MANUAL PARA USO DE WEB SERVICES

ISC. Erika Ávila

TABLA DE CONTENIDO

Pag.

INTRODUCCIÓN

La creación de aplicaciones como un conjunto de componentes distribuidos a través de una red de equipos que trabajan de forma conjunta se ha convertido en una práctica habitual. Tradicionalmente, la lógica de las aplicaciones distribuidas requería el uso de la tecnología de objetos componentes, como el Modelo distribuido de objetos componentes de Microsoft® (DCOM, Distributed Component Object Model), la Arquitectura de agente de peticiones de objetos común (CORBA, Common Object Request Broker Architecture) de Object Management Group o la Invocación de métodos remotos (RMI, Remote Method Invocation) de Sun.

Estas tecnologías ofrecían una arquitectura confiable y escalable para satisfacer las numerosas necesidades de las aplicaciones.

DCOM estaba construido sobre la arquitectura de llamadas a procedimientos remotos (Remote Procedure Call, RPC), era mucho más sencillo de utilizar que RPC pero mantuvo algunas de las limitaciones de esta arquitectura.

Estas tecnologías basadas en componentes funcionan bastante bien en entornos de Intranet. No obstante, si se utilizan a través de Internet, surgen dos problemas de gran importancia. En primer lugar, las tecnologías no interoperan. Aunque todas trataban con objetos, surgían discrepancias en cuanto a los detalles, como la administración de los ciclos de vida, la compatibilidad para constructores y el grado de compatibilidad para herencia. El segundo problema, y más importante, es su dependencia de la comunicación estilo RPC, que llevaba normalmente a un escenario de sistemas estrechamente asociados creados en torno a las invocaciones explícitas de los métodos de objetos.

Una problemática adicional que DCOM presenta es el hecho de que no es neutral con respecto a las plataformas. DCOM era básicamente una opción exclusiva de Microsoft Windows, y debido a eso, la increíble cantidad de versiones de Windows hacía que mantener un sistema de cierta envergadura basado en COM fuera una tarea difícil.

Actualmente el mundo de los negocios necesita técnicas más poderosas para escalar las soluciones de negocios sin incrementar la complejidad.

El modelo de servicios Web promete entregar estas soluciones de una manera más sencilla y simple que la que ofrecen las arquitecturas de llamadas a procedimientos remotos y las tecnologías orientadas a objetos.

Los servicios basados en tecnología Web han empezado a emerger como área de interés para la tecnología y los negocios. Cada vez más empresas están adoptando las tecnologías de servicios Web. Este concepto sirve para agrupar los proyectos que hacen posible la interconexión entre distintos sistemas, de una o diferentes empresas, con la finalidad de intercambiar información o realizar transacciones a través de Internet.

Aunque los clientes dinámicos facilitan el tratamiento de algunas de las limitaciones de la arquitectura Web desde la perspectiva del usuario final, los servicios Web serán la introducción de una gran tendencia en cuanto al modo en que las aplicaciones comparten entre sí los datos.

Los Servicios Web proporcionan un conjunto de protocolos que permiten a las aplicaciones exponer su funcionalidad y sus datos a otras aplicaciones a través de Internet.

Principalmente, se trata de componentes de software en la red. Al igual que los anteriores protocolos de computación distribuida, como DCOM, DCE, CORBA y Java RMI, los Servicios Web proporcionan un marco de trabajo de llamadas remotas a objetos para intercambiar datos.

Definiciones de Servicio Web

1. Un servicio Web es una colección de funciones que son presentadas como una sola entidad y es anunciada en la red para ser usada por otros programas.
2. Los servicios Web son los bloques de construcción para crear sistemas distribuidos abiertos.
3. Son aplicaciones auto-contenidas y modulares que pueden ser:
   - Descritas mediante un lenguaje de descripción de servicio, como el lenguaje WSDL (Web Service Description Language)
   - Publicadas al someter las descripciones y políticas de uso en algún Registro bien conocido, utilizando el método de registro UDDI (Universal Description, Discovery and Integration).
   - Encontradas al enviar peticiones al Registro y recibir detalles de ligamiento (*binding*) del servicio que se ajusta a los parámetros de la búsqueda.
   - Asociadas al utilizar la información contenida en la descripción del servicio para crear una instancia de servicio disponible o proxy.
   - Invocadas sobre la red al utilizar la información contenida en los detalles de ligamento de la descripción del servicio.
   - Compuestas con otros servicios para integrar servicios y aplicaciones nuevas

Descripción general

Los servicios Web actuales son bastante sencillos. Proporcionan un lenguaje y una sintaxis independiente de la plataforma para intercambiar datos complejos mediante mensajes. Las características esenciales de los servicios Web se implementan mediante XML, lo que permite que cualquier plataforma pueda utilizar fácilmente esta tecnología.

Los servicios Web se componen principalmente de dos estándares fundamentales: Protocolo simple de acceso a objetos (SOAP) y Lenguaje descriptivo de servicios Web (WSDL).

SOAP es el protocolo utilizado para intercambiar mensajes y datos entre aplicaciones.

WSDL tiene una sintaxis que permite describir las funciones de un servicio Web. Éste podría utilizarse como herramienta para generar el código necesario para acceder al servicio Web o para facilitar herramientas que proporcionen indicios de código a un desarrollador que utilice el servicio.

Las dos especificaciones son muy utilizadas por los líderes del sector, como IBM, Microsoft, Sun, BEA, Oracle y otros, además de Macromedia.

Existen muchos otros estándares que complementan o amplían estas tecnologías básicas.

Componentes de los servicios Web**:**

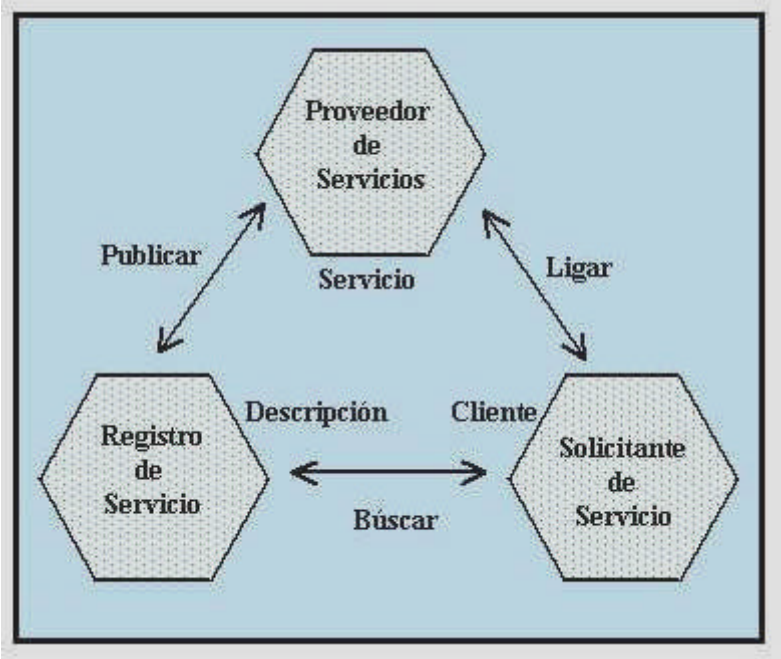
- **Servicio**. La aplicación es ofrecida para ser utilizada por solicitantes que cumplen los requisitos especificados por el proveedor de servicios. La implementación se realiza sobre una plataforma accesible en la red. El servicio se describe a través de un lenguaje de descripción de servicio. Tanto la descripción como las políticas de uso han sido publicadas de antemano en un registro.

- **Proveedor de Servicio**. Desde el punto de vista comercial, es quien presta el servicio. Desde el punto de vista de arquitectura, es la plataforma que provee el servicio.

- **Registro de Servicios**. Es un depósito de descripciones de servicios que puede ser consultado, donde los proveedores publican sus servicios y los solicitantes encuentran los servicios y los detalles para utilizarlos.

- **Solicitante de servicios**. Desde el punto de vista comercial, la empresa que requiere cierto servicio. Desde el punto de vista de la arquitectura, la aplicación o cliente que busca e invoca un servicio.

La siguiente figura muestra las relaciones operacionales entre los componentes:



Operaciones de Servicios Web:

- **Publicar/Cancelar**. Los proveedores de servicios publican la disponibilidad de su servicio comercial a uno o más Registros de servicios, o cancelan la publicación de su servicio.

- **Búsqueda**. Los solicitantes de servicios interactúan con uno o más Registros de servicios para descubrir un conjunto de servicios comerciales con los que pueden interactuar para encontrar una solución.

- **Ligar, Unir (Bind).** Los solicitantes de servicios negocian con los proveedores de servicios para acceder e invocar servicios comerciales.

Ejemplos de Servicios Web:

- *Passport* de Microsoft, servicio de autentificación.

- Servicio de verificación de crédito, que regresa historial de crédito cuando se proporciona el Registro Federal de Contribuyentes (RFC).

- Servicio de Bolsa de Valores, que regresa el valor o cotización de una acción de acuerdo a un símbolo designado.

- Servicio de compra, que permite a los sistemas de cómputo adquirir artículos de oficina cuando se proporciona un código de artículo y una cantidad.

Un servicio Web puede agregar otros servicios Web para proveer un conjunto de mayores características. Por ejemplo, un servicio Web pudiera proveer características de alto nivel en viajes, al organizar servicios Web de bajo nivel de renta de autos, boletos de avión, y hoteles.

Las aplicaciones futuras se realizarán con servicios Web que serán seleccionados dinámicamente en tiempo real basados en costo, calidad, y disponibilidad.

Entre las razones por las cuales los servicios Web serán importantes en la siguiente generación de sistemas distribuidos, están:

- *Interoperabilidad.* Cualquier servicio Web puede interactuar con otro servicio Web. El protocolo estándar *SOAP* permite que cualquier servicio pueda ser ofrecido o utilizado independientemente del lenguaje o ambiente en que se haya desarrollado.

- *Omnipresencia.* Los Servicios Web se comunican utilizando *HTTP* y *XML.* Cualquier dispositivo que trabaje con éstas tecnologías

puede ser huésped y acceder a los Servicios Web. Los Servicios Web serán utilizados en teléfonos, automóviles o aún en máquinas vendedoras de refrescos. Por ejemplo, una máquina de venta de refrescos puede comunicarse vía inalámbrica con el Servicio Web de un proveedor local y ordenar un pedido de suministro.

- *Barrera mínima de participación.* Los conceptos detrás de los servicios Web son fáciles de comprender y se ofrecen Herramientas de Desarrollo (*ToolKits*) por IBM, Sun Microsystems y la Organización de Apache, permiten a los desarrolladores crear e implementar rápidamente Servicios Web.

- *Apoyo de las Industrias.* Todas las compañías apoyan el protocolo *SOAP* y la tecnología derivada de los Servicios Web.

Razones para utilizar los Servicios Web XML

- Ofrecen un mecanismo sencillo para que las aplicaciones se comuniquen entre sí.

- Permiten que los componentes se compartan y que su funcionalidad trascienda a cualquier lugar y a cualquier persona.

- Son fáciles de distribuir y mantener.

- Abren la puerta a nuevas oportunidades empresariales.

- Ahorran tiempo y dinero al reducir la duración del ciclo de creación.

- Permiten volver a utilizar el código que otras personas hayan desarrollado sin tener que descargar, copiar o instalar algo.

Retos que deben vencer los Servicios Web

Para que los servicios Web tengan éxito, se requieren vencer algunos retos técnicos, entre los cuales se encuentran:

- *Descubrimiento.* ¿Cómo se anuncia un servicio Web para ser descubierto por otros servicios? ¿Qué sucede si el servicio es modificado o se cambia una vez que ha sido anunciado? Existen dos estándares nuevos que están diseñados para ello, el *WSDL* (Web Services Definition Language) y el *UDDI* (Universal Description, Discovery and Integration)

- *Confiabilidad*. Algunos sistemas huésped de servicios Web serán más confiables que otros. ¿Cómo se puede medir ésta confiabilidad y ser comunicada? ¿Qué sucede cuando un huésped de servicio temporalmente se sale de línea? ¿Cómo se localiza o utiliza un servicio alterno hospedado en otra compañía, o se pone en espera de que regrese el servicio original? ¿Cómo se sabe en que otra compañía se puede confiar?

- *Seguridad.* Algunos servicios Web estarán públicamente disponibles y con poca seguridad, pero la mayoría de los servicios comerciales utilizarán comunicaciones encriptadas con autentificación. Es probable que el protocolo HTTP sobre SSL proveerá la seguridad básica, pero los servicios individuales requerirán de un mayor nivel de especificación. ¿Cómo autentifica un servicio Web a sus usuarios? ¿Cómo distingue un servicio los niveles de privilegios entre los diversos usuarios?

- *Transacciones.* Los sistemas tradicionales de transacciones utilizan un método de compromiso de dos fases - se recolectan todos los recursos participantes y se aseguran éstos hasta que se lleva a cabo la transacción completa. Terminada la transacción se liberan los recursos. Este método puede funcionar bien en ambientes cerrados donde las transacciones son de corta duración, pero no trabaja bien en ambientes abiertos donde las transacciones pueden tomar horas o incluso días

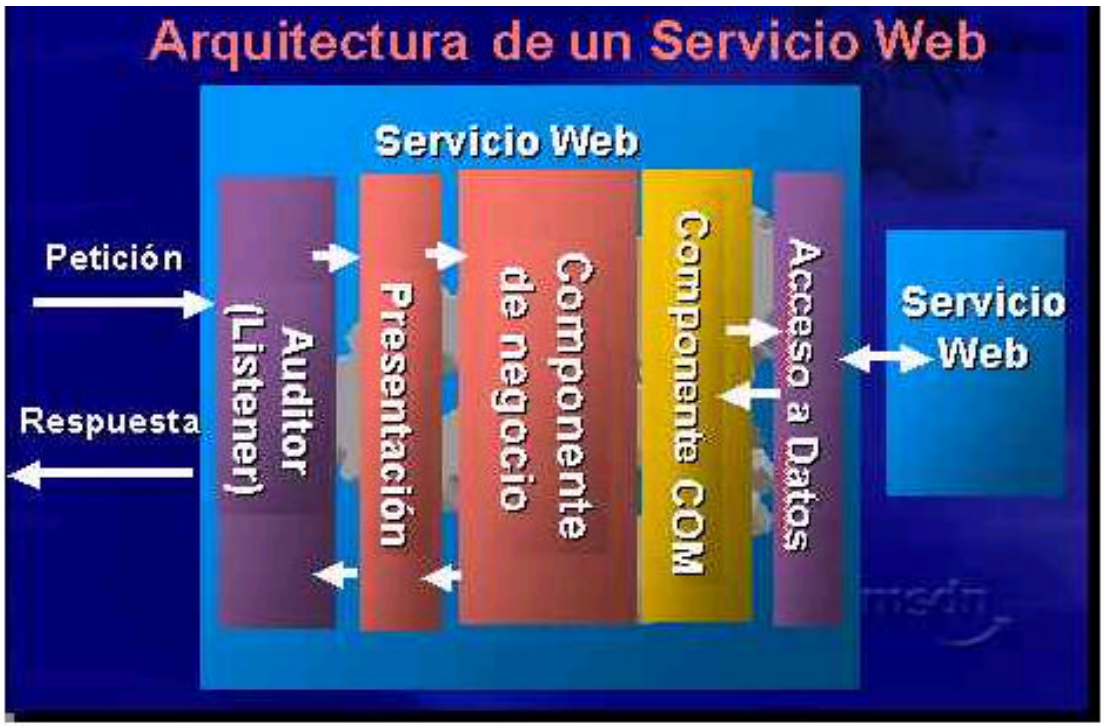
- *Administración*. ¿Qué tipos de mecanismos se requieren para administrar un sistema altamente distribuido? ¿Es posible delegar la administración de algunos servicios Web a otros?

- *Contabilidad.* ¿Cómo se define qué tanto tiempo puede un usuario acceder y ejecutar un servicio Web? ¿Cómo se pueden cobrar los servicios Web? ¿Cómo será la comercialización del servicio, bajo suscripción o pago por evento?

- *Pruebas.* ¿Cómo se puede depurar un servicio Web que proviene de diferentes compañías, que es hospedado en diferentes ambientes y en diversos sistemas operativos?

Arquitectura de un Servicio Web

¿Cómo se estructuran los servicios Web?. Para el mundo exterior, un Servicio Web es una aplicación que acepta peticiones de servicio, realiza algún proceso, y devuelve una respuesta. Por dentro suceden otras cosas. La siguiente figura muestra la arquitectura interna de un servicio Web.



Un Servicio Web contiene un Auditor (Listener) que está a la espera de peticiones. Cuando se recibe una petición, el auditor la reenvía a un componente que implementa la lógica de negocio requerida para la respuesta. El componente se puede diseñar para operar específicamente como un servicio Web o puede ser cualquier otro componente u objeto COM que el servicio quiera exponer al exterior. En este último caso, un desarrollador escribirá algún tipo de lógica que actuará como frente del servicio Web y enviará la petición al objeto COM mismo.

El objeto COM u otra lógica de negocio puede utilizar una base de datos u otro mecanismo de almacenamiento, al que se tiene acceso mediante una capa de acceso a Datos (Data Access layer). La capa de acceso a

datos se utiliza para leer información de una base de datos. Sin embargo, no hay nada que impida al servicio Web obtener sus datos de otro servicio Web - ya sea un servicio Web genérico de tipo componente, o uno de uso específico.

Estándares de Servicios Web

Los servicios Web se registran y se anuncian utilizando los siguientes servicios y protocolos. Muchos de estos estándares y otros están siendo desarrollados en el proyecto UDDI, un consorcio de industrias que coordina los esfuerzos de diseño y creación.

- **XML (eXtensible Markup Language)**, inició en Febrero de 1998 y ha revolucionado la forma en que se estructura, describe e intercambia información. Independientemente de múltiples formas en que se utiliza hoy en día el XML, todas las tecnologías de servicios Web se basan en XML. El diseño de XML se deriva de dos fuentes principales: SGML (Standard Generalized Markup Language) y HTML (HyperText Markup Language).

- **UDDI (Universal Description, Discovery and Integration)**, es un protocolo para describir los componentes disponibles de servicios Web. Este estándar permite a las empresas registrarse en un tipo de directorio sección amarilla de Internet que les ayuda a anunciar sus servicios, de tal forma que las compañías puedan encontrarse unas a otras y realizar transacciones en el Web. El proceso de registro y consultas se realiza utilizando mecanismos basados en XML y HTTP(S).

- **SOAP (Simple Object Access Protocol),** es un protocolo para iniciar las conversaciones con un servicio UDDI. El SOAP simplifica el acceso a los objetos permitiendo a las aplicaciones invocar métodos objeto o funciones, que residen en sistemas remotos. Una aplicación SOAP crea una petición en XML, proporcionando los datos necesarios para el método remoto así como la ubicación misma del objeto remoto.

- **WSDL (Web Service Description Language)**, es el estándar propuesto para la descripción de los servicios Web. Este estándar consiste en un lenguaje de definición de interfaz (IDL - Interface Definition Language) de servicio basado en XML que define la interfaz del servicio y sus características de implementación. El WSDL es apuntado en los registros UDDI y describe los mensajes SOAP que definen un servicio Web en particular.

Publicación y consumo de servicios Web

- Un proveedor parte de un desarrollo heredado o de un nuevo producto.
- El proveedor desarrolla y publica una interfase basada en un Servicio Web. Ésto incluye la publicación automática, en la Web, de un descriptor del servicio, en un formato estándar XML.
- Un consumidor construye automáticamente, a partir del descriptor del servicio, las clases “proxy” o “stub” que le permitirán acceder al Servicio Web desde sus aplicaciones.
- El consumidor desarrolla la parte cliente que consumirá el Servicio Web, usando las clases “proxy”.

TECNOLOGÍA .NET

Microsoft .NET es una plataforma para generar, ejecutar y experimentar la próxima generación de aplicaciones distribuidas. Abarca clientes, servidores y herramientas de desarrollo. Esta tecnología consta de:

- El modelo de programación .NET Framework, que permite a los desarrolladores crear aplicaciones Web, aplicaciones de cliente inteligente y servicios Web XML que exponen su funcionalidad mediante programación a través de una red utilizando protocolos estándar como SOAP, XML (Lenguaje de marcado extensible) y HTTP.
- Herramientas de desarrollo, como Visual Studio® .NET, que proporcionan un entorno de programación integrado de gran rapidez para la programación con .NET Framework.
- Un conjunto de servidores, incluidos Windows® 2000, SQL Server™ y BizTalk™ Server, que integra, ejecuta y administra aplicaciones y servicios Web XML.
- Software de cliente, como Windows XP, Microsoft Office XP, que permite a los desarrolladores ofrecer una experiencia de usuario convincente e intensa a través de una familia de dispositivos y productos existentes.

Una de las ideas centrales en las que descansa .NET es el concepto de "Web programable". De la misma forma que un componente instalado en un servidor y gestionado mediante **Transaction Server** o **Component Server** puede dar servicio a todas las máquinas del dominio de ese servidor, los sistemas construidos utilizando datos y servicios proveen de múltiples servicios Web. Los servicios Web suministran los elementos básicos para los sistemas, la Web dota de los sistemas de acceso a ellos, y los desarrolladores funden todos esos elementos en un modo útil.

Los servicios Web pueden ser específicos a una aplicación en concreto, sin embargo existe un gran número de servicios horizontales que la mayoría de las aplicaciones Web necesitan. Esto puede suministrarse mediante bloques de servicios Web. Un ejemplo es Microsoft Passport que pone a disposición una utilidad individualizada entre múltiples sitios Web.

También hay un mercado para servicios verticales de terceras compañías que implementen alguna funcionalidad común a partir de un segmento de negocio (finanzas y manufacturación, por ejemplo).

El consumidor de un servicio Web puede ser un navegador de Internet, otro dispositivo con conexión a Internet, o una aplicación. Incluso un servicio Web puede –en sí mismo- ser consumidor de otro servicio Web de la misma forma que un componente COM puede utilizar otro componente como parte de su implementación y funcionamiento.

A continuación se muestra un esquema que representa el funcionamiento de los Servicios Web, como bloques en la plataforma .NET:

Los estándares implicados

Desde un punto de vista lógico, se puede pensar en un Servicio Web como en un componente, o caja negra, que suministra algún servicio útil a los clientes, o consumidores. Lo mismo que DCOM es "COM de largo alcance", un Servicio Web puede asumirse de verdadero alcance real global. Sin embargo, a diferencia de DCOM, RMI, IIOP, o cualquier otro modelo de objetos común de uso específico, un consumidor accede a un

Servicio Web usando un protocolo estándar, aceptado, y bien conocido – HTTP – y un formato de datos basado en XML.

Un Servicio Web puede implementarse usando una gran variedad de lenguajes. En la actualidad, C++, Jscript, C#, y VB.NET están soportados, pero probablemente habrá muchos otros en el futuro. En lo que se refiere al consumidor, el lenguaje usado por el Servicio Web, es intranscendente. El punto de vista del consumidor del Servicio Web es el de una interfaz que expone un número de métodos bien definidos. Todo lo que necesita el consumidor es llamar a esos métodos usando los protocolos estándar de Internet, pasando parámetros en formato XML, y recibiendo respuestas también en formato XML.

La infraestructura de los servicios Web ASP.NET proporciona una API simple para dichos servicios basada en la asignación de mensajes SOAP a invocaciones de métodos. Para ello, ofrece un modelo de programación muy simple basado en la asignación de intercambios de mensajes SOAP a invocaciones de métodos individuales. Los clientes de los servicios Web ASP.NET no necesitan saber nada acerca de la plataforma, el modelo de objetos o el lenguaje de programación utilizados para crearlos. Ni siquiera los propios servicios tienen por qué saber nada acerca de los clientes que les envían mensajes. El único requisito es que ambas partes deben estar de acuerdo en cuanto al formato de los mensajes SOAP que se generan y se consumen, establecido en la definición de contrato del servicio Web que se expresa mediante esquemas WSDL y XML (XSD).

Las aplicaciones Web basadas en explorador, por el contrario, presentan un nivel de acoplamiento flexible y son extraordinariamente interoperables. Se comunican utilizando HTTP para intercambiar datos MIME en un gran número de formatos. Los servicios Web adaptan el modelo de programación Web tradicional para poder utilizarlo desde cualquier tipo de aplicaciones, no sólo las basadas en explorador. Intercambian mensajes SOAP utilizando HTTP y otros protocolos de Internet. Debido a que los servicios Web se basan en estándares industriales (entre los que se incluyen HTTP, XML, SOAP y WSDL) para exponer la funcionalidad de las aplicaciones en Internet, son independientes del lenguaje de programación, la plataforma y dispositivos utilizados.

Arquitectura del entorno .NET

.NET es la plataforma de Microsoft para servicios Web XML. Es la siguiente generación de software que conecta información, dispositivos y personas de una manera unificada y personalizada.

La Plataforma .NET permite la creación y uso de servicios de aplicaciones, procesos y sitios Web basados en XML, que compartan y combinen información y funcionalidad por su diseño, en cualquier plataforma o dispositivo inteligente.

La Plataforma .NET incluye una familia de productos integral, construida en estándares de la industria y de Internet, que provee servicios Web XML para cada aspecto del desarrollo, administración y uso.

Principales Componentes .NET

- Herramientas – para construir aplicaciones y servicios Web XML (.NET Framework y Visual Studio.NET)
- Servidores – sobre los que construir, proveer y desplegar esas aplicaciones y servicios
- Servicios – un conjunto central de servicios .NET ensamblables (servicios “HailStorm”)
- Software cliente – el software que provee dispositivos inteligentes, permitiendo a los usuarios interactuar y experimentar la plataforma .NET.

.Net Framework

Uno de los aspectos más interesantes de Microsoft® .NET Framework es la incorporación de los servicios Web XML (Lenguaje de marcado extensible). Los servicios Web permiten la comunicación y el intercambio de información entre aplicaciones a través de procesos basados en estándares. La implementación de servicios Web por parte de Microsoft conlleva numerosas ventajas inherentes. El intercambio de información se realiza en formato XML, utilizando el protocolo de transferencia de hipertexto (HTTP) y el protocolo de control de transporte/protocolo Internet (TCP/IP), por lo que los servicios Web pueden utilizar la infraestructura existente e implementarse en diferentes entornos fácilmente.

El .NET Framework es un componente del sistema operativo Microsoft Windows® que proporciona el modelo de programación para crear, implementar y utilizar aplicaciones basadas en Web, aplicaciones para dispositivos inteligentes y servicios Web XML.

.NET Framework es el modelo de programación de la plataforma .NET. Administra gran parte de los detalles de infraestructura, permitiendo a los desarrolladores centrarse en escribir el código de la lógica empresarial para sus aplicaciones. .NET Framework incluye el lenguaje común en tiempo de ejecución (CLR, Common Language Runtime) y bibliotecas de clases.

Lenguaje Común en Tiempo de Ejecución (CLR)

El CLR es responsable de los servicios en tiempo de ejecución y de la integración de lenguajes, la aplicación de seguridad y la administración de la memoria, los procesos y los subprocesos. Además, juega un papel importante en tiempo de desarrollo, puesto que características como la administración de la duración, la aplicación de nombres de tipos seguros, el control de excepciones entre lenguajes, la creación de enlaces dinámicos, etc., reducen la cantidad de código que debe escribir un desarrollador para convertir lógica empresarial en un componente reutilizable.

Bibliotecas de clases

Las clases base proporcionan funcionalidad estándar como funciones de entrada y salida, manipulación de cadenas, administración de la seguridad, comunicaciones de red, administración de subprocesos y de texto, características de diseño de la interfaz de usuario y otras funciones. Las clases de datos de Microsoft ADO.NET admiten la administración de datos permanentes e incluyen clases SQL para manipular almacenes de datos permanentes a través de una interfaz SQL estándar. Las clases XML permiten la manipulación de datos XML, así como la búsqueda y traducción de XML. Las clases de Microsoft ASP.NET admiten el desarrollo de aplicaciones Web y servicios Web XML.

Ventajas que presenta el .NET Framework

.NET Framework simplifica la tarea de exponer código .NET como servicio Web. Una de las razones para tal sencillez es que .NET Framework incluye especificaciones para transformar tipos de datos complejos de .NET en XML (serialización) y para el proceso inverso (deserialización). Algunos de los beneficios de utilizar esta herramienta se describen a continuación:

Confiabilidad mejorada

El .NET Framework toma los principales logros originalmente hechos en Windows 2000 y los lleva a nuevos niveles. Con maneras avanzadas de monitorear el estado de las aplicaciones que se ejecutan. Las aplicaciones creadas para el .NET Framework se mantienen en ejecución por más tiempo que antes.

Mejor desempeño

Gracias en parte a la avanzada compilación y a las técnicas de caché, las aplicaciones de servidor nunca habían sido más rápidas que las creadas para el .NET Framework y con tecnología Microsoft ASP.NET. Los usuarios que han migrado de ASP a ASP.NET comprueban el incremento en la velocidad en un rango de 300 a 500 por ciento.

Productividad del desarrollador

Los desarrolladores de todo tipo encuentran que pueden familiarizarse rápidamente con el .NET Framework. Lo intuitivo de su modelo de programación, la cantidad de código proporcionado en las bibliotecas de clases y la cantidad de trabajo hecho automáticamente por el .NET Framework en áreas como la administración de memoria ha permitido a los desarrolladores del .NET Framework aumentar su productividad a niveles muy altos.

Seguridad

La tecnología de seguridad de acceso al código del .NET Framework fue diseñada para los ambientes actuales de Internet. El .NET Framework puede recolectar evidencia acerca de dónde se origina una aplicación, quién la creó, cuál es su firma digital y cuál es su función. El ambiente de ejecución del .NET Framework puede combinar esa evidencia con

políticas de seguridad para decidir si una aplicación puede acceder a determinado recurso. Hasta puede "negociar" con la aplicación, por ejemplo, negando el permiso de escritura a un directorio con protección y permitiendo a la aplicación seleccionar si desea ejecutarse, aunque se le haya negado el permiso.

Integración con sistemas existentes

La tecnología del .NET Framework para la operación con objetos COM genera una envoltura en sus componentes COM existentes y aplicaciones basadas en Windows (como Microsoft Office), permitiendo que se programe para ellos como si hubieran estado diseñados originalmente para el .NET Framework. Las aplicaciones creadas usando el .NET Framework pueden conectarse con sistemas y aplicaciones existentes, sin importar su plataforma base, por medio de servicios Web XML u otros medios específicos de la aplicación. Finalmente, la herramienta Visual Basic Upgrade Tool, disponible con Visual Studio .NET, y la herramienta Java Language Conversion Assistant, ayudan a convertir código existente de Microsoft Visual Basic® 6.0 y Microsoft Visual J++® para que pueda ejecutarse en el .NET Framework.

Facilidad de implementación

El .NET Framework tiene características para hacer más fácil la implementación, ejecución y administración de aplicaciones. El aislamiento de aplicaciones y el control de versiones automático de componentes pueden ayudar a prevenir conflictos de versiones. Las aplicaciones creadas usando el .NET Framework pueden ser implementadas en un cliente o un servidor simplemente copiando el directorio de la aplicación al equipo, sin realizar cambios en el registro. Además, las aplicaciones de Windows pueden ser actualizadas simplemente copiando los componentes necesarios al servidor Web que puede ser accedido por usuarios finales.

Soporte nativo para servicios Web XML

El .NET Framework fue diseñado desde un principio con soporte para servicios Web XML, un modelo para computación distribuida en múltiples ambientes basados en protocolos estándar como XML, SOAP y HTTP. Los

servicios Web XML pueden ser usados para integrar aplicaciones que se ejecutan en diferentes plataformas, o para proporcionar software como un servicio. Con el .NET Framework, una aplicación puede ser transformada en un servicio Web XML con solamente una simple línea de código.

Soporte para más de 20 lenguajes de programación

El .NET Framework tiene soporte para la integración de más de 20 lenguajes de programación de una manera que antes solamente podía imaginarse, permitiendo que los desarrolladores seleccionen el lenguaje de programación adecuado para la tarea que desean realizar. Todos los lenguajes de programación utilizan bibliotecas de clases únicas y extensibles. Los componentes escritos en diferentes lenguajes soportados por el .NET Framework pueden interactuar entre ellos, sin necesidad de utilizar COM.

Acceso flexible a datos

La tecnología del .NET Framework, Microsoft ADO.NET, está diseñada para el estilo Web actual de acceder datos. Usando ADO.NET, los desarrolladores tienen la opción de trabajar con un caché de los datos requeridos basado en XML, en lugar de manipular directamente la base de datos. Esta aproximación al acceso a datos libera conexiones a bases de datos, lo que resulta en una escalabilidad significativamente más grande.

Requisitos para el consumo de servicios Web en .NET

Para consumir un servicio Web, es necesario disponer de varios elementos de forma adecuada. Al utilizar Microsoft Visual Studio® .NET, estos elementos se colocarán correctamente de forma automática.

- El kit de desarrollo de software Microsoft .NET Framework SDK
- Los Servicios del servidor de información de Internet (IIS) de Microsoft
- Un proxy de servicios Web
- Un formulario Web de ASP.NET

Microsoft .NET Framework SDK

El .NET Framework constituye el modelo de programación de la plataforma Microsoft .NET que permite crear servicios y aplicaciones Web eficaces. .NET Framework facilita la creación de páginas ASP.NET que se necesitan para consumir servicios Web. El .NET Framework SDK puede encontrarse en el sitio de descarga de .NET Framework SDK.

Es posible obtener el ASP .NET de dos formas distintas, a través del .NET Framework SDK o a través de Microsoft ASP .NET Premium Edition.

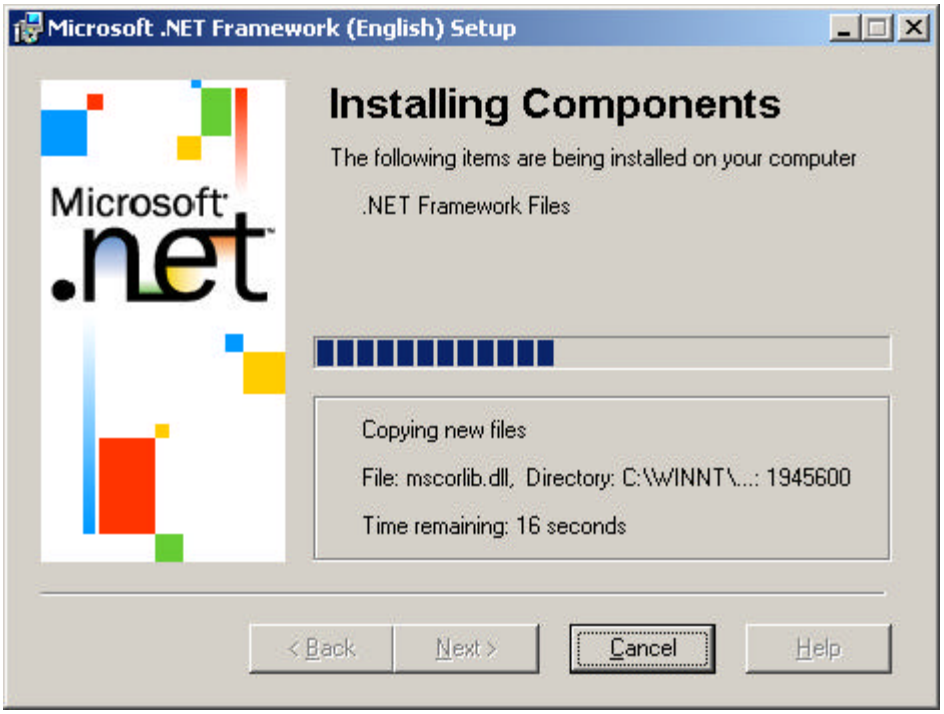
El .NET Framework SDK se puede obtener de la dirección de Microsoft http://msdn.microsoft.com/downloads o bien si se dispone de Visual Studio .NET, el .NET Framework SDK se encuentra el disco compacto llamado Actualización de Componentes de Windows (Windows Component Update).

Microsoft ASP .NET Premium Edition también se puede obtener de la misma dirección, pero es más recomendable instalar el .NET Framework SDK.

Instalación de ASP.NET

Una vez que se tienen los requisitos necesarios para instalar el .NET Framework SDK, es decir, Internet Explorer 6 y el Service Pack 2 de Windows 2000, se puede proceder con su instalación. Si se realiza la instalación de Visual Studio .NET y se utiliza el disco de Actualización de Componentes de Windows, los requisitos necesarios del .NET Framework SDK se instalarán previamente.

La instalación del .NET Framework SDK es muy sencilla, sólo se debe ejecutar el programa de instalación. El ejecutable mostrará un asistente que facilitará el proceso de instalación. En la figura que se muestra a continuación se puede ver la pantalla que aparece en la fase de instalación del .NET Framework.



Al realizarse la instalación del .NET Framework SDK se indicará que de forma opcional se puede instalar Microsoft Data Access Components 2.7. También preguntará si se desea instalar Microsoft Windows Installer 2.0, a esta pregunta se debe contestar afirmativamente.

Una vez instalado el .NET Framework SDK, resulta útil instalar los tutoriales que vienen con el paquete de desarrollo, éstos son denominados tutoriales rápidos (QuickStart Tutorials).

Para acceder al tutorial de ASP .NET se debe escribir la siguiente URL http://localhost/quickstart/ASPPlus/, esto permitirá el acceso a una página como la de la siguiente figura:

Se puede decir que para desarrollar páginas ASP .NET en algún equipo se debe instalar lo siguiente:

1. El servidor Web (Internet Information Server).

2. El navegador Internet Explorer 5.5 o superior y el Service Pack 2 de Windows 2000.

3. El .NET Framework SDK.

Servidor de Información de Internet de Microsoft

En primer lugar, se necesita un servidor Web compatible con ASP.NET. El servidor se puede utilizar tanto en Windows 2000 como en Windows XP.

Si se instala IIS tras instalar .NET Framework SDK, también se deberá instalar la compatibilidad ASP.NET.

Instalación de IIS en Windows XP Profesional

El servidor de información de Internet (IIS) es el servidor de páginas Web avanzado de la plataforma Windows. Se distribuye gratuitamente junto con las versiones de Windows basadas en NT: Windows 2000 Profesional, Windows 2000 Server y Windows XP en sus versiones Profesional y Server.

Estas normas de instalación son aplicables, a nivel general, en las distintas versiones de los sistemas operativos comentados anteriomente.

Nota: En Windows 95, 98, las versiones Home, de Windows XP, y ME de Windows 2000, no se admite la instalación de IIS. En su lugar es posible instalar el Personal Web Server.

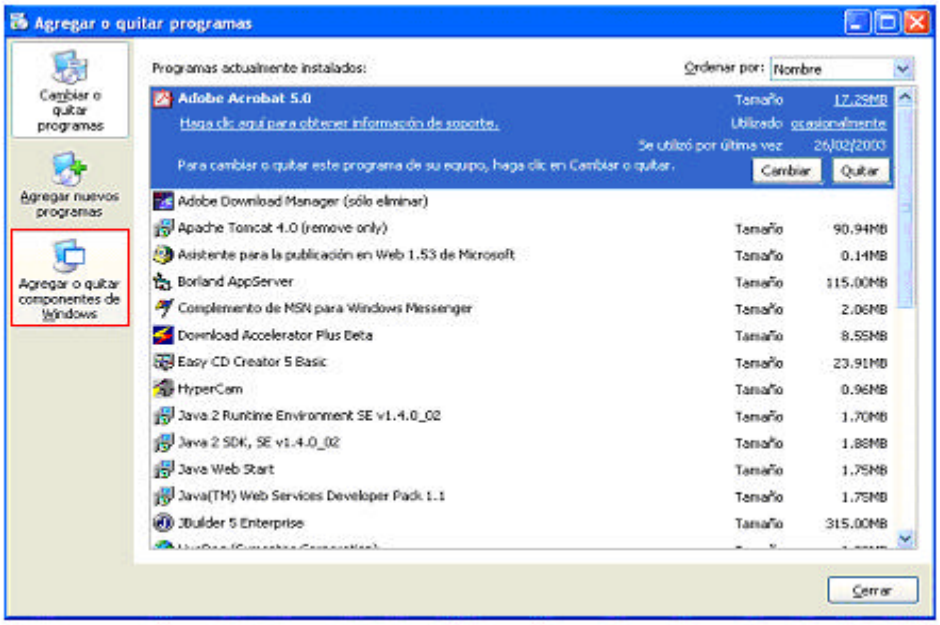
Proceso de instalación del IIS

Agregar componentes adicionales de Windows

IIS se puede encontrar en el propio CD de instalación de Windows XP Profesional. Se debe acceder a la opción de "Instalar componentes opcionales de Windows" para poder cargarlo en el sistema. Para ello se tienen dos opciones:

1) Insertar el CD de instalación de Windows y en la ventana de autoarranque que se muestra, seleccionar la opción: "Instalar componentes opcionales de Windows"

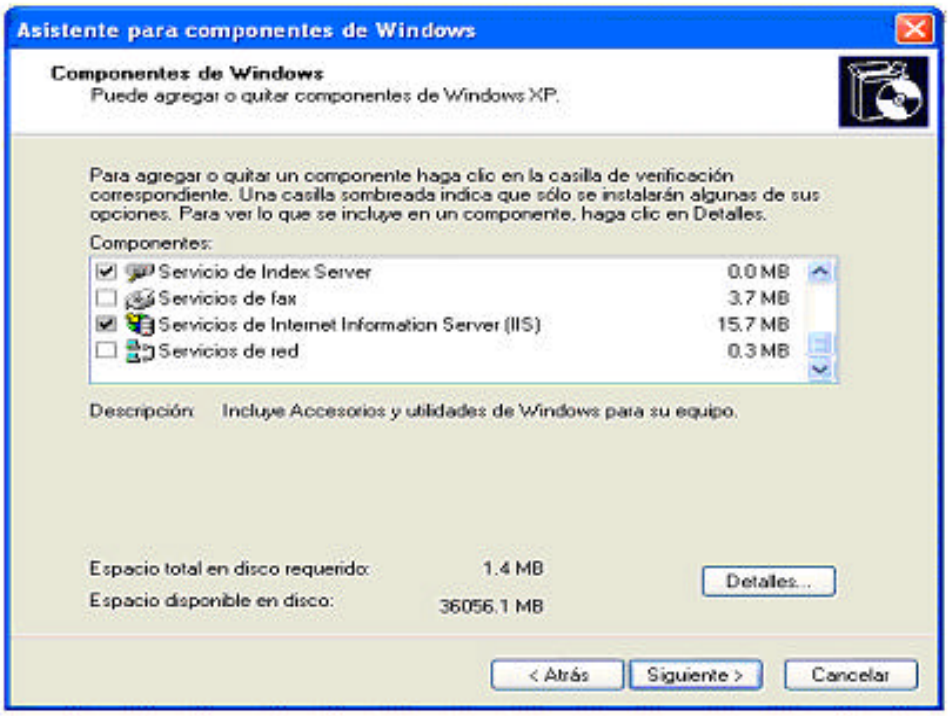
2) En el Panel de control, seleccionar la opción "Agregar o quitar programas" y en la ventana que aparece, pulsar sobre el icono de la izquierda marcado como "agregar o quitar componentes de Windows".



Ahora se muestra la ventana para seleccionar los componentes adicionales de Windows que hay disponibles. En la lista, se debe marcar la opción "Servicios de Internet Information Server (IIS)". Por defecto se seleccionan algunos componentes, entre ellos la instalación de IIS. También es posible elegir qué componentes se desean instalar pulsando el botón marcado como "Detalles". Entre los componentes posibles se encuentran las extensiones de Frontpage, documentación, servicios adicionales de IIS, un servidor de FTP (para la transferencia de ficheros con el servidor por FTP), incluso uno de SMTP (para el envío de correos electrónicos).

Si no se sabe qué componentes instalar se puede dejar las opciones como aparecen en un principio, para la mayoría de los casos serán válidas. Sólo un detalle: puede ser adecuado no instalar las extensiones de Frontpage en caso de que no se piensen utilizar.

La siguiente figura muestra la pantalla que aparece para seleccionar al servidor de información de Internet.



Una vez instalados los componentes deseados, se debe pulsar el botón "Siguiente" para iniciar la instalación, que se alargará unos minutos.

Acceder al servidor Web

Es posible acceder al servidor Web para comprobar si se ha instalado correctamente el IIS. Para ello simplemente se debe escribir en el explorador de Internet: http://localhost; debe aparecer una página Web informando que IIS está correctamente instalado.

Además, aparecerá la documentación de IIS en una ventana emergente, si es que fue instalada. Ésta se muestra a continuación:

Sitio Web predeterminado en IIS

Lo que se muestra cuando se accede a http://localhost es el sitio Web predeterminado, que se guarda en el disco duro de la computadora, concretamente en la carpeta C:\Inetpub\wwwroot.

Si se accede a dicha carpeta desde Mi PC se pueden ver los archivos que se están sirviendo como sitio Web predeterminado. Se encontrará, entre otros archivos uno llamado "iisstart.asp" que es el que se pone en marcha al acceder a este directorio.

Colocar páginas propias

Lo lógico ahora es que se desee colocar páginas Web propias para que las sirva el IIS. Si el sitio Web es bastante simple se podría colocar todos los archivos dentro de la carpeta del sitio Web predeterminado.

Por ejemplo, para hacer una prueba, se podría colocar un archivo llamado "hola.asp" en la carpeta C:\Inetpub\wwwroot. Para acceder a este archivo desde el explorador se debería escribir la dirección http://localhost/hola.asp.

Atención: No se debe acceder al archivo utilizando una ruta como esta: C:\Inetpub\wwwroot\hola.asp, pues de esa manera no se estaría pasando a través del servidor Web y la página ASP no se ejecutaría.

Proxy de servicios Web

Los clientes y los servicios Web se comunican a través de mensajes de SOAP (Simple Object Access Protocol). El proxy de servicios Web se encarga de la formación de estos mensajes y de su envío posterior a través de la red, lo que facilita considerablemente el consumo de un servicio Web, ya que no se tendrá que asignar parámetros a elementos XML. El proxy de servicios Web consta de una biblioteca de vínculos dinámicos (DLL) en el servidor Web local. La utilidad WSDL.exe, distribuida con .NET Framework SDK, permite generar un archivo de código fuente de Microsoft Visual Basic® .NET o C# y, a continuación, utilizarlo para compilar el proxy de servicios Web.

El paso más importante a la hora de crear consumidores de servicios Web es crear una clase Proxy que sea el medio entre el cliente y el servicio Web.

Hay dos formas de crear una clase Proxy.

- Usando la herramienta WebServiceUtil que es incluida en el marco SDK de Microsoft .Net. Teniendo como ventaja que no requiere una copia de Visual Studio.Net.

- Usando la opción de incluir una referencia Web y dejar que Visual Studio .Net cree la clase proxy automáticamente.

Crear la clase proxy con Visual Studio .Net:

Visual Studio.Net tiene una característica llamada “Add Web Referente” que automatiza todos los mecanismos necesarios para crear la clase proxy. Cuando se añade la referencia Web en el servicio Web del proyecto, Visual Studio.Net automáticamente genera una clase proxy con las interfaces Web.

Para añadir una referencia Web, hacer clic con el botón derecho sobre el proyecto en Solution Explorer y seleccionar “Add Web reference”. En el dialogo de “add Web reference”, introducir la ruta del servidor Http://servidor/, seleccionar el servicio Web que se desea y dar clic sobre “add reference”.

Cuando se da clic sobre “add Reference”, Visual Studio.Net crea una carpeta con el nombre del servicio debajo de la carpeta Web references.

Por defecto, el nombre del servidor es considerado como el namespace para la clase proxy local. Para asignar diferente namespace para el proxy, renombrar el nombre de la carpeta del servidor con el nuevo nombre. Ahora, importar el namespace del proxy dentro del proyecto. Una vez que se tiene el namespace importado dentro de la aplicación actual, los métodos accesibles del Web se pueden invocar transparentemente según se explicó en el paso anterior.

Preparación de servicios Web para su uso

Asumiendo que ha instalado Microsoft .NET Framework SDK y los Servicios del servidor de información de Internet de Microsoft, lo siguiente es analizar el consumo del servicio Web. Estos son los pasos necesarios para consumir algún servicio Web:

1. Examen del servicio Web.
2. Generación de un archivo de código fuente.
3. Compilación del proxy de servicios Web.
4. Diseño de la interfaz ASP.NET.
5. Conexión del servicio Web.
6. Copia del proxy.
7. Comprobación de la aplicación ASP.NET.

Para generar el archivo de código fuente se utilizan los siguientes modificadores con WSDL.exe:

/l - Este modificador especifica el lenguaje del archivo de código fuente.

/o - Este modificador especifica el nombre del archivo de salida. La extensión de un archivo de código fuente de Visual Basic es .vb. La de un archivo de código fuente de C# es .cs.

Nota: Si se desea obtener más información sobre WSDL, se debe utilizar el modificador /?.

La URL que se pasa al WSDL es la del contrato del servicio Web. Este contrato es un documento en el que se describe el tipo de información que el servicio Web espera que proporcione, así como el tipo de información que devolverá el servicio. La URL del contrato del servicio Web es simplemente la URL del punto de acceso del servicio Web con ?wsdl agregado al final.

Compilación del proxy de servicios Web

Una vez creado el archivo de código fuente de Visual Basic .NET, es preciso compilarlo en un proxy de servicios Web. Para ello, se utiliza el compilador de Visual Basic .NET que se distribuye con el .NET Framework SDK.

Para compilar la clase proxy, se debe escribir lo siguiente en la interfaz de comandos que se ha abierto:

Wsdl http://localhost/nombre_servicio.asmx

Para generar la clase proxy se utilizan los siguientes modificadores:

/t - Este modificador especifica el tipo de ensamblado que se va a crear. En este caso, se compilará un archivo DLL, por lo que se debe especificar el tipo de ensamblado como library.

/out - Este modificador especifica el nombre del archivo de salida. Debido a que se especifica el tipo de ensamblado como library, la extensión del archivo de salida debe ser .dll.

/r - Este modificador especifica las referencias del ensamblado. En este caso, se hace referencia a tres espacios de nombres de Microsoft .NET necesarios para todos los clientes proxy de servicios Web. Se trata de los espacios de nombres System, System.XML y System.Web.Services.

Una vez que el compilador finalice la compilación de la clase proxy, se deberá disponer de un archivo con extensión .dll en el directorio en la unidad actual. Se trata del cliente proxy.

Diseño de la interfaz ASP.NET

Una vez que se tiene una idea general acerca de la interfaz que se desea crear y también se ha compilado con éxito una clase proxy de servicios Web, el siguiente paso consiste en crear un formulario ASP.NET para interactuar con el servicio Web creado.

Gracias a Microsoft Visual Studio .NET, el diseño de un formulario Web resulta bastante sencillo. Basta con arrastrar y colocar los elementos del formulario deseados y configurar las propiedades de dichos elementos en la interfaz de usuario.

Creación del formulario Web

El primer paso consiste en crear el código del propio formulario Web, el cual es similar al código de los formularios de lenguaje de marcado de hipertexto (HTML). La única diferencia es la presencia del atributo runat como se muestra a continuación:

```
<form runat="server">
</form>
```

El valor de este atributo es server, lo que indica que este formulario es un control de servidor de ASP.NET.

Conexión del servicio Web

La conexión del servicio Web se refiere al procedimiento del lado del servidor responsable de lo siguiente:

- Creación de una instancia de la clase de cliente proxy.
- Llamada a las funciones del servicio, incluyendo el pase de parámetros.
- Definición del texto del control existente para que se muestre la respuesta recibida del servicio Web.

Todas las clases de proxy tienen un nombre asociado, que será el que se utilice a la hora de crear una nueva instancia de la clase.

Copia de la clase de proxy

El último paso que se debe realizar antes de probar la página ASP.NET consiste en copiar la clase de proxy compilada en la ubicación adecuada de la aplicación ASP.NET. Para esto:

1. Crear una carpeta nueva llamada **bin**.
2. Importar a esta carpeta el archivo con extensión .dll compilado anteriormente.

Comprobación de la aplicación ASP.NET

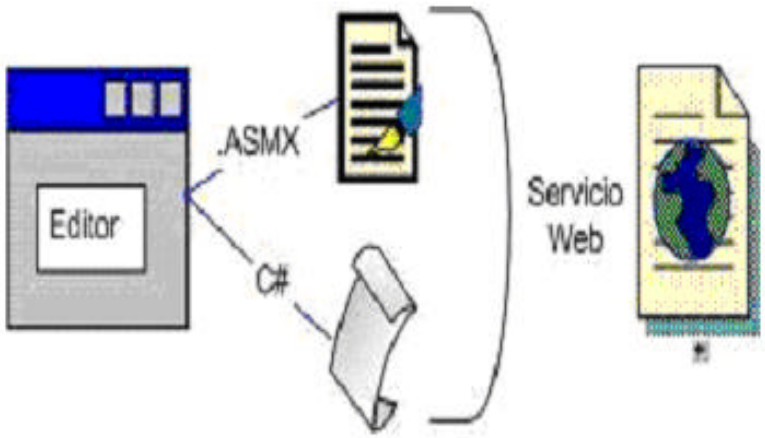
Para probar la aplicación ASP.NET (página con extensión aspx) se debe invocar la pagina aspx en el explorador escribiendo para ello http://localhost/nombre_pagina.aspx.

CONSTRUCCIÓN DE SERVICIOS WEB

Las fases de la puesta en marcha de un Servicio Web son:

A) Por parte del constructor del Servicio

1) **Creación del servicio**: es posible utilizar cualquier lenguaje para crear el servicio. La condición para hacerlo disponible para los demás (publicarlo), incluye la creación de descripciones en lenguajes estándar, como se muestra en el punto siguiente. La siguiente figura muestra la creación de un servicio Web utilizando el lenguaje de programación C#.



Etapas iniciales de la construcción de un servicio Web

2) **Puesta en marcha del servicio** (publicación del mismo): Supone la edición y publicación de un fichero de descripción de servicios que permita a los usuarios potenciales del mismo, conocer qué funciones se han expuesto para su utilización pública. El lenguaje utilizado en la descripción de un servicio Web es un estándar oficial de la W3C (World Wide Web Consortium), llamado **WSDL** (*Web Service Description Language*), que define una gramática XML que debe utilizarse a la hora de describir un servicio Web y permite describir todos los datos necesarios para utilizar el servicio. En otras palabras, para que un cliente utilice un servicio Web, necesita saber qué métodos (mensajes) es capaz de comprender el servicio, qué parámetros admite y qué tipo de valores devuelve.

Definición del estándar WSDL

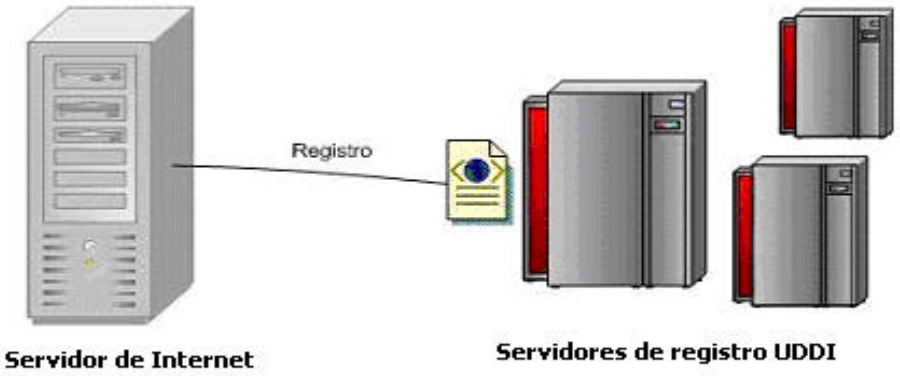
Un documento WSDL define servicios como una colección de puntos finales de una red (network endpoints) o puertos. En WSDL, la definición abstracta de los puertos y mensajes se separa de su distribución física real o de sus formatos de datos. Esto permite la reutilización de definiciones abstractas: los mensajes que son definiciones abstractas de los datos que se intercambian, y los tipos de puertos (port types), que son colecciones abstractas de las operaciones. El protocolo concreto y la especificación del formato de datos para un tipo de puerto concreto constituye un enlace de datos reutilizable (reusable binding). Un puerto se define asociando una dirección de red a uno de estos enlaces reutilizables, y una colección de puertos define un servicio. Por tanto, un documento WSDL utiliza los siguientes elementos en la definición de servicios:

- **Tipos**– un contenedor para definiciones de tipos de datos que utilicen algún sistema de tipos (como XSD).
- **Mensaje**– Una definición abstracta de la información comunicada incluyendo los tipos de datos.
- **Operación**– Una descripción abstracta de una acción soportada por el servicio.
- **Port Type**–Un conjunto de operaciones abstractas soportadas por uno o más puertos.
- **Binding**– Un protocolo concreto y una especificación de formato de datos para un tipo de puerto.
- **Port**– Un end-point simple definido como una combinación de un binding y una dirección de red.
- **Servicio**– Una colección de end-points relacionados

En la figura que se muestra a continuación se pueden observar los elementos necesarios para la publicación de un servicio Web.

Publicación y habilitación del Servicio Web

3) **Registro del Servicio en un buscador universal especializado**. Se trata, no de publicar el servicio en un servidor Web local, sino de registrarlo universalmente de forma que cualquiera pueda saber de su existencia. Aquí es posible basarse en un estándar: **UDDI** *(Universal Description, Discovery and Integration). Se puede* ver a **UDDI** como una especie de *Google para Servicios Web*, que permite saber qué es lo que hay disponible en Internet o en una Intranet. En la siguiente figura se puede apreciar lo mencionado anteriormente.



Registro de los servicios web en los repositorios UDDI

El protocolo **DISCOVERY**, también estándar, propuesto para el descubrimiento de servicios se basa en una gramática XML para describir la ubicación de un servicio. Lo que sucede es que para ello, se necesita saber en qué sitio Web hay servicios publicados, por lo que la propuesta UDDI aparece como una solución más amplia.

B) Por parte del consumidor del Servicio

1) Éste, por su parte, debe, primero, **descubrir** el servicio (literalmente, saber que existe, dónde está y cómo está construido) y posteriormente diseñar e implantar un mecanismo de programación que lo ponga en marcha. Para ello, deberá obtener una descripción del servicio, partiendo del sitio Web donde se encuentre.

2) Una vez localizado el servicio, obtener su descripción. Ésta consiste en el fichero WSDL que contiene todas las características de utilización.

3) Con la descripción del servicio, es posible ponerlo en marcha. Para ello, se necesita un *puente* que permita conectar alguna página con **algo** que sea capaz de invocar el servicio y pasarle peticiones en un formato adecuado. Ésta es la parte más compleja, y por ello .NET ha hecho un esfuerzo considerable para ayudar al programador en esta tarea.

Tanto las herramientas de .NET Framework, como el propio Visual Studio .NET disponen de herramientas para generar un fichero en el lenguaje .NET que se le pida a partir de la descripción WSDL de un servicio Web. Ese fichero de código fuente, invocará las funciones de librería de .NET necesarias para comunicarse con el servicio y recibir la respuesta de éste. Una vez recibida dicha respuesta, sólo se deben enlazar los datos con algún mecanismo de visualización de la página Web, y estará en marcha el servicio.

El proceso de comunicación mediante SOAP

La comunicación de una página Web con un servidor que albergue un servicio Web, se produce a través del protocolo estándar de comunicaciones HTTP -para poder comunicarse a través de cortafuegos de sistema- y la información que se envía utiliza en su formato un estándar basado en otra gramática XML: **SOAP** (***Simple Object Access Protocol***). SOAP define los paquetes de envío y recepción de datos

dentro de unos **sobres** (literalmente, ***envelopes***, en inglés), donde usando el lenguaje de marcas definido por SOAP se establece quién es el destinatario de la petición, la función a la que se desea llamar, los argumentos que se le pasan y los tipos que presentan y en general toda la información necesaria, para que el servidor destino sea capaz de poner en marcha la llamada al servicio.

Obviamente, tanto el emisor como el receptor deben haber instalado previamente las librerías de SOAP y el mecanismo necesario para permanecer a la escucha de peticiones SOAP, poner en marcha el proceso de la petición, y generar el fichero de respuesta. Para ello, no es necesario tener instalado .NET, sino solamente el paquete de SOAP que es de libre distribución y está disponible para las diferentes plataformas de ejecución.

DESARROLLO DE SERVICIOS WEB EN LA PLATAFORMA .NET

Visual Studio .NET acelera el proceso de desarrollo de aplicaciones haciendo que la programación .NET sea simple. Oculta detalles repetitivos de configuración y mejora la productividad. Sin embargo, existen casos en los que no se utiliza el Visual Studio .NET y se necesita programar. Estos casos son cuando se desea aprender la programación del .NET Framework sin tener acceso a Visual Studio .NET o cuando se desea saber qué es lo que ocurre realmente con el .Net Framework.

Existen muchos ejemplos disponibles en Internet para escribir servicios Web utilizando el Visual Studio .NET. Sin embargo, existen muy pocos ejemplos de cómo escribir servicios Web utilizando sólo el SDK Framework. A continuación se muestra un ejemplo para desarrollar servicios Web utilizando únicamente el .NET SDK.

En este ejemplo se escribirá y se publicará un servicio Web simple. También se escribirán dos clientes para el servicio Web: un cliente basado en Web (Aplicación ASP .NET) y otro cliente basado en una aplicación de Windows.

EJEMPLOS DE SERVICIOS WEB

SERVICIO WEB: FIRSTSERVICE

Lenguaje utilizado: C#.

Los servicios Web .NET utilizan la extensión .asmx y deben guardarse dentro de un directorio virtual en el servidor de información de Internet para que puedan ser utilizados por las aplicaciones. Éste es un estándar para una plantilla de servicio Web.

Nótese que un método expuesto como servicio Web tiene el atributo WebMethod y que la clase que utiliza (FirstService, en este ejemplo) debe ser pública, esto con la finalidad de que esté disponible para cualquier consumidor de servicios Web.

Descripción del servicio Web “FirstService”:

El siguiente servicio Web expone dos métodos (“Add” y “SayHello”). El método Add suma dos números y el método SayHello simplemente despliega el mensaje “Hello World”. El código del servicio se muestra a continuación:

FirstService.asmx

```
<%@ WebService language="C#" class="FirstService" %>

using System;
using System.Web.Services;
using System.Xml.Serialization;

[WebService(Namespace="http://localhost/MyWebServices/")]
public class FirstService : WebService
{
    [WebMethod]
    public int Add(int a, int b){
        return a + b;
    }

    [WebMethod]
    public String SayHello(){
        return "Hello World";
    }
}
```

Para probar un servicio Web, éste debe ser publicado ya sea en una Intranet o en Internet. Este ejemplo será publicado en el servidor de información de Internet en una máquina local. Se iniciará con la configuración del IIS.

- Abrir Inicio->Programas->Panel de control->Herramientas administrativas->Servicios de Administración de Internet.
- Expandir y dar click con el botón derecho del ratón sobre [Sitio Web por defecto]; Seleccionar Nuevo -> Directorio Virtual.
- Se abrirá el asistente de creación de directorio virtual. Dar clic en siguiente.
- Se abrirá una pantalla de "Alias para el directorio virtual". Escribir el nombre para el directorio virtual —por ejemplo, MyWebServices—y dar clic en siguiente.
- Se abrirá una pantalla de "Contenido de directorio de sitio Web". Aquí, se debe introducir la ruta para el directorio virtual —por ejemplo, c:\MyWebServices—y después dar clic en siguiente.
- Se abrirá una pantalla de "Permiso de acceso". Dar clic en terminar para finalizar la configuración del IIS.

Para probar el servicio Web, copiar FirstService.asmx al directorio virtual del IIS creado anteriormente (C:\MyWebServices). Abrir el servicio Web en el explorador (http://localhost/MyWebServices/FirstService.asmx). Ésto debe abrir la página del servicio Web. La página tiene enlaces a los métodos expuestos como servicios Web en la aplicación.

Cliente ASP.NET

La aplicación del cliente basado en Web debe ser guardada en el directorio virtual del servicio Web.

Esta aplicación tiene dos campos de texto que son utilizados para que el usuario introduzca los números que se sumarán. Tiene también un botón llamado ejecutar que cuando se presiona click en él se llaman los servicios Web “Add” y “SayHello”.

A continuación se lista el código de la aplicación **Firstconsumer.aspx:**

```
<%@ Page Language="C#" %>
<script runat="server">
void runSrvice_Click(Object sender, EventArgs e) {
   FirstService mySvc = new FirstService();
   Label1.Text = mySvc.SayHello();
   Label2.Text = mySvc.Add(Int32.Parse(txtNum1.Text),
   Int32.Parse(txtNum2.Text)).ToString();
}
</script>
<html>
<head> </head>
<body>
<form runat="server">
 <p>
    <em>First Number to Add </em>:
    <asp:TextBox id="txtNum1" runat="server"
     Width="43px">50</asp:TextBox>
 </p>
 <p>
    <em>Second Number To Add </em>:
    <asp:TextBox id="txtNum2" runat="server"
     Width="44px">50</asp:TextBox>
 </p>
 <p>
    <strong><u>Web Service Result -</u></strong>
 </p>
 <p>
    <em>Hello world Service</em> :
    <asp:Label id="Label1" runat="server"
     Font-Underline="True">Label</asp:Label>
 </p>
 <p>
    <em>Add Service</em> :
    & <asp:Label id="Label2" runat="server"
    Font-Underline="True">Label</asp:Label>
 </p>
 <p align="left">
    <asp:Button id="runSrvice" onclick="runSrvice_Click"
     runat="server" Text="Execute"></asp:Button>
 </p>
</form>
</body>
</html>
```

Después de crear un cliente, se necesita crear un proxy para acceder al servicio Web.

El proxy es creado utilizando la utilidad wsdl del .NET Framework. Esta utilidad extrae información del servicio Web para crear dicho proxy. Así, el proxy creado es válido únicamente para un servicio Web particular. Visual Studio .NET crea un proxy automáticamente cuando se agrega la referencia para el servicio Web.

Si se crea un proxy utilizando la utilidad wsdl del .NET .Framework será posible observar que se generará un archivo con extensión .cs en el directorio actual (directorio desde el cual se ejecuta la instrucción wsdl).

```
c:>WSDL http://localhost/MyWebServices/FirstService.asmx
```

Lo siguiente es compilar el archivo con extensión .cs para crear el proxy para el servicio Web, el cual es un archivo con extensión .dll. Para este propósito debe escribirse lo siguiente:

```
c:> csc /t:library FirstService.cs
```

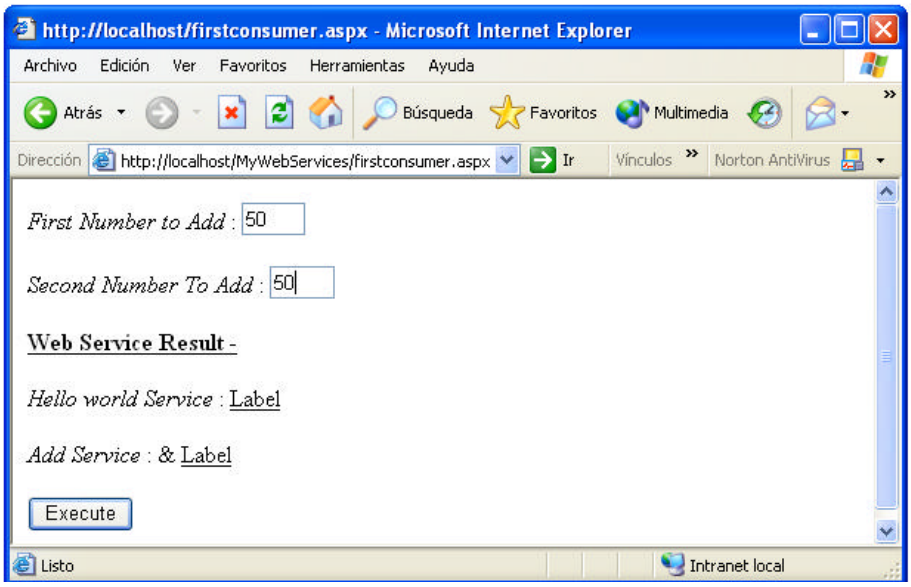
A continuación se debe colocar el proxy en el directorio bin del directorio virtual del servicio Web (c:\MyWebServices\bin) ya que el servidor de información de Internet busca el proxy en este directorio o cualquier directorio bin del servidor Web local.

Ahora se debe crear el cliente para el servicio, el cual ya esta hecho. Nótese que se ha instanciado un objeto del proxy del servicio Web en el cliente. Este proxy cuida la interacción con el servicio.

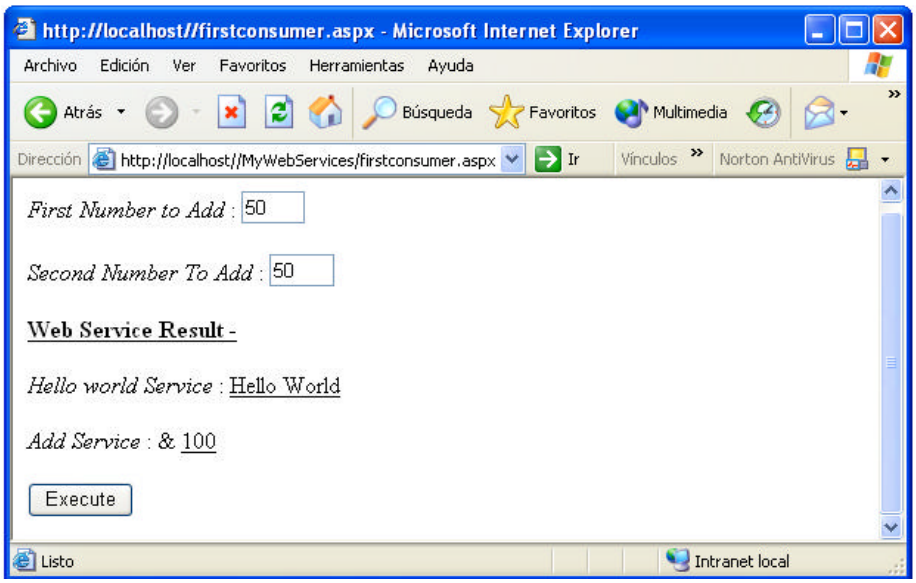
Para probar el consumidor del servicio Web, se debe escribir su URL en el explorador de Internet, por ejemplo:

http://localhost/MyWebServices/Firstconsumer.aspx

El cliente Firstconsumer.aspx despliega un formulario para que sea llenado por el usuario del servicio Web. Esta pantalla se muestra a continuación:



Una vez introducidos los datos (dos números) y habiendo presionado el botón “Execute” aparecerá la siguiente pantalla en la que se puede apreciar el resultado de invocar los métodos del servicio Web: suma y Hola mundo:



SERVICIO WEB: HOLA Y SUMA

Lenguaje utilizado: C#.

Cliente para el servicio Web basado en la aplicación Windows

Escribir una aplicación cliente del servicio Web basada en Windows es lo mismo que escribir cualquier otra clase de aplicación. El único trabajo que se hace es crear el proxy y referenciarlo cuando se compila la aplicación.

El siguiente código es la aplicación Windows que utiliza al servicio Web. Esta aplicación crea un objeto de servicio web (proxy) y llama a los métodos SayHello y Add.

WinApp.cs

```
using System;
  using System.IO;
  namespace SvcConsumer{
  class SvcEater
  {
     public static void Main(String[] args)
     {
        FirstService mySvc = new FirstService();

        Console.WriteLine("Calling Hello World Service: " +
                        mySvc.SayHello());
        Console.WriteLine("Calling Add(2, 3) Service: " +
                        mySvc.Add(2, 3).ToString());
     }
   }
 }
```

El código anterior puede copiarse y pegarse en algún editor de textos asegurándose de que se guarde como: **WinApp.cs.** para este ejemplo**.**

Se debe compilar utilizando el siguiente comando:

c:>csc /r:FirstService.dll WinApp.cs.

Esto creará una aplicación llamada “WinApp.exe”.

Finalmente, se debe ejecutar “WinApp.exe” para probar que dicha aplicación realmente es cliente del servicio Web.

Ahora la pregunta es: ¿Cómo estar seguro de que la aplicación creada realmente está utilizando al servicio Web?. Es simple probarlo. Se puede detener el servidor Web de tal forma que no pueda ser contactado.

Después ejecutar la aplicación "WinApp.exe". Ocurrirá una excepción en el tiempo de ejecución. Posteriormente, se puede iniciar nuevamente el servidor Web para observar que el servicio Web será llamado y ejecutado por la aplicación.

SERVICIO WEB: CALCULATOR

Lenguaje utilizado: Visual Basic.

En este ejemplo se demuestra la facilidad de crear un servicio Web y un cliente para el mismo utilizando la plataforma .NET.

Descripción del servicio Web "Calculator"

El servicio Web es una calculadora que permite realizar operaciones básicas como: sumar, restar y dividir.
El cliente del servicio Web es una aplicación con extensión html. Es una página sencilla que se presenta al usuario para que introduzca los números y ejecute la operación que éste desee.

El código para el servicio Web se muestra a continuación:

```
<%@ WebService Language ="Vb" Class="Calculator" %>
    option strict off
    Imports System.Web.Services
     Public Class Calculator: Inherits WebService
     <WebMethod()>  Public Function Add(Num1 as integer,Num2 as integer)
As Integer
      return Num1 + Num2
End Function
<WebMethod()> Public Function Substract(Num1 as integer,Num2 as integer)
As Integer
      return Num1 - Num2
End Function
<WebMethod()> Public Function  Divide(Num1 as integer,Num2 as integer) As
Integer
      return Num1 / Num2
End Function
End class
```

La directiva “<%@ WebService Language ="Vb" Class="Calculator" %>” indica que se trata de un servicio Web escrito en lenguaje Visual Basic y que utiliza una clase llamada Calculator.

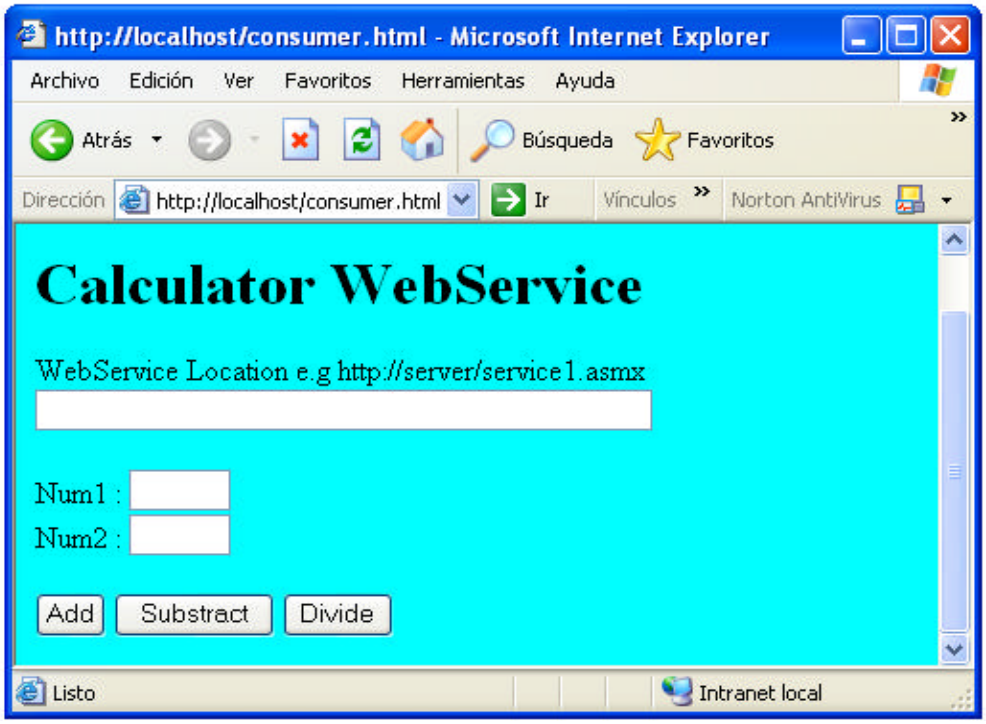
Este archivo es guardado como calculator.asmx dentro del directorio del servidor de información de Internet (wwwroot).

Como puede observarse el archivo Calculator.asmx tiene tres métodos. Cada uno de ellos acepta dos números y regresa un valor entero después de realizar la operación aritmética deseada.

El código para la aplicación cliente es el siguiente:

```
<html>
<h1> Calculator WebService </h1>
<body bgcolor = cyan>
<form id = frm method=POST >
WebService Location  e.g  http://server/service1.asmx <br>
<input type = text id = ServiceLocation style="WIDTH:
322px;"><br><br>
     Num1 : <input type="text" size="5" name='Num1' \"><br>
     Num2 : <input type="text" size="5" name='Num2' \"><br><br>
<input type = button Value = Add onclick = "Add()">
     <input type = button Value = Substract onclick = "Substract()">
<input type = button Value = Divide onclick = "Divide()">
</form>
<script Language = "vbScript">
Sub Add
msgbox frm.ServiceLocation.value
frm.action = frm.ServiceLocation.value & "/Add"
frm.submit
end sub
sub Substract
frm.action = frm.ServiceLocation.value & "/Substract"
frm.submit
end sub
sub Divide
frm.action = frm.ServiceLocation.value & "/Divide"
frm.submit
end sub
</script>
</body>
</html>
```

Este código es guardado con el nombre de consumer.html y la pantalla que se muestra al llamarlo en el servidor de información de Internet es la siguiente:

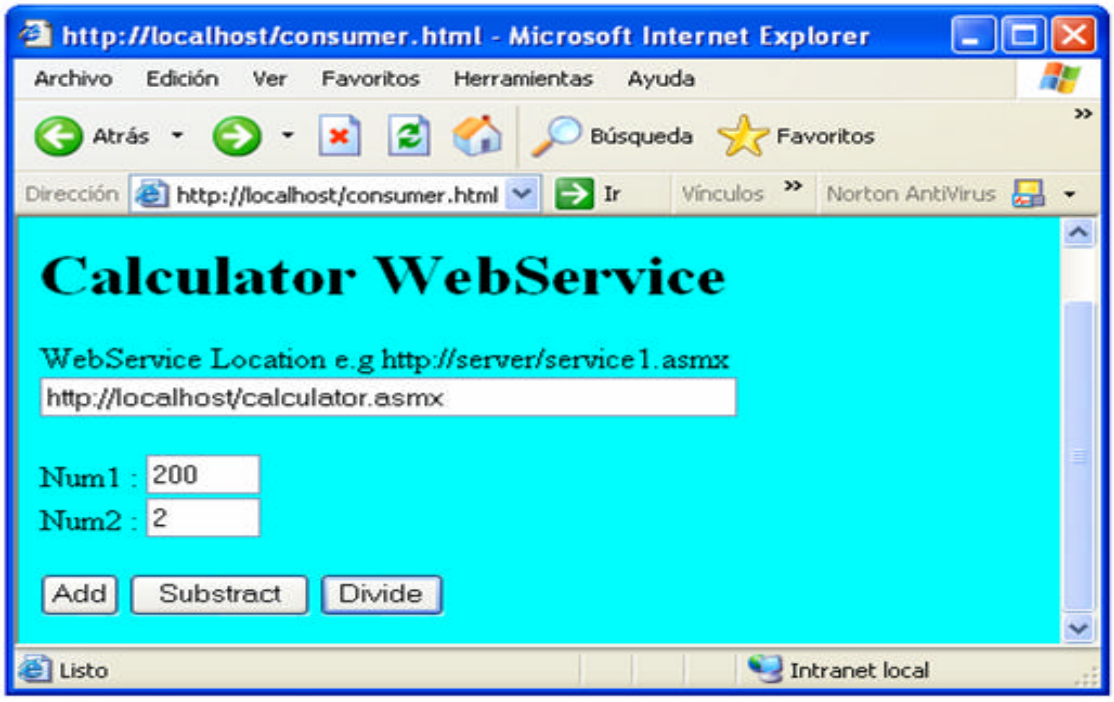


En el primer cuadro de texto el usuario debe introducir la dirección en la que se encuentra el servicio Web.

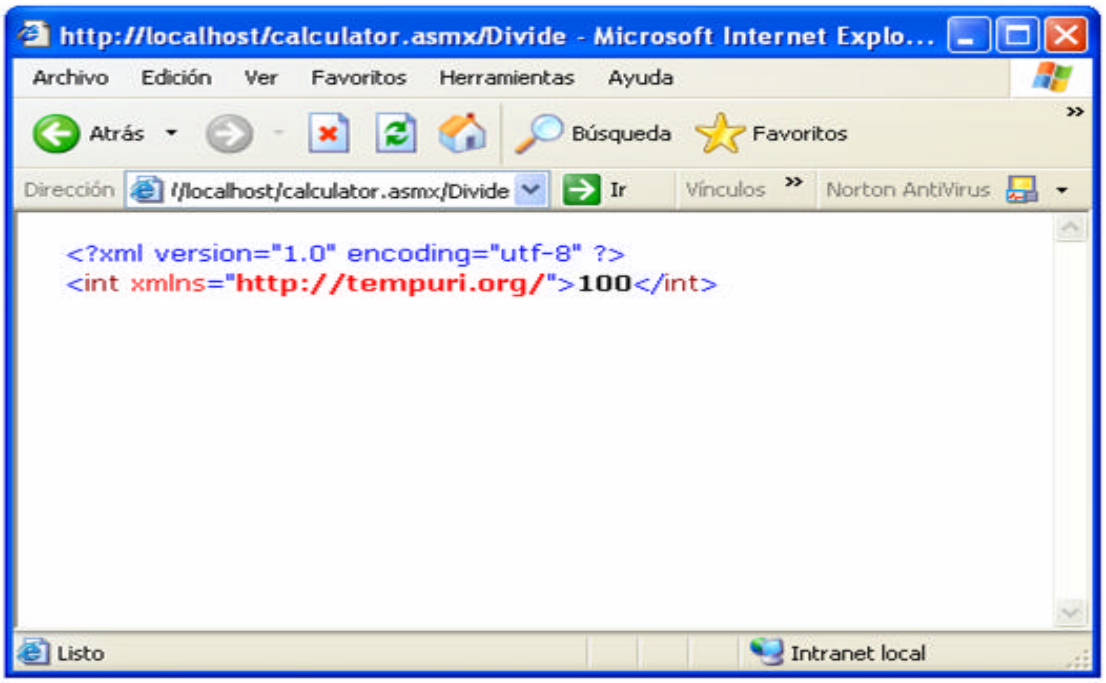
Los dos cuadros de texto inferiores son para que el usuario introduzca los números que desea utilizar para alguna operación.

Finalmente, los botones etiquetados con las palabras “Add”, “Substract” y “Divide” son para ejecutar alguna suma, resta o división respectivamente.

A continuación se muestra la pantalla con los datos deseados y todo listo para invocar al servicio Web.



La pantalla siguiente muestra el resultado de la invocación del método "Divide" con los números 200 y 2 en formato xml, ya que es el formato de respuesta por defecto utilizado por los servicios web xml.



SERVICIO WEB: SUMA

Lenguaje utilizado: C#.

Publicación de un Servicio Web en una página ASP.NET

```
<%@ WebService Language="C#" Class="suma"%>

// Para publicar este servicio Web, copiar el fichero al directorio
// de publicación de Web predeterminado de IIS, que por defecto
// es c:\Inetpub\Wwwroot
// Luego se puede invocar, para probarlo, desde un navegador
// con http://localhost/suma.asmx

using System;
using System.Web.Services;

public class suma : WebService
{
  [WebMethod]
  public int getSuma(int a, int b)
  {
    return a+b;
  }
}
```

Para que Internet Information Server (IIS) publique el servicio, el fichero tiene que estar localizado en un directorio de publicación. Se puede copiar directamente al directorio de publicación de Web predeterminado de IIS o configurar IIS para que publique ficheros de otros directorios.

IIS genera una página que permite probar "a mano" el Servicio Web. En esta página se publica también el descriptor del servicio (WSDL) generado por IIS, que es un fichero XML que describe formalmente el comportamiento de cada servicio web, y que se utiliza para generar automáticamente las clases "proxy" que permiten a los clientes acceder al servicio web, como veremos más adelante.

El fichero ".asmx" lo compila "al vuelo" el IIS cuando intentamos acceder a él, por lo que si hay errores, aparecen en el propio browser como resultado de la consulta.

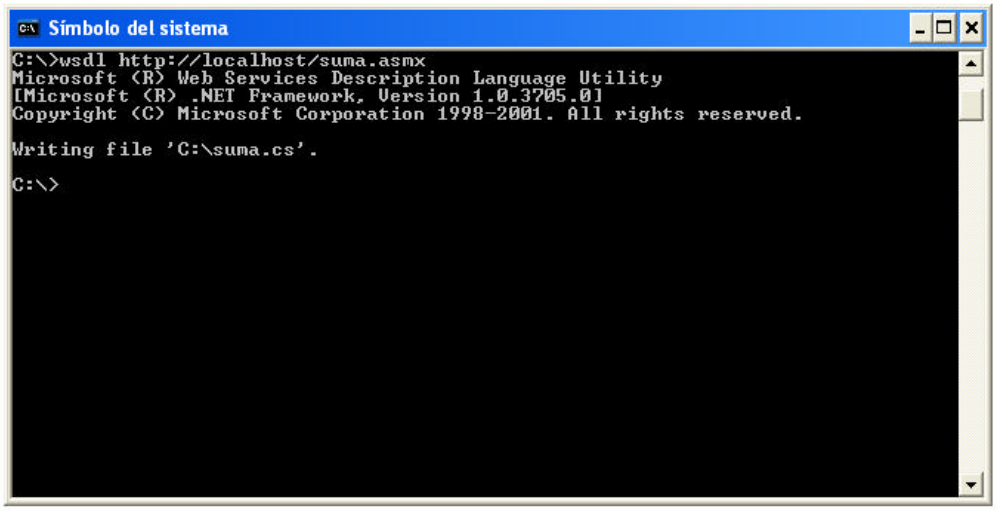
Acceso a un Servicio Web desde un programa cliente escrito en C# sobre .Net

Para acceder al Servicio Web desde un programa cliente tenemos primero que generar la clase "proxy" a partir del descriptor del servicio web. En el próximo ejemplo tenemos un archivo de proceso por lotes que genera esta clase con la utilidad de Microsoft "wsdl.exe"

Generación de una clase "proxy" en C#

Generación del proxy SOAP con la Utilidad de descripción de lenguaje de servicios Web de Microsoft (wsdl.exe) a partir del descriptor del servicio WDSL (Web Services Description Language) publicada por el servidor web:

wsdl /language:CS http://localhost/suma.asmx?WSDL



Con esto se generará un archivo llamado, suma.cs, que contendrá el siguiente código:

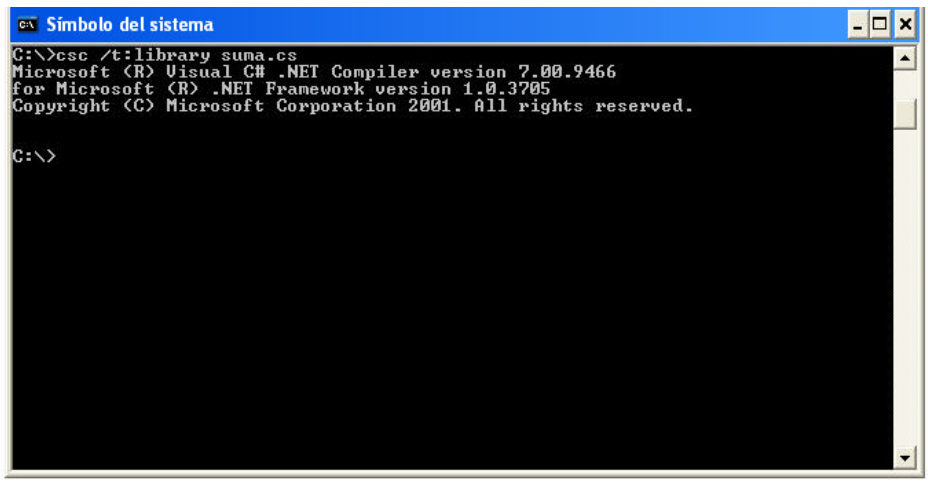
```
//------------------------------------------------------------------------------
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.0.3705.209
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </autogenerated>
//------------------------------------------------------------------------------

//
// wsdl genera automáticamente este código fuente, versión=1.0.3705.209.
//
using System.Diagnostics;
using System.Xml.Serialization;
using System;
using System.Web.Services.Protocols;
using System.ComponentModel;
using System.Web.Services;


/// <remarks/>
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Web.Services.WebServiceBindingAttribute(Name="sumaSoap",
Namespace="http://tempuri.org/")]
public class suma : System.Web.Services.Protocols.SoapHttpClientProtocol {

    /// <remarks/>
    public suma() {
        this.Url = "http://localhost/suma.asmx";
    }

    /// <remarks/>

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempu
ri.org/getSuma", RequestNamespace="http://tempuri.org/",
ResponseNamespace="http://tempuri.org/",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped
)]
```

```
public int getSuma(int a, int b) {
      object[] results = this.Invoke("getSuma", new object[] {
                a,
                b});
      return ((int)(results[0]));
   }

   /// <remarks/>
   public System.IAsyncResult BegingetSuma(int a, int b,
System.AsyncCallback callback, object asyncState) {
      return this.BeginInvoke("getSuma", new object[] {
                a,
                b}, callback, asyncState);
   }

   /// <remarks/>
   public int EndgetSuma(System.IAsyncResult asyncResult) {
      object[] results = this.EndInvoke(asyncResult);
      return ((int)(results[0]));
   }
}
```

Ahora se debe compilar suma.cs escribiendo:

csc /t:library suma.cs.

Con este comando se generará el suma.dll que será utilizado posteriormente. A continuación se observa la pantalla que resulta de la compilación:



En la pantalla anterior no aparece ningún mensaje, si embargo, puede comprobarse que con la instrucción anterior se ha creado el archivo suma.dll en la unidad actual.

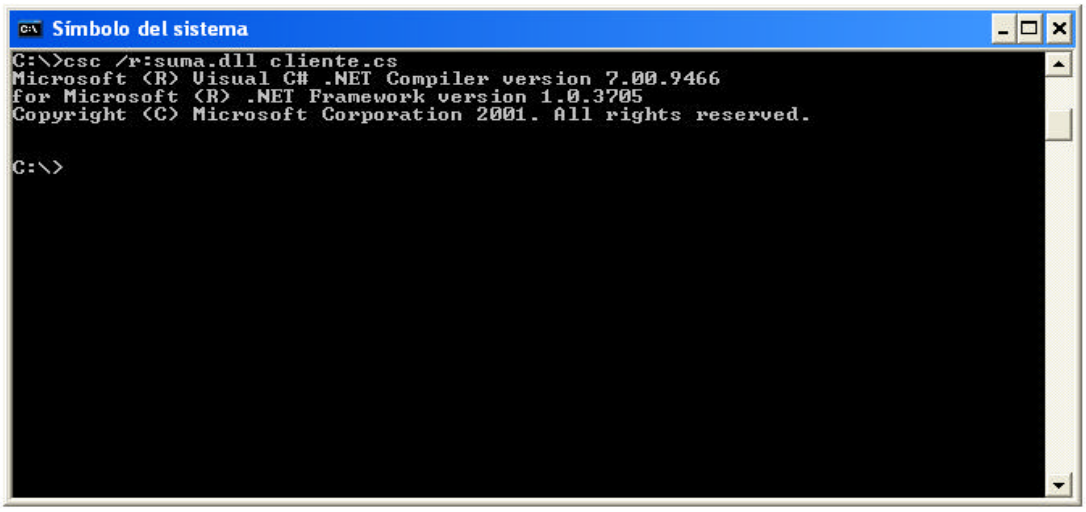
A continuación se escribe el cliente, que accederá al Servicio Web a través del "proxy" en lenguaje c#:

```
using System;
public class Cliente
{
   public static void Main(string[] args)
    {
         try {
               suma s = new suma();
               Console.Write("1+1={0}",s.getSuma(1,1));
              }
        catch (Exception e) {
               Console.WriteLine("Excepcion:");
                Console.WriteLine(e.StackTrace);
         }
     }
}
```

Para generar la aplicación que accederá al servicio web se utiliza el archivo generado con el nombre “suma.dll” y el cliente escrito anteriormente. Para crear el cliente ejecutable debe escribirse la siguiente instrucción:

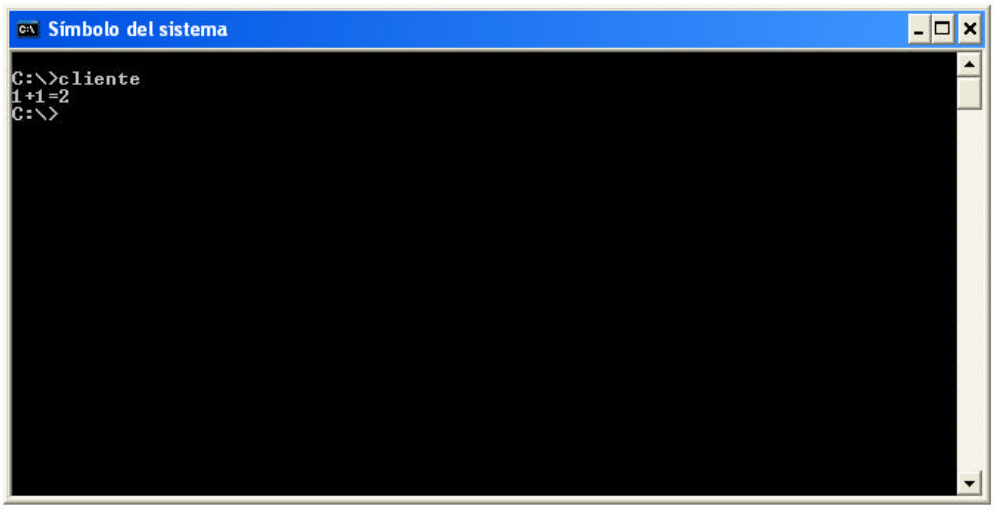
```
csc /r:suma.dll cliente.cs
```

A continuación se muestra la pantalla que debe aparecer en la generación del cliente ejecutable:



Finalmente se comprueba que la aplicación generada (cliente.exe) efectivamente acceda al servicio web (suma.asmx).

En la siguiente pantalla se observa el resultado de la ejecución del cliente:



PROYECTO APACHE AXIS

IBM ha proporcionado gran cantidad de código al grupo Apache, incluyendo SOAP4J, su grupo de herramientas SOAP. Los creadores de Apache SOAP y SOAP4J están trabajando en la grandiosa herramienta llamada Apache AXIS, cuyas características facilitan el trabajo de los programadores. Las acciones más comunes que se pueden realizar con esta herramienta son: “Exponer funcionalidad como un servicio Web” y "Acceder a ese servicio Web". Seguramente sería de mucha ayuda centrarse en la interfase y no tener que aprender sobre la plataforma principal que se utilice. Es la misma idea de no tener que aprender acerca de la capa de transporte del modelo de referencia OSI y del protocolo de Internet cuando se accede a un URL sobre HTTP.

Código como servicio Web en un solo paso

Los desarrolladores de Apache se dieron cuenta de lo agradable que sería colocar el código en cualquier lugar y convertirlo fácilmente en un servicio Web. Esta simplicidad es una tendencia actual que Microsoft presenta en su tecnología .NET y BEA en la plataforma WebLogic 7. Pero qué tan fácil es:

- ¿Instalar el Servicio Web?
- ¿Escribir un cliente que pueda acceder a ese servicio Web?
- ¿Obtener la descripción del servicio (documento WSDL) para el servicio Web instalado?

Estas preguntas se responderán a continuación.

Implementar una clase Java como un servicio Web

Simplemente se copia el archivo java en la aplicación Web AXIS, utilizando la extensión *.jws* en lugar de .java. La extensión .jws indica que se trata de un servicio Web en java:

Con sólo tener el código del servicio Web en la aplicación Web es posible instalarlo y accederlo. Si se abre el explorador y se accede al archivo .jws (ejemplo http://localhost:8080/axis/servicio.jws) se podrá observar que se está accediendo a un servicio Web.

Obtener el WSDL para el servicio Web instalado

Si se desea tener un archivo WSDL para que un cliente acceda al servicio (sin importar que esté utilizando .NET o cualquier otra plataforma), Apache resuelve este problema. Se puede tener el archivo de definición del servicio Web mediante el simple acceso al servicio agregando solamente *?WSDL* al final del URL. Ejemplo, si se escribe *http://localhost:8080/axis/servicio.jws?WSDL* en el explorador se obtiene el descriptor XML.

Trabajando con la implementación de un servicio Web

A pesar de que es realmente fácil y conveniente colocar el código Java debajo del directorio Axis como un archivo *.jws*, no será la forma de implementar servicios Web. La mayor parte del tiempo se querrá encontrar la forma de controlar el servicio Web y de utilizar características más avanzadas del mismo. Afortunadamente, con otras herramientas es más fácil trabajar con el código de una manera más formal.

APIs de Java para llamadas a procedimientos remotos basadas en XML (JAX-RPC)

JAX-RPC es una de las APIs de java para registrar y utilizar servicios Web.

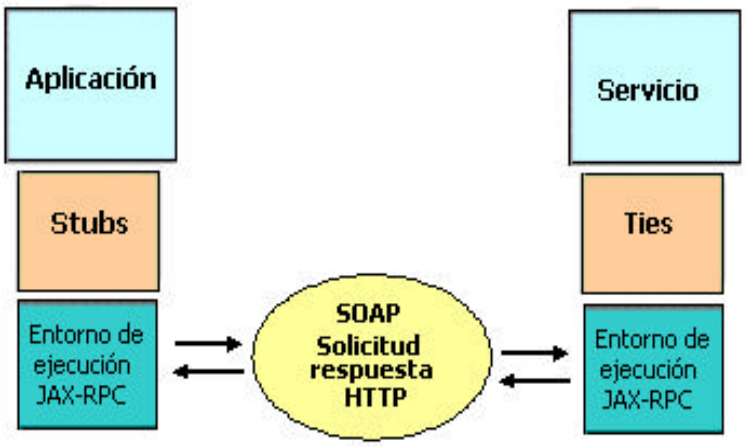
JAX-RPC define bibliotecas API de Java que los desarrolladores pueden utilizar en sus aplicaciones para desarrollar y consumir servicios web.

Mediante JAX-RPC, un cliente Java puede consumir un servicio web que reside en un servidor remoto a través de Internet, incluso aunque el servicio esté escrito en otro idioma y se ejecute en una plataforma diferente. Los clientes que no sean de tipo Java también pueden consumir servicios JAX-RPC.

JAX-RPC utiliza un protocolo de mensajería XML, tal como SOAP, para transmitir una llamada a procedimiento remoto a través de una red. Por ejemplo, un servicio web que devuelve la cotización de unas acciones recibiría una solicitud HTTP SOAP con una llamada a un método realizada desde el cliente. Mediante JAX-RPC, el servicio extrae del

mensaje SOAP la llamada al método, la traduce convenientemente y realiza esa llamada. A continuación, el servicio utiliza JAX-RPC para convertir la respuesta del método otra vez en un mensaje SOAP y enviar el mensaje de regreso al cliente. El cliente recibe el mensaje SOAP y utiliza JAX-RPC para traducirlo en una respuesta.

El entorno de ejecución JAX-RPC genera stubs y ties. Éstos son clases de bajo nivel que permiten la comunicación entre el cliente y el servicio. Los stubs y los ties realizan funciones parecidas, la diferencia es que los stubs operan en el lado del cliente y los ties en el lado del servidor, es decir, un stub que reside en el cliente es un objeto local que representa un servicio remoto y actúa como proxy para el servicio. Un objeto tie, que reside en el servidor, actúa como un proxy en el servidor. Esto puede observarse en la siguiente figura:



Una llamada a procedimientos remotos basada en servicios web es una colección de procedimientos que pueden ser llamados por un cliente remoto a través de Internet.

Un servicio web que contiene procedimientos disponibles para los clientes es registrado en un contenedor en el lado del servidor. El contenedor puede ser un contenedor de servlets tal como Tomcat.

Interoperabilidad

El requerimiento más importante para un servicio web es la interoperabilidad a través de los clientes y los servidores. Con JAX_RPC un cliente escrito en cualquier lenguaje de programación puede acceder un servicio web desarrollado y registrado en la plataforma Java.

Un cliente escrito en el lenguaje de programación Java puede comunicarse con servicios web que han sido desarrollados en diferentes plataformas.

Facilidad de uso de JAX-RPC

JAX-RPC es el API principal para servicios web. Es una herramienta muy fácil de utilizar tanto para aplicaciones cliente como para aplicaciones servidor.

JAX_RPC se basa en un mecanismo de llamadas a procedimientos remotos haciendo que los principales detalles de implementación sean invisibles tanto para el cliente como para el desarrollador del servicio web.

Un cliente de un servicio web simplemente realiza las llamadas a los métodos de Java. Los procesos de serialización, desserialización y los detalles de transmisión se llevan a cabo automáticamente.

En JAX-RPC existe una herramienta que se utiliza para generar stubs utilizando documentos WSDL, el nombre de dicha herramienta es wscompile.

También existe la herramienta wsdeploy que se utiliza para crear los ties. Esta herramienta también puede ser utilizada para crear los documentos WSDL.

Un sistema de ejecución JAX-RPC opera de la siguiente forma:

Primero convierte la llamada del método remoto del cliente en un mensaje SOAP y posteriormente lo envía al servicio como una petición http.

En el lado del servidor, el sistema de ejecución JAX_RPC recibe la petición, traduce el mensaje SOAP a una llamada al método y finalmente lo invoca.

Después de que el servicio web ha procesado la ejecución, el sistema de ejecución JAX-RPC realiza una serie de pasos similares a los anteriores para regresar un resultado al cliente.

REQUISITOS PARA EL CONSUMO DE SERVICIOS WEB EN JAVA

- Tener un servidor HTTP (compatible con SOAP)
- Crear las clases del servicio Web (programación Java pura)
- Dar de alta las clases con la herramienta deploy
- Crear un cliente que efectúe la llamada a los métodos del servicio

INSTALACIÓN DEL SERVIDOR

El servidor que se puede utilizar es Jakarta Tomcat + Apache Axis. Es posible obtenerlos de las siguientes direcciones:

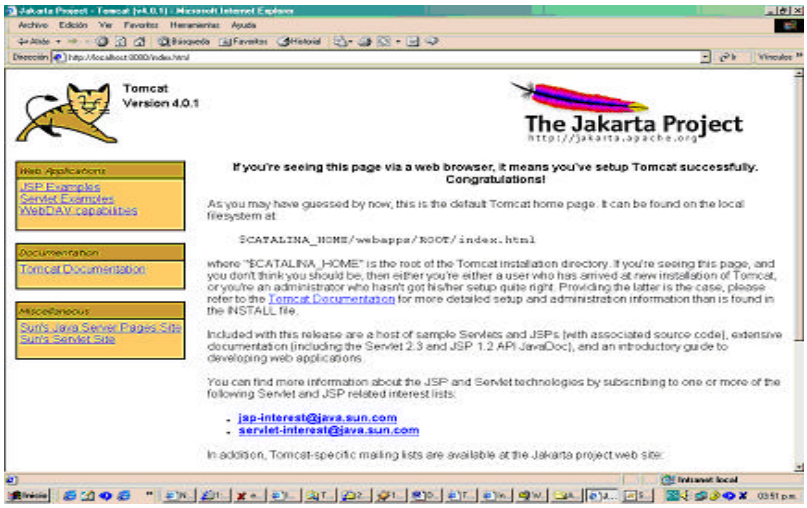
http://jakarta.apache.org/tomcat/
http://ws.apache.org/axis/

La instalación de Tomcat es realmente muy sencilla.

Lo primero que debe hacer es descargar el instalador. Una vez descargado el instalador, se debe ejecutar. Los pasos a seguir son bastante intuitivos y no presentan problema alguno.

Tomcat se ejecuta por omisión en el puerto 8080, así que una vez que se haya arrancado Tomcat se puede probar la instalación abriendo en el navegador la dirección *http://localhost:8080*. Si la instalación no tuvo problemas se mostrará una página de bienvenida semejante a ésta:

Ventana de bienvenida de Tomcat



Ambiente Java

Para poder ejecutar Tomcat se necesita tener instalado el kit de desarrollo de Java.

Una vez instalado el servidor lo que se debe hacer es tener el proyecto Axis descomprimido y localizar la carpeta llamada Axis. Esta carpeta debe ser colocada dentro del directorio webapps de Tomcat.

DEMOSTRACIÓN DE INTEROPERABILIDAD EN LOS SERVICIOS WEB

ACCEDER A UN SERVICIO WEB .NET MEDIANTE UN CLIENTE JAVA:

Los requerimientos para realizar este proceso son los siguientes:

- Servidor de información de Internet y el .net Framework
- Servicio Web realizado en el lenguaje de programación C#
- Paquete de desarrollo de servicios web en java(Java web services developer pack)
- Un servidor apache tomcat y proyecto axis

Es perfectamente posible acceder a un Servicio Web construido con herramientas Microsoft y desplegado sobre una arquitectura Microsoft, desde un cliente construido con otras herramientas corriendo sobre otras arquitecturas. A continuación se muestra cómo escribir un cliente en Java, que podría funcionar en cualquier plataforma. El software que se usa es el *Java Web Services Developer Pack 1.0*. Este paquete puede descargarse gratuitamente de http://java.sun.com/.

El java Web services developer pack durante su instalación incluye al servidor Tomcat.

Los pasos que se siguen para acceder a un servicio Web son los mismos que en el caso de un cliente .NET. En primer lugar, se necesita generar una clase "proxy" que actúe de puente entre el cliente y el servicio Web. Para ello se usará la herramienta "wscompile" de SUN. Se debe escribir un fichero de configuración para decirle dónde está el descriptor del Servicio. El archivo utilizado para este ejemplo es el siguiente:

Fichero de configuración config.xml

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

Formato del archivo de configuración config.xml

El comando wscompile lee el archivo de configuración config.xml. Este archivo contiene información que describe el servicio web. La estructura del archivo es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
<service> or <wsdl> or <modelfile>
</configuration>
```

El archivo de configuración puede contener exactamente un elemento <service>, <wsdl> o <modelfile>.

Elemento servicio

Si se especifica el elemento servicio (<service>), wscompile lee la interface RMI que describe el servicio y genera un documento WSDL.

En el subelemento interface (<interface>), el atributo nombre especifica la interface RMI del servicio, y el atributo servantName especifica la clase que implementa esta interface.

Elemento WSDL

Si se especifica el elemento <wsdl>, wscompile lee el archivo WSDL y genera la interface RMI del servicio. El atributo location especifica el URL del archivo WSDL y el atributo packageName especifica el paquete de clases que serán generadas. Por ejemplo:

```
<wsdl
location="http://tempuri.org/sample.wsdl"
packageName="org.tempuri.sample"/>
```

Si el archivo config.xml contiene un elemento <service> o <wsdl>, wscompile puede generar un modelo de archivo que contiene las estructuras de datos internas que describen al servicio.

Si el modelo de archivo ya esta generado puede reutilizarse la próxima vez que se utilice el wscompile. Por ejemplo:

<modelfile location="mymodel.Z"/>

Ejemplos

Wscompile para generar los stubs del lado del cliente

```
wscompile -gen:client –d outputdir -classpath classpathdir config.xml
```

Donde un artefacto del lado cliente es generado en el directorio que se especifique con outputdir para un servicio en ejecución definido en el archivo config.xml.

Wscompile para generar los stubs del lado del servidor

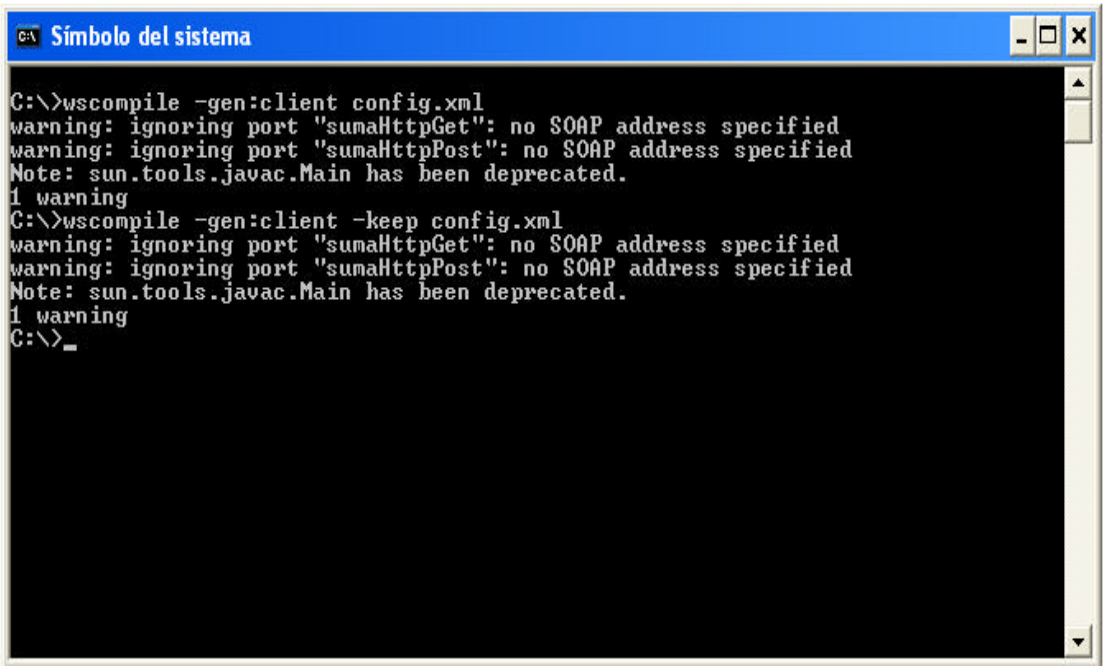
```
wscompile -gen:server -d outputdir -classpath classpathdir
-model modelfile.Z config.xml
```

Generación de una clase "proxy" en Java

La utilidad wscompile de Soap permite generar el proxy a partir del descriptor del servicio WDSL publicada por el servidor web. La sintaxis utilizada es la siguiente:

wscompile –gen:client -g -keep config.xml

A continuación se muestra la pantalla que resulta al utilizar esta instrucción:



La opción -g es para generar información de debug y la opción -keep para que no se borre el código fuente. En lugar de generar una única clase como en .Net, en este caso se tienen varias. La que interesa es "SumaSoap.java", que es una interface que publica un método con la signatura del método del Servicio Web y "Suma_Impl.java" que proporciona un método "getsumaSoap()" que devuelve una clase que implementa la interface.

Interface del "proxy" Java

```
// This class was generated by the JAXRPC RI, do not edit.
// Contents subject to change without notice.

package suma;

public interface SumaSoap extends java.rmi.Remote {
    public int getSuma(int a, int b) throws
        java.rmi.RemoteException;
```

```
}
```

Clase que proporciona el "proxy" Java

```
/ This class was generated by the JAXRPC RI, do not edit.
// Contents subject to change without notice.

package suma;

import com.sun.xml.rpc.encoding.*;
import com.sun.xml.rpc.client.ServiceExceptionImpl;
import com.sun.xml.rpc.util.exception.*;
import com.sun.xml.rpc.client.HandlerChainImpl;
import javax.xml.rpc.*;
import javax.xml.rpc.encoding.*;
import javax.xml.rpc.handler.HandlerChain;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.namespace.QName;

public class Suma_Impl extends com.sun.xml.rpc.client.BasicService
implements Suma {
   private static final QName serviceName = new
QName("http://tempuri.org/", "suma");
   private static final QName ns1_sumaSoap_QNAME = new
QName("http://tempuri.org/", "sumaSoap");
   private static final Class sumaSoap_PortClass = suma.SumaSoap.class;

   public Suma_Impl() {
      super(serviceName, new QName[] {
                  ns1_sumaSoap_QNAME
               },
         new Suma_SerializerRegistry().getRegistry());

   }

   public java.rmi.Remote getPort(QName portName, Class
serviceDefInterface) throws ServiceException {
      try {
         if (portName.equals(ns1_sumaSoap_QNAME) &&
            serviceDefInterface.equals(sumaSoap_PortClass)) {
            return getSumaSoap();
         }
      } catch (Exception e) {
         throw new ServiceExceptionImpl(new
LocalizableExceptionAdapter(e));
      }
```

```
        return super.getPort(portName, serviceDefInterface);
    }

    public java.rmi.Remote getPort(Class serviceDefInterface) throws
ServiceException {
        try {
            if (serviceDefInterface.equals(sumaSoap_PortClass)) {
                return getSumaSoap();
            }
        } catch (Exception e) {
            throw new ServiceExceptionImpl(new
LocalizableExceptionAdapter(e));
        }
        return super.getPort(serviceDefInterface);
    }
    public suma.SumaSoap getSumaSoap() {
        String[] roles = new String[] {};
        HandlerChainImpl handlerChain = new
HandlerChainImpl(getHandlerRegistry().getHandlerChain(ns1_sumaSoap_QNA
ME));
        handlerChain.setRoles(roles);
        suma.SumaSoap_Stub stub = new suma.SumaSoap_Stub(handlerChain);
        try {
            stub._initialize(super.internalTypeRegistry);
        } catch (JAXRPCException e) {
            throw e;
        } catch (Exception e) {
            throw new JAXRPCException(e.getMessage(), e);
        }
        return stub;
    }
}
```

Código del servicio web .net

```
<%@ WebService Language="C#" Class="suma"%>

using System;
using System.Web.Services;

public class suma : WebService
{
  [WebMethod]
  public int getSuma(int a, int b)
  {
    return a+b;
```

```
  }
}
```

Código del cliente Java para acceder al servicio web

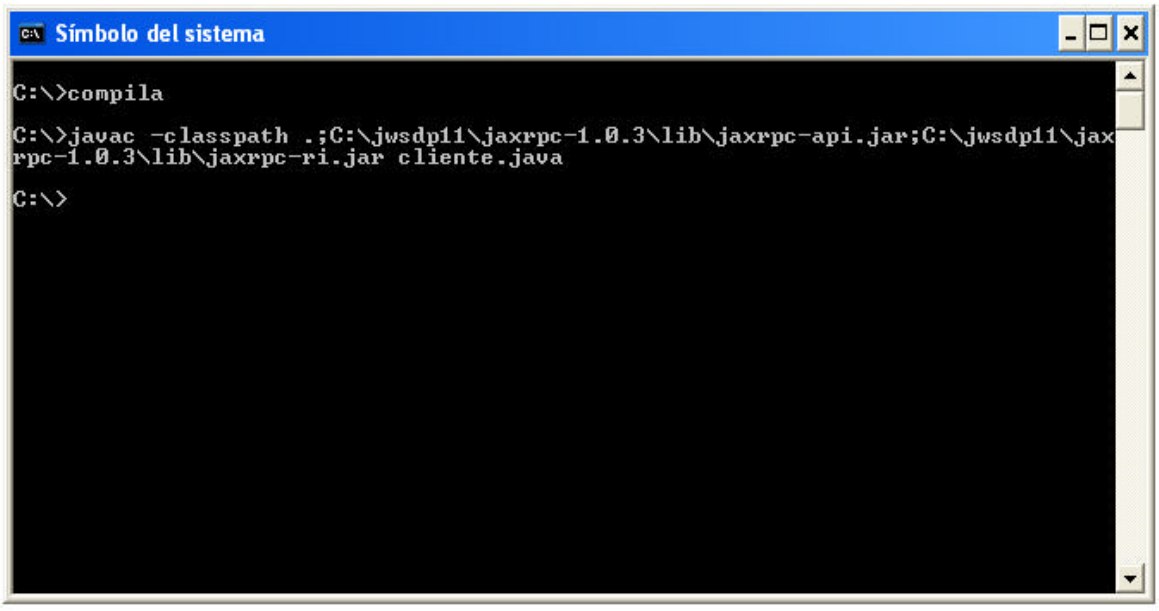
Cliente Java

```
import suma.*;
import javax.xml.rpc.Stub;

public class Cliente {
   public static void main(String[] args) {
      try {
         SumaSoap s = new Suma_Impl().getSumaSoap();
         System.out.println("1+1=" + s.getSuma(1,1));
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
}
```

Compilación del cliente Java

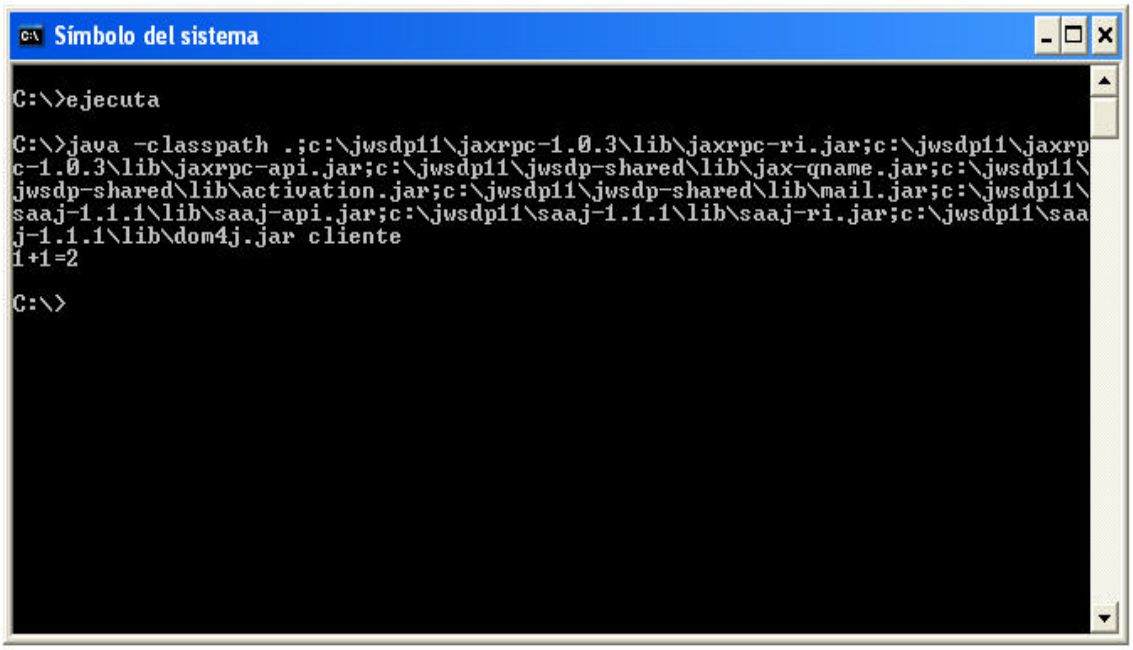
La compilación del cliente exige algunos archivos ".jar" en el classpath, distribuidos con el *Java Web Services Developer Pack 1.0*. Los archivos que se necesitan son jaxrpc-api.jar y jaxrpc-ri.jar.

En la pantalla que se muestra a continuación se puede observar la compilación:



Ejecución del cliente Java

Para la ejecución del cliente ocurre lo mismo que en la compilación, se requieren algunos archivos ".jar" en el classpath. En la siguiente pantalla se muestra el resultado que aparece al ejecutar el cliente:



El resultado de la ejecución es:

1 + 1 = 2

ACCEDER A UN SERVICIO WEB MEDIANTE UN CLIENTE JSP

Acceder un servicio web, construido en el lenguaje C# y funcionando en la plataforma .NET, mediante un jsp implica tener funcionando el servidor Tomcat.
El código del jsp que accede al servicio web se ha colocado dentro de una carpeta a la cual se le puso por nombre “Pruebas”. Esta carpeta a su vez se ha colocado dentro de la carpeta webapps que se crea durante la instalación del “Java Web Services Developer Pack”.

También es importante mencionar que dentro de la carpeta “Pruebas” se encuentra una carpeta llamada WEB-INF que contiene a su vez las carpetas llamadas classes y lib.
Dentro de la carpeta classes debe colocarse el paquete que se genera cuando se utiliza la herramienta wscompile. Este paquete contiene los archivos fuente y .class de java que hacen referencia al servicio web .asmx.

Dentro de la carpeta lib se han colocado los archivos que necesita el cliente java, es decir, aquellos que vienen con el paquete de java web services developer pack. El código del jsp es el siguiente:

Prueba.jsp

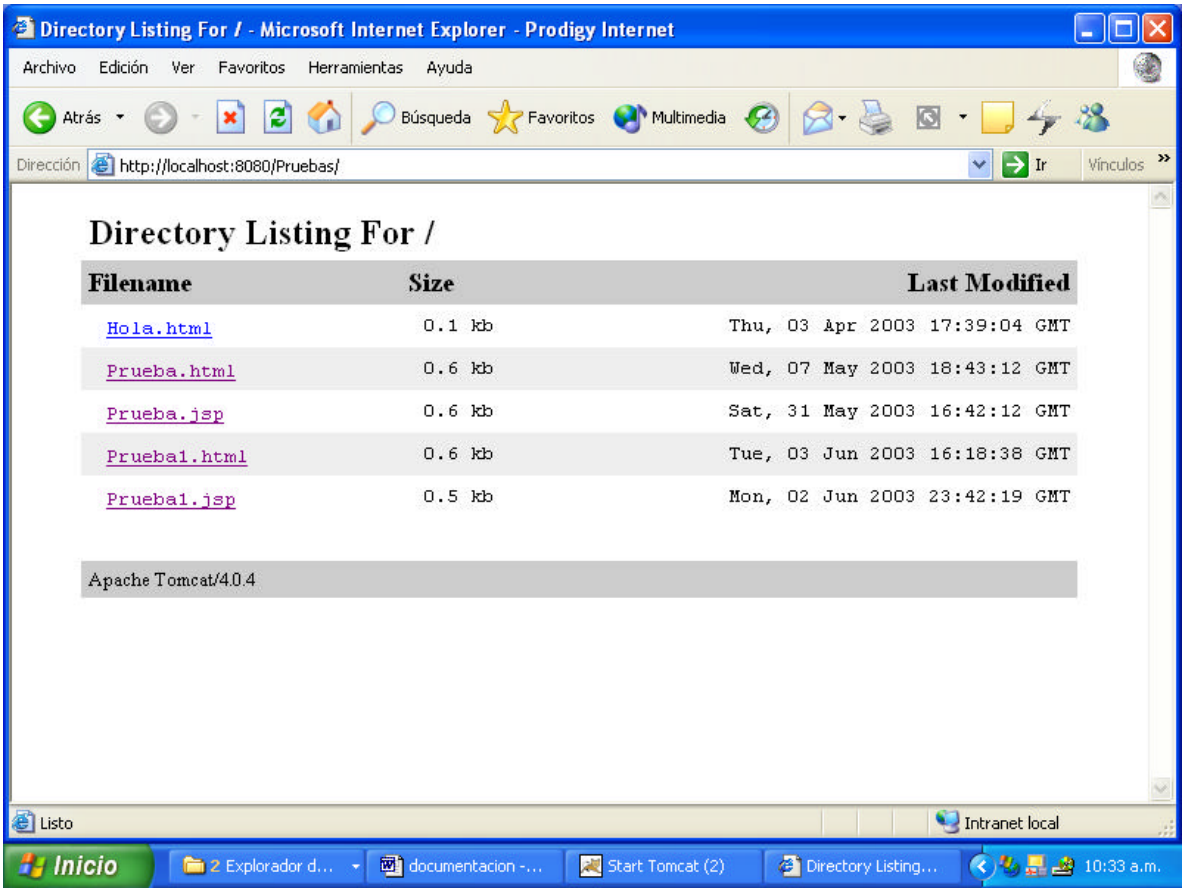
```
<html>
<head>
<title>Usuarios</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<%@ page import="servicio.*" %>
<%@ page import="javax.xml.rpc.Stub" %>
<body>
  <%
    FirstServiceSoap s= new FirstService_Impl().getFirstServiceSoap();
    String a = request.getParameter("param1");
    String b = request.getParameter("param2");
    int r=0;
    %>
   <%=a%>
   <%=b%>
   <%  r=s.add(Integer.parseInt(a),Integer.parseInt(b)); %>
  <H1>El resultado de la suma <%=a%> + <%=b%> es  <%=r%></H1>
</body>
</html>
```

Se ha realizado una página en html para introducir los parámetros, requeridos por el servicio, y posteriormente acceder al servicio. El código de la página se muestra a continuación:

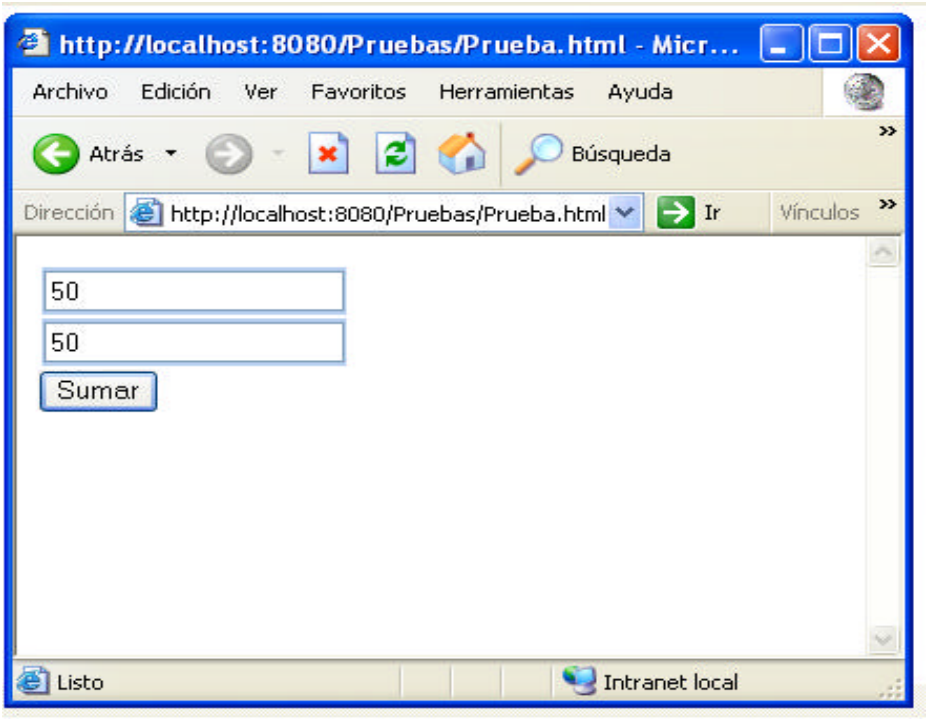
Prueba.html

```
<html>
  <body>
    <form method="post" action="Prueba.jsp">
    <table>
       <tr>
        <td width="100%" align="center" bgcolor="#B8D0F0"><small><font
face="Arial">
           <input type="text" name="param1">
         </font></small>
       </td>
       </tr>
       <tr>
        <td width="100%" align="center" bgcolor="#B8D0F0"><small><font
face="Arial">
           <input type="text" name="param2">
         </font></small>
       </td>
       </tr>
    </table>
     <input type="submit" name="sumar" value="Sumar">
    </form>
  </body>
</html>
```

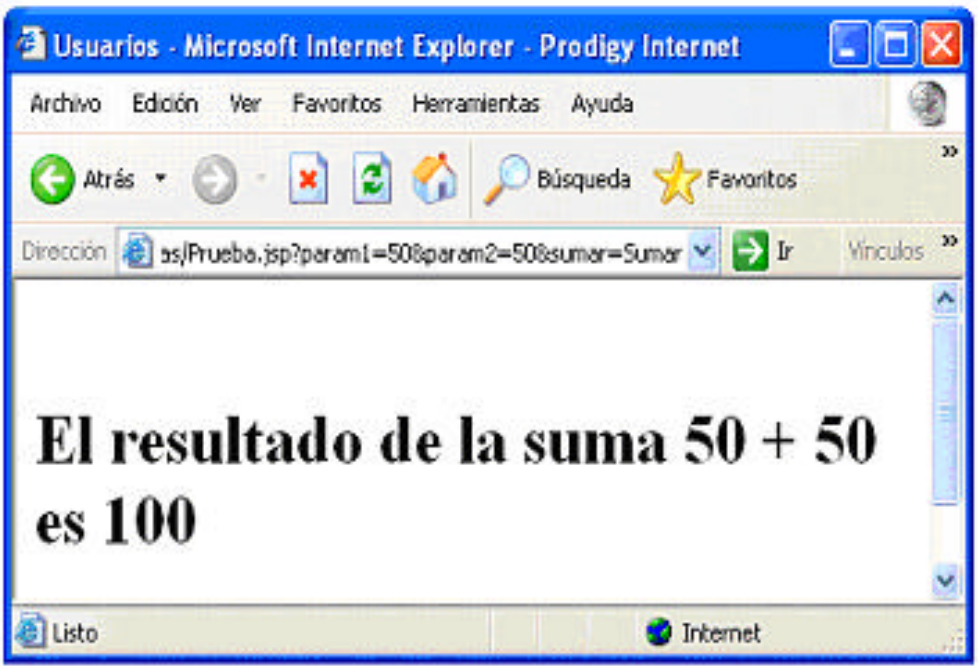
La siguiente pantalla muestra las aplicaciones existentes de la carpeta pruebas.



La que nos interesa, en este caso, es Prueba.html. En este formulario se introducirán los parámetros que utilizara el jsp para realizar la operación suma. El formulario se muestra a continuación:



Finalmente se obtiene el resultado de invocar al método suma del servicio web a través del jsp:



COMENTARIOS ACERCA DEL MANUAL

Este manual es una pequeña introducción a los servicios web. Ilustra con ejemplos sencillos el funcionamiento de los servicios web en diferentes plataformas y lenguajes de programación.

Explica detalladamente los requerimientos necesarios para el desarrollo de servicios web tanto en la plataforma .NET como en Java.

Muestra también el proceso de instalación de los diversos componentes requeridos para la creación de servicios web.

Finalmente, demuestra la interoperabilidad de los servicios web, con un servicio realizado en .Net que se accede mediante un cliente en java.

El contenido de este manual es muy fácil de entender y aplicar, el lector sólo debe contar con conocimientos básicos de programación y con el equipo y material suficiente para realizar las pruebas aquí mostradas.