Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias Y Sistemas

COMPARACIÓN DE LAS TECNOLOGÍAS .NET Y J2EE PARA EL DESARROLLO DE SERVICIOS WEB

Iván Nicolás García Solís

Asesorado por el Ing. José Ricardo Morales Prado

Guatemala, agosto de 2007

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

COMPARACIÓN DE LAS TECNOLOGÍAS .NET Y J2EE PARA EL DESARROLLO DE SERVICIOS WEB

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

IVÁN NICOLÁS GARCÍA SOLÍS

ASESORADO POR EL ING. JOSÉ RICARDO MORALES PRADO

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, AGOSTO DE 2007

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Inga. Glenda Patricia García Soria
VOCAL II	Inga. Alba Maritza Guerrero de López
VOCAL III	Ing. Miguel Ángel Dávila Calderón
VOCAL IV	Br. Kenneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Vides Leiva
SECRETARIA	Inga. Marcia Ivónne Véliz Vargas

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Jorge Armín Mazariegos Rabanales
EXAMINADOR	Ing. Freiry Javier Gramajo López
EXAMINADOR	Ing. Alfredo Valdés Matta
SECRETARIA	Inga. Marcia Ivónne Véliz Vargas

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

COMPARACIÓN DE LAS TECNOLOGÍAS .NET Y J2EE PARA EL DESARROLLO DE SERVICIOS WEB,

tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, en agosto de 2005.

IVÁN NICOLÁS GARCÍA SOLÍS

AGRADECIMIENTOS A:

DIOS	Por permitirme llegar a este día, conseguir este logro y poder compartirlo con mis seres queridos
MIS PADRES	Por su invaluable apoyo y esfuerzo para que completara con éxito esta etapa de mi vida
MIS AMIGOS	Por estar siempre presentes en los buenos momentos, así como en los más difíciles
MI ASESOR	Ing. Ricardo Morales, por su experiencia y tiempo compartido para la elaboración del presente trabajo

DEDICATORIA A:

DIOS	Porque sin ÉL nada de esto fuera posible
MIS PADRES	Nico y Loly, por enseñarme lo necesario para salir adelante y buscar ser una mejor persona cada día
MIS HERMANOS	Omar y Ana, ejemplos a seguir por todas las características que los hacen ser tan especiales y magnificas personas

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES V

GLOSARIO IX

RESUMEN XIII

OBJETIVOS XV

INTRODUCCIÓN XVII

1. PLATAFORMAS DE DESARROLLO 1

1.1 Microsoft Visual Studio .NET 1

1.1.1 Fundamentos del .NET Framework 4

1.1.1.1 *Common Language Runtime* 4

1.1.1.1.1 Compilación del *Managed Code* 5

1.1.1.1.2 Organizando el *Managed Code* 6

1.1.1.1.3 Ejecutando el *Managed Code* 7

1.1.1.2 *.NET Framework Class Library* 8

1.1.2 Lenguajes de programación 10

1.1.2.1 El lenguaje C# 10

1.1.2.2 Visual Basic .NET 12

1.2 *Java 2 Platform Enterprise Edition* 14

1.2.1 Lenguaje de programación Java 17

1.2.1.1 Organización de las clases Java 18

1.2.2 *Java Virtual Machine* 20

1.2.2.1 Componentes de la JVM 21

1.3 Las plataformas 22

2. DESARROLLO DE SERVICIOS WEB 25
2.1 *Web Services* 25
2.1.1 Las columnas que sostienen y forman los servicios Web 27
2.1.1.1 Describiendo la información a transmitir 28
2.1.1.2 Describiendo la comunicación y el acceso 29
2.1.1.3 Describiendo las capacidades y funciones 30
2.1.1.4 Localizando los servicios Web 32
2.2.2 Las tecnologías detrás de los servicios Web 32
2.2.2.1 XML 33
2.2.2.1.1 Estructura del XML 34
2.2.2.2 SOAP 37
2.2.2.3 WSDL 40
2.2.2.3.1 Estructura del WSDL 45
2.2.2.4 UDDI 48
2.2.3 Otras tecnologías aplicadas a los servicios Web 50
2.2.3.1 WS-I 52
2.2.3.1.1 *Basic Profile* 53
2.2.3.1.2 *Attachments Profile* 54
2.2.3.2 OASIS 55
2.2.3.2.1 *WS-Reliability* 55
2.2.3.2.2 *WS-Security* 56
2.2.4 Seguridad en los servicios Web 57
2.2.4.1 *Secure Socket Layer* y *Transport Layer Security* 59
2.2.4.2 *Firewalls* 60
2.2.4.3 Servicios Web seguros (*WS-Security*) 61
2.2.5 Los servicios Web y las plataformas de desarrollo 65
2.2.6 Los servicios Web y la plataforma .NET 65
2.2.6.1 .NET y SOAP 66
2.2.6.2 .NET y XML 67

2.2.6.3 .NET y WSDL 69
2.2.6.4 .NET y UDDI 70
2.2.6.5 *Web Services Enhancements* para Microsoft .NET 71
2.2.6.6 Seguridad en los servicios Web .NET 72
2.2.6.6.1 Credenciales de seguridad 72
2.2.6.6.2 Firma digital 74
2.2.6.6.3 Encriptación 75
2.2.7 Los servicios Web y la plataforma J2EE 77
2.2.7.1 J2EE y SOAP 78
2.2.7.2 J2EE y XML 80
2.2.7.3 J2EE y WSDL 81
2.2.7.4 J2EE y UDDI 82
2.2.7.5 *Java Web Services Developer Pack* 82
2.2.7.6 Seguridad en los servicios Web J2EE 83
2.2.7.6.1 Autenticación 84
2.2.7.6.2 Firma digital 84
2.2.7.6.3 Encriptación 88

3. CONSTRUCCIÓN DE SERVICIOS WEB 89
3.1 Servicios Web 89
3.1.1 Servicio Web en .NET 90
3.1.2 Servicio Web en J2EE 95
3.2 Aplicaciones cliente para los servicios Web 101
3.2.1 Aplicación cliente en .NET 101
3.2.2 Aplicación cliente en J2EE 102
3.3 Tecnologías servicios Web en .NET y J2EE 104
3.4 Ventajas y desventajas 106
3.4.1 Curva de aprendizaje y tiempo de desarrollo 107
3.4.1.1 Visual Studio .NET 108

3.4.1.2 *Java 2 Enterprise Edition* 109
3.4.2 Desempeño de los servicios Web 111
3.4.2.1 Prueba de desempeño servicio Web 113
3.4.2.1.1 Resultados prueba de desempeño 114

CONCLUSIONES 121
RECOMENDACIONES 123
BIBLIOGRAFÍA 125

ÍNDICE DE ILUSTRACIONES

FIGURAS

1	Esquema arquitectura plataforma .NET	3
2	*Namespace y assembly*	9
3	Esquema arquitectura aplicaciones Java	15
4	Documento XML	34
5	Documento DTD	35
6	Documento XML Schema	36
7	Mensaje SOAP	38
8	Llamada mensaje SOAP	39
9	Respuesta mensaje SOAP	39
10	Gramática documento WSDL	41
11	Continuación gramática documento WSDL	42
12	Ejemplo documento WSDL	44
13	Documento UDDI	49
14	Esquema general tecnologías servicios Web	51
15	Ejemplo mensaje SOAP utilizando *WS-Security*	64
16	Implementando credenciales de seguridad	73
17	Verificación de credenciales de seguridad	74
18	Firma digital del mensaje SOAP .NET	75
19	Encriptación del mensaje SOAP	76
20	Determinación si el mensaje SOAP está encriptado	76
21	Firma digital del mensaje SOAP J2EE	86
22	Validación de la firma digital	87
23	Archivo Service.asmx	90

24	Código fuente servicio Web .NET	91
25	Documento WSDL del servicio Web .NET	92
26	Continuación Documento WSDL del servicio Web.NET	93
27	Página inicial del servicio Web .NET	94
28	Página de prueba del servicio Web .NET	94
29	Respuesta del servicio Web .NET	95
30	Código fuente servicio Web J2EE	96
31	Archivo web.xml	97
32	Documento WSDL del servicio Web J2EE	98
33	Página Inicial del servicio Web J2EE	99
34	Página de prueba del servicio Web J2EE	100
35	Respuesta del servicio Web J2EE	100
36	Código fuente aplicación cliente .NET	102
37	Código fuente aplicación cliente J2EE	103
38	Solicitudes manejadas por unidad de tiempo .NET	115
39	Solicitudes manejadas por unidad de tiempo J2EE	115
40	Solicitudes completadas a lo largo del tiempo .NET	116
41	Solicitudes completadas a lo largo del tiempo J2EE	117
42	Tiempo de respuesta contra número de solicitudes .NET	118
43	Tiempo de respuesta contra número de solicitudes J2EE	118

TABLAS

I	Implementación de soluciones por plataforma	24
II	Implementación de tecnologías por plataforma	105
III	Ventajas o desventajas entre .NET y J2EE	106
IV	Características servidor de servicios Web	113
V	Parámetros prueba de desempeño	114
VI	Tiempo de repuesta (ms) contra número de solicitudes	117

GLOSARIO

API — *Application Programming Interface* (Interfaz de Programación de Aplicaciones) Conjunto de especificaciones de comunicación entre componentes de software, cuyo objetivo es abstraer la programación de rutinas.

Compilador — Programa especializado para la generación de código de bajo nivel, a partir de una entrada formada por un código fuente de un lenguaje de alto nivel.

DNS — *Domain Name System* (Sistema de Nombres de Dominio) Base de datos distribuida y jerárquica utilizada para la resolución de nombres de dominio a direcciones IP en una red de computadoras.

EAI — *Enterprise Application Integration* (Integración de Aplicaciones Empresariales) Utilización de los principios de arquitectura de computadoras y de software para la integración de los sistemas computacionales de una empresa.

E-business — *Electronic Business* (Negocios Electrónicos) Utilización de sistemas automatizados de información para la realización de procesos de negocios.

Escáner Programa especializado, parte de un compilador, encargado del reconocimiento de los componentes gramaticales de un código sobre la base de expresiones regulares.

Especificación Representa un documento técnico oficial que claramente establece términos y las reglas necesarias para la elaboración y utilización de algo.

Estándar Modelo o patrón de referencia a seguir, aceptado en términos generales por los involucrados.

Firewall (Cortafuegos) Dispositivo de hardware o software, dentro de una red de computadoras, encargado de prohibir ciertas comunicaciones sobre la base de las políticas de red establecidas.

Framework (Marco de Trabajo) Establece el ambiente, estructura, metodología y herramientas de trabajo necesarias para la elaboración de una actividad o producto.

Hardware Todo componente físico en el ambiente computacional.

HTTP *HyperText Transfer Protocol* (Protocolo de Transferencia de Hipertexto) Protocolo utilizado para la comunicación en la capa de Aplicación del modelo OSI, utilizado comúnmente en la Internet para la transmisión de páginas Web.

IANA *Internet Assigned Number Authority* (Autoridad de Asignación de Números de Internet) Registro central de los protocolos de Internet, codificación de puertos, códigos, etc. Sustituida por ICANN en 1998.

Internet Conjunto de redes de computadoras a escala mundial que utiliza ciertos protocolos para la comunicación y transmisión de datos, proveedora de distintos servicios, ejemplo WWW.

ICANN *Internet Corporation for Assigned Names and Numbers* (Corporación de Internet para la Asignación de Nombre y Números) Sucesora de la IANA para el manejo de la clasificación de estándares, protocolos, códigos, etc.

ISO *Internacional Organization for Standarization* (Organización Internacional para la Estandarización) Entidad no gubernamental para la elaboración de normas internacionales referentes a la industria y sus procesos.

MIME *Multipurpose Internet Mail Extensions* (Extensiones de Correo Internet Multipropósito) Especificación para el intercambio de contenido, archivos de cualquier tipo, a través de Internet de forma transparente.

Modelo OSI *Open System Interconecction* (Interconexión de Sistemas Abiertos) Modelo de red descriptivo, creado por la ISO, dividido en siete capas. Cada capa describe los estándares necesarios para la comunicación entre capas y con las capas superior e inferior.

Parser	Analizador Sintáctico, programa especializado, parte de un compilador, encargado de reconocer si una serie de componentes gramaticales forman parte de un lenguaje o gramática definida.
PKI	*Public Key Infrastructure* (Infraestructura de Clave Pública) Combinación de hardware y software, políticas y procedimientos que permiten asegurar el intercambio de datos usando criptografía pública.
Protocolo	Conjunto de reglas que definen la secuencia de sucesos que se deben llevar a cabo para establecer y mantener la comunicación entre entidades.
Software	Componentes intangibles en el ambiente computacional, por ejemplo, programas, sistema operativo, etc.
SSL	*Secure Socket Layer* (Capa de Conexiones Seguras) Protocolo para el establecimiento de conexiones seguras, funcionando a través de criptografía aplicado a los protocolos de la capa de aplicación.
TLS	*Transport Layer Security* (Seguridad de la Capa de Transporte) Protocolo que funciona bajo los mismos principios que SSL, pero aplicando dicha seguridad también a nivel de capa de transporte.
WWW	*World Wide Web*. Sistema de hipertexto que funciona sobre Internet para la visualización del contenido expuesto en ella.

RESUMEN

La comunicación ha sido vital para la evolución de la humanidad, desde los primeros mecanismos de comunicación por medio de señas hasta nuestros días con la utilización del lenguaje, ésta ha ido evolucionando y con ella los medios por los cuales se transmite. Este proceso también se ve reflejado en los sistemas informáticos, más específicamente entre los diferentes programas que se pueden estar ejecutando en dispositivos computacionales distantes entre sí. Los servicios Web surgen como un punto culminante en la evolución de la comunicación entre aplicaciones utilizando diferentes tecnologías por medio de Internet.

Los Servicios Web o *Web Services* se pueden definir como un conjunto de aplicaciones y protocolos, que desarrollan alguna actividad y permiten la comunicación entre aplicaciones sobre una red de computadoras. Los servicios Web hacen uso de cuatro tecnologías estándar dentro de Internet para posibilitar la comunicación entre las aplicaciones: XML, SOAP, WSDL y UDDI. Estas tecnologías definen un marco de trabajo para el establecimiento de la comunicación entre las diferentes aplicaciones, por medio de la descripción de la información a transmitir (XML), describiendo el método de comunicación (SOAP), definiendo la funcionalidad de la comunicación (WSDL), y permitiendo un método de localización entre las aplicaciones (UDDI).

Para trabajar con los servicios Web, es decir, desarrollar las actividades de construcción e implementación, es necesario un conjunto de herramientas. Las plataformas de desarrollo proveen estas herramientas, además de los recursos necesarios para el análisis, diseño, construcción e implementación de los mismos y de otros distintos tipos de aplicaciones. En el presente trabajo se realiza una comparación de dos plataformas distintas para el desarrollo de servicios Web: *Microsoft .NET* y *Java 2 Enterprise Edition*. En los primeros capítulos se desarrolla cada una de las plataformas y luego se introducen los conceptos relacionados con los servicios Web y cómo son implementados.

Ambas plataformas presentan las herramientas necesarias para el diseño, desarrollo, implementación y utilización de servicios Web complejos y altamente inter-operables, la selección de una u otra alternativa dependerá de la experiencia previa y de las habilidades existentes para trabajar con cada una de ellas. De igual forma, la diferencia en el desempeño de los servicios Web, creados en una u otra plataforma, dependerá de las mejores prácticas aplicadas en su construcción.

OBJETIVOS

- **General**

Establecer las ventajas y desventajas que presentan las tecnologías utilizadas por las plataformas .NET y J2EE para el desarrollo de servicios Web.

- **Específicos**

1. Determinar las tecnologías de las cuales hace uso cada plataforma para el desarrollo de servicios Web.
2. Determinar las herramientas que pone a disposición cada plataforma para el desarrollo de servicios Web.
3. Definir las características y particularidades que brindan las tecnologías de cada plataforma para el desarrollo de servicios Web.
4. Comparar la construcción de un servicio Web de ejemplo y su funcionamiento en ambas plataformas.
5. Presentar la confrontación del proceso de construcción y funcionamiento del servicio Web en cada plataforma.

INTRODUCCIÓN

Los medios de comunicación han servido para que el ser humano pueda estar enterado de los acontecimientos que suceden tanto alrededor de él como a la distancia. Y esa misma funcionalidad es la que ha potenciado la evolución de los mecanismos de comunicación hacia lo que hoy son. Ejemplo de ello son la Internet y la *World Wide Web*, brindando acceso globalizado a la información.

Así como los seres humanos vieron la necesidad de comunicarse entre sí, los programas de computadora que utilizamos lo hicieron. Los servicios Web vienen a suplir esa necesidad de una forma natural y acorde a las tecnologías por las cuales se realizará dicha comunicación: la Internet. Los servicios Web hacen uso de protocolos estándares de Internet para la comunicación entre aplicaciones, su desarrollo se basa en cuatro tecnologías: XML, SOAP, WSDL y UDDI. Entre las herramientas destacadas para el desarrollo de servicios Web se encuentran dos plataformas, estas plataformas se han consolidado como dos paradigmas para la creación de aplicaciones integrales.

Microsoft Visual Studio .NET y *Java 2 Platform Enterprise Edition* son, por el momento, las tecnologías de desarrollo más utilizadas para la creación de un sinnúmero de aplicaciones y entre ellas los servicios Web. Implementando y potenciando, cada una a su manera, las tecnologías utilizadas por los servicios Web, manteniendo la interoperabilidad de los mismos y dándole un valor agregado. Permitiendo la creación de servicios Web más seguros, confiables, altamente interoperables y funcionales. Determinar qué ventajas ofrece cada plataforma es vital para la correcta utilización y desarrollo de los servicios Web.

1. PLATAFORMAS DE DESARROLLO

Para el proceso de desarrollo e implementación que conlleva el trabajar con servicios Web y demás aplicaciones es necesario contar con un ambiente adecuado. Las plataformas de desarrollo proveen ese ambiente, además brindan las herramientas y los recursos necesarios para el análisis, diseño, construcciones, implementación y pruebas de distintos tipos de aplicaciones y entre ellas los servicios Web. El presente capitulo muestra como es la arquitectura de las plataformas de desarrollo más significativas, .NET y J2EE, como es el funcionamiento para el desarrollo de aplicaciones y cuales son las facilidades que otorgan al trabajar con los servicios Web.

1.1 Microsoft Visual Studio .NET

Microsoft Visual Studio .NET (o solo .Net) es la plataforma de desarrollo que Microsoft Corporation ha creado para introducir diferentes y nuevas tecnologías, que permitan el desarrollo de aplicaciones y servicios más complejos y que apunten a soluciones de gran envergadura. Desarrollado sobre la base de los estándares de servicios Web XML, .Net busca que los sistemas y las aplicaciones puedan comunicarse para transferir información independientemente del sistema operativo, lenguaje de programación o hardware donde se estén ejecutando.

El kit de desarrollo de .Net es llamado *.NET Framework SDK*, el cuál incluye las herramientas necesarias para el desarrollo, ejecución y distribución de aplicaciones de todo tipo: de ventanas, de consola, de Web, etc. .Net permite extrapolar todos los conceptos de elaboración de aplicaciones a Internet, para brindar servicios que puedan ser accesados desde cualquier lugar y por cualquier medio.

El núcleo de la plataforma .Net es el *.NET Framework*, componente esencial para la creación y ejecución de las aplicaciones y servicios Web XML que la plataforma permite. Los componentes principales del *.NET Framework* son el *Common Language Runtime* (CLR) y la librería *.NET Framework Class o Base Class Library* (BCL), los cuáles son la base de todas las posibles aplicaciones que tiene .NET.

El CLR provee un conjunto común de tipos de datos y otros servicios que pueden ser usados por todos los lenguajes soportados por el *.NET Framework*. Y la librería *.NET Framework Class* incluye una gran cantidad de clases estándar y otros tipos de clases que pueden ser usadas por cualquier aplicación creada en cualquier lenguaje soportado por el *.NET Framework*. Con estos dos componentes el *.NET Framework* establece:

- Un entorno coherente de programación orientada a objetos, en donde el código de los objetos se puede almacenar y ejecutar de forma local, de forma remota o de forma local pero distribuida en Internet.

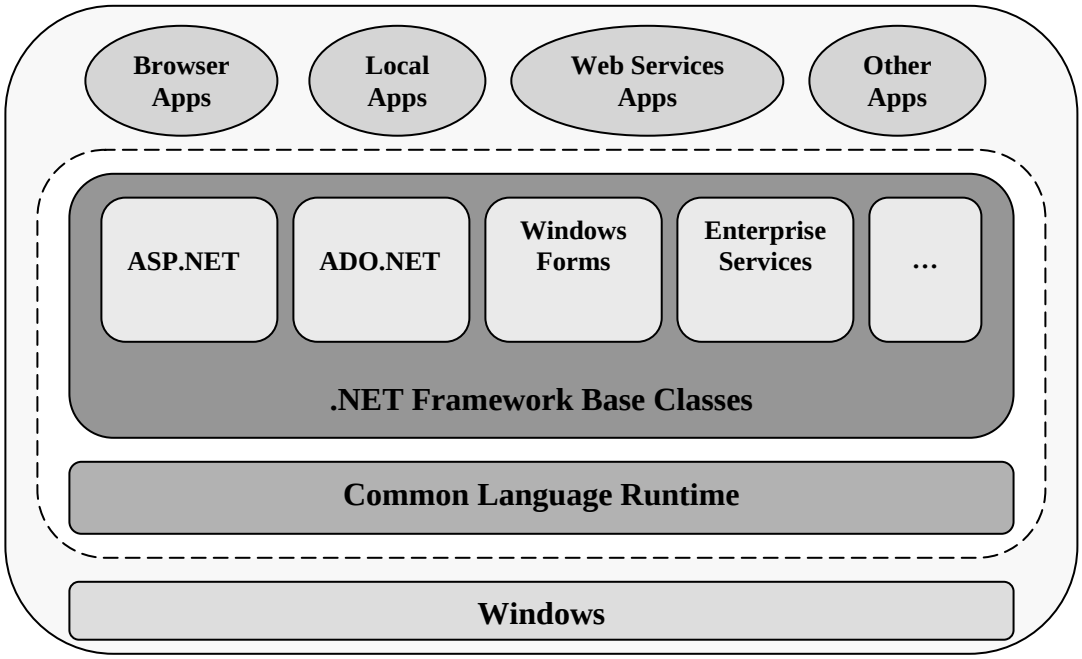
- Un entorno de ejecución de código que reduce lo más posible la implementación de software y el conflicto de versiones (el conflicto se da cuando una aplicación no funciona correctamente ya que sus componentes son de versiones diferentes).

- Un entorno de ejecución que permite el manejo de versiones en cada componente que integre una aplicación, haciendo uso de un número de versión compuesto físicamente por cuatro números. Los cuales permiten definir incompatibilidad entre versiones o que se realizaron pequeños cambios.

- Un entorno de ejecución de código que elimina los problemas de rendimiento de los entornos en los que se utiliza secuencias de comandos o interpretes de comandos.

- Se basa la comunicación en estándares del sector para asegurar que el código de *.NET Framework* se pueda integrar con otros tipos de código.

Figura 1 Esquema arquitectura plataforma .NET



En la Figura 1 se muestra la arquitectura sobre la cual se basa la plataforma .NET para el desarrollo de las aplicaciones. Parte fundamental de la misma es el *.NET Framework*, remarcado en la figura con un rectángulo de línea discontinua. A continuación se presenta más detalladamente los componentes y las funciones que realiza el *.NET Framework*.

1.1.1 Fundamentos del .NET Framework

1.1.1.1 *Common Language Runtime*

El CLR como se ha mencionado antes es la base para cualquier aplicación creada con el *.NET Framework*, no importando en que lenguaje se haya elaborado la aplicación. Además es la base también para la librería *.NET Framework Class*. Es el motor de tiempo de ejecución que administra el código en tiempo de ejecución y presta servicios centrales.

A parte de proveer el conjunto de tipos de datos, como son enteros, cadenas, clases, interfaces, etc. También provee varios servicios que dan mayor funcionalidad y eficiencia a las aplicaciones, administración de memoria, ejecución de subprocesos, ejecución de código, comprobación de la seguridad del código y compilación. A las aplicaciones construidas sobre el CLR se les llama *managed code* (código administrado), y el CLR provee fundamentalmente un conjunto estándar de tipos de datos usados por los lenguajes soportados por el CLR, un formato estándar de metadata para almacenar la información respecto a los tipos de datos usados, tecnología para empaquetar código administrado y un ambiente de ejecución para el código administrado.

Para que el CLR pueda soportar diferentes lenguajes es necesario que se encuentre la sintaxis de los lenguajes separados de la semántica. Esto lo logra el CLR por medio del *Common Type System* (CTS), el cual no especifica una sintaxis en particular sino que define un conjunto común de tipos que puede ser usado por cualquier sintaxis, así si el lenguaje es construido sobre el CLR, utilizará los tipos definidos por el CTS. Para permitir que diferentes lenguajes puedan tener inter-operabilidad, es decir, poder usar clases de un lenguaje en otro, deben compartir ciertos tipos definidos por la *Common Language Specification* (CLS). La CLS define un subconjunto del CTS que deben de implementar los lenguajes que deseen inter-operar con otros lenguajes que cumplen con la CLS.

1.1.1.1.1 Compilación del *Managed Code*

Al compilarse el código administrado se generan dos productos, uno de ellos es un conjunto de instrucciones llamadas *Microsoft Intermediate Language* (MSIL o CIL, *Common Intermediate Language*) y la metadata, que es información acerca del antes mencionado conjunto de instrucciones y los datos que manipulan.

No importa en que lenguaje soportado por el CLR esté escrito el código administrado, la compilación de éste transformará todo el código siempre en CIL y metadata. El CIL y la metadata son almacenados en un archivo ejecutable portable (PE por sus siglas en inglés) estándar de Windows, este archivo puede ser un DLL o un EXE, comúnmente se le llama módulo.

El CIL es el único código que puede entender el CLR, por ello todos los lenguajes basados en CLR generan CIL, es decir el CIL es el lenguaje ensamblador del CLR, cabe mencionar que este código no es código de máquina. La principal ventaja del CIL es que facilita la ejecución multiplataforma[1] y la integración entre los lenguajes soportados por el CLR, ya que, como cualquier código intermedio, es independiente del procesador.

La metadata, que es información que describe al CIL generado, contiene los nombre de los tipos, información sobre la herencia de los tipos, las interfaces que implementen los tipos, los métodos que implementen los tipos, las propiedades que usen los tipos y los eventos que tengan los tipos entre otra información. Además también incluye atributos, que son valores almacenados en la metadata que permiten controlar varios aspectos de la ejecución del código.

1.1.1.1.2 Organizando el *Managed Code*

Para el desarrollo, distribución y ejecución de las aplicaciones desarrolladas con *.NET Framework* es necesario un medio de encapsulación del código ejecutable, un *assembly*, que se define como la agrupación de archivos que conforman una unidad lógica de funcionalidad. Una aplicación puede tener sus recursos depositados en uno o varios *assemblies*. Los *assemblies* incluyen entre sus archivos un manifiesto, el metadata del *assembly*, que contiene la información acerca de los módulos y demás archivos que lo conforman.

[1] .NET permite, por el momento, la construcción y ejecución de sus aplicaciones únicamente sobre plataformas Windows (32 o 64 bits). Sin embargo gracias a que se han establecido CIL y C# como estándares bajo ECMA (334 y 335 respectivamente) e ISO/IEC (23271:2003 y 23270:2003 respectivamente); terceros han realizado su implementación, siendo Mono y DotGNU dos proyectos *open source* que permiten la construcción y ejecución de aplicaciones .NET en diversos sistemas operativos no Windows.

El manifiesto contiene entre otra información el nombre del *assembly*, el número de versión del *assembly*, la lista de los archivos que contiene y de que otros *assemblies* depende.

1.1.1.1.3 Ejecutando el *Managed Code*

Para la ejecución de las aplicaciones creadas utilizando el *.NET Framework*, se debe cargar los *assemblies* que conforman la aplicación, los *assemblies* son cargados a la memoria hasta que es necesitado alguno de los métodos que hay en ellos. Para la ejecución de la aplicación el CIL no puede ser ejecutado solo así, debe ser nuevamente compilado para generar el código de máquina específico del procesador donde está la aplicación. El proceso de compilar el CIL puede ser de dos formas: compilando cada método a la vez según cuando se ejecuta o se puede compilar todo el código de una sola vez antes de ejecutar el *assembly*.

A la acción de compilar los métodos la primera vez que son llamados se le llama compilación *just-in-time* (JIT), y se refiere a que se compilan únicamente los métodos llamados antes de ser ejecutados. Un método siempre se ejecuta en código nativo, no en CIL, por ello la compilación JIT. A la acción de compilar todo el código CIL del *assembly* antes de ejecutarlo se le llama crear una imagen nativa, que no es más que transformar todo el código CIL a código de máquina por medio del *Native Image Generador* (NGEN).

La mayor diferencia presentada entre la compilación JIT y la creación de Imagen Nativa es el momento en el que son compilados los métodos. La compilación JIT ofrece la ventaja de compilar únicamente el método que es llamado, evitando así, compilar los demás métodos que pueden no ser llamados. El método que es llamado es compilado, aplicándole optimizaciones propias de la arquitectura donde se está ejecutando, y el código generado es almacenado en memoria permitiendo así una rápida ejecución la próxima vez que sea necesitado.

La creación de una imagen nativa es un proceso por separado, lo cual indica, que se crean *assemblies* que contienen el código nativo en lugar de CIL. Al estar todo el código compilado, no es necesario compilarlo de nuevo al momento de ser llamado, por lo cual es más rápida la ejecución inicial de los métodos. Las desventajas del uso de NGEN (imagen nativa) es que los *assemblies* originales son igualmente necesarios por la metadata que contienen y más importante aún las imágenes nativas no pueden saber el estado del ambiente de ejecución del CLR, por lo cual las referencias sobre la memoria tendrán que volver a realizarse si es necesario.

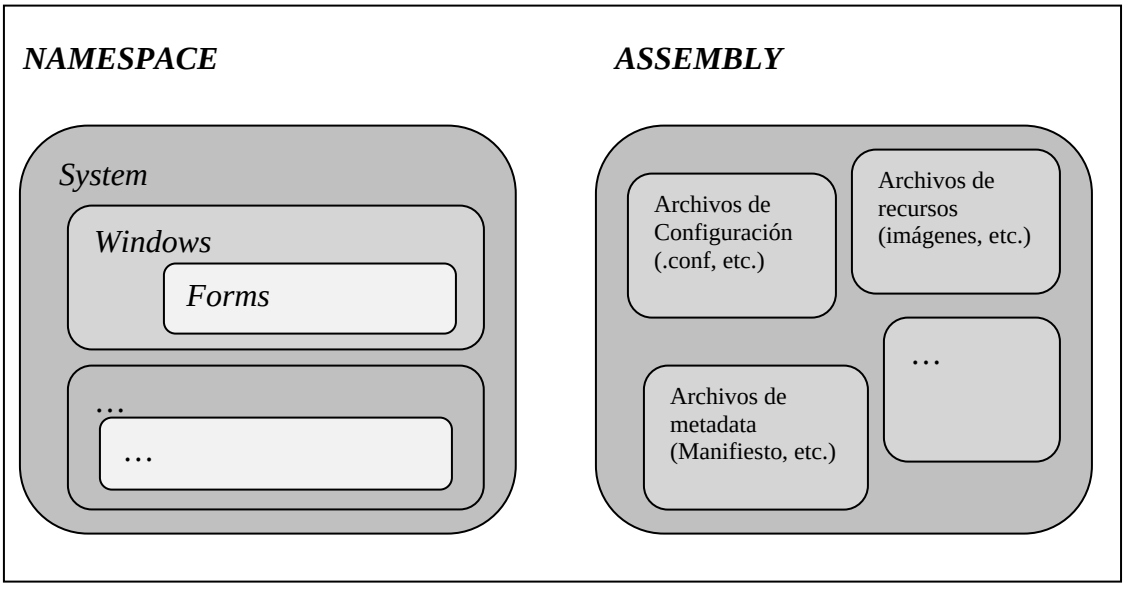
1.1.1.2 *.NET Framework Class Library*

La *.NET Framework Class Library*, también llamada *Base Class Library* (BCL), es una librería de clases, sobre la cual se pueden desarrollar todo tipo de aplicaciones. Además de la gran cantidad de clases que trae predefinidas, se puede hacer uso de ellas para la creación de nuevas clases y métodos. Debido a la gran cantidad de clases que posee la *.NET Framework Class Library*, su organización es como un árbol jerárquico, donde cada nodo es llamado *Namespace*.

Un *Namespace* es como un contenedor de clases, interfaces y otros *namespaces*, esto quiere decir que engloba cierta funcionalidad para un uso más ordenado de lo servicios que esta librería presta. La raíz del árbol es el *Namespace System*, éste incluye los tipos de datos estándares y fundamentales, de éste se derivan los demás *namespaces* principales, los cuales van especificando más su funcionalidad.

Los *namespaces* a diferencia de los *assemblies* son organizaciones lógicas de clases e interfaces, siendo así, se puede agrupar varios namespaces en un solo *assembly* o repartir un *namespace* en varios *assemblies*. Los *assemblies* por su parte son organizaciones lógicas de archivos, es decir un archivo (.dll o .exe) que contiene los archivos necesarios para la ejecución de la aplicación. La Figura 2 muestra una perspectiva más clara de las diferencias existentes entre *namespaces* y *assemblies*.

Figura 2 ***Namespace y assembly***



1.1.2 Lenguajes de programación

El *.NET Framework* fue construido pensando en soportar múltiples lenguajes arriba de él, es decir, permitir la codificación de instrucciones en varios lenguajes de programación y que estos pudieran inter-operar. Para conseguir esto, los lenguajes de programación deben ser construidos sobre la base del *Common Languaje Runtime* y conforme a las especificaciones del *Common Type System*.

Entre los lenguajes de programación soportados por .NET están C# (*C Sharp*) y VB .NET (*Visual Basic*), ambos fueron creados junto con el .NET Framework y son los más utilizados. Además de estos dos lenguajes varias empresas han desarrollado lenguajes compatibles con el *.NET Framework* y es posible utilizarlos en la construcción de aplicaciones. A continuación se presenta un breve análisis de los lenguajes más sobresalientes en .NET.

1.1.2.1 El lenguaje C#

C# (pronunciado C Sharp) es el leguaje de programación insignia de .NET, ya que fue creado específicamente para su utilización sobre el *.NET Framework*. Ahora C# ya es un estándar (ECMA-334 "Especificación del Lenguaje C#") y por lo tanto se pueden crear compiladores de C# independientes del *.NET Framework*, es decir que generen código para otras plataformas.

El lenguaje de programación C# se puede definir como la evolución de C++, es decir, se han tomado características del lenguaje C++ y del lenguaje Java, para la creación de un lenguaje orientado a objetos de alto nivel con sintaxis que parece de bajo nivel. Con una sintaxis muy parecida a C++ y Java, se permite que el programador se encuentre familiarizado con el lenguaje aunque no lo haya usado anteriormente.

Algunas características que hacen de C# una poderosa herramienta de programación son:

- Soporte completo para la integración de sistemas existentes que utilizan la tecnología COM (*Component Object Model*), el modelo de componentes basado en interfaces que permite la comunicación, a través de métodos, de objetos de naturaleza independiente.

- Manejo automático de la memoria de forma robusta a través del *Garbage Collection*, pero permitiendo el uso de "código no seguro", esto es, instrucciones para manejar la memoria parecido al uso de punteros de C++.

- Implementación de clases ADO (*Active Data Object*) para el acceso a fuentes de datos como lo son las bases de datos y la representación de los datos por medio de XML (eX*tensible Markup Language*).

- Soporte de todas las características del *.NET Framework* e inter-operabilidad con todos los lenguajes que también sean basado en el CLR, además de versionamiento para facilitar la administración y el *deploy*[2] de aplicaciones.

- La distribución y el *deploy* de aplicaciones es más fácil olvidándose del llamado "infierno de los DLLs", ya que el autocontenido de la meta-data en los *assemblies* permite la facilidad del versionamiento además del proceso de instalación llamado XCOPY, sin necesidad del registro de componentes.

1.1.2.2 Visual Basic .NET

Visual Basic .NET es la evolución del lenguaje de programación Visual Basic, de sintaxis amigable y alta aceptación entre los programadores. Visual Basic .NET es un lenguaje orientado a objetos que hace uso de toda la funcionalidad del *.NET Framework* y que permite el desarrollo de aplicaciones poderosas al igual que su hermano C#. Ambos lenguajes, y en sí, todos los lenguajes que se basen en el CLR para el *.NET Framework* generan el mismo código intermedio, CIL, por lo tanto las diferencias se presentan de forma evidente solo en su sintaxis. La mejor forma de notar dicha diferencia es aprender los fundamentos del *.NET Framework* ya que es la base de ambos lenguajes.

[2] *Deploy*: termino utilizado para indicar el conjunto de actividades necesarias para la puesta en marcha de las aplicaciones.

Todas las características presentes en el *.NET Framework* y las discutidas en el apartado del lenguaje C# son aplicables a Visual Basic .NET, ya que ambos lenguajes utilizan las clases definidas en la *.NET Framework Class Library*. De esta forma permiten no solo la creación de aplicaciones robustas sobre plataformas fijas, sino que permiten la creación de aplicaciones para dispositivos móviles a través de WML (*Wireless Markup Language*), WAP (*Wireless Access Protocol*), cHTML (*compact HTML*) y demás tecnologías móviles. Así mismo, hace uso de los estándares de Internet para la creación de servicios Web, unidades de programación disponibles en Internet.

Al conocer la base de la plataforma .NET y como los lenguajes hacen uso de los recursos del *.NET Framework*, además que para ser soportados por éste, deben generar código intermedio (CIL) y conformarse según el CTS, es fácil identificar que todos los lenguajes funcionan de la misma forma. Siendo la mayor diferencia entre ellos la sintaxis y alguna que otra funcionalidad muy especializada, por ejemplo: C# permite el uso del llamado “código inseguro”, que es el uso de instrucciones para el manejo de memoria (punteros); mientras que VB .NET no lo permite, evitando de este modo la aparición de errores de direccionamiento de memoria.

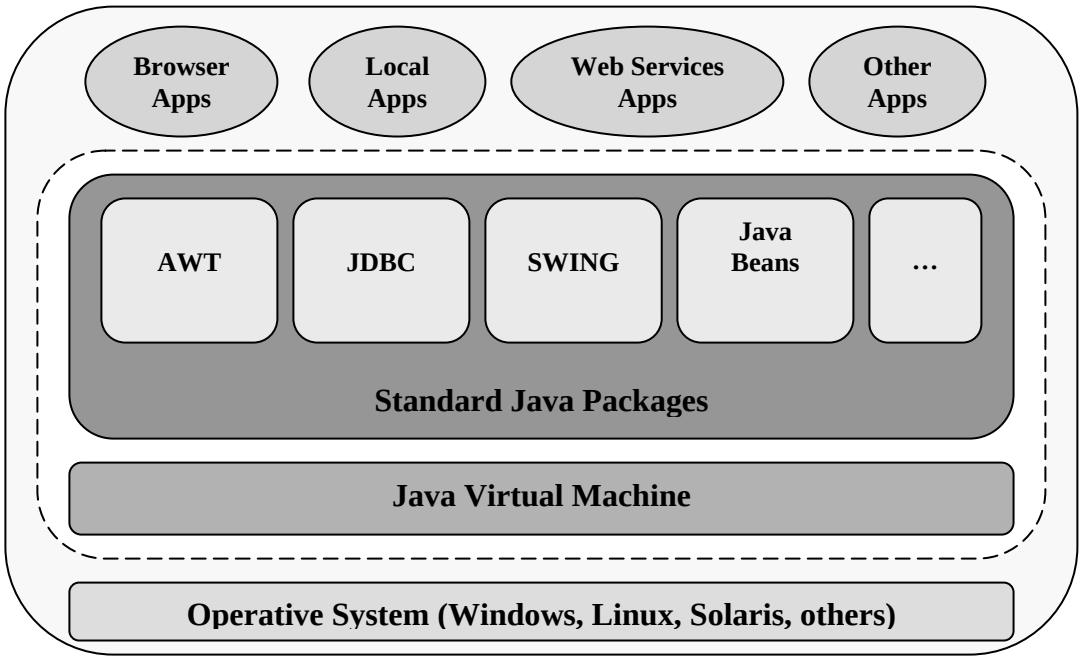
El estar basados sobre .NET Framework permite incluso la generación de traductores de código, es decir, programas que permiten pasar el código fuente de una aplicación realizada en C# a VB .NET y viceversa; dicha traducción, dependiendo de la complejidad de la aplicación, es completamente funcional sin realizar ningún cambio. Un ejemplo de dichos programas es *EndBracket* desarrollado por John Robbins y presentado en la publicación *MSDN Magazine* de Microsoft en Agosto de 2004. Así mismo, hay muchas más herramientas e inclusive hay traductores de código en línea.

1.2 *Java 2 Platform Enterprise Edition*

Java 2 Platform Enterprise Edition (o solo J2EE), referido de ahora en adelante solamente como J2EE, es la plataforma presentada por Sun Microsystems basada en el lenguaje Java para el desarrollo de aplicaciones empresariales multicapas. Como lo indican las iniciales en inglés es la Plataforma Edición Empresarial de Java 2, y la conforman una serie de componentes, servicios y protocolos, que permiten la construcción de aplicaciones sobre estándares de la industria. Siendo así J2EE definido como un estándar.

J2EE brinda de este modo la posibilidad de hacer aplicaciones multicapas, centradas en servidor, orientadas al Web, de forma rápida y segura, añadiendo estabilidad, escalabilidad, portabilidad e integración con demás aplicaciones, fuentes de datos y sistemas no desarrollados sobre Java. En la Figura 3 se muestra el esquema para entender como se establece la arquitectura dentro de la plataforma J2EE, para las distintas aplicaciones que se pueden construir.

Figura 3 Esquema arquitectura de aplicaciones Java



La plataforma J2EE cubre todos los aspectos de la programación multicapas, y se centra en brindar componentes y servicios que permitan encapsular la funcionalidad del sistema en cada una de ellas. Por lo tanto al hablar de la arquitectura de desarrollo sobre J2EE se incluye una gran cantidad de conceptos, de los cuales los más importantes son:

- Componentes, son todos aquellos elementos creados para el funcionamiento de la lógica del negocio, abarca desde applets, páginas JSP, Servlets, JavaBeans y todo aquello creado por lo programadores.

- Contenedores, son los proveedores de servicios que permiten el acceso de los clientes a los componentes, soportan de forma transparente transacciones y manejo de recursos. La principal característica es que permiten una administración de las aplicaciones fuera de las mismas aplicaciones.

- Conectores, son los encargados de permitir la flexibilidad de comunicación entre los componentes y demás sistemas, mediante la implementación de diversos servicios, por debajo de la plataforma J2EE.

Las tecnologías de las cuales hace uso J2EE se encuentran en forma de componentes, los cuales brindan los APIs (*Application Programming Interface*) necesarios para su implementación e integración en los sistemas. Algunas de las tecnologías más relevantes encontradas en J2EE son:

- Conectividad de forma transparente a bases de datos y otros sistemas de información por medio de JDBC (*Java Database Connectivity*) y CORBA (*Common Object Request Broker Architecture*) permitiendo la integración de más sistemas de información.

- *Enterprise JavaBeans* (EJB), contenedor encargado de manejar las transacciones, hilos, comunicación, y escalabilidad, en un ambiente distribuido, así como los Servlets encargados de brindar la infraestructura para los componentes, comunicación y manejo de sesiones en un contenedor. El equivalente de los EJB en la plataforma .NET son los servicios COM+. También existe la tecnología Hibernate para Java y NHibernate para .NET para manejar la persistencia de información.

- Independencia de plataforma, como se menciono antes la piedra angular de J2EE y fundamento es Java, cuya forma de trabajo es sobre una máquina virtual, obteniendo así la posibilidad de correr en múltiples plataformas de forma independiente.

- *Java API for XML-Based RPC* (JAX-RPC), que permite la construcción y el *deploy* de servicios Web utilizando SOAP (*Simple Object Access Protocol*) y demás estándares para la inter-operabilidad con clientes heterogéneos.

El kit de desarrollo de J2EE se llama J2EE 1.4 SDK, y permite la construcción e implementación de todas las aplicaciones y tecnologías descritas anteriormente.

1.2.1 Lenguaje de programación Java

El lenguaje de programación Java, único lenguaje sobre el cual se ha definido el J2EE, es un lenguaje orientado a objetos que posee gran cantidad de clases ya construidas, y sobre las cuales se pueden construir otras permitiendo así la implementación de grandes y complejas soluciones. Otras características sobresalientes de Java son su portabilidad, antes mencionada, y el ser un lenguaje dinámico debido a que carga únicamente las clases que necesita para su ejecución y lo hace conforme sean necesarias.

La forma de generar una aplicación en Java empieza con la creación de un archivo fuente, éste contiene instrucciones con la sintaxis de Java además de la definición de las clases a utilizar. También puede contener la inclusión de clases ya definidas y es aquí donde entra el poder de extensibilidad de Java, ya que las clases se pueden implementar sin mayores cambios o bien se pueden heredar a clases nuevas creadas en el programa. Las clases se pueden agrupar en paquetes obteniendo de este modo cierta agrupación funcional según se requiera. El código fuente se compila por medio del compilador de Java, *javac*, y se genera un archivo *class*, dicho archivo contiene código intermedio de Java llamado *bytecode*.

El código intermedio o *bytecode* es independiente de plataforma, es decir puede ser ejecutado en cualquier arquitectura de procesador y sobre cualquier sistema operativo, siempre y cuando exista una máquina virtual para dicho escenario. El *bytecode* generado sobre cualquier sistema operativo será siempre el mismo. La máquina virtual de Java es la encargada de ejecutar el código que se encuentra en *bytecode* ya que permite la compilación según el procesador donde se encuentre.

1.2.1.1 Organización de las clases Java

Java permite la agrupación de diferentes archivos *class* y otros recursos de lo cuales haga uso la aplicación en otro tipo de archivos, que brindan un formato de compresión y nivel de agrupación que permite una mejor distribución y administración. Algunos tipos de archivos son:

- JARS (*Java Archives*), permite agrupar archivos *class* otorgando compresión y reduciendo la carga de las mismas. Además puede incluir imágenes, archivos de sonido y demás recursos utilizados por las clases. Soporta el uso de firmas digitales brindando así seguridad al momento de su uso sobre Internet.

- WARS (*Web Archive*), permite agrupar archivos *class*, páginas jsp, html, css y otros documentos utilizados en una aplicación Web, ejecutada por medio de los Servlet Engines. También incluye un descriptor Web para el *deploy*.

  La estructura de directorio de un WAR incluye en la raíz todas las páginas html, jsp, css y demás recursos para el *front-end*. El archivo web.xml dentro del directorio WEB-INF contiene elementos de configuración, seguridad de la aplicación y otros detalles sobre el *deployment*. En el directorio WEB-INF se encuentran dos directorios más: classes, contiene las clases Java empleadas en las páginas jsp y los Servlets; el directorio lib, contiene los archivos jar utilizados en la aplicación.

- EJB-JAR (*Enterprise JavaBean JAR*), tiene funcionalidad similar a los WAR pero para el desarrollo de *Enterprise JavaBeans*, que son componentes ejecutables dentro de un Contenedor que brindan una funcionalidad completa, contiene uno o más *enterprise beans* y un descriptor de EJB para el *deploy*.

La estructura de directorio de un EJB-JAR incluye en la raíz todas las clases que forman el *Enterprise JavaBean*. El directorio META-INF contiene el descriptor para el *deploy* y otros archivos de configuración utilizados por el Contenedor de EJB.

- EARS (*Enterprise Archives*), es la agrupación de un WAR y un EJB-JAR para su utilización dentro de un *Application Server*, que es un servidor de aplicaciones con los contenedores necesarios para la ejecución de aplicaciones empresariales Java.

Todos lo archivos mencionados anteriormente se encuentran en el formato jar, por lo tanto presentan las características de este tipo de archivo. Entre estas características, como se menciono antes, se encuentra la compresión de archivos y el soporte para firmas digitales, obteniendo así una gran funcionalidad para el ambiente Web.

1.2.2 *Java Virtual Machine*

La *Java Virtual Machine* (JVM) o Máquina Virtual de Java es la encargada de realizar la ejecución del código de Java. Cuando se habla de la máquina virtual de Java, hay que tener presente que pueden haber pequeñas variaciones, ya que cualquier fabricante puede implementar la especificación de la JVM. También se encuentra el llamado *Java Runtime Environment* (JRE) o Ambiente de Ejecución de Java, él cual no es más que la JVM y las clases que conforman el núcleo de Java, además de otros archivos de soporte. De esta forma el JRE es más completo y orientado hacia la redistribución por parte de desarrolladores.

La máquina virtual de Java, como se ha indicado anteriormente, es el soporte para la ejecución de las aplicaciones desarrolladas en Java, y está conformada por varios componentes que son los que proporcionan las características de portabilidad, robustez, estabilidad y seguridad a la plataforma Java. La ejecución de las aplicaciones Java se puede realizar de dos formas, una es interpretando las instrucciones del archivo *class*, es decir el *bytecode*. Y la segunda forma es compilando los métodos necesarios una única vez y conforme sean solicitados, lo que es llamado la compilación justo a tiempo o compilación *just-in-time* (JIT).

Los componentes principales del entorno de ejecución de Java se presentan a continuación.

1.2.2.1 Componentes de la JVM

Los componentes de la JVM pueden ser implementados de diferentes formas según el fabricante haya seguido la especificación, pero todas deben cumplir con el comportamiento determinado por las especificaciones de Sun para el lenguaje Java.

El componente encargado de la ejecución del código de Java es el llamado Motor de Ejecución, y puede llevar a cabo su tarea de dos formas. Una de ellas es interpretar las instrucciones en *bytecode* contenidas en el archivo *class*, las cuales son independientes de procesador y de sistema operativo. La otra forma es ejecutar el código nativo, el cual es generado de la compilación del *bytecode* y corresponde a la arquitectura en la cual se encuentra. El proceso de compilación utilizado es la compilación *just-in-time* (JIT) y el componente encargado de realizarla es el Compilador JIT.

Además de los componentes principales para la ejecución de código, también forman parte de la JVM; el Cargador de Clases, encargado de cargar dinámicamente las clases encontradas dentro de los archivos *class*, así como del enlace de clases e inicialización de las mismas; el Manejador de Memoria que es el encargado de la administración de memoria, reservar y manejar el espacio de memoria utilizado y mantener un registro de los objetos que se están referenciando. Cuando un objeto deja de estar referenciado entra a funcionar el *Garbage Collection*, quien libera el espacio de memoria utilizado por objetos que ya no son utilizados.

El manejo de la seguridad para controlar las clases que son cargadas y la protección de los recursos del sistema, la administración de hilos para la ejecución de tareas en paralelo, el manejo, control y captura de excepciones y otras actividades se llevan también a cabo dentro del ambiente de ejecución de Java. Todos los componentes que intervienen hacen posible el funcionamiento de Java desde pequeños applets hasta grandes aplicaciones Web del lado del servidor, obteniendo así una arquitectura escalable, robusta, segura y estable.

1.3 Las plataformas

Tanto Microsoft .NET como Java 2 Enterprise Edition ofrecen todas las herramientas necesarias para la construcción de gran variedad de aplicaciones, y orientadas a todas los ambientes, desde aplicaciones *stand-alone*, *applets* incrustados en páginas Web, páginas dinámicas, aplicaciones móviles, aplicaciones empresariales del lado del servidor, orientadas al Web y con inter-operabilidad entre otros sistemas y fuentes de datos.

Ambas plataformas hacen uso de estándares de la industria en la interioridad de sus arquitecturas y en los servicios y protocolos que presentan; e inclusive J2EE se ha definido como una especificación y Microsoft ha liberado el lenguaje de programación C#, permitiendo la implementación de compiladores por parte de terceros. Todo esto en la lucha por definirse como el paradigma que se debe seguir para la construcción de software de cualquier tipo.

La forma de trabajar de ambas plataformas es parecida, desde el uso de ambientes de ejecución para la compilación de código intermedio, .NET con el CIL y J2EE con el *bytecode*; pasando por las páginas dinámicas, .NET con ASPX y J2EE con JSP; conectividad con bases de datos; .NET con ADO.NET y J2EE con JDBC; hasta la implementación de soluciones multicapas y tecnología emergente como lo son los servicios Web. En la Tabla I se muestra como las diferentes plataformas ofrecen la solución a algunas de las necesidades que presenta la tecnología actual.

Tabla I Implementación de soluciones por plataforma

		SOLUCIÓN	
		.NET	**J2EE**
TECNOLOGIA	*Código Intermedio*	MSIL (CIL)	ByteCode
	Ambiente de Ejecución	.NET Framework (CLR)	JVM, JRE (Java Virtual Machine, Java Runtime Environment)
	Lenguaje de Programación	VB.NET, C#, etc.	Java
	Acceso a Base de Datos	ADO.NET	JDBC
	Páginas Dinámicas	ASP.NET (ASPX)	JSP (Java Server Pages) y Servlets
	Aplicaciones StandAlone	WinForms y WebForms	Java Swing/AWT
	Enterprise Services	COM+ .NET	EJB (Enterprise JavaBeans)
	Mensajería	MSMQ (Microsoft Message Queuing)	JMS (Java Messaging Service)
	Componentes Distribuidos	.NET Remoting	Java RMI
	Servicios Web	XML Web Services	WSDP (Java Web Services Developer Pack)

2. DESARROLLO DE SERVICIOS WEB

Los servicios Web representan un desacoplamiento significativo en la construcción de aplicaciones, tanto para pequeñas aplicaciones como para aplicaciones empresariales de gran envergadura, gracias a su funcionamiento como proveedores de un servicio hacia las mismas aplicaciones. Es por ello esencial comprender como es que logran llevar a cabo su propósito y sobre que tecnologías se apoyan; esto y más es presentado en este capitulo, permitiendo una mejor comprensión sobre los servicios Web y el enfoque que le imprimen las plataformas de desarrollo .NET y J2EE.

2.1 *Web Services*

Los *Web Services* o Servicios Web se pueden definir como un conjunto de aplicaciones y protocolos, que desarrollan alguna actividad y permiten la comunicación entre aplicaciones sobre una red de computadoras. Es así como un servicio Web presta cierta funcionalidad a sus clientes a través de protocolos estándar de Internet, permitiendo que las llamadas hacia él sean de forma programática y sin intervención de una persona. Presentándose de este modo una nueva tecnología que no está orientada a los clientes y al contenido, sino a las aplicaciones y los servicios; es decir, mejorar los sistemas de comunicaciones entre aplicaciones para potenciar su funcionalidad, lo que conlleva un beneficio para las empresas y los usuarios finales.

Los servicios Web permiten el intercambio de información entre sí y entre aplicaciones, no importando lenguaje de programación, sistema operativo o dispositivo donde éstas se están ejecutando. Todo ello gracias a que los servicios Web trabajan sobre Internet y lo hacen por medio de estándares.

Las implementaciones de los servicios Web abarcan muchas posibilidades, agrupándolas, se pueden determinar tres grandes áreas en las cuales un servicio Web aporta su funcionalidad:

- Acceso programático a aplicaciones vía Internet, permitir el acceso vía protocolos estándar de Internet a ciertas aplicaciones para llevar a cabo una operación, existiendo la posibilidad de que se cobre por el uso del servicio Web o se presente en forma de valor agregado. Por ejemplo: permitir realizar reservaciones para hoteles o viajes simplemente anotando éstas en el software de agenda, y siendo ella la que se encargue de contactar a las empresas que prestan dichos servicios.

- Integración *Application-to-Application* (A2A), integrar las diferentes aplicaciones que se utilizan en una organización, y posiblemente siendo en diferentes plataformas y lenguajes, es posible de una manera más sencilla por medio de los servicios Web. Gracias a la independencia de plataformas y lenguajes de la arquitectura de servicios Web es posible encarar la Integración de Aplicaciones Empresariales (*Enterprise Application Integration*, EAI).

- Integración *Business-to-Business* (B2B), extendiendo el concepto anterior, integrar aplicaciones entre diferentes organizaciones es algo muy importante y necesario en la actualidad. Los servicios Web permiten esa inter-operabilidad entre organizaciones de forma transparente sobre Internet, utilizando unos y otros los servicios Web que faciliten.

2.1.1 Las columnas que sostienen y forman los servicios Web

Las prestaciones de los servicios Web no serían posibles sin las tecnologías subyacentes, las cuales en conjunto hacen realidad los servicios Web y sus diversas aplicaciones. Estas tecnologías se pueden dividir en cuatro áreas, dichas áreas se deben abarcar para permitir el uso de los servicios Web. Es decir, cada una brinda una parte fundamental de su arquitectura, existen así mismo, otras tecnologías que expanden su funcionalidad pero que no son indispensables, por ejemplo la seguridad, alto rendimiento, alta disponibilidad, etc.

Un servicio Web debe poder ser llamado de forma remota, de lo contrario sería una rutina que tendría que estar en cada aplicación que lo quisiera usar, para cumplir con este requisito es necesario cubrir el área de la comunicación, permitiendo el transporte de la información sobre Internet. Es por ello una de las áreas fundamentales a cubrir, otra área es la información. Un servicio Web debe poder manipular información, esto es, al momento de invocar un servicio Web, se espera que éste reciba información y después de procesarla transmita información de regreso. La información a transmitir y la descripción de cómo se transmite la misma es otro aspecto fundamental para los servicios Web.

Una tercer área que se debe cubrir es la descripción de los servicios Web, esto quiere decir, se debe poder definir cuáles son las funcionalidades que presta un servicio Web. Es necesario que exista una forma de describir cómo se debe interactuar con el servicio Web para que éste pueda ser utilizado por otros clientes. Y por último, ya que se sabe que es necesaria una forma de describir cómo usar un servicio Web, cómo invocarlo y cómo transmitirle información, solo hace falta un método para descubrirlo. Lo anterior se refiere a que debe existir una forma de saber qué servicios Web existen, ya que no se puede hacer uso de algo si no se sabe de su existencia.

Con las áreas anteriores cubiertas por las tecnologías correspondientes, ya se puede definir el marco de trabajo sobre el cual se apoyan los servicios Web. Permitiendo así, identificar qué servicio Web se necesita, cómo usarlo, cómo invocarlo y cómo transmitirle la información necesaria.

2.1.1.1 Describiendo la información a transmitir

Para que los servicios Web puedan otorgarnos la funcionalidad que esperamos, debe ser necesaria una forma de transmitir información. Al igual que es importante definir una forma de transmitir, es importante definir en qué formato estará la información. La forma en que los servicios Web se entienden y comunican es por medio de texto, en formato XML.

XML es un lenguaje extensible de etiquetas cuyo objetivo principal es la descripción de los datos. Un archivo XML es un documento de texto plano y su estructura se basa en etiquetas, las cuales utiliza para la delimitación de los elementos. Permite, como su nombre lo indica, extender las etiquetas en función de los datos que se están describiendo. Esta flexibilidad es la que ha hecho que XML sea adoptado como un estándar para el intercambio de datos entre componentes de software.

El W3C (*World Wide Web Consortium*) ha definido el estándar XML 1.1 como un formato de texto simple y flexible, derivado de SGML (*Standard Generalized Markup Language* - ISO 8879). Y se ha definido como una Recomendación del W3C para el intercambio de datos en la arquitectura de los servicios Web, sin que esto prohíba que otras entidades hagan uso de otras tecnologías para la implementación de los servicios Web.

2.1.1.2 Describiendo la comunicación y el acceso

Como se indicaba antes, la comunicación entre componentes de software es indispensable para el funcionamiento de los servicios Web. Debe existir alguna forma para invocar los servicios Web, y esto se logra por medio de protocolos de comunicación para envío de mensajes. Al describir la información por medio de XML, el protocolo encargado de transmitirlo debe permitir la manipulación de XML.

SOAP (*Simple Object Access Protocol*) es un protocolo estándar que define cómo dos aplicaciones pueden comunicarse por medio de intercambio de datos XML, y esto de forma independiente de plataforma y lenguaje de la aplicación. SOAP puede trabajar sobre cualquier protocolo de Internet, y permite trabajar las llamadas a procedimientos remotos (RPC, *Remote Prodedure Call*) como varios mensajes SOAP interactuando entre sí.

Microsoft, IBM y otras compañías fueron los creadores de SOAP, siendo ahora el W3C, el encargado de las especificaciones y el seguimiento de SOAP. El W3C ha definido SOAP Versión 1.2 como un protocolo ligero y extensible para el intercambio de información de forma descentralizada en ambientes distribuidos, basado en la tecnología XML para proveer la construcción de mensajes que puedan ser intercambiados sobre la base de cualquier protocolo. Así mismo, lo define como una Recomendación del W3C para la comunicación entre servicios Web dentro de su arquitectura, trabajando sobre HTTP (*HyperText Transfer Protocol*), sin negar el derecho de otras tecnologías para la implementación de servicios Web.

2.1.1.3 Describiendo las capacidades y funciones

Ya que se han abarcado las tecnologías que permiten a los servicios Web funcionar, debe cubrirse el aspecto de la definición de las capacidades y funciones que puede brindar un servicio Web. Debe existir una tecnología capaz de describir el servicio Web para que otros servicios Web o aplicaciones sepan cómo invocarlo y qué funciones utilizar de él. Esto es definir las interfaces para hacer uso de las funciones que nos brinda un servicio Web.

La tecnología que permite llevar a cabo lo anterior es llamada WSDL (*Web Service Descripction Language*), el cual es un lenguaje que utiliza XML para la definición de las interfaces públicas de los servicios Web, describe los requisitos del protocolo de comunicación y el formato de los mensajes necesarios para interactuar con el servicio Web. La descripción de las interfaces se hace de forma abstracta, es decir, independiente del protocolo de red o del lenguaje de la implementación de las aplicaciones.

El W3C define WSDL como un formato XML para describir los servicios como puntos de acceso para operar mensajes que contienen información. Las operaciones y mensajes son descritas de forma abstracta, permitiendo así la descripción sin importar el formato de los mensajes o el protocolo de comunicación. Se recomienda su uso junto con las tecnologías SOAP sobre HTTP.

Por el momento WSDL 1.1 se encuentra en fase de Propuesta de Recomendación en el W3C. Sin embargo es la tecnología más utilizada para la descripción de los servicios Web, y al igual que las demás tecnologías que intervienen en los servicios Web, permite que se utilicen otras implementaciones.

2.1.1.4 Localizando los servicios Web

Solventadas las áreas de información, comunicación y descripción, solo queda una por abarcar y ésta es, el descubrimiento de los servicios Web. Cómo encontrar que servicios Web hay disponibles para consumir es el objetivo de la tecnología llamada UDDI (*Universal Description, Discovery and Integration*). Dicha tecnología se enfoca en servir como un directorio donde se puedan publicar y ubicar los diferentes servicios Web que prestan las compañías, así se puede tener un panorama más amplio de los servicios Web disponibles.

UDDI es un directorio que define la especificación para el registro de los servicios Web disponibles, permitiendo a las compañías presentarlos y anunciarse como proveedores de ellos. OASIS (*Organization for the Advancement of Structured Information Standards*) es la organización encargada de UDDI y define actualmente el estándar UDDI Versión 3. UDDI hace uso de los estándares definidos por la W3C como lo son XML, SOAP y WSDL para la correcta inter-operabilidad de los servicios Web.

2.2.2 Las tecnologías detrás de los servicios Web

Cada tecnología se ha implementado gracias al trabajo en conjunto de un grupo de empresas. Debido a esto los servicios Web se han conformado por estándares de la industria. A continuación una descripción más especifica de cada una de las tecnologías que permiten el funcionamiento de los servicios Web.

2.2.2.1 XML

XML es el lenguaje de marcas utilizado para la descripción de la información que se transmite de y hacia un servicio Web. Así mismo es la tecnología sobre la cual se basan o hacen uso SOAP, WSDL y UDDI, así que es parte fundamental e integral de los servicios Web. XML es una recomendación de W3C, siendo XML 1.1 la última recomendación anunciada por ellos (4/2/2004) y siendo la versión XML 1.0 la utilizada por las plataforma .NET y J2EE.

XML es utilizado para almacenar información de forma estructurada, permitiendo su transmisión por cualquier protocolo de Internet que soporte la transmisión de texto o documentos de texto. XML es independiente de plataforma ya que únicamente especifica una forma estructurada de almacenamiento. Los documentos XML deben respetar las reglas de sintaxis definidas por la especificación. Hay dos tipos de documentos: “bien formados” y válidos.

Los documentos “bien formados” son aquellos que respetan las reglas sintácticas del lenguaje definidas por la especificación y que no están relacionados con un DTD (*Data Type Definition*). Los documentos válidos son aquellos que además de estar “bien formados” siguen una estructura y una semántica definida por un DTD. Un DTD es la definición de la estructura que debe presentar un documento XML, define el orden de los elementos y los atributos que puede tener cada elemento.

2.2.2.1.1 Estructura del XML

Un documento XML, como su nombre lo indica, está compuesto de *tags* (marcas). Los *tags* delimitan un elemento dentro del documento y son definidas por el creador del documento, es decir, no existen *tags* predefinidos de los cuales hacer uso. Esto permite que el documento sea auto-descriptivo.

Figura 4 Documento XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Programa>
<Funcion Nombre="F1">
    <Publica>SI</Publica>
    <Tipo>real</Tipo>
</Funcion>
<Funcion Nombre="F2">
    <Publica>NO</Publica>
    <Tipo>entero</Tipo>
</Funcion>
</Programa>
```

En la Figura 4 se aprecia como un documento empieza con la declaración, esto es, se define como un documento XML indicando la versión, la codificación utilizada (set de caracteres) y si hay un DTD que lo defina o no. El set de caracteres debe ser válido; los sets de caracteres válidos son definidos por la IANA (*Internet Assigned Numbers Authority*). Y si se utilizará un DTD, éste debe estar especificado en la siguiente línea (ya sea con referencia o incluido en el documento). En la Figura 5 se aprecia la estructura de un DTD.

Figura 5 Documento DTD

```
<!DOCTYPE Programa [
<!ELEMENT Funcion ( Publica, Tipo ) >
<!ATTLIST Funcion Nombre NMTOKEN #REQUIRED >
<!ELEMENT Programa ( Funcion+ ) >
<!ELEMENT Publica ( #PCDATA ) >
<!ELEMENT Tipo ( #PCDATA ) >
]>
```

Después de la declaración sigue en sí el contenido del documento. Los elementos son delimitados por *tags* y cada *tag* debe tener su *tag* de cierre con excepción de la declaración. El primer elemento es llamado *root* y solo debe existir uno, de los demás elementos pueden existir varios según se defina en el DTD, si éste existe. Los elementos pueden tener atributos y es el creador del documento quien los define, de este modo los datos pueden ser almacenados en atributos o en elementos. No existen reglas de cuándo usar cada uno de ellos.

El atributo *xmlns* define un *namespace* dentro del documento XML, permitiendo de este modo identificar los elementos que pertenecen a un conjunto de nombres. Los *namespaces* se utilizan dentro de XML para permitir la inclusión de dos *tags* iguales pero con significado diferente, ya que ambos deben pertenecer a diferentes *namespaces*. La opción de usar *namespaces* es decisión del usuario, pero se convierte en necesaria cuando se incluyen tags de distintas fuentes, ya que existe el peligro de que dos *tags* sean iguales. Un namespace se conforma de un prefijo que se antepone a los *tags* y de un URI (*Universal Resource Identifier*), que no es más que una serie de caracteres que identifican un recurso de Internet, por ejemplo una URL (Universal Resource Locator) o una URN (Universal Resource Name). El URI solo sirve como identificador único del *namespace*, no es necesario que exista.

Otra forma de definir la estructura de un documento XML es por medio de un *XML Schema* (XSD), que es simplemente un documento basado en XML que define la estructura del documento XML. XSD es una recomendación de la W3C, siendo la segunda edición la versión vigente emitida el 28 de octubre de 2004. XSD se perfila como el sucesor indicado de los DTD, ya que está escrita en XML y por lo tanto es extensible para soportar nueva funcionalidad. El principio sobre el cual funciona y cuyo objetivo pretende lograr es el mismo que los DTD, definir la estructura y semántica de un documento XML. La Figura 6 muestra un ejemplo de un documento *XML Schema*.

Figura 6 Documento *XML Schema*

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Funcion">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Publica" />
            <xs:element ref="Tipo" />
        </xs:sequence>
        <xs:attribute name="Nombre" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
</xs:element>
<xs:element name="Programa">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Funcion" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Publica">
    <xs:complexType mixed="true" />
</xs:element>
<xs:element name="Tipo">
    <xs:complexType mixed="true" />
</xs:element>
</xs:schema>
```

Las ventajas que presenta un *XML Schema* son:

- Uso de tipos de datos, permitiendo describir de mejor manera los elementos, validar los datos, trabajar con patrones de datos y convertir datos de un tipo a otro.

- Al estar basado en XML no es necesario aprender otro lenguaje y se tienen todas las cualidades que presenta XML, también se pueden utilizar todos los utilitarios para trabajar con XML, como editores, parsers, etc.

Alrededor de XML se han desarrollado muchas aplicaciones y una más es su aplicación dentro de toda la arquitectura de los servicios Web.

2.2.2.2 SOAP

El protocolo de comunicación SOAP sirve para intercambiar información entre aplicaciones por medio de Internet, está basado en XML y establece formatos para el traspaso de mensajes, es independiente de plataforma y del lenguaje de programación. Al estar basado en XML hereda las características de éste, siendo simple y extensible, además se maneja sobre HTTP permitiendo una mejor comunicación sobre firewalls y proxies.

Un mensaje SOAP es un documento XML que se compone de un elemento obligatorio llamado *Envelope*, que lo identifica como un mensaje SOAP; otro elemento obligatorio llamado *Body*, que contiene información de la llamada y la respuesta; un elemento opcional *Header* antes del *Body* y otro elemento opcional *Fault* en el *Body*; éste contiene información de los errores ocurridos en el procesamiento del mensaje.

Además todos los elementos anteriores deben ser utilizados bajo el *namespace* soap-envelope (URI http://www.w3.org/2001/12/soap-envelope) y el *namespace* soap-encoding (URI http://www.w3.org/2001/12/soap-encoding). En la Figura 7 se puede apreciar la estructura que tiene un mensaje SOAP. Las referencias a los *namespaces soap* y *encodingStyle* deben existir siempre, de lo contrario se debe considerar como un error en el mensaje.

Figura 7 Mensaje SOAP

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
...
...
</soap:Header>
<soap:Body>
...
...
<soap:Fault>
...
...
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

Si el elemento *Header* aparece en el mensaje, debe ser el primero. Éste contiene información específica de la aplicación e indica cómo debe ser procesado el mensaje por el destinatario. Existen tres atributos que pueden aparecer en el H*eader*: *actor* (a quien va dirigido el Header), *mustUnderstand* (si se debe procesar el *Header*) y *encodingStyle* (tipo de dato utilizado en el elemento).

El *Body* contiene en sí la información a transmitir, ya sea llamada o respuesta. Dentro de él se especifican los elementos que se consideren necesarios, siendo *Fault* el único elemento definido por el protocolo. Este elemento opcional debe estar presente una única vez y contendrá los errores que se ocurran en el procesamiento del mensaje. Consta de cuatro subelementos: *faultcode*, *faultstring*, *faultactor* y *detail*. La Figura 8 muestra un mensaje SOAP que llama una función, mientras que la Figura 9 muestra la posible respuesta a dicha llamada.

Figura 8 Llamada mensaje SOAP

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
    <m:CallFuncion xmlns:m="http://www.prog.edu/func">
        <m:Nombre>F1</m:Nombre>
    </m:CallFuncion>
</soap:Body>
</soap:Envelope>
```

Figura 9 Respuesta mensaje SOAP

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
    <m:FuncionResponse xmlns:m="http://www.prog.edu/func">
        <m:Valor>1.90</m:Valor>
    </m:FuncionResponse>
</soap:Body>
</soap:Envelope>
```

El W3C recomienda el uso de SOAP sobre HTTP, aunque se puede utilizar sobre cualquier protocolo de Internet, siendo los métodos SOAP solicitudes o respuestas HTTP que respetan las reglas de codificación de SOAP. Es decir, son mensajes en XML que siguen las reglas definidas por el protocolo SOAP y que viajan sobre HTTP.

El protocolo HTTP define que para una solicitud POST se debe incluir al menos dos elementos en el *header*, los cuales son *Content-Type* y *Content-Length*. El *header Content-Type* define el tipo MIME (*Multipurpose Internet Mail Extensions*) del mensaje, esto es, el tipo de dato que se está transmitiendo y opcionalmente puede especificar el tipo de codificación de caracteres (*charset*) utilizado en el mensaje XML. El *header Content-Length* define el número de bytes que ocupa el cuerpo del mensaje XML.

2.2.2.3 WSDL

El lenguaje de descripción de servicios Web, WSDL, es el lenguaje que nos permite describir un servicio Web y define como utilizarlo, es decir, define la interfaz para consumirlo. Es un lenguaje basado en XML, por lo tanto, se presenta como un documento XML que describe las operaciones que ofrece el servicio Web y donde localizarlo. La versión actualmente utilizada es la 1.1, sin ser esta una Recomendación de W3C sino una Propuesta de Recomendación. Existe, así mismo, un Candidato a Recomendación para la versión WSDL 2.0 con fecha del 27 de marzo de 2006.

En la Figura 10 se presenta parte de la estructura gramatical de un documento WSDL, la estructura está definida en la Nota del 15 de marzo del 2001 de la versión 1.1 publicada por el W3C. Los elementos básicos que componen un documento WSDL se pueden dividir en dos secciones: definición de operaciones y definición de protocolos a utilizar.

Figura 10 Gramática documento WSDL

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri"?>
    <import namespace="uri" location="uri"/>*
    <wsdl:documentation .... /> ?
    <wsdl:types> ?
        <wsdl:documentation .... />?
        <xsd:schema .... />*
        <-- extensibility element --> *
    </wsdl:types>
    <wsdl:message name="nmtoken"> *
        <wsdl:documentation .... />?
        <part name="nmtoken" element="qname"? type="qname"?/> *
    </wsdl:message>
    <wsdl:portType name="nmtoken">*
        <wsdl:documentation .... />?
        <wsdl:operation name="nmtoken">*
            <wsdl:documentation .... /> ?
            <wsdl:input name="nmtoken"? message="qname">?
                <wsdl:documentation .... /> ?
            </wsdl:input>
            <wsdl:output name="nmtoken"? message="qname">?
                <wsdl:documentation .... /> ?
            </wsdl:output>
            <wsdl:fault name="nmtoken" message="qname"> *
                <wsdl:documentation .... /> ?
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:portType>
```

La Figura 11 continúa con la siguiente parte de la estructura del documento WSDL, completando las definiciones que faltan junto con los *tags* finales del documento. Los símbolos cierre de interrogación (?) y asterisco (*), indican que el elemento puede aparecer cero o una vez y que puede aparecer cero o más veces respectivamente, estos símbolos no son parte de la gramática.

Figura 11 Continuación gramática documento WSDL

```
<wsdl:binding name="nmtoken" type="qname">*
    <wsdl:documentation .... />?
    <-- extensibility element --> *
    <wsdl:operation name="nmtoken">*
       <wsdl:documentation .... /> ?
       <-- extensibility element --> *
       <wsdl:input> ?
          <wsdl:documentation .... /> ?
          <-- extensibility element -->
       </wsdl:input>
       <wsdl:output> ?
          <wsdl:documentation .... /> ?
          <-- extensibility element --> *
       </wsdl:output>
       <wsdl:fault name="nmtoken"> *
          <wsdl:documentation .... /> ?
          <-- extensibility element --> *
       </wsdl:fault>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="nmtoken"> *
    <wsdl:documentation .... />?
    <wsdl:port name="nmtoken" binding="qname"> *
       <wsdl:documentation .... /> ?
       <-- extensibility element -->
    </wsdl:port>
    <-- extensibility element -->
</wsdl:service>
<-- extensibility element --> *
</wsdl:definitions>
```

Fuente *Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001*

Como se puede apreciar en las figuras 10 y 11, el documento WSDL está formado de elementos de una forma estructurada, como lo manda el XML. Ambas figuras separan de buena forma los tipos de elementos necesario que deben especificarse. Los elementos más importantes de la definición de operaciones son *types*, *message* y *portType*.

El elemento *type* define los tipos de datos utilizados por el servicio Web, para mayor independencia de plataforma los define utilizando la sintaxis indicada por *XML Schema* para la definición de tipos de datos. El elemento *message* define los datos de una operación, es decir, los mensajes que puede enviar o recibir un servicio Web. Un mensaje se puede componer de varias partes, cada parte se puede ver como un parámetro para la ejecución de una función. El elemento *portType* define en sí el servicio Web, define las operaciones que puede realizar y los mensajes involucrados.

La sección que enlaza las operaciones con los protocolos por los cuales se pueden utilizar el servicio Web, se compone principalmente de dos elementos: *binding* y *service*. El elemento *binding* define el formato del mensaje y los detalles del protocolo, por lo general SOAP sobre HTTP, por el cual se utilizará cada operación. El elemento *service* define explícitamente la localización, en forma de URL, del servicio Web. La Figura 12 muestra un ejemplo de un documento WSDL, éste es para un servicio Web compuesto de una única función, la cual recibe un número realizando una operación cualquiera con él y devuelve otro número.

Figura 12 Ejemplo documento WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="DemoWService"
   targetNamespace="http://www.ecerami.com/wsdl/DemoWService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.ecerami.com/wsdl/DemoWService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <message name="CallFuncion">
      <part name="parametro" type="sxd:number"/>
   </message>
   <message name="FuncionResponse">
      <part name="resultado" type="sxd:number"/>
   </message>
    <portType name="DemoWS_PortType">
      <operation name="Funcion">
         <input message="tns:CallFuncion"/>
         <output message="tns:FuncionResponse"/>
      </operation>
   </portType>
    <binding name="DemoWS_Binding" type="tns:DemoWS_PortType">
      <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="Funcion">
         <soap:operation soapAction="Funcion"/>
         <input>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:DemoWService"
               use="encoded"/>
         </input>
         <output>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:DemoWService"
               use="encoded"/>
         </output>
      </operation>
   </binding>
    <service name="DemoWS_Service">
      <documentation>WSDL File for DemoWService</documentation>
      <port binding="tns:DemoWS_Binding" name="DemoWS_Port">
         <soap:address
            location="http://www.demows.edu:8080/soap/servlet/rpcrouter"/>
      </port>
   </service>
</definitions>
```

2.2.2.3.1 Estructura del WSDL

El elemento *definitions* debe ser el elemento *root*, define el nombre del servicio Web y declara los múltiples *namespaces* a utilizar en el documento.

El elemento *message* representa los mensajes a utilizar en la comunicación con el servicio Web, consta de uno o más elementos *part*. Dichos elementos representan los parámetros a enviar o los valores a retornar, dependiendo de si el mensaje va hacia o viene del servicio Web respectivamente. El elemento *part* tienen el atributo *type* que indica el tipo de dato, este tipo de dato se recomienda sea un tipo de dato definido por el *XML Schema* y debido a esto debe estar dentro de dicho *namespace*, definido anteriormente en el elemento *definitions*.

El elemento *portType* define un conjunto de operaciones, a manera de librería, que ofrece el servicio Web. Puede contener uno o varios elementos *operation* los cuales son en sí las operaciones, y estos deben contener los mensajes por medio de los cuales se realizará la comunicación. Los mensajes deben pertenecer a un *namespace*, el cual se especificó al inicio al igual que el nombre de los mensajes. Los mensajes se encapsulan en los elementos *input* y *output*, además el elemento *operation* opcionalmente puede contener el elemento *fault*.

WSDL soporta 4 patrones básicos de operación, los cuales se presentan a continuación:

- *One-way* (de una vía): El servicio recibe un mensaje y no genera respuesta. Contendrá un único elemento *input*.

- *Request-response* (solicitud-respuesta): El servicio recibe un mensaje y manda una respuesta. Contendrá un elemento *input* seguido de un *output*.

- *Solicit-response* (solicita respuesta): El servicio envía un mensaje y recibe una respuesta. Contendrá un elemento *output* seguido de un *input*.

- *Notification* (notificación): El servicio envía un mensaje y no espera respuesta. Contendrá un único elemento *output*.

El elemento *binding* brinda detalles específicos de como una definición *portType* será transmitida, es decir, los protocolos a utilizar para su comunicación. Se pueden definir varios elementos *binding* para un mismo elemento *portType*, entre los cuales puede ser HTTP GET, HTTP POST o SOAP (recomendado).

La versión 1.1 de WSDL incorpora extensiones para SOAP, esto permite definir detalles específicos sobre el protocolo, detalles como el encabezado y el estilo de codificación. A continuación se presentan los elementos más importantes a definir:

- *soap:binding* este elemento indica que el enlace se hará por medio del protocolo SOAP; el atributo *style* indica el estilo de formato de todo el mensaje SOAP, cuyos valores puede ser *rpc* o *document (*valor *default* si no se especifica*),* la diferencia es únicamente el formato del contenido del mensaje; el atributo *transport* indica el medio de transporte del mensaje, algunos valores son http://schemas.xmlsoap.org/soap/http para HTTP (recomendado) o http://schemas.xmlsoap.org/soap/smtp para SMTP.

- *soap:operation* este elemento indica el enlace de una operación especifica con una implementación especifica; el atributo *soapAction* indica que encabezado soapAction de HTTP será utilizado para identificar el servicio.

- *soap:body* este elemento permite especificar los detalles de los mensajes de entrada y salida, indicando el estilo de codificación y el URN *namespace* asociado al servicio especificado.

El elemento *service* específica la localización del servicio Web en forma de URL y esta dirección debe existir, ya que será la dirección a utilizar cuando se llame al servicio Web.

2.2.2.4 UDDI

El servicio que presta UDDI (*Universal Description, Discovery and Integration*) es el que permite el descubrimiento de servicios Web, es decir, permite a una organización exponer sus servicios Web y al mismo tiempo poder descubrir los servicios Web que otra organización haya expuesto. UDDI se puede definir, según el W3C, como un marco de trabajo independiente de plataforma para describir servicios, descubrir negocios e integrar los servicios Web utilizando el Internet.

UDDI funciona como un directorio para almacenar información acerca de los servicios Web, con una arquitectura similar al DNS (*Domain Name System*) en lo que respecta a la distribución de información sobre distintos servidores alrededor del mundo, en el llamado *UDDI business registry*. La descripción de las interfaces de los servicios Web se realiza por medio de WSDL y la comunicación se hace por medio de SOAP, basándose de este modo, en la tecnología XML para toda la estructura. Y por ello se considera UDDI como un servicio Web en sí, un servicio que provee de información útil a otros servicios Web.

Una organización que desee exponer sus servicios Web debe crear un registro de negocios UDDI, el registro de negocios es un documento XML cuyo formato esta especificado en el *UDDI-defined schema*. La Figura 13 muestra un ejemplo de un registro UDDI.

Figura 13 Documento UDDI

```
<businessEntity businessKey=
    "E7CD0D00-1827-11CF-9946-4444553540000">
    <name>OrganizacionEdu</name>
    <description>Institucion Educacional</description>
    <contacts>
        <contact>
            <personName>Webmaster</personName>
            <email>webmaster@orgedu.edu</email>
        </contact>
    </contacts>
    <businessServices>
        <businessService serviceKey="15940A43-1956-63DC-0124-4444553540000">
            <name>ServicioDePrueba</name>
            <bindingTemplates>
                <bindingTemplate bindingKey="9E43F20A-1963-22EC-1053-4444553540000">
                    <accessPoint>http://www.orgedu.edu/servicio.asmx</accessPoint>
                    <tModelInstanceDetails>
                        <tModelInstanceInfo tModelKey="uuid:F7A90326-EB0D-40E0-9013-F55606BD4804">
                            <description>Acceso al servicio</description>
                        </tModelInstanceInfo>
                        <tModelInstanceInfo tModelKey="uuid:43CAC139-D822-40B5-A004-82DDDC0A12F2">
                            <description>Referemcia de enlace WSDL</description>
                            <instanceDetails>
                                <!-- Detalles de la interface WSDL -->
                            </instanceDetails>
                        </tModelInstanceInfo>
                    </tModelInstanceDetails>
                </bindingTemplate>
            </bindingTemplates>
        </businessService>
    <businessServices>
</businessEntity>
```

El elemento raíz en este documento es *businessEntity*, contiene el atributo *businessKey* que provee un identificador único de forma global[3]. Los elementos que componen un *businessEntity* van desde información general sobre la organización, por ejemplo *name*, *description* y *contact*; hasta el más importante que es *businessServices*.

[3] UUID (*Universally Unique Identifier*) consiste de de un valor único de 6 bytes compuesto de la dirección Ethernet de la máquina donde fue creado y otros campos. También es conocido GUID (*Globally Unique Identifier*)

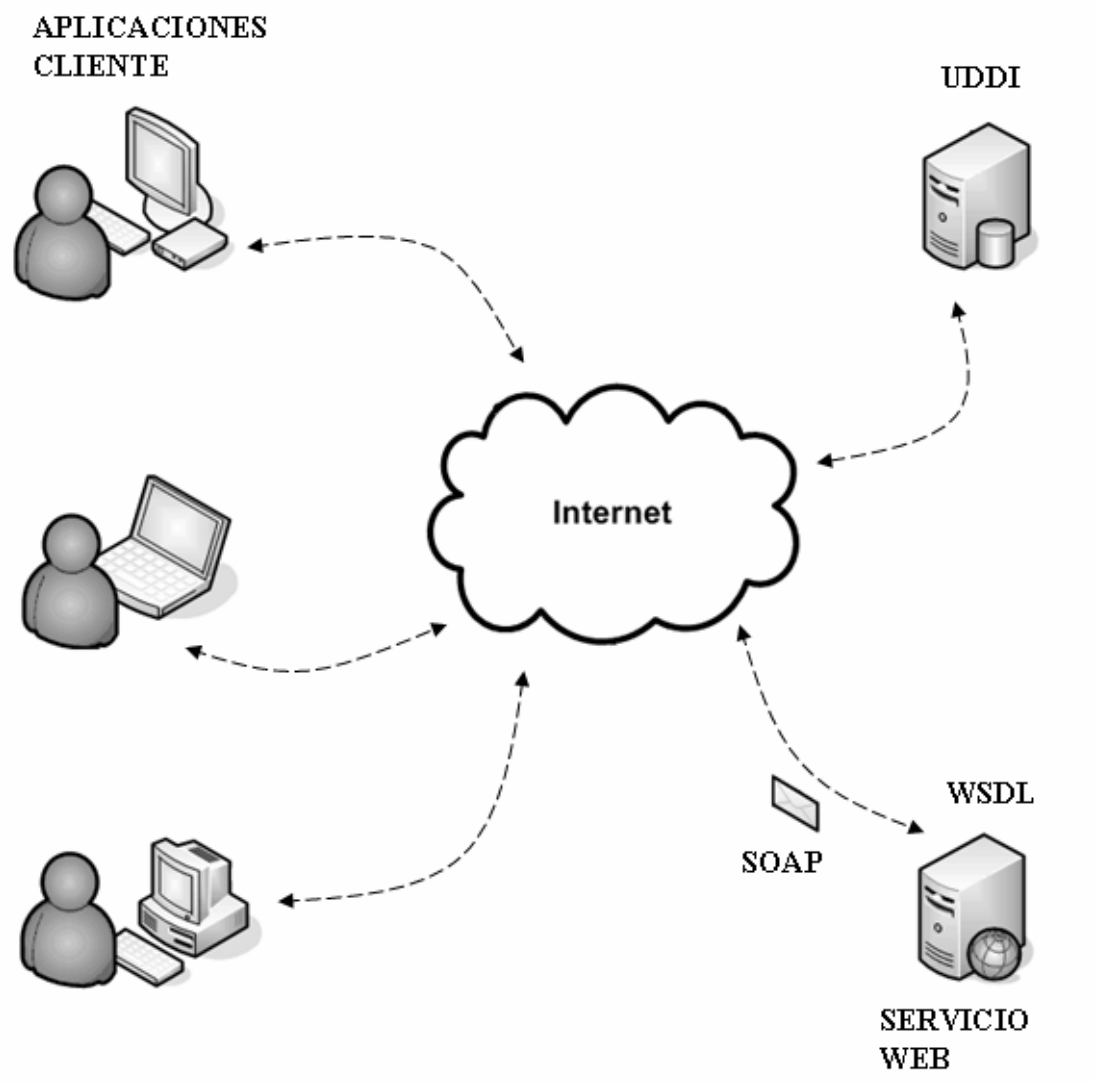
El elemento *businessServices* contiene uno o varios elementos *businessService*, cada uno describiendo un servicio Web e identificándolo con un nombre y una clave UDDI que identifica a la definición del servicios de manera única. Contiene al elemento llamado *bindingTemplates*, el cual puede contener uno o varios elementos *bindingTemplate*.

El elemento *bindingTemplate* contiene un identificador único, una clave UDDI, y la información necesaria para que un cliente pueda hacer uso del servicio Web, esta información incluye el elemento *accessPoint*, que indica la URL donde se encuentra el servicio; y varios elementos *tModels* (*Technology Model*), que indican ciertos aspectos específicos del servicio y cada uno identificado con una clave UDDI.

De este modo se permite conformar un registro con los datos necesarios para la integración de servicios Web entre organizaciones. UDDI es una columna importante en la base de los servicios Web, permitiendo ubicar los servicios y brindando la información necesaria para utilizarlos.

Para que se pueda llegar a utilizar un servicio Web lo primero a realizar es su descubrimiento, la función de UDDI es brindar un directorio de servicios Web en el cual se pueda identificar el servicio Web requerido y la información asociada a él, por ejemplo, la localización del documento WSDL. Con la información del documento WSDL se conocen los detalles de las operaciones que brinda el servicio Web y se puede crear en la aplicación cliente las invocaciones necesarias a dichas operaciones. La comunicación entre la aplicación cliente y el servicio Web se realiza por medio de mensajes SOAP, y en la mayoría de los casos se realiza sobre el protocolo HTTP. Es importante resaltar que las aplicaciones cliente pueden o no requerir intervención humana para disparar la invocación al servicio Web.

Figura 14 Esquema general tecnologías servicios Web



Sobre las cuatro tecnologías descritas anteriormente se basa el funcionamiento de los servicios Web. La Figura 14 muestra un esquema para identificar la localización de estas tecnologías que como se explico en el párrafo anterior están fuertemente relacionadas. No se debe pasar por alto que el documento UDDI, el documento WSDL y los mensajes SOAP están basados en XML.

2.2.3 Otras tecnologías aplicadas a los servicios Web

Los servicios Web además de conformarse por los cuatro pilares, XML, SOAP, WSDL y UDDI, vistos anteriormente, también pueden hacer uso de otras tecnologías para ofrecer seguridad, alto rendimiento, alta disponibilidad, etc. Estas otras tecnologías permiten extender la funcionalidad de los servicios Web y pueden ser aplicadas de diversas formas, y con ello, el riesgo de crear servicios Web que pierdan la inter-operabilidad. La utilización de los servicios Web deben mantener la inter-operabilidad y debido a esto han surgido organizaciones que ayudan a mantener el orden en el ambiente de los servicios Web y las diferentes tecnologías que los rodean.

2.2.3.1 WS-I

La WS-I (*Web Services Interoperability Organization*) es una organización abierta que busca la inter-operabilidad de los servicios Web a través de plataformas, sistemas operativos y lenguajes de programación. Para ello hace uso de grupos de trabajo que permiten la generación de guías, recomendaciones de mejores prácticas y recursos, todo con el fin de que la evolución de los servicios Web sea de una forma ordenada y coherente. La WS-I está conformada por gran cantidad de organizaciones, entre ellas se puede mencionar algunas las empresas fundadoras: Oracle, Microsoft, Sun Microsystems, IBM, BEA Systems, Intel y otras.

La WS-I a través de sus grupos de trabajo ha definido hasta el momento algunos perfiles (*profiles*) para la implementación de los servicios Web. La WS-I define un perfil (profile) como un conjunto de definiciones y especificaciones sobre los estándares comúnmente aceptados por la industria y guías sobre como dichas especificaciones deben ser implementadas para el desarrollo de servicios Web inter-operables. Entre los perfiles que ha definido la WS-I se encuentran *Basic Profile, Attachments Profile* y *Simple SOAP Binding Profile*, además se encuentra trabajando en el *Basic Security Profile*.

2.2.3.1.1 *Basic Profile*

El *Basic Profile* es un conjunto de guías que recomiendan como implementar el conjunto de especificaciones estándar que conforman los servicios Web. Las áreas que cubre el perfil son: mensajería, descripción, descubrimiento y seguridad. A continuación se presentan las especificaciones que menciona el *Basic Profile* 1.1 publicado el 24 de agosto del 2004:

- SOAP 1.1
- WSDL 1.1
- UDDI 2.0
- *XML 1.0 (Second Edition)*
- *Namespaces in XML 1.0*
- *XML Schema Part 1: Structures*
- *XML Schema Part 2: Data Type*
- RFC2246: *The Transport Layer Security Protocol Version 1.0*
- RFC2459: *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*

- RFC2616*: HyperText Transfer Protocol – HTTP 1.1*
- RFC2818: *HTTP over Transport Layer Security (TLS)*
- RFC2965: *HTTP Sate Management Mechanism*
- *The Secure Sockets Layer Protocol – SSL Version 3.0*

Únicamente si el servicio Web cumple con las especificaciones del Basic Profile, puede decirse que es completamente inter-operable con otros servicios Web y clientes que también cumplan con dicho perfil. La última versión de los perfiles se puede encontrar en la página de WS-I (www.ws-i.org).

2.2.3.1.2 *Attachments Profile*

El *Attachments Profile* define las guías para la implementación de servicios Web que hagan uso de mensajes SOAP con datos adjuntos (*attachments*). Este perfil está basado en el *Basic Profile* 1.1 y su versión actual es *Attachments Profile* 1.0 publicada el 24 de agosto de 2004. Las especificaciones de las cuales hace mención el perfil se presentan a continuación:

- *SOAP Messages with Attachments*
- WSDL 1.1
- *XML 1.0 (Second Edition)*
- *Namespaces in XML 1.0*
- RFC2557: *MIME Encapsulation of Aggregate Documents*
- RFC2045: *MIME Part 1, Format of Internet Message Bodies*
- RFC2046: *MIME Part 2, Media Types*
- RFC2392: *Content-ID and Message-ID Uniform Resource Locators*

2.2.3.2 OASIS

OASIS (*Organization for the Advancement of Structured Information Standars*) es un consorcio internacional no lucrativo dedicado a la adopción, convergencia y desarrollo de estándares para los *e-Business* (negocios electrónicos), entre ellos los servicios Web. Al igual que la WS-I, OASIS está conformado por muchas organización y cuenta entre ellas a empresas muy importantes como Oracle, Microsoft, IBM, Sun Microsystems, Nokia, etc.

Entre los estándares referentes a los servicios Web que ha presentado OASIS, además del UDDI y ebXML (*e-business XML*, *Electronic Business using eXtensible Markup Language*), se encuentran: *WS-Reliability, WS-Resource, WS-Security* (WSS) y *Web Services for Remote Portlets* (WSRP) entre otros.

2.2.3.2.1 *WS-Reliability*

El propósito del estándar *WS-Reliability* es completar la especificación de fiabilidad necesaria en el intercambio de mensajes en los servicios Web. Pretende ayudar en la definición de fiabilidad y seguridad en el contexto actual de los estándares de servicios Web, en la comunicación SOAP sobre HTTP, permitiendo ser utilizado en combinación con otros protocolos estándares. *WS-Reliability* se relaciona con las siguientes especificaciones:

- SOAP 1.1/1.2
- *OASIS ebXML Message Service Specification 2.0*
- *OASIS Web Services Security: SOAP Message Security 1.0*
- *WS-I Basic Profile 1.1*

El propósito final de *WS-Reliability* es establecer un modelo genérico y abierto que garantice la entrega fiable de mensajes a los servicios Web, WS-Security 1.1 fue establecido como estándar el 15 de noviembre de 2004. Entre las características que especifica el estándar se encuentra:

- *Garantía de entrega al menos una vez:* el mensaje debe ser entregado al destinatario o de lo contrario una notificación de un posible fallo en la entrega debe ser entregado al remitente.

- *Eliminación de duplicados – entrega al menos uno:* los mensajes duplicados son detectados y eliminados por el destinatario.

- *Garantía de orden de mensajes:* los mensajes son entregados en el orden que fueron enviados.

2.2.3.2.2 *WS-Security*

El propósito del estándar *WS-Security* (WSS), actualmente en la versión 1.1 publicada en Febrero del 2006, es proveer un medio para aplicar seguridad a los servicios Web. El estándar contiene especificaciones sobre cómo se debe hacer cumplir la integridad y la confidencialidad en los mensajes de los servicios Web. WSS incluye detalles del uso de SAML[4] (*Security Assertion Markup Language*), *Kerberos*[5] y certificados X.509 y TLS (*Transport Layer Security*). WSS incorpora características de seguridad en el encabezado de los mensajes SOAP.

[4] SAML (*Security Assertion Markup Language*): estándar XML para el intercambio de información de autenticación y autorización entre dominios seguros, esta definido por OASIS, versión actual SAML 2.0.
[5] *Kerberos* es un protocolo de autenticación de red que permite a dos computadores en una red insegura demostrar su identidad mutuamente de manera segura utilizando criptografía.

2.2.4 Seguridad en los servicios Web

Un aspecto que debido a su importancia debe tratarse con mayor profundidad es la seguridad en los servicios Web. Los servicios Web, como cualquier otro tipo de comunicación por una red abierta como Internet, pueden ser blanco de diversos ataques o fallos en la transmisión de datos. También pueden significar un recurso utilizado por aquellos a los que no está precisamente ofrecido.

Los riesgos a los cuales se exponen los servicios Web se pueden resumir en los siguientes:

- **Modificación de datos:** los datos transmitidos en los mensajes SOAP pueden ser alterados intencionalmente, por terceros que estén pendientes de la comunicación establecida; sin intención, debido a fallas o ruido en el canal de comunicación.

- **Escucha de datos:** los datos transmitidos en los mensajes SOAP pueden estar siendo monitoreados y capturados por terceros, con lo cual se pueden hacer de datos privados como números de tarjetas de crédito, contraseñas o cualquier otra información de importancia.

- **Acceso no autorizado a los datos:** cuando un servicio Web es publicado se convierte en una puerta hacia la información que éste pueda generar, por lo que se puede dar el caso que el servicio Web sea utilizado no por las personas a quienes estaba orientado sino por otras, quienes no deberían tener acceso.

- **Repudio de transacciones:** las operaciones utilizadas dentro de los servicios Web no dejan explícitamente constancia de quien fue el cliente que solicito dicha operación, lo cual permite en caso de un conflicto que ambas partes puedan negar o cambiar las condiciones bajo las cuales se realizo la transacción.

Ante estos riesgos los servicios Web necesitan proveerse de mecanismo de seguridad, entre los cuales se presentan:

- **Integridad de datos:** proporcionar la integridad de los datos significa que si los datos son modificados, ya sea de forma intencional por terceros o debido a fallas en la comunicación, se pueda determinar que así ha ocurrido y que es necesario repetir la transmisión.

- **Encriptación de datos:** para asegurar la privacidad de los datos transmitidos es necesario que si alguien obtiene acceso al canal de comunicación y puede ver los datos no los logre entender; para ello se han creado mecanismo de cifrado y descifrado, permitiendo únicamente a las partes involucradas conocer el significado del mensaje.

- **Autenticación:** si se desea asegurar el no-repudio de transacciones, es necesario un mecanismo que permita establecer quienes son los involucrados en dicha operación. La autenticación es el proceso por medio del cual cada una de las partes involucradas provee un medio de identificación a la otra.

- **Autorización:** el acceso a los recursos se puede lograr a través del proceso de autenticación, pero poder utilizarlos, es decir, permitir la invocación de una operación y ejecución de sus tareas conlleva el otorgamiento de una serie de privilegios; esto es la autorización, controlar el acceso a los recursos limitando que puede hacer el cliente sobre la base de sus privilegios.

Cómo se pueden implementar los diversos mecanismos de seguridad ha sido una de las principales preocupación de la industria ante el crecimiento en la utilización de los servicios Web. Los mecanismos comúnmente utilizados para asegurar las comunicaciones por Internet no se adecuan completamente a los servicios Web.

2.2.4.1 *Secure Socket Layer* (SSL) y *Transport Layer Security* (TLS)

SSL y TLS son protocolos cuya finalidad es brindar comunicaciones seguras en Internet haciendo uso de la criptografía permitiendo con esto el despliegue de una Infraestructura de Clave Publica (PKI, *Public Key Infrastructure*)[6]. Ambos protocolas utilizan la criptografía por medio de claves publicas y privadas, estas claves están relacionadas matemáticamente y sirven de entrada para los algoritmos de encriptación. Comúnmente se presentan por medio de Certificados Digitales, los cuales funcionan como una identificación, en cuyo contenido se encuentran los datos del dueño, la entidad emisora del certificado, la firma digital y la clave publica del dueño entre otros datos.

[6] PKI: una infraestructura de clave pública es una combinación de hardware y software, políticas y procedimientos que permiten asegurar la identidad de los partes involucradas en una transacción utilizando para ello criptografía pública.

Una comunicación bajo SSL nos brinda autenticación, encriptación e integridad de los datos, ya que las partes involucradas establecen una serie de algoritmos bajo los cuales se realiza la conexión. La conexión en este caso es de punto a punto, es decir, desde el cliente hasta el servidor. Para acceder a un sitio Web este escenario no presenta ningún problema pero en el caso de los servicios Web se debe asegurar no solo la conexión sino cada uno de los mensajes SOAP, ya que estos pueden pasar varios puntos intermedios hasta llegar a su destino final.

En los servicios Web debemos tener una comunicación segura entre los llamados *end-points*, los cuales son el origen y el destino del mensaje, no importando cuantos puntos intermedios éste atraviese nadie debe poder alterar o conocer su contenido. Incluso si el mensaje es muy grande debe existir un método para cifrar únicamente una parte del mensaje evitando de esta forma una sobrecarga de procesamiento debido a la encriptación. Debido a estos requerimientos es que SSL no es del todo adecuado en lo que respecta a la seguridad de los servicios Web.

2.2.4.2 *Firewalls*

Un *firewall* o cortafuegos es un dispositivo de hardware o software que sirve para prohibir ciertas comunicaciones dependiendo de las políticas de seguridad establecidas. Es un mecanismo de seguridad que puede funcionar en distintas capas del modelo OSI[7], prohibiendo las comunicaciones según filtros de direcciones MAC, direcciones IP, puertos o incluso URLs (en el caso de un Proxy). Los *firewalls* funcionan como un punto de control de acceso.

[7] OSI: *Open System Interconection*, el modelo OSI es un modelo de red descriptivo creado por ISO y que sirve como referencia para establecer la arquitectura de red a utilizar. Se divide en siete capas: Física, Enlace de Datos, Red, Transporte, Sesión, Presentación y Aplicación. Cada contiene especificaciones para su implementación.

En un ambiente como Internet permiten salvaguardar los recursos de accesos no autorizados o por medio de canales de comunicación no convencionales. Un sitio Web es comúnmente accedido por medio del protocolo HTTP o HTTPS, puerto 80 o 443 respectivamente, permitiendo cerrar el acceso a los demás puertos por seguridad. El inconveniente con los servicios Web es que su punto de entrada es comúnmente también por el protocolo HTTP o HTTPS y dicha comunicación no es filtrada según su contenido; por lo tanto, los mensajes SOAP que viajan sobre HTTP pueden contener datos que hagan operar al servicio Web de una forma inesperada o hagan uso de servicios Web que aún no han sido ofrecidos. Es por ello que los *firewalls* solo son uno de los diferentes mecanismos de seguridad que se deben implementar y claramente no son uno valido en el caso de los servicios Web.

2.2.4.3 Servicios Web seguros (*WS-Security*)

Los esfuerzos por lograr la creación y ejecución de servicios Web seguros han llevado a la creación e implementación del estándar *WS-Security* o WSS. El estándar WSS es estándar de comunicación presentado por OASIS y que busca extender SOAP para el soporte de seguridad a nivel de mensaje. Proporciona un marco de trabajo seguro para las comunicaciones entre *end-points*, brindando los servicios de Integridad de Mensajes, Confidencialidad de Mensajes y Autenticación de Mensajes. Este marco de trabajo proporciona la especificación para la construcción de protocolos de seguridad para intercambio de mensajes SOAP, dicha especificación es flexible ya que provee un conjunto de mecanismos para la implementación.

WSS permite definir un modelo de seguridad de mensaje abstracto, esto es, permite definir un *token* de seguridad combinado con una firma digital lo cual permite un mecanismo de autenticación. Al referirse al *token* de seguridad la especificación es abierta por lo que éste puede ser desde una dupla usuario/contraseña hasta un certificado digital (X.509) pasando por autenticación básica o *Kerberos*. Así mismo permite firmar y/o cifrar cualquier parte de un mensaje SOAP, ya sea el *body*, el *header*, un *attachment* o cualquier combinación de estos. La especificación permite su extensión para soportar incluso múltiples firmas de diversas entidades que intervengan en la comunicación. La confidencialidad y la integridad de los datos esta dada por la utilización de *XML Encryption* y *XML Signature*, pero esto no significa que excluya la utilización de una infraestructura PKI.

WSS no es la única especificación respecto a seguridad de servicios Web pero si es la base sobre la cual se edifica, algunas especificaciones con las que se relaciona son:

- ***WS-Policy*:** especificación que permite a los servicios Web la utilización de XML para anunciar sus políticas sobre seguridad, calidad de servicio, etc. Compatible con WSDL 1.1, UDDI 2.0 y UDDI 3.0. Versión actual 1.2 enviada a W3C el 25 de abril de 2006.

- ***WS-Trust*:** especificación que describe un modelo sobre el establecimiento de comunicaciones confiables directas e indirectas manejando los diferentes *tokens* de seguridad. Esta especificación es un esfuerzo conjunto de varias empresas, entre ellas IBM, Microsoft y VeriSign, por el momento su estado es un borrador público (Public *Draft*) emitido en febrero de 2005 sometido a revisión y evolución.

- ***WS-Privacy*****:** especificación que indica como se pueden implementar políticas de privacidad dentro de los servicios Web, describe un modelo para embeber un lenguaje de privacidad en las descripciones de WS-Policy y como los mecanismos de WS-Trust pueden evaluar estos requisitos de privacidad para ambas partes de la comunicación. Es un trabajo en progreso realizado por IBM, Microsoft y VeriSign entre otros.

- ***WS-Secure Conversation*****:** especificación que busca establecer un contexto de seguridad sobre el cual se llevan a cabo las operaciones y se basa en tokens de seguridad y el manejo de estos por WS-Trust. Otro esfuerzo por parte de IBM, Microsoft, VeriSign y otros. Por el momento su estado es un borrador público (Public Draft) emitido en febrero de 2005 sometido a revisión y evolución.

- ***WS-Federation*****:** especificación que busca la estandarización para compartir el manejo de identidades a través de distintos sistemas de autenticación y autorización en ambiente distribuidos. Es también parte del marco de trabajo de seguridad establecido por IBM, Microsoft, VeriSign y otros. Su estado actual es de borrador público (*Public Draft*) emitido en julio de 2003 sometido a revisión y evolución.

- ***WS-Authorization*****:** especificación que busca ser un estándar para autorización de datos y manejo de políticas de acceso dentro de los servicios Web.

Así como el concepto de seguridad abarca un campo muy grande en la informática en general, así es la búsqueda de implementación de seguridad en los servicios Web, y solamente un acercamiento de forma global y consistente nos puede brindar la seguridad necesaria en la publicación de servicios Web.

Figura 15 Ejemplo mensaje SOAP utilizando *WS-Security*

```
<?xml version="1.0" encoding="utf-8"?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
        wssecurity-secext-1.0.xsd"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
       wssecurity-utility-1.0.xsd"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
 <S11:Header>
  <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
                             wssecurity-secext-1.0.xsd">
   <wsse:UsernameToken wsu:Id=" MyID ">
    <wsse:Username>...</wsse:Username>
   </wsse:UsernameToken>
   <ds:Signature>
    <ds:SignedInfo>
     <ds:CanonicalizationMethod Algorithm= "http://www.w3.org/2001/10/xml-exc-c14n#"/>
     <ds:SignatureMethod Algorithm= "http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
     <ds:Reference URI="#MsgBody">
      <ds:DigestMethod Algorithm= "http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
     </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
    <ds:KeyInfo>
     <wsse:SecurityTokenReference>
      <wsse:Reference URI="#MyID"/>
     </wsse:SecurityTokenReference>
    </ds:KeyInfo>
   </ds:Signature>
  </wsse:Security>
 </S11:Header>
 <S11:Body wsu:Id="MsgBody">
  ...
 </S11:Body>
</S11:Envelope>
```

La Figura 15 es un ejemplo de un mensaje SOAP utilizando WS-Security. Como se aprecia en la figura, al *header* del mensaje se le ha agregado el elemento *Security* que contendrá la información referente a la seguridad. El elemento dentro de *Security* define el tipo de token de seguridad a utilizar (*UsernameToken*, *BinarySecurityToken* o *EncriptedData* si se utiliza encriptación). El elemento *Signature* contiene la información acerca de la firma digital según la especificación *XML Signature*: que parte del documento se firmo, con qué algoritmo y donde puede verificarse el token de seguridad.

2.2.5 Los servicios Web y las plataformas de desarrollo

De este modo se completa el marco de trabajo de los servicios Web, el cual plantea las áreas básicas a cubrir para el correcto funcionamiento de los servicios Web. Partiendo de la información que procesan, pasando por los métodos de comunicación, hasta la descripción de sus funcionalidades y de ellos mismos mediante directorios de servicios. Además de estas tecnologías, los servicios Web permiten la extensión de su funcionalidad, y debido a ello, existen más tecnologías que se les pueden acoplar como la seguridad, el alto rendimiento, la alta disponibilidad, etc.

Las plataformas de desarrollo permiten la extensión e implementación de los servicios Web orientados al marco de trabajo donde se desenvuelven, de este modo, añadiendo más funcionalidad según las extensiones que se realicen. Todo esto siempre sin perder la inter-operabilidad, clave en los servicios Web, que al ser basados en estándares abiertos permiten una implementación por parte de cualquier fabricante.

2.2.6 Los servicios Web y la plataforma .NET

Visual Studio .NET[8], la plataforma de Microsoft, fue lanzada junto con la idea de la evolución de las aplicaciones hacia los servicios Web. Es por ello que las tecnologías, de la cuales hacen uso, vienen integradas dentro de la plataforma. Para el desarrollo de servicios Web, .NET hace uso de las cuatro tecnologías mencionadas anteriormente: XML, SOAP, WSDL y UDDI. Permitiendo de este modo un marco de trabajo basado en los estándares y buscando la simplicidad de los conceptos.

[8] La versión de *Visual Studio .NET* que se hace referencia en este trabajo es la versión 2005

Además existe el *add-on* llamado *Web Services Enhancements* (WSE), para *Visual Studio .NET* y el *.NET Framework*, que permite extender la funcionalidad de los servicios Web y adaptarse a la evolución de los estándares sobre los que se basan. Así mismo por medio del WSE, Microsoft permite la generación de servicios Web que sean conformes al *Basic Profile* definido por la WS-I. A continuación se presenta como .NET hace uso de las tecnologías que conforman los servicios Web.

2.2.6.1 .NET y SOAP

La implementación de SOAP en .NET permite la construcción de aplicaciones distribuidas, proporcionando una integración con el motor de ejecución del lenguaje común y permitiendo la creación de constructores, delegados, métodos sobrecargados, paso de parámetros, jerarquía de clases, interfaces, métodos, propiedades, campos y cálculos de objetos entre aplicaciones utilizando un flujo de objetos en los SOAP *Headers* independientemente de los parámetros. .NET ofrece una elevada compatibilidad de SOAP para funciones como mensajes unidireccionales y SOAP *Headers*.

Además .NET permite trabajar SOAP con cualquiera de los lenguajes que soporta, utiliza XML para el consumo y generación de mensajes siendo compatible con el estándar de SOAP 1.1 y también la versión sucesora SOAP 1.2, incluyendo la codificación de la sección 5 de SOAP (SOAP *encoding*), permitiendo una sólida inter-operabilidad con otras aplicaciones SOAP.

2.2.6.2 .NET y XML

El framework de .NET proporciona un conjunto completo e integrado de clases que permiten el trabajo sobre documentos y datos XML, permitiendo de esta forma, una utilización nativa del componente fundamental de los servicios Web: el XML. Las clases XML que están incorporadas en el framework funcionan como elementos básicos para desarrollar una solución abierta, inter-operable y compatible con los estándares de la W3C. Las clases incorporadas se pueden dividir en diferentes grupos, según la funcionalidad: de análisis y escritura (*XmlReader*, *XmlWriter*), de validación (*XmlValidatingReader*), edición de documentos (*XmlDocument*), manejo de XSLT (eX*tensible Stylesheet Language Transformation*, por ejemplo conversión de XML a HTML), edición de esquemas XSD (*XslTransform*, *XmlSchema*, *XPath*) y otras.

Además de las clases ya mencionadas, .NET permite la extensión de dichas clases mediante el uso de clases base abstractas y métodos virtuales, permitiendo de esta forma el desarrollo de nuevas implementaciones para diferentes orígenes de datos. Entre los APIs que expone se encuentra *XPathNavigator* el cual incorpora un motor de consultas de *XPath*, para la realización de consultas tipo *select* contra sistemas de archivos, registros y bases de datos relacionales. Así como *XPath* permite su extensibilidad, lo mismo ocurre con las demás clases relacionadas con el manejo de XML.

El *namespace* que encierra el manejo de XML es *System.Xml*, y dentro de las normas W3C que lo definen se pueden mencionar las siguientes:

- XML 1.0, incluida la compatibilidad con DTD
- Espacios de nombres XML, a nivel de secuencias y DOM[9] (*Document Object Model*)
- Esquemas XSD
- Expresiones *XPath* (versión 1.0 W3C)
- Transformaciones XSLT (versión 1.0 W3C)
- *Core DOM Level 1*
- *Core DOM Level 2*

Entre los objetivos de .NET en la implementación de XML y su uso en los servicios Web están la inter-operabilidad, extensibilidad, rendimiento e integración; lo cual trata de lograr realizando una implementación nativa de XML y adoptando los estándares dictados por el consorcio W3C.

[9] *Document Object Model*: el modelo de objetos de documento ofrece un API para la definición y manipulación de documentos estructurados, por ejemplo HTML y XML, de forma independiente del lenguaje o plataforma. Un estándar de W3C.

2.2.6.3 .NET y WSDL

La plataforma .NET es compatible con la especificación WSDL 1.1 y la generación de los documentos los hace de manera automática, es decir, no es necesario realizar la codificación de un documento WSDL de forma manual. De igual forma permite la codificación y modificación de documentos WSDL proveyendo los siguientes Namespaces para ello: *System.Web.Services*, *System.Web.Services.Protocols* y *System.Xml.Serialization*. *Serialization* es el nombre que se le da al proceso de convertir objetos a XML, proceso realizado por ejemplo al realizar una comunicación con un servicio Web, ahí se convierte de un mensaje SOAP a una llamada de una función .NET. *Deserialization* es el proceso inverso.

La herramienta para la generación de un documento WSDL se llama Disco.exe, además de la generación automática del documento WSDL, .NET trae, entre sus utilitarios, una herramienta, llamada Wsdl.exe, para la generación de código de un cliente para consumir servicios Web. Dicha herramienta se basa en un documento WSDL para la generación del código, el código que genera es una clase que implementa un cliente de dicho servicio Web. Basado en la información del documento WSDL hace la conversión que más se adecue con los tipos de datos, la conversión puede no ser exacta por lo cual siempre en conveniente revisar dicho código fuente y realizar las correcciones necesarias.

2.2.6.4 .NET y UDDI

La plataforma .NET, al igual que con las demás tecnologías detrás de los servicios Web, incorpora una implementación para el trabajo sobre UDDI, permite la generación automática de un documento UDDI a partir de un documento WSDL; y al igual que permite la manipulación de los archivos WSDL también ofrece un API para la manipulación de los archivos UDDI. El API que ofrece .NET para trabajar con UDDI brinda la posibilidad de realizar el registro de forma programática, lo cual facilita las actualizaciones que se puedan realizar a los servicios Web que se estén exponiendo.

Microsoft inicio el trabajo sobre la versión 1 de UDDI, junto con otras compañías (más de 200, entre ellas IBM, Oracle y SAP). Ahora .NET maneja la versión 2 de UDDI gracias al Microsoft UDDI SDK 2.0, siendo compatible con el .NET Framework 2.0. Las características del UDDI SDK se limitan al manejo de la especificación 2.0 de UDDI siendo no soportadas las especificaciones 1.0 o 3.0. Incluye varios *Namespaces* para una mayor interacción en el proceso del registro, entre los *Namespaces* se encuentran: *Businesses*, *Extensions*, *Services*, *TModels* y UDDI. Entre las características que ofrece la implementación dentro de .NET resaltan la integración con Active Directory para la administración de Microsoft Enterprise UDDI Services, la clase *GetRelatedCategories* que permite navegar entre las entradas de un categoría y la clase *ManagedUrl* que maneja *fail over* y *round-robin* para un conjunto de *access points* (puntos de acceso).

De esta forma .NET engloba todas las tecnologías necesarias para el desarrollo, publicación y utilización de los servicios Web. Además .NET ofrece, como se menciono anteriormente, un *add-on* llamado *Web Services Enhancements* (WSE). Actualmente se encuentra en la versión 3.0.

2.2.6.5 *Web Services Enhancements* para Microsoft .NET

Web Services Enhancements (WSE) es un *add-on* para la plataforma .NET, y tal y como lo indica su nombre ofrece una mayor facilidad para potenciar el desarrollo de servicios Web. Entre las características que permite incluir dentro de los servicios Web se pueden mencionar: implementación de seguridad de extremo a extremo así como una mayor inter-operabilidad entre los servicios Web, basándose en especificaciones abiertas de la industria. De una forma más detallada WSE presenta las siguientes funcionalidades a incluir en los servicios Web:

- **Manejo de seguridad a nivel de mensaje.** Provee las herramientas necesarias para manejar la seguridad a nivel de mensaje en lugar de manejar a nivel de comunicación, es decir sin utilización de SSL/TLS (*Secure Socket Layer* / *Transport Layer Security*). Permite la utilización de firmas digitales y certificados digitales para asegurar ya sea todo o solo parte del mensaje, adaptándose a la especificación *WS-Security*.

- **Inter-operabilidad con *Windows Communication Foundation* (WCF).** *Windows Communication Foundation* es una tecnología de servicios Web y un API unificado que trata de resolver las dificultades para la creación de sistemas conectados dentro y fuera de la organización. Se basa en tres principios: compatibilidad integrada con un amplio conjunto de protocolos de servicios Web, uso implícito de principios de desarrollo centrados en los servicios y un solo API para la creación de sistemas conectados.

- **Soporte MTOM, recomendación de la W3C.** *Message Transmisión Optimization Mechanism* permite el envío de gran cantidad datos binarios dentro de los mensajes SOAP de forma segura y eficiente. La recomendación de W3C es un reemplazo a la especificación *WS-Attachments* para el envío de datos adjuntos.

2.2.6.6 Seguridad en los servicios Web .NET

En la plataforma .NET la implementación de la seguridad se realiza por medio del *add-on* llamado WSE (*Web Services Enhancements*). Dicho *add-on* permite aplicar credenciales a nivel de mensaje, cifrado y firmas digitales a los mensajes SOAP independientes del protocolo de transporte. La plataforma .NET junto con WSE implementan a través de diferentes clases la seguridad conforme a la especificación *WS-Security* y esquemas asociados.

Los tres principales servicios de seguridad que brinda WSE son:

2.2.6.6.1 Credenciales de seguridad

La funcionalidad de credenciales de seguridad es poder autenticar al cliente que invoca el servicio Web, al incluir estas credenciales en el mensaje SOAP se permite mantener la seguridad incluso al pasar por intermediarios. Las credenciales de seguridad pueden ser desde una dupla usuario/password, certificados digitales X.509 hasta credenciales personalizadas.

La implementación de esta funcionalidad se lleva a cabo utilizando las siguientes *namespaces*:

- System.Security.Cryptography
- Microsoft.Web.Services3

En la Figura 16 se presenta el código necesario para habilitar el envío de un usuario y password como credenciales de seguridad en la aplicación cliente; el procedimiento que se lleva a cabo requiere realizar un *override* al método SecureMessage instanciando el *token* de seguridad para luego añadirlo al *header* de SOAP.

Figura 16 Implementando credenciales de seguridad

```
public override void SecureMessage(SoapEnvelope envelope, Security security)
{
    UsernameToken userToken;
    userToken = new UsernameToken(userName, userPasswordEquivalent, PasswordOption.SendNone);
    security.Tokens.Add(userToken);
}
```

Así como el cliente debe configurarse para enviar las credenciales de seguridad el servicio Web debe estar configurado para verificar dichas credenciales. El archivo web.config encargado de las configuraciones del servicio Web debe incluir el elemento soapServerProtocolFactory que define la utilización de WSE para el manejo de la seguridad.

La figura 17 muestra un ejemplo del código que evalúa si las credenciales de seguridad son las exigidas por el servicio Web. Al igual que en la aplicación cliente, en este caso, el método ValidateMessageSecurity requiere un *override* que verifica el *header* SOAP en busca de las credenciales requeridas. Con las credenciales necesarias se pueden realizar las validaciones que se consideren convenientes, permitiendo incluso hacer la validación con servicios de directorio como *Active Directory*.

Figura 17 Verificación de credenciales de seguridad

```
public override void  ValidateMessageSecurity(SoapEnvelope envelope, Security security)
{
    bool IsSigned = false;
    foreach (ISecurityElement element in security.Elements)
    {
        if (element is MessageSignature)
        {
            MessageSignature sign = element as MessageSignature;
            SignatureOptions expectedOptions = SignatureOptions.IncludeTimestamp |
                                               SignatureOptions.IncludeSoapBody |
                                               SignatureOptions.IncludeTo |
                                               SignatureOptions.IncludeAction |
                                               SignatureOptions.IncludeMessageId;
            if ((sign.SignatureOptions & expectedOptions) == expectedOptions)
            {
                if (sign.SigningToken is UsernameToken)
                    IsSigned = true;
            }
        }
    }
    if (!IsSigned)
        throw new SecurityFault("No Security Credentials");
}
```

2.2.6.6.2 Firma digital

La utilización de una firma digital en los mensajes SOAP permite verificar que el contenido del mensaje no ha sido alterado en el camino. Para realizar el proceso de firma de un mensaje SOAP se puede hacer uso de diferentes *tokens* de seguridad, entre los cuales se puede mencionar certificados digitales X.509, *ticket* de Kerberos, usuario/password y *tokens* personalizados.

Para realizar el firmado digital de un mensaje SOAP primero se debe establecer un *token* de seguridad como en el apartado anterior, para luego instanciar la clase MessageSignature con el token obtenido y agregarlo al *header* de SOAP. La Figura 18 muestra un ejemplo del código necesario a agregar en el método SecureMessage.

Figura 18 Firma digital del mensaje SOAP

```
MessageSignature sig = new MessageSignature(userToken);
security.Elements.Add(sig);
```

El procedimiento que debe llevar a cabo el servicio Web para verificar la firma digital es similar al que se muestra en la Figura 17. La seguridad implementada en estos ejemplos se puede extender aún más, por ejemplo encriptando el *token* de seguridad a utilizar y firmando otras partes del mensaje de SOAP.

2.2.6.6.3 Encriptación

Tanto el uso de un *token* de seguridad como la firma digital de un mensaje SOAP son puntos importantes en la seguridad de las comunicaciones, sin embargo, incluso con la implementación de ambos el contenido del mensaje es transmitido en texto plano. Pero es gracias a estos dos servicios de seguridad que se puede implementar la encriptación del mensaje SOAP, evitando así que si alguien intercepta el mensaje pueda saber su contenido.

La encriptación que se puede llevar a cabo puede ser simétrica o asimétrica, es decir, con una clave compartida o por medio de un par de claves pública y privada. Para realizar el proceso de encriptación se recomienda no utilizar el token de seguridad usuario/password, comúnmente llamado UsernameToken sino generar el EncryptedKeyToken.

La Figura 19 muestra un ejemplo de la instrucción a agregar después de la generación de un *token* de seguridad y la obtención de la firma digital para indicar que se encripte el mensaje SOAP.

Figura 19 Encriptación del mensaje SOAP

```
security.Elements.Add( new EncryptedData(userToken));
```

Para determinar si el mensaje está encriptado se debe realizar un override del método ValidateMessageSecurity, después de determinar el método y el token de seguridad utilizado para la encriptación se procede al descifrado del mensaje. La Figura 20 presenta un ejemplo del método ValidateMessageSecurity.

Figura 20 Determinación si el mensaje SOAP está encriptado

```
public override void  ValidateMessageSecurity(SoapEnvelope envelope, Security security)
{
    bool IsEncrypted = false;
    foreach (ISecurityElement element in security.Elements)
    {
        if (element is EncryptedData)
        {
            EncryptedData encrypt = element as EncryptedData;
            if (encrypt.SecurityToken is UsernameToken)
                IsEncrypted = true ;
        }
    }
    if (!IsEncrypted)
        throw new SecurityFault("No Security Credentials");
}
```

Los mensajes SOAP generados tanto por el cliente como por el servicio Web son conformes a la especificación *WS-Security* por lo que la estructura del mensaje se puede ver en la Figura 15.

2.2.7 Los servicios Web y la plataforma J2EE

Java 2 Enterprise Edition[10] o J2EE es una serie de especificaciones para el desarrollo de aplicaciones empresariales de diversos tipos, entre ellas los servicios Web. La arquitectura presentada por J2EE para el desarrollo de los servicios Web se basa, por la misma naturaleza de los servicios, en los pilares antes mencionados: SOAP, XML, WSDL, UDDI. El nombre que se le otorga a cada una de estas tecnologías en este contexto varia, al igual que su implementación, pero lo que no varía es el proposito de cada una de ellas. Por lo tanto se establece que el desarrollo de los servicios Web bajo J2EE debe enfocase en solucionar el paso de información, la comunicación, la descripción y el descubrimiento en los servicios Web.

J2EE incluye dentro de la plataforma las herramientas necesarias para el desarrollo de servicios Web, y brinda la inter-operabilidad necesaria para la arquitectura de dichos servicios permitiendo la creación de servicios Web conformes con el *Basic Profile* definido por la WS-I. La manera de presentar las herramientas de desarrollo de J2EE es por medio de su *Java 2 SDK Enterprise Edition 1.4* (J2EE 1.4 SDK), el cual incluye entre muchas otras utilidades, el Java API para RPC basado en XML (JAX-RPC).

[10] La versión de *Java 2 Enterprise Edition* que se hace referencia en este trabajo es la versión 1.4

El JAX-RPC permite el desarrollo de servicios Web, inter-operables y portables, basados en SOAP y utilizando WSDL para la descripción de los servicios. La versión del API disponible en esta plataforma es la 1.1, además la plataforma soporta la especificación de servicios Web para J2EE (JSR 921) del *Java Community Process*. También hay varios APIs que permiten una mayor funcionalidad en los servicios Web, y el JWSDP versión 2.0 (*Java Web Services Developer Pack*) para facilidad de desarrollo; por ejemplo, implementando seguridad, traspaso de datos adjuntos y enrutamiento, todo bajo estándares definidos. A continuación se presentan los pilares de los servicios Web bajo la perspectiva de J2EE.

2.2.7.1 J2EE y SOAP

La utilización de SOAP dentro de la plataforma J2EE se encuentra de forma nativa, y utilizando el API de Java JAX-RPC (*Java API for XML-based RPC*) soporta llamadas remotas a procedimientos (RPC, R*emote Procedure Call*) en las aplicaciones utilizando un protocolo basado en XML. JAX-RPC 1.1 permite el desarrollo de servicios Web portables e inter-operables basándose en SOAP, facilitando así que puedan ser consumidos tanto por clientes Java como clientes no-Java.

El API de Java JAX-RPC reduce la complejidad de la creación de servicios Web estandarizando la creación de solicitudes y repuestas SOAP, estandarizando la utilización de parámetros y el proceso de *deploy*, facilitando la construcción con SOAP proveyendo librerías o herramientas que realizan estas funciones, proveyendo soporte para diferentes escenarios de mapeos entre tipos de datos: WSDL/XML a Java y viceversa. Al trabajar con JAX-RPC, todos los detalles de SOAP son manejados en *background*, de este modo es necesario únicamente conocer el API y no su implementación interna.

JAX-RPC soporta tres modos de operación:

- *Modo Solicitud – Respuesta Síncrono.* Después que un método remoto es invocado, el cliente del servicio se bloquea hasta que es retornada una respuesta, ya sea un valor o una excepción.

- *Modo RPC de una vía.* Después que un método remoto es invocado, el cliente del servicio continúa su procesamiento sin bloquearse; ningún valor o excepción es esperado como respuesta.

- *Modo Invocación RPC sin bloqueo.* El cliente del servicio, en este caso, invoca un método remoto y continúa con su procesamiento sin bloquearse; más adelante el cliente procesa el valor de retorno realizando un "pedido" para el valor retornado.

JAX-RPC además de soportar SOAP, especifica una forma estándar para adicionar manejadores de mensajes, permitiendo así un pre y post procesamiento de las solicitudes y respuestas SOAP; permitiendo al servicio Web la realización de mayor procesamiento.

La plataforma J2EE también presenta el API de Java para SOAP con adjuntos (SAAJ, *SOAP with Attachments API for Java*), el cual permite producir y consumir mensajes SOAP con datos adjuntos conforme a la especificación SOAP 1.2 y la nota *SOAP with Attachments*. SAAJ 1.2 permite un trabajo sobre los mensajes SOAP más directo y la posibilidad de soportar adjuntos en los mensajes, ya sé documentos XML, fragmentos XML u otros tipos MIME.

Al trabajar con SAAJ se puede hacer uso de los siguientes modos de intercambio de mensajes:

- *Mensajes solicitud-respuesta síncronos*. El cliente envía un mensaje y espera por la respuesta.

- *Mensajes de una vía asíncronos*. El cliente envía un mensaje y continúa con el procesamiento sin esperar por una respuesta.

JAX-RPC y SAAJ son tecnologías que se encuentran dentro de la plataforma J2EE por lo que la portabilidad e inter-operabilidad es inherente.

2.2.7.2 J2EE y XML

El procesamiento de XML en la plataforma J2EE se puede realizar por medio del API de Java para el procesamiento de XML (JAXP, *Java APIs for XML Processing*). JAXP 1.2 es un conjunto liviano de APIs para el parseo y procesamiento de documentos XML. JAXP no es una tecnología adoptada recientemente dentro de J2EE, éste viene utilizándose desde los inicios de J2SE (*Java 2 Standard Edition*), ahora se le ha agregado el soporte de esquemas XML.

JAXP procesa los documentos XML utilizando los modelos SAX[11] (*Simple API for Java*) o DOM, así mismo permite el uso de motores XSLT durante el procesamiento del documento. La plataforma J2EE presenta también JAXB (*Java Architecture for XML Binding*), tecnología que permite la creación de una representación estándar tipo objeto de un documento XML en memoria.

JAXB esta formado por tres componentes principales: *binding compiler,* *c*ompilador que crea las clases Java, llamadas *content classes*, para un esquema dado; b*inding framework*, el cual provee los servicios en tiempo de corrida que pueden ser ejecutados en las *content classes*; *binding language*, lenguaje que describe el enlace entre las clases de Java y el esquema, permitiendo trabajar sobre las *content classes* de una manera más directa.

2.2.7.3 J2EE y WSDL

La generación de documentos WSDL para los servicios Web construidos con JAX-RPC en la plataforma J2EE se realiza de forma automática, J2EE provee la herramienta *wscompile*. Dicha herramienta permite la generación de un documento WSDL a partir del código fuente Java y también permite la generación de un cliente Java a partir de un documento WSDL. Con ello permite la construcción de clientes Java que consuman servicios Web no importando si ellos son servicios no construidos bajo la plataforma J2EE.

[11] SAX (*Simple API for Java*): es un API que ofrece un parser de acceso serial para documentos XML, esta implementado como un modelo manejado por eventos.

2.2.7.4 J2EE y UDDI

J2EE incluye el API de Java para registros XML (*JAXR, Java API for XML Registries*), este API permite acceder a los registros de negocios y a las especificaciones de los registros de UDDI y ebXML (e-business XML, *Electronic Business using eXtensible Markup Language*).

JAXR 1.0 se compone de dos partes: una especificación de proveedor JAXR específica al registro y un proveedor JAXR conectado (*pluggable*). El proveedor JAXR *pluggable* esconde los detalles específicos del registro. Ambos trabajan en conjunto para realizar el registro del servicio Web en los directorios de servicios así como para hacer la búsqueda de servicios Web.

2.2.7.5 *Java Web Services Developer Pack*

Java Web Services Developer Pack (JWSDP) es un *toolkit* integrado y gratuito con el que se puede desarrollar, probar y publicar aplicaciones XML, aplicaciones Web y servicios Web, utilizando las últimas tecnologías de servicios Web que a la vez son implementaciones estándar. Java WSDP 2.0 comprende en su "*integrated stack*" de tecnología JAX-WS 2.0 EA, JAXB 2.0 EA, SAAJ 1.3 EA y Fast Infoset 1.0.1 FCS, brindando una actualización y mayor funcionalidad a las herramientas que permiten el desarrollo de servicios Web.

JWSDP define un marco de trabajo para el desarrollo de servicios Web sobre la base de todas las tecnologías ya incorporadas en el estándar J2EE, e incluye nuevas versiones que permitan de una manera integrada trabajar con cada una de las facetas de los servicios Web: XML, definiciones WSDL, registros UDDI, SOAP y seguridad, por mencionar las principales. Además JWSDP incorpora compatibilidad hacia atrás, esto es, se incorporan versiones anteriores de los componentes de JWSDP.

De esta forma J2EE ofrece un conjunto de APIs enfocados en el desarrollo de servicios Web, altamente funcionales, integrados, inter-operables y basados en estándares de la industria.

2.2.7.6 Seguridad en los servicios Web J2EE

La plataforma J2EE, bajo el *toolkit* JWSDP, permiten el desarrollo de servicios Web seguros aplicando la autenticación a través de usuario/password, utilizando firma digital de mensajes SOAP y realizando la encriptación de mensajes SOAP. La especificación *WS-Security* es implementada bajo el *Web Services Security framework* (*XWS-Security* o XWSS) incluido en el JWSDP. Los principales paquetes a utilizar al trabajar con la seguridad se encapsulan en:

- *java.xml.crypto*
- *java.security*

2.2.7.6.1 Autenticación

El procedimiento de autenticación permite al servicio Web determinar la identidad del cliente. La implementación de la autenticación está basada en las especificaciones de WS-I y OASIS. Las especificaciones utilizadas en el caso de *XML Signature* y *XML Encryption* son las implementaciones realizadas por Apache, XML-Dsig y XML-Enc respectivamente.

Las configuraciones necesarias del lado del cliente como del lado del servicio Web se realizan por medio de archivos de configuración XML. Los elementos para realizar la autenticación pueden ser desde una dupla usuario/password, certificados digitales X.509 o *tokens* de seguridad personalizados.

2.2.7.6.2 Firma digital

Para realizar un procesamiento adecuado de la seguridad en los mensajes SOAP es muy importante la utilización de firmas digitales. Con la utilización de firmas digitales se puede garantizar que la información, que es transportada por el mensaje, no ha sufrido ninguna modificación. La firma digital se puede aplicar a solo una parte del mensaje o a todo el mensaje.

La implementación de los procesos de firma de mensajes así como su verificación requiere la utilización de diversas funciones. La Figura 21 muestra el código necesario para llevar a cabo la firma digital de un mensaje SOAP. El código presentado en la figura fue tomado del ejemplo *genenveloped* incluido en el JWSDP 2.0. El procedimiento que se lleva a cabo en dicho ejemplo consiste en la instanciación del mensaje a firmar, la obtención de las claves y definición de los algoritmos a utilizar, la generación de la firma y por último la aplicación de la misma, obteniendo como resultado el mensaje firmado.

Como se puede apreciar, al momento de instanciar un objeto SignedInfo se especifica toda la información referente a la firma digital que posteriormente aparecerá en el *header* del mensaje SOAP bajo el elemento homónimo SignedInfo mostrado en la Figura 15.

La Figura 22 a su vez presenta parte del código necesario para realizar la validación de la firma digital incluida en un mensaje SOAP. El procedimiento es similar al que se lleva cabo para realizar la firma ya que se debe instanciar el mensaje a validar, luego se procede a encontrar el elemento de la firma y con ello la información que permitirá validar la firma. El código mostrado en la figura fue tomado del ejemplo *validate* incluido en el JWSDP 2.0.

Figura 21 Firma digital del mensaje SOAP

```
    // Create a DOM XMLSignatureFactory that will be used to generate the
// enveloped signature
    String providerName = System.getProperty
        ("jsr105Provider", "org.jcp.xml.dsig.internal.dom.XMLDSigRI");
XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM",
     (Provider) Class.forName(providerName).newInstance());
    // Create a Reference to the enveloped document (in this case we are
// signing the whole document, so a URI of "" signifies that) and
// also specify the SHA1 digest algorithm and the ENVELOPED Transform.
Reference ref = fac.newReference
    ("", fac.newDigestMethod(DigestMethod.SHA1, null),
         Collections.singletonList
      (fac.newTransform
    (Transform.ENVELOPED, (TransformParameterSpec) null)),
     null, null);
// Create the SignedInfo
SignedInfo si = fac.newSignedInfo
    (fac.newCanonicalizationMethod
     (CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS,
      (C14NMethodParameterSpec) null),
     fac.newSignatureMethod(SignatureMethod.DSA_SHA1, null),
     Collections.singletonList(ref));
    // Create a DSA KeyPair
    KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
kpg.initialize(512);
    KeyPair kp = kpg.generateKeyPair();
    // Create a KeyValue containing the DSA PublicKey that was generated
KeyInfoFactory kif = fac.getKeyInfoFactory();
    KeyValue kv = kif.newKeyValue(kp.getPublic());
// Create a KeyInfo and add the KeyValue to it
    KeyInfo ki = kif.newKeyInfo(Collections.singletonList(kv));
// Instantiate the document to be signed
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
Document doc =
    dbf.newDocumentBuilder().parse(new FileInputStream(args[0]));
    // Create a DOMSignContext and specify the DSA PrivateKey and
    // location of the resulting XMLSignature's parent element
DOMSignContext dsc = new DOMSignContext
    (kp.getPrivate(), doc.getDocumentElement());
// Create the XMLSignature (but don't sign it yet)
XMLSignature signature = fac.newXMLSignature(si, ki);
    // Marshal, generate (and sign) the enveloped signature
    signature.sign(dsc);
```

Figura 22 Validación de la firma digital

```
// Instantiate the document to be validated
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
Document doc =
        dbf.newDocumentBuilder().parse(new FileInputStream(args[0]));
// Find Signature element
NodeList nl =
    doc.getElementsByTagNameNS(XMLSignature.XMLNS, "Signature");
if (nl.getLength() == 0) {
    throw new Exception("Cannot find Signature element");
}
// Create a DOM XMLSignatureFactory that will be used to unmarshal the
// document containing the XMLSignature
String providerName = System.getProperty
        ("jsr105Provider", "org.jcp.xml.dsig.internal.dom.XMLDSigRI");
XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM",
        (Provider) Class.forName(providerName).newInstance());
// Create a DOMValidateContext and specify a KeyValue KeySelector
    // and document context
DOMValidateContext valContext = new DOMValidateContext
    (new KeyValueKeySelector(), nl.item(0));
// unmarshal the XMLSignature
XMLSignature signature = fac.unmarshalXMLSignature(valContext);
// Validate the XMLSignature (generated above)
boolean coreValidity = signature.validate(valContext);
// Check core validation status
if (coreValidity == false) {
        System.err.println("Signature failed core validation");
    boolean sv = signature.getSignatureValue().validate(valContext);
    System.out.println("signature validation status: " + sv);
    // check the validation status of each Reference
    Iterator i = signature.getSignedInfo().getReferences().iterator();
    for (int j=0; i.hasNext(); j++) {
    boolean refValid =
        ((Reference) i.next()).validate(valContext);
    System.out.println("ref["+j+"] validity status: " + refValid);
    }
} else {
        System.out.println("Signature passed core validation");
}
}
```

2.2.7.6.3 Encriptación

La encriptación del contenido de los mensajes es la utilización última de las firmas digitales y métodos de autenticación, permitiendo asegurar la transmisión de mensajes entre el origen y el destino sin importar cuantos intermediarios pase el mensaje. Los mecanismos utilizados para llevar a cabo la encriptación en XWSS cumplen con todos los estándares anteriormente mencionados asegurando la inter-operabilidad de aplicaciones.

La estructura que presentan los mensajes, a quienes se les han aplicado diferentes mecanismos de seguridad, se puede apreciar en la Figura 15, ya que al utilizar los estándares estos deben ser conformes a la especificación *WS-Security*.

3. CONSTRUCCIÓN DE SERVICIOS WEB

3.1 Servicios Web

A continuación se presenta un mismo servicio Web desarrollado bajo las distintas plataformas planteadas en el presente trabajo. La funcionalidad del servicio Web es genérica y pretende mostrar las similitudes existentes en la implementación de los servicios Web. Así mismo se presenta la implementación de las aplicaciones clientes que consumen los servicios Web, en este caso son aplicaciones tipo consola para simplificar y centrar el entendimiento en la utilización de servicios Web.

La funcionalidad del servicio Web se resume en los siguientes enunciados:

- El servicio Web tiene como entrada un parámetro tipo *String*.
- El servicio Web realiza las operaciones que se hayan definido dentro de su estructura para procesar el parámetro y obtener un resultado.
- El servicio Web devuelve el resultado tipo *String*.

La funcionalidad de las aplicaciones clientes se resume en los siguientes enunciados:

- La aplicación cliente establece el valor del parámetro del servicio Web.
- La aplicación cliente realiza la llamada al servicio Web enviando el parámetro tipo *String*.
- La aplicación espera el resultado del servicio Web solicitado y lo despliega.

3.1.1 Servicio Web en .NET

Para la creación del presente servicio Web se utilizó el IDE[12] Microsoft Visual Studio 2005, el cual brinda la interfaz necesaria para trabajar sobre los diferentes archivos de código fuente y accesos abreviados para la compilación y ejecución de las aplicaciones. Además incluye un servidor Web integrado sobre el cual se ejecuta el servicio Web, brindando un ambiente de pruebas completo. El lenguaje seleccionado fue C# por su similitud con Java.

La Figura 23 presenta el contenido del archivo asmx, el cual es el punto de entrada al servicio Web y en cuyo contenido se implementa el código del servicio Web. Para el presente ejemplo se utilizó la opción denominada *CodeBehind*, la cual permite separar del archivo asmx el código de programación que implementa el servicio Web para facilitar la comprensión.

Figura 23 Archivo Service.asmx

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Service.cs" Class="Service" %>
```

[12] IDE (*Integrated Development Environment*): ambiente integrado para desarrollo de aplicaciones.

En la Figura 24 se presenta el código fuente para implementar el servicio Web. Éste consiste en la inclusión de los *Namespaces* necesarios para implementar el servicio Web en la clase denominada Servicio y los llamados *WebMethods*, que son las operaciones que expone el servicio Web.

Figura 24 Código fuente servicio Web .NET

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://wsdemo.edu/dotnet_demo")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service () {

    }

    [WebMethod]
    public string Funcion(string Parametro) {
        string Resultado;
        Resultado = "Valor Procesado: " + Parametro;
        return Resultado;
    }

}
```

El documento WSDL que describe el servicio Web mostrado en la Figura 25 se genera automáticamente con las herramientas provistas por .NET. Este documento es necesario para el entendimiento, por parte de terceros, de las operaciones que expone el servicio Web.

Figura 25 Documento WSDL del servicio Web .NET

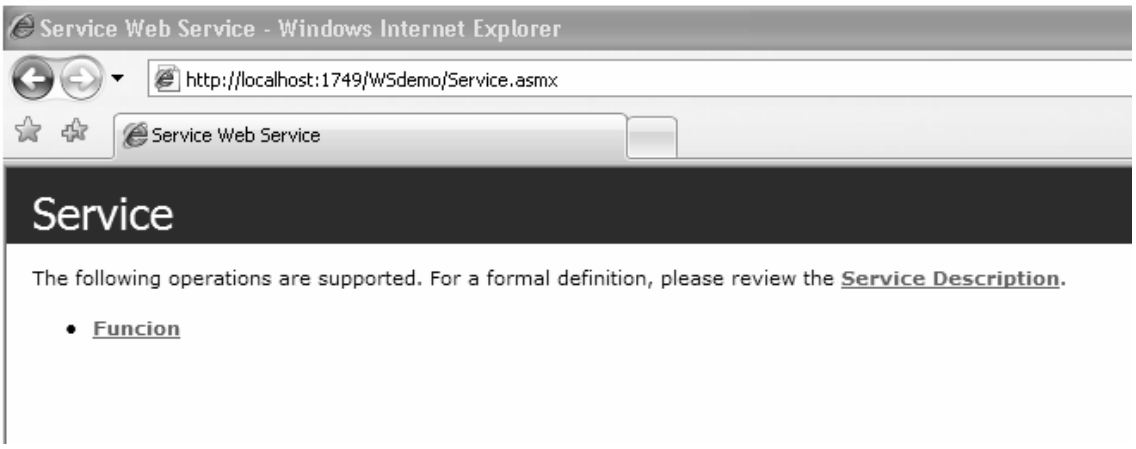
```
<?xml version="1.0" encoding="utf-8" ?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tm=
"http://microsoft.com/wsdl/mime/textMatching/" xmlns:soapenc=
"http://schemas.xmlsoap.org/soap/encoding/" xmlns:mime=
"http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://wsdemo.edu/dotnet_demo"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12=
"http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:http=
"http://schemas.xmlsoap.org/wsdl/http/" targetNamespace=
"http://wsdemo.edu/dotnet_demo"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
 <wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace=
  "http://wsdemo.edu/dotnet_demo">
   <s:element name="Funcion">
    <s:complexType>
     <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="Parametro" type="s:string" />
     </s:sequence>
    </s:complexType>
   </s:element>
   <s:element name="FuncionResponse">
    <s:complexType>
     <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="FuncionResult" type="s:string" />
     </s:sequence>
    </s:complexType>
   </s:element>
  </s:schema>
 </wsdl:types>
 <wsdl:message name="FuncionSoapIn">
  <wsdl:part name="parameters" element="tns:Funcion" />
 </wsdl:message>
 <wsdl:message name="FuncionSoapOut">
  <wsdl:part name="parameters" element="tns:FuncionResponse" />
 </wsdl:message>
 <wsdl:portType name="ServiceSoap">
  <wsdl:operation name="Funcion">
   <wsdl:input message="tns:FuncionSoapIn" />
   <wsdl:output message="tns:FuncionSoapOut" />
  </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="ServiceSoap" type="tns:ServiceSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="Funcion">
   <soap:operation soapAction="http://wsdemo.edu/dotnet_demo/Funcion" style=
   "document" />
```

Figura 26 Continuación Documento WSDL del servicio Web .NET

```
    <wsdl:input>
     <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal" />
    </wsdl:output>
   </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ServiceSoap12" type="tns:ServiceSoap">
   <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
   <wsdl:operation name="Funcion">
    <soap12:operation soapAction="http://wsdemo.edu/dotnet_demo/Funcion"
    style="document" />
    <wsdl:input>
     <soap12:body use="literal" />
    </wsdl:input>
    <wsdl:output>
     <soap12:body use="literal" />
    </wsdl:output>
   </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Service">
   <wsdl:port name="ServiceSoap" binding="tns:ServiceSoap">
    <soap:address location="http://localhost:1749/WSdemo/Service.asmx" />
   </wsdl:port>
   <wsdl:port name="ServiceSoap12" binding="tns:ServiceSoap12">
    <soap12:address location="http://localhost:1749/WSdemo/Service.asmx" />
   </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

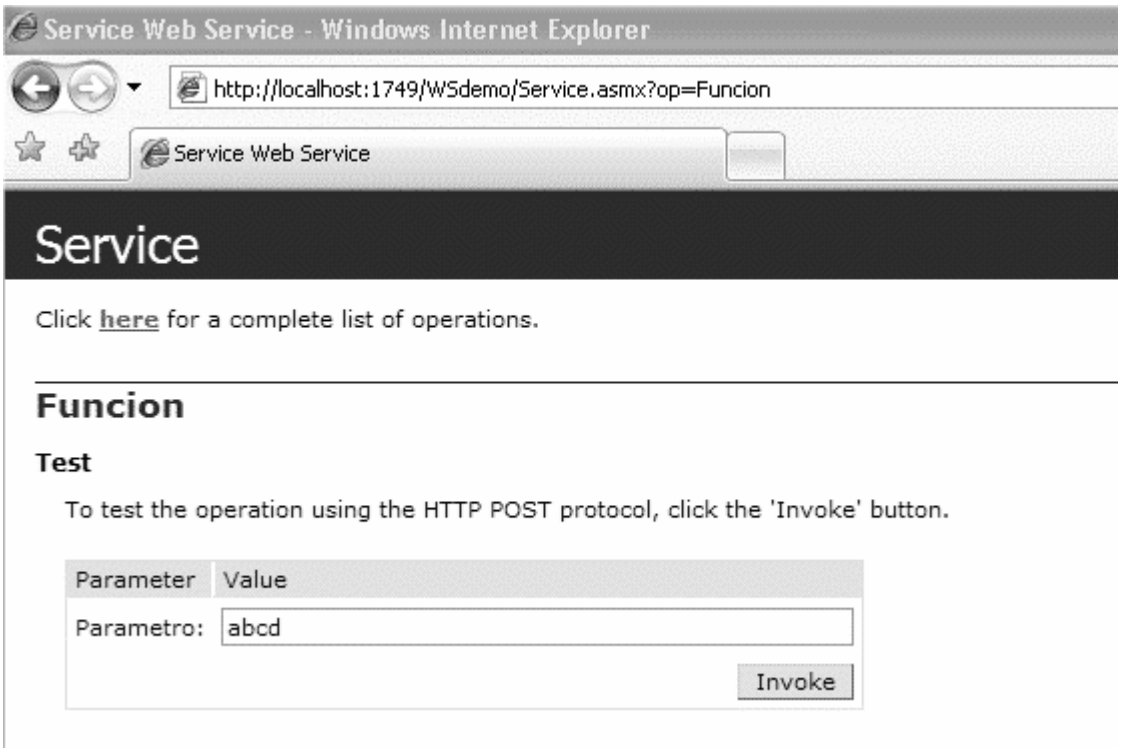
En la Figura 26 continúa el documento WSDL que describe al servicio Web creado en .NET y se puede identificar en el *tag address location* del elemento *service* la URL donde se encuentra el servicio Web. La Figura 27 muestra la página inicial que se presenta al ingresar al URL definido para el servicio Web, ésta proporciona el documento de descripción (WSDL), un listado de las operaciones soportadas, así como la posibilidad de realizar pruebas con ellas.

Figura 27 Página inicial del servicio Web .NET



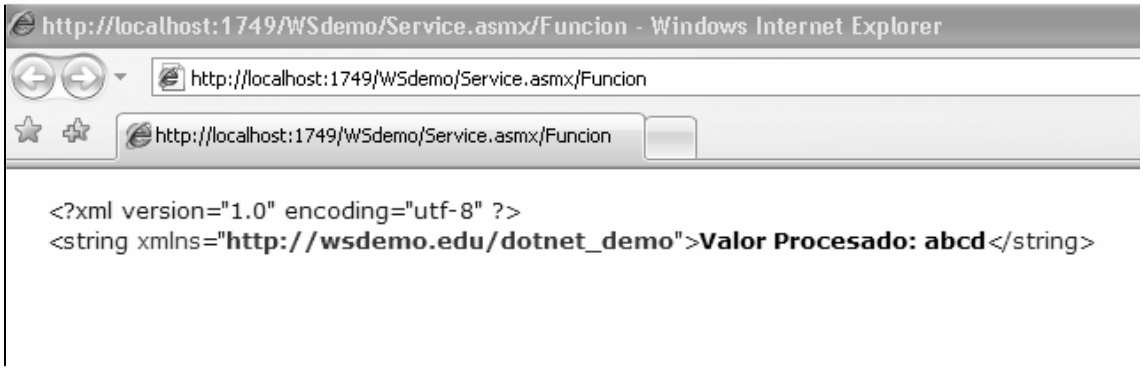
La Figura 28 muestra la página de prueba de la operación expuesta por el servicio Web junto con los parámetros requeridos, en este caso un parámetro tipo *String* y el valor enviado “abcd”.

Figura 28 Página de prueba del servicio Web .NET



La Figura 29 muestra la respuesta obtenida después de invocar la operación “Funcion” del servicio Web. Ésta consiste en el *String* “Valor Procesado: abcd”.

Figura 29 Respuesta del servicio Web .NET



El servicio Web creado en .NET es completamente funcional y cualquier aplicación puede consumirlo para utilizar la operación que expone.

3.1.2 Servicio Web en J2EE

En la creación del servicio Web de ejemplo sobre la plataforma J2EE se utilizo el IDE Oracle JDeveloper 10.1.2.0.2, como se comentó anteriormente, J2EE representa un estándar y permite que terceros implementen diferentes ambientes de desarrollo para trabajar sobre el J2EE SDK. La utilización del IDE acelera la construcción de aplicaciones automatizando y facilitando tareas de modelado, contracción, *deployment*, etc.

La Figura 30 muestra el código fuente del servicio Web, el cual se encapsula en el paquete WSdemo y consiste en una clase denominada Service con un método llamado “Funcion” al igual que su contraparte en .NET.

Figura 30 Código Fuente Servicio Web J2EE

```
package WSdemo;

public class Service
{
  public Service()
  {
  }

  /**
 *
 * @webmethod
 */
  public String Funcion(String Parametro)
  {
    String Resultado;
    Resultado = "Valor Procesado: " + Parametro;
    return Resultado;
  }

}
```

El archivo web.xml es el archivo descriptor de *deployment* y contiene la información necesaria para tratar la clase Service como un servicio Web a exponer. La estructura y la información que presenta se encuentran definidas en la especificación J2EE. Entre está información se encuentran las clases con las cuales se implementa la funcionalidad del servicio Web. En la Figura 31 se encuentra el contenido del archivo web.xml para el servicio Web construido.

Figura 31 Archivo web.xml

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <description>Empty web.xml file for Web Application</description>
  <servlet>
    <servlet-name>MyWebService1</servlet-name>
    <servlet-class>oracle.j2ee.ws.StatelessJavaRpcWebService</servlet-class>
    <init-param>
      <param-name>class-name</param-name>
      <param-value>WSdemo.Service</param-value>
    </init-param>
    <init-param>
      <param-name>interface-name</param-name>
      <param-value>WSdemo.IMyWebService1</param-value>
    </init-param>
  </servlet>
  <servlet>
    <servlet-name>Service</servlet-name>
    <servlet-class>oracle.j2ee.ws.StatelessJavaRpcWebService</servlet-class>
    <init-param>
      <param-name>class-name</param-name>
      <param-value>WSdemo.Service</param-value>
    </init-param>
    <init-param>
      <param-name>interface-name</param-name>
      <param-value>WSdemo.IService</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyWebService1</servlet-name>
    <url-pattern>/MyWebService1</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Service</servlet-name>
    <url-pattern>/Service</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>35</session-timeout>
  </session-config>
  <mime-mapping>
    <extension>html</extension>
    <mime-type>text/html</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>txt</extension>
    <mime-type>text/plain</mime-type>
  </mime-mapping>
</web-app>
```

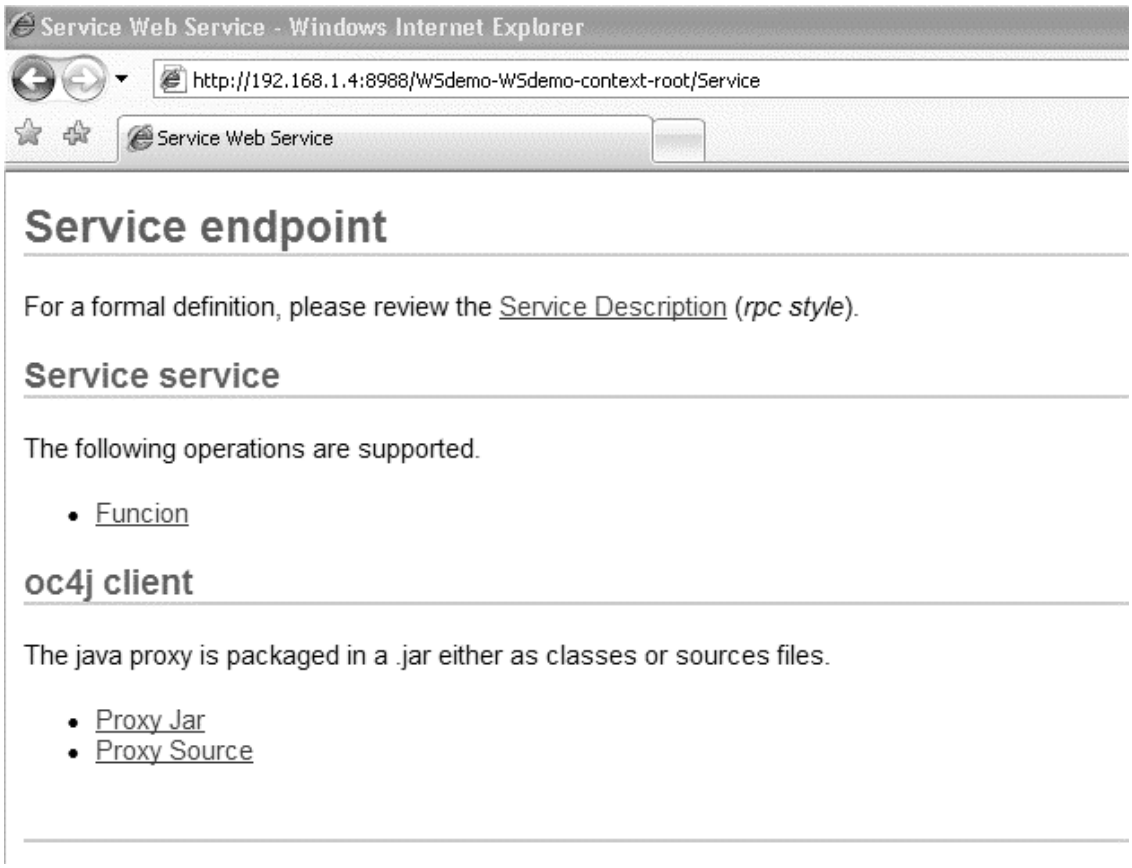
Además de la generación de los archivos necesarios para el *deployment* del servicio Web, también es necesaria la generación del archivo WSDL, este archivo permitirá a terceros conocer las operaciones expuestas en el servicio y así poder consumirlo. La generación del archivo WSDL se hace de forma automática utilizando las herramientas provistas por el IDE a través del SDK de J2EE. La Figura 32 muestra el archivo WSDL para este servicio Web.

Figura 32 Documento WSDL del servicio Web J2EE

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="Service" targetNamespace="http://WSdemo/Service.wsdl" xmlns=
"http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns=
"http://WSdemo/Service.wsdl" xmlns:ns1="http://WSdemo/IService.xsd">
 <types>
  <ns0:schema targetNamespace="http://WSdemo/IService.xsd" xmlns=
"http://www.w3.org/2001/XMLSchema" xmlns:SOAP-ENC=
"http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns0=
"http://schemas.xmlsoap.org/wsdl/" />
 </types>
 <message name="FuncionORequest">
  <part name="Parametro" type="xsd:string" />
 </message>
 <message name="FuncionOResponse">
  <part name="return" type="xsd:string" />
 </message>
 <portType name="ServicePortType">
  <operation name="Funcion">
   <input name="FuncionORequest" message="tns:FuncionORequest" />
   <output name="FuncionOResponse" message="tns:FuncionOResponse" />
  </operation>
 </portType>
 <binding name="ServiceBinding" type="tns:ServicePortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
   <operation name="Funcion">
    <soap:operation soapAction="" style="rpc" />
    <input name="FuncionORequest">
     <soap:body use="encoded" namespace="Service" encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output name="FuncionOResponse">
     <soap:body use="encoded" namespace="Service" encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
   </operation>
 </binding>
 <service name="Service">
  <port name="ServicePort" binding="tns:ServiceBinding">
   <soap:address location="http://higher:8888/WSdemo-WSdemo-context-root/Service" />
  </port>
 </service>
</definitions>
```

Ejecutando el servicio Web en el servidor Web integrado a la herramienta de desarrollo, se puede ingresar al URL donde está publicado el servicio Web. Esta página inicial permite conocer la descripción del servicio Web, un listado de las operaciones soportadas y realizar pruebas con ellas. La Figura 33 muestra la página inicial del servicio Web.

Figura 33 Página inicial del servicio Web J2EE



En la Figura 34 se muestra la página de prueba de la operación "Funcion" expuesta por el servicio Web y el parámetro requerido tipo *String* con el valor "abcd". Y la Figura 35 muestra la respuesta de la invocación al servicio Web, la cual consiste en el *String* "Valor procesado: abcd".

Figura 34 Página de prueba del servicio Web J2EE

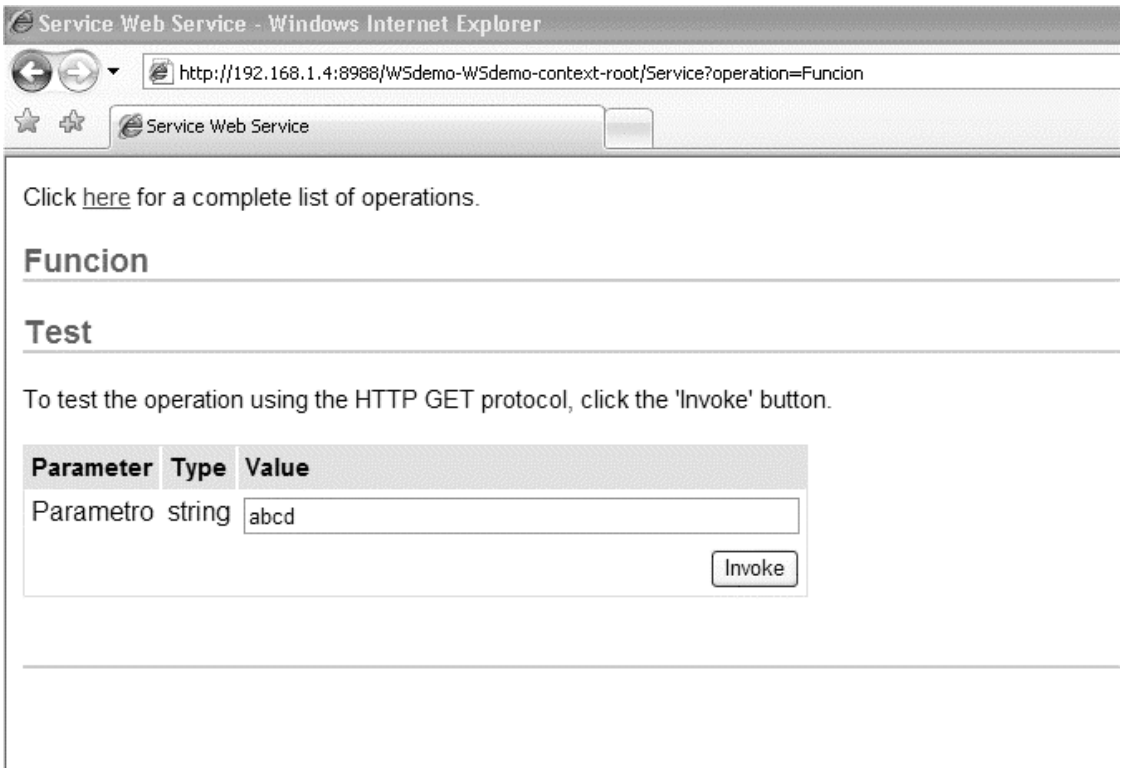
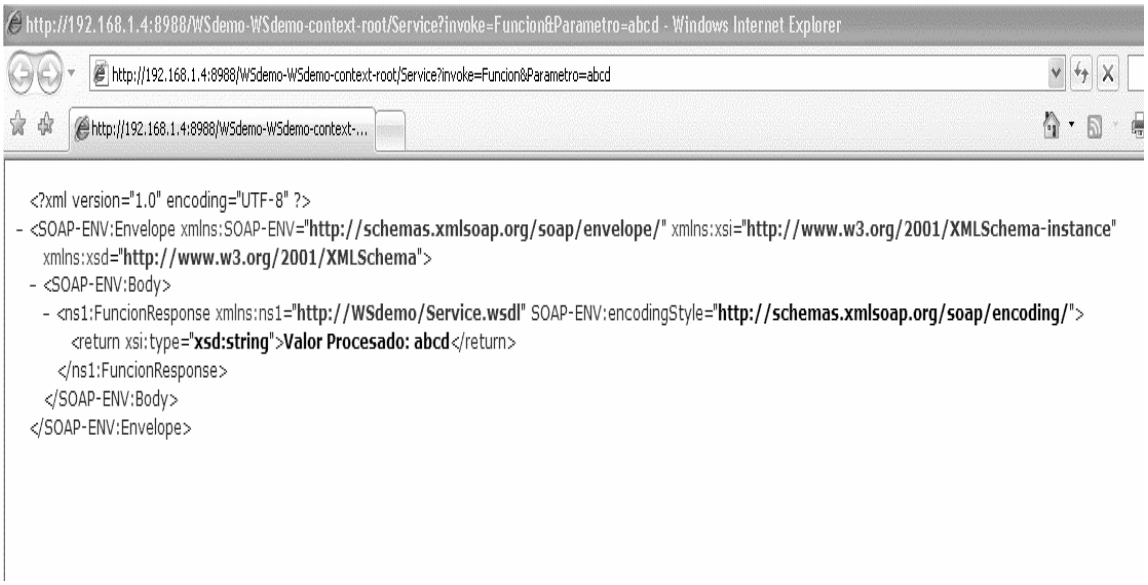


Figura 35 Respuesta del servicio Web J2EE



3.2 Aplicaciones cliente para los servicios Web

Cuando los servicios Web ya se encuentran publicados en un servidor de aplicaciones, los clientes ya pueden empezar a consumirlos. Lo primero que se realiza es el descubrimiento a través de los directorios UDDI, encontrando el servicio Web que brinde la funcionalidad necesaria. El siguiente paso es utilizar el documento WSDL para obtener la descripción de las operaciones que expone el servicio Web y cómo utilizarlas. Finalmente, se incluye el código necesario en la aplicación cliente para invocar el servicio Web.

3.2.1 Aplicación cliente en .NET

Si una aplicación quiere hacer uso de las operaciones de un servicio Web, es necesario la utilización de una clase denominada proxy, la cual permite la invocación de los métodos asociados al servicio Web que se desea consumir. El documento WSDL sirve de entrada para la generación automática de dicha clase por parte de .NET.

Para la creación de la clase proxy es necesario indicar la URL donde se puede encontrar el documento WSDL. Indicada la URL, .NET realiza las operaciones necesarias para permitir consumir el servicio Web, las cuales incluyen agregar las referencias correspondientes que permiten realizar la instancia de la clase proxy y su utilización dentro de la aplicación. La Figura 36 muestra el código utilizado para invocar la operación “Funcion” del servicio Web de ejemplo.

Figura 36 Código fuente aplicación cliente .NET

```
using System;
using System.Collections.Generic;
using System.Text;

namespace WScliente
{
    class Program
    {
        static void Main(string[] args)
        {
            //Declaracion de Variables
            string ResultadoWS;
            string Variable;

            //Declaracion de clase proxy para acceder al Web Service
            WSdemo.Service proxy = new WSdemo.Service();

            //Llamada a la operacion Funcion del Web Service
            Variable = "abcd";
            ResultadoWS = proxy.Funcion(Variable);
            System.Console.WriteLine(ResultadoWS);
        }
    }
}
```

3.2.2 Aplicación cliente J2EE

Cuando se desea hacer uso de un servicio Web en una aplicación es necesario la utilización de una clase denominada proxy, bajo la arquitectura J2EE se le llama stub, la cual permite utilizar los métodos asociados a las operaciones que expone el servicio Web. Para la creación de la clase Proxy es necesaria la URL hacia el documento WSDL o en su defecto se puede obtener, si está disponible, de la página inicial del servicio Web la clase ya empaquetada para incluirla en la aplicación.

La Figura 37 muestra el código fuente de la aplicación cliente J2EE, la primera instrucción identifica la utilización de la clase proxy obtenida a partir del documento WSDL, luego se instancia la clase y la invocación a la operación “Funcion” del servicio Web.

Figura 37 Código fuente aplicación cliente J2EE

```
import Prox.Service;

public class WScliente
{
  public WScliente()
  {
  }

  /**
 *
 * @param args
 */
  public static void main(String[] args) throws Exception
  {
    //Declaracion de Variables
    String ResultadoWS;
    String Variable;

    //Declaracion de clase Proxy para acceder al Web Service
    ServiceStub proxy = new ServiceStub();

    //Llamada a la operacion Funcion del Web Service
    Variable = "abcd";
    ResultadoWS = proxy.Funcion(Variable);
    System.out.println(ResultadoWS);
  }
}
```

3.3 Tecnologías servicios Web en .NET y J2EE

El servicio Web de ejemplo pone de manifiesto la similitud existente entre las plataformas .NET y J2EE para la creación y utilización de los servicios Web. Debido al paradigma que representan los servicios Web y las tecnologías necesarias para implementarlos, tecnologías que poco a poco son transformadas en estándares de la industria, las plataformas de desarrollo y ejecución planteadas en el presente trabajo, brindan herramientas similares que proveen funcionalidades acordes a las necesidades establecidas. Ambas plataformas buscan cubrir el mismo mercado, los servicios Web, y por lo tanto presentan y presentarán funciones muy parecidas sino es que iguales.

Entre los motivos por los cuales ambas plataformas son muy similares, en el caso de los servicios Web, se debe a que las compañías que las engendraron son parte, al igual que muchas otras compañías, de las organizaciones o comités que participan en la especificación de los estándares de los servicios Web y las tecnologías involucradas. La Tabla II muestra en resumen la implementación realizada por .NET y J2EE de las tecnologías utilizadas en servicios Web.

Tabla II Implementación de tecnologías por plataforma

		P L A T A F O R M A	
		.NET	**J2EE**
T E C N O L O G Í A	*Información*	XML 1.0, XSD XPath 1.0, XSD, XSLT 1.0 Core DOM Level 1 Core DOM Level 2	XML 1.0, XSD XPath 1.0, JAXB 1.0 JAXP 1.2, XSLT 1.0 SAX 1.0 y 2.0, DOM 1.0 y 2.0
	Comunicación	SOAP 1.1, 1.2	JAX-RPC (SOAP 1.1, 1.2)
	Descripción	WSDL 1.1	JAXR 1.0 (WSDL 1.1)
	Localización	UDDI 2.0	UDDI 2.0, ebXML
	Seguridad	*WS-Security* (WSE)	*WS-Security* (XWSS)
	Mejoras	WSE 3.0 MTOM (*WS-Attachments*)	JWSDP 2.0 SAAJ 1.2 (*SOAP with Attachments*)

La evolución de los servicios Web provoca que las mismas plataformas se transformen e implementen mejoras, muchas veces con componentes extras, para mantenerse siempre soportando todas aquellas tecnologías que permitan el diseño, modelado y desarrollo de servicios Web seguros, confiables y altamente inter-operables.

3.4 Ventajas y desventajas

Las plataformas .NET y J2EE presentan muchas similitudes, sin embargo, existen diferencias. Dichas diferencias pueden ser tomadas como ventajas o desventajas de una plataforma sobre otra. Según objetivo a cumplir se debe analizar qué alternativa se implementará, y si la plataforma escogida satisface las necesidades presentadas. En la Tabla III se confrontan las diferencias generales presentadas entre las plataformas .NET y J2EE.

Tabla III Ventajas o desventajas entre .NET y J2EE

	.NET	J2EE
C A R A C T E R I S T I C A	Plataforma implementada y soportada principalmente por una sola compañía.	Especificación elaborada y soportada por un gran número de compañías.
	Implementación de los servicios Web desde la etapa de diseño. Plataforma pensada y orientada a los servicios Web.	Especificación realizada anteriormente al surgimiento de los servicios Web. Inclusión de los servicios Web sobre la base de APIs generados para ello.
	La implementación de las soluciones se puede realizar utilizando diferentes lenguajes de programación, dependiendo únicamente de su disponibilidad sobre la plataforma .NET.	La implementación de las soluciones se debe realizar utilizando únicamente el lenguaje de programación Java.
	Las soluciones por el momento únicamente se pueden ejecutar sobre sistemas operativos Windows.	Las soluciones se pueden implementar sobre distintos sistemas operativos y sobre distintas arquitecturas de procesador.

3.4.1 Curva de aprendizaje y tiempo de desarrollo

La evolución en la informática avanza a pasos agigantados, y las tecnologías de desarrollo no son una excepción y debido a esto es que el proceso de aprendizaje de las nuevas tecnologías debe ser de fácil asimilación. Poder trabajar siempre con la tecnología de punta requiere una actualización constante, y cuando surgen nuevos paradigmas es muy importante que la llamada “Curva de Aprendizaje” sea lo más rápida posible.

La curva de aprendizaje se refiere al tiempo que lleva a una persona aprender a desenvolverse con una nueva herramienta, en el presente caso las plataformas de desarrollo y las tecnologías de servicios Web. Los tiempos estimados para el proceso de familiarización con ambas plataformas y la implementación de servicios Web se fijó en 2 semanas por cada una. La dedicación para cada plataforma fue la misma, sin embargo, la información referente a .NET fue más fácil de asimilar y de poner en práctica los conocimientos aprendidos.

El por qué de la diferencia requiere de la observación de muchas y diferentes variables: disponibilidad de material, idioma del material, disponibilidad de recursos para practicar, experiencia previa con las herramientas, interés por las herramientas, facilidad del ambiente de desarrollo (IDE), entre otras. Todas estas variables tienen un rol muy importante en el proceso de aprendizaje, y por lo tanto, en la selección de una plataforma de desarrollo.

3.4.1.1 Visual Studio .NET

La plataforma .NET ha crecido de manera significativa con el transcurso de los años, incluyendo innumerables clases agrupadas en igual número de *namespaces*. Tanto recurso de programación requiere de un enfoque al momento de aproximarse al desarrollo con dicha herramienta. Visual Studio cuenta con un extenso soporte de parte de Microsoft, lo cual permite que el proceso de aprendizaje sea altamente efectivo. Los servicios que brinda a través de MSDN permiten adentrarse y especializarse en el tema de interés, cualquiera que fuere. Los *Developer Centers* existentes para cada tecnología y herramienta que brinda Visual Studio .NET son una fuente excepcional de información, tanto por parte de Microsoft como por parte de terceros que utilizan dichas tecnologías.

La posibilidad de utilizar más de un lenguaje para la creación de aplicaciones con .NET es una ventaja significativa para un desarrollo más acelerado. Si se realiza un estudio formal sobre el .NET Framework, la utilización de cada lenguaje dependerá únicamente de la sintaxis ya que previamente se obtuvo el conocimiento del funcionamiento e implementación de las clases.

En la construcción del servicio Web de ejemplo el tiempo de desarrollo fue sumamente rápido, .NET permite exponer cualquier método como operación del servicio Web. Además la tecnología *IntelliSense* que incorpora Microsoft en el IDE permite la generación de código de una forma más rápida ya que éste agrega el código que posiblemente falta por escribir en cada línea.

3.4.1.2 *Java 2 Enterprise Edition*

La plataforma J2EE es sumamente extensa, comprende una cantidad innumerable de paquetes y extensiones que se puede utilizar. Es gracias a ese crecimiento constante que mantiene que su utilización abarca muchos campos, y los servicios Web son uno de ellos. La evolución hizo que J2EE incluyera dentro de sus APIs los servicios Web y toda la especificación se vio extendida.

El proceso de aprendizaje de J2EE es más complejo que su contraparte .NET debido a su naturaleza de especificación, lo cual determina que existe un modelo y un método a seguir en la implementación de las diversas tecnologías que conforman una aplicación. .NET también tiene una metodología para el desarrollo de aplicaciones pero se ve limitado con las opciones para implementar cada tipo de tecnología a utilizar, es decir, para implementar un servicio Web por ejemplo, .NET trae las herramientas necesarias para hacerlo mientras de J2EE ofrece diversos enfoques para llevar a cabo la misma implementación. Al final si se utilizan bajo las especificaciones estándar ambos pueden llegar a presentar la misma funcionalidad.

Pero es esta extensibilidad de J2EE la que produce una primera aproximación más difícil de sobrellevar. El punto fuerte en el soporte del lado de J2EE es la creación y el sentido de pertenencia de una comunidad, implantado mucho antes que en .NET en los desarrolladores, y es por ello que la llamada *Java Comunity* permite a los desarrolladores adquirir mejor conocimiento a través de la experiencia compartida. No por ello queda de lado el soporte y toda la información que pone a disposición de todos Sun Microsystems por medio de su gran biblioteca en línea.

Para llevar a cabo el desarrollo de aplicaciones con J2EE se puede utilizar cualquiera de los muchos IDEs existentes, lo cual permite al usuario escoger aquel que le brinde mejores herramientas para desarrollar su trabajo. Para el desarrollo del servicio Web de ejemplo, el proceso de desarrollo de la funcionalidad del servicio Web fue la parte más lenta; ésta consistió en la creación de una clase de java, y la generación del servicio Web se basó simplemente en escoger la opción de publicar dicha clase como un servicio Web. En definitiva se puede exponer:

- La curva de aprendizaje de .NET tiende a ser menor que J2EE debido a la abundancia de lenguajes de programación a utilizar en el .NET Framework, la experiencia previa con cualquiera de ellos brinda una ventaja en la adquisición de nuevo conocimiento.

- El soporte para .NET se encuentra de forma más explicativa, tanto en las muchas páginas dedicadas a .NET por parte de terceros así como por lo sitios de Microsoft. En el caso de J2EE la mayoría de información hace referencia a la especificación y detalles técnicos. Existen también sitios de terceros dedicados a J2EE pero en una proporción aproximada de 1 por cada 15 de .NET (información obtenida sobre la base de búsquedas en el motor de búsqueda *Google*).

- El proceso de desarrollo puede llegar a ser igual en ambas plataformas, esto dependerá del grado de conocimiento de las mismas y la complejidad del proyecto a realizar. Ambas plataformas son completamente capaces de llevar cabo la implementación de soluciones de servicios Web que respeten las especificaciones estándar actuales, tanto de inter-operabilidad como seguridad.

3.4.2 Desempeño de los servicios Web

En la selección de una plataforma para la implementación de soluciones, en el presenta caso los servicios Web, no debe dejarse por un lado un punto muy importante como lo es el desempeño (*performance*). Además de los puntos tratados anteriormente el desempeño que presenten los servicios Web es un factor que se debe tomar en cuenta para la escalabiliada y extensibilidad futura.

Con los servicios Web, como en los demás tipos de aplicaciones, algunos de los tantos aspectos a tomar en cuenta son: diseño, plataforma (tanto de hardware como software) y ancho de banda, por mencionar los más generales. De una forma más detallada también cabe mencionar la cantidad de información a transmitir (tamaño del mensaje SOAP), número de usuarios concurrentes, número de solicitudes concurrentes, etc.

La preocupación por el desempeño llevo al W3C ha realizar una publicación sobre recomendaciones para mejorar el desempeño en los servicios Web. Dichas recomendaciones buscan atacar la principal causa asociada al bajo desempeño de los servicios Web: el manejo de datos binarios. En sí el problema se traduce en el consumo de ancho de banda y los tiempos de transmisión debido a mensajes SOAP de gran tamaño, esto debido a que un servicio Web no es más que una invocación remota, y el proceso de fabricación del mensaje, transmisión del mensaje y entendimiento del mismo son los factores de mayor relevancia en el consumo de los recursos.

Las Recomendaciones del W3C para mejorar el desempeño son:

- **XOP (*XML-binary Optimized Packing*):** su objetivo es proveer un método estándar para la inclusión de datos binarios, tal como son, en documentos XML formando un paquete. Con esto se logra reducir el tamaño del documento XML y con ello el ancho de banda a utilizar. Recomendación publicada el 25 de enero de 2005.

- **MTOM (*Message Transmisión Optimization Mechanism*):** su objetivo es hacer uso de las funcionalidad que provee XOP para aplicarlo a los mensajes SOAP, define una implementación utilizando HTTP y XOP para incluir datos binarios y el mensaje SOAP en un sobre MIME. Con esto se logra reducir el tamaño del mensaje y el proceso de codificación/descodificación de la data. Recomendación publicada el 25 de enero de 2005.

- **RRSHB (*Resource Representation SOAP Header Block*):** su objetivo es permitir que los destinatarios de los mensajes SOAP puedan acceder a representaciones, almacenadas en cache, de recursos externos. Permitiendo escoger la utilización de la data original almacenada de forma externa o la data en cache que acompaña al mensaje SOAP acelerando el procesamiento de la información. Recomendación publicada el 25 de enero de 2005.

3.4.2.1 Prueba de desempeño servicio Web

Muchas compañías y organizaciones han realizado pruebas de desempeño en casos de estudio similares, por ejemplo Sun Microsystems y su sitio Web *Pet Store* contra Microsoft Corporation y su sitio Web *PetShop*[13], muchas veces llegando a conclusiones contrarias, dependiendo de quien lleve a cabo el estudio. Un ejemplo de ello se puede encontrar en *theserverside.com* y *theserverside.net* cada una de las comunidades defiende su plataforma de forma sutil pero sin embargo llegando a la misma conclusión: el desempeño puede llegar a ser muy similar en ambas plataformas si se realiza el *tuning* adecuado. Refiriéndose a *tuning* como el proceso de aplicación de las mejores practicas en cada una de las fases del desarrollo de la aplicación.

Para la realización de la prueba de desempeño con el servicio Web de ejemplo, construido en el presente trabajo, se dispuso de los servicios Web bajo un servidor con las características presentadas en la Tabla IV.

Tabla IV Características servidor de servicios Web

Hardware	
Procesador	Intel Pentium 4, 2.0 GHz (20 x 100)
Memoria	1 GB (DDR SDRAM) (PC2100 @ 2x133MHz = 266MHz)
Software	
Sistema Operativo	Microsoft Windows XP Service Pack 2

[13] Información adicional en Sun.com, Microsoft.com, IBM.com y Oracle.com.

El utilitario utilizado para simular la realización de varias solicitudes al servicio Web fue AdvenNet QEngine 6, programa gratuito para la realización de pruebas de funcionalidad y desempeño en aplicaciones y servicios Web.

3.4.2.1.1 Resultados prueba de desempeño

El servicio Web evaluado presentaba la siguiente funcionalidad: recibía como parámetro un *string*, procedía a concatenarlo con otra variable tipo *string* y devolvía el resultado de dicha operación.

La Tabla V muestra las características de la prueba de desempeño realizada:

Tabla V Parámetros prueba de desempeño

Prueba servicio Web de Ejemplo		
Parámetro	***Valor***	***Explicación***
# Usuario iniciales	**1**	Número de clientes iniciales
# Incremento de usuarios	**5**	Cantidad de usuarios que se crearan cada x intervalo de tiempo
Intervalo (segundos)	**5**	Tiempo que indica el intervalo para incrementar x usuarios
Repetición de llamada (segundos)	**1**	Tiempo que indica cuando tiempo espera un cliente para repetir la llamada
Intervalo muestra (segundos)	**2**	Tiempo entre cada toma de muestra.

El objetivo de la prueba es determinar el desempeño del servicio Web bajo una situación de *stress*. Los resultados se presentan en las siguientes graficas.

Figura 38 Solicitudes manejadas por unidad de tiempo .NET

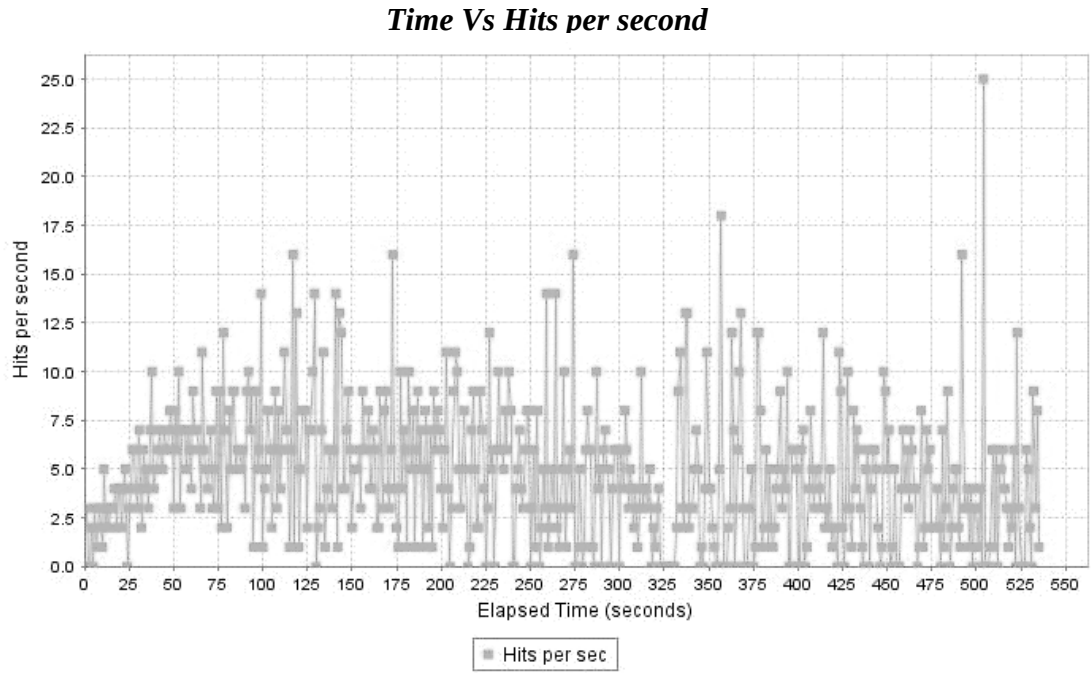
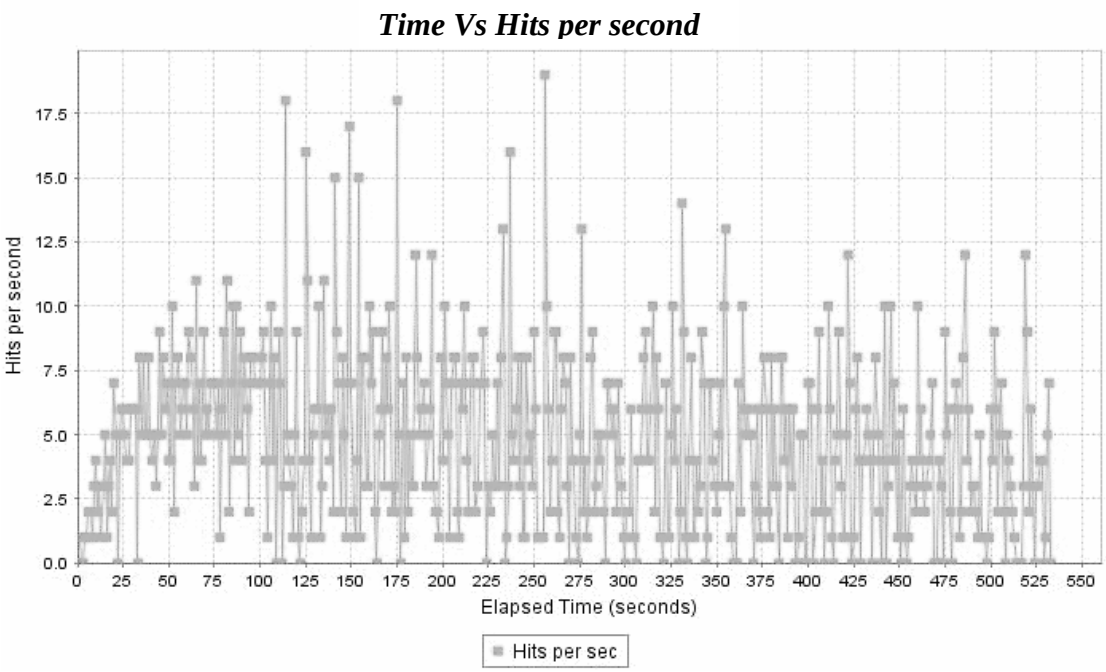


Figura 39 Solicitudes manejadas por unidad de tiempo J2EE



La Figura 38 y la Figura 39 muestran la cantidad de solicitudes simultáneas que fueron atendidas de manera exitosa por unidad de tiempo lo largo de la prueba. En el tipo de prueba realizada conforme avanza el tiempo se están generando cada vez más solicitudes debido al incremento de usuarios, y presenta mejor desempeño la grafica que muestra mayor cantidad de puntos cerca del eje horizontal. Esto debido a que significa que el servicio Web de .NET esta procesando de forma más rápida las solicitudes y no se acumulan.

La Figura 40 y la Figura 41 muestran el número total de solicitudes que se completaron de forma exitosa a lo largo del tiempo. En este caso la grafica muestra gran similitud y efectivamente se completaron el mismo número de solicitudes de forma correcta.

Figura 40 Solicitudes completadas a lo largo del tiempo .NET

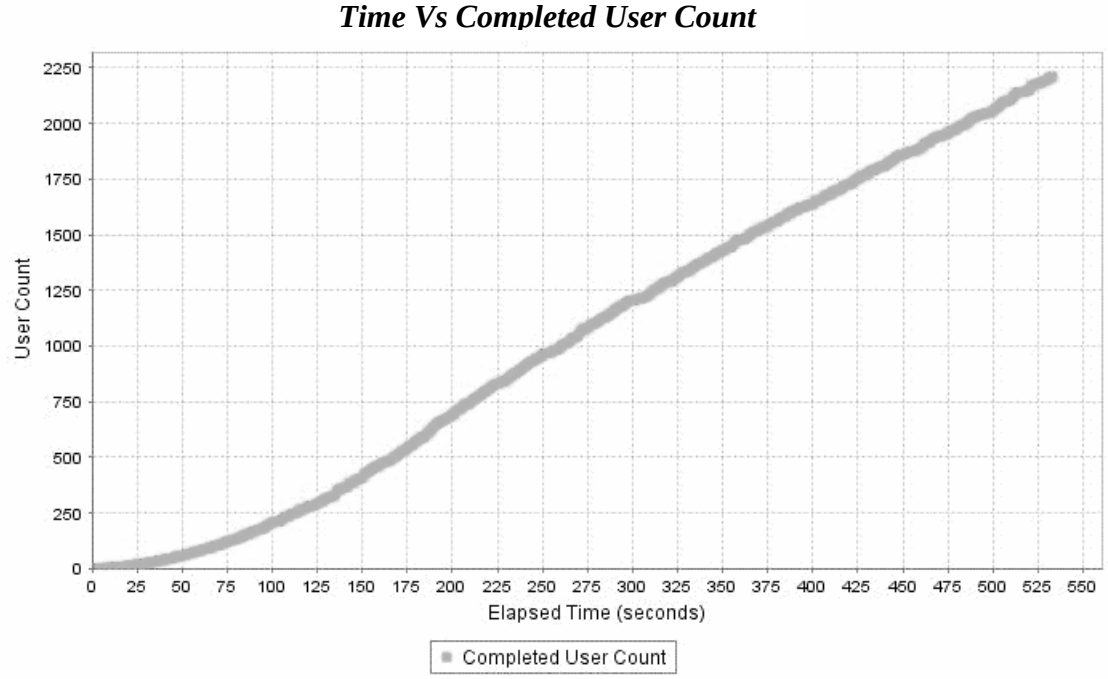
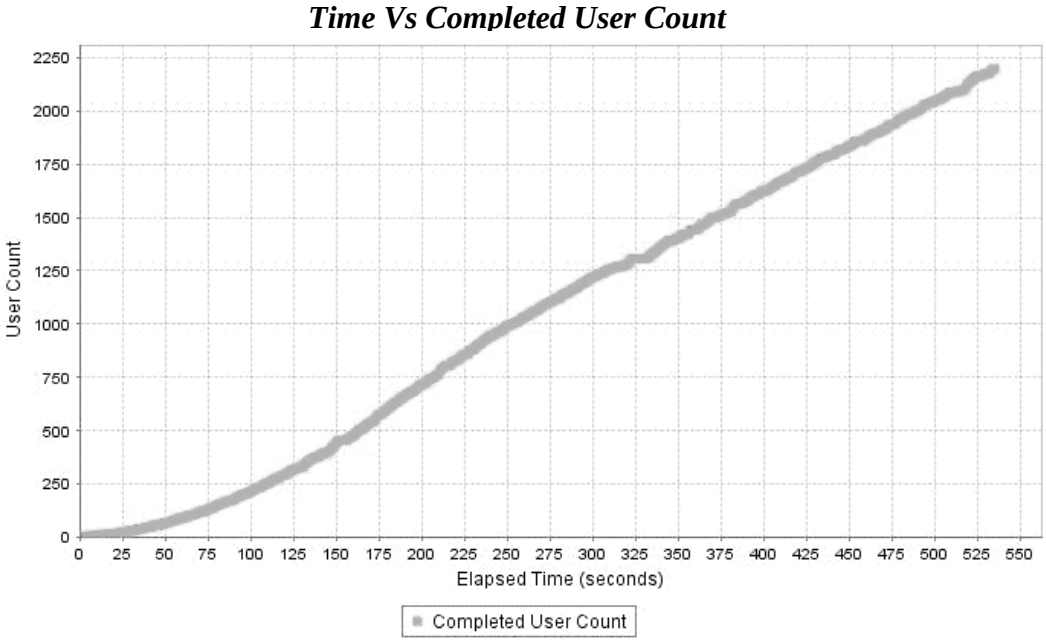


Figura 41 Solicitudes completadas a lo largo del tiempo J2EE



La Figura 42 y la Figura 43 muestran el tiempo de respuesta del servicio Web conforme se incrementa el número de solicitudes. Sobre la base de los datos recabados se puede apreciar que el servicio Web de .NET ofrece un tiempo de respuesta levemente menor que su contraparte J2EE. Dicha diferencia no es significativa y puede verse modificada según lo cambios que se realicen en el manejo de los mensajes SOAP y las configuraciones en los servidores para permitir un mejor desempeño. La Tabla VI presenta algunos datos puntuales correspondientes a las graficas.

Tabla VI Tiempo de respuesta (ms) contra número de solicitudes

Número de Solicitudes	Servicio Web .NET	Servicio Web J2EE
750	54	56
1250	57	70
2500	70	77

Figura 42 Tiempo de respuesta contra número de solicitudes .NET[14]

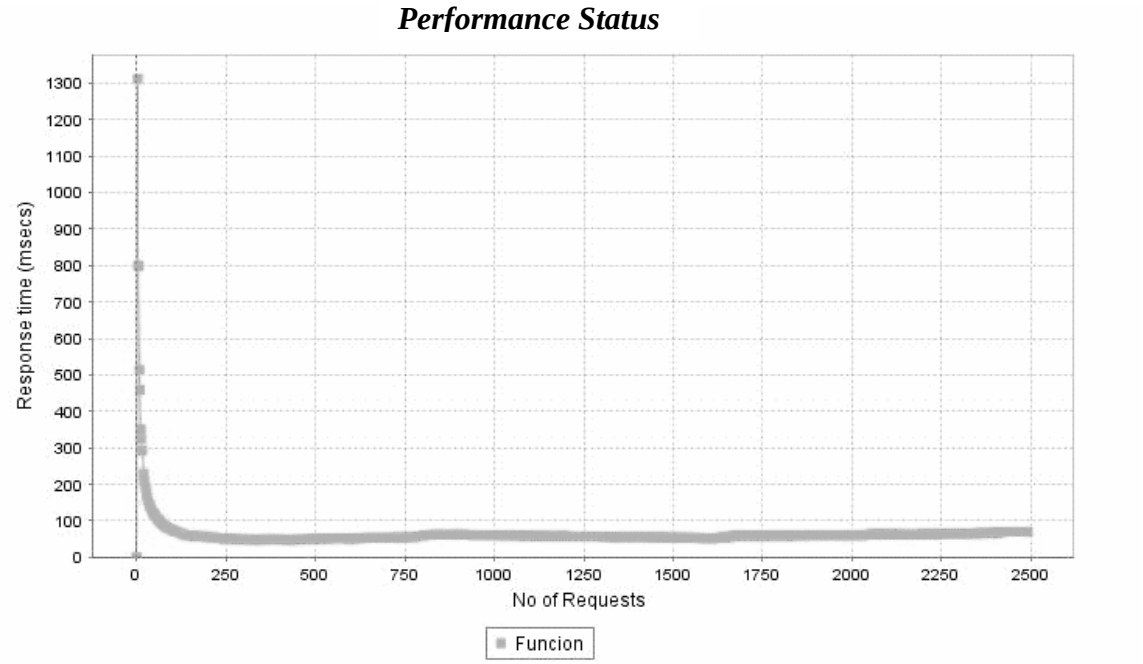
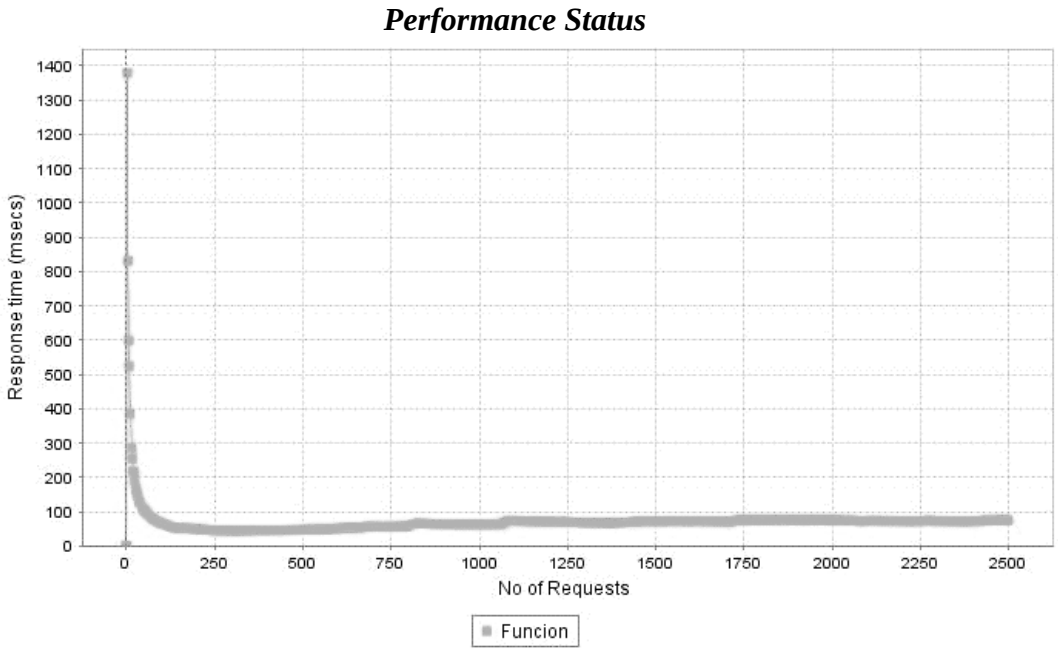


Figura 43 Tiempo de respuesta contra número de solicitudes J2EE



[14] El tiempo de respuesta elevado al inicio es debido al arranque de la aplicación de pruebas.

Decisión, disponer de dos alternativas para satisfacer una necesidad nos presenta el dilema de escoger. Seleccionar una alternativa sobre otra conlleva un estudio sobre cada una de las opciones, dicho estudio, expuesto en el presente trabajo permite plantear el siguiente enunciado: Decidir entre una u otra plataforma para el desarrollo de servicios Web no representa la exclusión de la alternativa no seleccionada, gracias al principio fundamental de los servicios Web, la inter-operabilidad, se puede convivir con ambas y al estar ambas trabajando con las especificaciones estándar de la industria, las dos plataformas son completamente funcionales siendo posible alcanzar similar desempeño, dependiendo del proceso de desarrollo de la solución.

CONCLUSIONES

1. Los servicios Web son piezas fundamentales para la creación de sistemas heterogéneos e integrados, proveyendo la comunicación de forma transparente entre las distintas aplicaciones, permitiendo la automatización de tareas y otorgando la posibilidad de comunicar aplicaciones sin intervención de los usuarios. Todo ello sobre la base de estándares de la industria.

2. Los servicios Web se basan en cuatro tecnologías fundamentales que son XML, para plasmar la información; SOAP, para la transmisión de la información; WSDL, para la descripción de las funcionalidades presentadas por un determinado servicio Web y UDDI, para la publicación y descubrimiento de servicios Web por parte de las organizaciones. Alrededor de estas tecnologías se acoplan aquellas que permiten asegurar la comunicación, mejorar la comunicación y traspasar documentos con el fin de aumentar la funcionalidad en sí del uso de los servicios Web.

3. La plataforma .NET presenta un abanico de lenguajes de programación para el desarrollo de servicios Web y es compatible con las especificaciones determinadas por W3C y OASIS, y esto la hace una herramienta capaz para fabricar una arquitectura de servicios Web completa. Siendo una plataforma concebida y construida con la orientación hacia los servicios Web, por ello la integración nativa de estas tecnologías.

4. La plataforma J2EE presenta una compatibilidad con los estándares tanto del W3C, de OASIS, como de la misma especificación de Java, siendo una especificación estándar para fabricar una arquitectura de servicios Web altamente transportable y escalable, pero apoyada únicamente en un lenguaje de programación que es Java. Existente antes que los servicios Web supo integrarlos gracias a los APIs creados para ello.

5. Ambas plataformas presentan las herramientas necesarias para el diseño, desarrollo y utilización de servicios Web complejos y altamente inter-operables, la selección de una u otra alternativa dependerá de la experiencia previa y de las habilidades existentes para trabajar con cada una de ellas.

6. El desempeño de cualquiera de las plataformas puede ser igualado y superado por la otra, esto, si se aplican las mejores prácticas en el desarrollo del software así como aplicando optimizaciones al sistema en conjunto; por lo tanto, el desempeño debe verse no sólo como un atributo de la plataforma sino que también debe verse como un efecto de los conocimientos que se tengan de la herramienta de trabajo.

RECOMENDACIONES

1. En el análisis y diseño de las aplicaciones debe tenerse en cuenta qué funcionalidades de los servicios Web se desean, así como determinar cuáles son los estándares o especificaciones presentes para dicha funcionalidad, ya que sobre la base de esto se pueda seleccionar la plataforma de desarrollo que nos permita una arquitectura escalable, confiable y altamente inter-operable.

2. La utilización de las funcionalidades que existen alrededor de los servicios Web, esto es, la implementación de la seguridad, las capacidades de intercambio de datos, la alta disponibilidad, etc. deben ser analizadas con detenimiento para no desarrollar servicios Web que resulten siendo poco compatibles con las otras aplicaciones no desarrolladas bajo la misma plataforma.

3. En el proceso de selección de la plataforma de desarrollo para la creación de servicios Web, debe incluirse también las características del ambiente de programación que permitirán un mejor desarrollo, lo cual dependerá tanto de los conocimientos previos que se tengan como de la curva de aprendizaje presentada por cada lenguaje.

4. Las habilidades existentes, la experiencia previa, el ambiente actual y los socios o clientes con quienes se interactúa, deben servir como guía para la toma de decisión de una plataforma de desarrollo.

BIBLIOGRAFÍA

1. Alonso, Gustavo *et. al.* ***Web Services - Concepts, Architectures and Applications***. Berlin: Springer Verlag, 2004. 354 pp.

2. Apshankar, Kapil *et. al.* ***Web Services Business Strategies and Architectures***. Illinois: A-Press, 2003. 348 pp.

3. Browne, Christopher *et. al.* ***Professional Open Source Web Services***. New York: Springer Verlag, 2002. 560 pp.

4. Chappel, David. ***Understandig .NET A Tutorial and Analysis***. 4ª ed. U.S.A.: Addison-Wesley, 2002. 348 pp.

5. O'Neill, Mark. ***Web Services Security***. U.S.A.: McGraw-Hill, 2003. 312 pp.

1. Apshankar, Kapil. *WS-Security: Security for Web Services*, http://www.webservicesarchitect.com/content/articles/apshankar04.asp, 2002.

2. Mahmoud, Qusay H. *Securing Web Services and the Java WSDP 1.5 XWS-Security Framework*, http://java.sun.com/developer/technicalArticles/WebServices/security/index.html , 2005.

3. Mírelo Cuervos, Juan Julián. Servicios Web y Microsoft .NET, http://geneura.ugr.es/~jmerelo/ws/, 2002

4. Microsoft Corporation*, Comparison: J2EE/EJB vs. Microsoft .NET*, http://www.microsoft.com/seminar/shared/asp/view.asp?url=/Seminar/en/20011009devt1-30/manifest.xml, 2004

5. OASIS, *OASIS Standards and Other Approved Work*, http://www.oasis-open.org/specs/index.php, 2006

6. Seely, Scott. *WS-Security*, http://www.microsoft.com/spanish/msdn/articulos/archivo/121202/voices/understw.asp, 2002.

7. Sun Microsystems, Inc., *Developing Web Services with J2EE 1.4*, http://java.sun.com/developer/technicalArticles/J2EE/j2ee_ws/, 2004

8. Sun Microsystems, Inc., The Java (TM) Web Services Tutorial, http://java.sun.com/webservices/docs/1.3/tutorial/doc/index.html, 2003

9. Vawter, Chad y Ed Roman. *Enterprise Java Community: J2EE vs. Microsoft.NET: A comparison of building XML-based web services*, http://www.theserverside.com/tt/articles/article.tss?l=J2EE-vs-DOTNET, 2001

10. Web Services: SLMM-RPC, SOAP, sobre P.D., Per., y otros conceptos, http://web-services.bankhacker.com/, s.a.

11. WS-I, *WS-I Deliverables*, http://www.ws-i.org/deliverables/Default.aspx, 2006