes en la lógica de negocio (business logic), imposible de realizar rápidamente si las componentes que la implementan están repartidos en varias máquinas pequeñas.

Además, los cambios que se realicen en la interfase de usuario no deben incidir en las componentes que implementan la lógica de negocio.

Una solución es separar el cliente y la instancia de la componente que se invoca en dos procesos diferentes, donde el proceso cliente trabaja en computadoras orientadas a la interacción con el usuario, y los procesos componentes en computadoras especializadas (y preparadas) para compartir y administrar la lógica de negocio.

Estando en diferentes computadoras, la comunicación entre el proceso cliente y el proceso componente se implementaría de la siguiente forma: en el proceso cliente, se agrega una representación del componente (*instance surrogate*). En el proceso componente, se crea una representación del cliente (*client surrogate*).

Veamos como funciona: el cliente hace un requerimiento a la componente, a través de su interfase. Ese requerimiento llega al *instance surrogate* que está en el mismo proceso cliente. Esta representación del componente es quien se comunica con el *client surrogate*, empaquetando el requerimiento y mandándolo a través de la red, a la máquina donde reside el proceso componente.

Una vez allí lo recibe el client surrogate quien le hace el requerimiento a la instancia real del componente, quien contesta al client surrogate.

La respuesta es empaquetada por el client surrogate, llegando a la máquina del proceso cliente, donde la recibe el instance surrogate, quien es quien se la manda al proceso cliente.

Conclusión: el proceso cliente se abstrae de la ubicación del componente, implementándose los que en Sistemas Distribuidos se llama transparencia de acceso, que es la abstracción por parte del solicitante sobre la ubicación del recurso, local o remoto.

Las componentes residirán en una computadora prepara para este fin, permitiendo una fácil administración (por ejemplo, en caso de ser necesaria la modificación) y seguridad.

Esta arquitectura de distribución es la que usan los sistemas con componentes (CORBA, DCOM/COM/MTS, EJB de Sun).

Una ventaja de la tecnología de componentes es la separación entre lógica de ejecución y lógica de negocio.

2.3.2 El problema sin resolver: la seguridad y la escalabilidad

Cómo asegurar que la componente que responde es realmente la que se invocó? Cómo asegurar la no intercepción de la respuesta?

En cuanto al acceso a bases de datos, el modelo distribuido de componentes exige interactuar con múltiples bases, con procesos cooperantes. Las transacciones deben adaptarse a esta nueva forma de trabajo, acostumbrados como están a la relación de requerimiento proceso-Base de datos. El sistema de transacciones tradicional trabaja adecuadamente en ambientes “cerrados”, con transacción de corta vida. En el nuevo modelo a plantear, la lógica de negocio distribuida es implementado con lenguajes como el BPEL4WS (ver 5)

Con respecto a la escalabilidad, en el modelo distribuido de componentes, cada cliente es representado en el proceso cliente por el surrogate y la instancia de la componente. Esas instancia seguramente se conectarán con distintas bases generando varios mensajes, y además, procesarán gran cantidad de datos. Si cada PC es un cliente potencial, estaremos generando una actividad que no se puede sostener. Hay que lograr una escabilidad que permita un aumento importante en el número de clientes.

Para Roger Sessions, algunas soluciones las da TPM, Transaction Processing Monitors, de Microsoft.

2.4 Sobre las aplicaciones

2.4.1 Net Centric Computing (NCC)

Net Centric Computing incluye networking y comunicaciones. Pero es algo más. Ha afectado la forma de usar (y por lo tanto diseñar) tanto el software como el hardware. Y las aplicaciones.

NCC es un ambiente distribuido donde las aplicaciones y datos son trabajados desde los network servers según las necesidades del usuario. Este enunciado no es de por sí, muy claro sobre cuáles son las ventajas con el modelo que se utiliza actualmente.

Scott Tilley, en TIL000, hace una importante diferenciación entre NCC, network computer y personal computer.

NCC no es lo mismo que network computer (NC). NC equivale al concepto de network client o thin client que interactúa con la red.

En el ambiente PC se opta primero por el acceso y uso de los recursos locales, accediéndose a la red para entrar o interactuar con un site.

NCC en cambio, permite que una aplicación cliente interactúe con aplicaciones altamente portables ubicadas en network servers, o incluso entre clientes, abstrayéndose de la arquitectura.

Cuando Tilley habla de Thin client, separa clientes NC, NetPc y WBT (Windows Based terminals). Su descripción es la siguiente:

- El cliente NC usa Java para el procesamiento local. Puede verse como una computadora que no corre software Windows y que no necesariamente usará un chip Pentium de Intel. Inconvenientes: acceso a sistemas legacy, que, en gran parte se basan en Windows).
- NetPc es una solución intermedia para competir con NC. ES una PC "recortada", que puede administrarse centralmente y que, si se quiere, puede correr aplicaciones Java. El abaratamiento reside en la imposibilidad de agregar hardware o software.
- WBT se basa en un modelo centralizado, donde las aplicaciones y datos están en un server central. Un WBT es un dispositivo de display, semejante a un X-station en Unix.

La ventaja es más clara para los desarrolladores: ellos al escribir una aplicación NCC extienden su base de clientes, al codificar sólo una vez accediendo a múltiples plataformas ("write once, run anywhere").

Pero hay que ser cuidadoso con enunciar la reducción de costos como una ventaja de NCC: se requiere un alto costo operativo por la complejidad del manejo de ambientes distribuidos heterogéneos. No deben despreciarse tampoco los costos en capacitación y especialización,.

También debe considerarse que NCC cambia la forma de tratar con el concepto de versionado del software. Cuando se libera (y se modifica) soft constantemente, es difícil determinar la “current version”.

2.4.2 Web Enabled Applications

Cuando se describen aplicaciones que corren en la WEB o en redes basadas en Internet como las intranets, se habla indistintamente de aplicaciones Web enabled y web based.

Si bien ambas exigen un Web Browser para accederlas, son conceptos diferentes.

- *Web Based Applications* son desarrolladas para correr en la Red.
- *Web Enabled Applications* son aplicaciones tradicionales que no fueron inicialmente concebidas para ejecutarse en la Web y posteriormente se le agrega una interfase Web.

Supongamos que contamos con un sistema de liquidación de impuestos y como estrategia de acercamiento al cliente decidimos incorporar la consulta a través de la Web. Si no se está pensando en una reingeniería, una alternativa es agregar una interfase Web a la aplicación existente, lo que la convierte en disponible por la Web.

Estas aplicaciones tendrán un gateway Web para integrarse con el viejo soft.

Como vemos esta alternativa hace viable el poder llevar sistemas legacy a la Web. No obstante hay que tener en cuenta que la posibilidad de crecimiento en la tendencia será limitado. En este tipo de alternativa, lo único que se hace es escribir (y reescribir) sólo los módulos necesarios para la integración con la interfase: el sistema en sí no se modificará y seguirá trabajando como antes.

Decidir si una aplicación será Web enabled o based, implica analizar, además de lo que se tendría en cuenta en cualquier aplicación accesible por la Red (seguridad, vínculos de comunicación, expertise) que objetivo de crecimiento se tienen, es decir, si se quiere continuar explotando las capacidades de la Web. Si es así, es recomendable una reingeniería y transformar a la aplicación en Web based.

2.4.3 Las aplicaciones Three-Tier

Las aplicaciones de este tipo tiene importantes ventajas.

- Separación entre la interfase de usuario y la lógica de negocio.

La interfase de usuario se ubica en máquinas a tal efecto, diferentes de las máquinas donde reside la lógica de negocio. Físicamente están separadas... y lógicamente también.

Esta característica brinda flexibilidad tanto para el diseño de la interfase, como para la reusabilidad de la lógica de negocio. Por ejemplo, una componente que calcula el precio de un producto de acuerdo a diferentes pautas, puede ser accedida por páginas Web, PCs clientes de negocios mayoristas, servicios autorizados, etc.

- Rapidez en el desarrollo y actualización

Si la lógica de negocio ha sido implementada y testada exitosamente, nuevas interfases puede agregarse rápidamente. También modificar las componentes, sin modificar la interfase de usuario. O construir componentes nuevas e incluirlas posteriormente, ya testeadas.

- Facilidad de administración

2.4.4 Aplicaciones three tier vs. aplicaciones cliente servidor.

Hasta aquí vale volver hacia atrás y preguntarse ¿Cuál es la ventaja con respecto al modelo cliente servidor?

La diferencia (y la ventaja) es la posibilidad de agrupar la lógica de negocio, implementada en componentes, en maquinas dedicadas a ello, lo que se llama *central middle tier*, en lugar de tenerlas dispersadas en diferentes máquinas.

2.4.5 Building Blocks Services. Loose coupling. Shared Context

Los **Building Blocks Services**, también llamados Foundation services, son componentes basados en Internet que ofrecen funcionalidad de una forma reutilizable y programable.

Estas componentes a las que puede recurrir el desarrollador para "armar" su aplicación, pueden combinar con otras componentes.

Ejemplo de estos building blocks services, es Microsoft Passport. Este site es muy visitado, pero no por acceder expresamente a través de su URL (http://www.passport.com). Aplicaciones Web de otros sitios, acceden a él para utilizar su servicio.

Una característica de este tipo de modelos es lo que se llama **Loose Coupling**, que permite que los servicios de la Web puedan realizar distintas acciones, en diferentes lenguajes, a través de distintas aplicaciones que se ejecutan en diferentes plataformas. De allí, su importancia en el modelo .NET.

El concepto no es nuevo: ya se trabajada en DCOM, CORBA y RMI, modelos donde se permitió ejecutar en la red servicios que estaban en sistemas remotos, para dar más poder en la red local. El problema de estos modelos es la falta de escalabilidad sobre Internet, uno de los objetivos de .NET. Por ejemplo, no se podía cambiar la interfase del server sin cambiar la interfase del cliente.

Otra característica es el llamado **shared context**.

Supongamos un sistema distribuido que ofrezca personalización para el usuario, de manera tal que cuando un individuo usa el servicio, éste automáticamente lo reconoce y personaliza.

Las alternativas pueden ser que la identidad se determine por una cookie que se mantiene en el sistema o puede ser por el ingreso de userid/password.

Ambas soluciones son problemáticas considerando la cantidad de cookies o de userid/password que se deben mantener por sites accedidos.

Es necesario para lograr la integración, que los servicios puedan compartir información sobre identidad y de contexto en general.

Esto se puede lograr con lo que se llaman *Smart Web Service*. Este tipo de implementación, permite reconocer un contexto y compartirlo con otros servicios, actuando de acuerdo a quién lo invoca, cómo, cuando y porqué.

2.5 Notas

En este capítulo, las lecturas de [CAR002], [COL001], [SHI000] y [TIL000] fueron útiles para hablar de la nueva generación de aplicaciones. También [THO001].

También [MIC002], si bien habla de .NET introduce el artículo con conceptos referentes al tema del capítulo.

En cuanto a la tecnología de objetos y componentes, fueron SUMAMENTE provechosos los libros [PLA000] y [COU001].

Capítulo III

Arquitecturas Distribuidas

3 Arquitecturas distribuidas

3.1 Lenguajes de definición de interfases. Representación externa de datos.

El lenguaje de definición de interfases (IDL) fueron desarrollados para que los objetos en lenguajes diferentes puedan invocarse entre sí.

Corba usa CORBA IDL, Sun propone XDR para su RPC, DCE usa su IDL para RPC, Microsoft usa DCOM IDL. Un punto interesante es si estos IDLs exponen las interfases de manera tal que sean comprendidos por cualquier objeto invocante.

Se llama **representación externa de datos** al estandar que acuerdan dos o más computadoras para intercambiar datos. de esta forma se superan problemas como el ordenamiento de enteros, diferentes formas de representar flotantes o la cantidad de caracteres que toman los distintos formatos.

En RMI o RPC se debe llevar cualquier estructura que desea pasarse como argumento o devuelto como resultado a una forma plana, que viaje y sea interpretada correctamente en el destino (Serialización de objetos, 2.2.4).

El siguiente código es para invocar read y write en un file server, usando XDR.

```
const MAX = 1000;
typedef int IdentificadorArchivo;
typedef int ApuntadorArchivo;
typedef int Longitud;

struct Datos {
int longitud;
char bufer[MAX];
};
struct argumentosEscribe {
IdentificadorArchivo f;'
ApuntadorArchivo posicion;
Datos datos;
};
struct argumentosLee {
IdentificadorArchivo f;'
ApuntadorArchivo posicion;
Longitud longitud;
};
program LEEESCRIBEARCHIVO {
        version VERSION (
        void ESCRIBE(argumentosEscribe) = 1;
```

Data LEE(argumentosLee) = 2;
)=2;
}= 9999;

Nótese que se indican versiones (9999 es la versión de LEEESCRIBEARCHIVO), y códigos de cada procedimiento (ESCRIBE(argumentosEscribe) = 1 indica que el código es 1; de LEE, es 2).

3.2 Comparación de Tecnologías de objetos distribuidas

Cuando se debe optar por una arquitectura distribuida, se deben considerar distintos aspectos, entre otros:

- Escalabilidad
- Performance
- Activación
- Manejo de estados
- Garbage Collection
- Seguridad

Los distintos protocolos que trabajan en sistemas distribuidos, cubren algunos de estos aspectos.

Los modelos de programación tradicional han sido extendidos para aplicarlos en la programación distribuida.

El llamado a procedimiento convencional, local, se extendió al remoto (RPC).

Al surgir POO, fue necesaria la invocación remota de métodos (RMI). RMI permite que un objeto que está dentro de un proceso, invoque los métodos de un objeto que está en otro proceso.

El modelo de programación basado en eventos para progresar a la versión distribuida, necesitó extender la capacidad de sus objetos de recibir notificaciones de eventos que ocurren en otros objetos a los que quieren acceder.

El uso de interfases para la comunicación entre distintos módulos, permite "esconder" implementación y ofrecer solamente una forma de comunicación con el módulo.

La implementación puede cambiar pero la interfase permite que los otros módulos sigan comunicándose con el "modificado" de igual forma, si esto es deseable.

En el modelo RPC, hablamos de interfases de servicio. En el modelo RMI, de interfases remotas.

Las de servicio, usadas en el modelo C/S, ofrecen la forma de acceder a los servicios que brinda un servidor. La interfaz de servicio es la especificación de los procedimientos que ofrece un servidor, define los argumentos de entrada y salida, etc.

La interfaz remota especifica los métodos de un objeto factibles de ser invocador por objetos de otros procesos. Define argumentos de entrada y de salida.

La diferencia entre ambas interfases, es que en la remota, los métodos pueden pasar objetos como argumentos y como resultados. También se pueden usar referencias de objetos remotos.

Cuando un lenguaje quiere usar RMI, debe considerarse como interpretar la definición de las interfases que indiquen como invocar los argumentos de entrada y salida.

Un ejemplo de esto es Java RMI. Pero esta solución exige que los objetos que se quieren comunicar estén en Java.

Para que objetos implementados el lenguajes diferentes puedan invocarse, existen los lenguajes de definición de interfases (IDL).

ORB (Object Request Broker) es el middleware (ver 3.3) que establece una relación cliente servidor entre objetos. A través del ORB un cliente invoca de forma transparente un método de un objeto servidor ya sea en la misma máquina o a través de la Red.

El ORB provee interoperabilidad entre objetos en ambientes distribuidos. Su tarea comprende:

- Interceptar la invocación al objeto
- Encontrar el objeto invocado
- Pasar los parámetros
- Invocar al método
- Retornar los resultados

El cliente no tiene que conocer la ubicación del objeto ni en qué lenguaje está escrito. Sólo necesita saber cómo invocarlo.

En las aplicaciones típicas cliente/servidor, el desarrollador tiene dos alternativas: diseñar el mismo un protocolo o usar un standard que depende del lenguaje, el transporte, etc. En ORB, el protocolo es definido por el IDL.

Para poder implementar la reusabilidad, una característica clave de la tecnología de objetos, la capa de middleware es imprescindible para soportar la comunicación entre objetos. Así, la interfase del objeto sirve para especificar los métodos.

3.2.1 CORBA. GIOP

CORBA son las siglas de Common Object Request Broker Architecture.

En el año 1989, se formó la OMG (Object Management Group) para propiciar los sistemas de objetos distribuidos que garanticen la interoperabilidad. Para ello, trabajó dentro de la filosofía de los sistemas abiertos.

El OMG fue quien inició la terminología ORB. Y en el año 1990, un grupo de empresas acuerdan una especificación para una arquitectura de este tipo, que llamaron CORBA.

En 1996, se definió la especificación CORBA 2.0. A través de ella se dictan los standares para implementar la comunicación entre objetos definidos en lenguajes diferentes. Estos standares reciben el nombre de GIOP (General Inter-ORB Protocol) y la idea era que este GIOP puede ejecutarse sobre cualquier capa de transporte.

La especialización TCP/IP de GIOP es IIOP (Internet Inter-ORB protocol). En otras palabras: GIOP es el protocolo base, y IIOP lo "adapta" a TCP/IP.

Los ORB del cliente y del servidor interactúan a través de mensajes GIOP cuando quieren ubicar un objeto. El ORB del cliente local, una vez ubicado el objeto remoto, le pasará al cliente una referencia para que éste pueda invocar los métodos del objeto remoto.

El paquete GIOP, tiene un header donde se almacena la medida del mensaje, la versión de GIOP, el ordenamiento de los bytes. Los datos se alinean según la implementación usada: no hay especificación global.

Los datos son almacenados en formato CDR, lo cual, obviamente, debe conocer el receptor para deserializarlo adecuadamente.

CORBA utiliza un formato para las referencias a objetos remotos, el IOR (Interoperable Object Reference) que se puede usar si el objeto es activable o no.

Nombre de tipo de interfase IDL	Protocolo y dirección			Clave de Objeto	
Identificador de repositorio de interfase	IIOP	Nombre del dominio del host	Número de puerto	Nombre de adaptador	Nombre de objeto

En el primer campo del IOR, se especifica el nombre de tipo de interfaz IDL del objeto CORBA. Si el ORB tiene un repositorio de interfaz, este nombre debe coincidir con el identificador en el repositorio.

Veamos cómo actúa un cliente que requiere un objeto remoto usando CORBA.

El cliente se linkea en tiempo de compilación, a un stub que al activarse actúa como un proxy.

Busca en el repositorio dinámicamente para invocar el objeto remoto.

Cuando el cliente quiere invocar el objeto remoto, llama al método que hace el binding, lo que provoca que el runtime CORBA determine donde está el objeto y fuerza al server para que lo active.

Cuando desde el objeto llega la notificación sobre su inicialización, el server retorna al cliente, una referencia al objeto. Y entonces si: el cliente puede invocar el método remoto.

<table>
<tr><th colspan="2">Cliente</th><th colspan="2">Servidor</th><th></th></tr>
<tr><td colspan="2">Código cliente, nivel aplicación</td><td></td><td>Implementación de server de objetos</td><td>Nivel de programación de aplicaciones con objetos</td></tr>
<tr><td>Stub Cliente</td><td rowspan="2">Repositorio Interfase</td><td>Implementación del repositorio y activación</td><td>Server (esqueleto del objeto)</td><td rowspan="2">Nivel de arquitectura remota</td></tr>
<tr><td>Runtime CORBA</td><td></td><td>Adaptador Objeto/ runtime CORBA</td></tr>
<tr><td colspan="2">ORB</td><td colspan="2">ORB</td><td rowspan="2">Nivel del wire protocol</td></tr>
<tr><td colspan="2">TCP/IP</td><td>Socket</td><td></td></tr>
</table>

3.2.2 COM, COM+ y DCOM

Microsoft Component Services se compone de las siguientes tecnologías:

- **COM** (Component Object Model)
- **DCOM** (Distributed Component Object Model)
- **MTS** (Microsoft Transaction Server)
- **MIIS** (Microsoft Internet Information Server)
- **MSMQ** (Microsoft Message Queue)

A través de estas tecnologías, Microsoft trata con la construcción e implementación de aplicaciones distribuidas.

Cuando en 1993 surge COM, tecnología de Microsoft, se avanzó en la posibilidad de comunicar aplicaciones escritas en distintos lenguajes.

En COM, el cliente llama a funciones en el server a “nivel binario”, y por lo tanto, no necesita conocer en qué código fue implementado. También a través de COM se podía acceder a cierta funcionalidad del sistema operativo, como manejo de transacciones y queueing.

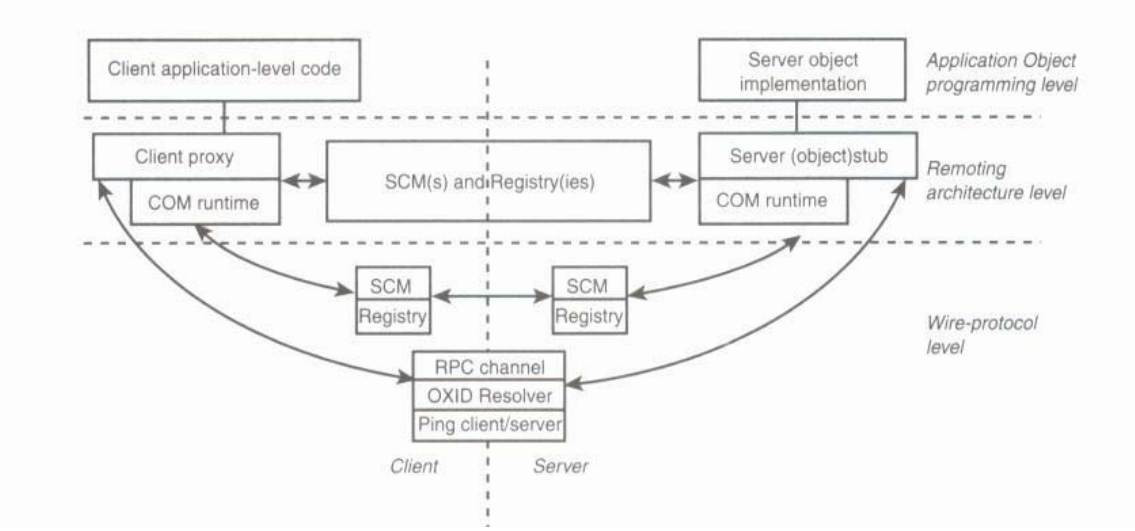
Pero tiene dos desventajas:

- requiere participar de la infraestructura de la aplicación. Se debe analizar el marshalling que usa y adaptarlo si es necesario. Las diferencias las debe tener en cuenta el desarrollador para subsanarlas.

- COM trata con las aplicaciones a través de interfases externas, es decir, que no comparte la implementación interna. Por ejemplo si las aplicaciones a comunicarse tratan el tipo string de distinta forma, lo debe prever el desarrollador para zanjar las diferencias.

Para trabajar con estas diferencias, el aporte de las características de la programación orientada a objetos es innegable.

De la aplicación local en COM, se creció al ambiente distribuido con DCOM.
DCOM presenta una arquitectura compleja.



Esta figura pertenece a la referencia [SCR000].

En DCOM cuando el cliente necesita acceder a un objeto remoto, se crea un proxy local. A través del SCM (Service Control Manager) local se tiene la información del sistema remoto. También puede ofrecer esta información la Registry.

El SCM local contacta al SCM remoto. Este activa el objeto remoto y pasa la referencia del objeto remoto al sistema local.

Esa referencia llega al proxy del objeto y el cliente puede crear instancias del objeto remoto. EN este punto, el cliente está en condiciones de tratar con el objeto remoto usando el proxy y RPC.

DCOM usa diferentes tipos de mensajes para invocar métodos remotos.

El paquete OBJREF se usa para codificar la respuesta de un sistema remoto, para identificar la instancia particular de un objeto.

El paquete ORPCTHIS es usado por el cliente para invocar un método.

En cuanto a escalabilidad, DCOM presenta algunos problemas, pues su mecanismo de garbage collection genera mucho tráfico. Para garantizar que todos los participantes de una invocación remota están activos mientras se resuelve la llamada al método, deben enviar “ping” al server cada dos minutos.

Cuando el sistema sospecha que se perdió una conexión con el cliente, por ejemplo ante la pérdida de un ping, espera dos períodos de time out. Si luego de ese

tiempo el cliente no respondió con un ping, el server destruye el objeto remoto y restituye los recursos.

DCOM, además, intenta implementar transparencia en cuanto a la ubicación del objeto. Además considera que los clientes no necesitan saber si el objeto está activado, local o remoto. Estas características requieren un constante manejo de estados lo que hace muy compleja la arquitectura.

En cuanto a la seguridad, DCOM permite seleccionar el grado de seguridad en la comunicación. Se puede transmitir texto claro, encriptar el header y los datos de la llamada al método y otras alternativas.

MTS soporta escalabilidad, seguridad, manejo de transacciones... en fin: todo lo necesario para que una aplicación en un server pueda crecer sobre una capa de servicios en forma standard. Es el primer software comercial que combinó transacciones con componentes, en 1996.

Cada aplicación basada en MTS se construye desde componentes COM y los clientes acceden remotamente a través de DCOM.

Mientras DCOM provee una comunicación asincrónica usando RPC, las aplicaciones basadas en Web usan, preferentemente, HTTP.

MSMQ provee dentro de Windows NT server la forma de implementar comunicación asincrónica y no bloqueante. Así, una aplicación puede enviar mensajes a otra aplicación sin esperar la respuesta. Los mensajes se envían a una cola.

MSMQ basa su acceso en COM, se integra con MTS, soporta múltiples plataformas, maneja prioridades para los mensajes, puede firmar digitalmente y encriptar mensajes, etc.

Toda esta tecnología puede trabajar junta.

Supongamos que queremos implementar un sistema usando Microsoft Component Services.

Puede haber uno o más componentes COM que tienen la lógica de negocio. Acceden a bases de datos, por ejemplo, para realizar consultas. A su vez estas componentes COM pueden trabajar bajo MTS para dar soporte transaccional entre bases de datos y otros servicios.

Puede usar MSMQ si la componente quiere enviar un mensaje sin esperar la respuesta.

Puede haber estaciones Windows , clientes que se comunican a través de DCOM.

COM+ provee soporte automático para desarrollar objetos que participan dentro de transacciones.

El desarrollador “marca” el objeto como que va a requerir una transacción. Cuando el objeto se activa, crea una.

Supongamos que el objeto quiere hacer cambios a una base de datos. Si todos los objetos que intervienen están de acuerdo, la transacción puede llegar a buen término, informan a COM+ que hace un commit de la transacción. Si no están de acuerdo, COM+ aborta la transacción.

El artículo de Berkeley citado en la bibliografía (BER097), trata de las diferencias entre DCOM y CORBA.

Mientras DCOM surge desde Microsoft, CORBA es creada por una coalición de empresas, para proveer un standard abiertos para ambientes heterogéneos.

CORBA permite independizarse de la plataforma y ofrece ubicuidad. Existe desde 1990 y la implementación "comercial" data de 1992.
DCOM estaba disponible en beta en 1992.

CORBA tiene las ventajas inherentes a la tecnología Java. Hay muchos ORBs Java lo que hace posible escribir clientes y servidores Java que se pueden ejecutar en cualquier plataforma donde la máquina virtual Java esté disponible.

DCOM especifica la interfase entre objetos, separando interfases de implementación y buscando API´s para encontrar dinámicamente las interfases, cómo cargar e invocar. Según BER097, DCOM tiene una API compleja.

Además, DCOM no soporta herencia múltiple (CORBA si), si no múltiples interfases. Está basado en componentes ActiveX y trabaja en forma nativa con TCP/IP, Java y HTTP.

DCOM ofrece soporte para transacciones distribuidas, messaging, seguridad, multithreading.
CORBA ofrece servicios como naming, eventos, life cycle, transacciones distribuidas, seguridad, persistencia.

3.2.3 RPC

RPC es una comunicación entre máquinas que están interconectadas.
Si se usa TCP/IP, esta conexión se implementará a través de direcciones IP y números de puertos.

El cliente debe determinar dirección IP o hostname del server con la que se desea conectar, y el puerto que lo relaciona con la aplicación que le interesa.
El server debe tener registrada la relación entre puertos y aplicaciones, para que, cuando se recibe una petición, pueda ejecutarse el procedimiento apropiado.

El endpoint mapper es una implementación específica de DCOM que hace esta función, manteniendo una tabla que relaciona puertos e interfases a aplicaciones específicas. En CORBA esta relación la establece el ORB (Object Request Broker).

SUN RPC fue diseñado inicialmente para la comunicación Cliente Servidor para NFS sobre SUN (ONC RPC). Usa un lenguaje de interfaz que es XDR y un compilador de interfaz que es rpcgen.

Este paradigma se basa en que tenemos dos computadoras unidas por algún vínculo de comunicación donde una de ellas quiere ejecutar un procedimiento que esta en la otra. Normalmente usa IPC (interprocess communication) para implementarse. Es una comunicación basada en mensajes. Se puede usar sobre TCP o UDP.

En el sistema remoto hay un proceso demonio RPC que esta constantemente escuchando en el port para ver si llega algo. Este mensaje que llega tiene una estructura bien definida: el identificador de la función que quiere ejecutar y los parámetros que quiere pasar a esa función.

RPC permite que los programas llamen a los procedimientos ubicados en otra maquina, imitando la función de llamada a un procedimiento local: el programa invoca los procedimientos indicando los parámetros, espera que se ejecute y al terminar retorna el control a la dirección siguiente a la llamada.

En el caso de los sistemas distribuidos, entre los parámetros se debe indicar la referencia a otra maquina y lo que complica el esquema es que las maquinas que contienen el programa y el procedimiento, pueden no ser iguales: pueden tener distinta arquitectura, son distintos espacios de direcciones, etc.

La máquina que contiene el servidor debe recibir una solicitud que debe poder comprender, de la misma manera que el cliente debe poder comprenderlos mensajes devueltos por el servidor..

Otro punto es como reacciona el sistema ante fallas, tanto de la maquina que contiene el proceso servidor como la del cliente.

Por cada servicio se almacena en el servidor, el número de programa, la versión y el número de puerto local por el cual se atenderá la solicitud de servicio.

Al arrancar el servidor, le da estos datos a su demonio portmapper. De esta manera el cliente antes de mandar una petición, solicita al portmapper cuál es el puerto indicado, numero de programa y versión.

Los parámetros pueden ser pasados por valor, por referencia o por copia/restauración. Los dos primeros ya son conocidos por Uds. La copia/restauración permite que quien recibe la llamada copia la variable en la pila (como si fuera por valor) y cuando retorna sobrescribe el valor original.

El objetivo es la transparencia: que el programa no sepa que se está ejecutando en otra máquina (que sea como un llamado a procedimiento local).

Para lograrlo veamos como se procede. Sea A el programa que corre en una máquina y supongamos que el procedimiento a ejecutar es read, que tiene como parámetros el descriptor de archivos, un puntero a la estructura donde se deben transferir los datos y un contador de bytes. Supongamos que queremos leer un archivo

que está en otra máquina (un servidor de archivos), o sea que esta invocación será un llamado a procedimiento remoto.

En la máquina donde esta ejecutándose A hay una versión de read que se llama resguardo de cliente. Esta versión toma los parámetros, los mete en un mensaje y le pide al kernel que lo envíe al servidor. Luego de invocar send, esta versión invoca a receive y se bloquea hasta que regrese la respuesta.

La función del resguardo de cliente es semejante a la de un proxy. Si bien se comporta como un procedimiento local del cliente, empaqueta el identificador del remoto y los argumentos, y se envía como una petición al servidor. Al retornar el mensaje de respuesta desempaqueta los resultados.

Cuando el mensaje llega al servidor es enviado a un resguardo del servidor que está conectado con el servidor real. Este resguardo, que estaba esperando la llegada del mensaje, habrá ejecutado un receive y estaba bloqueado, a la espera. Invoca entonces al servidor a la manera usual, siendo una invocación local.

Cuando el servidor termina retorna al resguardo del servidor quien empaqueta los resultados en un mensaje y llama al send para remitirlo al cliente. Vuelve a su ciclo llamando a receive para esperar nuevos mensajes.

Al regresar a la máquina del cliente, el kernel ve que está dirigido al cliente (en realidad, al resguardo del cliente pero el kernel no lo sabe). El resguardo de cliente desempaqueta el resultado, la copia al verdadero proceso cliente y regresa de modo usual.

El proceso cliente cuenta con los datos y no se enteró que el llamado se hizo en modo remoto. Esta transparencia es la que se quiere lograr: el cliente no usó send y receive. Estos son procedimientos de biblioteca tratados por el resguardo de cliente.

En Esta tabla se debe leer de arriba hacia abajo. Por ejemplo: programa cliente invoca al resguardo de cliente. Luego se realizan las actividades asociadas al resguardo de cliente (Prepara el buffer, ordena parámetros..., etc)

Cliente	Llamada a procedimiento de resguardo
Resguardo del cliente	Prepara el buffer de mensajes Ordena parámetros dentro del buffer Llena los campos de encabezado del mensaje Señala al núcleo
Núcleo	Cambio de contexto al núcleo Copia el mensaje al núcleo Determina la dirección de destino Coloca la dirección en el encabezado del mensaje Establece la interfaz de la red Inicia el cronómetro

Servidor	realiza el servicio
Resguardo del servidor	Llama al servidor Establece los parámetros en la pila Desempaqueta los parámetros
Núcleo	Cambio de contexto del resguardo del servidor Copia el mensaje al resguardo del servidor Ve si el resguardo está esperando Decide a cuál resguardo dárselo Verifica la validez del paquete Interrupción del proceso

La tabla de actividades en el servidor debe leerse de abajo hacia arriba. Por ejemplo: al llegar la petición desde el cliente, en el servidor hay una interrupción al proceso que se estaba ejecutando, se verifica la validez del paquete, etc). Cuando se realiza el servicio, se reinvierte el camino: se llama al resguardo del servidor, se toman los parámetros de la pila, se empaquetan los resultados, etc.

El distribuidor selecciona uno de los procedimientos de resguardo, donde se desempaquetan argumentos, se llama al procedimiento de servicio correspondiente y se empaquetan los datos. Estos procedimientos de servicio implementan lo que se define en la interfaz.

El resguardo de cliente debe empaquetar los parámetros para enviarlos al servidor. El ordenamiento de parámetros, como se llama al empaquetamiento de parámetros, puede ser muy simple si tratamos con máquinas del mismo tipo. Pero en un sistema distribuido puede haber máquinas diferentes que se quieren comunicar diferentes códigos de caracteres (EBCDIC o ASCII), diferente representación de enteros (complemento a 2), etc.

Para solucionarlo se debe conocer la organización del mensaje, la identidad del cliente y cada máquina debe contar con todas las conversiones de los formatos de los demás.

Entre diferentes alternativas, podemos analizar una donde el cliente usa su formato, original. Cuando llega el mensaje al resguardo del servidor, éste examina 1er byte del mensaje para ver quien es el cliente. Si la maquina es del mismo tipo que la del servidor, no necesita conversión. En caso contrario, el resguardo del servidor se encarga de la conversión.

También se puede acordar una representación a la que se adapten todos (formas canónicas).

Otro problema a analizar es el pasaje de parámetros por referencia. Supongamos que uno de los parámetros de la llamada del cliente sea un apuntador a un arreglo (que está dentro del espacio de direcciones del proceso llamador).

Cuando el resguardo del cliente se hace cargo, sabiendo que ese parámetro apunta a un arreglo y conociendo su tamaño, puede decidir copiar TODO el arreglo en el mensaje.

Sun RPC usa un lenguaje de interfaz que se llama XDR y un compilador de interfaces que se llama rpcgen.

En un ambiente distribuido que utiliza RPC se pueden dar diferentes fallas, a saber:

1. El cliente no puede localizar el servidor
2. Se pierde el mensaje de solicitud del Cliente al servidor.
3. Se pierde el mensaje de respuesta del servidor al cliente.
4. Fallas en el servidor

En el caso 1, puede ocurrir que el cliente no se haya ejecutado por mucho tiempo y tener una versión vieja para la comunicación con el servidor y por eso la llamada no sea la adecuada para ubicar el servidor. También puede ser que el servidor esté inactivo.

Puede que retorne un código de error, pero debe analizarse como manejarlo, distinguiendo el error del resultado de retorno válido (si el procedimiento retorna -1 podemos interpretarlo como error, pero también puede ser el resultado del cálculo solicitado).

En el 2do caso, se puede poner un cronómetro que se inicie al mandar el mensaje. Si luego de un tiempo no hay respuesta p reconocimiento (ACK) se vuelve a mandar.

Si debe repetirlo muchas veces el cliente puede interpretarlo como un caso 1, es decir, que no se localiza el servidor.

En el 3er caso, puede haber una solución semejante al 2, es decir, cronometrar. Pero a veces supone que se perdió una respuesta y en realidad ocurre que el servidor es muy lento, o no sabe si se perdió la solicitud o la respuesta, y no puede volver a mandarse sin consecuencias graves.

Veamos el caso de una transferencia de dinero. Supongamos que desde un cliente se quiere solicitar una transferencia de dinero a un servidor. EL servidor realizó la transferencia de dinero pero se perdió la respuesta al cliente. Si el cliente vuelve a hacer la solicitud de transferencia pensando que no se realizó su solicitud, se hará una segunda transferencia de dinero. .

Cuando una solicitud se puede repetir n veces sin que ocurran daños, se dice que es idempotente.

Esta situación se puede solucionar con el secuenciamiento del mensaje, es decir, asociando un numero de secuencia al mensaje para que el servidor diferencia entre el original y la retransmisión (en este ultimo caso, el número de secuencia será mayor pues es posterior). Así, se puede rechazar una retransmisión si en realidad se llevó a cabo la operación.

Puede agregarse un bit en el mensaje para distinguir original de retransmisión (siempre se llevan a cabo las originales; las retransmisiones se analizan...).

En el 4to caso, se plantean las fallas del servidor.

En el servidor la secuencia de eventos es recepción, ejecución y respuesta.

Las fallas deben analizarse según:

- Si ocurrió luego de ejecutar la solicitud del cliente y antes de la respuesta: si es así, hay que informar al cliente que se llevó a cabo la operación.

- Si la falla ocurrió antes de ejecutar la solicitud del cliente: Sólo exige que el cliente retransmita. El servidor debe indicar este tipo de falla.

Siempre está la alternativa de no hacer nada, es decir, cuando el servidor falla, el cliente no tiene ayuda. La RPC puede realizarse desde ninguna a un número grande de veces.

Hay diferentes formas de atender este tipo de problema. Es lo que llamamos semánticas RPC en fallas del servidor.

Semántica al menos una vez: se espera a que el servidor vuelva a arrancar y se intenta realizar la operación hasta poder mandar una respuesta al cliente. Garantiza que la RPC se realice al menos una vez (puede realizarse más veces). Es la que se usa normalmente en RPC.

Semántica a lo sumo una vez: se informa de la falla. Puede ser que no se haya realizado y si se realizó, será una sola vez. Lo ideal sería una semántica de exactamente una, pero no hay forma de garantizarla.

Semántica de no garantizar nada: Fácil de implementar. El RPC puede realizarse una vez, muchas o ninguna.

Fallas en el cliente

El cliente envía una solicitud y falla antes de recibir la respuesta. A esta operación se la llama huérfana. Esto provoca varios problemas pues se bloquean archivos, se desperdician ciclos de CPU, etc.

Para este problema se plantan 4 soluciones diferentes:

- Exterminación

El resguardo de cliente envía un RPC y crea una entrada en un registro (bitácora) indicando lo que va a hacer y este registro se mantiene en un disco o donde sobreviva a las fallas. Al rearrancar se analiza y se elimina el proceso huérfano (el realizado en el servidor). Las consecuencias de eliminar el huérfano pueden ser que los bloqueos no se liberen en realidad, y si el proceso huérfano creó otros procesos en otros servidores, estos también quedarán huérfanos y no son localizables.

- Reencarnación

Se divide el tiempo en “épocas” numeradas en forma secuencial. Cada vez que el cliente arranca se considera una nueva época y así es informado a todos los servidores. De esta manera se destruyen cálculos remotos de épocas anteriores o, si llega al cliente una respuesta a una petición de una época anterior, se rechaza.

- Reencarnación sutil

Cuando el servidor recibe un mensaje de “nueva época” analiza si tiene cómputos remotos generados por épocas anteriores. Si hay, se los manda al cliente para que él decida su eliminación.

- Por expiración:

A cada RPC se le asigna un período de tiempo T para completarse. Si no le alcanza, debe pedir otro quantum. Debe asegurarse que aún el cliente está activo.

El servidor esperará un tiempo T para rearrancar, para asegurarse la eliminación de los huérfanos.

El problema de esta solución es fijar el valor de T.

Nota: en caso de querer profundizar sobre RPC, ver RFC 1831 (Srinivasan, 1995) donde se describe SUN RPC.

3.2.4 RMI

Supongamos que un objeto A, a nivel de aplicación invoca un método en el objeto remoto B (también a nivel de aplicación).

Para ello es necesario que A tenga una referencia remota a B. De ahora en más, A está en el proceso cliente y B en el proceso servidor.

En el cliente debe haber un *proxy* de B. El proxy, considerado parte del soft RMI, tiene la función que la invocación al método remoto sea transparente para los clientes, comportándose como un objeto local y mandándole, en realidad, un mensaje de un objeto remoto.

El proxy se encarga de la referencia al objeto remoto, empaquetado de argumentos, desempaquetado de resultados y envío y recepción de los mensajes ante el cliente.

Cada objeto remoto del que el cliente tenga una referencia, tiene un proxy.

El proxy llama al *módulo de referencia remota*.

Cuando se empaquetan y desempaquetan las referencias a objetos remotos. Este módulo traduce las referencias a objetos locales y remotos y crea referencias. Hay un módulo por proceso.

Cada módulo tiene una tabla sobre los objetos locales y remotos.

El objeto remoto B, estará en la tabla del servidor. Pero su proxy, que está en el proceso local, figura en la tabla del cliente.

Cuando se pasa un objeto remoto por primera vez, se le pide a este módulo que cree la referencia remota y que lo añada a su tabla.

Si se invoca a un objeto remoto que no está en la tabla del módulo, se le pide al soft RMI que le cree un proxy.

En el servidor, está el resto del soft RMI: el distribuidor y el esqueleto para la clase del objeto B.

El distribuidor recibe el mensaje de petición que llega desde el proceso cliente, selecciona el método apropiado del esqueleto y la pasa el mensaje de petición.

El esqueleto implementa los métodos de la interfaz remota. EL método desempaqueta los argumentos del mensaje e invoca al método correspondiente.

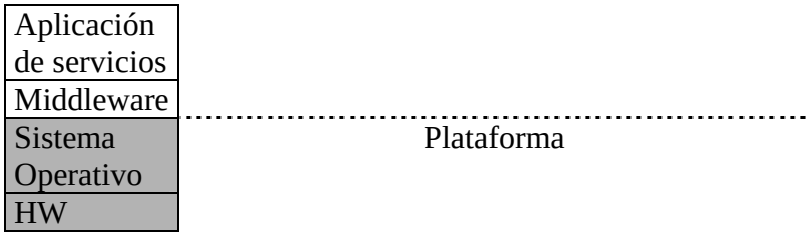
Luego de ejecutada la invocación, empaqueta el resultado para que sea enviado al proxy del objeto en el cliente en un mensaje de respuesta.

Para la comunicación entre el proceso cliente y el servidor, hay módulos de comunicación de ambos lados.

3.3 Middleware: definiciones

El *middleware* es la capa de soft que enmascara la heterogeneidad ofreciendo al programador de aplicaciones un ambiente unificado. Ofrece un modelo uniforme al programador de servidor y aplicaciones distribuidas.

La llamada a procedimientos remotos, sistemas de comunicación en grupo, invocación de objetos remotos en Java, DCOM en Microsoft (modelo común de objetos distribuidos), pueden verse como middleware.



Veamos diferentes definiciones.

Según Derek Slater, middleware es, esencialmente, software para conectar aplicaciones permitiéndoles intercambiar datos. Ofrece beneficios tales como la simplicidad, persistencia y su orientación al servicio.

La simplicidad la explica considerando cuando las aplicaciones deben compartir datos. Si “en el medio” hay una capa de middleware, cada aplicación dialoga con esta capa, necesitando sólo la interfase para acceder a ella.

En cuanto a la persistencia, el middleware permite capturar datos y almacenarlos hasta que la aplicación o la base de datos que lo necesita defina cómo usarlo.

Slater cita “tipos” de middleware: RPC, Message-Oriented (MOM), Transaction Processing Monitors, Object Monitors, Object Request Brokers (ORBs) y sus arquitecturas, Enterprise Application Integration (EAI).

Por ejemplo, Slater recomienda usar *Database Gateways* en vez de RPC, para facilitar el acceso a los datos, pues conectan aplicaciones a un tipo particular de Bases de datos.

Los conceptos que Slater vertió en el artículo citado al pie, han evolucionado y hoy se trata de manejar aún un nivel mayor de abstracción.

Por otra parte, en http://dsonline.computer.org/middleware, se habla de áreas de Middleware: Reflective, Event-Based, Wide-Area Distributed, Object-Oriented y Message-Oriented.

Por ejemplo, **Object-Oriented Middleware** extiende el paradigma POO a los sistemas distribuidos, citando como ejemplos CORBA y COM; **Message-Oriented Middleware (MOM)** es una extensión del paradigma de comunicación por paquetes.

Event-Based Middleware está pensado para la construcción de aplicaciones descentralizadas que deben ir monitoreando y reaccionando a cambios en el ambiente, como procesos de control, stock, canales de noticias en Internet (Internet news channels).

MOM, es una forma de comunicación asincrónica, que no bloquea al sender al esperar respuesta, ni el receiver necesita estar activo en la comunicación. Lo que hay que tener en cuenta, es que en MOM general la estructura interna de los mensajes es responsabilidad de la aplicación. Ejemplo de MOM son MQSeries de IBM.

En este mismo site, se define que ***el rol del middleware es facilitar el diseño, programación y manejo de las aplicaciones distribuidas para proveer un ambiente simple, consistente e integrado de programación distribuida***. Lo describe como una capa de software que abstrae la complejidad y heterogeneidad de los diferentes ambientes que constituyen un sistema distribuido, con sus diversas tecnologías de network, arquitecturas de las máquinas, sistemas operativos y lenguajes de programación.

En http://middleware.internet2.edu, se define Middleware como un amplio conjunto de herramientas y datos que ayudan a que las aplicaciones usen los recursos y servicios de la red.

Lo ubica entre el nivel de red y el de aplicación. Y lo justifica ante el creciente número de aplicaciones, y la necesidad de personalizar los accesos. Es decir, la escalabilidad[1] y extensibilidad[2] que caracteriza a los sistemas distribuidos.

El proyecto I2-MI promociona el desarrollo y distribución de middleware que soporte los objetivos de las instituciones y miembros que constituyen Internet2.

Los principios que rigen el diseño y los protocolos son, entre otros:

- ✓ El software debería ser “loosely coupled”: los cambios a los que nos tiene acostumbrados la tecnología debe permitir continuar trabajando sobre la capa de middleware subyacente.
- ✓ El software debería ser lo más barato posible, tanto en lo económico como en lo técnico, considerando que las organizaciones que utilizan IT en educación, por ejemplo, y a los que I2 les da un importante rol en esta evolución, tienen recursos limitados.
- ✓ Las soluciones deben adaptarse a las características de IT de las organizaciones tales como Universidades, donde la dinámica de los estudiantes es diferente en cuanto a la disponibilidad, cambios de

[1] *Escalabilidad* es la característica por la cual un sistema mantiene su funcionalidad cuando aumentan el número de recursos y usuarios significativamente.

[2] *Extensibilidad* es la característica por la cual un sistema mantiene su funcionalidad al añadir nuevos servicios.

equipamiento y diversidad, o la privacidad o requerimientos legales que pueden exigir otras instituciones.

- ✓ El software debe ser fácil de usar.

Uno de los puntos álgidos del advenimiento del networking fue los altos costos demandantes en equipamiento y los costos operativos (accesos, mantenimiento, etc.). A esto debemos sumarle los costos de expertise.

En middleware, los costos a nivel de equipamiento son bajos. Es más: podemos verlo como una forma de reducir esos costos. Los costos relacionados con el software tienen que ver con la adaptación de los sistemas heredados (legacy) en la evolución hacia aplicaciones basadas en middleware.

Pero el desarrollo de un middleware tiene un alto costo. Acordar el deployment, las relaciones entre objetos y grupos, la propiedad de los datos y la relación entre las diferentes entidades, establecer un marco legal para la convivencia de diferentes softwares.

3.3.1 RPC es middleware?

RPC (Remote Procedure Call) fue siempre una herramienta importante para explicar el contexto de conectividad en sistemas distribuidos.

Hoy no es considerado middleware, de acuerdo a las definiciones antes citadas, pues RPC requiere que los programadores reescriban sus aplicaciones una y otra vez a medida que las aplicaciones cambian o se multiplican. Y esto no es coherente con la simplicidad que enuncia en su definición.

No obstante en muchos libros de Sistemas Distribuidos (Colouris), se sigue considerando parte del middleware.

Por ejemplo, MOM, al contrario que RPC, es una forma de comunicación asincrónica, donde el sender no hace Block waiting a la espera de respuesta.

Con las definiciones vertidas hasta aquí, puede interpretarse erróneamente que el middleware sólo permite trabajar con ambientes totalmente distribuidos.

Es importante considerar que los servicios que se quieran ofrecer descansan sobre una infraestructura, donde una parte puede requerir centralización, y otras pueden ser mantenidas por otros departamentos.

Por lo tanto, un objetivo importante es no perder de vista que a veces se hace necesario crear un servicio centralizado que provea autorizaciones, herramientas, para poder implementar la distribución.

El middleware debe proporcionar servicios para que sean usados por parte de las aplicaciones: servicios para gestión de nombres, seguridad, notificación de eventos, almacenamiento persistente, etc.

3.4 Notas

Para aprender sobre tecnologías de componentes en Microsoft, fueron muy útiles [KIR002], [MIC001], [SCR002] y [PLA002].

El artículo de Berkeley [BER097], trata de las diferencias entre DCOM y CORBA.

En cuanto al concepto de middleware el artículo [MID000]. También contribuyó [COU001].

Capítulo IV

Web Services

4 Web Services (WS)

Este nuevo modelo, debe permitir la interacción entre servidores, servidores y clientes, y clientes entre sí.

A medida que crece la necesidad de comunicar aplicaciones para lograr la interoperabilidad, se va haciendo más necesario el uso de Web Services, que provean un medio de comunicación standard entre diferentes aplicaciones de software que corren en distintas plataformas y frameworks.

4.1 Conceptos y ventajas

Los Web Services pueden verse más como una evolución en el campo de los sistemas distribuidos que como una revolución. El uso de aplicaciones XML ha sido un paso definitorio para esta evolución.

Según Graham Glass, Web Services es *una colección de funciones que están empaquetadas en una entidad simple y ofrecida en la Red para que otros programas la usen. Son building blocks para crear sistemas distribuidos y permitir que empresas y usuarios en general puedan lograr, de una forma rápida y barata, su inserción en la Web.*

El objetivo es desarrollar aplicación distribuidas altamente integradas que interactúen por XML entre los servicios Web y múltiples servicios WEB.

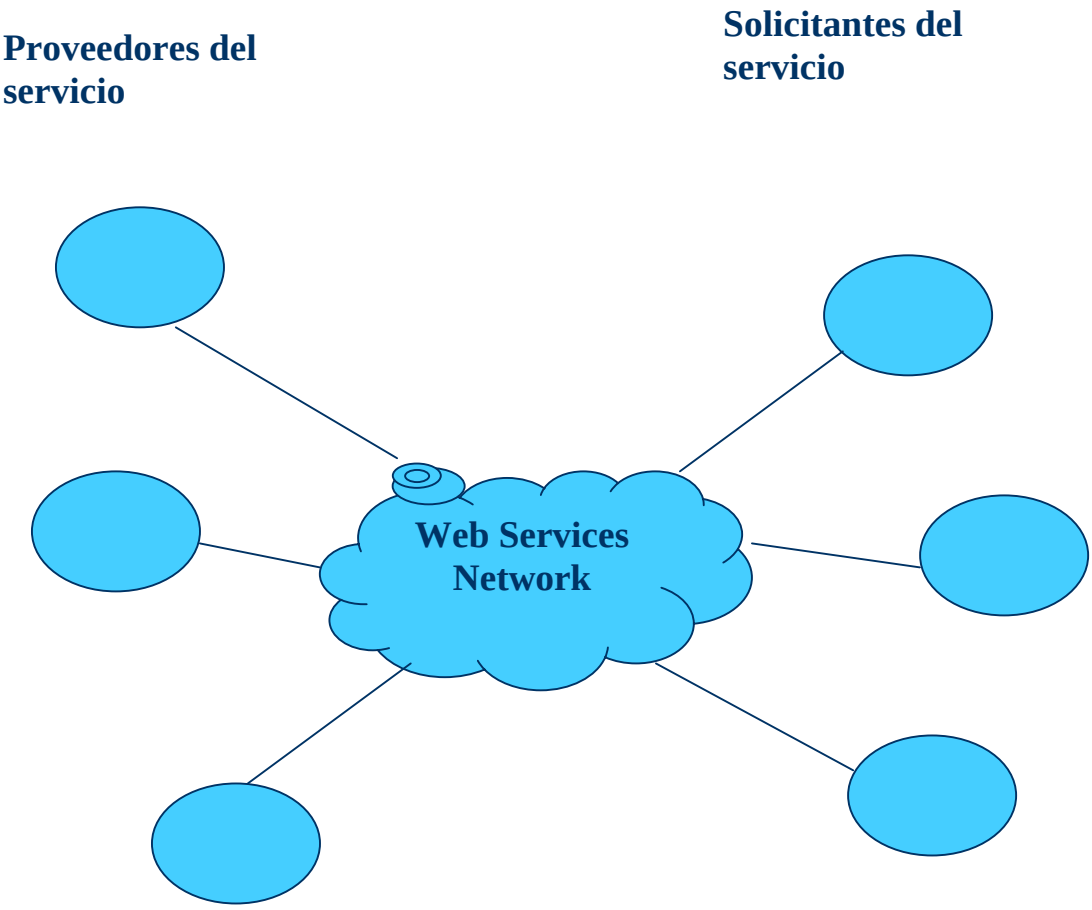
El uso de XML, que será analizado en otro capítulo, se prioriza en este modelo por permitir la transmisión de información auto descriptiva a una plataforma neutral.

Supongamos que un usuario visita un sitio Web, y quiere transmitir información directamente a una aplicación cliente, para su proceso, lo cual hoy se hace, la mayoría de las veces, con la intervención del usuario. El objetivo es que las aplicaciones en Internet puedan comunicarse entre sí. El usuario podría obtener un resultado del procesamiento de información proveniente de distintos sitios.

Otra situación ilustrativa es que el usuario pueda usar distintos dispositivos cliente, con resoluciones de pantalla diferentes, o utilizar un nuevo dispositivo. De manera transparente para el usuario, la aplicación en el servidor, generará XML, y la transformación necesaria para el dispositivo particular, lo cual se puede hacer en el servidor o en el cliente. De allí la necesidad que todos los dispositivos tengan un soporte para XML, para implementar esta interacción.

En caso de utilizar un nuevo dispositivo, éste se autentica, y elegirá cuáles servicios usará para manejar la información y recuperarla.

Web services networks actúan como intermediarios en las interacciones de los Web services.



Hoy, la computación ubicua (ubiquitous computing), es una necesidad. Llamamos computación ubicua al uso de dispositivos pequeños y baratos presentes en el entorno de usuario tanto en el hogar como la oficina. Debemos diferenciarla de la computación móvil o nómada, que es la que permite la realización de tareas de cómputo mientras el usuario se traslada o está en lugares diferentes a su entrono habitual.

Por lo tanto, se requiere la innovación tecnológica para cubrir estas necesidades de la forma más transparente posible.

Veamos por ejemplo, como evoluciona la distribución del software. Hasta no hace mucho tiempo, la única posibilidad era distribuir las aplicaciones en CD's con actualizaciones periódicas. La alternativa de cambio que estamos analizando debe ofrecer una conexión directa con el proveedor para la entrega de actualizaciones, soporta, etc. En fin: servicios. El cambio debe estar orientado al servicio a través de la Red.

Analicemos ahora la relación del usuario con las interfases gráficas. Este modelo debe permitir que la interfase se presente al usuario según una personalización, basada en la identidad. Ante la necesidad de parte del usuario de agregar reconocimiento de voz, no sería necesaria la adaptación de la aplicación para, por ejemplo, un usuario con algún tipo de discapacidad visual o auditiva.

Si el usuario se autentica en algún site, deberían viajar las personalizaciones propias que regirán la relación con otros clientes o servidores.

Por lo tanto, con estas pautas, se deduce la necesidad de usar agentes inteligentes, para que en los servicios Web se pueda trabajar sobre el tipo de información que el cliente quiere recibir.

Para lograr la implementación del modelo es necesario el avance tanto del HW como del SW para soportar los protocolos de alto nivel.

Los servidores para implementar este modelo deben permitir la escalabilidad desde el SO para mostrar un conjunto de servidores como un único servidor lógico.

Los beneficios de utilizar WS se manifiestan en:

- Usa standards abiertos, basados en texto. Se pueden comunicar componentes escritos en diferentes lenguajes y distintas plataformas.
- Fácil de implementar. No es costoso: se usa una infraestructura existente.
- La mayoría de las aplicaciones pueden re-empaquetarse como Web Service.

Ethan Cerami, autor de Web Services Essentials (CER002), se basa en 10 preguntas para guiar al lector en la comprensión del concepto de Web Services y links relacionados.

Esas preguntas son las siguientes:

1. *What is a Web service?*

2. *What is new about Web services?*
3. *I keep reading about Web services, but I have never actually seen one. Can you show me a real Web service in action?*
4. *What is the Web service protocol stack?*
5. *What is XML-RPC?*
6. *What is SOAP?*
7. *What is WSDL?*
8. *What is UDDI?*
9. *How do I get started with Web Services?*
10. *Does the W3C support any Web service standards?*

Estas 10 preguntas, acertadamente escogidas, son una guía para analizar los conceptos relacionados con el tema.

Web Services ha cambiado no sólo la forma de concebir el desarrollo de aplicaciones, sino las aplicaciones en sí. Para poder entrar en el mundo de los Web Services se hace necesario aclarar, redefinir y definir algunos conceptos.

Hoy, el desafío es asociar las nuevas interfases a aplicaciones ya existentes, componer múltiples Web Services para lograr un business service e interoperar con ambientes multivendor.

4.2 Arquitectura de los Web services

Definición avalada por W3C, donde "*Web Service es un sistema de software identificado por una URI, cuyas interfases y enlaces (bindings) son definidos y descriptos usando XML. Este sistema puede ser "descubierto" y usado por otros sistemas de software. Esa interacción debe darse según la manera descripta en la definición, usando mensajes de acuerdo a protocolos de Internet*".

En esta definición no se nombra SOAP ni WSDL. Otros mecanismos pueden usarse para empaquetar XML y otra forma de descripción, como DAML-S, en vez de WSDL.

No obstante, WSA (Web Service Architecture, de W3C) acuerda lo conveniente de basarse en una base común para messaging y descripción (lo define como "una necesidad práctica").

Cuando hablamos de Arquitectura de Web service, nos referimos a identificar las componentes funcionales, la relación entre ellas y un conjunto de restricciones para lograr que funcione como deseamos. Por lo tanto, de acuerdo a los requerimientos funcionales, se definen las tecnologías adecuadas.

Las fuentes que movilizaron el surgimiento de los Web Services fueron: la integración de aplicaciones y distribución de objetos, el intercambio de documentos electrónicos sobre la red (EDI en ambientes B2B[4]), y la propia WWW, en su necesidad de acceder a documentos legibles por los humanos, además de los requerimientos de servicio, posting, products.

[4] B2B: business to business

La **Arquitectura básica de Web Services** debe incluir lo necesario para:

- el intercambio de mensajes entre las entidades
- publicar en la Red, poner en conocimiento de la existencia de ese Web Service, para que pueda ser utilizado por algún cliente
- encontrar otros Web Services

El esquema cliente servidor se plantea como el intercambio de mensajes donde ahora el cliente es un agente móvil que solicita un servicio y el proveedor del servicio, otro agente móvil. A su vez, el proveedor puede solicitar un servicio transformándose a su vez en cliente.

4.3 Dinámica de los WS. Protocolos

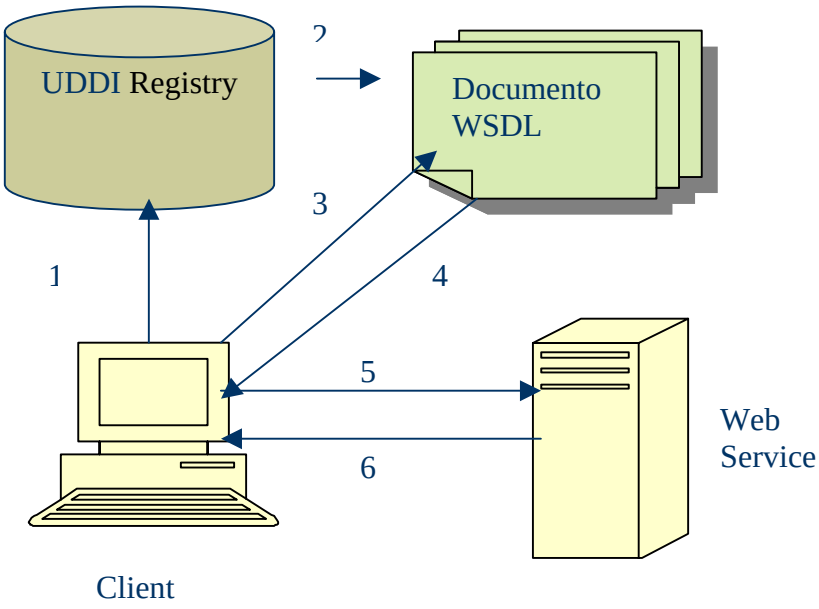
Una de las partes interesantes (y fundamentales) de Web Services es la combinación de protocolos standard sobre Internet. Se enfatiza la necesidad de usar standards para el intercambio de datos entre aplicaciones de distintas plataformas y lenguajes.

¿Porqué hablar de SOAP, XML, WSDL? ¿Porqué no seguir con COM, o CORBA?

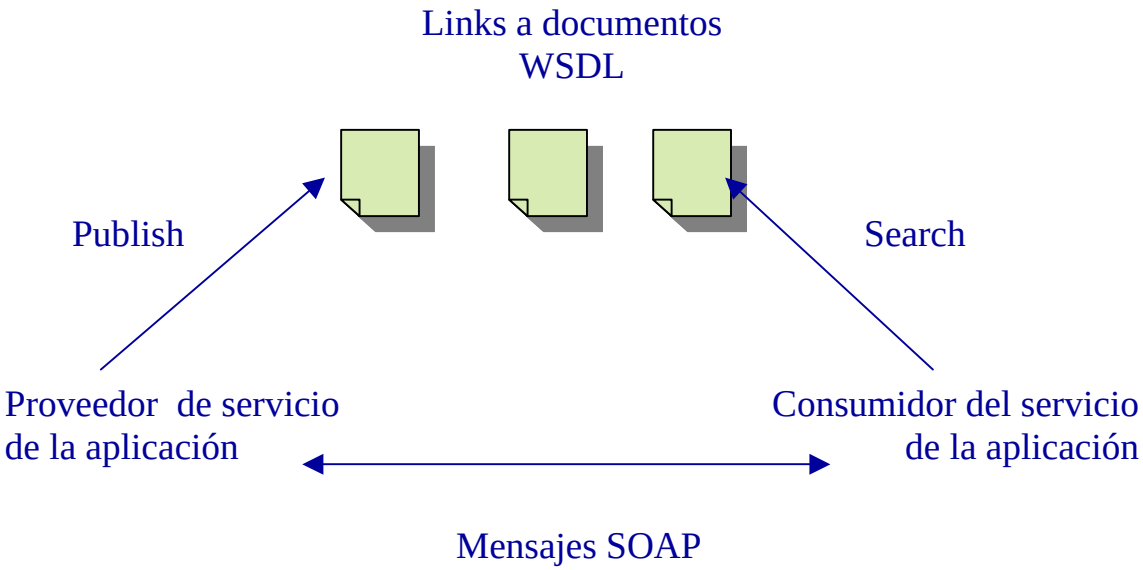
Básicamente es el modelo de interoperabilidad de plataformas y aplicaciones lo que exige optar por estos protocolos. COM (de Microsoft), y OMG CORBA no interoperan, y no han demostrado la extensibilidad que se quiere lograr sobre la Red.

El uso de protocolos standard es necesario para lograr la Interoperabilidad en ambiente heterogéneos, con Independencia de SO, Lenguaje, versiones.

- HTTP en transporte
- XML para codificar los datos
- WSDL para describir el servicio
- SOAP para invocar llamados remotos y “envolver” el intercambio
- UDDI, para publicarlos (serían las páginas amarillas de servicios Web)



- 1. Cliente pregunta al registry para ubicar un servicio.
- 2. Registry le indica al cliente un documento WSDL.
- 3. Cliente accede al documento WSDL.
- 4. WSDL provee lo necesario para interactuar with Web service.
- 5. Client envía un requerimiento usando SOAP.
- 6. Web service retorna una respuesta SOAP.



Podemos resumir, que los WS tienen una dinámica:

1. Cliente pregunta al registry para ubicar un servicio.
2. Registry le indica al cliente un documento WSDL.
3. Cliente accede al documento WSDL.
4. WSDL provee lo necesario para interactuar con Web service.
5. Client envía un requerimiento usando SOAP.
6. Web service retorna una respuesta SOAP.

La interacción se da a través de un patrón de mensajes (message exchange patterns, MEPs) que definen la secuencia de intercambio.

Se debe definir una descripción del web service, que indique su funcionalidad.

Descripción incluye: como invocar el Web Service y qué retorna. Involucra tipos de datos, estructura, MEPs, y la dirección del proveedor del servicio.

En esta arquitectura hay tres roles bien definidos:

- Proveedor del servicio
- Lugar o guía para la búsqueda de servicios
- Solicitante del servicio

Uno de los pilares de esta tecnología es la reusabilidad del software. Por lo tanto, el desarrollador "publica" el web service desarrollado para que esté disponible para el cliente que lo necesita.

Para ello, se debe "registrar" el web Service en un lugar que se llama ***service discovery agency***, done el proveedor "hostea" el módulo de software para que sea accesible desde la Red.

La interacción se da a través de un patrón de mensajes, **MEP**s, que definen la secuencia de intercambio.

El proveedor debe además, definir una descripción del web service para que el cliente analice si realmente otorga la funcionalidad que necesita. En esa descripción también explica como invocar el Web Service y qué retorna, por lo tanto, involucra tipos de datos, estructura, MEPs, y la dirección del proveedor del servicio.

Un agente en este tipo de arquitectura, puede ser proveedor, solicitante o cumplir ambos roles, discovery agency y los otros dos.

Cuando el solicitante "descubre" el servicio en la agency, lo puede invocar de acuerdo a la descripción ofrecida, para que se cumpla el binding.

Lo interesante de tecnologías que usan XML, SOAP, es que se pueden integrar en aplicaciones ya existentes sin demasiada complejidad.

Productos que usan SOAP pueden usar servidores HTTP existentes y procesadores XML que tenga el cliente.

Si, en cambio, se utiliza CORBA, DCOM o Java RMI, se debe instalar el ambiente apropiado, configurar el sistema para adaptarlo a la nueva infraestructura distribuida.

También puede exigir reconfigurar firewalls. De allí que a estos sistemas se los llama “heavyweight systems”.

La computación ubicua requiere un conjunto de protocolos disponibles y listos para ser usados, que implemente:

- un mecanismo de serialización que transforme la llamada al método en la forma adecuada para su transmisión sobre la red
- una capa de transporte que lleve esos datos inherentes al método entre los sistemas remotos
- un mecanismo para encontrar, activar y desactivar objetos distribuidos
- un modelo de seguridad para proteger el sistema local y remoto.

Cuando el objetivo es la interoperabilidad, la elección de los protocolos debe ser cuidadosa.

Por ejemplo, sistemas distribuidos de objetos como COM, de Microsoft, y OMG CORBA son standards que no interoperan.

Por lo tanto, se hace imprescindible adaptarse a los standares.

Usar XML no garantiza por si sólo la interoperabilidad: la combinación del conjunto de protocolos que se utilicen, son los que me garantizarán la interoperabilidad deseada.

La tecnología SOAP 1.1 y WSDL 1.1 se desarrollaron en principio fuera de W3C, pero sus sucesores se trabajaron dentro del consorcio.

La especificación SOAP 1.2 está siendo usada como la base para crear un mesagging framework escalable, y WSDL 1.2 se usa como el lenguaje de definición de interfases.

4.4 Operaciones

En WS, las operaciones básicas son publicar (publish), encontrar (find) e interactuar (interact).

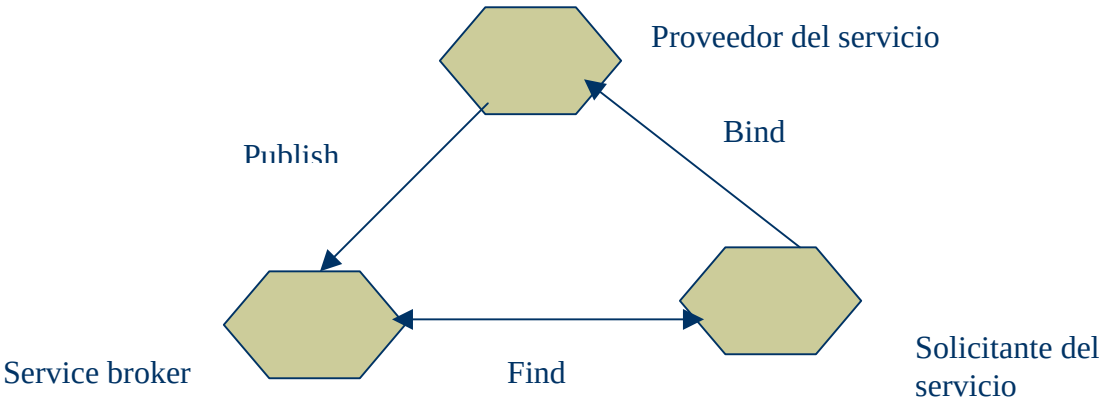
Publish: el desarrollador publica su WS para ofrecerlo a la comunidad de potenciales usuarios del servicio. Lo que se publica es la **descripción del servicio**.

Find: a través de esta operación, un desarrollador busca un servicio directamente o haciendo queries al registro de servicios.

La operación se invoca tanto cuando se busca la descripción del servicio, como cuando se desea acceder al servicio en runtime (*binding invocation)*.

Interact: en tiempo de corrida (runtime) el que requiere el servicio inicia una interacción al invocar el WS, haciendo uso de la descripción para ubicación, contacto e invocación del servicio.

La interacción puede ser sincrónica o asincrónica. El modelo puede ser desde broadcast a varios servicios, procesos de negocios (business process), conversación multilenguaje, etc.



La enunciada es la arquitectura básica. Esa arquitectura puede extenderse sumando otras características, entre otras:

- Attachmnets: permitir empaquetar varios documentos. Para esto se usa XML Protocol Attachment. Se trabaja sobre SOAP with attachment y DIME.
- Caching
- MEPs (message exchange patterns)
- Confidencialidad (usar SSL o TSL o SOAP con encriptación/desencriptación), autenticación (autenticación HTTP o SOAP con username/password, X.509), integración (uso de firma digital sobre digest provisto por un módulo SOAP) y ruteo de mensajes.

A este tipo de arquitectura estaría encuadrada en lo que se llaman arquitecturas orientadas al servicio (service oriented architecture). Esta arquitectura se basa en el intercambio de mensajes entre componentes. En este caso se identifican tres roles y tres operaciones entre esos roles, encontrando y usando los servicios dinámicamente.

4.5 Web Services stacks

Para poder utilizar las operaciones que enuncia la arquitectura básica es necesario identificar el stack de tecnología que se pude usar.

Este stack tendrá distintos niveles o copias pero algunas capas pueden no ser independientes de otras.

Analicemos el ***wire stack***. El wire stack comprende conceptos y tecnologías que trata con el intercambio físico entre los distintos roles del WS.

Incluye transporte de red, empaquetado de mensajes y extensiones de mensajes (para facilitar intercambio de datos).

Obviamente, los WS deben estar accesibles a través de la red.

HTTP es el standard de facto en protocolo de red para WS, pero también se soporta SMTP y FTP.

En los dominios intranet, pueden usarse plataformas o protocolos específicos como Mqseries, CORBA, etc.

En un ambiente de múltiples infraestructuras de red HTTP es la opción para lograr interoperabilidad.

En la capa de empaquetamiento, se trata la tecnología que se puede usar para el intercambio de información.

XML ha sido adoptado como protocolo para el empaquetado de mensajes en WS.

A través de SOAP, basado en XML, se pueden crear paquetes de datos estructurados.

A continuación se reproduce el gráfico sobre el stack de protocolos en WS, en forma conceptual.

<table>
<tr><td>WSFL</td><td>Flujo de servicio</td><td rowspan="6">Seguridad
Privacidad</td><td rowspan="6">Management</td><td rowspan="6">Calidad del servicio</td></tr>
<tr><td rowspan="2">UDDI</td><td>Servicio de búsqueda</td></tr>
<tr><td>Servicio de Publicación</td></tr>
<tr><td>WSDL</td><td>Descripción del servicio</td></tr>
<tr><td>SOAP</td><td>Mensaje basado en XML</td></tr>
<tr><td>HTTP, FTP, MQ, etc</td><td>Red</td></tr>
</table>

4.6 Web Services estático y dinámico

En los estáticos, el cliente invoca a un proveedor fijo y ubica el archivo WSDL a través de mail, ftp, ó registro UDDI. Luego, la aplicación cliente, invoca al proveedor de WS.

En los dinámicos, no se sabe a qué proveedor se va a terminar invocando.

La aplicación cliente interactuará con el registro UDDI a través de APIs. Recupera la información desde el registry y busca a los proveedores que ofrecen ese servicio y entonces, invoca.

4.7 ws-i: Web Services y standares

WS-I es una organización que promueve la interoperabilidad entre plataformas y lenguajes de programación usando WS.

Hace recomendaciones y provee una guía de trabajo.

Fue fundada para crear y promover el uso de protocolos genéricos para el intercambio de mensajes entre servicios.

Protocolos genéricos son protocolos que son independientes de cualquier acción especifica que no sea la seguridad, confiabilidad o entrega eficiente del mensaje.

El WS-I Basic Profile 1.0 es un conjunto de especificaciones de Ws no propietarios, junto con aclaraciones y recomendaciones para promover la interoperabilidad.

El Profile se basa en distintas pautas, entre otras:

- No hay garantía de interoperabilidad. No se puede garantizar completamente la interoperabilidad de un servicio, pero se pueden definir pautas para soluciones a los problemas que surgen de la experiencia.
- El Profile define claramente y sin lugar a ambigüedad las restricciones, en los términos MUST , MUST not.
- Cuando se hace necesario extender o modificar una restricción, se trata de no relajarla (no cambiar un MUST por un MAY).
- Ante múltiples mecanismos que puedan usarse indistintamente para una especificación, el Profile selecciona aquél que sea clara, utilizable y útil.
- Se analiza el crecimiento de la especificación en sucesivas revisiones (SOAP 1.2, por ejemplo).
- El Profile pone especial foco en la interoperabilidad.
- El Profile asume que existe interoperabilidad en la capas bajas del protocolo. Por lo tanto, trata la interoperabilidad a nivel de la capa de aplicación.

A continuación se indican las versiones y RFC´s de las especificaciones del profile.

- Simple Object Access Protocol (SOAP) 1.1
- Extensible Markup Language (XML) 1.0 (Second Edition)
- RFC2616: Hypertext Transfer Protocol -- HTTP/1.1
- RFC2965: HTTP State Management Mechanism
- Web Services Description Language (WSDL) 1.1
- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes
- UDDI Version 2.04 API Specification, Dated 19 July 2002

- UDDI Version 2.03 Data Structure Reference, Dated 19 July 2002
- UDDI Version 2 XML Schema
- RFC2818: HTTP Over TLS
- RFC2246: The TLS Protocol Version 1.0
- The SSL Protocol Version 3.0
- RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile

4.8 Notas

Bueno... considerando el tema de este capítulo, podemos decir que es donde más bibliografía consulté.

Los Working draft de la W3C y notes, fueron sumamente claras y me permitieron asegurarme cuando hablábamos de standares y cuando de standares dentro de una empresa... ;-)

Cito como *SUMAMENTE INTERESANTES* [AUS002], [CER002], [CHA001], [GLA000], [SAL000].

Los artículos de Seven, [SEV001] y [SEV002], si bien se orientan a .NET, son claros, con ejemplos, y el libro [CHA002].

Un sitio de cabecera es **http://www.ws-i.org**, el sitio "oficial" de WS-I.

El Basic Profile, **http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0a.html#references**, es de lectura obligada para determinar cuáles protocolos genéricos y versiones se están usando, y toda la terminología asociada.

Otras lecturas, si bien son específicas de otros capítulos, como los orientados a .NET, o a Websphere contribuyeron a la elaboración de éste.

Capítulo V

El proceso de negocio en WS

5 El proceso de negocio en WS

Web Services permite la interoperabilidad entre aplicaciones. Hasta ahora hemos definido el espacio de los WS a través de SOAP, UDDI, WDSL, XML.

Pero nuestro objetivo es la interacción de sistemas heterogéneos ya sean BC (business-consumer), B2B (business to business) o EAI (entreprise application integration). Y hasta aquí hemos visto WS como la forma de integrar sistemas a través de la interacción simple de protocolos standard. No hemos incorporado al proceso de negocio como parte de este modelo. Sin él, no hay verdadera integración.

Los negocios se benefician con el efecto network. Para ejemplificar este efecto se utiliza el caso de la máquina de Fax: la primera, única en el mercado, no tenía ningún beneficio. Cuando 100 personas tuvieron máquina de fax, empezó a considerarse productiva, pues permitía la interacción de esos 100 usuarios. Cuando se extendió a millones, pasamos hablar de una poderosa herramienta para el crecimiento del negocio.

Lo mismo ocurre con las aplicaciones. Si permito que cada vez más usuarios se conecten a ellas, facilitando la conexión, amplío mi posibilidad de negocio.

IT (Information Technology) necesita cada más interoperabilidad, a menor costo, y con un mecanismo de negocio soportado por la mayor cantidad de empresas para trabajar juntos.

Las relaciones de negocio, por ejemplo entre un cliente y un proveedor, se componen de diferentes partes, unas públicas, otras privadas.

Una lista estimativa de precios puede ser pública, también los servicios que se ofrecen. Pero el cálculo del precio final o qué se asume como costo a incorporar al producto, o la justificación de un rechazo de pedido, puede desearse mantener en privado.

Por ello, en el proceso de negocio se habla de partes “visibles” de la interacción (la pública) y otras “no visible” (la privada).

Otra razón por la que puede ser importante separar estas partes es que los cambios en los aspectos privados de la implementación de negocio (por ejemplo, una nueva forma de fabricar un producto, o el cambio de las relaciones con los proveedores de la materia prima) no afecten la parte pública.

5.1 Transparencia y opacidad

Los datos transparentes (de uso público o externo) afectan al protocolo de manera directa. Los opacos (de uso privado o interno) lo afectan de manera no determinística.

Por ejemplo, en la relación vendedor-comprador, el vendedor ofrece un servicio. El comprador libera una orden de compra que el vendedor acepta o rechaza. Esta decisión se basa en un criterio, pero este proceso de decisión es “opaco”. Cómo se

refleja esta opacidad en el protocolo externo? Enumerando alternativas que se presentarán como el resultado: el vendedor tiene derecho a reservarse el criterio por el cual acepta o rechaza una orden de compra

5.2 Niveles en el acuerdo de partes

Se acuerdan tres niveles entre las partes del negocio: nivel de transporte, de documentos de negocio y proceso de negocio.

El nivel de transporte es responsable por mover los mensajes o documentos entre una parte y la otra, de una manera segura, confiable y consistente. Incluye el aspecto transaccional (considerando que hablamos de aplicaciones distribuidas).

El nivel de documento se enfoca hacia el contenido del mensaje. El documento debe ser entendido por ambas partes.

El tercer nivel es el nivel de proceso de negocio. Define la interacción, los procesos internos y externos de ambas partes.

Considerando el efecto network citado anteriormente, es importante que la mayor cantidad de nodos soporten los mismos protocolos de la capa de transporte. Para cumplir este objetivo, en estos protocolos se den cumplir estos cuatro requerimientos:

1. Deben ser neutrales en cuanto a lenguaje, sistema operativo, modelo de bases de datos y arquitectura.
2. Deben ser modulares y “componibles”.
3. Deben ser de propósito general adecuados para la integración de los distintos ambientes (BC, B2B, procesos internos, etc.).
4. Deben permitir la extensibilidad de las aplicaciones.

5.3 Conceptos asociados al proceso de negocio

Se llaman procesos ejecutables de negocio (Executable Business Processes) a los procesos que modelan el comportamiento real de las partes que participan en una interacción.

Los Protocolos de negocios (Business Protocols) son protocolos que especifican el intercambio de mensajes de las partes que intervienen (parte visible). No se ocupan del comportamiento interno (parte “no visible”).

Las descripciones de estos procesos que manejan estos protocolos se llaman procesos abstractos (abstract processes).

Una forma de modelar los procesos ejecutables y los abstractos, puede implementarse a través de BPEL4WS. Es un lenguaje que permite especificar formalmente los procesos de negocio y la interacción de las partes, usando los protocolos standard.

Esta funcionalidad extiende la interoperabilidad de los WS, al incorporar la formalización de los procesos de negocio.

5.4 Business Protocols

La interacción en los negocios normalmente se establece a través del intercambio de mensajes entre las partes de una forma peer-to-peer, sincrónica o asincrónica. Para poder formalizar esta interacción se define un protocolo (business protocol).

Estos protocolos deben describirse independientemente de la plataforma. Su inherencia abarca todos los aspectos de significancia dentro de una empresa, en el ambiente de negocio.

Estos protocolos cuentan con estas características:

- Hay dependencia de datos que intervienen en la interacción. Por ejemplo, en una orden de compra, hay un límite de la cantidad de productos por orden, tiempo de entrega, etc.
- Tener en cuenta condiciones de excepción y las consecuencias. Es tan importante prever el comportamiento cuando todo sale bien, como cuando hay fallas o excepciones.
- Prever la interacción de diferentes unidades de trabajo, cada una con sus propios requerimientos.

5.5 Algunas características de BPEL4WS

Cuando los protocolos tratan con los procesos internos ejecutables, no se deben describir los datos, pues son privados. En BPEL4WS se puede identificar dentro del mensaje la parte que es pública de la que es privada.

Permite esconder los datos opacos para no hacerlos visibles.

La interacción entre las partes participantes del negocio se implementan a través de interfases WS.

Se llama *service link* al nivel de la interfase que implementa la interacción entre las partes del negocio.

Un proceso en BPEL4WS define la interacción entre servicios y partes, coordinación, estado y lógica.

También contempla el manejo de excepciones y fallas, considerando dentro de un proceso qué actividades deben hacerse en caso de excepción o revocación de un servicio.

En BPEL4WS distingue entre procesos abstractos y ejecutables.

Los procesos abstractos describen aspectos públicos del protocolo. Usan *message propierties*.

Cuando se definen los procesos ejecutables, se determina la secuencia de los WS con que interactúan partes y protocolos.

BPEL4WS tiene una estrecha relación con WSDL. Los procesos y servicios son modelados como servicios WSDL.

A continuación se nuestra el formato de un documento WSDL. Este diagrama se repite en 8.3.

Definiciones abstractas	**Types**	*Contiene definiciones tipos independientes de máquinas y lenguajes*
	Messages	*Contiene definiciones sobre parámetros de funciones o descripciones de documentos.*
	Port types	*Se refiere a definiciones de Messages, para descripción de nombres, parámetros de input y output.*
Definiciones concretas	**Bindings**	*Especifica los bindings según lo especificado en Port Types*
	Services	*Especifica la dirección de cada binding.*

En PortType se definiría por ejemplo, el cálculo del precio, la selección del embarque o envío y planificación, producción.

En el documento WSDL en BPEL4WS, no habrá ni binding ni service pues estamos definiendo en abstracto. Sólo se detalla el PortType.

5.6 Notas

Me gustó mucho este tema. Debo confesar que me hubiera gustado extenderme mucho más, pero, hubiera "opacado" el objetivo principal.

De los artículos consultados, destaco *ESPECIALMENTE* [CUR002] y [RIO002].

También recomiendo [GAR002].

Creo que la automatización de los procesos de negocio es un tema poco implementado y que integrado con web services, puede ser un interesante tema de investigación.

En este punto, no obstante, es importante establecer la standarización de los lenguajes de notación formales.

BPEL4WS supera a *XLANG* y *WSFL*, y es la convergencia de ambos. La ventaja de este lenguaje es que surge de BEA, Microsoft, IBM y se basa fuertemente en los protocolos genéricos de ws: WSDL, XML, etc.

Capítulo VI

XML

6 XML

6.1 Qué es XML?

XML proviene de eXtensible Markup Language, es decir, lenguaje extensible de marcado. Fue desarrollado por W3C para superar las limitaciones de HTML.

La ventaja de HTML está dada por que es soportado por la mayoría de las aplicaciones. Ocurre con el software de correo electrónico, exploradores. editores, bases de datos, etc.

Pero HTML fue creciendo desde su primer versión. Hoy la gran cantidad de etiquetas y sus combinaciones, sumado a nuevas tecnologías de soporte (JavaScript, Flash, CGI, ASP, etc.), y las aplicaciones van necesitando aun más etiquetas.

El advenimiento de pequeños clientes menos poderosos que las PCs no permiten procesar un lenguaje de acceso a la WEB muy complicado.

Por ello se desarrolló XML. Se basó en HTML, dada su popularidad, su éxito. Pero XML vino para cumplir con nuevas demandas.

Hay dos clases de aplicaciones XML: la publicación electrónica y el intercambio de datos.

Los dos cambios esenciales que XML hace a HTML es no predefinir etiquetas y ser estricto.

Veamos un ejemplo como definiríamos en XML una etiqueta para precio:

```
<precio moneda=”pesos”>850.00</precio>
```

La X de XML proviene de eXtensible, porque no predefine etiquetas, pero le permite al usuario crear las que precise para su aplicación.

Cómo sabe el explorador que una definición XML equivale a un conjunto de etiquetas en HTML? Es a través de las hojas de estilo.

XML es compatible con la generación actual de exploradores.

Otra ventaja, que, desde el desarrollador puede verse como no tan beneficiosa, es la sintaxis estricta de XML. HTML no la tiene.

La falta de dicha exigencia, si bien es cómoda para el desarrollador, requiere exploradores cada vez más complejos.

Al tener una sintaxis estricta XML puede ser usado en exploradores simples como exigen los dispositivos de mano actuales.

A los archivos XML se les llama documentos. Esto hace que pueda confundirse el uso de XML y pensarlo sólo como una solución para la publicación electrónica. Más adelante veremos la importancia y trascendencia que ha tenido XML en el intercambio de datos.

6.2 Estructura de un documento XML

En general, en un documento podemos distinguir:

- ***Título***
- ***Encabezado***, donde se detalla remitente, destinatario, asunto
- ***Texto***, con párrafos, donde pueden incluirse URLs, firmas.

Separemos la estructura del documento de su apariencia.

La estructura del documento da la base para la apariencia o presentación.

Cuando se intercambian documentos se trata de asegurar su apariencia para que el despliegue sea casi idéntico en las distintas plataformas.

XML actúa de manera tal que graba la estructura de documento que se deriva de su formato.

Algunas definiciones

Qué es el marcado?

En la publicación tradicional es una actividad independiente que se realiza después de la escritura de un documento y antes de la composición de las páginas.

Estas indicaciones guían al tipógrafo en cuanto a la apariencia del documento (fuentes, destacados, etc.).

El documento electrónico es el código que se inserta en el texto del documento que almacena la información necesaria para el procesamiento electrónico. En el caso de un documento XML, el marcado contribuye a identificar la estructura.

Qué es el marcado procedural?

En el procesamiento de texto, el usuario especifica la apariencia del texto (negrita, párrafo en cursiva, fuente, o una posición particular del texto dentro de la página).

Esto se almacena como código dentro del texto. Se le llama marcado procedural pues deriva en un procedimiento para un dispositivo de salida, por ejemplo, la impresora.

El RTF, Formato de Texto Enriquecido, es un marcado procedural desarrollado por Microsoft, y está soportado por la mayoría de los procesadores de texto.

No obstante sus ventajas, adolece de algunos problemas:

1. No almacena información sobre estructura.
2. No es flexible (cambios en las normas de formato, exigen cambios en el documento).
3. Es un proceso lento y propenso a errores.

Qué es la codificación genérica?

En la codificación genérica se utilizan macros que pueden mejorar el marcado procedural.

En vez de implementar controles, se puede invocar procedimientos externos de formato. La etiqueta (o identificador genérico) se agrega al elemento de texto. estas etiquetas están asociadas a normas de formato.

Tex es un ejemplo de codificación genérica.

Los beneficios con respecto al marcado son:

1. Hay mayor portabilidad y flexibilidad
2. Se graba información sobre estructura.

6.3 SGML

Extiende la codificación genérica al agregar descripción sobre estructura del documento y se ajusta a un modelo, lo que permite ser procesado por software o guardarlo en una base de datos.

A través de SGML empieza la corriente por la cual los autores cuentan con un lenguaje para descubrir la estructura de sus documentos y los marquen.

La gran diferencia entre la codificación genérica y el SGML es que el marcado describe la estructura del documento.

La estructura del documento se describe en una DTD (data type definition) que también recibe el nombre de aplicación SGML. En esta DTD se especifican elementos, relaciones y etiquetas que definen una estructura de documento y lo marcan.

En SGML el marcado sigue a un modelo. Algunos lenguajes se basan en SGML como es el caso de HTML (lenguaje de marcado para documentos WEB), CALS (standard para intercambio de documentos del DoD, Department of Defense of EEUU), DocBook, etc.

6.4 HTML

Es la aplicación más popular de SGML. Es un conjunto de etiquetas que siguen la norma SGML.

HTML ha evolucionado, considerando que:

- integra etiquetas de formato logrando características de marcado procedural
- agrega el atributo class y hojas de estilo, logrando características de codificación genérica.

6.5 Las aplicaciones XML. Documentos y datos.

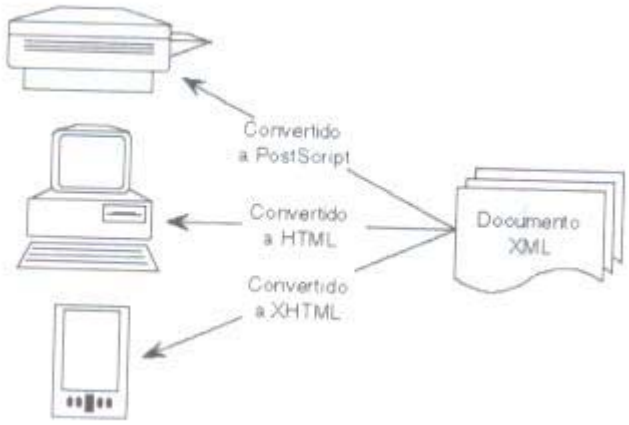
Las aplicaciones XML se clasifican en:

- Aplicaciones orientadas a documento, información dirigida al consumo humano;
- Aplicaciones de datos que manejan información dirigida al software.

Es una diferencia cualitativa. El standard es el mismo.

Aplicaciones de documento

Veamos la publicación de un documento que quiere imprimirse, mostrarse en la WEB y llevarlo a una palm.



XML me da la posibilidad de publicarlos automáticamente en distintos medios.

Aplicaciones de datos

Si la estructura de un documento puede expresarse como XML, también la estructura de una base de datos.

Veamos como incluiríamos una lista de productos con su precio en XML.

```
<?xml version="1.0" enconding ="ISO-8859-1"?>
<productos>
        <id producto="p1">
                <nombre>Libro de Silverschatz</nombre>
                <precio>140</precio>
        </producto>
        <id producto="p2">
                <nombre>Libro de Tenenbaum</nombre>
                <precio>150</precio>
        </producto>
        <id producto="p3">
                <nombre>Libro de Stallings</nombre>
                <precio>180</precio>
        </producto>
</productos>
```

El uso de XML en los canales es otro ejemplo sobre como las aplicaciones pueden beneficiarse con XML. Un canal es un sitio Web al cual uno se puede suscribir. Es un concepto que introdujo Internet Explorer 4.0.

Atrás de cada icono en la barra de canales que muestra el Explorer, hay un documento XML. Los canales se describen mediante CDF (Formato de definición de Canales). CDF es una aplicación XML.

Veamos como es un archivo CDF de un canal.

```
<?xml version="1.0" enconding ="ISO-8859-1"?>
<CANAL BASE=http://www.canalsoft.com/boletin/  HREF="index.html"
PRECACHE="YES" LEVEL="0">
        <TITULO>canalsoft Link</TITULO>
        <RESUMEN>Boletin mensual gratis</RESUMEN>
        <LOGO HREF="logo.gif" ESTILO="IMAGEN"/>
        <AGENDA FECHAINICIO="2004-01-01">
                <TIEMPOCAMBIO DIA="14"/>
        </AGENDA>
</CANAL>
```

Se describe, con la sintaxis XML, el canal, la frecuencia de actualización (AGENDA FECHAINICIO) y el ícono que se usa (logo.gif).

6.5.1 Standards acompañantes

La ventaja de XML es que es un standard será más fácil encontrar bibliografía, servicios y software para trabajar con documentos XML.

W3c, entre otros, ha desarrollado otros standards que acompañan a XML, tal es el caso de XML namespaces, hojas de estilo, DOM, SAX, XLink, Xpointer.

XML namespaces

El namespace asocia un elemento con su dueño.

Previene conflictos de nombres y es una forma de permitir la reutilización de estructuras standards.

Un espacio de nombres se puede ver como una forma de identificar elementos XML. Además, permite la reusabilidad de los elementos ya declarados o utilizados.

La extensibilidad de XML puede ser una desventaja en un ambiente distribuido.

En un ambiente colaborativo, una lista de recursos definida por un usuario puede ser modificada por otro, agregándose por ejemplo, un elemento adicional.

Por eso es necesario administrar el espacio de nombres.

Supongamos una lista de libros. Uno podría considerar una categoría sobre entretenimiento y otra sobre nivel educativo.

```
<nombre>Harry Potter</nombre>
        <link href="http://www.harrypotter.com"/>
        <entret:categoria>5</entret:categoria>
        <educ:categoria>5</educ:categoria>
```

El prefijo indica el tipo de categoría (entret, educ), SI varias personas trabajan en la lista, puede haber distintas interpretaciones.

Los prefijos se pueden declarar.

```
<?xml version="1.0" enconding ="ISO-8859-1"?>
<referencias
xmlns:entret="http://www.entretenimiento.com/categoriaLibros/1.0"
        xmlns:educ=http://www.normaseducativas.com/categoriaLibros/1.0
                <nombre>Harry Potter</nombre>
                        <link href="http://www.harrypotter.com"/>
                        <entret:categoria>5</entret:categoria>
                        <educ:categoria>5</educ:categoria>
                                .
                                .
                                .
        </referencias>
```

Hojas de estilo

XML utiliza dos lenguajes de hojas de estilo: XSL y CSS. Las hojas de estilo especifican como se muestran los documentos XML.

6.6 Software de XML

Explorador XML

Se usa para ver o imprimir documentos XML. A partir de la versión 4.0 Microsoft Internet Explorer soporta XML y se ha ido mejorando.

Mozilla, la versión Open Source, lo soporta.

Se han desarrollado otros exploradores XML como InDelv XML Browser.

Editores XML

Hay un gran número de editores disponibles. Algunas son versiones reducidas de editores SGML (es el caso de Adobe Framemaker) y hay nuevos (XMLPro), Xmetal, etc.

Notepad XML de Microsoft es un editor sencillo y es gratuito.

Analizadores XML

Los analizadores ayudan a los programadores a controlar la sintaxis en XML.

Entre otros está disponible el analizador XML for Java. Muchas aplicaciones hoy incluyen un analizador XML como Oracle 8i.

Procesadores XSL

En algunos casos para no forzar al usuario a ir a un explorador compatible con XML, se usa HTML e internamente se transforma a XML

XSL permite crear código HTML conservando de manera interna XML con sus ventajas. Ese código puede trabajar con los exploradores que no soportan XML. Para usar XSL se debe contar con un procesador XSL.

6.7 DTD (Data Type Definition)

DTD es un mecanismo para describir cada objeto que puede aparecer en el documento. Es una descripción formal del documento.

En la DTD se declaran los atributos.

El documento se relaciona con una DTD a través de la declaración de tipo de documento, que tiene la forma

```
<?xml version="1.0" enconding ="ISO-8859-1"?>
<!DOCTYPE  agenda SYSTEM "agenda.dtd">
<entrada>
        <nombre>Perez Jacinto</nombre>
        <direccion>
                <calle>70 nro 700</calle>
                <telefono>4220332</telefono>
        </direccion>
        <correo href=mailto:jacintop@ciudad.com.ar/>
</entrada>
<entrada>
        <nombre>Paez Pedro</nombre>
        <direccion>
                <calle>132 nro 1458</calle>
                <telefono>4512896</telefono>
        </direccion>
        <correo href=mailto:pedropaez@ciudad.com.ar/>
</entrada>
```

En la DTD hay subconjuntos internos y externos.

Los internos están insertados dentro del documento y los externos apuntan a una entidad externa.

El subconjunto interno se incluye entre corchetes en la declaración de tipo de documento.

```
<!DOCTYPE  direccion [
<!ELEMENT direccion (calle,telefono)>
<!ATTLIST direccion preferente (true |  false) "false">
<!ELEMENT calle  (#PCDATA)>
<!ELEMENT telefono  (#PCDATA)>
]>
```

El subconjunto externo es referenciado desde la declaración. Veamos dos ejemplos:

<!DOCTYPE agenda SYSTEM "http://www.todoxml/dtd/agenda.dtd">

<!DOCTYPE agenda PUBLIC ".//softline//agenda//SP"http://... agenda.dtd">

Hay identificadores de sistema y públicos.

El identificador de sistema es una URI (Universal Resource Identifier) que apunta a la DTD. Podemos ver una URI como un conjunto de URL's. El procesador XML debe bajar el documento desde una URI.

El identificador público apunta a una DTD grabada con el Standard ISO según norma 9070.

El identificador de sistema debe ir después del público. El público se usa para manejar DTD´s locales.

Existe un archivo de catálogo que tiene los identificadores públicos y los URI´s asociados.

El procesador XML accede a este catálogo y usa los URI's para acceder a los documentos. Si los URI´s son locales, el procesador accederá localmente.

Veamos un ejemplo del catálogo.

```
<XMLCatalog>
    <Base Href=http://softline.com/dtd//>
    <Map PublicId=".// softline//agenda//SP"
        Href="agenda.dtd"/>
    <Map PublicId=".// softline//estructuraCarta//SP"
        Href="estructuraCarta.dtd"/>
</XMLCatalog>
```

Un documento XML es independiente cuando todas las entidades están en subconjuntos internos de la DTD. Es decir que el procesador XML no necesita descargar entidades externas.

Un documento XML está bien formado cuando siguen la sintaxis XML.

No tienen DTD, así que el procesador XML no puede verificar la estructura. En este caso, el procesador verifica solamente que sigan normas de sintaxis.

Un documento XML es válido cuando se cumplen las normas de sintaxis XML, logrando una estructura específica descripta por una DTD.

Los documentos válidos tienen DTD: el procesador XML verifica sintaxis y que el documento se ajusta a la DTD.

6.8 XML y las aplicaciones

6.8.1 Los analizadores XML

El analizador XML es una componente de software que se ubica entre la aplicación y los archivos XML. De esta forma, el desarrollador se abstrae de la sintaxis XML. Es una herramienta de muy bajo nivel.

Internet Explorer 4.0 incluyó dos analizadores XML.

Hay analizadores de validación y otros que no validan. Ambos controlan la sintaxis, pero sólo los de validación "validan" el documento contra una DTD.

El analizador lee el documento XML y genera un árbol en memoria. La aplicación trabaja sobre ese árbol.

6.8.2 La interfaz entre el analizador y la aplicación. DOM y SAX

La interfaz entre el analizador y la aplicación puede ser basada en objetos o en eventos.

En el primer caso, el analizador crea en memoria un árbol de objetos con los elementos del documento XML.

En el segundo caso, el analizador lee el archivo y genera eventos a medida que halla elementos, atributos, etc., en el archivo. La aplicación tiene que "escuchar" los eventos y determinar qué se está describiendo.

La interfaz basada en objetos son ideales para aplicaciones que tratan con documentos XML (exploradores, editores, etc.).

Las interfases basadas en eventos es para las aplicaciones que tienen su propia estructura de datos, no en XML. Las aplicaciones que importan XML a bases de datos, por ejemplo, sólo asignan una estructura XML a una estructura interna.

Los eventos indican a la aplicación que algo ocurrió por si la aplicación desea reaccionar a ese evento.

En las aplicaciones hay controladores de eventos que son funciones que los procesan.

En el analizador existen eventos para etiquetas de apertura o cierre de elementos, contenido, entidades, errores de análisis.

Se debe tener en cuenta que la interfaz basada en objetos genera un árbol en memoria lo que requiere la capacidad necesaria de este recurso, algo que no requiere en demasía la interfaz de eventos.

Nuevamente, se destaca la necesidad de fijar standares. El objetivo es poder desarrollar software sin importar el tipo de analizador a usar.

Estas dos interfaces (objetos y eventos) se adaptan a 2 standards diferentes:

- DOM (Document Object Model), que es el standard para la interfaz orientada a objetos;
- SAX (Simple API for XML), que es el standard para la opción por eventos.

SAX es un API simple que fue desarrollado por la lista de correo XML-DEV y editada por David Meggison (www.meggison.com/SAX).

Muchos analizadores dan soporte a ambas interfases, como XML para Java de IBM y ProjectX de SUN.

En la interfaz orientada a objetos, con DOM, la aplicación debe esperar hasta que el documento se lea completamente. En la interfaz por eventos, la aplicación puede comenzar a procesar el documento a medida que el analizador lo va recorriendo.

6.8.3 XML en la comunicación entre aplicaciones

Cuando XML se usa para comunicar aplicaciones debemos tener en cuenta que esa aplicación pueden estar en distintas computadoras conectadas a través de una red.

En la aplicación Cliente/servidor los métodos más populares para la comunicación son el uso de CORBA, DCOM, RPC por un lado, y formatos de intercambio usando HTML o XML.

Cuando varias organizaciones comparten aplicaciones conviene tratar con un formato común.

XML es una alternativa interesante por su extensibilidad, estructura, escalabilidad y versatilidad.

6.8.3.1 XML y JAVA

Java tiene características que lo hacen atractivo para el desarrollo con XML:

- Hay herramientas de XML disponibles en Java
- Java es altamente portable
- Es un lenguaje de tipos y compilado, por lo que el compilador capta muchos errores.

6.9 Web Services usando XML (XML Web Services)

XML Web Services fue diseñado para que sistemas heterogéneos puedan interactuar, comunicándose e intercambiando información.

En el marco que nos ocupa, la heterogeneidad está orientada a sistema operativo, lenguaje o arquitectura.

Para esta interacción, XML Web Service usan HTTP, XML y SOAP, es decir, standards en el mundo Internet, lo cual es coherente con el objetivo de interoperabilidad. De esta manera, el servicio es accesible desde cualquier cliente.

Estos servicios pueden verse como caja negras (black boxes). La aplicación que los invoca se abstrae de la funcionalidad: la aplicación necesita saber qué hacen, no cómo lo hacen.

Para poder accederlos deben saber cómo invocarlos y qué retornará de la ejecución (que, obviamente, será remota).

En otras palabras: los desarrolladores crean métodos y los exponen para que otros los usen.

Veamos tres protocolos que rigen este intercambio entre el cliente y el XML Web Service.

- HTTP GET: el XML Web Service es invocado usando un requerimiento GET sobre HTTP. El input se pone en un string (query string) y el resultado vuelve como un documento XML.
- HTTP POST: el XML Web Service es invocado usando un requerimiento POST sobre HTTP. Los valores de input se ponen en el cuerpo del HTTP POST y el resultado vuelve como un documento XML.
- SOAP: el XML Web Service es invocado usando un mensaje SOAP sobre HTTP. Los valores de input se ponen en el cuerpo del mensaje SOAP y el resultado vuelve como un documento XML.

El uso de uno u otro está limitado por el tipo de datos: para tipos simples, como strings o enteros, los argumentos de input pueden hacerse a través de un query string, form post o usando un mensaje SOAP. Pero para tipos complejos, como imágenes, objetos o data sets, se serializan los argumentos a XML y son incluidos en el cuerpo de un mensaje SOAP.

6.9.1 Cómo se expone el servicio al cliente?

El usuario que desarrolló el servicio, necesita exponerlo, para que aquellos usuarios interesados puedan accederlos.

Cómo publicita el desarrollador su servicio? Si bien siempre se puede hacer llegar al desarrollador interesado directamente la URL del lugar donde está el servicio, existe una forma más práctica: registrar el servicio en un site, que funciona como una guía de los servicios en la Red, una gran base de datos de servicios.

Esta es la función de la base de datos UDDI (Universal description, Discovery and Integration)[3]. Más adelante se detallan las características de UDDI (ver **¡Error! No se encuentra el origen de la referencia.**).

[3] ver http://www.uddi.org o también, http://uddi.microsoft.com

UDDI es una organización para el registro, esencialmente, de XML Web Service.

De esta manera, potenciales clientes que precisan determinado servicio, lo pueden buscar en UDDI, y si existe, saben cómo accederlo y dónde.

6.9.2 Cómo invoca el cliente el servicio?

Cuando se crea el XML Web Service, se debe considerar que el cliente que lo necesite lo invocará.

Para ello, debe informarse cómo hacerlo, cuáles son los métodos que lo forman, cuáles serán los argumentos de llamada y qué devolverá.

Para eso se agrega al XML Web Service, un documento con toda esta descripción. Este documento está construido en WSDL (ver 8).

En el registro UDDI correspondiente al servicio, se hace referencia a la URL donde está este documento WDSL. Ese documento WSDL tendrá toda la información necesaria para acceder al XML Web Service.

El documento WSDL describe como los argumentos de invocación al servicio incluyendo tipo de dato y nombre, para la invocación HTTP GET, HTTP POST y SOAP.

Si bien el proveedor del servicio puede hacer cambios en la lógica dentro de los métodos, mientras no modifique la invocación, el documento WSDL no cambiará.

Se podría, por ejemplo, cambiar las bases de datos a las cuales se conecta el servicio, y el cliente lo invoca no se enteraría.

6.9.3 El consumidor y el retorno del XML Web Service

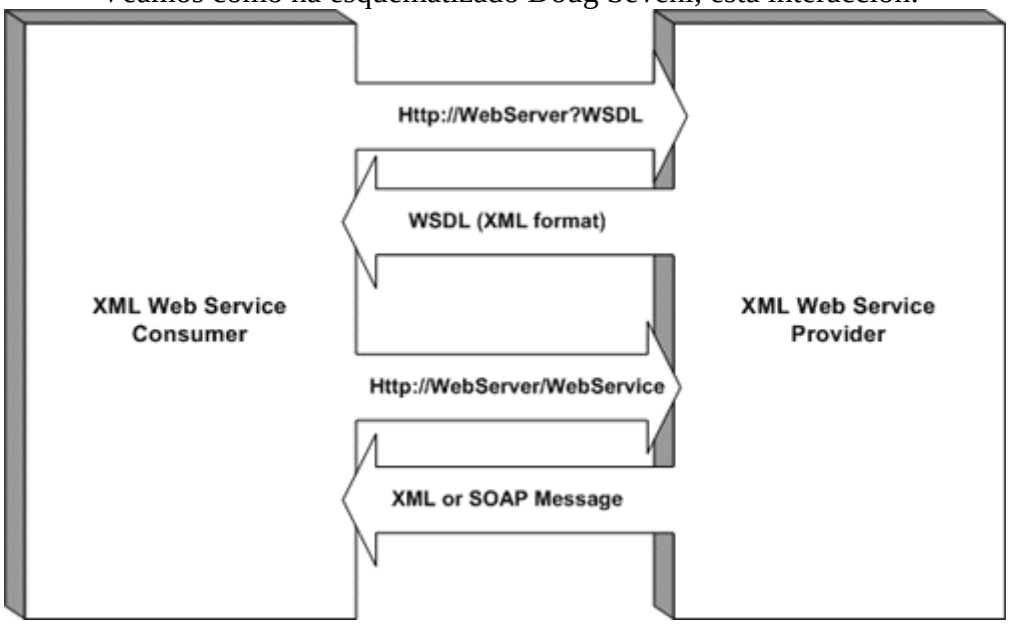
Evidentemente, XML Web Service se basa en un modelo donde existe un proveedor (la aplicación que expone el XML Web Service) y un consumidor (la aplicación que usa el XML Web Service).

Supongamos que un usuario de una aplicación usa el XML Web service. Cuando se dice que el servicio retorna un documento XML, el usuario no debería necesariamente, ver ese documento. Es más: lo deseable, para obtener la abstracción deseada es que no lo vea.

La aplicación que recibe la respuesta del XML Web Service, debe estar preparada para “consumir” ese XML, y transferir la información que está en él a la aplicación de usuario, de una manera amigable.

O sea: una aplicación Web actúa como consumidor, invoca al XML Web Service, recibe el documento XML que retorna, y lo muestra de manera apropiada al usuario.

Veamos como ha esquematizado Doug Seveni, esta interacción:



El ciclo de uso de un XML Web Service sería entonces:

1. El usuario que necesita el servcio, lo busca en el registro UDDI (o accederlo directamente si se tiene la URL donde está).
2. Si está, requiere el documento WSDL.
3. El proveedor envía el documento WSDL en formato XML.
4. El proveedor hace un requerimiento del XML WEB SERVICE que figura en el WDSL, pasando los argumentos según la forma enunciada en el WSDL.
5. El proveedor procesa el requerimiento y retorna el resultado en un documento XML o en un mensaje SOAP.

6.10 Notas

En este capítulo, fueron muy útiles artículos específicos de XML, como aquéllos que lo trataban lateralmente, cuando trataban implementaciones específicas.

El libro [MAR001] fue excelente para la introducción al tema, considerando la cantidad de ejemplos que detalla.

También recomiendo [MIC002], [SEV001], [SEV002], [XML000].

[PLA002] siempre hizo su colaboración. Si bien es un libro sobre .NET, su capítulo sobre Web Services (y los protocolos participantes) está muy bien desarrollado.

Para visitar:

- www.w3.org/xml
- http://www.xml.com

Capítulo VII

SOAP

7 SOAP

Para introducirnos en el concepto de SOAP, pensemos en esencia que es lo que hacemos con un Web Service.

El Web Service es invocado, a través de parámetros de entrada y retorna un resultado, como parámetro de salida.

Bueno, pensemos en esa comunicación como si fuera un sobre con información que es intercambiada.

SOAP define, por un lado, ese "sobre" que envuelve la información. Especifica una serie de reglas para representar la información que viaja.

Pero tiene, además, una función muy importante: definir una convención la invocación a procedimientos remotos y su respuesta.

Resumiendo: SOAP define el sobre, las reglas y la convención de invocación remota, esta última, opcional.

7.1 Qué es SOAP?

SOAP fue definido por W3C como "**a lightweight protocol for information exchange in a decentralized, distributed environment**".

El éxito de SOAP es que está basado en texto (usando XML) lo que lo hace independiente de distintas plataformas (y vendedores).

De esta forma al invocar un método, no es necesario saber en qué lenguaje está desarrollado, si es remoto o no, etc.

SOAP no incluye, por sí mismo:

- Garbage collection distribuido
- Manejo de objetos por referencia (requiere garbage collection)
- Activación (requiere manejo de objeto por referencia)

SOAP es lo que se llama "lightweight protocol". Tiene capacidades fundamentales como:

- Poder enviar y recibir paquetes HTTP o de otros protocolos
- Poder procesar XML

Lo interesante de estas tecnologías que utilizan XML y SOAP es que pueden integrarse en aplicaciones ya existentes sin demasiada complejidad.

Productos que usan SOAP, pueden usar servidores HTTP existentes y procesadores XML que tenga el cliente.

Si en cambio, se utiliza CORBA, DCOM o Java RMI se debe instalar el ambiente apropiado, configurar el sistema para adaptarlo a la nueva infraestructura distribuida. También puede exigir reconfigurar firewalls. Por eso a esta tecnología se lo

asocia con “heavyweights systems”.

SOAP permite serializar la invocación de métodos remotos, transformando la llamada al método en la forma adecuada para su transmisión sobre la red, sobre una capa de transporte que lleve esos datos inherentes al método entre los sistemas remotos.

7.2 SOAP y su relación con otros protocolos

Si bien SOAP puede usar otros protocolos de transporte, el uso de HTTP permite ampliar el rango de los puntos de acceso, al pasar fácilmente a través de firewalls (otros protocolos tienen los puertos deshabilitados por seguridad).

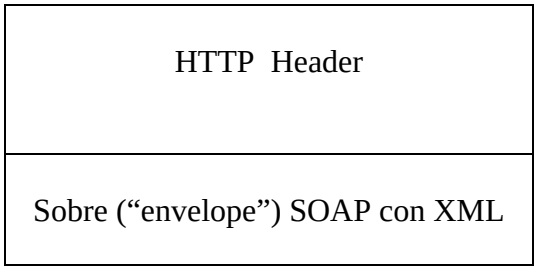
SOAP puede integrar sistemas que hasta ahora no podían comunicarse. Y esto es gracias a XML. Todos los mensajes SOAP son codificados usando XML.

SOAP define dos namespaces:

- el del envelope, http://schemas.xmlsoap.org/soap/envelope
- para la serialización, http://schemas.xmlsoap.org/soap/encoding

SOAP fue diseñado para transformar los parámetros de la invocación al método desde su forma nativa (binaria) a XML, donde en el puerto de destino el procesador SOAP toma esa información XML y lo lleva a un estado propio del destino para que pueda procesarse.

Paquete HTTP



En el header HTTP se indica el destino, en qué está el contenido del paquete (texto basado en XML).

Otra ventaja de SOAP, es que permite serializar la invocación de métodos remotos usando un protocolo de serialización disponible y listo para ser usado, sobre la capa de transporte que requiera la computación ubicua.

SOAP permite implementar un mecanismo de serialización y el transporte necesario de los datos de intercambio en la invocación remota de métodos.

SOAP no define por sí mismo ninguna semántica de aplicación (por ejemplo, un modelo de programación) ni cubre aspectos de activación/desactivación, garbage collection, o seguridad. Es decir: SOAP no es una arquitectura entera distribuida.

7.3 El modelo de intercambio de mensajes de SOAP

Fundamentalmente los mensajes SOAP son de transmisión de una sola vía (one way). Pero pueden combinarse para ofrecer un modelo request/response.

Por ejemplo, se puede implementar que los mensajes de respuesta SOAP sean librados como respuestas HTTP.

Ejemplo de Mensaje SOAP, dentro de una solicitud HTTP

```
POST /StockQuote HTTP/1.1
Host: www.productosHome.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
      <m:PidoUltimoPrecio xmlns:m="Some-URI">
         <symbol>DIS</symbol>
      </m: PidoUltimoPrecio >
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Ejemplo de Mensaje SOAP, dentro de una respuesta HTTP

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
      <m: PidoUltimoPrecio Response xmlns:m="Some-URI">
         <Price>34.5</Price>
      </m: PidoUltimoPrecio Response>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Ejemplo: mensaje SOAP de invocación para acceder a una cuenta en QuickBank

```
<soap:Envelope xmlns:soap=
```

```
"http://schemas.xmlsoap.org/soap/envelope/"> <soap:Body>
<GetBalance
xmlns="http://www.qwickbank.com/bank"> <Account>729-1269-
4785</Account>
</GetBalance> </soap:Body>
</soap:Envelope>
```

Ejemplo: mensaje SOAP de respuesta que retorna el saldo de la cuenta (balance) en QuickBank

```
 <soap:Envelope xmlns:soap= "http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<GetBalanceResponse xmlns="http:llwww.qwickbank.com/bank">
<Balance>3,822.55</Balance> </GetBalanceResponse>
</soap:Body>
</soap:Envelope>
```

7.3.1 Qué pasa cuando llega un mensaje SOAP?

Los mensajes que llegan, no necesariamente son para ese nodo. El mensaje puede seguir un “message path”, y tocar nodos intermedios.

Cuando llega un mensaje SOAP, la aplicación que lo recibe debe realizar los siguientes pasos:

- Identificar todas las partes del mensaje
- Verificar que todas las partes obligatorias son soportadas por la aplicación y procesarla. Si no es así, descartar el mensaje.
- Si no es el lugar de destino, remover las partes que correspondan antes de forwardear el mensaje.

El procesador SOAP del mensaje debe contar con la información necesaria para saber qué modelo de intercambio se usa, si se usan mecanismos de RPC, cómo se representan los datos, etc.

7.4 Estructura SOAP

SOAP como protocolo, consiste de tres partes:

- el **envelope**, (que hay en el mensaje, a quién va dirigido, si es opcional o mandatorio)
- **reglas de codificación**, que definen el mecanismo de serialización para el intercambio de tipos de datos definidos por la aplicación
- **una representación RPC** que define la convención sobre invocación remota de métodos y su respuesta

En un mensaje SOAP se distingue:

- un envelope, obligatorio
- un header, opcional
- un body, obligatorio

El “envelope”

El “envelope” SOAP envuelve la información a transmitir, sin especificar una dirección. Está formado por grupos de notas e información separados por tabs.

Es el elemento de mayor nivel.

Puede contener declaraciones de namespaces y atributos. Si están atributos deben estar calificados por namespaces. Puede contener subelementos.

El “header”

El header es opcional. Tiene información para manejo y control, como números de cuenta o identificación de cliente. Puede tener también información sobre el algoritmo usado para encriptar el body o la clave pública necesaria para leer el texto encriptado o un identificador de transacción o algún mecanismo para prevención de deadlock.

Si está presente, debe ser el primer elemento hijo del elemento envelope.

Puede usarse el atributo encodingStyle, y también los atributos mustUnderstand y SOAP actor.

Veamos un ejemplo de un header que tiene un identificador de elemento Transaction, con un valor mustUnderstand de 1 y 5.

```
<SOAP-ENV:Header>
  <t:Transaction
    xmlns:t="some-URI" SOAP-ENV:mustUnderstand="1">
      5
  </t:Transaction>
</SOAP-ENV:Header>
```

El “body”

En el Body se registra la informacióna transportar.

Debe ser un elemento hijo del envelope, y debe seguir al header si este está presente.

Provee un mecanismo para intercambiar información con el receptor último del mensaje.

Normalmente incluye el marshalling para llamadas RPC y reportes de error.

Las reglas de codificación del body incluyen:

- La entrada debe ser identificada por un nombre de elemento totalmente calificado, por un namespace URI y nombre local. Los elementos hijos pueden ser calificados por namespace.
- Puede usarse el atributo encodingStyle para indicar la serialización.

7.4.1 El atributo encodingStyle

El atributo encodingStyle puede ser usado cuando se quieren indicar pautas de serialización (ver 2.2.4) dentro de un mensaje SOAP.

Este atributo puede aparecer en cualquier elemento, y tiene alcance sobre ese elemento y sus hijos que no tengan específicamente el atributo.
No hay un encoding default definido para mensajes SOAP.

En el atributo se indican una o más URI's que identifican las reglas de serialización-deserialización.

Ejemplo:

"http://schemas.xmlsoap.org/soap/encoding/"
"http://my.host/encoding/restricted http://my.host/encoding/"
""

7.4.2 El atributo actor

Un mensaje SOAP puede ir desde el origen al destino, pasando por nodos intermedios, cubriendo lo que se llama un *message path*.

Un intermediario SOAP es una aplicación que es capaz de recibir y forwardear mensajes SOAP. Los intermediarios y el destino se identifican a través de URIs.

Por lo tanto, hay partes del mensaje que son para los intermediarios y, por lo tanto no deben forwardearse una vez que se llega al intermediario al que iban dirigidos.
Un receptor del header no debe forwardear ese elemento del header a la próxima aplicación del mensaje..
El valor del atributo actor es una URI, que indica que el elemento header es para la primer aplicación SOAP que procese el mensaje.
Cuando se omite el atributo actor, quiere decir que el receptor es el destino último del mensaje SOAP.

7.4.3 El atributo mustUnderstand

Este atributo se usa para indicar cuando un header es obligatorio u opcional para el receptor del proceso.
Ese receptor lo define el atributo actor. El atributo mustUnderstand es 1 ó 0.

7.5 SOAP y RPC

Usando XML, SOAP puede intercambiar RPC.

Lo importante, para lograr la interoperabilidad es obtener una representación uniforme de los llamados y respuestas RPC.

Ya vimos el atributo encodingStyle. Este atributo que se usa para definir la serialización es importante en el momento de invocar métodos y las respuestas.

Para invocar un método, es necesario contar con la URI del objeto target, el nombre del método y los parámetros.

Las invocaciones a los métodos y las respuestas se llevan a cabo dentro del body del mensaje SOAP.

La invocación al método y su respuesta, se definen en una estructura. Estas estructuras involucran los parámetros o valores de retorno.

Siempre puede utilizarse el atributo encodingStyle para especificar la serialización.

7.6 Notas

Para este artículo fue esclarecedor [BOX000] que, como Note de W3C, es una garantía.

El libro [SCR002], es un libro sumamente interesante y completo sobre SOAP.

También contribuyeron las lecturas de otros artículos, no específicos de SOAP pero que lo trataban en el contexto de WS. Reitero especialmente la consulta a [PLA002], y también a [CHA002].

Capítulo VIII

8 WDSL (Web Service Description Language)

8.1 Generalidades sobre WDSL

Cuando un proceso cliente y un proceso servidor quieren interactuar, los que desarrollan el software deben acordar cuál es el nombre de la operación que invocará la aplicación cliente, como la invocará (parámetros de entrada, tipos) y qué responderá la aplicación servidor.

En COM y CORBA, estas interfases y las operaciones que ellas describen están construidas en IDL (Interface Definition Language).

En Web Service se necesita algo similar. Y para esta tecnología se usa **WDSL**.

Para explicar porqué se prefiere usar WSDL en lugar de COM, por ejemplo, valga aclarar que la IDL que se usa en COM tiene una sintaxis derivada de C. En cambio, WSDL se define usando XML. Y ahí está la razón: el uso de un protocolo standard en Internet que facilita la interoperabilidad.

WSDL no es simple. Para hacer una interfase WSDL se debe tener muy buen conocimiento de XML. La ventaja es que "entendido" por prácticamente cualquier plataforma.

8.2 WSDL y SOAP

WSDL extiende los beneficios de SOAP, al proveer una forma de que los usuarios y proveedores de Web Services se puedan comunicar y trabajar más allá de plataformas y lenguajes.

SOAP no necesita un formato que lo describa. En cambio WSDL es imprescindible como lenguaje de descripción, pues si no no sabremos cómo invocar el Web Service, su nombre, sus argumentos.

Si bien esta comunicación podría establecerse estáticamente, e involucrar al usuario (averiguando y luego invocando) se pierde la característica de dinamismo del modelo de Web Services.

WSDL puede hacer bindings a otros protocolos, no sólo a SOAP. Pero para obtener la interoperabilidad deseada, se recomienda SOAP sobre HTTP.

8.3 Estructura de una Interfase WSDL

Una interfase WSDL es un documento XML. Veamos la estructura en el diagrama de la página siguiente.

En este diagrama se muestran las relaciones entre las distintas secciones de un documento WSDL.

El documento WSDL tiene dos grupos de secciones: las llamadas **abstractas** y las **concretas**.

Las abstractas, están en la parte superior del documento. Definen el mensaje SOAP de una manera independiente del lenguaje y la plataforma (no se refiere a máquinas específicas o lenguajes).

En el grupo de secciones concretas hay más especificaciones sobre sites o serialización.

En el siguiente cuadro resumimos la pertenencia de las secciones.

Definiciones abstractas	**Types**	*Contiene definiciones tipos independientes de máquinas y lenguajes*
	Messages	*Contiene definiciones sobre parámetros de funciones o descripciones de documentos.*
	Port types	*Se refiere a definiciones de Messages, para descripción de nombres, parámetros de input y output.*
Definiciones concretas	**Bindings**	*Especifica los bindings según lo especificado en Port Types*
	Services	*Especifica la dirección de cada binding.*

En el diagrama siguiente, las flechas indican la relación entre secciones del documento.

Un punto y una flecha, representa la relación “se refiere a” o “usa”.

El conector con flecha doble representa la relación “modificado”.

El conector con flecha en 3D, representa la relación “contiene”.

Por ejemplo: la sección Message usa definiciones que están en la sección Types.

A su vez, PortType contiene elementos.

Los elementos de operaciónen PortType son modificados o descriptos posteriormente por elementos de Bindings section.

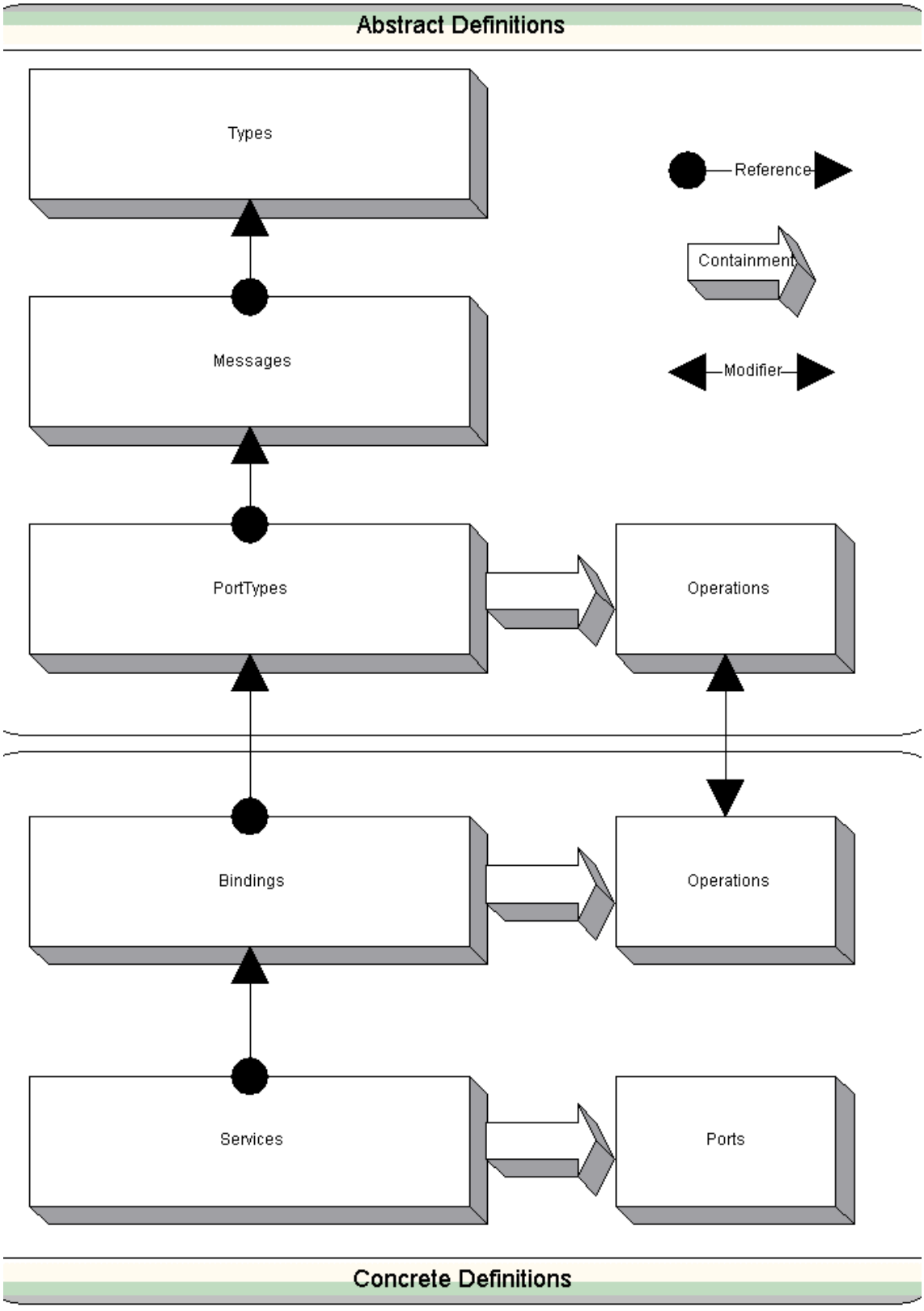


Diagrama de un documento WSDL

A continuación veremos como sería una interfase WSDL de una operación (GetDescripcion), que, a través de un código de producto, devuelve su descripción.

```
<definitions name="InfoProducto">
        <types>
              <element name="SolicitoDescripcion">
                    <!acá se define el tipo del códigode producto, por
                    ejemplo
              </element>
               <element name="ResultadoDescripcion">
              <! acá se define el tipo del producto, por ejemplo
              </element>
        </types>
        <message name="GetCodigoInput">
              <part name="body"  element="SolicitoDescripcion"/>
        </message>
        <message name="GetDescripcionOutput">
              <part name="body"  element="ResultadoDescripcion"/>
        </message>
        <portType name="InfoProductoPortType">
               <operation name="GetDescripcion">
              <input message="GetCodigoInput"/>
              <output message="GetDescripcionOutput"/>
              </operation>
        </portType>
        <binding name="InfoProductoSoapBinding"
              type="InfoProductoPortType">
               <soap:binding transport=
              "http://schemas.xmlsoap.org/soap/http"/>
              <operation name="GetDescripcion">
              </operation>
        </binding>
        <service name="InfoProductoService">
                    <port name="InfoProductoPort"
                    binding="InfoProductoSoapBinding">
                     <soap:address  location=
                    "http://www.grandesalmacenes.com/productos.asmx"/>
                    </port>
        </service>
</definitions>
```

Nombre de la operación: GetDescripcion.

Dentro de la definición, tenemos distintos elementos: types, message, portType, Binding y Service.

El contenido de una interfase WSDL tiene dos partes.

En una parte se definen los elementos de manera abstracta a través types, message y PortType.

Luego, a través de binding y service, los relaciona con un determinado protocolo.

WSDL no asume ningún protocolo en particular para la invocación de esa interfase, por lo tanto, las operaciones deben poder ser accedidas por más de un protocolo.

8.3.1 El elemento Type

Sirve para definir tipos básicos que se necesitarán posteriormente para la definición.

En nuestro ejemplo, lo tipos son SolicitoDescripcion y ResultadoDescripcion.

Ellos son parámetros de entrada y salida respectivamente.

8.3.2 El elemento Message

Estos elementos contienen información sobre mensajes que se envían o reciben desde el web service.

Cada operación tiene un mensaje para la entrada y otro para la salida.

En nuestro ejemplo, ellos son GetCodigoInput y GetDescripcionOutput.

Estos usan los elementos type ya definidos.

8.3.3 El elemento PortType

Cada elemeno PortType describe una operación del Web Service.

Cada operación define su mensaje deinput y de output, ya definidos en el elemento message.

En el ejemplo, la operación GetDescripcion, hace referencia a los mensajes GetCodigoInput y GetDescripcionOutput.

8.3.4 El Binding y el Service: la asociación al protocolo

En nuestro ejemplo, asociaremos la operación con el protocolo SOAP.

El elemento Binding apunta por un lado a la operación (portType) y describe el protocolo y formato que se usará en ella.

En nuestro ejemplo, InfoProductoSoapBinding, asocia GetDescripcion con el protocolo SOAP.

El elemento Service asocia InfoProductoSoapBinding, con la URL donde está realmente el servicio a través de un Port. Los Ports son métodos para acceder al servicio.

En nuestro caso, el Port es InfoProductoPort.

```
<soap:Envelope
    xmlns:soap=
        "http://schemas.xmlsoap.org/soap/envelope/">
```

```
    <soap:Body>
                <GetBalance
                xmlns="http://www.qwickbank.com/bank">
                 <Codigo>729-1269-4785</Codigo>
                 </GetBalance>
   </soap:Body>
 </soap:Envelope>


<soap:Envelope
        xmlns:soap=
                "http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
        <GetBalanceResponse
                xmlns="http://www.qwickbank.com/bank">
                 <Balance>3,822.55</Balance>
        </GetBalanceResponse>
   </soap:Body>
</soap:Envelope>



POST /InfoProducto/Codigos.asmx HTTP/1.1
Host:  www.qwickbank.com
Content-Type:  text/xml;  charset="utf-8"
Content-Length:  ...
SOAPAction:
<soap:Envelope
        xmlns:soap=
           "http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
        <GetDescripcion>
                xmlns="http://www.qwickbank.com/bank">
                <Codigo>729-1269-4785</ Codigo>
        </GetDescripcion>
       </soap:Body>
</soap:Envelope>


HTTP/1.1      200  OK
Content-Type:  text/xml;  charset="utf-8"
Content-Length:  ...

<soap:Envelope
        xmlns:soap=
                "http://schemas.xmlsoap.org/soap/envelope/">
        <soap:Body>
                <GetBalanceResponse
                xmlns="http://www.qwickbank.com/bank">
```

```
            <Balance>3,822.55</Balance>
            </GetBalanceResponse>
        </soap:Body>
</soap:Envelope>
```

8.4 Notas

Los diagramas del capítulo son de [TAP001].

Los artículos más utilizados fueron [ROB002] y [TAP001].

Como ocurre con los capítulos sobre protocolos, intervienen otras referencias, que lo tratan lateralmente.

Nuevamente recomiendo el tratamiento de [PLA002].

Para referencia:

- http://www.w3.org/TR/wsdl

Capítulo IX

UDDI

9 UDDI

9.1 Qué es UDDI?

Supongamos que un desarrollador necesita encontrar la interfase WSDL para implementar un servicio. No sólo debe saber si ya existe, sino cómo es el intercambio de forma compatible.

UDDI (Universal Description, Discovery and Integration) es una especificación para registros distribuidos de información sobre Web Services. Es tanto una especificación para registrar como para buscar.

Para mostrar los servicios que ofrece una empresa, puede crear una registración UDDI, que es nada menos que un documento XML (cuyo formato especifica un esquema UDDI).

Cuando se crea, esta registración es almacenada en una base que se llama ***UDDI Business Registry***. Y una copia se mantiene en cada site de operadores UDDI, replicada. Estos sites están comunicados para mantener la consistencia.

El UDDI Business Regsitry es usado tanto por programas como por programadores para encontrar información acerca de servicios y para saber cómo invocarlos y trabajar con ellos. También es usado para publicar lo propio para que pueda ser consumido por otro.

El UDDI es en sí mismo un Web Service, que provee información para poder utilizar otros Web Services.

Los desarrolladores que quieran escribir software para acceder o modificar la información en la registry, deben usar una API definida por UDDI.

El esquema XML UDDI sirve para definir una estructura común para todos los documentos de registración.

La información que se registra en el UDDI tiene una orientación a la lógica de negocio. Está organizada en:

- ***white pages***: información para contactar a la compañía, tal como nombre, descripción de la compañía, identificadores tipo CUIT, etc.
- ***yellow pages***: información sobre tipo de negocio, tal como índice de productos o servicios que comercializa.
- ***green pages***: información técnica sobre los servicios que expone, tal como reglas de negocio, descripción de servicios, invocación a aplicaciones, binding de datos.

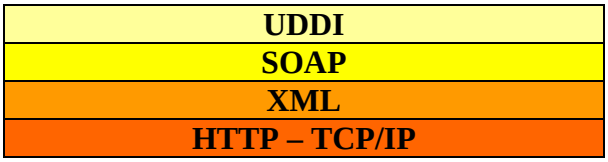
Resumiendo, veamos cuáles son las características principales de UDDI:

- Es un mecanismo para describir servicios de manera standarizada y transparente para el usuario.

- Es una forma simple de invocar un servicio.
- Permite contar con un registro accesible de los servicios.

La ventaja de usar UDDI sobre otras formas de encontrar información (por ejemplo, web crawlers accediendo a URL´s regsitradas, con riesgos para ubicar cada web site y la descripción del servicio) es que los registros y descripciones del servicio están en formato XML.

UDDI usa XML (para codificar y formatear) y SOAP (para empaquetar) para continuar con el camino de este modelo sobre tratar con estándares. Ese stack de protocolos se ordenan según el siguiente gráfico.



De esta forma se logra un stack de protocolos uniforme para encontrar servicios y describirlos.

La especificación UDDI describe una nube de Web Services y una interfase que define un framework para describir cualquier tipo de WS. Consiste de varios documentos relacionados, sumados a un esquema XML que definen un protocolo basado en SOAP para registrar y descubrir WS.

La información se agrega al UDDI business registry a través de un Web site o usando herramientas que adhieren a la especificación de la API para programadores de UDDI.

Si bien el UDDI Business Registry está centralizado “lógicamente”, físicamente sus nodos están distribuidos y replicados regularmente.

El schema XML para mensajes SOAP y descripción de la especificación de la API es el modelo base y framework para poder publicar la información sobre WS.

Este schema en UDDI define 4 tipos de información imprescindibles para establecer la relación entre las partes.

- Información del negocio (***Business Information***)
- Información del servicio (***Service Information***)
- Información para conexión (***Binding Information***)
- Información acerca de la especificación de los WS

9.2 Elementos en el documento UDDI

9.2.1 Business Information: El elemento Business Entity

El elemento Business Entity describe el nombre de la organización, forma de contacto, etc.

Este elemento tiene un atributo BusinessKey para identificar unívocamente esta entrada de registro. También se le llama UUID (universally unique identifiers) o GUID (globally unique identifiers).

La UUID está formada por la dirección Ethernet de la máquina donde se creó, tiempo de creación y otros campos.

Esta estructura sería la correspondiente a la *yellow page*.

9.2.2 Business Service: El elemento BusinessService y BindingTemplate

Ambos elementos están como estructuras dentro del BusinessEntity.

El elemento BusinessServices indica cuáles son los Web Services que provee la organización.

Es la descripción técnica del ws. Sería la green page. Agrupa una serie de WS relacionados al proceso de negocio o categorías.

El BusinessService tiene asociado una UUID que sirve para identificar la definición del servicio. También incluye el nombre del servicio.

El elemento BindingTemplate tiene información para el cliente sobre cómo conectarse y usar el servicio, incluyendo el punto de acceso (que es una URL donde se puede encontrar el servicio) y más información adicional.

9.2.3 El elemento Tmodel

Como no basta saber donde está el WS para contactarlo, es necesaria otra estructura con información adicional.

El elemento Tmodel identifica algunos aspectos del servicio: identificación de la interfase WSDL, qué protocolos usar para acceder al servicio, algunos requerimientos de seguridad.

Si, por ejemplo, el WS permite enviar un orden de compra, es necesario proveer lo necesario para hacer ese envío en forma apropiada. Por eso, en alguna bibliografía, cita al Tmodel como un elemento que contiene una lista de especificaciones.

9.3 Las APIs para el programador

Al hablar de la especificación UDDI, también estamos considerando las especificaciones que permiten al programador definir las interfases para los WS y poder acceder a informaciónen el UDDI Registry.

Esta API tiene dos partes: la Inquiry API y la Publishers API.

A su vez, Inquiry API tiene dos funcionalidades: una para desarrollar programas para buscar y ver informaciónque está en un UDDI Registry y otra parte que sirve si las invocaciones WS fallan.

Esta API tiene dos tipos de llamada, para que quien desee ubicar un WS y su especificación.

La llamada *find_xx* es la que invoca a datos registrados, permitiendo distintos criterios de búsqueda. SI ya se conoce donde se quiere acceder, se pueden obtener copias de las estructuras (BusinessEntity, BusinessService, etc.) con la llamada *get_xx*.

La Publishers API es invocada por los programadores para crear interfases que interactúan con herramientas que tratan directamente con el UDDI registry.

Consiste de 4 funciones *save_xx* y cuatro funciones *delete_xx* cada una de ellas para cada estructura (BusinessEntity, BusinessService, etc). Quien desee publicar, una vez que es autorizado puede registrar todas las businessEntity que quiera o Tmodel, o alterar ws ya publicados.

Cada una de los UDDI Business Registry replicados, tiene una lista de participantes autorizados, con información sobre cuáles servicios publicaron. Los cambios y/o eliminaciones sólo se permiten si son realizados por quien creó ese servicio.

En cada site donde esté una réplica, se permite mantener el mecanismo de autenticación que use el site. No obstante, se fijan algunas pautas de criterio común para ofrecer una protección similar.

9.4 Notas

Para este capítulo quiero destacar [UDT000] de http://www.uddi.org. También [UDE000], del mismo site.

[SAL000] lo destaca dentro de la terna de WS: SOAP + UDDI + WSDL.

Conversando con implementadores de WS, noto que el uso de la publicación no es aún una técnica muy extendida. Creo que profundizar sobre este tema termina de desarrollar ese punto que aún es poco explotado.

Publicar servicios en esta modalidad exige un buen conocimiento de WSDL, y además, asumir esta filosofía de “ofrecer” el trabajo realizado... Hay costumbres de “propiedad” difíciles de superar...

Recomiendo, nuevamente, [PLA002].

Para referencia:

http://www.uddi.org

Capítulo X

Web Services en .NET

10 Web Services en .NET

10.1 El modelo .NET

.NET es un conjunto de tecnologías de Microsoft. Es más una plataforma que un producto. Para simplificar podemos verlo también como un framework para aplicaciones.

Sobre esta tecnología pueden construirse aplicaciones y servicios que funcionan independientemente de la plataforma. Permite la comunicación entre diversos sistemas y aplicaciones, integrando distintos dispositivos cliente.

El objetivo de Microsoft con la plataforma .NET fue establecer una nueva generación de sistemas, integrando la funcionalidad de sistemas sobre Internet. Ha sido pensada como una plataforma para sistemas altamente distribuidos e interoperables, tratando de simplificar el acceso a la Web, a través de cualquier dispositivo y usando cualquier plataforma.

.NET se implementa a través de un framework, sumado a herramientas, building blocks y servicios.

Algunos autores comparan a .NET como una forma de sistema operativos distribuido a través de Internet, con un soporte optimizado a aplicaciones y servicios.

A través de las experiencia de los usuarios, se ve la necesidad de integrar datos y las preferencias del usuario en una sola aplicación. A través de XML, HTTP y SOAP logran una solución integrada.

.NET usa Web Services para realizar la comunicación entre componentes sobre la red. Por ello puede lograr un buen grado de distribución y puede acceder a módulos escritos en otras plataformas en forma transparente.

.NET está compuesto por:

- Un framework
- Visual Studio .NET
- .NET My Services
- .NET Enterprises Services

.NET provee versioning. Esto permite especificar la versión EXE ó DLL, para evitar problemas cuando el cliente invoca una versión particular.

10.2 El Framework

.NET necesita proveer una infraestructura donde los servicios de WEB puedan brindarse dentro de las pautas que estuvimos analizando. De esta manera, los programadores pueden enfocarse en la lógica de negocio, más que en la infraestructura de la programación.

El **framework** es lo que permitirá desarrollar servicios, aplicaciones e incluso servicios Web con XML.

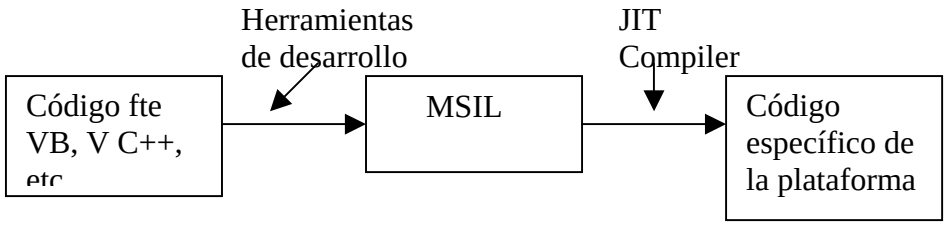
Provee garbage collection y permite acceder fácilmente a otros servicios del sistema, como Internet, Bases de datos, etc.

Algunos autores citan ASP.NET como la tercer componente del framework; otros, lo incluyen al hablar de la class library.

Entre las características de impacto en .NET, está la independencia de lenguaje y la posibilidad de interoperar con aplicaciones escritas en diferentes lenguajes.

Una componente de .NET puede estar escrita una parte en VB.NET (la versión .NET de Visual Basic) y otro en C#.

Para implementar esta capacidad multilenguaje, el código fuente es llevado a un código intermedio MSIL (Microsoft Intermediate Language), también llamado IL. . Este lenguaje es análogo al bytecode de Java.



El código MSIL debe posteriormente ser interpretado y traducido a código ejecutable, nativo de la máquina donde quiere ejecutarse. Quien realiza esta tarea es el CLR (Common language Runtime), de tarea análoga al JRE de Java.

Podemos ver al CLR como un intermediario entre lo desarrollado en código fuente y el hardware.

Lo importante de CLR es que provee automatic garbage collection (manejo de memoria automática), exception handling, herencia cross-language, debugging, etc.

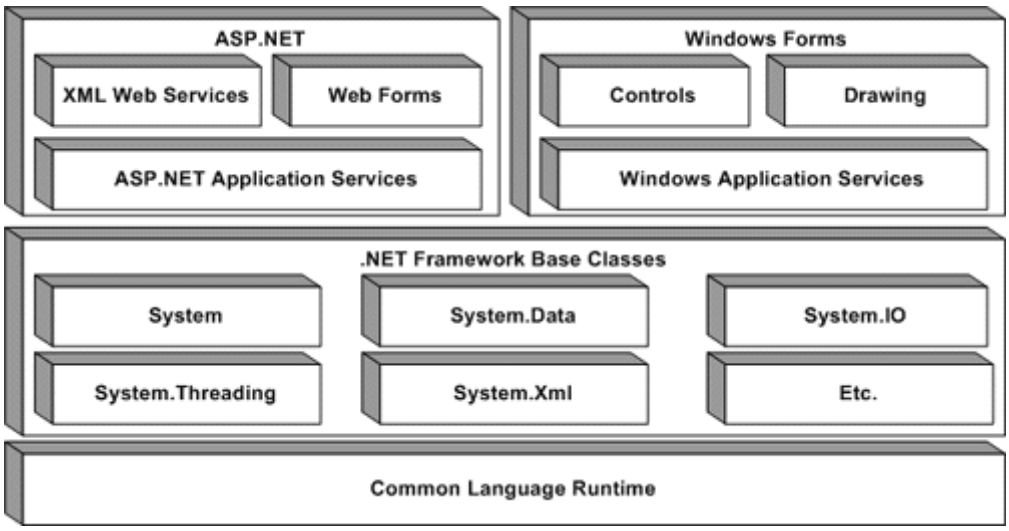
Lo que se crea en el ambiente .NET es un entorno virtual para esa ejecución del MSIL, que es llamado VES (Virtual Execution System).

El framework puede verse como un ambiente runtime. Los programas y componentes que se escriban se ejecutarán dentro de ese ambiente.

Está compuesto por :

- CRL, Common Language routine
- Una librería de clases (.NET framework class library)

Sobre el CLR, está la librería de clases y sobre ésta se “apoyan” ASP.NET y WINDOWS FORMS.



El framework organiza la funcionalidad del sistema operativo a través de un namespace System. Esto hace que los objetos, interfases y funciones del sistema operativo estén organizados jerárquicamente.

El framework ASP.NET se divide en los servicios básicos para aplicaciones que manejan todas las funciones comunes a las aplicaciones WEB, como HttpRequest y HttpResponse.

XML Web Service actúa dentro de ASP.NET, interactuando con las clases HttpRequest y HttpResponse.

Mejora a COM en cuanto al mecanismo de reuso de código. Además como no requiere seteos de registry, es muy fácil de distribuir (deployment).

.NET interactúa con COM y sus derivados. El objeto COM es visto gracias al framework como si fuera un objeto .NET

10.2.1 CLR, Common Language routine

CLR es la base para construir un conjunto de aplicaciones.
Es una evolución de cómo se trabaja en la tecnología COM, antecesora de .NET, donde los programadores estar atentos al manejo de la memoria, los threads, etc.

CLR maneja automáticamente estas cuestiones, permitiéndole al programador abstraerse de ellas. Es un runtime común, que puede ser usado por diferentes lenguajes. De allí, que se diga que .NET es *language-free*.

10.2.2 La librería de clases (.NET framework class library)

La librería de clases sirve para la implementación standard de servicios que se basan en aplicaciones CLR. Provee la funcionalidad, implementando el tratamiento de archivos, acceso a bases de datos, protocolos de red).

Cualquier clase .NET puede ser implementada en cualquier lenguaje soportado por .NET.

Diferentes lenguajes usan APIs (Applications Programming Interfaces). Las librerías de clases funcionan como API's.

Normalmente, cuando un programador usa varios lenguajes, maneja las librerías de clases que aporta cada lenguaje.

.NET ofrecer estas clases unificadas, un conjunto común de API's para todos los lenguajes que se usen, considerando que los lenguajes interactúan unos con otros.

Las tecnologías que considera la librería de clases son:

- ASP.NET
- ADO.NET
- Soporte par construir y usar Web services
- Otros

ASP.NET (Active Server Pages .NET)

Dentro de NT 4 option pack, está ISS (Internet Information Server). Parte de ISS es ASP (Active Server Page) que es un ambiente runtime para la Web.

Cuando un usuario requiere páginas Web, ISS se encarga de satisfacerlo. A través de ASP, el programador puede escribir programas que toman las páginas suministradas por ISS, mezclando HTML y código de scripting.

Cuando el cliente requiere una página ASP, ISS ubica la página y activa el proveedor ASP, quien leerá la página y de acuerdo a lo especificado, copiará los elementos HTML a una página de salida.

ASP.NET es una mejora sobre ASP y que surge de rehacer ASP para trabajar con .NET.

Visual Studio .NET

Visual Studio .NET soporta distintos lenguajes que pueden utilizar el framework: Visual Basic .NET, C++, C# (C sharp). Este último es un lenguaje diseñado para .NET framework. Es el ambiente de desarrollo de Microsoft.

Incluye:

- Project Manager
- editor de código fuente
- diseñadores UI
- Wizards, compiladores, linkers, herramientas y utilitarios
- documentación
- debuggers

Se pueden escribir aplicaciones para plataformas WIN de 32 o 64 bits, y para la .NET.

Pero lo más interesante es que ofrece un ambiente integrado para todos los lenguajes de programación.

SDK
SDK incluye todos los compiladores, herramientas, documentación.
Permite escribir aplicaciones en el framework sin usar Visual Studio.NET.

.NET My services

Es un grupo de servicios para que los usuarios almacenen y accedan a información personal como agendas, libretas de direcciones, en servidores accesibles por Internet. Incluso proveen servicios para autenticación.

De esta forma pueden obtenerse aplicaciones user centrics que antes eran imposibles de implementar, con la transparencia e interoperabilidad que permiten los nuevos modelos.

.NET Passport es una solución de Microsoft para autenticar usuarios desde distintos dispositivos y aplicaciones. Permite que los usuarios creen credenciales que les permitirán posteriormente conectarse a otros sitios que soportan .NET Passport. Ofrece personalización, para que cada usuario cree su propio perfil, para que lo “acompañe” en cada sitio que ingresa.

.NET Passport ofrece también .NET Passport Express Purchase Service (para compra on line).

.NET Alerts es un servicio que permite la entrega de mensajes a clientes.

Una alerta es un mensaje que un proveedor de servicio (el .NET Alert provider) puede mandar a clientes. Se envía de acuerdo a las preferencias del usuario y a todo tipo de dispositivos cliente (PDAs, celulares, direcciones de e-mails).

La diferencia con el mail tradicional es que la alerta no se pierde en la bandeja de entrada del cliente. .NET Alerts lleva el “registro” de las preferencias de cada cliente y las alternativas de envío al poder ser enviado a cualquier dispositivo en cualquier lugar.

Enterprise Servers

Es software para servidores como BizTallk2000, Application Ceter 2000, Commerce Service 2000, Integration Server 2000, SQL Server 2000, Exchange Server 2000, Mobile Information Server 2001, Internet Security y Acceleration Server 2000.

A través de Enterprise Servers se ofrece la infraestructura de servidor y aplicaciones.

10.3 Los objetos .NET y COM

Considerando que COM es anterior a .NET, y era utilizado para intercomunicar aplicaciones, es imprescindible que .NET interactúe con COM.

Un cliente .NET accede a un server COM a través de un *runtime callable wrapper* (RCW).

El RCW presenta al objeto COM a la CLR como si el objeto fuera .NET nativo.

Si se está usando Visual Studio, el desarrollador genera el RCW fácilmente simplemente seleccionando Add Reference en el Context Menu. Allí se mostrarán los objetos COM y al seleccionar, se generará el RCW.

Si no se está usando Visual Studio, puede generarse con .NET SDK, con una herramienta a nivel de línea de comando llamada *TlbImp.exe*.

Cuando el cliente tiene componentes COM y quiere interactuar con .NET puede recurrir a un *COM callable wrapper* (CCW), funcionalidad contenida dentro del framework .NET.

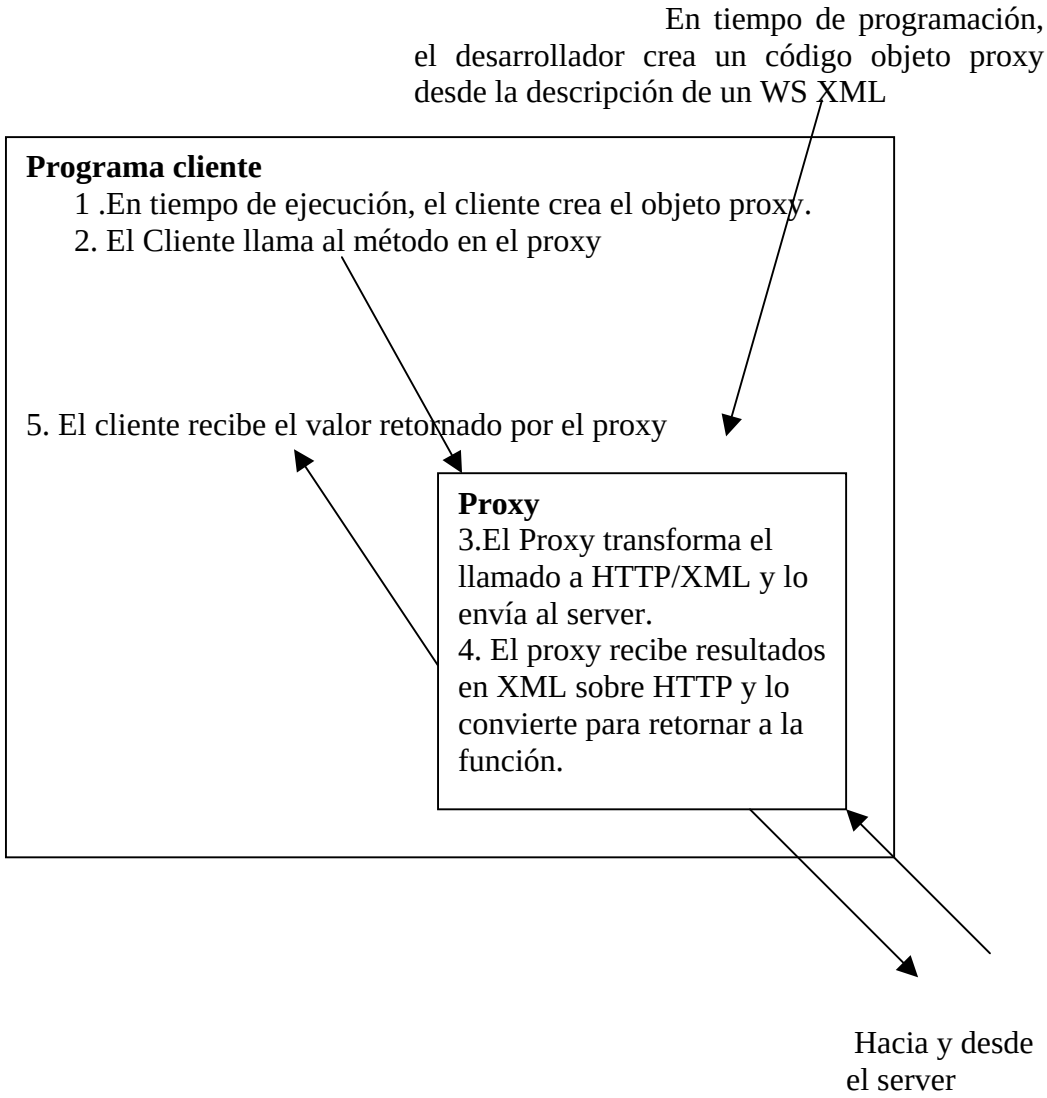
10.4 XML Web Services en Microsoft

Un servicio XML WS es una forma simple para que los objetos en un server acepten requerimientos desde clientes usando HTTP/XML.

Para escribir XML WS, se escribe un objeto .NET como si fuera a accederse localmente y se marca para indicar que puede accederse desde la Web.

Luego ASP.NET se encarga de crear la infraestructura necesaria que acepta los requerimientos HTTP que llegan y lo relaciona con el objeto.

Del lado del cliente, .NET provee clases proxy para poder acceder fácilmente a cualquier server mediante HTTP.



10.4.1 Seguridad en XML ws

Hay XML ws públicos y privados. En estos últimos casos, en la mayoría hay que implementar autenticación y autorización. Pero no debemos olvidar que los que interactúan en muchos casos son aplicaciones, no usuarios interactuando con una aplicación.

Si estoy en ambiente Windows, y usando ASP.NET ante un usuario no autorizado, se le puede presentar un Web Forms para autenticación. Pero, en WS la

interacción puede no ser con el usuario y además, los ambientes a conectarse es muy probable que no sean Windows.

XML ws necesita un mecanismo de seguridad más allá del HW o ambiente de SW que usen los clientes.

Veamos un ejemplo propuesto por David S. Platt en "Introducing Microsoft .NET". Supone una situación donde un usuario quiere acceder a un artículo en un diario, al cual se debe tener suscripción. Establece una interacción en dos fases.

En la primera, se invoca un método para autenticación que devolverá un ticket para autorizar el acceso.

El 2do método invocado, hace la solicitud de un service, pero presentando el ticket generado en la devolución de la 1ra invocación.

El método para adquisición del ticket se invoca sobre SSL (Secure Socket Layer) cambiando HTTP por HTTPs, lo que producirá la negociación de las claves.

10.5 Serialización en .NET

Dentro de la librería de clases de .NET, por ejemplo, existe un formatter que provee un método para serializar y deserializar.

Vale aclarar que, en el caso de .NET, hay 2 tipos de formatter: el binary formatter y el SOAP formater.

El binary, serializa el objeto de una forma compacta y fácil de "parsear".

El SOAP, que nos interesa en nuestro caso, serializa el objeto dentro de un mensaje SOAP.

Por ejemplo, supongamos la clase empleado

```
Class empleado
    Public nombre as String
    Public edad As Integer
End Class
```

Supongamos que el estado de un objeto empleado es Juan Rodríguez, 45 años.

Sometido este objeto al formatter, éste extraerá el estado del objeto en una forma particular.

El binary formatter generará una forma del tipo siguiente, almacenando los enteros en forma binaria:

Juan Rodriguez
45

El SOAP formatter, generará un mensaje SOAP, como este:

```
<soap:Envelope>
        <soap:Body>
                <empleado>
                        <nombre>Juan Rodriguez</nombre>
                        <edad>45</edad>
                </empleado>
        </soap:Body>
</soap:Envelope>
```

El SOAP formatter lo llevará primero a XML y luego a un mensaje SOAP.

Cuando el objeto se serializa, se pone en un stream. Una vez obtenido el stream, puede almacenarse en disco (hacerlo persistente), o usarlo. O mandarlo a través de la Red.

La clase XmlSerializer

No obstante hay que considerar que también existe otra clase para serializar: la XmlSerializer. Cuál es la diferencia con la Soap Formatter?

Si sabemos que en la comunicación cuando quiero enviar un objeto, en ambas partes está . NET, me conviene usar el SOAP formatter.

Pero, si queremos producir XML standard, usando schemas. El soporte de ASP.NET para Web Services permite usar el XmlSerializer, lo cual es útil cuando al comunicación será entre ambientes heterogéneos.

Por ejemplo, en C# se puede trabajar con objetos con miembros privados (private). XML no maneja este tipo de característica, entonces, al ser tratado por el XmlSerializer sólo incluirá los miembros públicos.

Siendo C# un lenguaje que fue creado en el ambiente .NET, se supone que usará este tipo de objetos en la comunicación con otro ambiente .NET. En ese caso, convendría usar el SOAP formatter. Pues, un objeto CLR serializado con SOAP formatter será reconstruido exactamente en el que lo recibe.

10.6 WSDL en .NET

ASP.NET puede generar una descripción examinando el metadata de un assembly.

Un assembly es una colección de uno o más EXE o DLL que contienen el código de la aplicación y recursos.

También tiene un “manifiesto” o declaración (metadata) donde se describen el código y los recursos que están “dentro” del assembly.

Esa descripción se almacena en un archivo XML con el vocabulario que usa WSDL.

WSDL se puede obtener desde ASP.NET , al requerir el archivo .ASMX con ?wsdl agregado a la URL.

10.7 SOAP en .NET

ASP.NET acepta tres formas de requerimientos que ingresan: HTTP GET, HTTP POST y SOAP.

Los dos primeros se mantienen por compatibilidad.

Cuando el cliente invoca un método en el proxy, este invoca al método *Invoke* que crea un mensaje SOAP que contiene el nombre del método y los parámetros del método a invocar XML WS y lo envía al server sobre HTTP.

Cuando vuelve el paquete SOAP, la clase base hace el parse, para tomar el valor de retorno y se lo pasa al proxy.

10.8 Creando un XML Web Service en .NET

A continuación vamos a ver como implementaríamos un web service.

Definiremos, entonces, la clase y los métodos.

Lo veremos dentro de la plataforma .NET, usando el framework ASP.NET.

Los archivos de extensión .asmx son archivos XML Web Service en ASP.NET. Lo que desarrollaremos lo consume otra aplicación, no es accedido directamente por el usuario. Se definen clases y métodos. La interfase se trata en otro lugar: la aplicación orientada al usuario.

A continuación se analizará un ejemplo simple.

```
<%@ WebService Language="c#" Class="MyService" %>

using System;
using System.Web.Services;

[WebService()]
public class MyService (1) : WebService
{
  [WebMethod()]  (2)
  public String SayHello( String YourName )
  {
    return "Hello, " + YourName;
  }
}
```

Este es el típico “Hello word!”.

Se crea una clase MyService, ver (1), y el método que se ofrece es SayHello(), ver (2).

El método toma un string que contiene el nombre del usuario y responde la palabra Hello seguido del nombre.

Este simple XML Web Service puede invocarse usando los protocolos HTTP GET, HTTP POST o SOAP.

Como el argumento de llamada es un dato simple (string), podemos usar cualquiera de los protocolos.

Analicemos ahora este simple XML Web Service:

@WebService Directive

Es la primer parte del archivo.asmx (veamos la primer línea).

Esta línea se usa para identificar los atributos del archivo XML Web Service.

En Language, obviamente, indicamos el lenguaje que se usó para crear el XML Web Service.

En class, se indica el nombre con el que esta clase será mostrada como un XML Web Service. Se puede definir dentro del .asmx mismo (como en este caso) o en un assembly que está en el directorio /bin de la aplicación Web.

Using

La sentencia using que continúan a la directiva WebService, permite acceder a las clases en los namespaces especificados sin tener que calificar los nombres (se puede indicar string sin tener que poner System.String).

Por ejemplo, el atributo WebService que se explica a continuación, en realidad es la clase System.Web.Services.WebServiceAttribute.

El atributo WebService

El atributo WebService es opcional y puede aplicarse a una clase pública que se expone como un XML Web Service. Se destaca: sólo puede aplicarse el atributo cuando está la keyword public declarado en la clase.

A través de este atributo se pueden agregar otros valores al XML Web Service como nombre, descripción y namespace.

A continuación, se muestra una definición de clase usando el atributo en C#.

```
 [WebService(Name="MyService", Description="Esto es una
descripcion", Namespace="http://www.dotnetjunkies.com/")]
public class MyWebServiceClass{
//...
}
```

Cuando especificamos namespace, estamos poniendo el namespace del XML Web Service.

La clase System.Web.Services.WebServiceAttribute

Analicemos ahora la línea

public class MyService (1) : WebService

Como en el using está System.Web.Services, de allí que sólo se ponga en la línea WebService: en realidad hace referencia a la clase System.Web.Services.WebService.

Derivar de esta clase, es heredar características comunes tales de ASP.NET tales como Application, Session, User y Context (para indicar, por ejemplo, si el manejo del estado de la XML Web Service es a través de toda la aplicación o sólo de una sesión).

Si bien no es imprescindible derivar de esta clase, puede ser un agregado de funcionalidad importante para nuestro XML Web Service.

El atributo WebMethod

Este atributo no es opcional. Es aplicado a todos los métodos que se deseen exponer en un XML Web Service. Y se debe declarar con la keyword public.

Lo que estamos indicando con este atributo es que los métodos serán accesibles por los protocolos standard de Internet. Se declara public, para que pueda ser accedido desde fuera de la clase padre.

Si bien pueden crearse métodos dentro de una XML Web Service con atributos private, internal o protected, no podrán ser “ofrecidos” como métodos Web.

A través de este atributo puedo indicar las siguientes características: BufferResponse, CacheDuration, Description, EnableSession, MessageName, TransactionOption.

En nuestro ejemplo, en C#, podríamos haberlo escrito así.

```
{
  [WebMethod(Description:=”Devuelve hola con el nombre”)]
  public String SayHello( String YourName )
  {
    return "Hello, " + YourName;
  }
```

La siguiente etapa: compilación, exposición

Una vez que se termina de escribir la clase del XML Web Service, las alternativas son ponerla en un archivo separado (con extensión de acuerdo al lenguaje

utilizado: si es Visual Basic, será un .vb; si es en C#, .cs), se compila y se coloca el assembly en el directorio /bin..

Si la clase está definida dentro del archivo.asmx, no es necesario la compilación, y el XML Web Service será submitido a un JIT compiler[4].

La página de testing la generará .NET Framework. Incluirá un link al WSDL del XML Web Service, y un link a cada método que tenga

El resultado de la invocación

Veamos el documento XML que retornaría este XML Web Service al invocarse (si el usuario fuera yo, Lía Molinari):

```
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://tempuri.org/">Hello, Lia Molinari</string>
```

Tempuri.org

En el documento XML devuelto, se indica en xlmns, http://tempuri.org/.

Este es un namespace temporario que se asigna a los xws creados en .NET framework, si no se especifica nada.

El namespace XML se usa para identificar el xws. De esta manera no habría confusión entre dos o más Web Services que se llamen igual, pero que figuren en distintos namespaces.

El namespace se indica en el atributo WebService, pues una de las propiedades es namespace.

Por ejemplo:

```
WebService(Namespace="http://www.dotnetjunkies.com")
```

10.9 Comparación entre J2EE y .NET orientado a WS

A continuación se comparan J2EE y .NET.

La aplicación J2EE está dentro del container que provee seguridad, manejo de transacciones, servicios de persistencia, etc.

La capa de business provee procesamiento y data logic.

[4] JIT (just in time) es un concepto que surgió con la tecnología Java. Al escribir un programa, el compilador Java genera un bytecode y lo pone en un archivo. Luego será interpretado por JVM (Java Virtual Machine).
Si está el JIT compiler, la JVM lee el archvio y se lo pasa al JIT, quien compila los bytecodes y lo lleva al código nativo de la máquina donde se está procesando.
El JIT compiler compila método por método, dinamicamente, a medida que son invocados.

Bussines logic se construye con componentes EJB (Enterprise JavaBeans). La conexión a las bases de datos la hace a traves de JDBC (Java Database Connectivity) o SQL/J. Si es necesaria la conexión con otros sistemas, se usa JCA (Java Connector Arquitecture).

También puede usar tecnologías WEB para la conexión (SOAP, UDDI, WSDL, ebXML[5]) a través de JAX APIs (Java APIs for XML).

La forma de conectarse de los business partners con una aplicación J2EE es a través de las tecnologías antes citadas.

Un servlet[6] puede aceptar un web service requerido por un business partner.

El servlet usa JAX API's para las operaciones de Web Service

Las applets y aplicaciones se conectan a la capa EJB a través del protocolo IIOP (Internet Inter-ORB). Browsers y dispositivos wireless se conectan con JSPs (JavaServer Pages) que le brinda al usuario HTML, XHTML o WML.

Según los autores, .NET es una estrategia de producto y J2EE s un standard para escribir productos. Y allí residiría la primer gran diferencia.

En .NET, la parte de la aplicación reside en un container semejante al previsto en J2ee que provee calidad de servicio.

La capa de negocio está construida por componentes. NET. La conexión a bases de datos es a través de ADO.NET (Active Data Objects) y la interacción con sistemas existentes es a través de Microsoft Host Integration Server 2000 (por ejemplo, COM TI, COM transaction Integrator, es parte de ella). También hay conexión a través de la tecnología de Web Service.

Los business partners se pueden conectar a través de tecnologías Web Services como SOAP, UDDI, WSDL, y además, BizTalk).

Los clientes , browsers y dispositivos wireless pueden conectarse a través de Active Server Pages (ASP.NET), que le brinda al usuario HTML, XHTML o WML. Las interfases de usuario también pueden desarrollarse en Windows Forms.

10.9.1 Tabla comparativa

En el artículo citado, Chad Vawter and Ed Roman muestran esta tabla comparativa para simplificar la comparación entre J2EE y .NET.

Característica	J2EE	.NET
Interprete	JRE	CLR

[5] EbXML es un conjunto de especificaciones XML y procesos relacionados para la colaboración e integración en la implementación de fraestructuras de e-commerce (e-infraestructure) en ambientes B2B.
[6] Un servlet es un objeto Java rientado al servicio de request/response.

Página Web dinámicas	JSP	ASP.NET
Componenetes Middle-Tier	EJB	Componentes manejadas por .NET
Acceso a BD	JDBC, SQL/J	ADO.NET
SOAP, WSDL, UDDI	Si	Si
Otras características como load balancing	Si	Si

En ese mismo paper se citan las diferentes ventajas y desventajas que ofrecen estos modelos.

Con respecto a ambas:

- ✓ Exigen capacitación de los desarrolladores. En el caso de J2EE, en Java, y en el caso de .NET, en Orientación a Objetos.
- ✓ En cualquiera de ellas se pueden construir Web Services.
- ✓ Ambas tienen un bajo costo de sistema. En el caso de J2EE se puede usar jBoss[7] o Linux, y en el caso de .NET, Windows/Win32.
- ✓ Ambas ofrecen, teóricamente, escalabilidad (yo agregaría que también, extensibilidad).

.NET supera a J2EE en:

- ✓ .NET comenzó con Web Services antes que J2EE.
- ✓ Tiene más experiencia en shared context
- ✓ Tiene un modelo de programación más simple
- ✓ Mantiene una neutralidad de lenguaje cuando se desarrollan nuevas aplicaciones de Ebusiness, mientras que J2EE trata a los otros lenguajes como aplicaciones separadas.

J2EE supera a .NET en:

- ✓ es una plataforma probada, con nuevas API's para web services.
- ✓ el código ya desarrollado en J2EE se puede llevar fácilmente a J2EE Web Services, con poca reescritura, lo que no ocurre en la portabilidad de Windows DNA y .NET
- ✓ .NET Web Services no es interoperable con standards de la industria. El framework BizTalk tiene extensiones SOAP que son propietarias y no soportan ebXML

10.10 Notas

Importantísima la colaboración de los distintos artículos de Seven, [SEV001] y [SEV002].

La comparación entre J2EE y .NET, de [VAW001] merece destacarse.

[7] JBoss es un server de desarrollo de aplicaciones open source. Se basa en J2EE. JBoss es FREE SOFTWARE. Ver http://www.jboss.org/

Obviamente, reitero y con más fuerza en este caso, [PLA002] y [CHA002]. [MIC000], [MID002], [RUB002] son otros artículos recomendables.

Capítulo XI

Sun Web Services

11 SUN Web Services

La inclinación natural a usar Java para comunicar aplicaciones en diferentes plataformas, dado su código portable, se reforzó cuando se incorporaron las APIs necesarias para trabajar con XML.

J2EE incluye características tales como seguridad, manejo de transacciones distribuidas y lo necesario para implementar WS.

La herramienta de desarrollo de WS en Java, JAVA WSDP (Java Web Services Developer Pack) pone a disposición del programador estas APIs. El WSDP incluye el TOMCAT Container y el J2EE container, cuando los archivos JARWSDP se instalan en el J2EE SDK.

11.1 APIs para Web Services

J2EE provee las APIs necesarias para implementar WS, pensando en la interacción entre procesos que se ejecutan en distintas plataformas.

Veamos cuáles son las APIs que se proveen para WS, usando XML.

JAXP (Java Api for XML processing): procesa documentos XML usando DOM, SAX y XSLT. A través de JAXP la aplicación puede "parsearse" y trabajar con documentos XML.

JAXR (Java Api for XML Registries): pemite trabajar con registros sobre la Web. Por eso soporta especificación UDDI y otros standards.

JAX-RPC: usa SOAP y HTTP para que los clientes puedan hacer RPCs basados en XML sobre la red. Gracias JAX-RPC se puede trabajar con documentos WSDL. El uso de JAX-RPC y WSDL hace posible la interacción entre plataformas Java y no-Java.

SAAJ (SOAP with attachments API for Java): permite trabajar con mensajes SOAP.

Además hay un conjunto de herramientas para el manejo de performance, testing y monitoreo.

11.2 JAXM

Java API for XML Messagging brinda una forma standard de enviar documentos XML sobre la red desde plataformas JAVA.

Se basa en SOAP y puede extenderse para trabajar con ebXML.

Cuando se usa un servicio de messaging provider, éste hace todo el trabajo para el transporte y el ruteo de mensajes. de esta forma, puede implementarse el uso de nodos intermedios para llegar al nodo destino.

Cuando se usa un messaging provider, JAXM le entrega los mensajes y el cliente JAXM se desentiende de la tarea del provider. Invoca métodos Java y el resto lo hace el provider.

Si no se usa un message provider, también se puede usar JAXM. El cliente JAXM, en este caso llamado *standalone client*, envía mensajes point-to point directamente al servidor de ese servicio en forma request-response (sincrónica).

11.3 JAXR

A través de JAXR, los desarrolladores de aplicaciones Java tienen una manera uniforme de usar Business Registries que se basan en standares como ebXML ó UDDI (ver 9).

11.4 JAXP

Como trabaja con DOM y SAX, se puede optar por trabajar en el modelo orientado a eventos o construyendo un árbol para representar los datos.

JAXP también soporta XSLT y gracias a ello el desarrollador puede controlar la presentación de los datos y transformarlo a otros XML´s u otros formatos como HTML. Lo soporta a través del paquete *javax.xml.transform*.

Puede trabajar con schemas pues tiene soporte para namespaces.

Lo interesante es que se puede usar cualquier parser compatible con XML. Esto le permite conectarse con otra implementación de APIs de SAX y DOM, o en procesador XSL, garantizando lo que se llama *pluggability layer*.

11.5 JAX-RPC

Un remote procedure call JAX-RPC se implementa como un mensaje SOAP de solicitud-respuesta.

El cliente puede invocar por RPC un método JAVA y JAX-RPC se encargará del marshalling, unmarshalling, detalles de transmisión automática.

Para que un cliente pueda comunicarse con un servicio remoto, necesita construir stubs, clases de bajo nivel. Por ello es importante contar con el documento WSDL sobre ese servicio para construir el stub en base a esa especificación.

En JAX-RPC la herramienta para generar stubs a partir de un documento WSDL, se llama *wscompile*.

Del lado del servidor se deben crear *ties*, que son clases de bajo nivel necesarias para la comunicación con el cliente.

La herramienta para crear ties es *wsdeploy*.

11.6 SUN y los Web Services, hoy

En Junio de 2003, SUN hizo anuncios que demuestran su interés en continuar con la plataforma JAVA para el desarrollo de WS.

- Incorporó soporte para WS en el J2EE 1.4 SDK y especificaciones según dicta la WS-I (Web Services Interoperability Organization).
- Puso disponible JAVA WSDP (Java Web Services Developer Pack) con capacidades mejoradas para WS.
- Donó APIs incluidas en J2EE 1.4 SDK a la iniciativa Java.net para versiones open source. Incluye las APIs: JAX-RPC, JAXB, SAAJ.
- Pondrá disponible cuanto antes y capacidades de identidad sobre la red usando SAML, especificaciones Liberty Alliance y standards de OASIS Ws-Security. Además agrega APIs como JAAS para autenticación, soporta XML y SOAP.

WS-I ha especificado lo que se llama Basic Profile, que asegura que las componentes para la implementación de la tecnología WS, trabajan juntas.

11.7 Notas

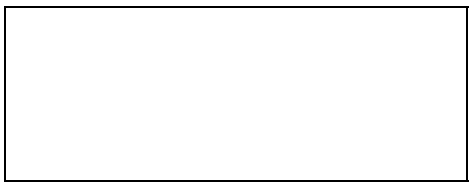
Para este capítulo consulté [JAV002] y [MID002].

[SUN002] y [SUN003] fueron útiles, para conocer la oferta actual de SUN en este modelo.

También consulté tutoriales sobre Java, ofrecidos en la Red.

Capítulo XII

Web Services y Websphere



12 Web Services y Websphere

IBM introdujo Websphere a mediados de los 90s.

Inicialmente era un server para aplicaciones Web, con una fuerte orientación hacia la construcción de websites de e-commerce.

Websphere fue creciendo y se transformó en una herramienta importante para la integración de sistemas Legacy, con sus IBM Host on demand, MQSeries, CICS Transaction Gateway según el artículo de la nota al pie.

IBM ve los WS como una forma de actualizar sus aplicaciones 3270 y 5250 y llevarlas a Internet. El problema de hacer estas aplicaciones Web Enabled es que hay muchas formas de hacerlo, pero para la interoperabilidad que exige hoy Internet puede complicarse si no se adapta a standards.

12.1 Websphere Studio Application Developer Integration Edition.

IBM ofrece productos para mejorar las aplicaciones legacy, un poco separadas de Websphere con el objetivo de poner rápidamente a disposición de usuarios en la red, datos del legacy.

Websphere Host on demand puede usarse standalone (sin el application server) o HATS (Host Access Transformation Server) que permite llegar a las viejas aplicaciones a través de una interfase Web.

Pero IBM apostó a Websphere para la implementación de WS, accediendo a CICS, IMS y DB2 y tratando tanto las aplicaciones del mainframe (legacy) como las de e-commerce, como WS.

Esto se realiza con **Websphere Studio Application Developer Integration Edition**.

12.2 IBM WSDK (Websphere SDK for Web Services)

WSDK es la herramienta para implementar WS en Websphere permitiendo crear, buscar, invocar y testear WS.

Está orientado a Java y permite crear, por ejemplo, web services, a partir de componentes Java ya existentes.

A través de WSDK se pueden crear y testear WS e acuerdo a las pautas de la organización WS-I.

Soporta SOAP, WSDL, UDDI.

Los WS creados con WSDK pueden integrarse a través de websphere Studio y extendido y distribuido a través de **Websphere Application Server**.

Contiene aplicaciones que muestran como escribir WS, ó clientes que consumen WS y cómo publicar y encontrar WS usando UDDI.

12.3 Herramientas en Websphere

Websphere Application Server: soporta standards abiertos y la tecnología necesaria para desarrollar, publicar y distribuir WS, usando UDDI, SOAP, J2EE, WSDL y la interacción con registros UDDI.

Websphere Studio Technology Preview for WS: es útil para desarrollar e integrar WS en procesos de negocio existentes.

Websphere Business Integrator: se utiliza para integrar y manejar el flujo de aplicaciones WS. tiene las características transaccionales heredadas de MQSeries pero trabaja intercambiando mensajes SOAP.

DB2 soporta UDDI y SOAP y permite que las aplicaciones WS accedan a datos almacenados en DB2.

Tivoli WS Manager: monitorea la performance del ambiente WS.

Lotus Software Portfolio: permite utilizar ambientes colaborativos, e-learning. Puede acompañarse con Domino Application Server, Domino Worflow, Knowledge Discovery System, Lotus Smartime y Lotus Learning Space.

12.4 Cómo construir WS con Websphere Studio

Websphere Studio V5 está basado en el proyecto Eclipse (http://www.eclipse.org). La versión 5.1 conforma el WS-i Basic Profile 1.0.

Mejora la versión 4 en cuanto a la creación de web services desde archivos DAXD existentes (Documents Access Definition eXtension). Estos archivos definen el "mapeo" entre definiciones de bases de datos y web services.

También mejora el uso de registros UDDI privados usando DB2 o Cloudscape y XML Schema para crear componentes Java y WS.

v5.1.1 permite crear WS desde definiciones WSDL. El editor WSDL es una herramienta de v5.1.1.

Veamos cómo se construye un WS en Wesphere Studio v5. Recomiendo la lectura de BEN002, sobre este tema: es un artículo muy claro, con la información imprescindible, sin detalle.

Supongamos que deseamos exponer en la Web lo necesario para que los clientes puedan librar una orden de compra sobre productos que vendemos o servicios que ofrecemos.

Cada cliente tiene su propio ambiente, SO, lenguaje, etc.

Lo primero que se debe crear es el servicio *crearOrdenCompra*. Esta orden de compra debe tener todos los datos necesarios para que la orden se libre satisfactoriamente.

El cliente invoca ese servicio y se le devuelve un número de orden que le servirá para identificar la orden y controlar su ejecución.

Los datos para la orden son pasados como un documento XML.

Pero puede haber nuevos clientes y, por lo tanto, no "conocen" cómo invocar esa orden. Por eso es importante "publicar" ese servicio que puede atraer futuros clientes.

Este ciclo de descubrimiento-invocación tendría el siguiente orden:

1. Descubrimiento del WS en un registro UDDI
2. El cliente crea el proxy usando el documento WSDL, según registros UDDI
3. Se genera el mensaje SOAP desde el cliente que hace la invocación

Para implementar el servicio en Websphere Studio, se debe crear un *web project*. Y luego se debe crear la clase *OrdenCompra*.

La clase OrdenCompra tiene todos los datos necesarios para hacer la orden de compra. Lo que se creará es una clase Java a través del Dialog Java Class.

```
.
.
.
public class OrdenCompra {

        // input data
        private string NombreCliente;
        private string DomicilioCliente;
        private string CiudadCliente;
        private string CodigoCliente;
        private date FechaSolicitud;

        // output data
        private int NumeroOrden

        static int ProxNumeroOrden = 0;

        public OrdenCompra()  {
                esteNumeroOrden = ProxNumeroOrden;
                ProxNumeroOrden++;
        }
        .
        .
        .
        public void setNombreCliente (string NombreCliente) {
```

```
                    Este.NombreCliente=NombreCliente;
                public void setDomicilioCliente (string DomicilioCliente) {
                    Este.DomicilioCliente=Domicilio.Cliente;
                ...
                public int getNumeroOrden() {
                    return this.NumeroOrden;
                }
}
```

La otra clase que se debe crear es la clase *AdminOrdenCompra*.

Esta clase maneja un vector de ordenes de compra. También implementa los métodos para crear una nueva orden de compra.

```
    public class AdminOrdenCompra  {

    static Vector orden = new Vector() ;

    // entrada de input data y crea una nueva orden y devuelve el número de oden
      public int createOrdenCompra (
            private string NombreCliente;
            private string DomicilioCliente;
            private string CiudadCliente;
            private string CodigoCliente;
            private date FechaSolicitud)   {

                OrdenCompra newOne = new OrdenCompra();
                newOne.set NombreCliente (NombreCliente);
                newOne.set DomicilioCliente (DomicilioCliente);
                ...
                orden.addElement (newOne);
                return newOne.getOrderNumber();
          }
}
```

En Websphere Studio, el tipo de web service, por default se asume que es un JavaBean Web Service. Se hace entonces, un “wrapping” de una clase existente como web service.

En un check box, se elige *Generate a proxy*. Y entonces, el proxy cliente pasa a ser un Java proxy. Esto generará un proxy del lado del cliente.

En una página de Web Service Java Bean Identity, se seleccionan los métodos que quieren exponerse como WebSErvices. En nuestro caso sería *createOrdenCompra*.

Con respecto a WSDL, en Websphere Studio V5 viene con un editor WSDL que simplifica la tarea de trabajar con documentos WSDL. Incluso se cita como el mejor logro de v5.1.1.

A través de este editor, podemos ver el documento WSDL generado.

El documento se nos muestra separado en tres paneles: uno muestra la relación entre servicios, portTypes y mensajes.

Otro punto es la elección entre codificar SOAP o usar XML literalmente. Es lo que llamamos fijar el encoding style (ver 7.4.1).

Usar la codificación SOAP permite trabajar rápidamente, pues se delega la generación del documento a SOAP y sus librerías..

Si uso el XML literal, debemos extraer el dato y ponerlo dentro de un elemento XML. Se debe crear explícitamente el documento XML que contendrá los elementos que se necesitan.

12.5 Notas

Sobre cómo construir un WS en Wesphere Studio v5, recomiendo la lectura de [BEN002], sobre este tema: es un artículo muy claro, con la información imprescindible, sin detalle.

Websphere fue creciendo y se transformó en una herramienta importante para la integración de sistemas Legacy. Para analizar ese crecimiento recomiendo [KEN003].

Recomiendo *MUY ESPECIALMENTE* [WAH002]. También recomiendo [CHA003], [FLU002], [IAW003].

Capítulo XIII

El Proyecto MONO

13 El proyecto MONO

13.1 Qué es Mono?

Mono es la implementación open source, versión Unix, del framework de desarrollo .NET.

Se inicia a principios del 2001, como un proyecto dentro de Ximian.

GNOME tiene una plataforma de desarrollo que presentaba algunos problemas como falencias en el garbage collection, y los bindings a los distintos lenguajes.

GNOME ofrece trabajar en distintos lenguajes: C++, C, Ruby, Python, etc. Pero para ello deben prepararse las librerías de C, para que puedan trabajar con ellos.

Esto tomaba mucho tiempo pues había que preparar estas librerías antes de liberar el uso del lenguaje. Como consecuencia de este inconveniente, se retrasaban muchos proyectos que se iniciaban sobre Ximian.

Entonces, Microsoft con su proyecto .NET estaba logrando esa interoperabilidad desde el lenguaje, a partir de crear desde el compilado, un código intermedio. La idea no es nueva, pues se acerca al concepto que utiliza Java.

Este código intermedio es llamado CIL (Common Infraestructure Language) y acuerda el standard ECMA 335.

La gente de Ximian decidió analizar la plataforma .NET. Y una vez convencidos de ésta y otras ventajas adicionales que ofrecía el ambiente de Microsoft, lo primero que se comenzó a desarrollar fue el compilador de C#, el lenguaje desarrollado por Microsoft para .NET.

Mono incluye un compilador para C#, un conjunto de librerías de clases (compatible con .NET), una infraestructura de lenguaje común (CLI) que contiene un class loader, compilador Just-in-time y un runtime para garbage collection.

Tiene implementaciones para ADO.NET y ASP.NET como parte de la distribución.

Esta plataforma permite desarrollar y ejecutar aplicaciones que interoperan con el ambiente .NET.

El compilador C# de Mono, MCS, se puede usar tanto para compilar programas C#, como para compilarse a sí mismo (está escrito en C#).

Dado que surgió de Ximian, en Mono es importante lograr el acceso a bases de datos tales como POstgresSQL y MySql. Provee herramientas que pueden usarse sobre ADO.NET y otras tecnologías como ASP.NET, XML.

13.2 Algunas herramientas en Mono

SQL#: Herramienta para querys a nivel de línea de comando para ejecutar sentencias SQL o batchs de comandos SQL.

SQL# CLI se distribuye con el runtime de Mono y la librería de clases como un ejecutable sqlsharp.exe.

SQL# for GTK#: herramienta de query para SQL en modo gráfico. En Mono CVS figura como sqlsharpgtk.

XML schema Definition tool (xsd.exe): Permite generar schema XML desde XDR (usado anteriormente en Microsoft), desde un file XML, desde un assembly (usando la clase serializer).

Application XML Configuration File Editor: herramienta para crear y editar un archivo para configuración de la aplicación como usa .NET, para que un usuario pueda conectarse a l base de datos que usa el programa (en .NET se llama Accounting.exe).

El soporte para ASP.NET se divide en dos partes: Web Forms y Web Service.
XSP es un Web Server escrito en C# que puede ser usado para correr aplicaciones ASP.NET.

13.3 Desarrollo de aplicaciones

Si escribiéramos el famoso programa HolaMundo en C# quedaría algo así:

```
class HolaMundo
{
        static public void main()
        {
                system.console.writeline("Hola Mundo #")
        }
}
```

Como vemos, este código es muy parecido a Java.

Luego se compila mediante el compilador C# para Mono.

```
mcs –o HolaMundo.exe  HolaMundo.cs
```

El .exe que sale de esta compilación, ya tiene formato CIL.
En GNU/LINUX al poner la extensión .exe se indica que puede llevarse a un sistema Windows. No obstante, no es imprescindible que en el ambiente Mono tenga esta extensión.

Para ejecutarlo, se invoca al intérprete de Mono:

```
mono HolaMundo.exe
```

13.4 Estado actual

Actualmente, Mono permite usar la plataforma .NET en sistemas GNU Linux sin necesidad de usar Windows.

Ya están disponibles la mayor cantidad de las clases, incluso para el acceso a base de datos remotas.

Tiene soporte para ASP.NET.

Se han enunciado proyectos basados en Mono y se está tratando de integrarMono a aplicaciones ya existentes.

En el servidor se ha implementado un módulo para Apache para ejecutar el código CIL dentro de Apache.

13.5 Mono y los Web Services

De acuerdo a lo indicado por mail enviado a Miguel de Icaza, la parte para implementar Web Services desde Mono, ya está implementada.

Esta implementación cumple con las especificaciones de .NET al respecto, utilizando los protocolos enunciados en el punto 10.4.

13.6 Notas

Como bien dice Miguel de Icaza, la ventaja de desarrollar MONO es no gastar en bibliografía, pues basta consultar la bibliografía de .NET.... ;-)

El sitio “oficial” de Mono, http://www.go-mono.com, es el lugar de consulta por excelencia. También recomiendo consultar http://www.gnome.org.

14 Conclusiones

Este trabajo me exigió el análisis de distintas metodologías de comunicación entre procesos/objetos.

Si bien hay una tendencia hacia el uso de protocolos genéricos, cada empresa tiene su propia terminología para designar los elementos que intervienen. Y algunas características que parecían diferir, analizándolas profundamente, indicaban lo mismo. Ocurrió con el término proxy, cuando en la referencia a objetos remotos, se crea un "espejo" del objeto para facilitar la transparencia.

Por otro lado, se debe distinguir entre lo que se define como standard, y lo que serían consorcios o reuniones de grandes empresas que propicien el uso de un protocolo particular.

Si bien algunos de los protocolos que hoy citamos como standares y avalados por la W3C surgieron de esta manera, sugiero la "lectura de la letra chica" ante las presentaciones de los protocolos: que un protocolo lo propicie BEA, IBM y Microsoft, no quiere decir que sea un standard.

El punto es: hasta dónde podemos confiar en los acuerdos? Las grandes empresas de IT, obviamente, viven en una continua competencia, y en el afán de captar nuevos clientes, pueden llegar a borrar con el codo aquello que firmaron con la mano.

También se debe considerar qué ocurre con la interoperabilidad ante nuevas versiones de los protocolos. Si bien en las IDLs y headers se definen las versiones que se están usando… se garantiza la *compatibilidad ascendente*?

En el ambiente open source también hay algunas controversias sobre qué usar?
CORBA, BONOBO, MICO? Incluso algunos miran con un poco de desconfianza la interoperabilidad con ambientes Microsoft.
CORBA es respetado por su surgimiento desde OMG, pero se lo define una y otra vez como "complejo". Pero es injusto criticarlo: fue una de las primeras iniciativas, cuando el modelo era otro.
BONOBO es un arquitectura de GNOME para crear componentes reusables de software. Y la gente del mundo de open source miran BONOBO con cariño, pues fue diseñado e implementado por la comunidad del software libre para desarrollar aplicaciones de gran escala.

Algunas voces ya se encuentran en la Red, pidiendo REST en vez de SOAP (sugiero leer http://www.prescod.net/rest/security.html).

También sobre SOAP se quejan en sourceforge:

"…Since then, however, SOAP has been turned over a W3C working group. Unfortunately, the working group has been adding a laundry-list of strange features to SOAP. As of the current writing, SOAP supports XML Schemas, enumerations, strange

hybrids of structs and arrays, and custom types. At the same time, several aspects of SOAP are implementation defined.
Basically, if you like XML-RPC, but wish the protocol had more features, check out SOAP. :-) … **ver http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html**

Por eso muchos recomiendan XML-RPC, como una forma simple de hacer llamados a procedimientos remotos usando XML y HTTP.

En cuanto a desarrollo de Web Services, he tenido oportunidad de conversar con desarrolladores de .NET y si bien usan SOAP y XML, en algunos casos usaron SOAP para envolver binarios, para mejorar el throughput general.

El uso de WS en .NET si bien va a tener la trascendencia que asegura la cantidad de usuarios Microsoft en el mundo, exige un framework en el cliente, "quebrando" un poco la filosofía de interoperabilidad que propicia WS-I.

En general, noto aún reticencia para publicar los web services, a través de UDDI y WSDL. Creo que aún no ha madurado el proceso de "oferta" de servicios: el desarrollador no ve el "negocio" de publicar y ofrecer al mundo un producto.

Vale la aclaración que queda fuera del alcance de esta tesis, el tratamiento de la seguridad en Web Services. Si bien se garantiza SSL sobre HTTP, protocolo dominante en WS, es un tema de gran trascendencia y relevancia que amerita un mayor análisis e investigación.

Por último, los lenguajes formales para la descripción del proceso de negocio creo que es un tema interesante para investigar. Exige un conocimiento acabado de WSDL y, obviamente de XML. Integrar la modelización de los proceso de negocio, la definición de procesos abstractos o privados dentro de WSDL a WS, es un desafío atractivo.

Tendrá esta nueva tendencia hacia la interoperabilidad y los standares el mismo destino que el esperanto? Hablaremos un idioma común, o construiremos nuevas Torres de Babel?

Lía Molinari
Abril 2004

Bibliografía y Referencias

15 Bibliografía y referencias

AUS002 Web Services Architecture Requirements. W3C Working Draft 14 November 2002. Daniel Austin (, W. W. Grainger), Abbie Barbir (Nortel Networks), Christopher Ferris (IBM), Sharad Garg (Intel Corporation).

BEN002 Building Web Services, Part 1: Build and Test. http://www-136.ibm.com/developerworks/websphere

BER097 CORBA vs DCOM. http://www.sims.berkeley.edu/courses/is206/f97/GroupG/projects/corba.html

BOX000 Simple Object Access Protocol (SOAP) 1.1. W3C Note 08 http://www.w3.org/TR/2000/NOTE-SOAP-20000508. Don Box (DevelopMentor), David Ehnebuske (IBM), Gopal Kakivaya,(Microsoft), Andrew Layman (Microsoft), Noah Mendelsohn (Lotus Development Corp.), Henrik Frystyk Nielsen (Microsoft), Satish Thatte (Microsoft), Dave Winer (UserLand Software). May 2000

CAR002 What Are Web-Enabled Applications? by Kathleen s. Carr

CER002 Web Services Essentials. Ethan Cerami. Top Ten FAQs for Web Services, By Ethan Cerami. http://webservices.xml.com/pub/a/ws/2002/02/12/webservicefaqs.html

CHA001 Web Services Architecture. W3C Working Draft, 14 November 2002. Editors: Champion (Software AG), Ferris (IBM), Newcomer (Iona), Orchard (BEA Systems)

CHA002 Understanding .NET: a tutorial and Analysis. By David Chappell. Editorial Addison-Wesley.

CHA003 Integrating applications with Web Services using Websphere. By Nicholas Chase. Presented by Websphere Developer Domain.

CLA002 Business Architecture for a Web Services Brokerage. Understanding the Business Context of Web Services. Del libro Web Services Business Strategies y Architectures. Mike Clark.

COL001 What is Net Centric Computing?. By Bernard Cole

COU001 Sistemas Distribuidos. Coulouris, Dollimore, Kindberg. Editorial: Addison-Wesley. ISBN 84-7829-049-4

CUR002 Business Process Execution Language for Web Services. Version 1.0. Authors: Curbara, Golan, Kein, Leymann, Roller, Thatte, Weerawarana (IBM, BEA,Microsoft). July 2002

Artículo muy interesante, mostrando en detalle el lenguaje BPEL4WS.

FLU000 Web Services and IBM. Greg Flurry.

GAR002 Choosing the right messaging foundation for business application connectivity. By Scott Garvey. Director Web Services, Microsoft Corporation

GIS001 Web services architect, Part 3: Is Web services the reincarnation of CORBA? http://www-106.ibm.com/developerworks/webservices/library/ws-arc3/ Dan Gisolfi. July 2001

GLA000 The Web Service ®evolution. Graham Glass. Nov 2000.

IAW003 IBM Announce Websphere SDK for Web Services V5.0.1. http://www.webservices.org

JAV002 Java Technology and Web Services Overview. http://java.sun.com/webservices/overview.html

KEN003 Moving forward with Websphere and Web Services. Joe McKendrick. Abril 2003

KIR002 Object-Oriented Software Development Made Simple with COM+ Runtime Services. Mary Kirtland.

MAR0001 XML con ejemplos. Benoit Marchal. Editorial: Pearson Educación. ISBN: 970-26-0163-0. Año 2001

MIC001 Microsoft Component Services. Server Operation System: A technology Overview. Microsoft Corporation

MIC002 Introducing XML Serialization NET Framework Developer's Guide. http://msdn.microsoft.com/library/en-us/cpguide/html

MIC003 Building User-Centric Experiences with .NET Services. http://www.microsoft.com/netservices/userexperiences.asp

MID000 Middleware Demystified, http://www.cio.com/archive/051500_middle.html, May 15, 2000

MID002 J2EE vs. Microsoft .NET Application Server and Wer Services Benchmark. Middleware Company. October 2002.

MOY Taller Práctico de CORBA en GNOME- By: Rodrigo Moya. Congreso HispaLinux

NUT992 Open Systems. Gary Nutt. Editorial: Prentice Hall Series. ISBN 0-13-636234-6. Año 1992.

Libro con conceptos básicos sobre sistemas abiertos.

PLA002 Introducing Microsoft .NET. By David S. Platt. Editorial: Microsoft Press. ISBN 0-7356-1571-3

RIO002 Business Process Standards for Web Services. David O' Riordan.

Excelente artículo sobre business process , donde se describen distintos protocolos y su grado de standarización

ROB002 Basic Understanding of WSDL documents and how to Use a WSDL File to Access a SOAP Service. Lance Robinson

RUB002 Microsoft® .NET Explained. Por Daniel Rubiolo, J.D. MEIER, Edward JEZIERSKI y Alex MACKMAN

SAL000 Mark Colan (IBM): SOAP + UDDI + WSDL = Web Services. Ken Sall. Diciembre 2000

SCR002 Understanding SOAP. Kenn Scribner, Mark Stiver. Editorial: SAMS. ISBN 0-672-31922-5. Marzo 2002

SES000 COM+ and the Battle for the Middle Tier . Roger Sessions. Editorial Wiley. ISBN 0-471-31717-9. Año 2000.

SEV001 Creating a Simple XML Web Service. By: Doug Seven.

SEV002 Understanding XML Web Services and the .NET Framework. By: Doug Seven.

SHI000 What is P2P... and what isn't. Clay Shirky. Noviembre 2000.

SIL001 Operating Systems Conceptos. Silberschatz, Galvin. Editorial: Addison-Wesley. ISBN 0-201-54262-5

STA002 Operating Systems. William Stallings. Prentice Hall. ISBN: 84-205-3177-4

STE001 Why UDDI will succeed, quietly: Two factors Push Web Servcie Forward. An Analysis Memo from The Stencil Group. 2001

SUN002 Open Net Environment (Sun ONE) http://wwws.sun.com/software/sunone/Sun[tm]

SUN003 Sun Web Services and J2EE Version 1.4. San Francisco, JavaOne Developer Conference. June 11, 2003.

TAP001 Web Services Description Language (WDSL) Explained. By Carlos C. Tapang. Infotects

THO001 Enabling Open, Interoperable, and Smart Web Services *The Need for Shared Context,* March 2001. Anne Thomas Manes, Sun Microsystems. http://www.w3.org/2001/03/WSWS-popa/paper29#top.

TIL000 The Era of the Net- Centric Computing. By Scott Tilley

UDE000 UDDI Executive White Paper. Sept 2000. UDDI.org

UDT000 UDDI Technical White Paper. Sept 2000. UDDI.org

VAW001 J2EE vs. Microsoft .NET. A comparisson of building XML-based Web Services. By Chad Vawter and Ed Roman (SUN). June 2001.

WAH002 Self Study Guide:Websphere Studio Application Developer and Web Services. Ueli Wahli. Redbook IBM. Febrero 2002.

XML000 Introduction to XML Serialization. http://developer.apple.com/techpubs/webobjects/XML_Serialization/Introduction